Aspect-oriented User Interface Design for Android Applications[1]

Jiří Šebek, Karel Richta

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague, Karlovo nám. 13,
121 35 Praha 2, Czech Republic
{sebekji1,richta}@fel.cvut.cz

Abstract. This paper deals with the design of an effective Android framework that will allow a developer to create Android applications easily in a short time with the help of aspect-oriented approach. Our solution enables to deal with separated aspects like security, layout, input validation, data binding and presentation independently. Our presented framework is compared to conventional development approach of mobile applications and also is compared to the framework Aspect Faces that is also uses aspect-oriented approach, but is designed for Java EE applications. Each aspect of framework was tested and it was proven that our framework is effective in the following areas. It does not slow down the developed application according to the same application created with XML, it makes the code to be more readable, and it makes development faster, and reduces the number of code lines that developer has to write down.

Keywords aspect-oriented approach, aspect-driven design, entity inspection based approach, run-time aspect model, reduced maintenance and development efforts

1 Introduction

The main aim of this paper is to present a design of a new Android framework that will allow a developer to create Android applications with minimal effort, in a short time, and with the help of an aspect-oriented approach.

In the conventional approach, we are mixing a code of all miscellaneous aspects together in one big code. The aspect-oriented approach in a software development means that we are focusing on separated aspect. These aspects are: security, layout,

[1] This work has been partially supported by the Grant Agency of CTU No. SGS15/210/OHK3/3T/13 and partially also by the AVAST Foundation

input validation, data binding and presentation. As a result, the developer using aspect-oriented approach can write less amount of code, which is moreover reusable. We also avoid a spaghetti code (code that has tangled structure), and a redundancy of code, and other bad habits in the programming.

2 Background

Within all operation systems (OS) for a mobile device, the Android is the most expanded as shown in Figure 1. Because of that, applications targeted for Android are very desirable. As you can see from Figure 1, the ratio of applications targeted for the Android OS for mobile device is steadily growing up. This is the reason why developers cannot omit the Android in their analysis.

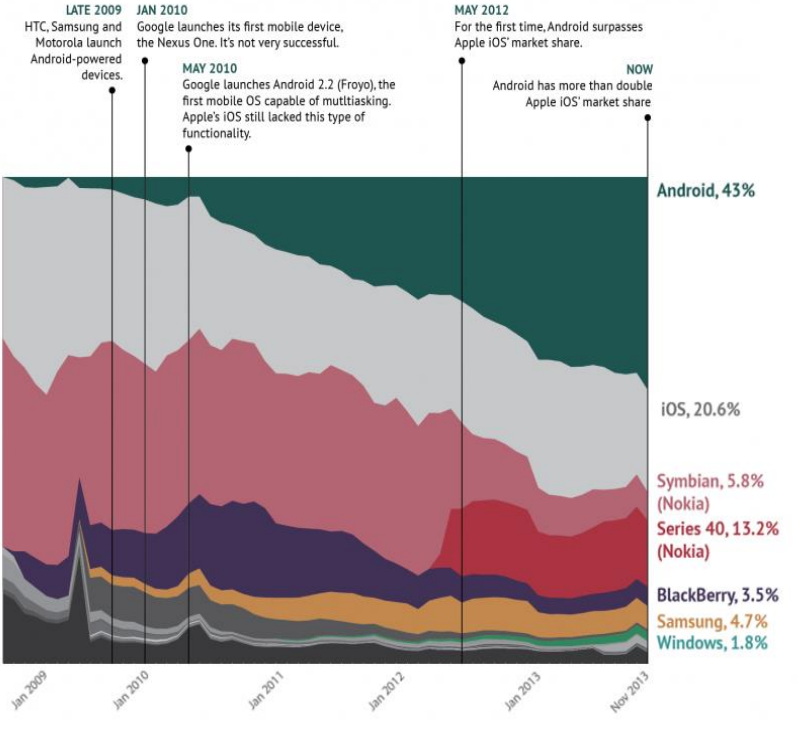

Fig 1. Distribution of all OS for mobiles (adopted from [11])

2.1 Conventional Approach

In conventional development approach of Android applications, every screen has two parts. The first part is Java class that extends Activity class and where you can place

your logic and you have to connect your Java class to view. That view is a second part of the screen description and can be done in XML or generated by a program. Usually, XML choice is better because you can separate, at least, layout from remaining code. The readability depends on the developer and sometimes it's hard to maintain resulting code.

2.2 Aspect-Oriented Approach

Aspect-oriented programming (AOP) approach is a paradigm whose main goal is increasing modularity. Usually the code of applications can be separated into logical sections. These logical sections are called aspects. AOP supports separating development of any aspects option, resulting in a better code. In OOP designs, we use classes to describe only instances and their attributes. That is the representation for data only. Therefore, the best way to do things in aspect way approach is to add these additional pieces of information to the same representation (class). This is called Rich Entity Aspect/Audit Design (READ) [2,3]. Above any instance, attribute or method you can place annotation containing additional information. By this procedure, we can add all required aspects.

3 Related Works

Approach in the articles [2,3] is not the only way how to generate User Interfaces (UI) from the model. The topic of UI generated from domain objects is mentioned in [6,7]. The framework is called Meta-widget and it is based on Model driven development (MDD). The user just creates objects and puts them to Meta-widget's framework. The UI is generated according to the model. Meta-widget supports a lot of technologies from Android, Google Web Toolkit (GWT), HTML 5 (POH5), JavaScript to JSF and JSP. Meta-widget works in three basic steps. First, Meta-widget comes with a UI component native to your existing front-end. Second, Meta-widget inspects, either statically or in the run-time, your existing back-end architecture. Third, Meta-widget creates native UI subcomponents matched to the back-end. In articles [2,3], the other aspects were added based on annotations. Meta-widget adds this information based on existing back-end of any applications.

Model driven development (MDD) is based on the idea that the model should be primary centralized place for all information. This model is then compiled or transformed by another way into the deployed application code. The benefits are reduction of information in application and concentration of the structure of information into one place. The disadvantages can be adaptation and evolution management [8]. This approach does not go well with OOP, because we need to maintain the interconnection between the models with the back-end of the application. There exist another tools, how to describe an additional information. These tools are called in the MDD the Domain-Specific Languages (DSL). Sometimes, they are informally called mini-languages, because they describe the additional information inside the other language. There are a wide variety of DSL. Domain-specific languages can be a visual dia-

gramming language, programmatic abstractions, declarative language (OCL) or even whole languages like XSLT. As we can see, some of them evolve into the programming tools that are frequently used (XSLT).

Generative programming (GP) is a specific type of a programming that generates the source code from domain-specific code. The goal is to improve productivity of developer, make the way between application code and domain model, support reuse, adaptation, and simplify management of components [12].

Meta-programming (MP) is a technique, which allows the developer to modify the structure and the behavior of the applications at the run-time. The reflection is one of the options how to implement the MP [5]. The developers can inspect the classes, the fields, the methods at the compile time and they do not even have to know their names at the compile time. The MP allows developers to adapt the application to the different situations. The bottleneck of this solution is the performance. The applications are significantly slower with the MP and are harder to test or debug then the applications without the MP. To deal with this problem the developer can use some cache.

4 Design of Aspect-Oriented Framework

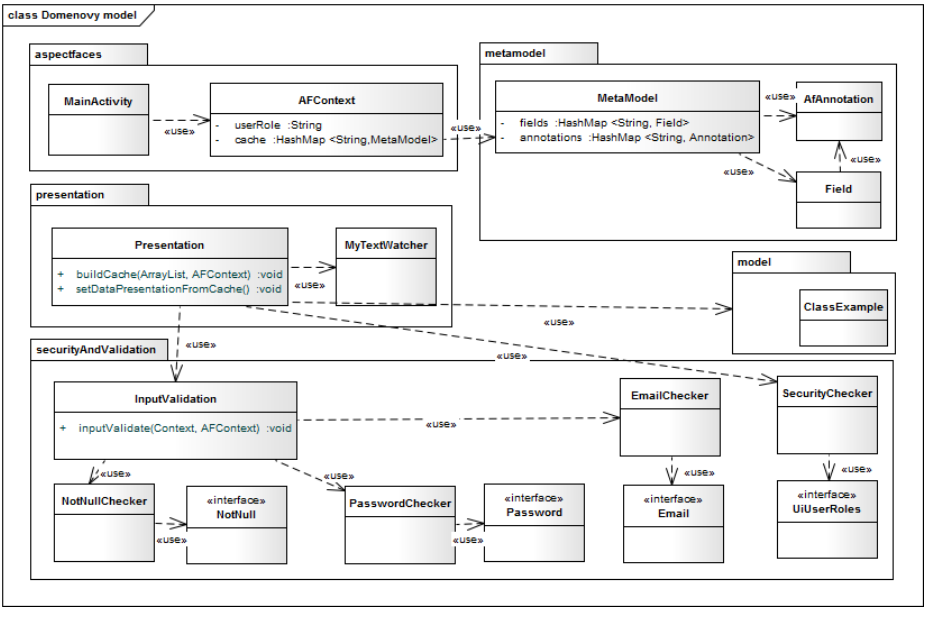

Fig. 2. Analytic model of classes

The analytic model of classes is a diagram which captures a general static view of the application. The purpose of this is to illustrate types of objects, variables and their relationships. Figure 2 shows class diagram of our framework. It does not contain all of the files (classes) because there would be much more objects and the diagram would not be easy to read, but it contains all packages and main functionality.

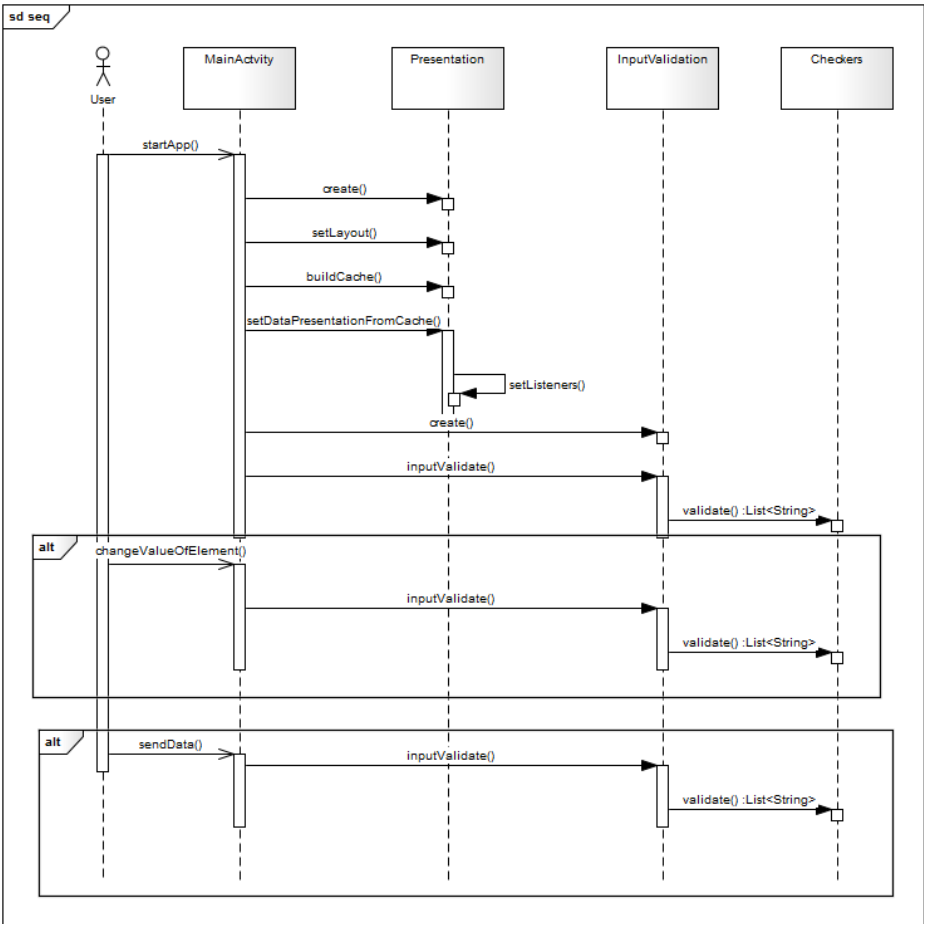

Fig. 3. Sequence diagram of the framework

The sequence diagram is used for a visualization of interactions between processes (objects). It also displays the right order of these interactions. It is a behavioral type of a diagram. Therefore, it is the best choice for showing how the framework works. The diagram includes parallel vertical lines called lifelines and horizontal arrows that represent the messages exchanged between them in the right order as they appear. In figure 3, there is shown basic sequence diagram of our framework. It shows what happens from the start of the application using the framework.

The mandatory action is the creation of a new *Presentation* object. Then, the main activity calls methods *buildCache()*, and *setDataPresentationFromCache()* of this object. *BuildCache()* method creates new cache from given instances. UI can be created much faster from cache, which includes information in hashmaps. The time to retrieve this information is then constant. It makes final application much faster, for example in the case of a fragment style application, where user is often sliding between screens that he/she already visited. When cache is already created, it is not created

again. *SetDataPresentationFromCache()* method creates whole UI from cache. It means layout, data presentation and data binding. The other options of the framework are voluntary, like on the diagram. If the developer wants to validate default data in created instances, he/she just creates the *InputValidation* object and call *inputValidate()*. The framework will care about the rest via created rich entity (normal instance with attributes and with annotations). In this diagram, it is also captured when the user changes the value of element it will call the listener which will call the *inputValidate()*. If the user sends the form with data, it will also call *inputValidate()*.

Fig. 4. Usage of the framework

4.1 Usage of the Framework

In the Figure 4, it is shown how our framework works all together. The basic element is entity (Java object) enriched by other information in form of annotations. Then we have to call the framework and also give over *arrayList* of instances. Everything is generic; framework does not need to know exact type of object which was inserted into *arrayList*. The last part in the figure 4 shows the result after launching the application.

5 Comparison of Aspect-Oriented Approach and Conventional Approach for Android Platform

In the conventional approach, the presentation layer was implemented in XML, the data binding in Java, input validation in Java, layout in XML and security in Java. The project, which was created in conventional way, was developed to implement one form that is the same as the first one in the example of aspect-oriented approach with four attributes (created with our framework). All aspects was coded in common way and not stored in annotations like in the aspect-oriented approach. If we want to do another form, we will have to write the similar amount of code. This code is redundant and is making application hard to maintain. The big difference against AOP is in security, where in aspect-oriented approach we just do not create particular element. Here, we create all elements and then we are changing visibility if the user has particular user role. The advantages of the aspect-oriented approach are shown in Table 1.

Table 1. Comparison of AOP and conventional approach

Feature	AOP	Conventional approach
Reuse	yes	no
Run-time approach	Yes	no
Reduce code	Yes	no
Better to maintain	Yes	no
Separated each aspects	Yes	no
Readable code	Yes	No (depends on developer)
Time to launch the form (average)	119,5ms	193,1ms
Standard deviation (std)	5,35ms	14,7ms
Lines of code (LOC)	29	495

Here, we can see that AOP is definitely better in reuse, reduction of the code, maintenance of the code, separation of aspects, and readability of code. The reuse in conventional approach means copy, paste and edit. That is not a good approach. From Table 1, we can also see the difference in lines of code (LOC). The AOP has 4 lines of Java code and 25 lines of Java class code. The conventional approach has 16 LOC of Java class, 377 lines of Java code and 102 lines of XML code. LOC where counted by regular expression in search function in eclipse IDE. LOC is counted from the view of developer so the body of the framework is not counted. As we know, this is example that has only 4 attributes. If this number rises for example to eight, the LOC for conventional approach will also rise by similar amount of lines. On the other hand, AOP will increase only by about ten LOC (four lines plus some annotation).

The time to launch the form is also an interesting item, because as you can see in Table 1, AOP is faster than conventional approach. The reason of this is the creation of view by XML which is slower than the view created by a program. This time is calculated when the form is first launched. When user is returning to this activity it is even faster for AOP because it is using a cache with the constant asymptotic com-

plexity of access to data. The average time to launch any form was calculated from the list of ten data that was taken as you can see in Table 2.

Table 2. Table of launching times

Test number	Launch time with AOP (ms)	Launch time with Conventional approach (ms)
1	128	196
2	114	171
3	121	210
4	117	222
5	110	183
6	126	175
7	119	201
8	121	188
9	115	196
10	124	189

6 Comparison of Aspect-Oriented Programming (AOP) for Android Platform and Java EE

The aspect-oriented framework, called Aspect Faces for Java EE [2,3], was created on a similar idea as for Java EE, but they are not implemented in a same way as presented framework for Android. The framework for Java EE [2,3] is called by the tag in a view part as we can see the usage in Listing 1. Mostly it is placed in JSP page or in some xhtml page. In Listing 1, there is an example how to create two forms.

Listing 1. Usage of Aspect Faces in Java EE

```
<!-- Form1 generated via Aspect Faces -->
<af:ui instance="#{bean.entity1}" edit="true"/>
<!-- Form2 generated via Aspect Faces -->
<af:ui instance="#{bean.entity2}" edit="true"/>
```

Instead of this approach, the framework for Android is called in Java activity class that user creates. The reason for this is simple. In Android application structure, the Java classes are mandatory unlike in Java EE where if you just want the UI you do not have to create Java bean or some logic behind. Furthermore, the XML files that represent the UI are optional, because you can create UI by a program. If you create a button to another screen in Android application, you are not connecting the button to call another view, but first, you call the Java activity class and in this class, we can choose how to create UI. In Listing 2, there is the basic example how to create form by our framework in Android.

Listing 2. Usage of framework in Android

```
Presentation p = new
Presentation(this,listOfInstances);
p.buildCache(arraylistOfInstances, afContext);
View v = p.setDataPresentationFromCache();
setContentView(v);
```

Both frameworks are using meta-models to save information about instances that are rendered in forms and also both of them are also working in the run-time. One big difference is in the future potential of the development. A native application has the advantage against web application, that it is compatible with the devices hardware such as motion sensors, environmental sensors, position sensors, and camera. The motion sensors include accelerometers, gravity sensors, gyroscopes and rotational vector sensors. The environmental sensors include barometers, photometers and thermometers. The position sensors include orientation sensors and magnetometers. The web applications are limited in this way. Aspect Faces on Java EE can get a position from GeoIP, but nothing more. This information from sensors can be used by framework and can react to that information.

7 Conclusion and Future Work

This paper results from diploma thesis [9] and contains background of aspect-oriented approach, and describes the design of the new framework for an Android application development. We compare our framework with the conventional approach to Android application development, and to Java EE framework called Aspect Faces. Our framework seems to be fast, clear, easy scalable, readable, reusable, improves the maintenance and it was tested. The results of tests show us, that our framework enables faster development than standard conventional approach to Android application development. It is true that, when we comparing programming approach with the approach using XML, XML has the disadvantage in the speed of launching.

In the future work we will focus on extending the framework context with the hardware devices such as motion sensors, environmental sensors, position sensors, and camera. The application can then easily react to change devices position etc. Also the framework will be tested not only against XML approach, but also against another programmatic approach. Another aim of future work will be to test our framework not only by the launching time, but also by time, when screens are just changing. The expectation is, of course, that our framework will be much faster due to the cache system.

References

1. Czarnecki, K. and Eisenecker, U. W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
2. Černý, T., Donahoo, M. J., and Song, E.: Towards effective adaptive user interfaces design. In Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS '13). ACM, New York, NY, USA, 373-380. DOI=10.1145/2513228.2513278, http://doi.acm.org/10.1145/2513228.2513278 (2013)
3. Černý, T., Čemus, K., Donahoo, M. J., and Song, E.: Aspect-driven, Data-reflective and Context-aware User Interfaces Design. In: Applied Computing Review, Vol. 13, Issue 4, ACM, New York, NY, USA, 53-65. ISSN 559-6915, http://www.sigapp.org/acr/Issues/V13.4/ACR-13-4-2013.pdf (2013)
4. Černý, T. and Song, E.: UML-based enhanced rich form generation. In: Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS '11). ACM, New York, NY, USA, 192-199. DOI=10.1145/2103380.2103420, http://doi.acm.org/10.1145/2103380.2103420 (2011)
5. Forman, I. R. and Forman, N.: Java Reflection in Action (In Action series). Manning Publications Co., Greenwich, CT, USA (2004)
6. Kennard, R. and Leaney, J.: Towards general purpose architecture for UI generation. Journal of Systems and Software, 83(10) http: / / metawidget . sourceforge . net / media / downloads / Towards a General Purpose Architecture for UI Generation.pdf (2010) 1896-1906
7. Kennard, R. and Robert, S.: Application of software mining to automatic user interface generation. In SoMeT'08. http: / / metawidget . sourceforge . net / media / downloads / Application of Software Mining to Automatic User Interface Generation.pdf (2008) 244 - 254
8. Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg. A.: Models@run.time to support dynamic adaptation. Computer, 42(10) (Oct. 2009) 44-51
9. Šebek, J.: Aspect-oriented user interface design for Android applications, diploma thesis. Department of Computer Science, CTU FEE, Prague (2014)

Internet resources

10. Android Activity Lifecycle. Android Activity Lifecycle [online]. 12/22/2011, [cit. 2014-04-28]. http: / / www . mikestratton . net / 2011 / 12 / android-activity-lifecycle (2011)
11. Android vs iOS. Android vs iOS [online]. November 11 2013 3:22 PM [cit. 2014-04-28]. http: / / www . ibtimes . com / android-vs-ios-whats-most-popular-mobileoperating-system-your-country-1464892 (2013)
12. Introduction To Android Mobile Operating System. Android Development, Tutorials [online]. August 1, 2011 [cit. 2014-04-29]. http://www.blogsaays.com/tutorial-part1-introduction-android-mobile-operating-system (2011)
13. Jak vypadá Android uvnitř. Android developers [online]. 31. December 2011 [cit. 2014-04-28]. http://www.androidmarket.cz/android/jak-vypada-android-uvnitr-anebco-je-rom-kernel-bootloader-a-dalsi (2011)