Korištenje Android Annotations i Active Android razvojnih okvira

Using Android Annotations and Active Android development frameworks

Alen Huskanović, David Ante Macan, Pavlović Milan

SAŽETAK

Android je danas najpopularniji operacijski sustav za mobilne platforme. No, unatoč svojoj popularnosti, nativan razvoj aplikacija za Android zna biti poprilično dugotrajan muktorpan posao, te je ponekad čak i za jako male i jednostavne aplikacije potrebno ubaciti puno koda. Upravo iz tog razloga, danas se javlja sve više i više novih i inovativnih open-source rješenja koja smanjuju taj problem iz pojedinih aspekata. Android Annotations i Active Android upravo su takva rješenja koja se danas koriste sve više u sve ozbiljnijim projektima. Active Android ORM je alat koji uvelike olakšava manipulaciju bazom podataka, dok je Android Annotations razvojni okvir koji nam pruža skraćeno pisanje mnogih popularnih dijelova koda koji se s vremenom i veličinom aplikacije sve više gomilaju. Oba ova razvojna okvira koriste anotacije u programskom jeziku Java, te uvelike skraćuju pisanje koda, što omogućuje developerima da se više fokusiraju na inženjering i dizajn aplikacije, nego na samo pisanje koda.

ABSTRACT

Nowadays, Android is the most popular mobile operating system. But, although its popularity, native Android application development can be quite time and energy consuming work. Sometimes, it is necessary to write a lot of code even for small and simple applications. That is exactly why there are more and more various new open-source solutions that take care of those problems from different aspects. Android Annotations and Active Android are such solutions that are being more and more used in lot of serious projects. Active Android is an ORM tool that eases database manipulation a lot. On the other hand, Android Annotations is an framework that offers us shortened writing of the popular parts of code, and increases the complete code readability. Both of these frameworks use annotations in Java programing language and reduce the code writing a lot, which enables the developers to be more focused on engineering and design of the application, instead on focusing on code writing.

1. INTRODUCTION

The successful development of mobile applications is largely dependent on the quality of written code. Unfortunately, today the code is increasing rapidly and it should be reduced. Specifically, regarding the development for Android, very often for some general things there must be code written that is already mostly predefined and in the high ratio it does not change from application to application. By writing such code, developers spend precious time reading it and it becomes cumbersome and unreadable.

From the above mentioned reasons, some developers are trying to make their own development frameworks that will replace the entire predefined code with only a few lines of their own, and with that save a lot of time.

The aim of this paper is to introduce developers who develop applications for Android that there are developmental frameworks that can help them with the speed and quality of development of mobile applications.

This paper is divided into three chapters; in the first one, Active Android development framework will be explained, which is ORM for Android. The next chapter is intended for Android Annotations development framework, which with the help of Java Annotations helps in reducing a lot of code. The final chapter is intended for the comparison of native approach with the usage of a development frameworks where the advantages will be shown, as well as the disadvantages of using those very same frameworks.

2. ACTIVE ANDROID

2.1. Introduction to Active Android

As an Android operating system framework, Active Android is an ORM (object relational mapper) library. Active Android allows you to manipulate data in database over SQLite database management system. It provides quick and easy usage without using any of the SQL queries directly. Developers don't even need to know SQL to use this framework[1]. Active Android provides usage with the records in the database via objects, so that the developers don't have problems with conversion of tabular entries into objects and vice versa. Model class represents a table and class instance represents a record in table. Database management with Active Android is much simpler and faster because developers can create a class and with annotation tell the system that that class represents table in the database. Active Android does all the work around the database setup so that the developers don't have to write CRUD methods themselves.

Nowadays the Active Android is one of the most useful frameworks for programming applications on Android operating system as shown by the increasing number of users. The source code of the framework can be found at Github where everyone can take it and use it[2].

2.2. Using Active Android

In this section, we will create a very simple mobile application with database for chocolate sale. We will create this database in both standard and Active Android way. Let's create a use-case here and say that we wish to delete the first receipt with a salesman "Mario" who sold the chocolate "Dorina".

Figure 1 - ER model of example database

For that, we need to do the following steps:

- ✓ Create tables and model classes
- ✓ Insert required data
- ✓ Read the inserted data
- ✓ Print the data to the screen

2.2.1. Native approach:

Using standard approach without Active Android, we usually need to create DatabaseHelper class that contains methods and attributes for setting up and updating the database. Using this class, we need to specify both database name and database version, alongside with adding SQL code for creating and dropping every single table that our database will contain.

To improve efficiency and reduce typo errors, we usually put the CREATE TABLE string into the class that represents the table that is to be created. We created the SalesmanSQL.java and ReceiptSQL.java in the same manner. After the representative model classes were created, we need to create DatabaseHelper class that helps us connect to the database and creates tables using previously mentioned CREATE TABLE strings.

And final class that we need to create is called DatabaseAdapter. This class provides us with all the CRUD methods for our tables. Unfortunately, coding of this class usually takes quite a lot of time. The problem with database adapter is that it is usually very large, because we need to add qute a lot of code for each table individually. After setting up the database, we can start using it.

2.2.2. Active Android approach:

To use Active Android, we just need to setup few things and we are ready to code. First of all, we need to override default Application class for our app and add a reference to it in the manifest file. Besides that, we also need to write database name and version inside the manifest file. We do this like in the following example:

AndroidManifest.xml

```
...
<application
    android:allowBackup="true"
    android:name="CaseApp"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <meta-data
      android:name="AA_DB_NAME"
      android:value="CaseApp.db" />

    <meta-data
      android:name="AA_DB_VERSION"
      android:value="1" />
...
```

Now we are ready to start coding our database. Instead of creating DatabaseHelper and DatabaseAdapter classes, we just need to add few modifications to the existing model classes:

Receipt.java

```
@Table(name = "receipts")
public class Receipt extends Model {

  @Column(name = "chocolatte")
  private Chocolatte chocolatte;
  @Column(name = "salesman")
  private Salesman salesman;
  @Column(name = "time")
  private String time;

  public Receipt() {
  }
...
```

As shown above, using Java's annotation processing tool, Active Android allows us to define our table inside it's appropriate model class. Notice also that this time we didn't add the id attribute in our class. That is because Active Android itself handles it and Model class allows our classes to inherit that attribute. As for the foreign keys, Active Android improves that part as well. If we put into a column an attribute that is a child of Model class, Active Android automatically creates foreign key for that column. Another noteworthy fact is that Active Android provides us a way to just query the database and it handles the whole "linking and communication" process with the database, which allows the developer to focus more on designing and developing, instead of mindless coding.

Receipt.java

```
...
//Some query methods
public static Receipt readReceiptById(long id) {
    return new
Select().from(Receipt.class).where("Id =? ",
id).executeSingle();
}

public static List<Receipt> readAllReceipts() {
    return new
Select().from(Receipt.class).execute();
}

public static Receipt
readReceiptForSalesmanAndChocolatte(Salesm
an salesman, Chocolatte chocolatte) {
    return new
Select().from(Receipt.class).where("salesman
=? AND chocolatte =? ",
    salesman.getId(),
chocolatte.getId()).orderBy("Id").executeSingle(
);
}
...
```

Finally, we can show how to use the database:

- ✓ Creating chocolatte, salesman and receipt:

```
Chocolatte dorina = new Chocolatte("Dorina",
4.00);
dorina.save();
Salesman mario = new Salesman("Mario", 22);
mario.save();
new Receipt(dorina, mario,
currentDate()).save();
```

- ✓ Reading that receipt from the database

```
Receipt receipt =
Receipt.readReceiptForSalesmanAndChocolatte
(mario, dorina);
```

- ✓ Deleting that receipt from the database

```
receipt.delete();
```

3. ANDROID ANOTATIONS

3.1. Introduction to Android Annotations

Lately, among Android developers is discussed the concept of Diet Driven Development, and anyone who is familiar with this concept is also familiar with Android Annotations framework which will be presented in this chapter[3].

Android Annotations is a development framework for the Android operating system which reduces the unnecessary code; using Java annotations, developers can show their intent and let Android Annotations generate the plumbing code at compile time[4].

This development framework helps developers in writing neat and readable code, and thus allows faster production and faster debugging while developing Android applications. Android Anotations is free to use, and full source code is available on GitHub[5].

3.2. Using Android Anotations

In this section, we will try to induce some of the practical usages for Android Annotations framework. Let's start with the first class that is created when the project is generated. That is MainActivity class that (usually) extends Activity. Every Android developer knows that, by default, every Activity has its onCreate method which gets called every time the activity gets created. But the problem is that this method looks almost always the same, but it takes couple of lines of code just to write it and specify the layout for the activity. Using Android Annotations, we can clean that code as shown in next example:

```
@EActivity(R.layout.activity_main)
public class MainActivity extends Activity {
  @AfterViews
  void main() {
        // Do something
  }
}
```

As shown in the example above, it is very simple to specify the designated layout for the activity, as well as the method that is to be invoked after activity gets created. If our activity is to be used with Android Annotations, we must provide an annotation @EActivity (as shown in the example above). Using this analogy, it is clear that @AfterViews annotations specifies the method that is called after the activity is created. This is a very simple, but effective demonstration of Android Annotations power.

Another thing in Android that can easily increase the code amount a lot (especially if it is done multiple times) is the process of binding xml elements together with the Java variables. Using native approach, we need to take following steps:

- ✓ Create designated View variable
- ✓ Specify that the variable is to be used for View element with certain id
- ✓ Cast the View if necessary

Using Android Annotations, this is done simply by Adding @ViewById annotation as shown in the next example:

```
@ViewById
TextView txtLabel;
```

The code above automatically binds View element with id "txtLabel" from given layout to the txtLabel variable. Fortunately, Android Annotaions can detect variable name and recognize it as an id, but we aren't forced to use that naming convention. If we decide to call id different than the variable name, we can simply add an annotation attribute with the elements id like this:

```
@ViewById (R.id.txtMyId)
```

There are plenty more situations like this in which Android Annotations provide a simple and elegant solution. For example, there is a @Click annotation which is used to handle click events for designated views, @ItemClicked annotation that handles list item click events, etc.

Another great example of necessary native complex code is the AsyncTask. By default, AsyncTask is

used for safe and proper synchronization of the background thread and UI thread. The only problem is that it is relatively large and complex to write. Using AndroidAnnotations, this is solved in again, very simple and elegant way. Simple way of replacing AsyncTask with Android Annotations is shown below:

```
@Background
private void doInBackground() {
  // Do some background work
  updateUIThread();
}

@UiThread
private void updateUIThread() {
  // Do some UI work
}
```

So, the big and complex AsyncTask is replaced with two simple methods. When called, "doInBackground" method will do its work in background thread and update UI Thread when done.

There are many more annotations in this framework that aim to ease native Android development a lot. Some of the things that are improved a lot with this framework are:

- ✓ Rest API implementation (@Rest, @Get, @Post, …)
- ✓ Activity saved instance state (@InstanceState)
- ✓ SeekBar events (@SeekBarProgressChange, @SeekBarTouchStart, …)
- ✓ Text change events (@TextChange)
- ✓ Options menu (@OptionsMenu, @OptionItem, …)
- ✓ And many more….

4. DISCUSSION

4.1. Comparing Active Android and Android Anotations with native approach

As stated before, both Android Annotations and Active Android frameworks are used for reducing the amount of code needed for implementing some functionalities in our Android applications.

Android Annotations framework makes our code more readable and easier to use and understand by replacing (not removing) a lot of code that is either generated or necessary for the right implementation. A lot of Android developers that use native approach agree that they often find themselves losing quite a lot of time just for setting up and preparing some functionalities.

As for the Active Android framework, not only does it reduce the code needed for implementation, but it also allows us to design our database using Object-Oriented approach, without having to think a lot about modeling the database itself.

4.2. Advantages and disadvantages

Using a framework in application development along with benefits also brings a number of disadvantages and limitations. Usually, the benefits are far above the limit, and in most cases development frameworks are worth using.

Using Active Android and Android Annotations developers get a great set of advantages. Both developmental frameworks are easy to use and maintain, and perhaps one of the most important benefits is the reduction of errors. It is widely known that errors are mainly due to human factors, and by using development frameworks the vast majority of the code is automatically generated and developers are not obliged to write these parts, which reduces the probability of one of them to make a mistake.

Both development frameworks have a lot of influence on the number of lines of code, and it goes in the direction of Diet Driven Development. According to some informations, using just Android Annotations in average reduces the average number of lines of code for 42%, and using Active Android gives even more reduction[3]. Code written using a development framework is in most cases more readable, and easier to determine later why is some section of code used.

The disadvantages the developers face using development frameworks are largely related to the inability of customizing individual parts, while Active Android and Android Annotations offer these possibilities. These problems are eliminated by an open source development framework that anyone can customize to their liking. With all attempts to eliminate drawbacks, there are still some that the community is trying to solve, for example in Active Android function for adding records to a table is void, and the only way to check whether a record

successfully entered or not requires selecting the base immediately after entry. Development frameworks are quite large, and it is not easy to manage them while trying to customize.

Active Android as an ORM does support only the code-first approach, so a base model (classes) can not be made from tables[6].

Each development framework, including these two, requires configuration and customization of a project to their mode before it can be used at all, and to inexperienced programmers it can add complications with the development which leads to droping out the development framework at the first step, the configuration.

5. CONCLUSSION

This paper briefly summarizes the advantages and disadvantages of using frameworks such as Android Annotations and Active Android for native android development. Android is a tremenduous platform, and it's development is great. But, as allways, there is room for improvement. While developing applications in a standard way, developers often have to add or generate relatively large ammounts of code to acomplish some really trivial functionalities. But with usage of additional frameworks, as shown in this paper, that code could be easilly reduced and it's readibility improved, which is often a very important factor.

To conclude this paper, we will add a quote from Robert C. Martin to show how important clean and readible code is:

„The ratio of time spent reading [code] versus writing is well over 10 to 1 [therefore] making it easy to read makes it easier to write.“ [7]

REFERENCES

[1] "ActiveAndroid Guide," *CodePath*. [Online]. Available: http://guides.thecodepath.com/android/ActiveAndroid-Guide#overview. [Accessed: 21-May-2014].

[2] M. Pardo, "Active Android source code," 21-May-2014. [Online]. Available: https://github.com/pardom/ActiveAndroid.

[3] P.-Y. Ricau, "Diet Driven Development," *Devoxx*. [Online]. Available: http://www.devoxx.com/display/DV12/Android+DDD+(Diet+Driven+Development)! [Accessed: 16-May-2014].

[4] "AndroidAnnotations offical site," *AndroidAnotaions*. [Online]. Available: http://androidannotations.org/. [Accessed: 21-May-2014].

[5] "AndroidAnnotations source code," *Github*. [Online]. Available: https://github.com/excilys/androidannotations. [Accessed: 21-May-2014].

[6] Ž. Plesac, "Infinum Talks - Active Android." [Online]. Available: http://www.slideshare.net/Infinum/infinum-android-talks-02-activeandroid. [Accessed: 21-May-2014].

[7] R. C. Martin, Ed., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

INFORMATION ON AUTHORS:

Alen Huskanović
e-mail: ahuskano@foi.hr
Faculty of Organization and Informatics

Alen Huskanović is a regular student of the third year of undergraduate study Information Systems at the Faculty of Organization and Informatics in Varazdin. He has participated in various national and international competitions, among which stands out IEEEmadC international competition for the design of mobile applications where he won first prize. Actively working on the system PEAS with whom he came to Croatian finals of Microsoft Imagine Cup. He is primarily engaged in the development of applications for the Android operating system.

David Ante Macan
e-mail: amacan@foi.hr
Faculty of Organization and Informatics

David Ante Macan is a third year undergraduate student of Information Systems at Faculty of Organization and Informatics. He has participated in various competitions, including the Croatian Microsoft Imagine Cup finals, won the best design award at IEEEmadC contest, won the third place at Infinum student Hackathon 2014, etc. His main area of interest is mobile development, especially Android. He is actively working on several projects, including the Personal Exam Assistant (PEAS) project at Faculty of Organization and

Informatics. He is also a member of IEEE Institute and is one of the student coordinators and founders of the Laboratory for mobile technologies (MT Lab) at Faculty of Organization and Informatics.

Milan Pavlović
e-mail: mpavlovi2@foi.hr
Faculty of Organization and Informatics

Milan Pavlović is a third year full time undergraduate student at Faculty of Organization and Informatics, University of Zagreb. He is also one of initial member of FOI MT Lab. His main interests are Information Systems engineering and development. He is also interested in Java desktop, mobile and web application development and technology.