User scripting on Android using BladeDroid

Ravi Bhoraskar[☺*], Dominic Langenegger[†*], Pingyang He[☺*], and Michael D. Ernst[☺]
[☺]University of Washington {bhora, pingyh, mernst}@cs.washington.edu
[†]ETH Zurich, Switzerland dominicl@ethz.ch

Abstract

User scripts allow users to customize their app use experience. In web apps, for instance, a user may use Greasemonkey [1] scripts and browser extensions to customize the layout of a page, automate repeated tasks, block ads, and so on. We bring user-side programmability to mobile applications. Using our tool, BladeDroid, users can write scripts that enable them to customize their experience within Android apps.

We motivate our work using three example applications that can be built using BladeDroid — an Ad Blocker, a Social Media plugin, and a Runtime Testing harness. We describe the design and implementation of BladeDroid, and propose evaluation metrics to measure its usability, robustness and performance.

1 Motivation

Mobile apps are central to the smartphone experience. However, outside of what the developer provides, the user has no control on customizing the behaviour of the app. The only feasible way today to modify the behaviour of an app is through bytecode rewriting [2]. This technique, however, remains largely impractical, since it requires recompiling an app for every new feature to be added, removed or modified. BladeDroid does away with this effort — in our model, a user may install, uninstall and modify arbitrary scripts, without needing to recompile or even reinstall the app. Additionally, a script is not bound to a specific app, and can be reused to add the same feature to multiple apps.

This kind of user-side scripting enables a variety of use-cases. We list a few examples here to motivate our work.

Ad Blocker: An ad blocker searches for an Ad-View within every page, and makes it invisible.

Social Media Plugin: A "Like" or "Share" button is on every page to share the current app context to a social media platform, for example to say "I like level 10 of Angry Birds". Clicking the like/share button would scrape the interesting content from the page, the name of the app, and post it to a user's feed.

Runtime Test Harness: Runtime app-state exploration is a well-studied technique in the mobile testing community [3]. User-scripting would ease this process, by using a script that systematically invokes interactions with all UI components in a page.

* Student Author

2 Design and Implementation

In order to enable an app to run user-scripts (which we call "Blades"), BladeDroid rewrites an application to add a BladeLoader in every page. The BladeLoader uses dynamic class loading to search through all installed Blades, and then pass the current page object to the appropriate ones. Installing new Blades is a simple matter of copying a `jar` file to an appropriate location on the phone filesystem. Since the scripts are loaded at runtime, the app need not be recompiled (or even reinstalled) in order to add, disable or modify a script.

A Blade is a Java class written against the `Blade` interface, which contains methods hooked to each of the Android Activity Lifecycle methods, allowing the Blade-writer to exert control over the app at all interesting points.

3 Status and Next Steps

We have implemented BladeDroid, including the `Blade` interface and the `BladeLoader` instrumentation. Currently, we are developing Blades to implement the applications described in the motivation above. At NSDI, we shall demonstrate a few working Blades, running on real apps from the Google Play store.

In order to evaluate BladeDroid, we wish to measure the performance, robustness, and usability of the system. Specifically, we want to compare the overheads of an BladeDroid-instrumented app against an unmodified one, or a manually instrumented one. Robustness shall be tested by using BladeDroid against real Android apps from the store. This will evaluate the reliability of the entire toolchain, including frameworks like Soot [4] that we use for bytecode rewriting. We shall evaluate the usability of the BladeDroid interface by writing several scripts, from different domains, and measuring the ease in the modification process for a developer, compared to manual binary rewriting.

We will use the limitations revealed during the evaluation of the current version to improve the BladeDroid architecture and implementation.

4 References

[1] GreaseMonkey — Official Site. http://www.greasespot.net.
[2] S. Hao, D. Li, W. G. Halfond, and R. Govindan. SIF: A Selective Instrumentation Framework for Mobile Applications. MobiSys '13, June 2013.
[3] K. Lee, J. Flinn, T. Giuli, B. Noble, and C. Peplin. AMC: Verifying User Interface Properties for Vehicular Applications. MobiSys '13, June 2013.
[4] R. Valle-Rai, P. Co, E. Gagnon, L. J. Hendren, P. Lam, and V. Sundaresan. Soot - a Java bytecode optimization framework. IBM centre for advanced studies conference, 1999.