Weave Droid: Aspect-Oriented Programming on Android Devices

Fully Embedded or in the Cloud

Yliès Falcone
UJF - University of Grenoble I, France
Laboratoire d'Informatique de Grenoble, France
Ylies.Falcone@ujf-grenoble.fr

Sebastian Currea
UJF - University of Grenoble I, France
Laboratoire d'Informatique de Grenoble, France
Sebastian.Currea@imag.fr

ABSTRACT

Weave Droid is an Android application that makes Aspect-Oriented Programming (AOP) on Android devices possible and user-friendly. It allows to retrieve applications and aspects and weave them together in several ways. Applications and aspects can be loaded from Google Play, personal repositories, or the local memory of a device. Then, two complementary weaving modes are provided: local or remote, using the embedded aspect compiler or the compiler in the cloud, respectively. This provides flexibility and preserves the mobility of the target devices. Weave Droid opens a world of possible applications, not only by benefiting from the already existing uses of AOP on standard machines, but also by the various uses related to the mobile devices. Effectiveness of Weave Droid is demonstrated by weaving aspects with off-the-shelf applications from Google Play.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering, Software Librairies*

General Terms

Design, Experimentation, Languages, Measurement

Keywords

Android, Mobile Devices, Aspect-Oriented Programming, Embedded Systems, Weaving, Cloud Computing

1. CONTEXT AND MOTIVATIONS

The rise of Android. The mobile market is exploding. Over the last five years, the number of user-friendly mobile devices has grown dramatically. As an operating system for mobile devices, Android enjoys an increasing popularity. The market-share of Android cellphones has just come over 50% at the end of 1Q2012 [1]. Moreover, Moores's and Koomey's laws for mobiles announce that the processing speed and the energy efficiency double after every 18-month period, respectively. Being based on a generic Linux kernel, Android runs on various mobile devices from smartphones to tablets. Being based on the Java programming language, Android applications enjoy desirable features for developers. Consequently, a large number of applications is available both inside and outside Google Play.

Aspect-Oriented Programming. An established and active software development methodology is aspect-oriented programming (AOP) [3]. AOP allows one to implement crosscutting concerns through aspects. Aspects rely on three concepts: joinpoints, pointcuts, and advices. A jointpoint is an identifiable point in the execution of a target program. A pointcut selects a set of jointpoints. Then, an advice is a piece of code associated to some pointcuts. They augment or alter the execution of the program when the execution reaches a jointpoint selected by the pointcuts. Many compilers are available to implement AOP. Among them, AspectJ [4] is certainly the most successful one due to its user-friendly language and the performance of both the weaving process and the resulting weaved application (compared to the initial application).

Since its introduction by Kiczales in [3], AOP has been an asset in many software development projects, a supporting technology for many research areas, and a research subject on its own. It has been used in several techniques for software reliability and verification such as runtime verification and testing. Moreover, AOP can be used to augment software with logging capabilities, profile software, check pre and post conditions, deploy security policies, etc. Finally, many research efforts are conducted to give a formal semantics to this popular software development methodology.

Motivations and challenges. Using AOP with Android applications is appealing. To develop applications, an Android developer uses a well identified set of methods whose purposes are precisely described in the SDK. For instance, methods in the package `com.google. ads` are used to display ads. Consequently, it becomes very easy to write aspects implementing abstract functionalities that work with any Android application on any device. For instance, in Section 3 we show how to write an ad-blocker for any Android application (that will necessarily go through the mentioned interface) using a very simple aspect.

However, AOP is not possible on Android devices, or, at least, not in a satisfactory manner. Indeed, the necessary constraints to use AOP on Android applications seriously hinder the mobility of the device and restrain the possi-

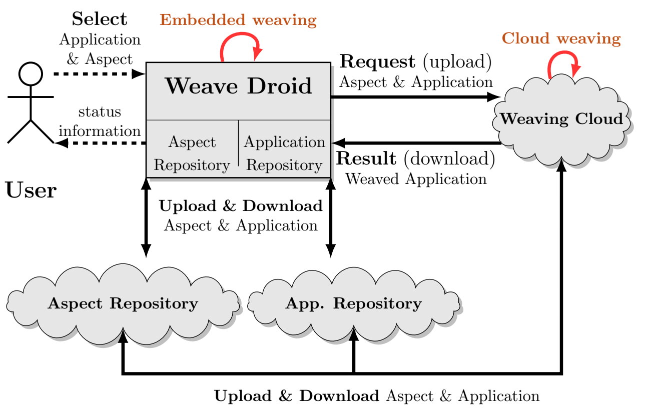


Figure 1: Weave Droid context

ble usages of AOP. Currently, to weave an aspect with an Android application, one has to use an auxiliary computer plugged to the target device. Weaving of the aspect is done in the auxiliary computer and then the weaved application is transferred to the device. What is worse, weaving is possible only for self-developed applications (using the binary or source files), as opposed to existing Android application (downloaded for instance from Google Play). This is due to the incompatibility of existing aspect-compilers with the Android `.apk` files (Android target binary file format).

Another challenge is that computation resources may become an issue on embedded devices such as smartphones or tablets. On the one hand, aspect weaving can be arbitrarily expensive memory-wise. Depending on the hardware device, the Dalvik virtual machine (VM)[1] enforces some limitation on the available memory. For instance, in the best case, on a state-of-the art tablet, the available memory is limited to 288 MB. Moreover, on some older versions of Android (before v3.1), the developer was not authorized to use all the available physical memory. On the other hand, the processing speed on embedded device is generally much slower than the one available on standard machines.

2. AN OVERVIEW OF Weave Droid

Weave Droid is an Android application that allows user-friendly and flexible AOP on Android devices. Weave Droid takes any existing Android application plus an aspect as input and weaves them together in a transparent way for the user. Applications can be downloaded from personal repositories or Google Play. As we shall see, weaving can be done in two complementary fashions: inside the device (embed mode) or on a dedicated weaving cloud (cloud mode).

Weave Droid is compatible with Android HoneyComb 3.1 or higher. It can be freely downloaded and tested from [2] or Google Play. In this paper, we provide a description of its features and some insights about its internal architecture.[2]

Weave Droid is a Java application of approximately 1200 LLOC built above several third-party tools. How Weave Droid interacts with the user is represented in Figure 1. Using the interface shown in Figure 2, the user selects an Android application directly as an `.apk` file plus an aspect as an `.aj` file. These entities can be chosen either from local or remote repositories (whose IP address can be configured). Then, the user chooses to weave the aspect within the application either locally with the embedded weaver or remotely within the dedicated weaving cloud (whose IP address can be configured). The two repositories and the weaving cloud can be located on two different servers. If embedded weaving is selected, several steps and additional third-party tools are involved. This is due to the initial incompatibility between formats as announced in Section 1. If cloud weaving is selected, the application is transferred using FTP to the weaving cloud that embeds a version of the aspect compiler as a Web service interacting with Weave Droid. Both modes use the AspectJ compiler since it is the most popular and efficient aspect compiler currently available. Two weaving modes are proposed to the user since these modes are used in different situations and have different performance (see also Section 4). Embedded weaving mode is suitable when the device is not connected to the internet or when the user wants to locally modify a downloaded application (e.g., according to some local security policy). Cloud weaving is suitable when an appropriate internet connection is available and/or that the resources required for weaving exceeds the available resources on the target device. Ignoring application and aspect transfer, cloud weaving performs faster than embedded weaving. After weaving according to one of the $2 \times 2 \times 2$ possible combinations, the user is notified about weaving success or failure. If weaving succeeds, the woven application is available locally and ready to be executed.

[1]The Dalvik virtual machine is the software in the operating system that run the applications on Android devices.

[2]A detailed technical manual is currently being finalized and will be available at [2].

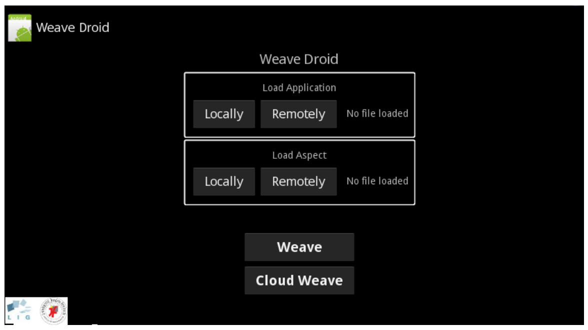


Figure 2: Weave Droid main screen

3. EXAMPLES

Weaving applications on embedded devices affords a world of possible uses. This section presents some of them using *off-the-shelf* Android applications downloaded from Google Play. In addition to the applications mentioned in this section, the aspects were also successfully woven in several other applications to assess their effectiveness.

Note that the methods used in aspects were selected by a simple analysis of Android SDK (without any knowledge about the internal code of the applications).[3]

3.1 Simple Ad Blocker

Tic-Tac-Toe is a classic game that uses the device's Internet connection to display banners in the bottom of the screen (see Figure 3(a)).

Listing 1: Aspect blocking banners

```
aspect BannerAspect  {
Object around() : execution(* com.google.ads..*(..))
    && !within(BannerAspect) {
        return null;
    }
}
```

[3]A selection of interesting packages and methods (with their descriptions) is available for users of Weave Droid at [2].

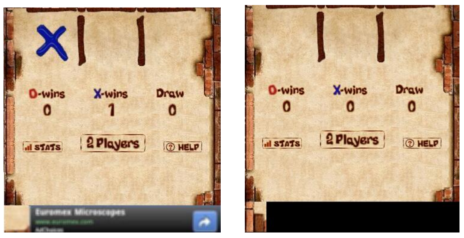
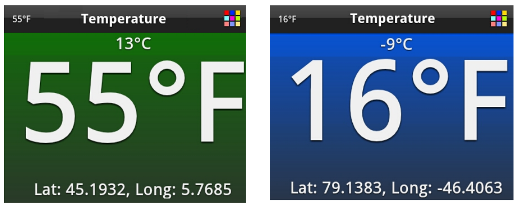

(a) With Banner (b) Without Banner

Figure 3: Tic-Tac-Toe application



(a) Real coordinates (b) Greenland coordinates

Figure 4: Temperature application

With the simple aspect shown in Listing 1 and Weave Droid, we can obtain an ad blocker. The aspect determines whether Tic-Tac-Toe invokes a Google banner by catching any call to the methods in the `com.google.ads` package. Upon such a call, the aspect blocks the banner using the around advice that cancels the call (by not calling `proceed()` and returning `null`). The woven application does not display the banner as shown in Figure 3(b).

3.2 Modifying the Device Location

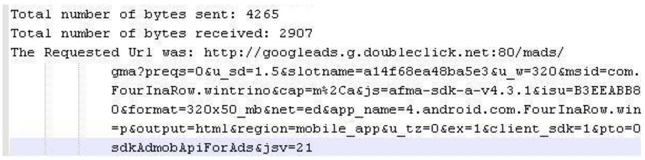
Temperature is an application that shows the current local outdoor temperature, measured by a nearby weather station. Temperature uses the current device location (see Figure 4(a)). The aspect in Listing 2, and more specifically the pointcut `location`, determines whether the application calls the method `getLastKnownLocation` of the class `LocationManager` in the package `android.location` to determine the location. Upon such a call, the advice changes the latitude and longitude coordinates to some point in Greenland. After weaving the application with the aspect, the woven application shows different coordinates and temperature as expected (see Figure 4(b)).

Listing 2: Change Location Aspect

```
aspect ChangeLocationAspect{
 // Pointcut to Android location method.
 pointcut location(String provider):call(* android.
    location.LocationManager.getLastKnownLocation(..))
  && args(provider) && !within(ChangeLocationAspect);
 // Advice to change the device location.
 Location around(String provider):location(provider){
...
  Location location = new Location(provider);
  // New latitude and longitude values in Greenland.
     location.setLatitude(79.13826);
     location.setLongitude(-46.40625);
  return location;
...
```

3.3 URL and Bytes Consumed

We downloaded several games: Four in Line (G1), Tic-Tac-Toe (G2), Matchup cards (G3), Bubble pop (G4), Memory pairs (G5), and Melimots (G6). These games display advertisements in the form of banners those content is retrieved by connecting to internet. Our purpose was to disclose the URL they connect to and measure the number of bytes exchanged with the remote servers. Each application behaves as follows: it loads an URL and consumes a certain number of bytes encoding the advertisement. Listing 3 shows an extract of the aspect that reveals and stores in a text file the information about the URL and the bytes consumed. Figure 5 shows an example of file created by the aspect.



```
Total number of bytes sent: 4265
Total number of bytes received: 2907
The Requested Url was: http://googleads.g.doubleclick.net:80/mads/
            gma?preqs=0&u_sd=1.5&slotname=a14f68ea48ba5e3&u_w=320&msid=com.
            FourInaRow.wintrino&cap=m%2Ca&js=afma-sdk-a-v4.3.1&isu=B3EEABB8
            0&format=320x50_mb&net=ed&app_name=4.android.com.FourInaRow.win
            =p&output=html&region=mobile_app&u_tz=0&ex=1&client_sdk=1&pto=0
            sdkAdmobApiForAds&jsv=21
```

Figure 5: File UrlBytes.txt created by the aspect

Traffic	G1	G2	G3	G4	G5	G6
Sent	3,559	15,453	16,223	2,426	16,197	14,557
Received	1,205	2,364	2,466	1,123	3,194	2,283

The above table shows how traffic statistics about the games can be gathered. In average, to open its banners an application sends 11,402.5 bytes and receives 2,105.8 bytes.

4. PERFORMANCE EVALUATION

Due to the embedded nature of the targeted systems, some performance limitations arise in terms of time necessary to complete the weave process and the amount of memory that the Dalvik Virtual Machine allocates to the process. Two hardware configurations were used to assess weaving performance. First, we used a Samsung Galaxy Tab 10.1 (Android device for embedded weaving) with 1 GB RAM, 1 Ghz dual core NVIDIA® Tegra™ 2 processor, 288 MB maximum Dalvik VM heap size, running on Android Honeycomb 3.1. Second, we used a Laptop Dell Latitude D620 (the weaving cloud) with 2 GB of RAM, Intel Core 2 T7200 2.00 Ghz, running Windows XP SP3. Based on this latest configuration we set up two wireless-network configurations differing by their transfer time. In the first configuration (referred to as cloud1), the transfer rate from the Android device are 14,930 kbps for download and 15,892 kbps for upload. In the second configuration (referred to as cloud2), the transfer rate from the Android device are 1,615 kbps for download and 1,068 kbps for upload.

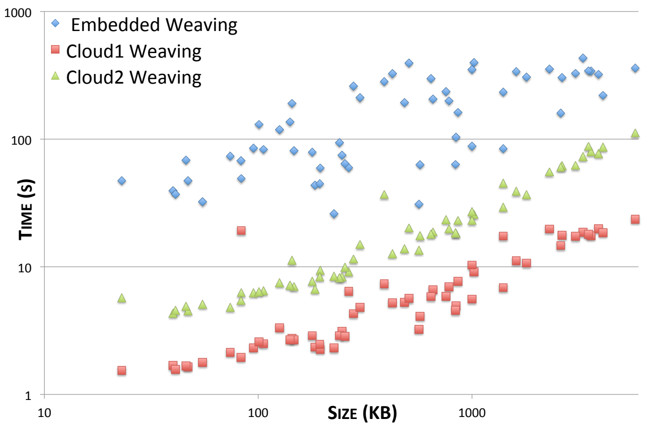


Figure 6: Comparing weaving performances

Figure 6 compares performances of the three weaving modes using 57 free games retrieved from Google Play. The x-axis indicates the size of the application in KB, while the y-axis

Listing 3: URL And Bytes Aspect

```
aspect UrlBytesAspect {
 // Pointcut to the method that loads an URL
 pointcut pageName(String page) :  (execution(* android.webkit.WebView.loadUrl(..))
                    || execution(* android.webkit.WebView.loadDataWithBaseURL(..)) )
                    && args(page,..) && !within(UrlBytesAspect);
 ...
 startRX = android.net.TrafficStats.getTotalRxBytes(); // variable to count received bytes
 startTX = android.net.TrafficStats.getTotalTxBytes(); // variable to count transferred (sent) bytes
 ...
 // Advice that stores the Bytes consumed and the Url in the file UrlBytes.txt
 after(String page): pageName(page) {
   long totalRx = android.net.TrafficStats.getTotalTxBytes()- startTX;
   long totalTx = android.net.TrafficStats.getTotalRxBytes() - startRX;
   String data = "Total number of bytes sent: " + totalTx + "\n Total number of bytes received: " + totalRx
     + "\n The Requested Url was: " + page + "\n";
   FileOutputStream fOut = contextWrapper.openFileOutput("UrlBytes.txt", Context.MODE_APPEND);
   OutputStreamWriter osw = new OutputStreamWriter(fOut);
   osw.append(data);
 ... }
```

indicates the weaving time in seconds. Note the logarithmic scale used on both axis. All applications were woven with the ad-blocker aspect (Listing 1) in the three different weaving modes (embedded, cloud1, and cloud2 weaving).

In all cases, cloud weaving is faster than embedded weaving.[4] As the size of the application augments, using cloud weaving becomes more and more preferable as it almost does not depend on the size of the input application.

Note that, for all woven applications described in the previous section, no degradation of runtime performance was noticed compared to the original application.

Memory limitations in embedded weaving. Some memory limitation may occur when performing the weaving process, more precisely when the Dalvik VM tries to allocate more memory than available on the heap. Should it happen, Weave Droid throws an out of memory exception and the weaving process aborts. For instance, on embedded weaving of some applications of approximately 15 MB of size, the weaving process requires more than 288 MB of memory. In this case, cloud weaving is mandatory.

5. CONCLUSIONS AND PERSPECTIVES

Conclusions. To the best of our knowledge, Weave Droid is the first tool to propose (embedded and in the cloud) aspect-oriented programming (AOP) for mobile devices running the Android operating system. Weave Droid proposes flexible and user-friendly aspect weaving. It allows to retrieve aspects and applications from various repositories. Embedded and cloud weaving are supported in order to adapt to the context of the mobile device. Moreover, existing applications of AOP to program development can be reused in the context of mobile application development (e.g., logging, profiling, etc).

Future work and developments. In the near future, we plan to make Weave Droid open-source and available for Android developers to be used as a third-party tool and modified. Moreover, several extensions are currently in the roadmap of Weave Droid. For instance, we plan to 1) support customized invocation to the AspectJ compiler and support a larger set of its options, 2) support more aspect compilers, 3) support additional protocols for transferring aspects and applications between the device, the repositories, and the weaving cloud.

[4]Measured timings for cloud weaving comprise transfer time.

Some Benefits and Perspectives. More elaborated uses of Weave Droid can be considered (e.g., commercial ones). For instance, opposite to the ad blocker, one could imagine aspect-based services gathering information about the user behavior.

Our longer-term research purpose is to use Weave Droid as an essential component in an embedded validation framework such as runtime verification (RV) or testing. For instance, most of the existing RV tools use AOP as a supporting technology to instrument a monitored program and observe the relevant events in its execution. Consequently, weaving modes of Android opens new challenges and opportunities for the field of RV. Besides the opportunity of transferring existing RV frameworks to target mobile devices, the weaving modes of Weave Droid allows to introduce some dynamicity in the verification process where monitored applications and requirements change as the target mobile device evolves in space and time.

Additional research opportunities are opened by Weave Droid in the context of security. For instance, Weave Droid can be used to determine the non-special permissions used by an application. No information about these permission is available at installation time. Second, Weave Droid can be a base brick of runtime reference monitors. As another example, one could imagine implementing a security tool that enforce a local policy by weaving some aspects implementing the policy with any downloaded application.

6. ACKNOWLEDGMENTS

The authors would like to thank Mohamad Jaber, Yves Ledru and the anonymous reviewers for their comments.

7. REFERENCES

[1] comScore Reports. U.S. mobile subscriber market share, april 2012. http://www.comscore.com.

[2] S. Currea and Y. Falcone. Weave Droid, May 2012. Available at http://droid.forge.imag.fr.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[4] Xerox Corporation. Aspectj programming guide. http://www.eclipse.org/aspectj/.