Demo: API Virtualization for Platform Openness in Android

Taeyeon Ki, Alexander Simeonov, Karthik Dantu, Steven Y. Ko, Lukasz Ziarek
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
{tki, agsimeon, kdantu, stevko, lziarek}@buffalo.edu

Extended Abstract

We propose a novel technique called *API virtualization* to enable open innovation in Android. API virtualization inserts a shim layer between the Android platform layer and the app layer as shown in Figure 1, which can intercept any and every platform API call made by an app . In addition, API virtualization allows third-party developers to inject custom code, so that they can modify, reimplement, or customize existing Android APIs. This is achieved by (i) injecting a *wrapper class* for each platform API class that a third-party developer wants to replace, and (ii) rewriting the binary of an app so that the app code uses wrapper classes instead of platform API classes.

Our API virtualization is motivated by the lack of openness in mobile systems at the platform level. For example, Android is known to be an open platform since the source code is open; third-party developers easily access and modify the source. However, when it comes to deploying their platform-level modifications, there is a stiff barrier. Only Google and other mobile vendors such as Samsung, LG, etc. have the privilege to distribute platform modifications at a large scale. In other words, there are only a select few players who can control the innovation on Android.

Our API virtualization aims to mitigate this lack of openness by allowing third-party developers to modify the behavior of platform API classes without touching any part of the platform. By injecting a shim layer into an app that executes custom code for an API call, a third-party developer can modify the behavior of an API call and deploy it through app deployment mechanisms. For example, standard app deployment channels such as Google Play can be used.

To faithfully show the benefits of API virtualization, we have implemented a prototype of API virtualization called *Reptor* on Android. To correctly implement Reptor, we have carefully addressed many practical challenges. These challenges mainly come from Android's event-based programming model and its development language, Java, which supports multiple features such as polymorphism and sub-typing. Previous systems such as SIF [2] and RetroSkeleton [1] have focused on specific app instrumentation mechanisms for Android, but our focus is on addressing a unique set of challenges that API virtualization brings in order to *correctly* and *completely* handle all features of Android and Java.

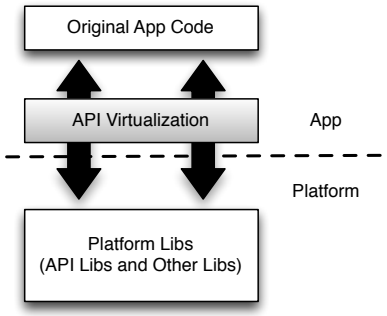

Figure 1: API Virtualization

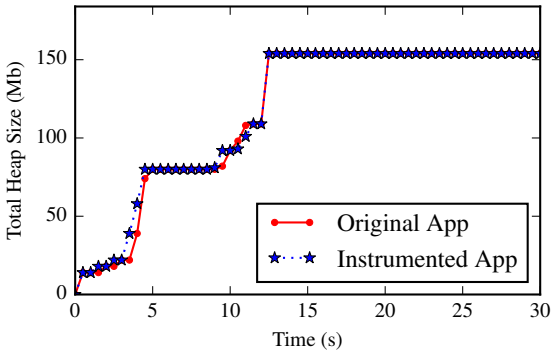

Figure 2: Heap Usage of the non-instrumented and instrumented Temple Run

We have evaluated Reptor with a real app, Temple Run, downloaded from Google Play Store. We have played Temple Run for 90s. We use Android tools to record heap allocation sizes every 0.3s while the app is running. We cut x-axis at 30s since heap size is the same after 12s. Figure 2 shows that Reptor has no noticeable impact on performance of Temple Run. The heap usage for the instrumented version is very similar to the usage for the non-instrumented version of the app.

1. REFERENCES

[1] B. Davis and H. Chen. RetroSkeleton: Retrofitting Android Apps. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, 2013.

[2] S. Hao, D. Li, W. G. Halfond, and R. Govindan. SIF: A Selective Instrumentation Framework for Mobile Applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, 2013.

MobiSys'16 Companion June 25-30, 2016, Singapore, Singapore

ACM ISBN 978-1-4503-4416-6/16/06.
DOI: http://dx.doi.org/10.1145/2938559.2948646