On-Device Bug Reporting for Android Applications

Kevin Moran, Richard Bonett, Carlos Bernal-Cárdenas, Brendan Otten, Daniel Park & Denys Poshyvanyk
College of William & Mary
Department of Computer Science
Williamsburg, VA 23187-8795, USA
{kpmoran, rfbonett, cebernal, bwotten, dhpark01, denys}@cs.wm.edu

ABSTRACT

Bugs that surface in mobile applications can be difficult to reproduce and fix due to several confounding factors including the highly GUI-driven nature of mobile apps, varying contextual states, differing platform versions and device fragmentation. It is clear that developers need support in the form of automated tools that allow for more precise reporting of application defects in order to facilitate more efficient and effective bug fixes. In this paper, we present a tool aimed at supporting application testers and developers in the process of **O**n-**D**evice **B**ug **R**eporting. Our tool, called **ODBR**, leverages the uiautomator framework and low-level event stream capture to offer support for recording and replaying a series of input gesture and sensor events that describe a bug in an Android application.

1. INTRODUCTION

In order to aid in the often difficult process of reproducing and fixing bugs related to mobile apps, in this paper we present a tool that enables developers to effectively carry out the process of **O**n-**D**evice **B**ug **R**eporting. Our prototype **ODBR** app is capable of running on a standalone physical or virtual Android device and recording precise user touch interactions coupled to specific GUI-components as well as sensor data streams for a target app. Then, this information is sent to a Java web application that displays a detailed, actionable bug report including screenshots, and series of user events that can be replayed on a target device allowing developers and testers to debug the app.

2. BACKGROUND AND RELATED WORK

Given that effective bug and error reporting is a problem faced by nearly all Android developers, there are several existing commercial and research-oriented tools geared toward improving the reporting process. Several commercial bug reporting and analytics platforms are available to support Android apps including Airbrake [1], TestFairy [4], Appsee [2], and Bug Clipper[3]. Typically, these solutions consist of an external library that a developer can include when writing their app. This library then enables the collection of information, such as device logs, performance information, screenshots and video recordings. To the best of the authors' knowledge, none of these services is capable of collecting fine-grained user input and GUI information capable of being replayed on a target device as **ODBR** does. This precise collection methodology could capture critical information that a developer needs to successfully reproduce and fix an incoming bug report.

Two recent automated input generation tools for mobile devices, CrashScope [10, 11] and Sapienz [7] are capable of producing detailed *crash reports*. CrashScope uses a combination of static and dynamic analysis to perform targeted testing of application features according several touch, text, and contextual feature input generation strategies and is capable of producing an `html` crash report with images, and a replayable script. Sapienz formulates the process of automated input generation as a multi-objective search problem, and is capable of producing videos and replayable scripts. While these automated tools are able to provide relatively robust *crash-reports* for apps they test, they are plagued by the oracle problem, in that while they can recognize and report when an app has crashed, they may miss user facing bugs that are more subtle. Additionally, FUSION [8, 9] is an off-device bug reporting system for Android applications that leverages static and dynamic analysis performed before the reporting process to help guide users through reporting detailed reproduction steps for an application. **ODBR** is complimentary to FUSION in that it provides another avenue for developers and testers to create detailed bug reports with minimal effort. Finally, previous Record & Replay approaches for Android apps have been proposed including RERAN [5], VALERA [6], and MobiPlay [12]. While each of these solutions offers the ability to record user actions and replay them later, none of the tools offers the ability to record user actions on a standalone device without a connection to a host machine or server. **ODBR** leverages the highly efficient and accurate event stream recording introduced by RERAN and VALERA and combines this with detailed GUI-hierarchy information collected via the `uiautomator` tool to enable the capture of precise information to aid in the debugging process.

3. THE ODBR BUG REPORTING TOOL

The ODBR Workflow: The overall architecture of our **ODBR** tool is illustrated in Figure 1. The entry point for a tester or developer using this tool is the *ODBR Android Application*, which is available as an open source project accessible from our tool's webpage[1] along with further infor-

MobileSoft'17 May 22-23, Buenos Aires, Argentina

ACM ISBN 123-4567-24-567/08/06…$15.00
DOI: 10.475/123_4

[1] https://www.android-dev-tools.com/odbr

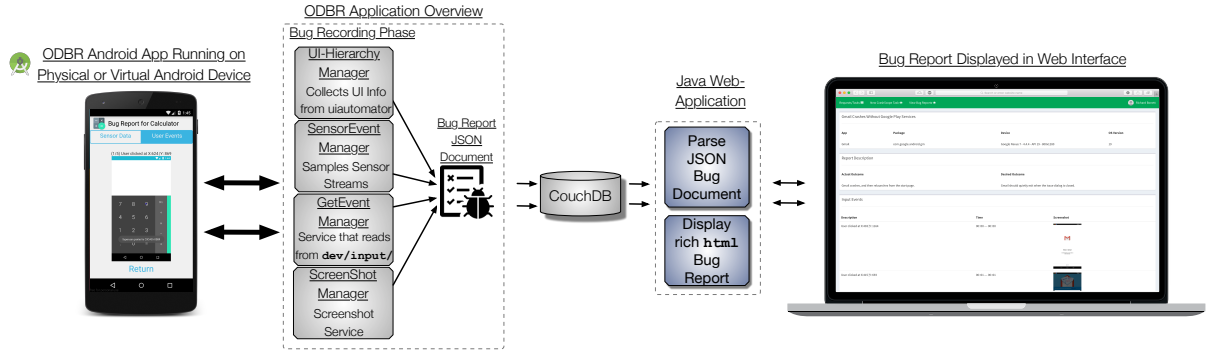

Figure 1: On-Device Bug Reporting Tool Architecture

mation about the project and demo videos. To use our tool, a reporter simply needs to install the **ODBR** app on their target virtual or physical device[2] select the app for which they wish to report a bug from the list of installed applications, hit the record button and perform the actions on the device that manifest a bug. After the **ODBR** app (which is running in the background) detects that no touch-based inputs have been entered after a certain period of time, it asks the user if they have finished the report or if they wish to continue. Once the user has finished, they will have a chance to enter additional information about the bug, including a natural language description of the expected and actual behavior, and a title for the report. Additionally, the user can view the screenshots and sensor data traces from the device and even replay the touch events to ensure they were properly captured. This replay feature relies on a custom written Java implementation of the `sendevent` utility[3]. Once the report has been created, a `BugReport` is translated to a `json` document where it is sent over the web to a `couchDB` server instance. A Java web-application then reads the bug report information from the `json` file and converts the information into a fully expressive html report. From the java web app, a developer can view the reproduction steps of the report, complete with screenshots and action descriptions, as well as download a replayable script that will reproduce the actions via `adb` or `sendevent` commands.

ODBR App components: The ODBR Application has 4 major components that aid in the collection of information during the bug reporting process. These include: (i) the ***GetEvent Manager***, (ii) the ***UI-Hierarchy Manager***, (iii) the ***Sensor Event Manager*** and (iv) the ***Screenshot Manager***. The ***GetEvent Manager*** is responsible for precisely and efficiently reading in the user event stream. To accomplish this, during the reporting process the app creates threads that read from the underlying Linux input event streams at `/dev/input/...` These input events provide highly detailed information about the user's interaction with the phone's physical devices, including (where applicable) the touch screen, power button, keyboard, etc. This information is identical to what is gathered using the Android `getevent` utility as used by RERAN [5]. Next, these low-level input event streams are parsed into higher level user interactions (e.g. *swipe from (a,b) to (c,d)*). The low-level input events are retained to support precise analysis and replayability, while the higher level interactions are used to summarize the report in natural language. Whenever the ***GetEvent Manager*** detects the user is taking a new action, it notifies the applicable managers to take a screenshot and dump of the UI-hierarchy, associating these with the new interaction. The ***UI-Hierarchy Manager*** interfaces with the Android `uiautomator` framework to capture dumps of the Android view hierarchy in `ui-dump.xml` files for each new user action. Because these dump files contain information about the screen location of each UI-component, we can use the event information obtained from the previous component to precisely infer the UI-component that the user interacted with at each step in the bug reporting process. This component also extracts attributes of the various components on the screen including information such as the type (e.g., button, spinner) and whether the component is clickable. The ***Sensor Event Manager*** is responsible for efficiently sampling the sensor input streams (e.g., accelerometer, GPS) during the bug reporting process. This component accomplishes this by registering `SensorEventListener` instances for each sensor which sample the sensor values at appropriate rates. Finally, the ***Screenshot Manager*** is responsible for capturing an image of the screen for each new user interaction using the `screencap` utility included in Android.

[2]Our tool's app currently requires a rooted target device, a requirement for accessing the `/dev/input` event stream

[3]https://goo.gl/EEUSfu

4. REFERENCES

[1] Airbrake tool website https://airbrake.io.

[2] Appsee tool website https://www.appsee.com.

[3] Bug clipper tool website http://bugclipper.com.

[4] Testfairy tool website https://testfairy.com.

[5] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing- and touch-sensitive record and replay for android. ICSE '13, pages 72–81, 2013.

[6] Y. Hu, T. Azim, and I. Neamtiu. Versatile yet lightweight record-and-replay for android. OOPSLA 15, pages 349–366.

[7] K. Mao, M. Harman, and Y. Jia. Sapienz: Multi-objective automated testing for android applications. ISSTA 2016, pages 94–105.

[8] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk. Auto-completing bug reports for android applications. FSE 2015, Bergamo, Italy.

[9] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk. Fusion: A tool for facilitating and augmenting android bug reporting. In *ICSE'16*, May.

[10] K. Moran, M. Linares-Vasquez, C. Bernal-Cardenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *ICST'16*, pages 33–44.

[11] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Crashscope: A practical tool for automated testing of android applications. ICSE 2017, page to appear.

[12] Z. Qin, Y. Tang, E. Novak, and Q. Li. Mobiplay: A remote execution based record-and-replay tool for mobile applications. ICSE '16, pages 571–582.