RAZVOJ APLIKACIJA ZA ANDROID PLATFORMU

DEVELOPMENT OF ANDROID MOBILE APPLICATIONS

Ognjen Ribičić, mag.inf. Boris Tomaš, mag.inf. Zlatko Stapić

SAŽETAK:

Današnji trendovi u razvoju mobilnih aplikacija odraz su velike dinamike na tržištu mobilnih uređaja. Ovaj rad se temelji na predstavljanju osnovnih koncepata i načina razmišljanja kod razvoja aplikacija za Android mobilnu platformu. Nastoji se ukazati na jednostavnost korištenja platforme kao i na standardni stil i pristup pisanja kvalitetnog programskog kôda. U radu su prikazane dobre prakse korištenja osnovnih i naprednijih koncepata u razvoju, uključujući i rad sa razvojnim okruženjem kao i specifičnosti Java jezika. U konačnici, temeljem iskustva autora, rad prikazuje i najčešće probleme s kojima se susreću razvojni inženjeri te načine njihovog rješavanja.

ABSTRACT:

Current trends in mobile applications software development are results of significant dynamics in mobile device market. This paper aims to introduce the basic concepts and ways of thinking regarding application development for the Android mobile platform. This work tries to point out a simple use of the platform as well as a standard style and an approach in defining a good programming code. Good practices are shown regarding the use of basic and advanced development concepts in Java language. Paper also states a few guidelines regarding the usage of integrated development environment. At the end, based on the authors experiences, few and common issues and problems for developers are identified and their solution is presented.

1. UVOD

Android je od prve inačice iz 2008. godine [1] do danas daleko napredovao kao skup softverskih alata koji uključuju operacijski sustav, sustav zajedničkih funkcija (*engl. middleware*) i aplikacija. Mobilni uređaji postali su brži, SDK sofisticiraniji, a korisnici zahtjevniji. Indikator prije navedenog je nagli porast popularnosti platforme od strane korisnika, kao i programera od njezine publikacije. Sama platforma je otvorenog kôda (*engl. open source*) što omogućava razvoj bogatih i naprednih aplikacija u relativno kratkom vremenu uz nerijetko besplatnu podršku zajednice. Ta razvojna zajednica je također opsežna, te omogućava putem grupa, poput „Google Groups“, komunikaciju s istom. Kao i svaka platforma, Android sadrži neke specifičnosti.

Ovaj rad se temelji na predstavljanju osnovnih koncepata i načina razmišljanja kod razvoja aplikacija za Android mobilnu platformu. Nastoji se ukazati na jednostavnost korištenja platforme kao i na standardni stil i pristup pisanja programskog kôda. U radu su prikazane dobre prakse korištenja osnovnih i naprednijih koncepata u razvoju, uključujući i rad sa razvojnim okruženjem kao i specifičnosti Java jezika. U konačnici, temeljem iskustva autora, rad prikazuje i najčešće probleme s kojima se susreću razvojni inženjeri te načine njihovog rješavanja.

Većina prikazanih koncepata u radu je prikazana i odgovarajućim primjerima koji prikazuju osnovu primjene koncepta o kojem se diskutira. Većina predložaka za izradu primjera je preuzeta iz projekta *Remoter* koji su autori razvili prije pisanja ovog rada i koji je trenutno u fazi objave na Android tržište.

Ovaj rad bi trebali čitati programeri koji su upoznati i imaju iskustva sa Java programskim jezikom. Svakako prije navedeno treba shvatiti kao prijedlog, te zainteresirane ne bi trebalo demoralizirati i odvratiti od čitanja istog.

2. STRUKTURA APLIKACIJA

Kako je prije navedeno aplikacije se pišu u programskom jeziku Java. Jezik kao takav ima neke svoje specifičnosti koje svakako prije samog početka proučavanja izrade Android aplikacija valja pogledati, pa čak i proučiti. Razvojna okolina se sastoji od seta razvojnih alata (*engl. SDK tools*), koji služe za prevođenje (engl. compile) izvornog kôda Android aplikacije, pripreme aplikacija (*engl. deployment*) kako bi se one mogle izvršavati na mobilnom uređaju.

Sve podatke o aplikaciji, uključujući izvršni kôd, resurse potrebne za izvršavanje aplikacije, metapodatke i manifest podatke o aplikaciji, razvojno okruženje kod finaliziranja aplikacije pohranjuje u obliku jedne arhive sa ekstenzijom *.apk* [3].

Prilikom instalacije aplikacije na mobilnom uređaju, ista dobiva svog vlastitog korisnika odnosno jedinstveno korisničko okruženje na Android platformi; također aplikaciji se dodjeljuje vlastito virtualno okruženje (*engl. virtual machine*) te vlastiti „Linux ID“ [3]. Pri pokretanju, svakoj aplikaciji se dodjeljuje vlastiti proces unutar kojeg se aplikacija izvršava, ovaj mehanizam štiti Android platformu od neispravnih aplikacija tako da aplikacija može utjecati samo na svoj proces dok su ostali procesi izolirani od možebitnog štetnog procesa. Na ovaj način Andorid implementira princip dodjele minimalnih privilegija (*engl. “principle of least privilege“*) čime osigurava visoku sigurnost okruženja u kojima aplikacije ne mogu pristupati dijelovima sustava za koji nemaju dozvolu [3].

2.1 Aplikacijske komponente

Aplikacijske komponente su na Android platformi najvažniji elementi od kojih se sastoje aplikacije. Svaka komponenta ima svoj životni ciklus, te omogućava sustavu rad s aplikacijom na sebi svojstven način. Postoje četiri osnovne komponente [3]:

- Aktivnost (*eng. Activity*) – Najčešće korištena komponenta u aplikacijama, reprezentira ekran odnosno sučelje za korisničku interakciju.
- Servis (*eng. Service*) – Operacija koja se izvršava u pozadini, te se obično koristi za neke vremenski odnosno procesorski zahtjevne proračune. Service je u stanju pokrenuti vlastiti proces te ostvariti interakciju s istim.
- Pružatelj sadržaha (*eng.Content provider*) – Dio API-a koji omogućava rad sa zajedničkim podacima kao što su interna i eksterna memorija ili SQLite baza.
- Primatelj vanjskih događaja (*eng. Broadcast receiver*) – Služi za primanje poruka iz sustava, na primjer gašenje ekrana, stišavanje glasnoće uređaja i ostali događaji.

2.2 Predefinirane varijable, resursi

Android za razliku od ostalih sustava ima vrlo dobro riješen problem internih resursa. Na primjer, svi elementi forme se dohvaćaju putem jedinstvenih identifikatora (*engl. ID*) ili oznaka (*engl. tag*). Ukoliko koristimo dohvaćanje putem identifikatora, postupak pretraživanja elemenata je veoma brz sobzirom da istog reprezentira cjelobrojna (*engl. integer*) varijabla. Također spremanje preddefiniranih konstanti je omogućeno na sličan način kao i prije navedeni elementi forme.

2.3 Asinkrone poruke

Tri od četiri aplikacijske komponente aktiviraju se asinkronim porukama, *intentima*. To su „Activity“, „Service“ i „Broadcast reciver“. *Intent* omogućava pozivanje i povezivanje istih na izvršno okruženje (*engl. runtime environment*). Dakle, ako želimo pozvati novu formu, sve što trebamo napraviti jest stvoriti novi *intent* objekt, i kroz konstruktor objekta proslijediti klasu koja će se pokrenuti. Rezultat akcije je, na primjer pri kreiranju nove forme, novi zaslon odnosno forma koju smo kreirali i pozvali.

```
Intent myIntent =
   new Intent(getBaseContext(),
frmMain.class);
startActivity(myIntent);
```

Kôd 1 Primjer pozivanja/stvaranja forme (klase)

Stoga, pozivanje aplikacijskih komponenata i ako je to potrebno slanje podataka istima je prvi bitan koncept na koji ćemo se osvrnuti u ovom radu. Ukoliko želimo poslati određene podatke novoj formi, dodajemo paket ili svežanj (*engl. bundle*) na *intent*. Spomenuti svežanj sadrži skup podataka koje želimo poslati. Također kada se stvori nova forma iste podatke dohvaćamo putem “getera” iz *intenta* te ih po potrebi koristimo na novoj formi. *Intenti* zajedno sa aplikacijskim komponentama, Manifest datotekom i formama čine osnove programiranja u Android okruženju.

2.4 Forme i raspored elemenata

Za razliku od C-srodnih jezika kao što su C# ili Objective-C koji se koriste u Windows Phone odnosno na iPhone platformi, i koji omogućavaju “*crtanje*“ formi, Android platforma preferira kôdiranje dizajna formi. Najčešće se radi sa XML datotekama koje sadrže vizualni kôd forme, pomalo nalik na HTML, no moguće je iste i napraviti programski, što dakako oduzima dosta vremena i nije preporučljivo. Sukladno tome postoje komponente pregleda kontrola (*engl. ViewGroup components*), vrste razmještaja elemenata unutar zaslona kao što su LinearLayout, RelativeLyout, i tako dalje. Isto tako, postoje pogledi (*engl. view*) odnosno elementi koje prikazujemo. Koncept rada XML datoteka sa dizajnom temelji se na hijerarhiji. „ViewGroup“ unutar sebe može sadržavati elemente forme: „View“ ali i druge načine prikaza – „ViewGroup“ [9].

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android=

"http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
<Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Klik" />
</LinearLayout>
```

Kôd 2 Jednostavna forma (XML)

Kôd prikazan u isječku (Kôd 2) će rezultirati jednostavnom formom koja sadrži jednu tipku (engl. Button).

Klasa koja procesira formu je prikazana u isječku (*Kôd 3*). Kôd te klase je odgovoran za korisnički klik na tipku definiranu prije navedenim XML kôdom. Ukoliko se klikne na tipku, ispiše se preko brze obavijesti (*engl. toast notification*) tekst “Ovo je test”.

```
public class HelloAndroid extends Activity
{
   /* Called when the activity is first
created. */
   @Override
   public void onCreate(Bundle
savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

      //Trazenje gumba unutar forme
      final Button btn =
(Button)findViewById(R.id.btn)

      //Postavljanje slusatelja
      button.setOnClickListener(
           new OnClickListener()
        { public void onClick(View v)
    { Toast.makeText(
HelloAndroid.this,  "Ovo je test",
Toast.LENGTH_SHORT).show();}});
   }
}
```

Kôd 3 Jednostavna klasa (Java)

2.5 Manifest datoteka

Manifest je XML datoteka koja je sastavni dio *.apk* arhive i ima specifičnu namjenu. Datoteka u potpunosti opisuje aplikaciju. Sve aktivnosti, a i ostale komponente, su opisane u datoteci na način da sustav razumije namjenu, korisničke dozvole i slično. Moguće je i ograničiti aplikaciju na verziju platforme obzirom da novije verzije donose sa sobom i različite funkcionalnosti [5].

Primjer jedne takve datoteke prikazuje isječak programskog koda (*Kôd 4*)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk
/res/android"
      package="app.test.example"
      android:versionCode="1"
      android:versionName="1.0">
 <uses-sdk android:minSdkVersion="5" />
 <application
android:label="@string/app_name">
      <activity android:name=".Test"

android:label="@string/app_name">
          <intent-filter>
              <action android:name=
          "android.intent.action.MAIN" />
                <category android:name=
         "android.intent. category.LAUNCHER"
/>
          </intent-filter>
      </activity>
  </application>
</manifest>
```

Kôd 4 Primjer AndriodManifest datoteke (XML)

3. OSNOVNI KONCEPTI

U ovom poglavlju pokušat ćemo približiti neke specifičnosti programiranja u Android okruženju. Bavit ćemo se najčešćim konceptima koji se pojavljuju prilikom izrade aplikacija kao što su osluškivači događaja (*engl. event listeners*), izbornici, obavijesti i ostali.

3.1 Osluškivači događaja

Osluškivač događaja je koncept Android platforme koji nam omogućava reagiranje na događaje kada se dogode, te po potrebi i obrađivanje istih. Koncept se temelji na povezivanju osluškivača na događaj nekog od prethodno navedenih elemenata te time i na izmijeni izvršavanja njegove funkcionalnosti. Također potrebno je spomenuti kako je moguće koristiti Java koncept koji omogućava premošćivanje standardnih metoda izvršavanja funkcionalnosti i zamjensko korištenje vlastitih metoda. Koncept premošćivanja se kao i u nekim drugim jezicima implementira korištenjem predefinirane naredbe „`@Override`”, a zanimljivo je spomenuti da se u Android razvojnom okruženju može premostiti sve metode definirane na razini sustava.

Sličan je pristup implementaciji funkcionalnosti pri reakciji na ostale „napredne“ događaje kao što su na primjer događaj dodira (*engl. touch event*) ili događaj geste (*engl. gesture event*), no o tome u sljedećim poglavljima.

```
final Button btn = (Button)
findViewById(R.id.btn);
btn.setOnClickListener(
           new OnClickListener()
       { public void onClick(View v)
    { Toast.makeText(
HelloAndroid.this,
"Ovo je test", Toast.LENGTH_SHORT).show();
```

Kôd 5 Primjer osluškivača događaja

3.2 Izbornici

Specifičnost Android okruženja je neposredna interakcija programera i razvojnog okruženja pri kreiranju izgleda formi, izbornika, animacija i ostalih pretežno XML baziranih rješenja. Kao i forme, izbornike (*engl. menu*) je moguće napraviti programski, ali ipak, kako je praksa pri izradi aplikacija korištenje XML datoteka, taj pristup ćemo koristiti i kod izrade izbornika. Izbornik je u pojedinim elementima sličan klasičnoj formi korisničkog sučelja, no ujedno ima i znatna pojednostavljenja ovog koncepta. Također hijerarhija elemenata kod izbornika slična je hijerarhiji elemenata formi.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.
            com/apk/res/android">
        <item android:id="@+id/settings"

android:title="@string/settings"/>
</menu>
```

Kôd 6 Definicija izbornika (XML) – Menu.xml datoteka

3.3 Obavijesti

U Android okruženju postoje tri različite vrste elemenata za prikaz obavijesti korisniku:

- Brza obavijest
- Obavijest u statusnoj traci
- Obavijest na dijaloškom okviru (*engl. dialog notification*)

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
        MenuInflater inflater =
getMenuInflater();
        inflater.inflate(R.menu.menu, menu);
        return true;
}
    @Override
public boolean onOptionsItemSelected(MenuItem
item){
        switch (item.getItemId())
{
        case R.id.settings:
           return true;
default:
   return super.onOptionsItemSelected(item);
}
}
```

Kôd 7 Završavanje definicije izbornika (Java)

Brza obavijest (*engl. toast notification*) je privremeni objekt koji služi za trenutno obavještavanje korisnika. Zauzima prostor koji mu je potreban da ispiše poruku, dok aplikacija i dalje ostaje vidljiva i interaktivna [9].

Obavijest u statusnoj traci (*engl. status bar notification*) označava trajniji tip notifikacije korisnika. Prikazuje se, kako ime govori, u statusnoj traci, te pri selekciji okida intent definiran u obavijesti [9].

```
final CharSequence[] items =
{getString(R.string.add),

getString(R.string.connect)};
final AlertDialog.Builder options =

new AlertDialog.Builder(this);
options.setTitle(getString(R.string.quick_act
ions));
options.setCancelable(true);

options.setItems(items, new
DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface
dialog, int item)
    {   //kod koji se izvršava prilikom
poziva }
});
options.show(); //aktiviranje dijaloga
```

Kôd 8 Definiranje dijaloga

Obavijest na dijaloškom okviru (*engl. dialog notification*) je definirana kao novi mali dijaloški okvir koji se prikazuje na vrh trenutnog dijaloškog okvira, te može pružati interakciju korisniku. Moguće ga je koristiti i za unos malih količina podataka [9]. Dijalog se stvara i prikazuje na način kako je to navedeno ovim programskim isječkom:

4. NAPREDNI KONCEPTI

Tijekom korištenja naprednih koncepata u razvoju aplikacija za Android uređaje, često se služimo svim navedenim osnovnim konceptima kako bi se zadovoljila funkcionalnost određene komponente i postigla željena nova funkcionalnost. Budući da SDK nudi mnoštvo različitih koncepata, ovaj rad opisuje prema našem izboru najčešće i najkorištenije koncepte.

4.1 Prilagodnici

Jedan od tih koncepata su prilagodnici (*engl. adapters*). Adapteri su koncepti koji omogućavaju izmjenu, to jest prilagodbu određene skupine podataka na neki od predefiniranih načina. Primjer takve prilagodbe je „SimpleAdapter“.

Ako primjerice koristimo listu za izbor jedne od ponuđenih vrijednosti, a stavke te liste su izvedene iz skupa jednostavnijih atributa, iste možemo putem prije navedenog adaptera prilagoditi, te umetnuti u listu kao njene vrijednosti. Postoji mnoštvo različitih adaptera, čime se još jednom približavamo shvaćanju opsežnosti Android razvojnog okruženja kao i njegovih mogućnosti.

```
ArrayList<Map<String, String>> server_lst
=list_values;
String[] from = {"name", "version"};
int[] to = {R.id.name, R.id.version};
SimpleAdapter adapter = new
SimpleAdapter(getApplicationContext(), list,

R.layout.list, from, to);
list.setAdapter(adapter);
```

Kôd 9 Korištenje adaptera

4.2 Dijeljena svojstva

Kako iz imena zaključujemo radi se o zajedničkim postavkama aplikacije (*engl. shared preference*). Moguća je promjena primitivnih tipova podataka u obliku *ključ => vrijednost.* Također „Activity“ ima indirektnu podklasu „*PreferenceActivity*“ koja nam omogućava pregled postavki, sa preddefiniranom funkcionalnošću, spremljenom unutar određene XML datoteke ili unutar klase koja koristi „*PreferenceActivity*“. Spomenuta klasa automatski omogućava izmjene nad prije navedenim primitivnim podacima [5].

```
final EditText name = (EditText)
findViewById(
R.id.text_name);
SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences
(
    this);
name.setText(prefs.getString("name",
"vrijednost"));
```

Kôd 10 Korištenje koncepta dijeljenih svojstava

4.3 Unutarnja memorija

Unutarnja memorija omogućava spremanje podataka aplikacija u internu memoriju uređaja. Uz razne dodatne API-je omogućeno nam je manipuliranje nad raznim tipovima datoteka. Valja napomenuti kako se ova vrsta pohrane podataka koristi za spremanje privatnih podataka koji ne bi trebali biti dostupni ostalim aplikacijama ili korisniku [5]. U sljedećem programskom isječku se može vidjeti način pohrane podataka u datoteku u unutarnjoj memoriji Android uređaja.

```
String FILENAME = "file";
String string = "test";
FileOutputStream fos = openFileOutput(FILENAME,
                        Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Kôd 11 Korištenje unutarnje memorije

4.4 Vanjska memorija

Gotovo uvijek pri ovom pojmu mislimo na SD memorijske kartice. Temeljna razlika između unutarnje i vanjske memorije je u dostupnosti podataka. Naime Andorid, kako je prije navedeno, definira više razina sigurnosti podataka, a podaci zapisani na ovaj način postaju dostupni svim aplikacijama pa čak i korisniku što postaje suština postojanja ovakve funkcionalnosti [5].

```
File path = getExternalStorageDirectory();
if (path != null) {
    File file = new File(path, "file");
    OutputStream os = new
FileOutputStream(file);
    os.write(new byte["test"]);}
```

Kôd 12 Korištenje vanjske memorije

4.5 Mobilna baza podatka

Jedan od dakako standardnih, ali i primamljivih koncepata mobilnih uređaja je funkcionalnost korištenja mobilne baze podataka. Android okruženje rabi sqlite3 verziju baza podataka, te postoji mnoštvo primjera kako spomenutu bazu koristiti. Potrebno je naglasiti kako je pisanje pomoćne (engl. helper) klase, koja će umjesto

nas odrađivati obradu upita, što na većim projektima može skratiti vrijeme izrade [5].

```
public class DBHelper extends
SQLiteOpenHelper
{
    private static final int DATABASE_VERSION
= 2;
    private static final String DATABASE_NAME
=
      "sqlite_db";
    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null,
        DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("some SQL query");
    }
}
```

Kôd 13 Korištenje SQLite baze podataka

4.6 HTTP

Kao pametni telefon treće i četvrte generacije - generacije mobilnog Interneta, Android pruža opsežni API za komunikaciju sa Internetom. Pokrivena je cjelokupna funkcionalnost HTTP protokola, što uključuje i mogućnosti korištenja raznih web servisa na razini GET i POST metoda.

```
URL connectURL = new URL("www.google.com");
HttpURLConnection http =

(HttpURLConnection)connectURL.openConnection(
);

http.setDoInput(true);
http.setDoOutput(true);
http.setUseCaches(false);
http.setRequestMethod("GET");

http.connect();//spajanje na poslužitelj
http.getOutputStream().flush();

String response = getResponse(http);
```

Kôd 14 Primjer jednostavnog dohvaćanja sadržaja web stranice

Isječak Kôd 14 dohvaća sadržaj web stranice www.google.com koristeći standardni HTTP protokol sa GET metodom.

4.7 XML

Kako se do sada moglo zaključiti, platforma je orijentirana ka korištenju XML standarda, što nas može dovesti do zaključka kako ista sadrži i «jaki» XML API. Android pruža podršku za DOM, SAX i Pull parser koncepte. Od verzije 2.2 Android API je proširen s xPath1.0 funkcionalnošću za navigaciju XML dokumentom.

4.8 Gesture

Dodavanjem „Multi-Touch“ funkcionalnosti, Android dobiva podršku za geste. Geste su interakcije sa uređajem prstima u vidu dodira (*engl. tap*), dvostrukog dodira (*engl. double tap*), približavanja prstima (*engl. pinch zoom*), pomaka (*engl. swipe*), i tako dalje. Kreiranjem osluškivača, predajemo događaj *gesture* klasi putem koje iste obrađujemo. Specifičnost ove podrške je u praktički neograničenosti primjene ovih gesti. Dakako korištenje istih je dosta olakšano putem Gesture API-ja. Geste trebaju svoju vlastitu osluškivačku klasu kao na primjer `TrackpadGestureListner` klasa koja implementira dva sučelja (*engl. interface*):

- OnGestureListener
- OnDoubleTapListener

```
DocumentBuilder builder =
       factory.newDocumentBuilder();
FileInputStream fin =
openFileInput(file_name);
Document dom = builder.parse(fin);

NodeList servers_from_xml =

dom.getElementsByTagName("some_elem");

for (i=0; i < servers_from_xml.getLength();
i++)
{
   Node node = servers_from_xml.item(i);
   String name =

node.getAttributes().item(0).getNodeValue().
      toString();
}
```

Kôd 15 Čitanje podataka iz XML datoteke

Sama klasa sadrži sljedeće događaje:

- onDown
- onFling
- onLongPress
- onScroll
- onShowPress
- onSingleTapUp
- onDoubleTap
- onDoubleTapEvent
- onSingleTapConfirmed

4.9 Grafika

Kompletno Android sučelje i komponente su u suštini 2D grafike - svi elementi su 2D elementi. Transformacije ovih elemenata svode se na manipuliranje širinom i visinom. Također moguće je animirati elemente, no o tome u sljedećim poglavljima [6].

Uz 2D podržane su 3D animacije i to na dva načina: *OpenGL i Renderscript* animacije. *OpenGL* je opće poznat koncept 3D animacija te je dovoljno reći kako razvojno okruženje podržava šest standardiziranih verzija istoga. Dakle radi se o klasičnoj manipulaciji kroz API sučelje. Za razliku od OpenGL-a, *Renderscript* koristi najbolji resurs za obradu grafike u konkretnom trenutku, što čak može biti i GPU (*engl. Graphic Processing Unit*) ukoliko ga uređaj ima.

Također, *Renderscript* se razlikuje po još nekim karakteristikama od standardnog programiranja u Androidu. Naime, isti je namijenjen korisnicima kojima nije problem programirati u C-u, točnije C99 standardu što za sobom može povesti i probleme. Naime, budući je moguće renderiranje na GPU-u, ispravljanje pogrešaka (*engl. debugging*) može postati problematičan. Predlaže se korištenje ove metode obrade animacije ukoliko je naglasak na performansama to jest gdje *OpenGL* ne rezultira dovoljno kvalitetnim i brzim programskim kodom [6].

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android=

"http://schemas.android.com/apk/res/android">
       <translate android:fromXDelta="100%p"
   android:toXDelta="0%p"
   android:duration="500" />
       <alpha     android:fromAlpha="0.0"
   android:toAlpha="1.0"
   android:duration="500" />
</set>
```

Kôd 16 Definiranje animacije (XML)

Android također podržava 2D animacije koje se mogu provoditi nad elementima forme. Sastoje se od deklarativne XML datoteke ili programskog kôda, te programskog kôda za obradu odnosno izvršavanje 2D animacija [6].

Isječak Kôd 16 prikazuje jednostavnu animaciju te ukazuje na opsežnost i snagu Android platforme prilikom rada sa grafikom odnosno 2D animacijama [6].

```
final Animation a =
AnimationUtils.loadAnimation(Trackpad.this,
R.anim.anim_xml);
a.reset();
some_element.startAnimation(a);
```

Kôd 17 Definiranje animacije (JAVA)

4.10 Rad s mrežnom infrastrukturom

Koncepti korištenja bežične mreže (WiFi, Bluetootha, GSM) su također kao i velika većina koncepata relativno jednostavni za korištenje. Dovoljno je zatražiti od sustava dozvolu za korištenje, te manipulirati objektima. Kod ovih koncepata može doći do problema obzirom na različite verzije SDK-a. Naime, starije verzije ne podržavaju neke funkcionalnosti, te je potrebno istima ograničiti pristup putem Manifest datoteke ili implementirati kontrolu verzije unutar same aplikacije.

GPS za razliku od bežičnih mreža nije doživio veće promjene, te korištenje istoga također ne predstavlja velike probleme. Rad s GPS-om se svodi na traženje dozvole i korištenje preddefiniranih objekata. Slijedeći isječci kôda prikazuju korištenje pojedinih infrastrukturnih koncepata na Android mobilnim uređajima:

```
WifiManager wm =
(WifiManager)getSystemService(Context.WIFI_SE
RVICE);
```

Kôd 18 Korištenje WiFi infrastrukture

```
BluetoothAdapter bm =
BluetoothAdapter.getDefaultAdapter();
```

Kôd 19 Korištenje Bluetooth infrastrukture

```
TelephonyManager tm =
(TelephonyManager)getSystemService(
          Context.TELEPHONY_SERVICE);
```

Kôd 20 Korištenje GSM infrastrukture za pozive

```
LocationManager lm =
(LocationManager)getSystemService(
           Context.LOCATION_SERVICE);
```

Kôd 21 Korištenje GPS infrastrukture

Ovisno o traženoj funkcionalnost potrebno je dozvoliti korištenje odgovarajuće infrastrukture unutar manifest datoteke. Korišteni objekti u primjerima postoje ukoliko postoji tražena infrastruktura na mobilnom uređaju, u suprotnom vrijednost objekta je `null`.

4.11 Google APIs

Podrška za rad s Google vanjskim API-jima unutar vlastitog mobilnog operacijskog sustava je i za očekivati. Tako je na primjer korištenje Google Maps servisa moguće ugraditi u aplikaciju, odabirom odgovarajućeg API-a prilikom stvaranja novog projekta.

Koncept Google Maps API-ja je drukčije prirode zbog toga što se koristi na relativno jednostavan način, ali dodatne kontrole na Google kartu dodatno komplicira implementaciju [4].

```
< com.google.android.maps.MapView
android:id="@+id/mapView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:enabled="true"
android:clickable="true"
android:apiKey="some api key"/>
```

Kôd 22 Dodavanje Google Maps komponente (XML)

Unutar XML kôda forme stvaramo element tipa „com.google.android.maps.MapView“ kako to prikazuje isječak u kodu (22). Također, unutar manifest datoteke je potrebno dodati i odgovarajući isječak definiran u kodu koji slijedi (23).

```
<uses-library

android:name="com.google.android.maps"/>
<uses-permission

android:name="android.permission.INTERNET"/>
```

Kôd 23 Proširenje manifest datoteke

4.12 Rad s dretvama

Android podržava višedretvenost (*engl. multi-threading*), što nam uvelike olakšava izradu i funkcioniranje složenijih aplikacija. Dretve se mogu koristiti na dva načina: tako da se izvršavaju na „UI Thread-u“ što naravno blokira isti, ili kao pozadinske dretve, čiji je posao obično odraditi neki vremenski opsežniji posao, tako da se „UI Thread“ ne blokira, što znači da korisnik može nastaviti rad s aplikacijom dok čeka na rezultat izvršavanja. Komunikaciju između dretvi moguće je realizirati na više načina. Jedan od poznatijih i primjenjenijih je korištenje „Handlera“, kako bi se slale poruke među dretvama i dogovorio tijek izvršavanja. Još jedan važan koncept preuzet iz Jave je korištenje tako zvanih sinkroniziranih (*engl. syncronized*) blokova za sinkronizaciju između dretvi [10].

4.13 Dinamična izmjena formi

Kod Android platforme moguće je izmjenjivati prikaz elemenata (View-ove) prilikom izvršavanja aplikacije ukoliko prikaz ovisi o prijašnjoj akciji. Primjer ovog koncepta je mogućnost izmjene sadržaja u sklopu dijaloga, kako je prije navedeno za unos malih količina podataka. Isti postupak vrijedi također i za izbornik, svojstva, animacije i ostale koncepte.

4.14 Ostali koncepti

Android SDK se pobrinuo i za ostale koncepte kao što su vibracije, zvuk, slika, pokretna slika i tako dalje, te je u platformu ugrađena opsežna funkcionalnost za manipulaciju nad istima kroz primjenu odgovarajućih klasa. Koncepti se relativno jednostavno koriste i u skladu su ostatkom Java odnosno Android filozofije [7].

5. PROBLEMI

5.1 Instalacija okruženja

Iz ranije navedenoga, programira se u programskom jeziku Java. Kako bismo isto mogli provoditi važno je stvoriti adekvatno razvojno okruženje. Ovisno o operativnom sustavu na kojem razvijamo aplikaciju, preuzimamo SDK sa službenog web mjesta: "http://developer.android.com/sdk/index.html". SDK instalacija je automatizirani proces. Kako se na stranicama preporučuje korištenje Eclipse integriranog razvojnog okruženja (*eng. IDE*), istoga također preuzimamo sa Interneta. Nadalje, Java JRE (*engl. Java Runtime Environment*) koji dolazi sa OS-om nije namijenjen razvoju, te je potrebno preuzeti Java JDK (*engl. Java Development Kit*).

Pokretanje razvojnog okruženja vremenski ne traje dugo. Prije opisani proces zahtjeva, naravno ovisno o brzini Interneta kojim raspolažete, oko 30-tak minuta. Ovisno o izborima prilikom instalacijske procedure, ovo razvojno okruženje može zauzeti dosta prostora na tvrdom disku. To je shvatljivo obzirom da SDK može sadržavati više virtualnih slika mobilnih OS-a, koji razumljivo i zauzimaju dosta prostora.

5.2 Emulatori i fizički uređaji

U Eclipse-u je potrebno umetnuti putanju do novo kreirane SDK mape, te stvoriti emulator koji će nam omogućiti razvoj aplikacije bez korištenja stvarnog uređaja. Ukoliko imamo fizički uređaj emulator nije potreban, te je moguće izvršavati aplikacije i ispravljati pogreške na samome uređaju, što za sobom dovodi i prve probleme prilikom izrade aplikacija: emulirana verzija Androida ne može podržavati funkcionalnosti kao što su WiFi, BT, GPS, IR i tako dalje, iz razloga što je za te funkcionalnosti potreban fizički uređaj. Ukoliko se izrađuju aplikacije koje koriste baš ove koncepte, nužno je nabaviti fizički uređaj. Fizičke mobilne uređaje moguće je putem USB kabla spojiti na računalo. Kako bismo to mogli napraviti moramo biti upoznati s određenim konceptima. Naime USB veza radi besprijekorno samo na OSx platformi. Kod Windows-a potrebno je instalirati posebne USB pokretače (*engl. drivers*) kako bi se mobilni uređaj mogao koristiti za razvoj aplikacija. Linux OS ne razumije „Vendor id" uređaja te je potrebno napraviti tablicu ovih oznaka.

5.3 Ispravljanje pogrešaka

Ovisno o odluci o korištenju fizičkog uređaja, za efikasno traženje pogrešaka u kôdu, logiranje i ispravljanje pogrešaka, potrebno je i postaviti "AndroidManifest.xml" datoteku. U aplikacijski element dodaje se atribut `debuggable="true"` [8].

Kada imamo okruženje u kojemu možemo raditi, preostaje nam jedino opisati neke moguće probleme koji nastaju prilikom razvoja. Dakako tema rada nije rješavanje semantičkih i logičkih grešaka, već moguće nedoumice oko koncepata.

5.4 Dizajn formi

Tipični primjer primjene koncepata na krivi način su forme. Naravno valja naglasiti kako neki koncepti nisu namijenjeni da se koriste na ovaj ili onaj način. Problemi kod izgleda formi su ideje dizajnera koje nisu provedive na jednostavan način – putem ugrađenih funkcionalnosti, te ih treba doslovno programirati korištenjem onDraw metode ili slično. Kako je paleta uređaja na tržištu velika, postoji dosta razlika među modelima. Tako na primjer često problem predstavljaju širina i visina ekrana. Android podržava dva načina određivanja veličine elemenata: statični i relativni. Dolazimo do zaključka kako je bolje koristiti relativni način određivanja veličine elemenata s obzirom na raspon veličine ekrana. Dakako ovakav način pozicioniranja za sobom dovodi specifične probleme. Ne rijetko dolazi do problema prilikom pozicioniranja elemenata ili previše kompleksnog dizajna. Provođenje takvog dizajna ponekad nije efektivno, te bi se dizajnera trebalo obrazovati. Google tj. Andorid se i za to pobrinuo, te na svoje stranice stavio paket za dizajnere (*engl. designer pack*) u kojem se nalazi skoro sve za aktivni početak dizajniranja. Također se na istima nalazi i vodič kako započeti, na što paziti i slično. Poznati „problem" prilikom izrade formi jest pozicioniranje tekstualnih oznaka (*engl. label*) u odnosu na polje za unos podataka bez definiranja fiksnih veličina. Rezultat je često jedan element prikazan lijevo, a drugi desno. Nažalost rješenje ovog problema je široko prihvaćeni „hack" prikazan isječkom:

```
<TableLayout
     android:id="@+id/buttons"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content">
           <TableRow>
       <ImageButton
             android:id="@+id/button_left "
             android:layout_width="0dip"


android:layout_height="wrap_content"
             android:layout_weight="1" />
<ImageButton
       android:id="@+id/button_right "
       android:layout_width="0dip"

android:layout_height="wrap_content"
             android:layout_weight="1" />
          </TableRow>
</TableLayout>
```

Kôd 24 Pozicioniranje kontroli relativno jednu do druge

Isto tako važno je znati prednosti i mane određene grupe pogleda (engl. View), kako biste mogli iskoristiti sve njegove prednosti a možda i sakrili mane.

Ako se na kratko osvrnemo na prije navedeno, primjećujemo kako programiranje u Androidu može jako brzo postati jako komplicirano.

6.5 Dinamične liste

Tipičan primjer kompliciranja su dinamične liste vrijednosti. Takvom problemu se može pristupiti na dva načina, ovisno o mjestu i načinu primjene. Zamislimo dva scenarija, jedan scenarij potrebuje listu vrijednosti iz baze za izbor neke od ponuđenih vrijednosti, dok drugi predstavlja pretraživanje datoteke u sustavu.

Adresirajmo prvo prvi scenarij. Raditi novu formu za izbor radi ovakvog problema može narušiti dizajn. Među naprednijim konceptima spominjali smo “Adaptere”. Isti su kreirani baš za ovakve svrhe.

```
ArrayList<Map<String, String>> server_list =
                list_values;
String[] from = {"name", "version"};
int[] to = {R.id.name, R.id.version};
SimpleAdapter adapter = new
SimpleAdapter(getApplicationContext(), list,
R.layout.list, from, to);
list.setAdapter(adapter);
```

Kôd 25 Dinamične liste uz pomoć adaptera

Primijetimo neke prednosti ovoga kôda: vrlo jednostavna primjena ovog koncepta je kada je poznat cilj i svrha, može se napraviti efikasno i modularno rješenje zahvaljujući sofisticiranom adresiranju polja (može se mijenjati način prikazivanja bez da se dira prikazani dio kôda), te brzina. Naime, u programiranju manje je više (engl. “less is more”).

Što se tiče drugog primjera, liste datoteka koje biste pretraživali, koristili bismo drugačiji adapter. Malim pretraživanjem dokumentacije primijetili bismo kako Android podržava nešto nalik na “Activity” ukoliko koristimo “ListView” kao omotač naših elemenata unutar forme. “ListView” je pogled u obliku liste, koji ima predefinirane akcije kao na primjer pretraživanje (engl. scroll). Prije spomenuti “Activity” je u stvari “ListActivity” - klasa koja proširuje activity. Putem nje može se prikazati lista i koristiti sve prednosti ovog pogleda. Također kao u prethodnom primjeru postavljamo adapter na listu, u ovom slučaju na “Activity” što rezultira jednom listom, preko cijelog ekrana. Tada je jednostavno napraviti dodatnu funkcionalnost obzirom na neki od događaja.

Dakle, dva slična koncepta primijenjena na znatno različitim primjerima približavaju nas shvaćanju rada i razvoja aplikacija u ovom okruženju. Vrlo je važno, razlikovati i primjenjivati koncepte gdje su potrebni na način kako su zamišljeni.

5.6 Dretve

U napredne koncepte svrstane su i dretve. Višedretvenost je ponekad važna, no ne i neizbježna. Android posjeduje razne koncepte kojima omogućava izvršavanje aplikacije, te su dretve samo jedan segment istih. Prije same upotrebe dretvi trebali bismo znati prvo neke činjenice. Android posjeduje ANR događaj (*engl. Application Not Responding*) kako bi osigurao stabilnost OS-a [10]. Ukoliko je “UI Thread” blokiran dulje od približno 5 sekundi, javlja se ANR dijalog. Dedukcijom dolazimo do zaključka kako ne smijemo zaustaviti izvršavanje „UI Thread” iz razloga što to predstavlja loše korisničko iskustvo (*engl. User Experience*). Ako baš želimo koristi dretve, prebacujemo se na posebne, pozadinske dretve (*engl. backround threads*) ili dretve radilice (engl. worker threads) koje služe za izvršavanje zadatka na način da ne blokiraju glavnu dretvu. Android posjeduje razne načine komunikacije između dretvi, pa čak i komunikacije sa glavnom dretvom. Za razliku od kompliciranih dretvi, Android nudi sličnu zamjenu u vidu izvršenja asinkronog zadatka (*engl. AsyncTask*). Isti za razliku od dretve može biti pokrenut samo jedanput, te mora biti stvoren na glavnoj dretvi. Također ima mogućnost rada u pozadini i to putem ugrađene funkcije `doInBackground` [10].

```
new Thread(new Runnable()
{
    public void run()
    {
      final Bitmap b = loadImage();
      image.post(new Runnable() {
        public void run() {
         image.setImageBitmap(b);
        }
      });
    }
}).start();
```

Kôd 26 Stvaranje dretve

```
private class DownloadImage extends
AsyncTask<String, Void, Bitmap>
{
     protected Bitmap doInBackground(String
url)
     {
         return loadImage (url);
     }
     protected void onPostExecute(Bitmap
result)
     {
         image.setImageBitmap(result);
     }
 }
```

Kôd 27 Stvaranje AsyncTask-a

Kratkim osvrtom primjećujemo kako Android očito pruža dosta jako razvojno okruženje. Uz dobru dokumentaciju i dobru ideju, lako je razvijati bogate aplikacije. Nešto malo vremena za programiranje, te nešto više vremena za razmišljanje rezultira formulom uspjeha u ovom okruženju.

6. ZAKLJUČAK

U radu su opisani samo neki osnovni i neki napredniji koncepti koji se koriste prilikom izrade Android aplikacija. Ukazano je na mnoge prednosti i mane rada u ovoj platformi. Konzistentno provođenje kvalitetne prakse i kulture programiranja na Android platformi ju svakako čini jednostavnijom i primamljivijim izborom. Koncepte je uvijek potrebno koristiti na način kako su i zamišljeni. Takvim pristupom aplikacije mogu dobivati na svrsishodnosti, te platforma profitirati zbog svojih načela otvorenog koda. Android platforma je od samih početaka bila stigmatizirana kao konkurencija Apple iPhone platformi (iOS). I kao takva se morala boriti kvalitetom kako bi opstala na tržištu. U zadnje vrijeme se sve češće može čuti i pročitati kako Android polako nadilazi iPhone i iOS. Tome se ponajviše može zahvaliti velikoj zajednici otvorenog koda koja do Androida nije imala svog predstavnika na tržištu mobilnih platformi. Google kao pokretač Android pokreta puno energije i resursa ulaže u održavanje zajednice kako bi ta ista zajednica vratila uloženo kroz unapređenje Android platforme. Zbog svog ekskluzivnog odnosa prema svojim developerima Apple sigurno privlači jedan dio zajednice na svoju stranu unatoč tome što je financijski puno teže biti iOS developer nego Android developer gdje su svi resursi javno i besplatno dostupni osim, naravno, fizičkog uređaja.

U samo 3. godine, od ne tako davne 2008. godine, primjećujemo solidan utjecaj ove platforme na mobilni svijet. Napredak, dorade, performanse a i tržište svakako govore za sebe. Tržište mobilnih aplikacija je danas

jedno od najkonkurentnijih tržišta gdje svi veliki igrači ulažu mnogo resursa za razvoj. Prepoznato je da je mobilna platforma sama po sebi beskorisna ukoliko ne posjeduje široku paletu aplikacija. Model malih i relativno jeftinih aplikacija na jednom mjestu se pokazao izuzetno primamljivim za potencijalne korisnike mobilne platforme. Mobilna platforma mora svojim tehničkim karakteristikama i jednostavnošću razvoja aplikacija moći privući što više developera koji će oplemeniti platformu za dodatnim aplikacijama. Uz tehnički dio, proizvođač platforme mora developerima pružiti i primamljiv poslovni model po kojem developer sudjeluje u dobiti koja je nastala prodajom aplikacije koju je on razvio. Nedostatak Android platforme je jedino u činjenici da su konačni korisnici, u pravilu, došli iz zajednice otvorenog koda i navikli su na besplatne aplikacije.

Kao i u drugim područjima, sraz između kvalitete i kvantitete te između poslovnih modela koji uključuju i ne uključuju plaćanje ima značajne i duboke argumente koji vuku na jednu i na drugu stranu. Smatramo da je najbolje ostaviti čitatelju da sam odluči.

Literatura:

1 Morrill D.: Announcing the Android 1.0 SDK, release 1, Android Developers Blog, 2008.
Izvor: http://android-developers.blogspot.com/2008/09/announcing-android-10-sdk-release-1.html (učitano: 14.4.2011.)

2 Rogers R., Lombardo J., Mednieks Z., Meike B.: Android Application Development: Programming with the Google SDK, O'Reilly Media Inc., Sebastopol, 2009.

3 ***: Android Basics: Application Fundametals, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/fundamentals.html (učitano, 30.04.2011)

4 ***: Framework Topics: Audio and Video, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/media/index.html (učitano, 30.04.2011)

5 ***: Framework Topics: Data Storage, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/data/data-storage.html (učitano, 30.04.2011)

6 ***: Framework Topics: Graphics, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/graphics/index.html (učitano, 07.05.2011)

7 ***: Framework Topics: Location and Maps, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/location/index.html (učitano, 30.04.2011)

8 ***: Framework Topics: The AndroidManifest.xml File, Android Developers, 2011.
Izvor: http://developer.android.com/guide/topics/manifest/manifest-intro.html (učitano, 09.05.2011)

9 ***: Framework Topics: User Interface, Android Developer, 2011.
Izvor: http://developer.android.com/guide/topics/ui/index.html (učitano, 14.05.2011)

10 ***: Technical Resources: Painless Threading, Android Developers, 2011.
Izvor: http://developer.android.com/resources/articles/painless-threading.html (učitano, 30.04.2011)

Podaci o autorima:

Ognjen Ribičić, univ. bacc. inf.
e-mail: ognjen.ribicic@foi.hr

Ognjen Ribičić, univ. bacc. inf., je od svoje 13. godine u stalnom doticaju sa računalima i informatičkim svijetom, što i odabire kao svoju primarnu struku. 2009. godine završava svoje prvostupničko obrazovanje na Tehničkom veleučilištu u Zagrebu, te iste godine nastavlja naobrazbu na Fakultetu organizacije i informatike u Varaždinu. Područja interesa su mu razvoj mobilnih i Web aplikacija, te razvoj, održavanje i modeliranje baza podataka, što i jest njegovo usmjerenje na diplomskom studiju.

Boris Tomaš, mag. inf.
tel: +385 42 390 853
fax: +385 42 213 413
e-mail: boris.tomas@foi.hr

Boris Tomaš, mag. Inf. je od 2010. godine zaposlen kao asistent na Katedri za razvoj informacijskih sustava na Fakultetu organizacije i informatike u Varaždinu. Na istom fakultetu je i student poslijediplomskog doktorskog studija Informacijskih znanosti. Područja interesa su mobilne aplikacije, razvoj programskih proizvoda, razvoj GIS aplikacija, multimedijski sustavi i aplikacijski marketing te marketing programskih proizvoda. Posjeduje dugogodišnje iskustvo sa Microsoft tehnologijama za razvoj aplikacija za većinu arhitektura i platformi.

Zlatko Stapić, mag. inf.
tel: +385 42 390 853
fax: +385 42 213 413
e-mail: zlatko.stapic@foi.hr

Zlatko Stapić, mag. inf. je od 2006. godine asistent na Katedri za razvoj informacijskih sustava na Fakultetu organizacije i informatike u Varaždinu, te polaznik poslijediplomskog doktorskog studija Informacijske znanosti na istom fakultetu. Njegova nastavna aktivnost je prvenstveno usmjerena na kolegije koji se odnose na programsko inženjerstvo, analizu i

razvoj programa, modeliranje poslovnih procesa i razvoj informacijskih sustava, te značajne napore ulaže u radu sa studentima za što je dobio i posebna priznanja.
Iz znanstvenog i stručnog rada treba izdvojiti višegodišnje voditeljstvo stručnih projekata razvoja programskih proizvoda i sudjelovanje na različitim stručnim i znanstvenim projektima iz područja razvoja, unapređenja poslovnih procesa, projektnog menadžmenta i slično.
U posljednje vrijeme se intenzivno bavi razvojem aplikacija za mobilne uređaje, što je i predmet njegovog istraživanja u okviru doktorske disertacije, a osobito je vrijedno istaknuti da razvija za skoro sve platforme, uključujući između ostalog Android, Symbian te Windows Phone 7. Zlatkov detaljniji životopis, s popisom svih radova, projekata i nagrada, te drugih važnih podataka može se pronaći na njegovoj osobnoj web stranici, na http://www.foi.hr/djelatnici/zlatko.stapic.

Zlatko Stapić is a young researcher and teaching assistant at the Faculty of Organization and Informatics working at the Information systems development department. His main areas of interests include classic and agile software engineering methodologies, software and information systems development, business processes modeling and others.

Fakultet organizacije i informatike

Pavlinska 2

42000 Varaždin