The Report committee for Inaqui Raynaud Delgado

Certifies that this is the approved version of the following report:

A Deterministic, Nonintrusive Utility for Efficiently Testing Transient State Restoration on Android Applications

APPROVED BY

SUPERVISING COMMITTEE:

Sarfraz Khurshid, Supervisor

Adnan Aziz

A Deterministic, Nonintrusive Utility for Efficiently Testing Transient State Restoration on Android Applications

by

Inaqui Raynaud Delgado, B.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2015

This report is dedicated to my parents, Jose and Lorena, whose selflessness, hard work, and constant support allowed me to pursue higher learning and realize many of my lifelong dreams. It is also dedicated to my sister, Tatyana, who has endured the misfortune of proofreading my writing assignments since grade school.

Acknowledgments

I would like to thank my supervisor, Dr. Sarfraz Khurshid, for providing me with great guidance and feedback on this report. I would also like to thank Dr. Adnan Aziz for always encouraging me to pursue my own interests and ideas.

A Deterministic, Nonintrusive Utility for Efficiently Testing Transient State Restoration on Android Applications

Inaqui Raynaud Delgado, M.S.E.
The University of Texas at Austin, 2015

Supervisor: Sarfraz Khurshid

When a user interacts with an Android application, non-persistent data or transient state is created. Transient state may capture user input, data from resources on a device, or data retrieved from a network. Typically, Android applications retain their transient state throughout their entire life cycle. However, in some cases, Android may trigger the destruction of application components or the application itself resulting in transient state loss. If the transient state of an application is not properly saved and restored it may result in unexpected behavior or application crashes. The existing methods for testing transient state restoration on Android applications require using Android developer tools, advanced Android developer options, or running manual procedures on the device. There are no simple, efficient on-device options for testing transient state restoration on Android.

This report describes Android events that trigger transient state loss. It describes how improper transient state restoration can result in unexpected behavior or application crashes. It provides an overview of existing transient state testing options discussing their limitations and shortcomings. It describes the design and implementation of the Transient State Restoration Testing Utility (TSRTU) highlighting its advantages over existing options. It illustrates how TSRTU is used to test an application. Lastly, it shows how TSRTU, a deterministic, nonintrusive utility can efficiently test transient state restoration on any Android application in seconds with a single touchscreen event.

Table of Contents

List of Figures

Chapter 1

Introduction

As the Android platform grows rapidly with shipments of over 1 billion Android devices in 2014, there is a need for efficient test tools that can help developers find issues and reduce development effort [2].

Android is an "open source software toolkit for mobile phones that was created by Google and the Open Handset Alliance" [3]. It is a software stack for mobile devices that includes an operating system based on the Linux kernel. In addition, Android has a number of kernel enhancements that are optimized for mobile devices. These enhancements include an alarm driver, an Android shared memory driver, power management, low memory killer, kernel debugger, and logger [4].

The Android open source software toolkit also maintains a testing framework that provides a variety of ways to perform testing on Android applications [5]. Android provides the AndroidJunitRunner for unit and instrumentation testing, Espresso for functional UI testing, and the UI Automator framework for performing automated UI testing across multiple apps [6]. While Android provides an extensive set of frameworks and tools to obtain significant application testing coverage, there are still Android specific system behaviors

that are not covered well by these test frameworks.

A particularly challenging system behavior to test on Android is transient state restoration. Transient state is any data that is not permanently stored on the device and is instantiated as a static or dynamic field in an Activity or as part of the global scope of the application. An Activity is an application component that provides a screen which users can interact with to perform a task. Android applications usually consist of multiple Activities that are bound to each other [7]. The source of transient state can vary based on the functionality of the application. In most cases, transient state captures user input, data from resources on a device, or data retrieved from a network. Transient state restoration is the process of properly saving and restoring any transient state that is part of the application when an Activity or the entire application process are recreated. When transient state is not properly managed the user may experience unexpected behavior or an application crash.

Typically, Android applications retain their transient state throughout the entire life cycle of the application. However, there are several types of Android events that can trigger transient state loss in a specific Activity or in an entire application. Transient state loss occurs when an Activity or an application are destroyed causing all memory associated with that component to be deleted.

Reliably generating Android events that trigger transient state loss requires special steps that must be applied to the device and can be difficult to setup, run, or automate. However, it is still important to generate transient

state loss events to ensure that an application does not have any transient state restoration bugs. The existing methods for testing transient state restoration require using external Android developer tools, advanced Android developer options, or running manual procedures on the device. Each method falls short of providing a simple and straight-forward mechanism for performing transient state restoration testing on Android.

To improve transient state restoration testing on Android, an alternative approach is introduced that is deterministic and nonintrusive. This utility, the Transient State Restoration Testing Utility (TSRTU), can be run against any Android application installed on the device with a single tap on the screen. In the remainder of this report we describe what causes transient state loss in Android applications. We show how transient state loss can lead to unexpected behavior or application crashes. We provide an overview of the existing transient state restoration testing methods. We describe the design and implementation of the alternative transient state testing solution highlighting its advantages over existing methods. We provide a demonstration of how to setup and run TSRTU on an Android application. Lastly, we share the results of developing this utility and provide suggestions on how transient state restoration testing can still be improved.

Chapter 2

Transient State Loss

There are two primary ways that transient state loss can occur in an Android application. The first is due to specific types of Android device configuration settings changes that can occur at run time. The second is caused by the Android low memory killer, a memory management kernel extension that assigns priorities to processes running on the system to determine which processes to kill under low memory conditions.

2.1 Android Device Configuration Changes

There are several Android device configuration settings that can change at run time. Screen orientation, keyboard availability, and language settings can all change at run time. When a device configuration change of this type occurs, Android recreates the Activity. This behavior was designed to help the application adapt to new configurations by automatically reloading the application with alternative resources matching the new device configuration [8]. When an Activity is recreated, the Activity is destroyed, deleting all transient state. It is then recreated with a new Activity instance. If the Activity properly retained its transient state, it can now be restored.

2.2 Android Low Memory Killer

When an Android device runs low on memory, the Android "low memory killer iteratively terminates running applications starting with the most unimportant one" [9]. Each process is assigned a value, that can dynamically change during the lifetime of the process. The Android low memory killer places the highest importance on application processes that have Activities in the foreground. Whenever an application process is killed any transient state that existed in any Activities or in the application is destroyed. The entire memory footprint of the application process is deleted to provide memory to other processes. Typically, only application processes that are running in the background are killed by the Android low memory killer. This can be quite common since Android allows running multiple applications simultaneously. Users can navigate between applications very easily on Android and only one application can be in the foreground at one time. Unlike in an Android device configuration change where transient state loss is scoped just to the Activity, the destruction of the application process by the Android low memory killer ensures that all the transient state within the application is destroyed.

Chapter 3

Managing Transient State

The Android application life cycle provides mechanisms to save and restore transient state. In order to show how transient state is managed within an application we will describe how transient state can be managed at the Activity and Application level using an Android API demo application example.

3.1 Activity Transient State

An Activity may have user input state or "member variables that track the user's progress in the Activity" [1]. Android provides mechanisms in the Activity life cycle to save and restore transient state when an Activity is destroyed and must be recreated when the Activity is resumed by the user. The Android documentation states that "to properly handle a restart, it is important that your activity restores its previous state through the normal Activity lifecycle" [8]. This process is shown in Figure 3.1.

Figure 3.1: Activity life cycle for restoring transient state [1]

3.1.1 An Example of Activity Transient State Loss

To best illustrate transient state generated from user input we will describe an example of transient state loss in the Android API demo application example List6, followed by providing a solution that properly restores the transient state.

In the Android API demo application example List6, the Activity defined in List6.java, presents a list of expandable items to the user. Each expandable item presents text when the user clicks on an item as shown in Figure 3.2.

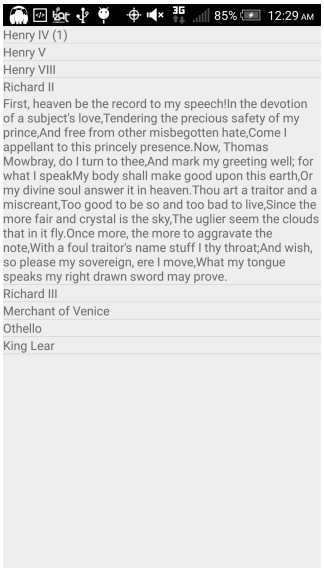

Figure 3.2: List6 with an expanded list item

In this case, the Activity instantiates a new SpeechListAdapter when it is created and does not save and restore the state of items that have been expanded as shown in Figure 3.3.

```
1  /**
2   * A list view example where the
3   * data comes from a custom
4   * ListAdapter
5   */
6  public class List6 extends ListActivity
7  {
8
9      @Override
10     public void onCreate(Bundle savedInstanceState)
11     {
12         super.onCreate(savedInstanceState);
13
14         // Use our own list adapter
15         setListAdapter(new SpeechListAdapter(this));
16     }
17
18
19
```

Figure 3.3: List6 expandable item instantiation

As a result, if transient state loss occurs, due to a screen orientation

change as an example, the Activity is recreated, and the list items are reset to a collapsed state as shown in Figure 3.4

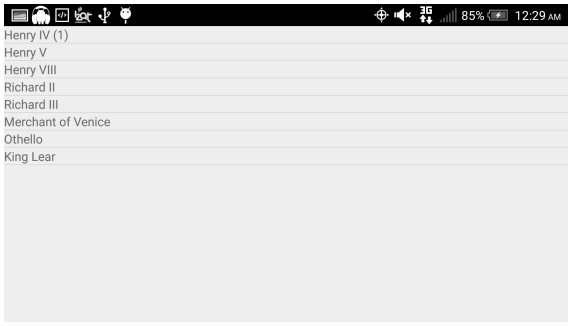

Figure 3.4: List6 with collapsed items with a horizontal screen orientation

To ensure that the transient state is restored when a transient state loss event occurs we must refactor the code to save the state of the collapsible list view as shown in Figure 3.5.

```java
/**
 * A list view example where the
 * data comes from a custom
 * ListAdapter
 */
public class List6 extends ListActivity
{
    private static final String STATE_LIST_ITEM_EXPANDED = "expanded";

    private SpeechListAdapter speechListAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        if (savedInstanceState != null) {
            // Restore value of expanded state from saved state
            boolean[] expanded = savedInstanceState.getBooleanArray(STATE_LIST_ITEM_EXPANDED);
            speechListAdapter = new SpeechListAdapter(this, expanded);
        } else {
            speechListAdapter = new SpeechListAdapter(this);
        }

        // Use our own list adapter
        setListAdapter(speechListAdapter);

    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putBooleanArray(STATE_LIST_ITEM_EXPANDED, speechListAdapter.mExpanded);
        // Always call the superclass so it can save the view hierarchy state
        super.onSaveInstanceState(savedInstanceState);
    }

```

Figure 3.5: List6 refactored to save and restore the list item transient state

After refactoring the Activity by storing the transient state of the collapsible list items in the Bundle provided in the Activity, the state of expanded items can be restored by using a new constructor that initializes the collapsible list with its previous state as shown in Figure 3.6.

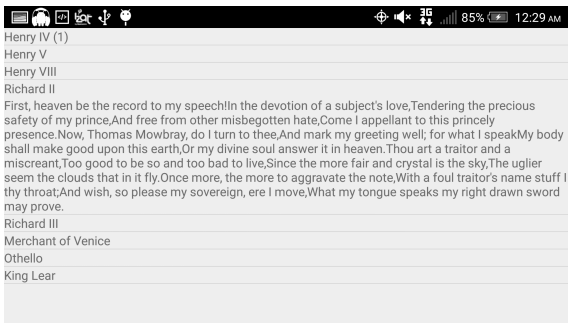

Figure 3.6: List6 restoring transient state

3.2 Application Transient State

It is not uncommon for transient state data to be dynamically stored in the Android Application object or in a globally accessible static variable. Once there, the data can be accessed from any Activity in the application.

Unfortunately, dynamically storing data in the Android Application object or in globally accessible static variables is still non-persistent. If the user places the application in the background, the Android low memory killer may destroy the application process to give other tasks running on the device more memory resources. Any dynamically allocated Android Application object data or any data stored in static variables will be purged from the device.

Often, developers will overlook this fact and attempt to access this data when the application is resumed from the background. If the data accessed is null this may lead to a null pointer exception or if the data accessed has a default value, it may lead to unexpected behavior.

To avoid application transient state loss, the data can be stored using persistent storage options like Shared Preferences or using file system storage on the device. Shared Preferences allow applications to store primitive data in key-value pairs into device storage [10]. Now, even if the application process is destroyed, the transient state data can be restored by fetching it from persistent storage.

Chapter 4

Existing Options for Testing Transient State Restoration

There are several options for testing transient state restoration on Android. Each option varies in terms of setup, external instrumentation, ease-of-use, and repeatability. Several of these options require enabling on-device developer options, using the Android Debug Bridge (ADB), using Android emulators, or performing manual steps. None of these options provide a truly straightforward way to test transient state restoration.

4.1 Android Device Configuration Changes

Screen orientation, keyboard availability, and language are the type of Android device configuration settings that cause Activity transient state loss when they are modified at run time.

4.1.1 Screen Orientation

A screen orientation configuration change is the most common way of testing Activity transient state restoration. Screen orientation changes can be performed with a few different tools. Monkeyrunner, a fuzz testing tool

designed by Android provides a seed-based test generation mechanism that provides support for screen orientation changes [11]. The Android Emulator provides a shortcut key and programmatic way to change screen orientation [12]. Genymotion, a popular third party Android emulator also provides a shortcut key and programmatic way to change screen orientation [13]. Lastly, the user can physically rotate the device to cause screen orientation changes.

It is fairly common for developers to disable this configuration change by adding `android:configChanges="orientation"` to the Activity defined in the Android Application Manifest file of the application. The Android Manifest file provides essential information about the application to the Android system [14]. Disabling orientation changes can greatly reduce the development effort required to support both horizontal and vertical layouts. However, disabling orientation changes does not actually prevent transient state loss from occurring since it can be triggered by other Android device configuration changes or the process being killed by the Android low memory killer. Transient state loss must be tested using an alternative method.

4.1.2 Keyboard Availability

Modifying the keyboard availability requires different keyboard inputs that can be changed at run time. Similar to screen orientation configuration changes, the keyboard availability configuration change can also be ignored by adding `android:configChanges="keyboardHidden"` to Activities in the Android Application Manifest file. As a result, this is not a reliable mechanism

to test transient state restoration.

4.1.3 Language

Android supports changing the language of a system at run time. Changing the language via the system locale setting requires navigating into the Android Settings system application. The Android Settings system application does not have a programmatic interface and can change based on Android API level. As a result, modifying the system locale requires manual steps and these steps may differ based on device.

Using Android device configuration changes to test transient state restoration issues has several limitations. Additionally, even when this capability is available to the user, the scope of the transient state loss when a configuration change occurs is limited to the Activity under test. This leaves any transient state restoration issues outside of the Activity untested.

4.2 High Memory Utilization

Another proposed method for testing transient state restoration is to intentionally allocate large portions of memory outside of the application with the aim of causing the Android low memory killer to eventually destroy the application process being tested.

This option will likely result in application transient state loss. However, depending on the amount of memory available on the device, and the settings of the Android kernel, it may take a substantial amount of time for

the application process to get destroyed by the Android low memory killer or the application process may never get killed at all.

4.3 Don't Keep Activities Developer Option

Android provides on-device developer options that can modify the behavior of the Android kernel. The "Don't Keep Activities" setting is an advanced developer option that destroys Activities whenever the user leaves the Activity.

Enabling this option modifies the behavior of the application when the user navigates through Activities since the option destroys Activities as soon as the user leaves them. In some cases, this presents issues that would not normally appear even in a production environment since it is changing the way that Activities behave. This option does not provide coverage for the transient state of the entire application since it only destroys Activities as the user leaves them. To enable this option, the user must navigate to the Android Settings system application, which requires manual steps.

4.4 Background Process Limit Developer Option

Another on-device developer option that is useful for testing transient state restoration is the "Background Process Limit" advanced developer option. This option enables the user to specify the number of background processes allowed to run on the device. You can allow zero or more background

processes to run in the background. Setting the background process limit to zero will destroy any application that is put in the background once another application process is launched.

This option provides a reliable and repeatable method for testing transient state restoration. The only limitation is that it requires manual steps to enable the option in the Android Settings system application and requires launching another application to cause the process of the application being tested to be destroyed by Android. If another application is not launched, it does not destroy the application process even though it is in the background. This is the best on-device developer option to test transient state restoration available. This option can be enabled by navigating to the Android Settings system application, which requires manual steps.

4.5 Killing the Application Process

Developers can access many Android facilities using ADB. With access to the device via ADB, it is possible to kill the process associated with the device using the `kill` command. In order to test transient state restoration with the kill command, the application must be started, placed in the background by pressing the Home button, and the tester must issue the kill command on the application process associated with the application through ADB.

The major downside to using this option is that it requires device root access, as well as an external device to issue commands using ADB. Root access gives the user access to all files and the required privileges to execute

any command. It is required to kill the application process running on the device. This is the best tool-based option to test transient state restoration because it provides a reliable, repeatable procedure, with transient state loss coverage for the entire application.

Chapter 5

The Transient State Restoration Testing Utility

Despite having numerous options for testing transient state restoration on Android, none of these options provide an efficient simple solution. The two best options require manual steps to enable advanced developer options that can change based on Android API level, manual steps to run the test procedure, or root privileges to issue kill commands through ADB. In the remainder of this chapter, we will describe the implementation of TSRTU.

5.1 Design

The design of TSRTU relies on a number of useful Android features coupled with an Android Service that monitors the user's navigation on the device to determine when to perform transient state destruction in a predictable and reliable way. On Android, Services "are application components that can perform long-running operations in the background and does not provide a user interface" [15]. These features simplified the implementation and enhanced the usability of the utility.

5.1.1 Implementing a Launcher Application

The TSRTU is designed as a launcher application. A launcher application provides a home screen on an Android device that typically displays applications that can be launched by the user. TSRTU displays a grid view of all the applications that can be run on the device. This is fairly typical for a home screen application on Android.

Additionally, to provide support for the TSRTU on Android Lollipop, as well as prior Android releases, a launcher application was necessary to track the package name of the application that was last run by the user. Prior to the release of Android Lollipop, the Android Manager API provided information about the most recent tasks running on the device. This provided a way of determining the last application run by the user. Unfortunately, in Android Lollipop, information about running tasks was deprecated and no longer accessible because of security concerns [16].

5.1.2 Using a System Alert Window

Android provides a mechanism for applications to launch system alert windows using the Android Manifest permission `SYSTEM_ALERT_WINDOW`. This feature is often used to overlay important notifications or system dialogs to the user regardless of what application is running on the device at that moment in time.

We use the system alert window feature to overlay a movable icon that hovers over the application that is run by the user. When the movable icon

is tapped, it performs the transient state restoration testing procedure. The movable icon is only shown when the application is in the foreground.

5.1.3 Killing the Application Process

In section 4.5 we described a way of killing application processes using the `kill` command with ADB as the root user. Fortunately, there is also an on-device programmatic way to kill processes. Android provides an Android Manifest permission `KILL_BACKGROUND_PROCESSES` that grants the application permission to kill background processes by providing the application package name.

5.2 Architecture

The TSRTU application uses two main components to run. The first is the home screen Activity that displays applications that are available on the device. The home screen Activity is also responsible for determining which application was last launched by the user. Secondly, whenever an application is launched, an Android Service is started. The Android Service is responsible for managing the life cycle of the movable icon that is shown to the user on top of the application they have launched. An overview of the architecture is shown in Figure 5.1

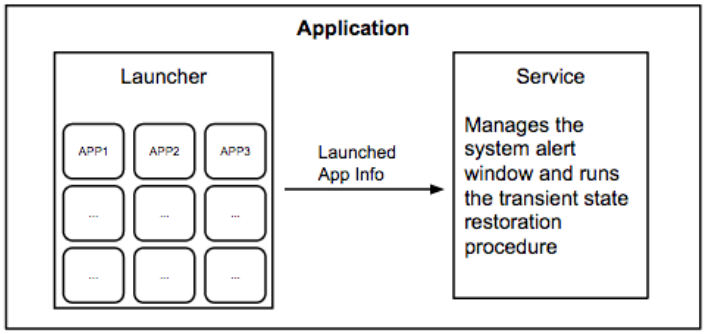

Figure 5.1: Application Overview

5.3 Implementation

When an application is launched from the TSRTU home screen Activity we display a movable icon corresponding to the icon of the application. The movable icon can be dragged around the device window to ensure it does not obscure functionality in the application that is running. The movable icon implements a touch listener which listens for touch events and single taps.

When a single tap on the movable icon is detected, the Service runs the transient state restoration test procedure. The procedure performs the following steps:

1. The single tap action listener re-launches the TSRTU home screen Activity causing the user selected application to go into the background while the TSRTU home screen Activity is shown in the foreground.

2. The user selected application process is now killed by using the Activity Manager API which relies on the package name of the application that

was saved in the TSRTU home screen Activity when the application was launched by the user. The application can be killed because it is in the background and the launcher application has permission to kill background processes.

3. The user selected application is re-launched by using the same mechanism as when the application is launched from the TSRTU home screen Activity.

4. At this point, if the application suffers from a transient state restoration bug the user may see unexpected behavior or the application may crash attempting to access data that is no longer available.

Developers can open any Android application on the device, navigate through the application to reach a desired transient state, and with a single tap they can test the transient state restoration of that particular application.

5.4 Development

The TSRTU source code is written in Java using the standard Android SDK libraries. A sample launcher application, called simplelauncher, was used to provide the basic home screen functionality. Simplelauncher is available at https://github.com/arnabc/simplelauncher [17]. Simplelauncher provides a grid based view for applications display on the home screen. The majority of the changes made to support the TSRTU were made in the Android Service responsible for managing the movable icon as well as running the

transient state restoration procedure. Minor modifications were made to the home screen Activity to provide the Service with the necessary application package name information to run the transient state restoration procedure. Overall, 1444 lines of code were added, and 1068 lines were deleted across 18 commits. The Service, the most significant addition, was 234 lines. TSRTU is open source and is available at https://github.com/idelgado/tsrtu [18].

Chapter 6

Using the Transient State Restoration Testing Utility

6.1 Installation

The TSRTU is installed like any other Android application. It can be installed via ADB or through Google Play. Once installed, it may be convenient to set the utility as the default home launcher.

6.2 Using the Utility

Once the utility is installed, anytime an application is launched the Service will display a movable icon that can be tapped to initiate the transient state restoration procedure. To illustrate we use the Android API demo application example List6 and expand one of the list items. In Figure 6.1 you can see the expanded item and the movable Android application icon on the right side of the screen.

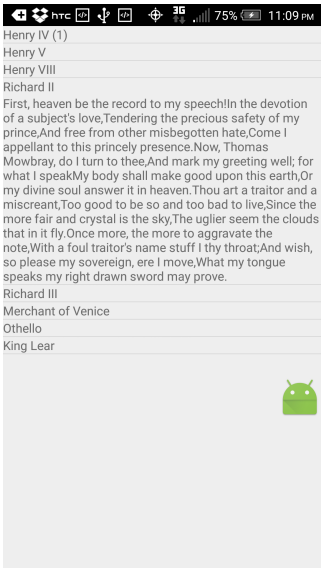

Figure 6.1: List6 with expanded item and movable icon

Once clicked, the transient state of the Activity is destroyed, the application is re-launched and as expected the expanded item is collapsed as shown in Figure 6.2.

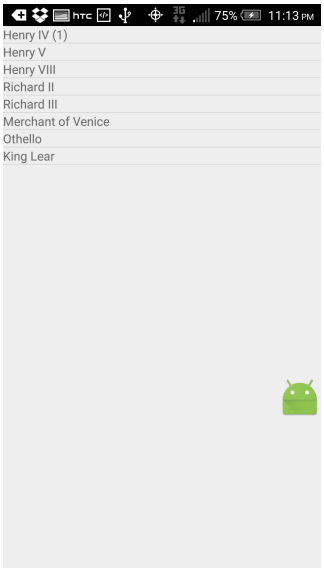

Figure 6.2: List6 after running the transient state restoration procedure

When the application is resumed, the utility restores the movable Android application icon and it can be clicked again to repeat the procedure at any time.

In this example, the transient state is scoped to the Activity, but as was highlighted in section 5, the transient state restoration procedure destroys the transient state of the entire application process providing transient state restoration coverage for the Activity as well as any transient state outside of the Activity that it may depend on.

Chapter 7

Conclusion

In this report we described the challenges with testing transient state restoration on Android applications. We highlighted the existing methods for testing transient state restoration. We proposed a novel alternative, that greatly simplified transient state restoration testing on Android applications.

7.1 Results

The TSRTU is simple to setup and does not require any special knowledge to run. The utility provides a definitive advantage over existing transient state restoration options with the following characteristics:

Nonintrsuive Once the TSRTU application is installed, the utility can be run repeatedly without any additional steps, requiring no modifications to applications, specialized instrumentation, or changes to settings on the actual device.

Efficient The procedure in the TSRTU takes just a few seconds to run and requires minimal device interaction.

Deterministic The user can navigate to the same application state and re-

peat the transient state restoration procedure as often as necessary. The procedure will perform the same action every time always destroying the entire transient state of the application.

7.2 Lessons Learned

There are a few lessons learned from the design and implementation of TSRTU. The initial design of TSRTU was not an Android launcher application. Instead, the first design attempted to replay the ADB kill command to intentionally destroy the application process to cause transient state loss. The ADB kill command packet was recorded, and was later replayed on the localhost interface of the device to attempt to kill the application process of interest. This never worked, adbd (Android Device Bridge Daemon) never responded to the packet being sent from the localhost interface. Fortunately, the design of TSRTU turned out to be much simpler to implement than this initial approach which relied on procedures that were not typical of an Android application.

As a general rule of thumb, it is important to check for Android SDK API changes in the entire range of Android releases an application needs to support. Initially, the TSRTU relied on the Android Activity Manager API to determine the last running application running on the device. Unfortunately, in Android Lollipop, this functionality was deprecated, and was no longer available. To overcome this limitation, TSRTU used a launcher application relying on the home screen Activity to track which application was last

launched by the user.

7.3 Future Work

An important step in improving the TSRTU is to provide test procedures for using this utility in Android system test frameworks like Appium or Selendroid [19, 20]. This would allow developers to automate transient state restoration testing on their application.

One challenge with testing transient state restoration is that any Activity within an application can experience transient state loss. In order to obtain full transient state restoration testing of an application, the developer must run the TSRTU on every Activity within the application. Tracking which Activities of the application are covered by the TSRTU would be a useful addition to provide developers with Activity coverage information during testing.

Bibliography

[1] Recreate An Activity. http://developer.android.com/training/basics/activity-lifecycle/recreating.html.

[2] Android and iOS Squeeze the Competition. http://www.idc.com/getdoc.jsp?containerId=prUS25450615.

[3] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. Pragmatic Bookshelf, 2009.

[4] Frank Maker and Y-h Chan. A Survey on Android vs. Linux. *University of California*, pages 1–10, 2009.

[5] Android Testing Fundamentals. http://developer.android.com/tools/testing/testing_android.html.

[6] Android Testing Support Library. https://developer.android.com/tools/testing-support-library/index.html.

[7] Activities. http://developer.android.com/guide/components/activities.html.

[8] Handling Runtime Changes. http://developer.android.com/guide/topics/resources/runtime-changes.html.

[9] Igor Kalkov, Dominik Franke, John F Schommer, and Stefan Kowalewski. A Real-Time Extension to the Android Platform. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, pages 105–114. ACM, 2012.

[10] Shared Preferences. `http://developer.android.com/training/basics/data-storage/shared-preferences.html`.

[11] Monkeyrunner. `http://developer.android.com/tools/help/monkeyrunner_concepts.html`.

[12] Android Emulator. `http://developer.android.com/tools/help/emulator.html`.

[13] Genymotion User Guide. `https://www.genymotion.com/#!/developers/user-guide`.

[14] App Manifest. `http://developer.android.com/guide/topics/manifest/manifest-intro.html`.

[15] Android Services. `http://developer.android.com/guide/components/services.html`.

[16] Running Tasks. `http://developer.android.com/reference/android/app/ActivityManager.html#getRunningTasks(int)`.

[17] Github Repository for Simplelauncher.

[18] Github Repository for Transient State Restoration Testing Utility. https://github.com/idelgado/tsrtu.

[19] Appium. http://appium.io/.

[20] Selendroid. http://selendroid.io/.