Affecting Applications in Android Using Aspects

Ivan Martoš and Valentino Vranić

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava, Slovakia
martos.ivan@gmail.com, vranic@stuba.sk

***Abstract*—Aspect-oriented programming enables to affect existing code without having to make changes to it. By this, changes remain modularized and as such are easily activated or deactivated with a high potential of direct reusability in other applications. However, aspect-oriented programming on mobile platforms is practically unexploited, despite the whole range of opportunities such as usage monitoring, power monitoring, altering sensor values, or altering applications. In this paper, we explore possibilities of using aspect-oriented programming in the AspectJ programming language and demonstrate how to alter the GPS sensor output, phone signals, and deal with the Context class as a gate to the Android services.**

***Keywords*— aspect-oriented programming; Android; mobile device; change; GPS; notifications; Android services; AspectJ**

I. Introduction

Although it may be very useful to separate crosscutting concerns right from the start of the application development, aspect-oriented programming can be used also in a more opportunistic fashion: to affect existing code without having to make changes to it [1, 2]. By this, changes remain modularized and as such are easily activated or deactivated with a high potential of direct reusability in other applications. Refactoring is another opportunistic use of aspect-oriented programming [3]. However, aspect-oriented programming can serve as a basis for software development in a use case driven manner [4, 5], to influence running systems without interruption in association with complex event processing [6, 7], or even to define new programming languages [8].

Despite aspect-oriented programming can be used on modern mobile platforms with only moderate restrictions, its employment there is very limited.

In this paper, we explore possibilities of using aspect-oriented programming in the AspectJ programming language in Android. After explaining how to employ AspectJ in Android (Section II), we demonstrate how to alter the GPS sensor output (Section III), phone signals (Section IV), and deal with the Context class as a gate to the Android services (Section V). Afterwards, we discuss general possibilities of using aspects in mobile devices, which is followed by a brief comparison to related work (Section VI) and conclusions (Section VII).

II. Employing AspectJ in Android

Android as such does not include any kind of aspect-oriented programming and currently there are no tools dedicated specifically to aspect-oriented development under the Android platform. To run applications, Android (up to version 4.4.4) uses a special type of Java virtual machine (VM) called Dalvik, which is designed for the systems with limited memory and processor speed, such as tablets or smartphones. This is in part why employing AspectJ in Android is a bit challenging.

We managed to make a working aspect-oriented programming constellation under Android 4.4.4. To create aspect-oriented code, we used AspectJ 1.7.3 in Eclipse for Java EE (Kepler version) in Windows 7.

Crucial to aspect-oriented programming is the process of so-called *weaving*: the aspect code is *woven*—i.e., put—into the code being affected. With the AspectJ compiler (ajc), this is fully transparent to developer. However, ajc creates Java VM bytecode, and not Dalvik bytecode. What is needed is to make use of weaving provided by ajc while producing Dalvik bytecode at the same time. The key is in manipulating the ajc compilation process by adapting the .project file to transform the Java VM bytecode using the dx tool, which transforms Java VM bytecode into bytecode executable on Dalvik, before the weaving takes place. This is the principle behind the procedure we followed [9], but it had to be adapted a bit to make it working with Android 4.4.4 by excluding the following element from the .project file:

```
<buildCommand>
        <name>org.eclipse.jdt.core.javabuilder
        </name>
        <arguments></arguments>
</buildCommand>
```

With this way of employment of AspectJ in Android it is not only possible to capture execution of methods in a given application, but also to capture application calls to Android API. Aspects affecting only calls to Android API are potentially reusable in other applications.

Inside of advices it is also possible to provide additional application logic depending on current state, preferences, or behavior. However, aspects can't modify permissions that application has declared in its manifest file since it is not possible to add any functionality that does not fulfill requirements declared there.

III. Altering the GPS Sensor Output

Many Android applications use location services API. This API uses data from the GPS sensor and data from the GSM network to obtain the location of the device. The biggest problem of this API is increased energy consumption when obtaining location by GPS. This is something we can affect with aspect-oriented programming.

Often, applications at startup require GPS sensor to be turned on (the maps application, for example). Otherwise, they refuse to provide any functionality or provide limited functionality only. However, the fulfilling of user's objectives may not involve necessity of providing the current location, so the GPS may be turned off. This problem can't be solved by any of publicly available applications, nor API. Using aspects, we can pick out API calls that check whether GPS sensor is turned on and return an adjusted response:

```
boolean around(String provider) : call(boolean
    android.location.LocationManager.isProviderEnabled(..)) &&
    args(provider) {
        //Aditional logic...
        return true;
}
```

This code is a part of an aspect. It includes an *advice* that acts upon a *pointcut*. The pointcut picks out API calls that

check whether GPS sensor is turned on, while the advice code runs instead ("around") these calls always returning true.

This is applicable to other sensors, too. It is possible to intercept calls that check whether the sensor is enabled, while also capturing requested its name (provider). In the given advice it is also possible to provide additional application logic, such as returning true only in a selected timespan.

As it was mentioned above, the biggest disadvantage of the location API is an increased energy consumption. Nowadays the biggest problem in smartphones is their battery life. The most popular way of avoiding energy consumption related to location obtaining, as well as undesired location uncovering, is to use so-called mock locations. They are based on providing fake locations by specialized applications. The user has to download this application and also enable mock locations inside system settings. However, aspects can provide a specialized solution for hiding the current location (or providing the fake one) and saving battery life while providing real location updates at the same time. Moreover, the user doesn't have to use any additional application, nor modify system settings.

The energy consumption problem can be solved by modifying the request for location updates—swapping the GPS as a location provider (which requires lots of energy) for the GSM network provider (has much lower energy consumption, but provides less accurate location). Although the location provided will be less accurate, it will still be the real location of the device and application itself won't be notified about this change. Here's the code:

```
void around(String provider, long timeChange,
        float distChange,
        LocationListener listener) :
        call(void android.location.LocationManager.
                requestLocationUpdates( String, long, float,
                android.location.LocationListener))
        && args(provider, timeChange, distChange, listener) {

        provider = LocationManager.NETWORK_PROVIDER;
        proceed(provider, timeChange, distChange, listener);
}
```

As can be seen above, not only the provider of the location can be changed, but also other parameters of the location update request (see the LocationManager documentation [10] for a detailed parameter description). Modifying these parameters can lead to less energy consumption or more accurate location.

Smartphones users are often concerned over security, which leads to attempts to hide their location. Using aspects, hiding the location can be achieved in two different ways. One way is based on altering response of the get methods (for example getLatitude() and getLongitude()) of the Location class, which is responsible for location representation. By this, every method in the application that requests the location—even the stored one—will be altered. This is useful when the goal is to modify every possible location (for example, in a specified time). The advantage is that the real value of attributes of the Location class won't be changed, since aspects alter only the result of its get methods. Thus, if advices affecting the get methods are disabled, the real location values will be provided.

Another way of location hiding is based on altering the locations that result from location update requests. Fake locations with fake data can be provided. The advantage is that only new location request will be affected. Previously obtained locations will provide real data (they won't be changed). However, if the newly "obtained" location would be stored, later attempts to retrieve the stored location would consequently return the fake one.

IV. Altering Phone Signals

One of the most common ways of signaling events is by using notifications. The notification API requires no special permissions declared in the application manifest, so by using aspects it is possible to add notifications to application at any place. For example, assume an application signals events to the user using vibrations. Assume we need to keep the device absolutely quite, so we want to change vibration by a visual notification. This can be achieved by catching system calls for vibration and modify them using the around advice:

```
void around(): call(void android.os.Vibrator.vibrate(..)) {
        createNotification();
}
```

The vibration (or any other form of signaling) can also be replaced by playing a notification sound. Playing a sound requires no special permission either.

Aspects provide a very comfortable way of altering phone signaling. Each advice that captures a given signaling call can contain specific application logic or rules to provide desired behavior.

V. Capturing Context

Most of the functionality provided by Android is accessible through the Context class. With an aspect, the instance of this class can be captured manipulated consequently changing the application behavior in various ways. For example, we could access the application screen. Since the Activity class, which represents the application screen, extends the Context class, it is only necessary to capture the creation of this class:

```
pointcut onCreatePointCut(Object activity) :
        execution(* Activity+.onCreate(..)) && target(activity);

before(Object activity) :
        onCreatePointCut(activity) {
                //capture the activity object
 }
```

VI. General Possibilities of Using Aspects in Mobile Devices

Usage of aspects in mobile devices can be categorized into four categories: usage and performance monitoring, affecting sensors, and added functionality.

A. Monitoring

People have their phone with them all the time. Times when phones were used only for calling are past. We use phones for calling, as alarm clock, internet, navigation, and many others. Using well defined pointcuts and advices, we can monitor customs and routines of users and analyze them.

As we already mentioned in Section III, one of the biggest problem of today in mobile devices is their battery life. Using aspects it is possible to monitor use of any sensor, use of services, or even the time when the display was turned on. By analyzing these data it is possible to identify which application or sensor consumes the biggest amount of energy and affect its activity to enhance the battery life. It is important to base the monitoring pointcuts on system calls, and not on application calls. Albeit the pointcuts can be defined so to capture custom application calls, the user can use other application for accessing desired functionality and thus bypass what has been changed by aspects.

B. Sensors

Sensors are the reason why smartphones are smart. Functionality of sensors is accessible by the API provided by the system. Using aspects, it is possible to define pointcuts to capture desired API calls and define advices that alter, affect, or even disabled chosen sensors.

C. Added Functionality

Aspects can be used to provide added functionality. Since use cases can be preserved in source code by aspects [4, 5], it is possible to use this logic on Android as an easy way for developing preferences based on alternative flows in use cases or by extension use cases. The main flow of a use case can be expressed by regular object-oriented code or even by an aspect introducing elements into stub classes, to achieve greater flexibility. Extension points can be captured by pointcuts and the logic triggered by preferences can be activated by the corresponding advices inside aspects.

Another field of interests is advertisement. Advertisements often annoy users with undesirable content. Advertisements are mostly based on the GoogleAds application, whose functionality is stored in the `com.google.ads` package. By defining advices to this package it is possible to disable any call to it and thus disable advertisements [11].

VII. Related Work

The concept of affecting applications without making changes to them is generally applicable to software development and maintenance in particular. This has been addressed by an aspect-oriented change realization model [4, 1] that provides several change types and ways of their realization using aspect-oriented programming in AspectJ. The implementation change types (denoted as generally applicable in this model) are applicable to mobile applications, too. For example, changing the GPS location provider corresponds to the Additional Parameter Checking/Modification change type, while affecting the result of the GPS location get methods corresponds to Additional Return Value Checking/Modifications. Others have addressed technical issues in applying aspects to modify compiled code in desktop applications [12, 13].

The approach presented in this paper is limited to compile time weaving and can't be used to achieve run time weaving. Falcone and Currea propose a solution of this problem based on a special application that weaves the provided aspect files into the provided `.apk` files (application installation file) and thus weaves aspects into every existing application [11]. Falcone and Currea also propose another way of modifying device location. They suggest modification of location API by replacing the return values of the `getLastKnownLocation(String provider)` method with custom values. This method returns last known location obtained by the device using the given provider. This alters only one old location while newer locations obtained as a result of the location request callback are not altered. Developers tend to use newest locations (obtained from the callback of the location update request), while the location obtained from the `getLastKnownLocation(String provider)` method is mostly used only at the application startup. However, using our approach, it is possible to modify new and old locations at the same time.

Several other attempts to use aspect-oriented programming in Android have been reported [14, 15, 16]. However, it was not possible to replicate the results of any of these.

VIII. Conclusions and Further Work

In this paper, we explored possibilities of using aspect-oriented programming in the AspectJ programming language and demonstrated how to alter the GPS sensor output, phone signals, and deal with the `Context` class as a gate to the Android services.

The biggest disadvantage in employing aspect-oriented programming in Android is the limitation to compile time weaving of aspects rooted in the build procedure itself [9], since run time weaving would only create Java VM bytecode, and not Dalvik bytecode that can run on Android devices.

Employment of aspects on the newest versions of Android and new ART virtual machine should be performed to test whether it is possible to weave aspects at run time. Next step should be exploring whether the build process of Android applications could be altered, so that aspects would affect selected calls in all applications.

While we targeted mobile devices, the results should be applicable to other devices running Android. Here, Android TV as an Android 5.0 extension is of particular importance. Of course, direct testing is necessary to prove this.

Acknowledgments

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12.

This contribution/publication is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] R. Menkyna and V. Vranić, "Aspect-Oriented Change Realization Based on Multi-Paradigm Design with Feature Modeling," in Proceedings of 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Revised Selected Papers, LNCS 7054, 2009, Krakow, Poland, Springer, 2012.

[2] V. Vranić, R. Menkyna, M. Bebjak, and P. Dolog, “Aspect-Oriented Change Realizations and Their Interaction,” e-Informatica Software Engineering Journal, vol. 3, num. 1, 2009, pp. 43–58.

[3] R. Pipík and I. Polášek, “Semi-Automatic Refactoring to Aspect-Oriented Platform,” in Proceedings of 14th IEEE International Symposium on Computational Intelligence and Informatics, Budapest, IEEE, 2013, pp. 141–145.

[4] I. Jacobson, “Use Cases and Aspects – Working Seamlessly Together,” Journal of Object Technology, vol. 2, num. 4, 2003, pp. 7–28.

[5] I. Jacobson and P.-W. Ng, Aspect-Oriented Software Development with Use Cases. Addison-Wesley, 2004.

[6] J. Lang, M. Jantošovič, I. Polášek, “Re-Usability in Complex Event Pattern Monitoring,” in Proceedings of IEEE 10th Jubilee International Symposium on Aplied Machine Intelligence and Informatics, Herľany, Slovakia, IEEE, 2012, pp. 265–270.

[7] J. Lang, J. Janík, “Reactive Distributed System Modeling Supported by Complex Event Processing,” in Proceedings of ECBS-EERC 2013, 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems, Budapest, Hungary, IEEE CS, 2013, pp. 163–164.

[8] J. Porubän, M. Sabo, J. Kollár, and M. Mernik, “Abstract Syntax Driven Language Development: Defining Language Semantics Through Aspects,” in Proceedings of the International Workshop on Formalization of Modeling Languages (FML ’10), ECOOP 2010, Maribor, Slovenia, ACM, 2010, pp. 6–10.

[9] D. Kramer, “Aspect Oriented Android Development – Tool Integration,” Dean’s Blog, July 18, 2011. https://deansserver.co.uk/~dean/2011/07/18/aspect-oriented-android-development-tool-integration/

[10] Android Developers, http://developer.android.com/

[11] Y. Falcone and S. Currea, “Weave Droid: Aspect-Oriented Programming on Android Devices,” in Proceedings of 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012). IEEE, 2012, pp. 350–353.

[12] I. Bluemke and K. Billewicz, “Aspect Modification of an EAR Application,” in Advanced Techniques in Computing Sciences and Software Engineering, Khaled Elleithy, Ed., Springer, 2010.

[13] I. Bluemke and K. Billewicz, “Aspects in the Maintenance of Complied Program,” in Proceedings of 3rd International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2008, Szklarska Poręba, Poland, IEEE, 2008.

[14] C. Anjos, “Aspect Oriented Programming (AOP) – A Case Study in Android, Slideshare. http://www.slideshare.net/anjosc/aop-codebits2011

[15] “AspectJ in Android: pointcut call(* Activity.onCreate(..)) doesn't pick out Activity.onCreate() calls,” Stackoverflow. http://stackoverflow.com/questions/6356375/aspectj-in-android-pointcut-call-activity-oncreate-doesnt-pick-out-acti

[16] F. Cejas, “Aspect Oriented Programming in Android.” http://fernandocejas.com/2014/08/03/aspect-oriented-programming-in-android/