SVEUČILIŠTE U ZAGREBU – GEODETSKI FAKULTET
UNIVERSITY OF ZAGREB – FACULTY OF GEODESY
Zavod za geomatiku; Katedra za geoinformatiku
Institute of Geomatics; Chair of Geoinformatics
Kačićeva 26; HR-10000 Zagreb, CROATIA
Web: http://www.geof.hr; Tel.: (+385 1) 46 39 222; Fax.: (+385 1) 48 28 081

Usmjerenje: geoinformatika

Diplomski rad

GPS KOLEKTOR – PROTOTIP ANDROID APLIKACIJE ZA PRIKUPLJANJE PROSTORNIH PODATAKA

Izradila:
Ana Marjanica
Kranjčevićeva 21
Split
amarjanica@geof.hr

1. mentor: prof. dr. sc. Damir Medak, dipl. ing. geod.

2. mentor: Mario Miler, dipl. ing. geod.

3. mentor: Dražen Odobašić, dipl. ing. geod.

Zagreb, rujan 2011.

Zahvala:

Zahvaljujem mentorima prof. dr. sc. Damiru Medaku i asistentima Mariu Mileru i Draženu Odobašiću koji su svojim znanstvenim i stručnim savjetima oblikovali ideju i pomogli mi u izradi ovoga diplomskog rada.

GPS KOLEKTOR – Prototip Android aplikacije za prikupljanje prostornih podataka

Sažetak: Ovaj rad opisuje postupak izrade mobilne geo-aplikacije za *Android* platformu. U izradi aplikacije korišteni su softverski alati *Eclipse*, *Android SDK*, *Geoserver*, *OpenLayers* i *SQLite* te *Android* mobilni uređaj. Također su korišteni *NTRIP* i *WFS-T* protokoli za razmjenu prostornih podataka putem interneta. Kao rezultat, razvijena je aplikacija kojom se mogu prikupljati i vizualizirati prostorni podaci te spremati radi potrebe naknadne analize prostornih podataka. Prikupljanje prostornih podataka je omogućeno pomoću integriranog *GPS* prijemnika, mreže primopredajnih uređaja i korištenjem *NTRIP* protokola za povezivanje sa udaljenim poslužiteljem. Vizualizacija prostornih podataka je omogućena korištenjem *Google Maps* servisa.

Ključne riječi: *Android*, *GPS*, *NTRIP*, *WFS-T*, *Eclipse*, *Geoserver*, *OpenLayers*, *SQLite*, *Google Maps*

GPS KOLEKTOR – Prototype of the Android application for spatial data collection

Abstract: This paper describes the process of writing a mobile geo-application for the *Android* platform. Software tools like *Eclipse*, *Android SDK, Geoserver*, *OpenLayers*, *SQLite*, and *Android* device were used in the development of such an application. This application implements *NTRIP* and *WFS-T* protocol for handling geospatial data via internet. As a result, application was produced, which can then be used to collect, visualise and save spatial data for purposes of further analysis. Collecting spatial data is provided by the integrated *GPS* receiver, base tranceiver station network, and *NTRIP* protocol are used for collecting geospatial data. Visualization of spatial data is enabled using the *Google Maps* service.

Keywords: *Android*, *GPS*, *NTRIP*, *WFS-T*, *Eclipse*, *Geoserver*, *OpenLayers*, *SQLite*, *Google Maps*

SADRŽAJ

1. Uvod

Dinamički razvoj komunikacijskih tehnologija (engl. *Communication Technologies*) u posljednjih nekoliko desetljeća donio je novu dimenziju u području geoinformacijskih sustava i geoinformacijskih tehnologija, tj. mobilnost. Mobilnost omogućava jednostavnije i brže prikupljanje te obradu podataka i prezentaciju istih neovisno o području primjene podataka. Sa tržišne perspektive, izuzetno je važno da korisnici imaju informacije na raspolaganju u svakom trenutku i na svakom mjestu. Većina informacija s kojima se korisnici susreću imaju vezu s geografskom lokacijom, tj. sa geoinformacijama. Najpopularniji medij za prikaz geoinformacija i komunikaciju među cjelokupnom populacijom je karta. Podaci sa karte primjenjuju se u orijentaciji i navigaciji, za vizualizaciju , upravljanje i planiranje te za razne analize. Na primjer, podaci sa karte se mogu koristiti za odabir pogodnog područja za izgradnju kuće, poljoprivredne poslove, prostorno planiranje i drugo. Statičke karte i tradicionalni *GIS* ne mogu u potpunosti zadovoljiti potrebe korisnika jer većina objekata koje želimo locirati nisu statični u prostoru i vremenu. Vrlo je važno imati podatke o brzini, ruti i smjeru kretanja objekta radi aplikacije kod praćenja prometa, prognoziranja vremena, kontroliranja širenja zarazne bolesti i sličnog. Mobilni *GIS* za razliku od tradicionalnog *GIS*-a može korisniku pribaviti tražene podatke. Mobilni *GIS* možemo definirati kao geoinformacijski sustav koji ima svojstva mobilnosti i servisa u realnom vremenu. Mobilni *GIS* definira interaktivni model između korisnika i svijeta. Kroz interakciju korisnika sa modelom, njegov prikaz se mijenja ovisno o ulozi. Mobilni *GIS* kombinira geo-aplikacije sa jednostavnim mobilnim uređajima koji pružaju informacije u bilo kojem trenutku i na bilo kojem mjestu. Ovim se načinom geoinformacije postavljaju na vozačkim pločama automobila, u rukama ljudi na terenu i pruža se usluga hitnim službama u realnom vremenu kako bi mogli na vrijeme odgovarajuće reagirati i asistirati (URL1). Trenutne mogućnosti mobilnih geo-aplikacija uključuju prikupljanje prostornih podataka, vizualizaciju, analizu i upravljanje sa prostornim podacima te njihovu primjenu u rješavanju kompleksnih problema koji uključuju geoinformacije.

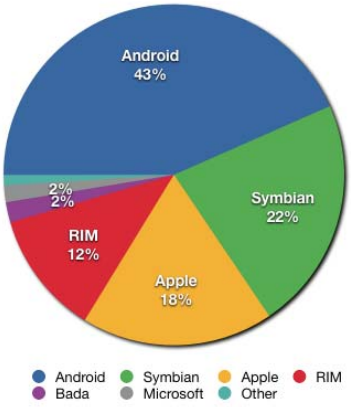
Među poznatim komercijalnim aplikacijama koje posjeduju navedene mogućnosti su *TerraSync* tvrtke *Trimble*, *ArcPad* tvrtke *ESRI* te *gvSIG* aplikacija otvorenog koda. Svrha izrade ovog diplomskog rada bila je istražiti mogućnosti razvijanja mobilne geo-aplikacije. Aplikacija je izrađena za Android platformu najviše iz razloga što je trenutno Android najpopularniji operativni sustav koji se neprestano usavršava. Obzirom da su mogućnosti razvijanja ovakvog tipa aplikacija zaista velike, stavljen je naglasak na istraživanje mogućnosti prikupljanja prostornih podataka. Ovaj diplomski rad pruža dobru podlogu za razumijevanje načina na koji funkcionira proces prikupljanja prostornih podataka kod mobilnih geo-aplikacija.

2. Osnovni pojmovi kod razvoja mobilne aplikacije

Primjenjivi program, također poznat kao aplikacija (eng. *Application software*), računalni je program dizajniran za pomoć korisnicima kako bi mogli izvršavati jedan ili više određenih zadataka (URL2). Aplikacije koje se mogu instalirati na mobilne uređaje zovu se mobilne aplikacije. Mobilne aplikacije koje pružaju razne usluge, pritom koristeći prostorne informacije, zovu se mobilne geoaplikacije. Razvoj mobilne aplikacije možemo definirati kao proces u kojem se aplikacija razvija za prijenosne uređaje male potrošnje, poput *PDA* uređaja i mobitela. Takve aplikacije su, ili unaprijed instalirane na uređajima, ili se mogu preuzimati sa raznih mobilnih platformi za distribuciju softvera. Aplikacija je konačni proizvod programskog koda, koji se piše u programskom jeziku, koje računalo zna i može izvršiti (URL3). Osnovna podjela je na niže (strojne jezike) i više (orijentirane ljudima). Viši jezici pak mogu biti:

- sekvencijalni
- proceduralni (*Pascal*, *C*)
- funkcijski (*LISP*, *Erlang*, *ML*)
- objektno orijentirani (*Java*, *C++*)

Programski jezici koji se koriste za pisanje koda mobilne aplikacije su pretežno objektno orijentirani, a prevladavaju *Java*, *C++* i *C#*. Objektno orijentirani jezici podrazumijevaju programiranje upotrebom skupa objekata, tj. zasebnih programa koji su sposobni obaviti određene zadatke i međusobno razmjenjuju poruke. Za razliku od tradicionalnih programskih jezika (npr. *Pascal*), glavni program nema izravan pristup objektima i ne može izravno obavljati operacije s njima. Programski kodovi moraju biti napisani u jeziku koji podržava operativni sustav za koji je aplikacija namijenjena. Mobilni operativni sustav je skup osnovnih sustavnih programa koji upravljaju sklopovljem mobilnog uređaja radi ostvarivanja osnovnih funkcija računala: ulaz, memoriranje, obrada i izlaz podataka (URL4). Najzastupljeniji OS-i namijenjeni *smartphone*-ima su *Android* i *Symbian* (Slika. 1.).



Slika 1. *Globalni udio mobilnog operativnog sustava na tržištu smartphone-a (kolovoz, 2011.g., URL4)*

2.1. Android operativni sustav

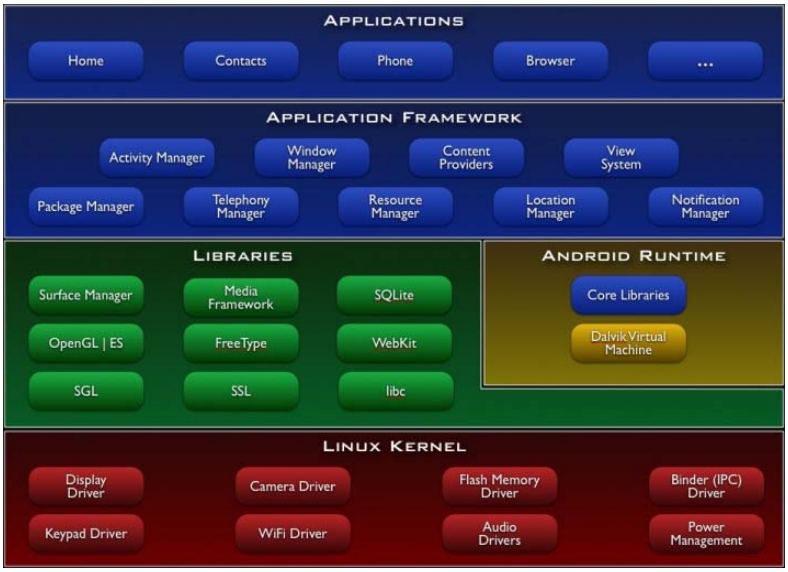
Android OS je mobilni operativni sustav otvorenog koda (engl. *open-source*) koji je izgrađen na podlozi *Linux Kernel* operativnog sustava (URL5). *Open-source* je termin za softver koji se može besplatno i slobodno koristiti te čiji je programski kod javno dostupan pod licencom koja osigurava zaštitu autorskih prava. *Android* je nastao 2003. g. u kalifornijskom gradu Palo Alto

od strane četvorice samostalnih programera: Andy Rubin, Rich Miner, Nick Sears i Chris White. Temeljna ideja je bila napraviti pametne mobilne uređaje s naglaskom na lokaciju korisnika. U kolovozu 2005. g. *Android* je postao podružnica tvrtke *Google Inc*, na čijem čelu su ostali njegovi utemeljitelji. S ovim potezom *Google* je uspješno proširio svoje djelovanje na tržište *smartphone*-a. *Android*-om upravlja *Open handset Alliance*, grupa osamdesetak tehnoloških kompanija među kojima su *Google*, *HTC*, *T-mobile* i drugi. Cilj ovog konzorcija je ubrzati inovacije na području mobilnih operativnih sustava s naglaskom na javnu dostupnost koda. Javnom dostupnošću koda želi se potrošačima ponuditi bogatije, jeftinije i bolje iskustvo. Iako je *Android* dostupan pod licencom slobodnog softvera, proizvođači mobilnih uređaja mogu koristiti *Google*-ov *Android* zaštitni znak samo uz uvjet da su uređaji u skladu s *Google*-ovim dokumentima o *CDD* kompatibilnosti. Uređaji također moraju ispunjavati uvjete za licenciranje aplikacija zatvorenog koda, uključujući *Android market*. *Android market* je aplikacija koja je automatski uključena u *Android* uređajima, a omogućava korisnicima pregledavanje i preuzimanje aplikacija koje objavljuju neovisni programeri. Android je od svojeg izvornog izdanja doživio niz ažuriranja koja se odnose na ispravak *bug*-ova prethodnih verzija *OS*-a te dodavanje novih mogućnosti. Inače, dosad svaka verzija *Android OS*-a je razvijena pod kodnim imenom nekakve slastice. Nedavno objavljena ažuriranja su verzije *2.0/2.1 Éclair*, *2.2 Froyo*, *2.3 Gingerbread* i *3.0/3.1 Honeycomb*. Trenutne mogućnosti *Android* mobilne platforme su prilagođenost za *VGA 2D* grafiku i *3D* grafiku temeljene na *OpenGL ES 2.0* specifikacijama. Spremanje podataka se radi u S*QLite bazi*, koja je uključena u instalaciji *Android OS*-a. Takvo spremanje je korisno za aplikacije koje zauzimaju puno memorije i omogućava efikasniji rad s uređajem. Što se tiče mrežnog rada, *Android* podržava *GSM/EDGE*, *IDEN*, *CDMA*, *EV-DO*, *UMTS*, *Bluetooth*, *Wi-Fi*, *LTE*, *NFC* i *WiMAX*. Svi Androidovi mobilni uređaji podržavaju veliki broj senzora poput akcelerometara, termometara, žiroskopa, magnetometara, senzora za pritisak i *GPS-a*. *Android* podržava *multitasking* aplikacija. *Multitasking* znači da se više aplikacija može izvršavati u isto vrijeme, ili jedna aplikacija može

istodobno obavljati više funkcija. Aplikacije za *Android* se uglavnom programiraju u *Javi*. Za izvršavanje koda aplikacije mora postojati *Javin* virtualni operativni sustav (*JVM*). *Virtual Machine* označava kompletno izolirani operativni sustav koji radi unutar normalnog operativnog sustava. Na primjer, unutar računala s *Windows XP*-a može se instalirati *VM* koji ima *Linux* operativni sustav, te se tretira kao drugo računalo jer je potpuno programski izolirano od matice. Unutar *Android* platforme ne postoji *JVM*, već *Dalvik* virtualna mašina (*DVM*) koja je specijalizirana za *Android,* optimizirana za očuvanje baterije uređaja i korištenje ograničene radne memorije. Prije izvršavanja Java koda, aplikacije se konvertiraju u *Dalvik*-ov izvršni format s ekstenzijom *.dex*. Takav format je prilagođen ograničenim mogućnostima mobilnih uređaja.

2.1.1. Struktura Android operativnog sustava

Komponente operacijskog sustava Android dijele se u pet cjelina (Slika 2.):



Slika 2. *Arhitektura Android mobilne platforme (URL6).*

- Aplikacije (engl. *applications*)

Android OS dolazi sa skupom osnovnih aplikacija, uključujući *e-mail* klijent, *SMS* program, kalendar, karte, preglednik, kontakte i druge aplikacije.

- Aplikacijski okvir (engl. *application framework*)

Aplikacijski okvir označava skup zajedničkih softverskih rutina koji pruža strukturni temelj za razvoj aplikacija. Drugim riječima, aplikacijski okvir olakšava proces programiranja. Kod *Android*-a je specifično da se pri programiranju aplikacije mogu koristiti funkcije osnovnih aplikacija *Android*-a. Na primjer, ako programiramo aplikaciju koja koristi usluge *SMS*-a ili telefoniranja, nije potrebno posebno za te funkcije pisati kod, već se koriste postojeće aplikacije *Contacts* i *Phone*.

- Biblioteke (engl. *library*)

Android uključuje skup osnovnih *C/C++* biblioteke koje koriste razne komponente *Android* sustava. Neke od osnovnih biblioteka su: sistemska *C* biblioteka, *SQLite*, medijske biblioteke i druge.

- *Android*-ovo radno okruženje (engl. *Android runtime*)

U *Android*-ovo radno okruženje pripadaju osnovne biblioteke koje omogućuju većinu funkcionalnosti osnovnih biblioteka programskog jezika *Java*. Svaka aplikacija se odvija u svom vlastitom procesu, s vlastitom instancom *Dalvik* virtualnog stroja.

- *Linux*-ova jezgra (engl. *Linux Kernel*)

Android OS se oslanja na Linux 2.6 verziju za sistemske servise za zaštitu uređaja te za upravljanje sa memorijom i procesima. *Linux*-ova jezgra djeluje kao odjeljujući sloj između hardvera i softvera *Android* mobilnog uređaja.

2.2. Java programski jezik

Java je objektno-orijentirani programski jezik razvijen u timu predvođenim James Goslingom u kompaniji *Sun Microsystems* početkom 1990. g. (URL7) Ideja je bila da se stvori 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. *Java* pripada skupini viših programskih jezika i ima svojstvo

prenosivosti (URL8). Prenosivost označava mogućnost izvršenja jednog te istog programa na različitim računalima. Program koji je pisan u višem programskom jeziku ne može biti izvršen izravno na računalu. Takav program se prevodi u strojni jezik. Tu pretvorbu obavlja zasebni program kompajler (engl. *compiler*). Nakon što je program jednom preveden, može se izvršavati neograničen broj puta, ali samo na jednom računalu. Da bi se mogao izvršavati na drukčijem računalu, potrebno ga je ponovno kompajlirati. Umjesto korištenja kompajlera, moguće je koristiti interpreter, koji prevodi naredbu po naredbu, prema potrebi. Interpreter omogućuje izvršavanje programa kompajliranog za jednu vrstu računala na drugom računalu. Kod programskog jezika *Java* se koristi kombinacija kompajlera i interpretera. Programi pisani u *Javi* se prevode u strojni jezik računala koje ne postoji. Takvo virtualno računalo se zove *Java Virtual Machine* (*JVM*). Sve što je potrebno je interpreter za *Java* bajt kod (engl. *bytecode*). Ideja je da ako se kod napiše i kompajlira na jednoj platformi (npr. *Mac OS X*), taj isti bajt kod se može izvršavati na svim ostalim platformama koje imaju *JVM* (npr. *Microsoft Windows XP*, *Linux*) bez potrebe za ponovnim kompajliranjem na toj platformi. U *Java* programskom jeziku su objektno-orjentirani principi obavezni. Sve je u Javi objekt, a sav izvorni kod je pisan unutar klasa.

Objekt u realnom svijetu može biti čovjek ili bilo što drugo. Objekt je definiran svojim stanjem i ponašanjem. 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 (promjenjivim vrijednostima), 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 koda ili cijeli programski kod. 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*). 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. 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.

2.2.1. 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:

- 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 gniježđenje, 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 3 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.

2.3. Struktura Android aplikacije

Android aplikacije su pisane u *Java* programskom jeziku. *Android SDK* kompajlira programski kod u izvršni kod. Izvršni kod je arhivska datoteka sa ekstenzijom *.apk*. Takva datoteka je aplikacija koja se može instalirati na *Android*-ovom mobilnom uređaju. Pravilo je da se pri instalaciji svakoj aplikaciji dodjeljuje jedinstveni *Linux* identifikator. Prema tom identifikatoru aplikaciji se daje pristup servisima, senzorima i drugim dijelovima uređaja i operativnog sustava. *Android* platforma primjenjuje princip najmanje privilegije, tj. aplikacija ima samo pristup onim komponentama koje su joj

potrebne za rad. Svaka se aplikacija pokreće u *DVM*-u i djeluje izolirano od drugih aplikacija.

2.3.1. Komponente aplikacije

Aplikacijske komponente su temeljni sastavni 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 (URL9):

- 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“*. Po pokretanju nove aktivnosti, prekida se prethodna aktivnost i ponovno pokreće po prekidu trenutne aktivnosti. 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 peroda. 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:

- *Started*

  Servis počinje kad aplikacijska komponenta pozove servis metodom S*tartService()*. 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 potklasu 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 razlučivača sadržaja (engl. *Content Resolver*). 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 neku 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 primanje i reagiranje 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.

2.3.2. Aktiviranje komponenata aplikacije

Komponente aktivnosti, servisi i primatelji prijenosa aktiviraju se preko asinkronih poruka koje se zovu namjere (engl. *intents*). Dobavljači sadržaja se aktiviraju zahtjevom *ContentResolver-a*. Namjere su objekti klase *Intent,* a sadrže informacije od interesa za komponentu koja prima asinkronu poruku. Za aktivaciju aktivnosti i servisa, namjera sadrži naziv komponente (npr. domena aplikacijskog paketa), ime tražene akcije (npr. editiranje podataka) i *URI* potrebnih podataka. Sadržaj namjere za primatelje prijenosa ime je tražene akcije. Prema sadržaju, namjere mogu biti eksplicitne ili implicitne. Eksplicitne namjere sadrže ime komponente (npr. naziv *Java* klase koja se poziva). Implicitne namjere definiraju određenu radnju bez imenovanja klase koja je potrebna za tu radnju (npr. pregled web stranice).

2.3.3. Android manifest dokument

Prije pokretanja aplikacije, sustav mora znati od kojih se komponenata aplikacija sastoji. Registracija aplikacijskih komponenata se radi u *Android-ovom manifest* dokumentu (URL10). *Manifest* je strukturirana *XML* datoteka koja se zove *AndroidManifest.xml* za sve aplikacije. Funkcije ovog dokumenta su identificiranje aplikacijskog pristupa sistemu uređaja, definiranje minimalne verzije operatvinog sustava na kojem će aplikacija raditi, deklariranje dijelova uređaja ili aplikacija koje program koristi, deklariranje eksternih biblioteka klasa s kojima je program povezan (Tablica 1.).

1.	<? xml version = "1.0" encoding = "utf-8"? >
2.	< manifest ... >
3.	< uses-permission />

4.	< uses-sdk />
5.	< application android : icon = "@drawable/app_icon.png" ... >
6.	< activity android:name = "com.example.project.ExampleActivity"
7.	< intent-filter >
8.	< action />
9.	< category />
10.	< data /> </ intent-filter >> </ activity >
11.	< service >
12.	< intent-filter > . . . </ intent-filter >
13.	< meta-data /> </ service >
14.	< receiver>
15.	< intent-filter > . . . </ intent-filter >
16.	< meta-data /> </ receiver >
17.	< provider >
18.	< grant-uri-permission />
19.	< meta-data /> </provider >
20.	< uses-library >… <uses-library /> </ application > </ manifest >

Tablica 1. *Općenita struktura Android manifesta.*

Neke konvencije i pravila vrijede općenito za sve elemente i atribute u Android manifest dokumentu. Jedino su elementi <*manifest*> i <*application*> obavezni dijelovi *manifesta* i mogu se samo jednom pojaviti u dokumentu.

- Elementi

Opcionalni elementi <*activitiy*>, <*provider*>, <*service*> i <*receiver*> se moraju nalaziti unutar elementa <*application*> po proizvoljnom redoslijedu. Sve vrijednosti koje opisuje pojedini element unose se pomoću vrijednosti atributa.

- Atributi

Atributi se imenuju na način *android : ime_atributa = vrijednost_atributa* (npr. *android:name = "com.example.project. ExampleActivity")*. Svi atributi su opcionalni osim onih koji su navedeni u petom i šestom retku Tablice 1.

- Nazivi klasa

Svaka klasa se mora deklarirati u *manifest* dokumentu. Definicija klase je moguća na dva načina. Kod naziva komponente se dodaje atribut *android:name*, a vrijednost atributa može sadržavati punu oznaku paketa ili

samo ime klase. Na primjer, klasu možemo imenovati sa <*service android:name="com.example.project.SecretService" . . .* > ili sa <*service android:name=".SecretService" . . .* >.

Već je spomenuto da se aplikacijske komponente aktiviraju preko namjera (engl. *intents*). Kako bi informirali sustav koje namjere mogu izvršiti, komponente mogu imati jedan ili više filtera za namjere (engl. *intent filter*). Svaki filter opisuje skup namjera koje je komponenta voljna primiti. Eksplicitna namjera se uvijek dostavi na odredište, dok implicitne namjere stižu do odredišta samo ako prođu kroz jedan od filtera. Obavezni filteri kod deklaracije početne aktivnosti su *Main* i *Launcher*. *Main* definira koji će se ekran otvoriti po pokretanju aplikacije, a *Launcher* pokretanje aplikacije.

Svaka *Android* aplikacija se izvodi u vlastitom procesu. Sigurnost između aplikacije i sustava je izvedena na procesnoj razini pomoću dodjeljivanja jedinstvenog *Linux* identifikatora. Dodatne sigurnosne mjere se provode pomoću mehanizama dopuštenja (engl. *permission*). Dopuštenje predstavlja ograničavajući pristup na dio koda ili podataka o uređaju. Osnovna *Android*-ova aplikacija nema nikakvih dopuštenja povezanih sa sobom. Kako bi se mogle koristiti zaštićene mogućnosti mobitela, potrebno je u manifestu deklarirati dopuštenja preko oznaka *<uses-permission android: name="…"/>.*

Svaka aplikacija je povezana sa standardnom *Android*-ovom bibliotekom koja se sastoji od osnovnih paketa za razvoj aplikacija (npr. *Activity*, *Service*, *View*, *Button*). Kad *Android*-ova biblioteka nije dostatna za potrebe programiranja, aplikacije koriste biblioteke s drugih izvora. Korištenje eksterne biblioteke deklarira se u *Android manifestu* putem oznake <*uses-library*>.

2.3.4. Aplikacijski resursi

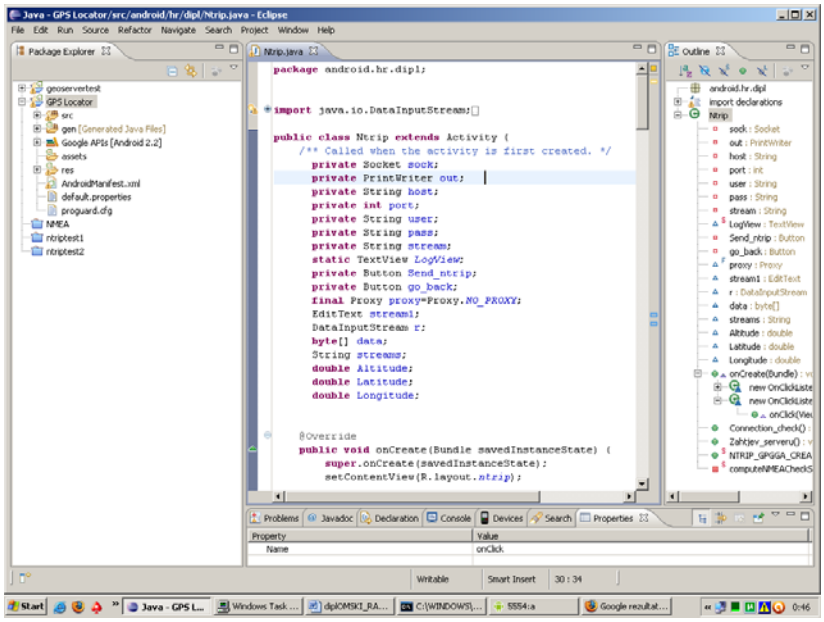
Za grafički prikaz aplikacije koristi se direktorij resursi u kojem mogu biti spremljene slike, ikone, audio datoteke, izbornici, boje, fontovi i ostalo što ima veze sa vizualnom prezentacijom. Korištenje resursa olakšava ažuriranje audio-grafike bez modificiranja koda. Za svaki resurs koji je uključen u

Android projekt, *Android SDK* definira jedinstveni identifikator, koji se koristi kao poveznica aplikacijskog koda i drugih izvora definiranih u *XML*-u.
Svi identifikatori se spremaju u autogeneriranoj datoteci *R.java. R.java* služi kao indeks svih resursa korištenih u projektu. Na primjer, aplikacija sadrži slikovnu datoteku pod nazivom *logo.png*. Slikovna datoteka *logo.png* je spremljena u *res / drawable* direktoriju. *Android SDK* generira identifikator resursa pod nazivom *R.drawable.logo* koji se koristi u programskom kodu za umetanje slike u korisničko sučelje. Jedan od važnijih aspekata izolacije resursa od programskog koda mogućnost je osiguranja alternativnih resursa za različite konfiguracije uređaja. Na primjer, moguće je kreirati aplikaciju koja će podržavati više jezika. Za svaki jezik se kreira zasebni direktorij u resursima sa sufiksom jezika, a u svakom direktoriju su datoteke na tom jeziku. Na temelju kvalifikatora jezika, *Android* sustav primjenjuje odgovarajući jezik na svom korisničkom sučelju.

3. Razvoj mobilne aplikacije GPS Kolektor

3.1. Eclipse IDE

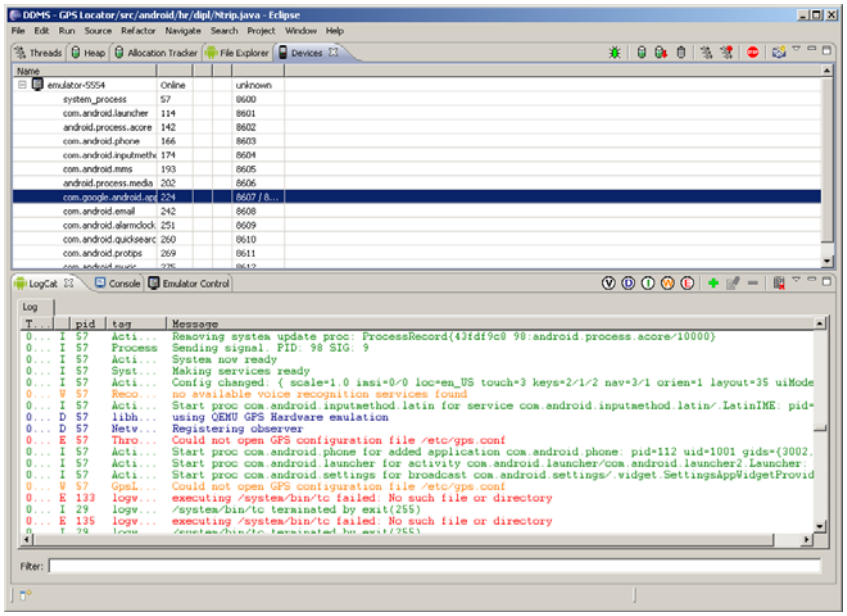
Programski kod diplomskog rada je pisan u *Eclipse* razvojnom okruženju. *Eclipse* je višejezično, softversko razvojno okruženje koje se sastoji od integriranog razvojnog okruženja (*IDE*) i proširivog *plug-in* sistema (URL11). *Eclipse* je razvijen od strane *Object Technology International* kompanije 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.). Na slici je prikazana *Java* perspektiva koja se sastoji preglednika programskog paketa (engl. *Package Explorer*), urednika za pisanje programskog koda i drugih opcionalnih prozora koje korisnik može odabrati (npr. *Java* konzola, prozor za upozorenje).



Slika 3. *Eclipse IDE.*

Za izradu diplomskog rada razvojno okruženje *Eclipse*-a je prošireno na *ADT* programski dodatak (engl. *plug-in*) za razvoj *Android* aplikacija. *ADT* omogućava kreiranje, uređivanje i uklanjanje grešaka (engl. *debugging*) kod *Android* aplikacija. *ADT* je paket softverskih razvojnih alata u kojem su uključene sljedeće komponente:

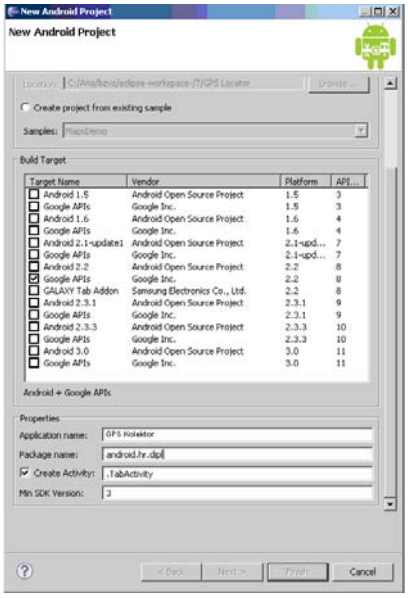
- Emulator koji služi za izvršavanje programa na računalu.
- *DDMS* (engl. *Dalvik Debug Monitor Server*) koji služi za pronalaženje i uklanjanje pogrešaka, te kontrolu i nadzor nad aplikacijama (Slika 4.).
- *AAPT* (engl. *Android Asset Packaging Tool*) koji koristi za stvaranje i distribuciju *Android*-ovog programskog paketa u *.apk* formatu.
- *ADB* (engl. *Android Debug Bridge*) koji se koristi za povezivanje koda na emulatoru ili uređaju sa *DDMS* alatom.
- Detaljna dokumentacija o *Android*-u.
- Primjeri koda za demonstraciju korištenja svih verzija *API*-a.



Slika 4. *DDMS perspektiva.*

3.2. Kreiranje projekta GPS Kolektor

Programiranje *Android* aplikacije u *Eclipse IDE*-u kreće od kreiranja novog *Android* projekta (Slika 5.). Naziv projekta i aplikacije je *GPS Kolektor*. Naziv projekta odgovara direktoriju u kojem se spremaju programski kod, biblioteke i resursi. Naziv paketa označava lokaciju na disku gdje će se spremati klase, a *Activity* označava početnu aktivnost. Početna aktivnost predstavlja prvi ekran koji korisnik vidi na uređaju pri pokretanju aplikacije. Naziv paketa mora biti jedinstven u odnosu na sve pakete koji su instalirani u *Android* sustavu. Iz tog razloga se koristi naziv u stilu domene (*android. hr. dipl*). Kod polja *Min SDK* definiramo minimalnu verziju operativnog sustava na kojem je moguće instalirati aplikaciju. Kod *Build target*-a odabiremo verziju platforme na kojoj se programski kod kompajlira. U ovom slučaju je odabrana verzija 2.2 *Google API-a*. *Google API* se koristi za aplikacije koje koriste *Google Maps*. Ove postavke je moguće kasnije mijenjati u *manifest* dokumentu.



Slika 5. *Postavke novog Android projekta.*

3.3. Deklaracija komponenata GPS Kolektora

Prije pokretanja aplikacije na mobitelu ili emulatoru, potrebno je opisati aplikaciju kako bi mobitel znao što treba napraviti sa aplikacijom, te koje resurse mobitela dopustiti da aplikacija koristi. Aplikacijski pristup sistemu uređaja uređuje se u *manifest* dokumentu (Tablica 2.).

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android = "http://schemas.android.com/apk/res/android"
3.    Package = "android.hr.dipl"  android:versionCode = "1" android:versionName = "1.0">
4.  <uses-sdk android:minSdkVersion = "3"  android:targetSdkVersion = "8"/>
5.  <supports-screens android:resizeable = "true"
6.                    android:smallScreens = "true"
7.                    android:normalScreens = "true"
8.                    android:largeScreens = "true"
9.                    android:anyDensity = "true"/>
10.
11. <uses-permission android:name = "android.permission.INTERNET"></uses-permission>
12. <uses-permission android:name = „android.permission.ACCESS_FINE_LOCATION" >
13. </uses-permission>
14. <uses-permission android:name = "android.permission.ACCESS_COARSE_LOCATION" >
15. </uses-permission>
```

```
16.  <uses-permission android:name = "android.permission.WRITE_EXTERNAL_STORAGE" />
17.  <application android:name = "Aplikacija" android:icon = "@drawable/icon" android:label =
18.  "@string/app_name">
19.  <uses-library android:required = "true" android:name = "com.google.android.maps" > </uses-
20.  library>
21.  <activity android:name = ".Provider" ></activity>
22.  <activity android:name = ".transf_koord" />
23.  <activity android:name = ".Karta" />
24.  <activity android:name = ".Tabactivity" >
25.      <intent-filter>
26.          <action android:name = "android.intent.action.MAIN" />
27.          <category android:name = "android.intent.category.LAUNCHER" />
28.      </intent-filter>     </activity>
29.  <activity android:name = ".GPS"/>
30.  <activity android:name = "ManageData" />
31.  <activity android:name = "Ntrip"></activity>
32.  <activity android:name="Nmealistener"></activity>
33.  <activity android:name="Wfs"></activity>
34.  </application>
35.  </manifest>
```

Tablica 2. *AndroidManifest.xml*

Prvi redak manifesta je obavezan početak svih *XML* dokumenata. U njemu se definira verzija *XML*-a i način zapisa (*UTF-8*).

Drugi redak označava početak *manifest* dokumenta. Unutar *<manifest> ... </manifest>* elementa unose se sve postavke aplikacije. Od drugog do četvrtog reda nalaze se autogenerirane postavke aplikacije. Te postavke su definirane kod pokretanja novog projekta (adresa paketa, minimalna verzija *API*-a i verzija *API*-a za koju je izrađen projekt). U petom redu su postavke za podržane uređaje. Aplikacija se može instalirati na uređaj bilo koje veličine ekrana, a da se pritom ekran aplikacije prilagodi tom uređaju. Od desetog do šesnaestog reda manifesta registrirana su dopuštenja aplikaciji za pristup servisima i dijelovima uređaja. Otklonjene su restrikcije za pristup internetu, preciznoj i gruboj lokaciji (engl. *FINE_LOCATION* i *COARSE_LOCATION*) te za spremanje podataka na memorijsku karticu. Od sedamnaestog do

tridesetidrugog reda nalazi se sadržaj elementa <*application*>. Unutar tog elementa su registrirane aplikacijske komponente i korištenje eksterne biblioteke *Google Maps*. Element <*application*> sadrži atribute za naziv i oznaku aplikacije te adresu ikone. Također sadrži elemente <*activity*> za svaku deklariranu klasu aplikacije koja ima korisničko sučelje. Deklarirane su aktivnosti pripadajućih klasa *Provider*, *Transf_koord*, *Karta*, *Tabactivity*, *GPS*, *ManageData* i *Ntrip*. Klase koje nisu navedene su *Aplikacija*, *DataHelper*, *DataXmlExporter*, *MapOverlay* i *ToBase64*. Te klase više djeluju kao pomoćne klase.Zbog preglednosti programskog koda premještene su u zasebne datoteke. Samo je jedna početna aktivnost *.Tabactivity* iz koje se pokreću ostale aktivnosti.

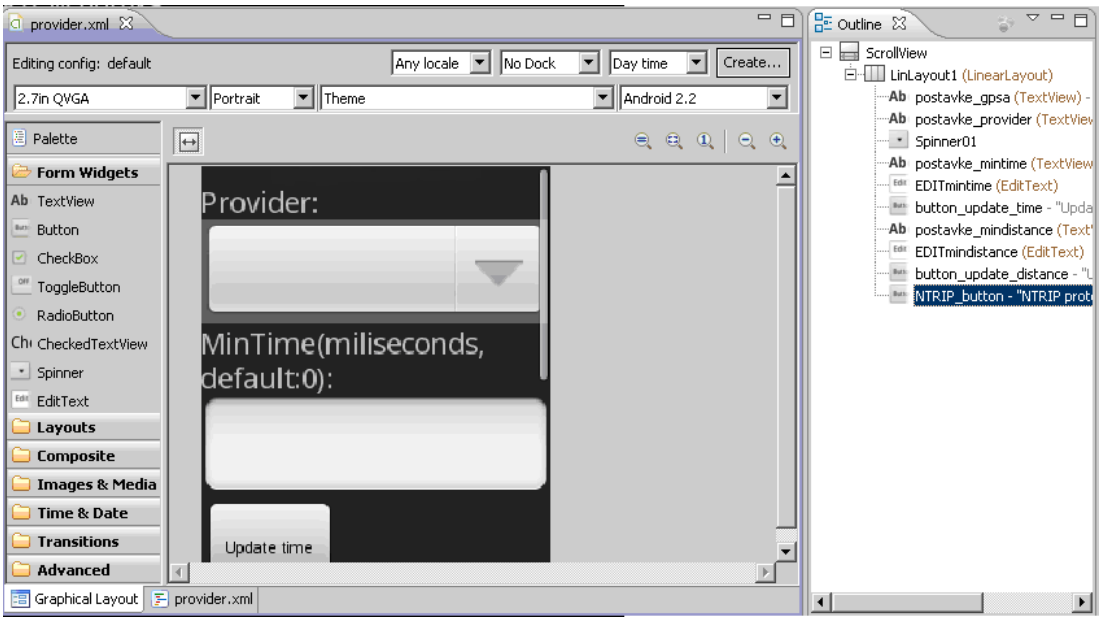
Obzirom da su dosad objašnjene struktura i komponente *Android* aplikacije, aplikaciju možemo razložiti na odgovarajuća poglavlja koja se bave tematikom pripadajuće funkcije:

- *GPS* i mrežnog pozicioniranja uređaja sa mogućnošću dobivanja sirovih podataka mjerenja (*NMEA0183*).
- Eksportiranje sirovih podataka mjerenja na *Geoserver*.
- Spremanje podataka mjerenja u bazu podataka i eksportiranje baze podataka na memorijsku karticu.
- kartografski prikaz koordinata iz baze podataka sa informacijama o koordinatama u *HDKS* sustavu.
- server-klijent dodatak za primanje *RTCM* korekcija sa korisnički definiranog servera (*NTRIP* protokol).

3.4. Korisničko sučelje GPS Kolektora

Korisničko sučelje Android aplikacije (engl. *User Interface*) može biti napravljeno pisanjem *XML* koda ili pisanjem *Java* koda. Uređuje se koristeći objekte iz *View* i *ViewGroup* klasa. Klasa *View* je osnovna klasa iz koje se izvode podklase pod nazivom *widgets*. *Widgets* predstavlja skup *UI* objekata koje nam *Android* nudi kako bi se ubrzala izgradnja korisničkog sučelja (tekstualno polje, gumb). Klasa *View* je osnovna klasa iz koje se izvode podklase koje sadrže gotove formate (engl. *layout*) koje *Android* nudi. *View*

objekti određuju način slaganja *ViewGroup* objekata po slojevima. Hijerarhija *View* objekata je organizirana po modelu roditelj – dijete. Izgled ekrana aplikacije se uređuje tako da se sloj dijete (engl. *child*) smješta unutar sloja roditelja (engl. *parent*). Unutar slojeva se dodaju *ViewGroup* objekti. Na slici 6. zadan je primjer grafičkog sučelja klase *Provider* iz aplikacije *GPS Kolektor*. Grafičko sučelje *Android* aplikacije moguće je uređivati u grafičkom uredniku koji je prikazan na slici 6. u sredini. Sa lijeve strane se nalaze widgeti, formati ekrana i druge opcije koje olakšavaju uređivanje ekrana bez potrebe pisanja u *XML*-u. Moguće je definirati za koji tip uređaja se radi *GUI* (*QVGA*), pejsažni ili portretni prikaz i verziju *Android* platforme za koju se radi prikaz. Sa desne strane se nalazi outline preglednik. Outline sadrži hijerarhijski prikaz *View* i *ViewGroup* objekata. Sloj-roditelj je *ScrollView* koji sadrži sloj-dijete *LinearLayout* i *widget*-e.



Slika 6. *Grafičko sučelje klase Provider.*

Osnovni tipovi slojeva u *Android*-u su linearni (engl. *LinearLayout*), tabularni (engl. *TableLayout*) i relativni (engl. *RelativeLayout*). Svaki od njih pruža jedinstveni skup parametara izgleda koji određuju pozicije elemenata prikaza

i strukturu izgleda. Primjer sa slike 6. preveden u *XML* oblik je vidljiv na tablici 3.

```
1.  <?xml version = "1.0" encoding = "utf-8"?>
2.  <ScrollView xmlns:android = "http://schemas.android.com/apk/res/android"
3.    android:scrollbars = "vertical"  android:layout_width ="fill_parent"
4.    android:layout_below = "@+id/postavke_gpsa"  android:layout_height = "match_parent">
5.
6.    <LinearLayout android:id = "@+id/LinLayout1" android:layout_width = "match_parent"
7.      android:background = "@color/DARK_GRAY" android:layout_below =
8.      "@+id/postavke_gpsa" android:orientation = "vertical" android:layout_height =
9.      "match_parent">
10.                                                    ...
11.   <TextView android:text = "MinTime(miliseconds,default:0): " android:id =
12.       "@+id/postavke_mintime" android:layout_height = "wrap_content"
13.       android:layout_gravity = "left" android:capitalize = "characters"
14.       android:layout_width = "match_parent"></TextView>
15.   <EditText android:layout_height = "match_parent" android:id = "@+id/EDITmintime"
16.        android:freezesText = "true" android:layout_width = "match_parent"></EditText>
17.   <Button android:layout_width = "wrap_content" android:text = "Update time"
18.         android:layout_height="wrap_content" android:id = "@+id/button_update_time">
19.   </Button>
20.                                                    ...
21.   </LinearLayout>   </ScrollView>
```

Tablica 3. *XML zapis grafičkog sučelja.*

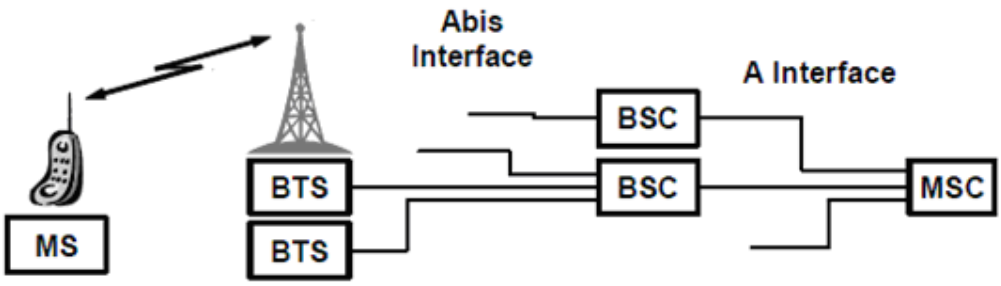
Glavni sloj je *ScrollView* koji sadrži trak za pomicanje. *ScrollView* je određen sa parametrima za dužinu i širinu sloja te orijentaciju traka za pomicanje. Unutar *ScrollView* sloja definiran je linearni sloj (engl. *LinearLayout*) i ostali objekti korisničkog sučelja sa svojim parametrima. Radi preglednosti *XML* dokumenta nisu dodani dijelovi koji su slični prikazanim objektima. Prednost deklariranja korisničkog sučelja *XML* datotekom je u tome što se prikaz aplikacije odvaja od koda koji upravlja njezinim ponašanjem.

3.5. Pozicioniranje mobilnog uređaja

Određivanje položaja korisnika moguće je pomoću *GPS*-a ili pomoću mrežne tehnike pozicioniranja. Određivanje pomoću *GPS*-a je moguće za mobitele koji imaju ugrađen *GPS* čip. Mrežna tehnika je prikladna za mobitele koji nemaju ugrađen *GPS*. Pozicioniranje pomoću *GPS* uređaja je preciznije, a pozicioniranje mrežnom tehnikom nudi samo aproksimaciju položaja. Uglavnom se kombiniraju obadvije metode kako bi međusobno korigirale svoje nedostatke.

3.5.1. Mrežno pozicioniranje

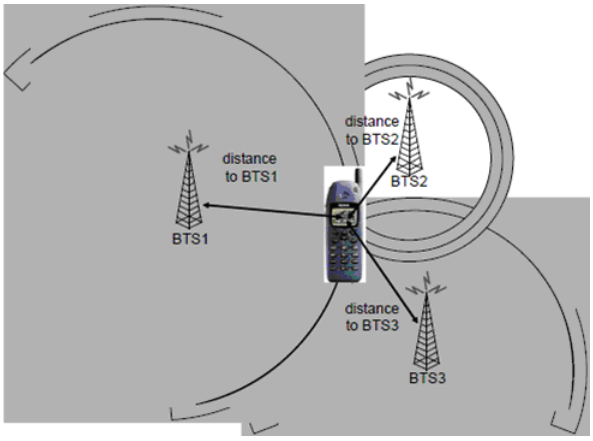
Metoda mrežnog pozicioniranja se koristi infrastrukturom mrežnog operatera kako bi se identificirala lokacija mobilnog uređaja (URL12). *GSM* mreža sastoji se od mobilne stanice, podsustava bazne stanice i mrežnog podsustava (Slika 7.). Mobilnu stanicu nosi sam korisnik. Sastoji se od uređaja i pametne kartice koja se zove *SIM* kartica. *SIM* kartica omogućava određenu neovisnost od uređaja. Vlasnik *SIM* kartice može koristiti usluge *GSM* mreže na bilo kojem uređaju.



Slika 7. *Struktura GSM mreže (URL12).*

Kontrolu i usklađivanje veza sa mobilnom stanicom radi podsustav bazne stanice. Sastoji se od primopredajne bazne stanice (*BTS*) i upravljačkog dijela bazne stanice (*BSC*). Primopredajna bazna stanica sadrži primopredajne uređaje koji definiraju ćeliju i usklađuju protokole radio

prijenosa sa mobilnom stanicom. U vrlo naseljenim mjestima zbog kvalitete signala mora postojati velik broj primopredajnih uređaja. Instaliraju se na povišena mjesta i to najčešće u gradovima na krovove visokih građevina. Zahtjevi koji se postavljaju za primopredajne uređaje su ekonomičnost, niska cijena održavanja i lako premještanje na druge lokacije. Upravljački dio bazne stanice rukovodi sa radio resursima primopredajnih uređaja, frekvencijskim skokovima i radio kanalima. U mrežnom podsustavu glavni dio je mobilni servisni komutacijski centar (*MSC*). *MSC* razmjenjuje pozive između mobilnih stanica i fiksne mreže, te kontrolira i upravlja cjelokupnim sustavom. *MSC* radi poput običnog komutacijskog čvorišta, te zatim vrši registraciju korisnika, provjerava autentičnost, usmjerava pozive pretplatnika (engl. *roaming*). Ove usluge obavlja nekoliko funkcionalnih cjelina, koje zajedno čine mrežni podsustav, te se dalje vežu na fiksnu telefonsku mrežu. Protokol koji se koristi unutar mrežnog podsustava za prijenos signala koristi *SS7* protokol, koji se također koristi u *ISDN* mrežama i telefonskim centralama. Informacija koja je dostupna iz *GSM* mreže je *CellID*. *CellID* predstavlja jedinstveni identifikacijski broj koji se dodjeljuje primopredajnom uređaju, tj. repetitoru *GSM* mreže. Pozicija se određuje na temelju brzine i vremena dospijeća signala od mobilnog uređaja do repetitora. Jačina signala ovisi o udaljenosti repetitora, *multipath* efektu, kretanju objekata na putu signala i drugim faktorima koji smanjuju točnost pozicioniranja. Točnost također ovisi o broju primopredajnih uređaja koji su dostupni za pozicioniranje. U urbanim područjima postiže se najveća točnost. U rijetko naseljenim područjima najmanja je točnost pozicioniranja zbog velike udaljenosti i male gustoće repetitora. Pozicija se određuje metodom triangulacije. Na temelju poznatih koordinata primopredajnih uređaja računa se aproksimacija položaja mobilne stanice (Slika 8.).

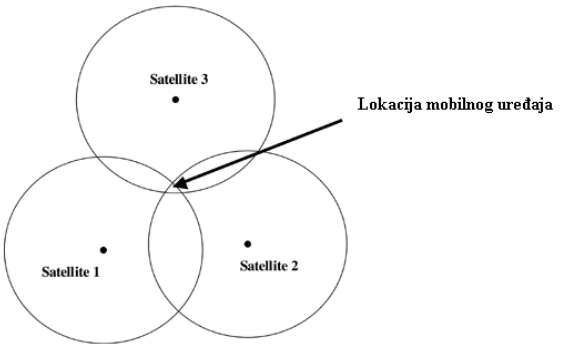


Slika 8. *Mrežna triangulacija mobitela.*

Pristup mrežnom servisu je moguć povezivanjem na internet putem *WiFi*, *GPRS* ili druge mreže dostupnoj mobilnoj stanici. Za uključivanje servisa mrežnog pozicioniranja potrebno je deklarirati dopuštenje *ACCESS_COARSE_LOCATION* u *Android manifest* dokumentu. Točnost mrežnog pozicioniranja se kreće oko 50 m u urbanim područjima, a u slabo naseljenim i ruralnim područjima točnost je znatno degradirana ovisno o udaljenosti primopredajnih uređaja (URL1).

3.5.2. Pozicioniranje pomoću GPS uređaja

Metoda pozicioniranja pomoću *GPS* uređaja je dostupna za mobitele koji imaju ugrađen *GPS* prijemnik. Za pozicioniranje mobilnog uređaja koristi se *TOA* (engl. *Time of Arrival*) kao mjerna metoda. *TOA* je metoda kod koje se za pozicioniranje uzima u obzir vrijeme dolaska signala sa satelita do prijemnika. Za trodimenzionalnu lokaciju potrebna su četiri satelita. U slučaju da je interni uređaj za mjerenje vremena usklađen sa satom satelita, potrebna su samo tri satelita za pozicioniranje (Slika 9.). Točnost *GPS* pozicioniranja se kreće između 5 i 100 metara (URL 1). Točnost degradira više faktora: atmosferski uvjeti, kišno i oblačno vrijeme, napučenost ljudima i zgradama radi *multipath* efekta, nesinkronizirani satovi satelita, broj vidljivih satelita i drugo.



Slika 9. *Lokalizacija mobilnog uređaja.*

Pozicioniranje može potrajati ovisno o vanjskim uvjetima. Primanje prve korekcije može se čekati i do 10 minuta dok *GPS* prijemnik locira vidljive satelite. Ova metoda ima prednost pred mrežnom metodom radi direktnog lociranja mobitela bez posredstva primopredajnog uređaja. Za uključivanje servisa *GPS* pozicioniranja potrebno je deklarirati dopuštenje *ACCESS_FINE_LOCATION* u *Android manifest* dokumentu.

3.5.3. GPS aplikacija

Lokacijskim servisima (*GPS*, *CellID*) pristupa se pomoću opcija klase *LocationManager*. Opcije *LocationManager-a* omogućuju se metodom *getSystemService()* u drugom redu tablice 4. U *Android* sustavu lociranje korisnika djeluje putem povratnog poziva (engl. *callback*). Korisnik predaje zahtjev uređaju za lokacijska ažuriranja pomoću metode *requestLocationUpdates()*. Korisnički zahtjev se dalje prosljeđuje klasi *LocationListener()* koja implementira nekoliko metoda povratnog poziva (engl. *callback*). Klasa *LocationManager* aktivira metode povratnog poziva nakon promjene lokacije ili statusa lokacijskog servisa. Metode zatim izvršavaju operacije koje im je korisnik zadao u programskom kodu.

Od devetog do četrnaestog reda tablice 4., dvije su metode lokacijskog ažuriranja. Korisnik može odabrati između dva lokacijska servisa, mrežnog i

GPS pozicioniranja. Parametri *mintime* i *mindistance* su parametri minimalnog vremenskog razmaka i udaljenosti pri kojoj se lokacija ažurira. Parametar *locationListener* je objekt klase *MyLocationListener* koja reagira na pozive iz određenog lokacijskog servisa. Od sedamnaestog do tridesetisedmog reda tablice 4. nalazi se klasa *MyLocationListener*. Klasa implementira metode *onLocationChanged ()*, *onProviderDisabled (r)*, *onProviderEnabled ()* i *onStatusChanged ()*. Podatke o promjeni lokacije koje želimo prikazati na korisničkom sučelju se uređuju unutar metode *onLocationChanged (Location loc)*. Informacije o nedostupnosti određenog lokacijskog servisa je moguće unijeti unutar metode *onProviderDisabled ()*. Podaci o lokaciji koji se prikazuju na korisničkom sučelju aplikacije *GPS Kolektor* definiraju se unutar metode *onLocationChanged()* od osamnaestog do tridesetog reda tablice 4.

```
1.  public class GPS extends Activity {…
2.     lm = (LocationManager) getSystemService (Context.LOCATION_SERVICE);
3.     locationListener = new MyLocationListener();
4.
5.     mintime=Long.valueOf (Provider.var_mintime);
6.     mindistance=Float.valueOf (Provider.var_mindistance);
7.     if (Provider.var_mintime==null){          mintime=0;          mindistance=0;
8.                                          }
9.     if (Provider.provider2==1) {          lm.requestLocationUpdates(
10.    LocationManager.GPS_PROVIDER, mintime, mindistance,  locationListener);
11.     }
12.    if (Provider.provider2==2) {      lm.requestLocationUpdates(
13.    LocationManager.NETWORK_PROVIDER, mintime, mindistance,  locationListener);
14.     } }
15. ...
16.  public class MyLocationListener implements LocationListener
17.  {
18. public void onLocationChanged (Location loc) {
19.             if (loc != null) {
20.              float Latitude=loc.getLatitude();
21.              float Longitude=loc.getLongitude();
22.              float Altitude=loc.getAltitude();
23.              …
```

```
24.     GPS.this.input.setText ( Latitude + (lat<0?"S":"N") + ":" +
25.                                         Longitude+(long<0?"W":"E")+":"+
26.                                         Altitude+" m"+":"+
27.                                         Accuracy+" m"+":"+
28.                                         Time+" ms since 1.1.1970."+":"+
29.                                         Speed+" m/s"+":"+
30.                                         Azimut+"From North"+":"+Provider+":");}}
31.  public void onProviderDisabled(String provider) { }
32.  public void onProviderEnabled(String provider) {  }
```

Tablica 4. *Dio programskog koda klase GPS.java iz aplikacije GPS Kolektor.*

3.6. Protokoli za razmjenu GPS podataka

U informacijskom sustavu, protokol je dogovoreni postupak za razmjenu informacija između dva ili više uređaja na mreži. Protokol uključuje sintaksu i semantiku informacije, te pravila za razmjenu informacije. Kako bi uređaji na mreži mogli uspješno komunicirati, mora se koristiti isti protokol. Jednak pristup je kod razmjene *GPS* podataka. Za razmjenu *GPS* podataka usvojena su dva protokola, *RTCM* i *NMEA0183*. *RTCM* je protokol koji se koristi za prijenos podataka između *GPS* referentne stanice i rovera, a *NMEA0183* se koristi za prijenos podataka među *GPS* prijemnicima.

3.6.1. RTCM SC-104 format

Radio-tehnička komisija za pomorske usluge (*RTCM*) je međunarodna neprofitna organizacija za standarde. Osnovana je 1947. g. kao savjetodavni odbor za američku vladu. Danas funkcionira kao nezavisna organizacija koju podržavaju članovi iz cijelog svijeta. *RTCM* je podijeljena na posebne odbore kojima je svrha kreiranje standarda vezanih uz pomorske usluge (*GPS*, elektroničke karte, *DGNSS*). Posebni odbor *RTCM SC-104* je stvoren u svrhu uspostave standarda i smjernica za povezivanje između podatkovnih veza baziranih na radio farovima i *GPS* prijemniku, te osiguranje standarda za terestričke *DGPS* stanice. Korekcijski podaci *NTRIP* protokola se distribuiraju u formatu *RTCM SC-104*. *RTCM SC-104* je binarni zapis

podataka koji je podijeljen na više tipova poruka sa različitim sadržajem. Svaka poruka zapisa se sastoji od zaglavlja i tijela. Zaglavlje sadrži vrijeme, vrstu i duljinu poruke, a tijelo sadrži podatke za svaki tip poruke. Postoji nekoliko verzija ovog formata za razmjenu *GPS* podataka internet protokolom:

- *RTCM v2.0* format podržava razmjenu kodnih korekcija za diferencijalni *GPS* (Tablica 5., tip poruke 1-17, 22, 27-30, 37-63).
- *RTCM v2.1* format podržava kodne i fazne korekcije za *kinematiku u realnom vremenu, tj. RTK* (Tablica 5., tip poruke 1-22, 27-30, 37-63).
- *RTCM v2.2* podržava kodne i fazne korekcije za *RTK* uz dodatak *GLONASS* almanaha i korekcija (Tablica 5., tip poruke 1-22, 27-30, 31-63).
- *RTCM v2.3* format podržava kodne i fazne korekcije za *RTK* uz dodatak antenskih parametara, *GLONASS* almanaha i korekcija (Tablica 5., tip poruke 1-24, 27-30, 31-63).

RTCM TIP PORUKE	**OPIS PORUKE**	**RTCM TIP PORUKE**	**OPIS PORUKE**
1	Differential GPS Corrections	20	*RTK Uncorrected Carrier Phases*
2	Delta Differential GPS Corrections	21	*RTK Pseudorange Corrections*
3	GPS Reference Station Parameters	22	*Extended Reference Station Parameters*
4	datum referentne stanice	23	*Antenna Type Definition*
5	zdravlje GPS konstelacije	24	*Reference Station: Antenna Reference Point (ARP) Parameter*
6	Reference Station Datum	25,26	*Undefined*
7	GPS Constellation Health	27	*Extended DGPS Radiobeacon Almanac*
8	GPS Null Name	28,29,30	*Undefined*

9	DGPS Beacon Almanac	31	*Differential GLONASS Corrections*
10	Pseudolite Almanac	32	*Differential GLONASS Reference Station*
11	GPS Partial Correction Set	33	*GLONASS Constellation Health*
12	P-Code Differential Correction	34	*GLONASS Partial Differential Correction Set GLONASS Null Name (N<=1)*
13	C/A Code, L1, L2 Delta Corrections	35	GLONASS Radiobeacon Almanac
14	Pseudolite Station Parameter	36	GLONASS Special Message
15	Ground Transmitter Parameter	37	GNSS System Time Offset
16	GPS Time of Week	38...58	Undefined
17	Ionospheric Delay Message	59	Proprietary Message
18	*GPS Special Message*	60...63	Multipurpose Usage
19	*GPS Ephemerides*		

Tablica 5. *Struktura RTCM SC-104 formata (URL14).*

Neadekvatna učinkovitost verzija *RTCM v2.x* formata dovela je do razvoja učinkovitijeg i jednostavnijeg formata, *RTCM v3.0*. *RTCM v3.0* primarno je namijenjen za podršku *RTK* mjerenjima. Sastoji se od poruka koje sadrže kodna i fazna opažanja, parametre antene, te pomoćne parametre sustava (Tablica 6.).

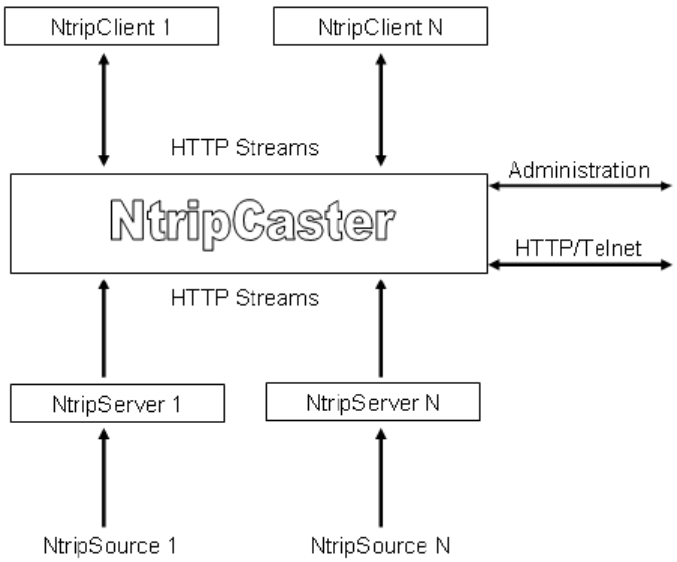
Group Name	Message Type	Message Description
Observations	1001	L1-only GPS RTK Observables
	1002	Extended L1-Only GPS RTK Observables
	1003	L1 & L2 GPS RTK Observables

	1004	Extended L1 & L2 GPS RTK Observables
	1005	L1 Only GLONASS RTK Observables
	1006	ExtendedL1-Only GLONASS RTK Observables
	1007	L1 & L2 GLONASS RTK Observables
	1008	Extended L1 & L2 GPS RTK Observables
Station Coordinates	1009	Stationary RTK Reference Station ARP
	1010	Stationary RTK Reference ARP with Antenna Height
Antenna Description	1011	Antenna Descriptor
	1012	Antenna Descriptor & Serial Number
Auxiliary Operation Information	1013	System Parameters

Tablica 6. *Struktura RTCM v.3.0 formata (URL14).*

3.6.2. NTRIP protokol i aplikacija

NTRIP (engl. *Networked Transport of RTCM via Internet Protocol*) je aplikacijski protokol namijenjen distribuciji *GNSS* podataka preko internet protokola (URL15). *NTRIP* je baziran na *HTTP 1.1* protokolu, a izgrađen je na temeljima *TCP/IP* grupe internet protokola. *TCP/IP* protokoli omogućavaju komunikaciju preko raznih, međusobno povezanih mreža, a također se na njima zasniva i globalna mreža Internet. *NTRIP* je razvijen od strane *Federalne Agencije za Kartografiju i Geodeziju* (njem. *Bundesamt für Kartographie und Geodäsie*). *NTRIP* podržava bežični pristup internetu preko *GSM*, *EDGE* ili *GPRS* mreže. Za prijenos podataka *NTRIP* je usvojio preporučeni standard *RTCM SC-104*. *NTRIP* sustav koristi *HTTP zahtjev-odgovor* protokol za komunikaciju između poslužitelja (engl. *server*) i korisnika (engl. *client*), pri čemu se prijenos informacija ostvaruje nakon što se uspostavi *TCP/IP* veza između servera i klijenta. *HTTP* protokol je implementiran u tri softverske komponente pod nazivom *NtripClient*, *NtripServer* i *NtripCaster* (Slika 10.).



Slika 10. *Struktura NTRIP sustava (URL15).*

NtripSource predstavlja referentnu *GNSS* stanicu koja svoje podatke šalje na *NtripServer*. *NtripCaster* funkcionira kao centrala koja povezuje *NtripServer* i *NtripClient*. Podaci se sa *NtripClient*-a šalju rover uređaju koji, prema primljenim podacima, ispravlja koordinate trenutne lokacije. Komunikacija između *NtripClient*-a i *NtripCaster*-a zasniva se na *HTTP 1.1* protokolu. *NtripCaster* konstantno osluškuje zahtjeve na određenom mrežnom priključku (engl. *port*), čekajući da klijent pošalje zahtjev. Komunikacija između klijenta i poslužitelja uspostavlja se sa *GET / HTTP/1.1* zahtjevom koji sadrži nekoliko slovnih nizova koji čine zaglavlje zahtjeva. U zaglavlju se navode aspekti zahtjeva i paket neobaveznih podataka. Zahtjev (engl. *request*) klijenta će rezultirati slanjem odgovora (engl. *response*) s poslužitelja. Odgovor poslužitelja sastoji se od zaglavlja *"200 OK"* i tijela poruke koje sadrži traženu datoteku ili poruku o grešci. Odmah po ispunjenju zahtjeva klijenta, poslužitelj prekida komunikaciju. Primjer zahtjeva klijenta je je zadan u tablici 7.

```
1.  sock = new Socket (proxy);
2.  InetSocketAddress dest = new InetSocketAddress (Ntrip.this.host, Ntrip.this.port);
3.  try {
4.              sock.connect (dest);
5.              out = new PrintWriter (sock.getOutputStream(), true);
6.              out.print ("GET /" + mountpoint + " HTTP/1.1\r\n");
7.              out.print ("User-Agent: NTRIP test\r\n");
8.              out.print ("Host: " + this.host + "\r\n");
9.              out.print ("Connection: close\r\n");
10.             out.print ("Authorization: Basic " +
11.             ToBase64.encode (this.user + ":" + this.pass) + "\r\n");
12.             out.print ("\r\n");
13.             out.flush();
14. catch (IOException e) {     e.printStackTrace();
15.                             Log.d("TCP","not connected");
16.                         }
```

Tablica 7. *Dio koda klase Ntrip.java iz aplikacije GPSLocator.*

U prvom redu tablice 7. inicijalizira se varijabla *socket*. *Socket* je objekt iz klase *java.net* i predstavlja krajnju točku komunikacije bazirane na internet protokolu između klijenta i poslužitelja. U drugom redu tablice 7. korisnik unosi podatke o adresi i mrežnom priključku poslužitelja na koji se spaja (engl. *host*, *port*). U petom redu tablice 7. kreiran je *Printwriter* objekt koji prevodi poruku zahtjeva u binarni zapis. Od šestog do trinaestog reda tablice 7. nalazi se poruka zahtjeva. Poruka se sastoji od informacija o klijentu (naziv i adresa klijenta), podataka o autorizaciji (engl. *username*, *password*), te zahtjeva za *mountpoint*-om. *Mountpoint* je izvor *GNSS* podataka i određen je svojim identifikatorom. *Mountpoint* predstavlja više referentnih *GNSS* stanica koje šalju podatke *NtripServeru*. *NtripServer* se povezuje sa *NtripClient-om* preko *NtripCaster-a* i šalje podatke traženog *GNSS* izvora. *NtripClient* podatke prosljeđuje roveru koji zatim korigira svoju poziciju (slika 10.). Ako klijent ne pošalje zahtjev sa *mountpoint*-om ili pošalje nevažeći *mountpoint*, *NtripCaster* šalje odgovor sa ažuriranom tablicom raspoloživih *izvora GNSS*

podataka (engl. *Sourcetable*). *Sourcetable* sadrži popis *mountpoint*-ova sa njihovim opisnim podacima na slici 11.

Slika 11. *Sourcetable CROPOS sustava.*

Na slici 11. nalazi se lista izvora *CROPOS* državne mreže referentnih *GNSS* stanica Republike Hrvatske. *NTRIP* izvori su međusobno odvojeni sa *STR;* nizom znakova. Lista izvora sadrži dodatne podatke o svakom *NTRIP* izvoru, uključujući podatke o *RTCM* verziji *NTRIP* izvora, navigacijskom sustavu (*GPS*, *GLONASS*) i nazivu državne mreže *GNSS* stanica (*CROPOS*). U tablici 8. je primjer zahtjeva sa poznatim identifikatorom Ntrip izvora. Poruka zahtjeva se sastoji od informacija o klijentu, traženog formata odgovora sa poslužitelja, autorizacije i slanja koordinata za koje se traži korekcija. Na slici 12. je vidljiv odgovor udaljenog poslužitelja na korisnički zahtjev iz tablice 8.

```
1.  out.print ( "GET /" +  mountpoint + " HTTP/1.1\r\n" );
2.  out.print ( "Host: "+this.host+"\r\n" );
3.  out.print ( "Accept: rtk/rtcm, dgps/rtcm\r\n" );
4.  out.print ( "User-Agent: NTRIP test\r\n" );
5.  out.println ( Ntrip.NTRIP_GPGGA_CREATOR (Latitude, Longitude, Altitude ));
6.  out.print ( "Connection: close\r\n" );
```

```
7.  out.print ( "Authorization: Basic " + ToBase64.encode (this.user+":"+this.pass)+"\r\n" );
8.  out.print ( "\r\n" );
9.  out.flush ();
```

Tablica 8. *Dio koda klase Ntrip.java iz aplikacije GPSLocator.*

Slika 12. *Odgovor poslužitelja u RTCM binarnom zapisu.*

U planu je bilo razviti *Android* aplikaciju koja, između ostalog, koristi transakciju podataka *NTRIP* protokolom. Podaci bi se dalje trebali proslijediti integriranom *GPS* čipu koji bi korigirao očitanje pozicije. Točnost koja se može postići ovim načinom je veća od uobičajenih tehnika pozicioniranja mobitela. Točnost ovisi o cijeni koju je korisnik spreman platiti za određeni servis. Na primjer, *CROPOS* sustav nudi tri servisa (URL16):

- *DSP* je diferencijalni servis pozicioniranja u realnom vremenu s točnošću ispod 1 m.
- *VPPS* je visokoprecizni servis pozicioniranja u realnom vremenu s centimetarskom točnošću.
- *GPPS* je geodetski precizni servis pozicioniranja sa naknadnom procesuiranjem podataka. Točnost ovog servisa je subcentimetarska.

Kod razvoja *GPS* aplikacija komercijalnih mobitela, onesposobljen je pristup kontroli integriranog *GPS* čipa iz nekoliko razloga. U komercijalne uređaje se ugrađuju čipovi manje kvalitete. Kod korištenja *NTRIP* protokola često se mijenjaju brzine prijenosa podataka. To može dovesti do raznih kvarova čipa ili neupotrebljivosti samog uređaja. Iz tog razloga se nije išlo u smjeru daljnjeg unaprijeđivanja ove opcije aplikacije *GPS Kolektor*.

3.6.3. NMEA0183 format i aplikacija

NMEA0183 je protokol za razmjenu podataka među pomorskim elektroničkim uređajima kao što su dubinomjeri, anemometri, sonari i *GPS* uređaji (URL17). *NMEA* protokol je uspostavila američka *Državna pomorska udruga za elektroniku* (engl. *National Marine Electronics Association*). *NMEA0183* format koristi *ASCII* način kodiranja znakova koji je temeljen na engleskoj abecedi. Podaci se razmjenjuju među *GPS* uređajima u obliku *NMEA* rečenica (engl. *sentence*). Rečenica se sastoji od tipa poruke, sadržaja i kontrolnog zbroja (engl. *checksum*). Tip poruke standardne *NMEA* rečenice je riječ koja može imati maksimalno osamdeset znakova,a sastoji od znaka *$*, dvoslovnog prefiksa i troslovnog niza. Dvoslovni prefiks definira tip uređaja koji koristi *NMEA* komunikacijski protokol (npr. za *GPS* prijemnike prefiks je *GP*). Troslovni niz označava vrstu *NMEA* rečenice i utvrđuje interpretaciju sadržaja rečenice. U sadržaju rečenice su podaci od interesa za *GPS* prijemnike (npr. pozicijski podaci, točnost podataka, vidljivi sateliti). Na kraju *NMEA* rečenice se nalazi kontrolni zbroj. Kontrolni zbroj je dvoznamenkasti heksadekadski broj koji se koristi za otkrivanje i ispravljanje pogrešaka kod *NMEA* rečenica. U tablici 9. se nalazi programski kod za primanje *NMEA* rečenica sa *GPS* prijemnika.

```
1. tv = (TextView) findViewById (R.id.textView1);
2. latitude = (EditText) findViewById (R.id.editText1);
3. longitude = (EditText) findViewById (R.id.editText2);
4. Stop_recording = (Button) findViewById (R.id.Stop);
5. nmeaListener = new MyNmeaListener();
```

```
6.  final LocationManager LM = (LocationManager)getSystemService
7.  (Context.LOCATION_SERVICE);
8.  ((LocationManager)getSystemService (Context.LOCATION_SERVICE)).
9.  requestLocationUpdates (LocationManager.GPS_PROVIDER, 0 , 0, new LocationListener() {
10.         public void onLocationChanged (Location loc) {}
11.         public void onProviderDisabled (String provider) {}
12.         public void onProviderEnabled (String provider) {}
13.         public void onStatusChanged (String provider, int status, Bundle extras)        });
14. LM.addNmeaListener ( nmeaListener );
15. Stop_recording.setOnClickListener ( new View.OnClickListener() {
16.       public void onClick (View v) {
17.       LM.removeNmeaListener (nmeaListener);        }}); }
18. public class MyNmeaListener implements GpsStatus.NmeaListener {
19.         public void onNmeaReceived (long timestamp, String nmea) {
20.                 if ( nmea.contains ("$GPGGA")){
21.                 String nmea_array [ ] =nmea.split(",");
22.                 //latitude
23                  String raw_lat  = nmea_array[2];
24.                 String lat_deg = raw_lat.substring(0,2);
25.                 String lat_min1 = raw_lat.substring(2, 4);
26.                 String lat_min2 = raw_lat.substring(5);
27.                 String lat_min3 = "0." + lat_min1 + lat_min2;
28.                 float lat_dec = Float.parseFloat (lat_min3)/.6f;
29.                 float lat_val = Float.parseFloat (lat_deg) + lat_dec;
30.                 String lat_direction = nmea_array[3];
31.                 If (lat_direction.equals("N")){
32.                  } else {                lat_val = lat_val * -1; }
33.                  // Longitude
34.                 String raw_lon = nmea_array[4];
35.                 ...   }
36.                 tv.append (" NMEA: "+"\n"+nmea+"\n");
37.                 latitude.setText ("\n"+lat_val+"\n");
38.                 longitude.setText ("\n"+lon_val+"\n");        };   }};  }
```

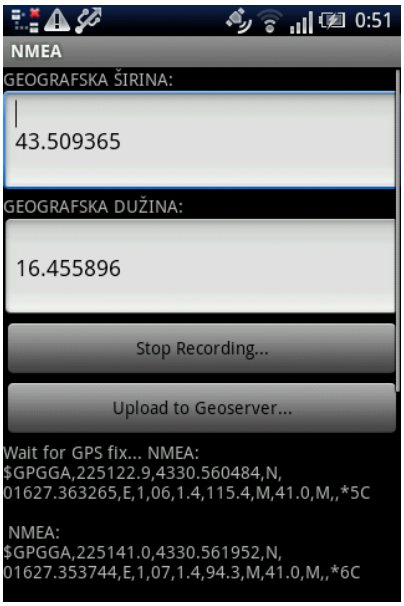
Tablica 9. *Kod klase Nmealistener.java iz aplikacije GPS Kolektor.*

Od prvog do šestog reda tablice 9. nalaze se deklarirane varijable koje se koriste u ostatku koda. U osamnaestom redu tablice 9. implementirano je sučelje *GpsStatus.NmeaListener* za primanje *NMEA* rečenica. Preko klase *LocationManager* traži se periodičko ažuriranje *GPS* informacija. Primitak

nmea rečenice registrira metoda *onNmeaReceived()* koja je definirana sa parametrima *timestamp* i *nmea*. *Timestamp* označava vrijeme primitka *NMEA0183* rečenice, a *nmea* je sama *NMEA0183* poruka. Postoji nekoliko tipova često korištenih standardnih *NMEA0183* tipova rečenica:

- *$GPBOD* (engl. *Bearing , Origin to Destination*)
- *$GPGGL* (engl. *Geographic Latitude and Longitude*)
- *$GPGSA* (engl. *GPS DOP*, *Active Satellites*)
- *$GPGSV* (engl. *GPS Satellites in View*)
- *$GPGGA* (engl. *Global Positioning System Fix Data*)

Za potrebe aplikacije korišten je samo *GPGGA* tip podataka. Rečenica raspolaže sa podacima o *UTC* vremenu, geografskoj širini i dužini, kvaliteti očitanja pozicije, broju vidljivih satelita, HDOP-u, nadmorskoj visini, visini geoida, vremenu zadnjeg *DGPS* ažuriranja, identifikatoru referentne *DGPS* stanice i kontrolni zbroj (Slika 13.).



Slika 13. *Grafičko sučelje za Nmealistener aplikacije GPS Kolektor.*

3.7. Geoserver

Geoserver je poslužiteljska aplikacija otvorenog koda (engl. *open source*) napisana u *Java* programskom jeziku (URL18). Svrha ove aplikacije je

posluživanje i uređivanje prostornih podataka. Obzirom da je aplikacija pisana u *Javi*, izvršava se u *JVM* virtualnom operativnom sustavu i moguće je koristiti na većini dostupnih operativnih sustava (*Windows*, *Linux*, *Mac OS X*). *Geoserver* je aplikacija izdana pod *GPL* (engl. *GNU General Public Licence)* licencom na čijem razvoju radi organizacija ljudi diljem svijeta. Ova aplikacija prostorne podatke poslužuje u popularnim i standardiziranim rasterskim i vektorskim oblicima, kao što su *SVG*, *PNG*, *GML* i drugi. Posluživanje se vrši preko tri protokola za rad sa prostornim podacima, kao što su *WMS* (engl. *Web Map Service*), *WFS* (engl. *Web Feature Service*) i *WCS* (engl. *Web Coverage Service*).

3.7.1. Web Feature Service –Transactional

WFS-T je protokol koji omogućuje transakciju prostornih podataka putem interneta koristeći standardizirane upite. Održava ga i razvija *OGC* (engl. Open Geospatial Consortium) standardizacijska organizacija. Za razliku od *WMS* protokola, kod kojeg je rezultat upita rasterska slika traženog područja, *WFS* protokol omogućuje vraćanje vektorskih podataka, tj. geometrije sa pripadajućim atributima. Prednost korištenja ovog protokola u odnosu na *WMS* protokol je mogućnost izvođenja bogatijih prostornih analiza. Prostorni podaci se *WFS* protokolom najčešće dostavljaju korisniku u *GML* formatu. Moguće je također koristiti *ESRI Shape*, *JSON* i ostale podatkovne oblike, ali je najzastupljeniji *GML*.

GML je format za vektorske prostorne podatke baziran na *XML*-u koji je definirao *OGC* konzorcij. Ovaj format omogućuje pohranu svih prostornih podataka u obliku geometrije (točke, linije i poligoni) sa pripadajućim tekstualnim atributnim podacima. *WFS-T* je proširena verzija *WFS*-a koja osim dohvaćanja prostornih podataka (engl. *GET*) omogućuje i njihovu izmjenu (engl. *POST*). Izmjena podataka uključuje stvaranje novih prostornih podataka i brisanje te izmjenu postojećih prostornih podataka. Kod rada sa *Geoserver*-om svaka transakcija je nedjeljiva (engl. *atomic*). To znači da se u slučaju neuspjele transakcije, postojeći podaci sa *Geoserver*-a ne mijenjaju. *WFS* upit se šalje *HTTP POST* protokolom između klijenta i poslužitelja

(Tablica 10.). *POST* je *HTTP* metoda zahtjeva koja se koristi kada klijent treba poslati neke podatke poslužitelju (npr. učitavanje datoteke, slanje popunjenog obrasca). Također postoji *GET HTTP* metoda, kod koje se poslužitelju šalje samo web adresa i zaglavlje.

```
1.  public class TestWfsPost extends Activity {
2.  public void onCreate(Bundle savedInstanceState) {
3.        super.onCreate(savedInstanceState);
4.        setContentView(R.layout.main);
5.  HttpPost httppost = new HttpPost ( "http://" + host + ":" + port + "/geoserver/wfs");
6.  try {
7.        DefaultHttpClient httpclient = new DefaultHttpClient ();
8.        httpclient. getCredentialsProvider(). setCredentials(
9.        new AuthScope( host , Integer.valueOf (port), AuthScope. ANY_REALM),
10.       new UsernamePasswordCredentials (username , password));
11.
12. String xml=
13. "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
14. "<wfs:Transaction service=\"WFS\" version=\"1.0.0\"\n" +
15. "xmlns:wfs=\"http://www.opengis.net/wfs\"\n" +
16. "xmlns:"+workspace1+"=\""+namespace_URI+"\"\n" +
17. "xmlns:gml=\"http://www.opengis.net/gml\"\n" +
18. "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"\n"+
19. "xsi:schemaLocation=\"http://www.opengis.net/wfs http://schemas.opengis.net/wfs/1.0.0/WFS-
20. transaction.xsd " + namespace_URI + "http://" + host  ":" + port +
21. "/geoserver/wfs/DescribeFeatureType?typename=" + workspace1 + ":" + layername1 + "\">\n"+
22. "<wfs:Insert>"+
23. "<feature:" + layername + " xmlns:feature=\"" + namespace_URI + "\">\n"+
24. "<feature:the_geom>"+
25. "<gml:Point xmlns:gml=\"http://www.opengis.net/gml\" srsName=\"EPSG:4326\">"+
26. "<gml:coordinates decimal=\".\" cs=\",\" ts=\" \">" + point + "</gml:coordinates>"+
27. "</gml:Point>"+
28. "</feature:the_geom>"+
29. "</feature:" + layername + ">"+
30. "</wfs:Insert>"+
31. "</wfs:Transaction>";
32. StringEntity se;
33. se = new StringEntity (xml, HTTP.UTF_8)
34. httppost. setHeader("Content-type", "text/xml");
35. httppost. setEntity(se);
```

```
36.  HttpResponse response = httpclient.execute(httppost);
37.  //DEBUG
38.  HttpEntity entityResp = response.getEntity();
39.  String entityString = EntityUtils.toString(entityResp);
40.  httppost. setHeader ("Content-type", "text/xml");
41.  httppost. setEntity (se);
42.  HttpResponse response = httpclient.execute(httppost);
43.   Log.d ("TCP", entityString);
44.  }
45.   catch (UnsupportedEncodingException e) {                    e.printStackTrace();}
46.   catch (ClientProtocolException e) {                  e.printStackTrace();}
47.   catch (IOException e) {                        e.printStackTrace();}
```

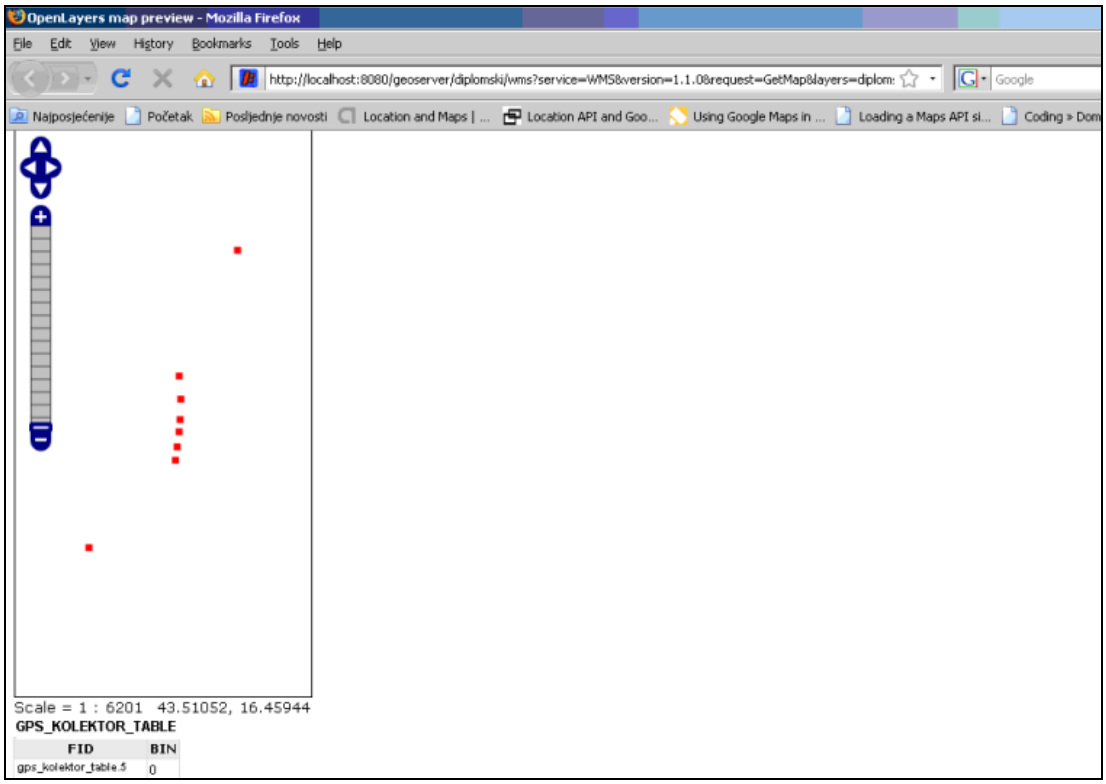
Tablica 10. *Kod klase Nmealistener.java iz aplikacije GPS Kolektor.*

Veza sa poslužiteljem Geoserver-om je ostvarena preko nekoliko objekata iz klase *org.appache.http.* Za otvaranje veze sa poslužiteljem koristi se objekt *DefaultHTTPClient* sa parametrima adrese poslužitelja i korisničkog imena sa lozinkom (tablica 10., red 7-10). Poruka zahtjeva koji se šalje je *WFS* upit u *XML* obliku (tablica 10., red 13-31). Prvo je potrebno referencirati resurse koji se koriste za unos prostornih podataka (tablica 10., red 15-21). Struktura podataka u *Geoserver*-u je sljedeća, kreira se radna datoteka (*workspace*) koja ima svoju adresu u stilu domene (*namespace_URI*). Obzirom da postoje različiti formati podataka koji se spremaju u jednu radnu datoteku kreira se više spremišnih datoteka (*store*). Svaka spremišna datoteka može sadržavati više slojeva (*layer*). Sve se navedeno treba referencirati u *WFS* upitu. Nadalje se definira tip transakcije, što je u ovom slučaju *insert*. Unutar *insert* transakcije se definira geometrija u *GML* obliku. Nakon slanja *WFS* upita pomoću *httppost* objekta, poslužitelj odgovara u formi objekta *httpentity* sa sadržajem uspjele transakcije, ili neuspjele transakcije sa tipom pogreške. Na slici 14. je prikaz grafičkog sučelja WFS upita aplikacije GPS Kolektor. Geoserver na primljeni ispravni zahtjev, šalje poruku o uspješnosti transakcije.

Slika 14. *Grafičko sučelje za WFS transakcije aplikacije GPS Kolektor.*

3.7.2. OpenLayers

OpenLayers je skup gotovih programskih funkcija (engl. *framework*) namijenjenih ubrzavanju pisanja *webGIS* aplikacija u *JavaScript* programskom jeziku (URL19). Razvoj *OpenLayers*-a je započet od tvrtke *MetaCarta* u lipnju 2006. godine, u obliku otvorenog koda. *OpenLayers* je od studenog 2007. godine postao projekt *OsGeo*-a (*Open Source Geospatial Foundation*), neprofitne organizacije sa ciljem promoviranja i razvoja tehnologija otvorenog koda za rukovanje prostornim podacima. Korištenjem *OpenLayers*-a, prostorni podaci se na relativno brz i lagan način prikazuju u internet pregledniku (Slika 15.).



Slika 15. *OpenLayers.*

Na slici je prikazan rezultat *WFS* transakcija umetanja položajnih točaka iz *NMEA0183* formata koji je registrirala aplikacija *GPS Kolektor.* Za iscrtavanje dobivenih prostornih podataka koriste se resursi korisničkog računala. U ovom slučaju su resursi smješteni u *POSTGIS* bazi podataca *gps_locator_table*. *OpenLayers* ima sličnu svrhu kao i *Google Maps* i *MSN Virtual Earth* sučelja za razvoj aplikacija (engl. *Application Programming Interface-API*), ali za razliku od njih je izdan u obliku otvorenog koda. Zbog toga je za krajnjeg korisnika besplatan i vrlo je široko korišten, te ga podržava i razvija velik broj individualaca i organizacija diljem svijeta. Ovaj skup gotovih funkcija podržava rad sa brojnim vektorskim i rasterskim formatima podataka (npr. *GML*, *PNG*) te protokolima (*WMS*, *WFS*, *WCS*).
Iz tih razloga je *OpenLayers* idealan alat za korištenje u kombinaciji sa *Geoserver*-om, kako bi se mogle razvijati internet aplikacije koje prikazuju, modificiraju i razmjenjuju prostorne podatke.

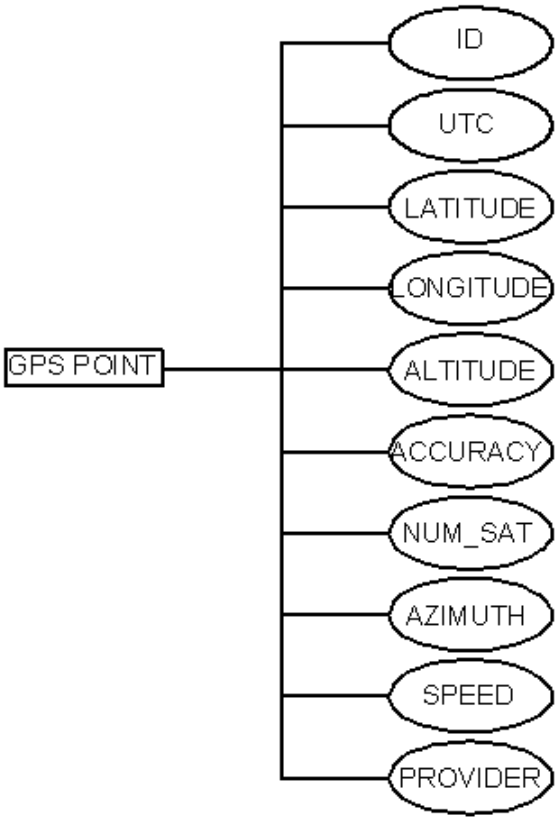
3.8. Baza podataka

Baza podataka je skup međusobno povezanih podataka. Podaci u bazi podataka su spremljeni na disku, bez redundancije, mogu se jednostavno koristiti (npr. sortirati i pretraživati) i mijenjati (npr. nadopunjavati i brisati). Baza podataka služi jednoj ili više aplikacija na optimalan način, gdje su podaci neovisni o programima kojima se obrađuju. Programi koji koriste podatke iz baze podataka ne pristupaju podacima direktno, već komuniciraju sa *DBMS* sustavom za upravljanje bazom podataka. *DBMS* je softver koji djeluje kao sučelje između korisnika i zapisa baze podataka na disku. *DBMS* omogućuje izradu, održavanje i rad sa bazom podataka. Zapisu iz baze podataka pristupa se preko upita (engl. *query*). Prema tipu strukturiranja baza podataka, *DBMS* se dijeli na hijerarhijski, mrežni, relacijski, deduktivni, objektni i objektno-relacijski model. Većina suvremenih baza podataka se služi relacijskim modelom. Relacijski model *DBMS*-a je zasnovan na matematičkom pojmu relacije. Podaci i veze među podacima prikazuju se preko dvodimenzionalnih tablica. Relacija se uspostavlja preko primarnih ključeva entitetnih tablica. Za uspostavljanje relacija među relacijskim tablicama primarni ključevi drugih tablica postaju strani ključevi. Za razliku od drugih modela, jezik za manipulaciju bazom podataka je standardiziran (*SQL-92*). Jezik nije samo upitni, već služi za definiciju i manipulaciju podacima. Osim manipulacije tablicama, *SQL* omogućuje definiciju integriteta podataka, prava pristupa i kontrolu transakcija.

3.8.1. SQLite i aplikacija

SQLite je relacijski sustav za upravljanje bazama podataka temeljen na biblioteci pisanoj u C programskom jeziku (URL20). Programski kod *SQLite*-a je zasnovan na *public domain* principu. To znači da se programski kod može kopirati, modificirati, objavljivati i distribuirati za bilo koju komercijalnu ili ne-komercijalnu svrhu. Transakcije u *SQLite*-u funkcioniraju po *ACID* principu, tj. kod transakcije podaci ostaju postojani pri padu sustava ili nestanku električne energije. Zbog malog zauzimanja prostora na disku i brze izvedbe standardnih operacija nad bazama podataka, *SQLite* je integriran u *Android*

sustavu. Koristi za spremanje aplikacijskog sadržaja i izmjenjivanje podataka među aplikacijama u *Android*-u. Podatke iz *SQLite* baze podataka dobavlja i raspoređuje *Android*-ova komponenta *Content Provider*, tj. dobavljač sadržaja. Korištenje *SQLite*-a u *Android*-u ne zahtijeva nikakve postavke i administraciju. Kreiranje baze podataka za aplikaciju *GPS Kolektor* možemo podijeliti na nekoliko cjelina, koje uključuju dizajniranje sheme baze podataka, kreiranje tablica i popunjavanje tablice s podacima. Relacijski model baze podataka čini veći broj tablica koje nazivamo relacijama. Relaciju čine redovi i stupci koji sadrže podatke o nekoj klasi entiteta. Entitet je dio stvarnog svijeta, a opisuje se sa atributima. Skup atributa jednog entiteta je relacija. Stupac relacije je određen nazivom atributa i njegovim dopuštenim vrijednostima. Redoslijed stupaca u relacijskom modelu nije bitan. Atribut je element relacijskog modela koji određuje neko svojstvo entiteta. U kreiranju baza podataka razlikujemo konceptualni, logički i fizički model. Konceptualni model baze podataka ima cilj opisati semantiku baze podataka. Za prikaz konceptualnog modela koristi se *DSD* (engl. *Data Structure Diagram) dijagram* koji raspolaže sa grafičkim zapisima o entitetima, njihovim relacijama i ograničenjima (engl. *constraint*) ako ih ima (Slika 16.).



Slika 16. *Model Entitet-veza baze podataka GPS Kolektor-a.*

Logički model baze podataka ima cilj opisati sadržaj baze podataka, a ne način rada. Nakon kreiranja logičkog modela baze podataka, slijedi kreiranje fizičkog modela. Fizički model uključuje implementaciju logičkog modela baze podataka na čvrstom disku. Za realizaciju fizičkog modela baze podataka u aplikaciji *GPS Kolektor* služi klasa *DataHelper. java* (Tablica 11.). U klasi su prikazana tri tipa interakcije sa bazom podataka,tj. unos (engl. *insert*), pretraživanje (engl. *query*) i brisanje podataka (engl. *delete*) na slici.

```
1.  public class DataHelper {
2.  ...
3.  public long insert (String latitude, String longitude, String altitude, String accuracy, String
4.  UTC_time, String speed, String azimut, String num_sat, String provider) {
5.      this. insertStmt. bindString (1, latitude);
6.       ...
```

```
7.      this. insertStmt. bindString (9, provider);
8.      return this. insertStmt. executeInsert();
9.   }
10.   public void deleteAll() {
11.      this.db.delete(TABLE_NAME, null, null);
12.     }
13.  public List<String> selectAll() {
14.      List <String> list = new ArrayList <String> ();
15.      Cursor cursor = this. db. query( TABLE_NAME, new String[] { "latitude", "longitude",
16.  "altitude", "accuracy", "UTC_time", "speed",  "azimut",  "num_sat",  "provider" },  null, null,
17.  null, null, "latitude desc");
18.      if  (cursor. moveToFirst()) {
19.        do {
20.                    list.add(cursor.getString(0));
21.                     ...
22.                    list.add(cursor.getString(8));
23.        } while (cursor. moveToNext());
24.      }
25.      if  (cursor != null && !cursor.isClosed()) {
26.        cursor.close();
27.      }
28.      return list;
29.      }
```

Tablica 11. *Dio koda klase DataHelper.java iz aplikacije GPS Kolektor.*

Pretraživanje podataka i prikaz filtrirane *SQLite* baze podataka u *Android*-u se radi pomoću pokazivača (engl. *cursor*). Pokazivač je objekt iz klase *android.database*. Za slanje upita pokazivač nudi dvije opcije, *rawquery* i *query*. Kod metode *rawquery* korisnik sam upisuje *SQL* upit, a kod metode *query* korisnik unosi parametre za rezultat iz definirane tablice, stupcima,sortiranjem i brojem redova. Rezultat upita pokazivač sprema u redove kojima korisnik pristupa po rednom broju dok god je pokazivač aktivan. Metodom *cursor.close()* pokazivač se deaktivira. Za kreiranje i ažuriranje baze podataka služi klasa *SQLiteOpenHelper* (Tablica 12.) koja implementira metode *onCreate* za kreiranje tablice i *onupgrade* za ažuriranje tablice.

```
1.  final class OpenHelper extends SQLiteOpenHelper {
2.      OpenHelper (Context context) {
3.        super (context, DATABASE_NAME, null, DATABASE_VERSION);     }
4.  @Override
5.  public void onCreate(SQLiteDatabase db) {
6.  db.execSQL ( "CREATE TABLE " + TABLE_NAME + " ( id INTEGER PRIMARY KEY, latitude
7.  TEXT , longitude TEXT, altitude TEXT, accuracy TEXT, UTC_time TEXT, speed TEXT,
8.  azimut TEXT, num_sat TEXT, provider TEXT )" );
9.      }
10. @Override
11. public void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) {
12. db. execSQL( "DROP TABLE IF EXISTS " + TABLE_NAME);
13. onCreate(db);
14.  }
```

Tablica 12. *Dio koda klase DataHelper.java iz aplikacije GPS Kolektor.*

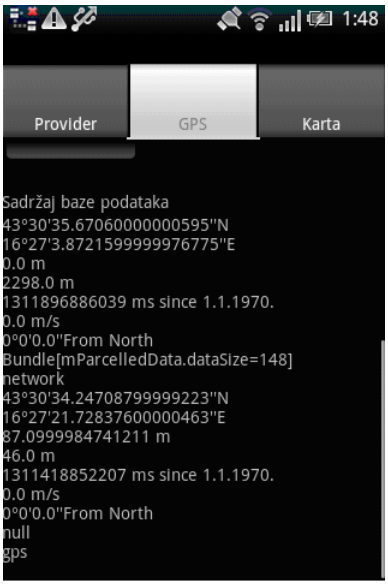
Kreirana baza podataka se automatski sprema u direktoriju aplikacije (npr. *DATA /data/APP_NAME/ databases/ ime_baze_podataka.db*). Za kodirane uređaje pristup tom dijelu nije moguć pa je dodana opcija eksportiranja baze podataka na memorijsku karticu radi mogućnosti naknadne analize baze podataka. Klasa *ManageData* iz aplikacije *GPS Kolektor* nudi opciju eksportiranja baze podataka u *SQLite* formatu baze podataka i *XML*-u (Tablica 13.). Također je uklonjeno ograničenje aplikaciji za pristup memorijskoj kartici uređaja (*permission-uses WRITE_EXTERNAL _STORAGE*).

```
1.  File dbFile = new File (Environment.getDataDirectory() +
2.  „/data/android.hr.dipl/databases/GPS_baza_podataka1.db");
3.  File exportDir = new File (Environment.getExternalStorageDirectory(), GPS_baza_podataka1");
4.                  if (!exportDir.exists()) {
5.                    exportDir.mkdirs();
6.                  }
7.   File file = new File (exportDir, dbFile.getName());
8.                  try {
```

```
9.                  file.createNewFile();
10.                 this.copyFile(dbFile, file);
11.                 return true;
12.                 } catch (IOException e) {
13.                 Log.e(Aplikacija.APP_NAME, e.getMessage(), e);
14.                 return false;
15. }}}
```

Tablica 13. *Dio koda klase ManageData.java iz aplikacije GPS Kolektor.*

U prvom redu tablice 13. definirana je varijabla baze podataka. Baza podataka je smještena u aplikacijskom direktoriju koji je nedostupan korisniku. U trećem redu tablice 13. definirana je prazna datoteka koja se zove *GPS_baza_podataka1*, a smještena je u memorijskoj kartici. Adresa memorijske kartice se dobije pomoću metode *Environment.getExternalStorageDirectory()*. Od sedmog do jedanaestog reda tablice 13. se kreira datoteka koja je kopija baze podataka iz aplikacijskog direktorija. Korisničko sučelje aplikacije *GPS Kolektor* koje se bavi sa spremanjem podataka u bazu i ispisom podataka je vidljivo na slici 17.



Slika 17. *Baza podataka aplikacije GPS Kolektor.*

3.9. Kartografski prikaz baze podataka

GPS Kolektor aplikacija ima opciju kartografskog prikaza lokacija koje su spremljene u bazi podataka. Za kartografski prikaz koristi se *Google Maps* servis*. Google Maps* je uslužna aplikacija za web kartiranje koja je besplatna za nekomercijalnu upotrebu. Pruža usluge pregledavanja satelitskih snimaka, planiranja trase putovanja, lokatora traženih mjesta, itd. Dopušta jednostavnu implementaciju na različite web stranice, prilagođavanje određenim potrebama i kombiniranje sa raznim aplikacijama. Za primjenu *Google Maps* servisa u *Android* aplikaciji, potrebno je proširiti *Android* platformu na *Google API* i zatražiti besplatnu licencu za nekomercijalnu upotrebu. *Google API* je programski dodatak za *Android* koji omogućuje razvoj aplikacija koje koriste *Google*-ove biblioteke i servise. Najvažniji dio tog programskog dodatka je eksterna biblioteka *com.google.android.maps*. Biblioteka raspolaže vrlo korisnim mogućnostima kartiranja, nudi opcije skidanja sadržaja sa *Google Maps* servisa, prikaz i pohranu skinutog sadržaja, te mnoštvo opcija prikaza i manipulacije sa sadržajem. *Google* izdaje besplatni *MD5* ključ za nekomercijalne aplikacije, a zahtjev za izdavanjem se šalje na web stranicu *http://code.google.com/android/add-ons/google-apis/maps-api-signup.html.*
MD5 otisak (engl. *fingerprint*) je kratka sekvenca bitova koja služi za autentikaciju ili pretraživanje certifikata. Certifikat definira korisnik pri eksportu aplikacije iz projekta. Generiranje *MD5* otiska radi se pomoću *Android*-ovog alata *debugkeystore* i pomoću sljedeće naredbe iz *command line*-a:

```
$ keytool -list -alias androiddebugkey \ -keystore
<path_to_debug_keystore>.keystore \ -storepass android -keypass
android
```

Parametar *–alias* označava korisničko ime, *-storepass* i *–keypass* su lozinke aplikacije, a *path_to_debug_keystore* je puni naziv direktorija u kojem se alat za dobivanje *MD5* otiska nalazi. Ključ se zatim upiše u polje XML sloja koji definira korisničko sučelje karte, što je vidljivo u tablici 14.

```
1.  <?xml version = "1.0" encoding = "utf-8"?>
2.  <RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
3.     android:layout_width = "fill_parent"
4.     android:layout_height = "fill_parent">
5.
6.     <com.google.android.maps.MapView
7.        android:id = "@+id/mapView"
8.        android:layout_width = "fill_parent"
9.        android:layout_height = "fill_parent"
10.       android:enabled = "true"
11.       android:clickable = "true"
12.       android:apiKey = "0f4cEhkpyTqA2GdepzKY68mFbYLPp8R4wp8s3KQ"
13.       />
14.
15. <Button android:text = "SatelliteView" android:id = "@+id/SatelliteV" android:layout_width =
16. "wrap_content"  android:clickable = "true"  android:layout_height = "50dip" android:gravity =
17. "center_vertical"  android:layout_alignParentBottom = "true" android:longClickable = "true" >
18. </Button>
19.
20. <Button android:layout_alignParentBottom = "true" android:layout_toRightOf = "@+id/SatelliteV"
21. android:layout_width = "wrap_content" android:layout_height = "50dip" android:text = "Trans
22. coord" android:id = "@+id/button_trans_coord" ></Button>
23.  </RelativeLayout>
```

Tablica 14. *XML GUI klase Karta.java iz aplikacije GPS Kolektor.*

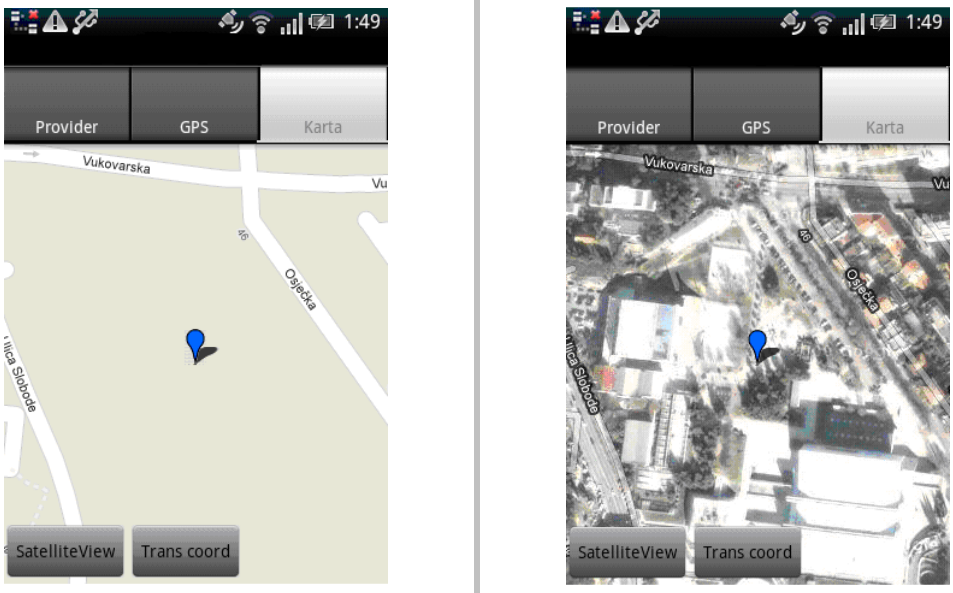
Za prikaz karte na grafičkom sučelju *Android* aplikacije služi klasa *MapView*. Klasa *MapView* raspolaže sa svim elementima korisničkog sučelja koji su potrebni za upravljanje kartom (npr. zumiranje, mijenjanje prikaza, geokodiranje). U aplikaciji *GPS Kolektor* su dodani elementi iz klase *MapView* za korisničko zumiranje na točku (Tablica 15.).

```
1.  ZoomControls zoomControls  = (ZoomControls) mapView. getZoomControls();
2.  zoomControls .setLayoutParams ( new ViewGroup. LayoutParams ( LayoutParams.
3.  WRAP_CONTENT, LayoutParams. WRAP_CONTENT ));
4.  mapView. addView ( zoomControls );
```

```
5.  mapView. displayZoomControls ( true );
```

Tablica 15. *Dio koda klase Karta.java iz aplikacije GPS Kolektor.*

Također su dodane opcije satelitskog i klasičnog prikaza karte preko metoda *mapView.setStreetView(true)* i *mapView.setSatellite(true)* na slici 18.



Slika 18. *Klasični i satelitski prikaz karte sa Google Maps servisa.*

Za lociranje na karti, koristi se objekt *GeoPoint* iz istoimene klase. GeoPoint je definiran sa parametrima *latitudeE6* i *longitudeE6* koji predstavljaju geografsku širinu i dužinu, izraženu u mikrostupnjevima (Tablica 16.).

```
1.  db = openOrCreateDatabase ("GPS_baza_podataka1.db",
2.  SQLiteDatabase.CREATE_IF_NECESSARY , null);
3.  db.setVersion(1);
4.  db.setLocale(Locale.getDefault());
5.  Cursor cur = db.query("GPS_table",null, null, null, null, null, null);
6.  cur.moveToFirst();
```

```
7.  while (cur.isAfterLast() == false) {
8.  String dlat=cur.getString(1).substring(0, (cur.getString(1).indexOf("°")));
9.  …
10. float latitude=Float.valueOf(dlat)+Float.valueOf(mlat)/60+Float.valueOf(slat)/3600;
11. String dlong=cur.getString(2).substring(0, (cur.getString(2).indexOf("°")));
12.  …
13. float longitude=Float.valueOf(dlong)+Float.valueOf(mlong)/60+Float.valueOf(slong)/3600;
14. Karta.this.p= new GeoPoint((int)(latitude*1E6) , (int)(longitude*1E6));
15. // move to location
16. mapView.getController().animateTo(Karta.this.p);
17. // redraw map
18. mapView.postInvalidate();
19. cur.moveToNext();
20. }
21. cur.close();
22. mc.setCenter(p);
23. mc.animateTo(p);
24. }
25. catch(Exception e) {          e.printStackTrace();  }
```

Tablica 16. *Dio koda klase Karta.java iz aplikacije GPS Kolektor.*

U prvom redu tablice 16. metodom *openorcreatedatabase()* otvara se pristup *GPS_table* bazi podataka. Zatim se kreira objekt *Cursor* za pretraživanje baze radi ucrtavanja položajnih točaka na *Google Maps* kartu. U četrnaestom redu tablice 16. kreiran je objekt *GeoPoint* za ucrtavanje položajnih točaka iz baze na kartu. Za svaku novu točku spremljenu u bazu podataka, karta se ažurira, dodavajući nove lokacije (*mapView. postinvalidate()*). Lokacije na karti se prikazuju pomoću markera. Kako bi se markeri mogli koristiti, potrebno je uključiti sloj koji preklapa kartu, tj. *ItemizedOverlay*. Programski kod zadužen za opciju markera na karti vidljiv je u tablici 17.

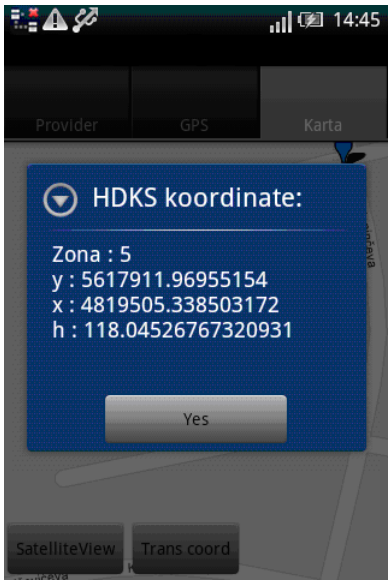
```
1.  import com.google.android.maps.ItemizedOverlay;
2.  import com.google.android.maps.MapView;
3.  import com.google.android.maps.OverlayItem;
4.  import…
5.  public class MapOverlay extends ItemizedOverlay {
```

```
6.  private ArrayList<OverlayItem> mOverlays = new
7.  ArrayList<OverlayItem>();
8.  Context mContext ;
9.  public MapOverlay(Drawable defaultMarker) {
10. super(boundCenterBottom(defaultMarker));
11. }
12. public MapOverlay(Drawable defaultMarker, Context   context) {
13.                                     this(defaultMarker);
14.                                      mContext = context;
15.                                    }
16.             @Override
17.              protected boolean onTap(int index) {
18.              OverlayItem item = mOverlays.get(index);
19.              AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
20.              dialog.setTitle(item.getTitle());
21.              dialog.setMessage(item.getSnippet());
22.              dialog.setPositiveButton("Yes", new OnClickListener() {
23.              public void onClick(DialogInterface dialog, int which) {
24.             dialog.dismiss();
25.                 }
26.              });
27.              dialog.show();
28.              return true;
29.           }
30.           @Override
31.           protected OverlayItem createItem(int i) {
32.             return mOverlays.get(i);
33.           }
34.           @Override
35.           public int size() {
36.             return mOverlays.size();
37.           }
38.   public void addOverlay(OverlayItem item) {
39.       mOverlays.add(item);
40.       populate();
41.   } }
```

Tablica 17. *Dio koda klase MapOverlay.java iz aplikacije GPS Kolektor.*

ItemizedOverlay je klasa iz *Google*-ove eksterne biblioteke koja upravlja sa elementima preklopa karte. Za svaku novu poziciju koja je spremljena u bazu

podataka, pozivaju se metode *itemizedoverlay.addOverlay(item)* i *mapOverlays.add (itemizedoverlay)*. *ItemizedOverlay* dodaje na kartu markere koji su spremljeni u direktoriju resursa (engl. *resource*) projekta. Markeri su interaktivni, tj. klikom na marker otvara se dijalog u kojem se prikazuju informacije o *HDKS* koordinatama i broj zone kojoj položajna točka pripada (Slika 19.).



Slika 19. *Informacije o HDKS koordinatama markera sa Google Maps-a.*

3.9.1. Transformacija WGS84 u HDKS

Za računanje položaja točaka kod nas se kao referentna ploha koristi rotacijski elipsoid *Bessel 1841* s ishodištem Hermanskogel kraj Beča. Položaj pojedine točke određuju elipsoidne koordinate geografska širina (φ) i geografska duljina (λ). Visinu točke predstavlja najkraća udaljenost od geoida (srednja razina mora). Orijentacija Besselovog elipsoida temelji se na astro-geodetskim opažanjima iz 1841. godine. Besselov elipsoid odstupa od *WGS84* elipsoida najviše po geografskoj dužini oko 17", dok je po širini 1''. Za prikaz položaja točaka u ravnini kod nas se koristi Gauss-Kruegerova projekcija sa dodirnim meridijanima na 15˚ i 18˚ geografske dužine, uz koeficijent smanjenja mjerila 0.9999 (*HDKS*). Za visinski prikaz koristi se apsolutni sustav s obzirom na nivo-plohu mora. Iako bi se za područje

Republike Hrvatske od 2004. godine trebao primjenjivati položajni sustav *HTRS96*, još uvijek se koordinate iskazuju u Hrvatskom državnom koordinatnom sustavu (*HDKS*). Koordinate dobivene *GPS* opažanjem iskazuju se u *WGS84* položajnom elipsoidnom sustavu. Koraci transformacije iz *WGS84* u *HDKS* sastoje se od konverzije elipsoidnih koordinata (*φ, λ, h*) na *WGS84* elipsoidu u trodimenzionalne (*X, Y, Z*) kartezijeve koordinate po sljedećim formulama (Slika 20.). Na slici 20. *a* i *b* predstavljaju malu i veliku poluos *WGS84* elipsoida, a *N* je polumjer zakrivljenosti prvog vertikala.

$$X = (N + h)\cos\varphi\cos\lambda$$
$$Y = (N + h)\cos\varphi\sin\lambda$$
$$Z = (N(1-e^2) + h)\sin\varphi$$

a = 6378137 m
b = 6356752.3141 m

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \varphi}}, \qquad e^2 = \frac{a^2 - b^2}{a^2}$$

Slika 20. Formule za konverziju elipsoidnih u kartezijeve koordinate.

Potom slijedi transformacija *WGS84* kartezijevih koordinata u kartezijeve koordinate na Besselovom elipsoidu pomoću 7-parametarske Helmertove transformacije (Slika 21.). Ova transformacija uključuje translaciju (*dX*, *dY*, *dZ*) i rotaciju (*rx*, *ry*, *rz*) po sve tri osi, te promjenu mjerila *m*.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_2 = \begin{bmatrix} dX \\ dY \\ dZ \end{bmatrix}_{12} + (1+m)\begin{bmatrix} 1 & rz & -ry \\ -rz & 1 & rx \\ ry & -rx & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Slika 21. Formula za 7-parametarsku Helmertovu transformaciju.

Nakon transformacije slijedi konverzija pravokutnih Besselovih koordinata u elipsoidne koordinate na Besselu postupkom iteracije. Potrebno je poznavati elemente Besselovog elipsoida (veliku i malu poluos i linearni ekscentricitet).

a = 6377397.155 m
b = 6356078.96325

$$\varphi_0 = arctg \frac{Z}{D(1-e^2)} \qquad \lambda = arctg \frac{Y}{X}$$

$$N_0 = \frac{a}{\sqrt{1-e^2 \sin^2 \varphi_0}} \qquad h = \frac{D}{\cos \varphi_0} - N_0$$

$$\varphi_0 = arctg \frac{Z + e^2 N_0 \sin \varphi_0}{D} \qquad D = \sqrt{X^2 + Y^2}$$

Slika 22. Formule za konverziju pravokutnih u elipsoidne koordinate.

Uz pomoćni parameter (*D*) i poznate formule na slici 22., rade se iteracije, koje se ponavljaju dok razlika izlaznih rezultata ne bude zanemariva. Nadalje se elipsoidne koordinate putem odgovarajućih formula za kartografsku projekciju preslikaju u ravninu te se dobiju ravninske Gauss-Kruegerove koordinate (*y*, *x*). Podaci o ravninskim (*y*, *x*) koordinatama i Besselovoj elipsoidnoj visini (*h*) su vidljivi na slici 19.

4. Zaključak

Ovim radom je prikazan postupak izrade mobilne geo-aplikacije za *Android* platformu pod imenom GPS Kolektor. Obrađene su mogućnosti prikupljanja, spremanja i vizualizacije prostornih podataka. Prikupljanje prostornih podataka je moguće mrežnim i GPS pozicioniranjem. Vizualizacija prikupljenih prostornih podataka je moguća korištenjem Google Maps servisa. Spremanje prikupljenih prostornih podataka je moguće na dva načina. Podaci se mogu spremati na memorijsku karticu u obliku baze podataka i na Geoserver poslužitelj u GML obliku u realnom vremenu. Ovakva aplikacija je samo jedan od mnogih primjera geoinformacijskih alata i medija mobilnog *GIS*-a. Prednost takvog tipa aplikacije je servis u realnom vremenu. Prostorni podaci se mogu prikupljati *GPS* ili mrežnim pozicioniranjem i u kratkom roku se mogu dostaviti na zaslonu mobilnog uređaja. Iduća prednost je praktičnost. Aplikacija se može instalirati na sve uređaje koji imaju *Android* operativni sustav. Uređaji su mobilni i raspolažu sa veličinom ekrana, snagom procesuiranja podataka i trajanjem baterije koji odgovaraju korisničkim potrebama. Nedostatak ove aplikacije je kvaliteta dobivenih prostornih podataka. Najveća točnost se postiže do 5 metara *GPS* pozicioniranjem, uz povoljne vanjske uvjete i malo dužim stajanjem na točki. Naravno, može se postići veća kvaliteta pozicioniranja koristeći *NTRIP* protokol koji je obrađen u ovom radu. Jedini uvjet je da mobilni uređaj mora posjedovati modul za ugradnju eksternog *GPS* čipa koji bi mogao primati pozicijske korekcije. Bitno je napomenuti da je postojeću geo-aplikaciju moguće dalje unaprijeđivati i nadograđivati. Provedenim istraživanjem teme diplomskog rada došlo se do saznanja da su brojne mogućnosti razvijanja ovakvog tipa aplikacija. Stav je autora ovog teksta da daljnje razvijanje ovog relativno jednostavnog alata mobilnog *GIS*-a može naći mjesta kao isplativije rješenje na tržištu prikupljanja i manipulacije sa prostornim podacima.

Popis kratica i stranih riječi

Kratica	Puni izvorni naziv	Prijevod i značenje	Str
GIS	Geographic Information system	Geoinformacijski sustav	6
PDA	Personal Digital Assistant	Digitalni prijenosni uređaj	7
Smartphone		Pametni mobilni uređaj	8
OS	Operating System	Operativni sustav	8
CDD	Compatibility Definition Document	Skup specifikacija za Android podržane mobilne uređaje	9
Bug		U programiranju označava grešku programa	9
OpenGL ES 2.0		Skup specifikacija za 3D grafičko sučelje namijenjeno mobitelima i igraćim konzolama	9
JVM	Java Virtual Machine	Javin virtualni operativni sustav	10
SDK	Software Development Kit	Skup alata za razvoj softvera	13
apk	Android package	Programska ekstenzija Android aplikacije	13
IPC	Inter-Process Communication	Međuprocesna komunikacija	13
URI	Uniform Resource Identifier	Adresa resursa na Internetu ili računalnom disku	15
IDE	Integrated Development Environment	Softversko razvojno okruženje	19
Plug-in		Termin za dodatnu funkcionalnost u softveru	19
ADT	Android Development Tools	Skup alata za razvoj Android aplikacija	20

Kratica	Puni izvorni naziv	Prijevod i značenje	Str
API	Application Programming Interface	Programsko sučelje aplikacije	20
UI	User Interface	Korisničko sučelje	25
GSM	Global System for Mobile Communication	Standard za mobilnu telefoniju	27
SIM	Subscriber Identity Module	Čip koji služi za identifikaciju korisnika na mobilnim uređajima	28
BTS	Base Transceiver Station	Primopredajna bazna stanica GSM mreže	28
BSC	Base Station Controller	Upravljački dio bazne stanice GSM mreže	28
MSC	Mobile services Switching Center	Mobilni servisni komutacijski centar	28
SS7	Signaling System #7	Skup protokola za telefonsko signaliranje	28
ISDN	Integrated Services Digital Network	Oznaka za digitalnu telefonsku tehnologiju	29
Multipath		Efekt poništavanja više ulaznih signala, koji zbog različitih duljina putanja, u prijemnik ulaze sa suprotnim fazama	29
RTCM	Radio Technical Commission for Maritime Services	Protokol za razmjenu GPS podataka	33
GNSS	Global Navigation Satellite System	Globalni navigacijski satelitski sustavi	36
HTTP	HyperText Transfer Protocol	Protokol za prijenos informacija na Internetu	36

Kratica	Puni izvorni naziv	Prijevod i značenje	Str
TCP	Transmission Control Protocol	Transportni mrežni protokol, sastavni je dio IP grupe protokola	36
IP	Internet Protocol	Mrežni protokol za prijenos podataka	36
ASCII	American Standard Code for Information Interchange	Američki standardni znakovnik za razmjenu obavijesti	41
UTC	Universal Coordinated Time	Koordinirano svjetsko vrijeme, primarni standard po kojem se ravnaju satovi i vrijeme	43
HDOP	Horizontal Dilution Of Precision	Relativna točnost horizontalne pozicije	43
DBMS	Database Management System	Sustav za upravljanje bazom podataka	49
ACID	Atomic, Consistent, Isolated, Durable	Pojam označava uvjete transakcija kod baza podataka: nedjeljivost, konzistentnost, izolaciju i trajnost	49
HDKS		Hrvatski Državni Koordinatni sustav	60
HTRS96/TM		Hrvatski terestrički referentni sustav za epohu 1995.55	61

Literatura

RTCM Special Committee No. 104 (2004.): Networked Transport of RTCM via Internet Protocol (Ntrip), SAD.

RTCM Special Committee No. 104 (2006.): RTCM Standard for Diferential

GNSS (Global Navigation Satellite Systems) Services-Version 3, SAD.

Medak, D. (2008.): Predavanja iz kolegija Baze podataka, Zagreb.

OpenGIS Implementation Specification (2005.): Web Feature Service Implementation Specification, v 1.1.0, reference nr. 04-094, SAD.

Frančula, N. (2004.): Kartografske projekcije, skripta za predavanja.

Popis URL-ova

URL1: Mobile geoinformation services, http://www0.hku.hk/dupad/asiagis/fall03/Full_Paper/Zhang_Jinting.pdf

URL2: Application software, http://en.wikipedia.org/wiki/Application_software (8.08.2011.)

URL3: Programski jezik, http://hr.wikipedia.org/wiki/Programski_jezik (8.08.2011.)

URL4: Mobile operating system, http://en.wikipedia.org/wiki/Mobile_operating_system (8.08.2011.)

URL5: Android, http://en.wikipedia.org/wiki/Android_(operating_system) (8.08.2011.)

URL6: Android Architecture, http://developer.android.com/guide/basics/what-is-android.html (8.08.2011.)

URL7: Java programming language, http://en.wikipedia.org/wiki/Java_(programming_language) (8.08.2011.)

URL8: Uvod u JAVA programiranje, http://laris.fesb.hr/java/index.htm (8.08.2011.)

URL9: Application components, http://developer.android.com/guide/topics/fundamentals.html (8.08.2011.)

URL10: Android manifest, http://developer.android.com/guide/topics/manifest/manifest-intro.html (8.08.2011.)

URL11: Eclipse IDE, http://hr.wikipedia.org/wiki/Eclipse_(software)

(8.08.2011.)
URL12: GSM Network, http://www.fig.net/pub/costarica_1/papers/ts08/ts08_01_schwieger_2407.pdf (8.08.2011.)
URL13: Mobile tracking, http://en.wikipedia.org/wiki/Mobile_phone_tracking (8.08.2011.)
URL14: RTCM, http://www.geodetic.gov.hk/smo/gsi/data/ppt/NMEAandRTCM.ppt (8.08.2011.)
URL15:NTRIP, http://www.fig.net/pub/athens/papers/ts03/ts03_2_lenz.pdf (8.08.2011.)
URL16: CROPOS, http://www.cropos.hr (8.08.2011.)
URL17: NMEA, http://www.gpsinformation.org/dale/nmea.htm (8.08.2011.)
URL18: Geoserver, http://en.wikipedia.org/wiki/Geoserver (8.08.2011.)
URL19: OpenLayers, http://en.wikipedia.org/wiki/OpenLayers (8.08.2011.)
URL20: SQLite, http://en.wikipedia.org/wiki/SQLite (8.08.2011.)
URL21: HDKS formule, http://www.geof.unizg.hr/~adapo/transformule.htm (8.08.2011.)

Popis slika

Popis tablica

Životopis C U R R I C U L U M V I T A E

OSOBNI PODACI:

Ime i prezime:	ANA MARJANICA
Adresa:	Kranjčevićeva 21, Split
Telefon:	021/ 572-635, 01/386-0612
Mobitel:	091/798-2355
E-mail:	amarjanica@geof.hr
Bračno stanje:	Neudana
Datum rođenja:	16. travnja, 1986.

OBRAZOVANJE I EDUKACIJA:

Školovanje:	Osnovno obrazovanje od 1. do 8. završila u OŠ “Dobri” u Splitu (1993.–2001.).
	Srednjoškolsko obrazovanje završila u Prirodoslovno-matematičkoj gimnaziji u Splitu (2001.–2005).
	Fakultetsko obrazovanje: Preddiplomski studij na Geodetskom fakultetu u Zagrebu (2005. –2009.), Diplomski studij na Geodetskom fakultetu u Zagrebu (2009.–2011.)

RADNO ISKUSTVO:	Promesa Usluge d.o.o., Zagreb posao: punjenje polica po trgovačkim centrima PAN-PEK d.o.o., Zagreb posao: prodavačica u pekari

OSTALE OSOBNE VJEŠTINE I PODACI

Strani jezici:	Engleski jezik, izvrsno poznavanje jezika u govoru i pismu.
Ostale vještine i sposobnosti:	Također znam raditi i na računalu s operativnim sustavom Windows te se odlično služim s programskim paketom Microsoft Office, grafičkim programom AutoCAD, GeoMedia Professional programom za rad s prostornim podacima, programskim jezikom JAVA, PostgreSQL-om te operativnim sustavom Android.