# Dynodroid: An Input Generation System for Android Apps

Aravind Machiry

April 6, 2014

# Part I

# Preface

# Chapter 1

# Organization of this Guide

Dynodroid is a system for automatically generating relevant inputs to Android apps. It is capable of generating both UI inputs (e.g., touchscreen taps and gestures) and system inputs (e.g., simulating incoming SMS messages). It also allows interleaving inputs from machine and human. The system is both *stand-alone*, in that it provides few testing strategies for users to test an app, and *extensible*, in that it allows users to write and run their own strategies. As a result, Dynodroid has two kinds of users: *end-users*, who only wish to use the system, and *developers*, who additionally wish to develop their own testing strategies.

For convenience, this user guide consists of two parts: a guide for end-users and a guide for developers. Unlike end-users, developers need to understand Dynodroid's source code and API, and code written by them is executed as part of a Dynodroid run. Hence, the guide for end-users concerns how to run Dynodroid, and the guide for developers concerns how to extend and debug.

# Chapter 2

# Acknowledgments

Dynodroid would not be possible without the following open-source software:

- Android SDK, Android SDK
- Android SDK tools, development and debugging tools for the Android SDK.

Dynodroid relies on the following open-source tools and libraries:

- Emma, a Java code coverage measuring tool
- Apk tool, a Tool for reverse engineering android apk files

Dynodroid was supported in part by DARPA (contract FA8750-12-2-0020) and gifts from Google and Microsoft.

# Part II

# Guide for End-Users

# Chapter 3

# What is Dynodroid?

Dynodroid is an automated input generation system for android apps which uses technique based on a novel observe-select-execute principle. Dynodroid works by viewing a mobile application as an event-driven program that interacts with its environment by means of a sequence of events, it considers both input events(specific to the app) and system events(common to all apps) to determine which events are relevant to the application in the current state. A selected event from this pool of possible events may then be executed to yield a new state, iteratively, It also provides various log monitoring clients which monitor the behaviour of app as it is being tested. It has the following key features:

- It works on apps both with sources and without sources (apks). For apps with sources, it additionally generates code coverage reports using Emma (cite this).
- It allows user to specify different selection strategies, which are the techniques used to pick an event from the set of available events.
- It observes relevant events for the app, uses the configured selection strategy to select a event and executes the event. This continues till it executes the provided number of events.
- It is multithreaded and can be used to test several apps with various combinations of selection strategies and event counts in a single run. As Dynodroid can take long time to run, it provides a notification feature which is an email that will be sent when it finishes the execution.

Dynodroid is intended to work on atleast Linux and MacOS. It is open-source software distributed under the New BSD License. Improvements by users are welcome and encouraged. The project website is `http://code.google.com/p/dyno-droid/`.

# Chapter 4

# Getting Started

This chapter describes how to download, deploy, and run Dynodroid. Section 4.1 describes how to download the source code of Dynodroid and Section 4.2 explains how to deploy it. Finally, Section 4.3 describes how to run Dynodroid.

## 4.1 Downloading Source Code

To obtain a stable version of dynodroid source code from the SVN repository run the following command:

```
svn checkout https://dyno-droid.googlecode.com/svn/branches/dynodroid.0.1 dyno-droid.0.1
```

Also, you can obtain the latest development snapshot from the SVN repository by running the following command:

```
svn checkout http://dyno-droid.googlecode.com/svn/trunk/ dyno-droid
```

## 4.2 Deploying

Deploying Dynodroid requires the following software:

- Python 2.7 or higher.
- Android SDK 2.3.x (API 10)
- Android SDK tools
- Android SDK Platform-tools

On ubuntu 10 or higher, you can follow the below instructions to setup the required software:

- **Installing Python:**

```
sudo apt-get install python
```

- **Setting up Android:**

```
Browse to the directory where you want to install sdk. For ex: ~/sdk_install
cd ~/sdk_install
wget http://dl.google.com/android/android-sdk_r22.3-linux.tgz -O sdk_tools.tgz
tar -xvzf sdk_tools.tgz
./android-sdk-linux/tools/android
A windows pops up, select Android SDK platform-tools
and Android SDK 2.3.x (API 10) and press install.
```

- **Setup environment variables:** After setting up android, you need to make changes to the environment variables by appending following lines to .bashrc:

```
export SDK_INSTALL=<absolute path of ~/sdk_install/android-sdk-linux from the above step>
export PATH=$SDK_INSTALL/tools:$SDK_INSTALL/platform-tools
```

To deploy Dynodroid, run command "`python dynodroidsetup.py . <directory_to_deploy>`" in the SVN directory. This will set up the provided deployment directory by copying the required sources, libraries and tools from SVN directory. It will also emit preliminary instructions on next steps to run Dynodroid.

## 4.3 Running Dynodroid

Running Dynodroid requires a JVM supporting Java 5 or higher.

- **Copy the required apps:** You need to copy the apps(either with sources or apk) that needs to be tested to the apps folder under deployment directory.

```
cp -r app1 directory_to_deploy\apps
cp app2.apk directory_to_deploy\apps
```

- **Run:**

```
cd directory_to_deploy
ant run
```

# Chapter 5

# Dynodroid Properties

The only way to specify inputs to Dynodroid is by means of properties. Dynodroid expects 2 command line arguments:

1. **Run mode:** Dynodroid supports 2 modes of running, standalone mode(argument : ser) where we run Dynodroid as a standalone application and Java Remote Message Invocation(RMI) mode (argument : rmi) where we run Dynodroid as RMI server to accept requests to test an app. But currently RMI mode is not well tested.

2. **Path of the properties file:** The path to the file containing all the properties required for the Dynodroid.

The ant build script generated by the deployment step takes care of these. `ant run` will run the Dynodroid in standalone mode with properties file `dynodroid.properties` present in the deployment directory. Section 5.1 explains how to set properties and Section 5.2 explains the meaning of properties recognized by Dynodroid. Notation `[<...>]` is used in this chapter to denote the value of the property named `<...>`.

## 5.1 How to Set Properties

There is only one way to pass properties for Dynodroid Via a user-defined *properties file* whose location is the second argument passed to Dynodroid. The default properties file is `dynodroid.properties` in the deploy directory. You can modify the `build.xml` in the deploy directory to use different properties file. Properties are provided as key value pair `<key>=<val>`, with one property per line. By default `dynodroid.properties` contain known good values, but you can change then or provide your own properties file as mentioned before by changing `build.xml`.

## 5.2 Recognized Properties

The following properties are recognized by Dynodroid. The separator for list-valued properties is a comma.

### 5.2.1 Input and Output directories

This section describes properties that control the input and output directories Dynodroid uses to read the apps and writes the results respectively.

`app_dir`

**Type:** location
**Description:** Directory containing the apps that need to be tested. Apps with sources should be in a folder, where as apks can be directly under this directory.
**Default value:** folder name apps under deploy directory.

`work_dir`

**Type:** location
**Description:** Directory where the test results should be stored. There will be a folder for each test variation in the format AppName_¡TestStrategy¿_¡SelectionStrategy¿_¡NoofEvents¿.
**Default value:** folder name workingDir under deploy directory.

### 5.2.2 Test Profile or variation

This section describes properties that control the different variations to be tested on each app.

`test_strategy`

**Type:** String list
**Description:** List of test strategies to run.
**Available values:**

1. **WidgetBasedTesting**: This is the actual Dynodroid testing strategy.
2. **RandomMonkeyTesting**: This is the android monkey testing strategy.

**Default value:** WidgetBasedTesting

#### WidgetBasedTesting Properties

This section describes properties that are used when WidgetBasedTesting is provided as one of the testing strategy.

`max_widgets`

**Type:** Integer list
**Description:** Maximum number of events that needs to be tested.
**Default value:** 1000

`sel_stra`

**Type:** String list
**Description:** The selection strategy that needs to be used to pick an event from the available events.
**Available values:**

1. **RandomBased**: An event is picked randomly from the available list.
2. **FrequencyBased**: An event is picked based on the frequency/number of times it has been triggered.
3. **RandomBiasBased**: An event is picked randomly with negative bias towards already triggered events.

For more details of these selection strategies, please refer our paper Dynodroid Paper.
**Default value:** RandomBiasBased

### RandomMonkeyTesting Properties

This section describes properties that are used when RandomMonkeyTesting is provided as one of the testing strategy.

`event_count`

**Type:** Integer list
**Description:** Maximum number of events that needs to be tested.
**Default value:** 100

`verbose_level`

**Type:** Integer
**Description:** Verbosity level of the logs to be displayed by monkey.
**Possible Values:** 1,2 or 3.
**Default value:** 1

`tch_pct`

**Type:** Integer (Maximum value 100)
**Description:** Integer indicating the percentage of touch events in the random event generator.
**Default value:** 15

`sml_pct`

**Type:** Integer (Maximum value 100)
**Description:** Integer indicating the percentage of small navigation events in the random event generator.

**Default value:** 25

`mjr_pct`

**Type:** Integer (Maximum value 100)
**Description:** Integer indicating the percentage of major navigation in the random event generator.
**Default value:** 15

`trk_pct`

**Type:** Integer (Maximum value 100)
**Description:** Integer indicating the percentage of track events in the random event generator.
**Default value:** 15

### 5.2.3 Results Notify

This section describes properties that are used to notify the completion of Dynodroid run.

`complete_notify`

**Type:** Email address
**Description:** Email-id to be notified post completion of entire run.
**Default value:** someone@example.com

`report_email_user`

**Type:** Email address
**Description:** **gmail** id to be used to send notification mail.
**Default value:** reportSourceUserName@gmail.com

`report_email_pass`

**Type:** string
**Description:** Password of the account specified by reportemailuser property.
**Default value:** password

All other properties can be ignored, as they configure the location of tools used by Dynodroid. The deployment script takes care of populating these with correct values.

# Chapter 6

# Dynodroid Test Profiles

As mentioned earlier Dynodroid is multithreaded. This chapter describes how Dynodroid interprets few properties and run various test variations for a given app. Each test variation is called **Test Profile**.

For each Test Profile:

- A clean emulator will be used.
- A Dedicated directory will be created under the provided working directory where all the logs will be stored.
- A Dedicated Java thread handles the execution.

Following logic is used is used to compute the number of Test Profiles to be run:

```
WidgetBasedTesting_Present = 1 If test_strategy contains WidgetBasedTesting else 0.

Number_Of_Selection_Strategies = Number of Selection Strategies specified
under sel_stra property.

Number_Of_Widget_Counts = Number of widget counts specified under max_widgets property.

WidgetBasedTesting_Combinations =
WidgetBasedTesting_Present * Number_Of_Widget_Counts * Number_Of_Selection_Strategies;

RandomMonkeyTesting_Present = 1 If test_strategy contains RandomMonkeyTesting else 0.

Number_Of_Event_Counts = Number of events counts specified under event_count property.
```

```
RandomMonkeyTesting_Combinations =
RandomMonkeyTesting_Present * Number_Of_Event_Counts;

    Total Number of Test Profiles =
    Number_of_Apps * (WidgetBasedTesting_Combinations + RandomMonkeyTesting_Combinations);

    Example:
       test_strategy=WidgetBasedTesting
   sel_stra=RandomBiasBased,FrequencyBased
   max_widgets=100,1000
   event_count=100,1000
   Assume number of apps provided = 2

    WidgetBasedTesting_Present = 1
Number_Of_Selection_Strategies = 2
Number_Of_Widget_Counts = 2

RandomMonkeyTesting_Present = 0
Number_Of_Event_Counts = 2

WidgetBasedTesting_Combinations = 1 * 2 * 2 = 4
RandomMonkeyTesting_Combinations = 0 * 2 = 0

Total Number of Test Profiles = 2 * (4 + 0) = 8

There will be total 8 Test profiles created and
you will find 8 different folders under specified working directory.
```

# Chapter 7

# Dynodroid Output

This chapter explains output of Dynodroid and its structure. There will be one folder for each of the test profile in the format: AppName_<Test_Strategy_Name>_<Selection_Strategy>_<No_of_Events>.
Each folder will has the following structure:

- **AppHandler**: This folder contains all the log files of app handler, for an app with sources this doesn't contain much info, but for apps provided as apk, this folder contains the extracted contents.
  - **ApkExtractDir**: for apps provided as apk, this contains the extracted (using apktool.jar) contents of the apk.
- **coverageHandler**: usually empty directory
- **CompleteTestProfile.log**: High level log of the status of the test profile. Contains brief description of the result of major steps.
- **TestStrategy**: This folder contains many useful and important logs regarding app behaviour.
  - **PreviousLogs**: Kernel logs and logcat contents after emulator start and before app install.
  - **MethodTraceFiles**: usually empty directory.
  - **SelectionStrategyLog.log**: This folder contains the log of the corresponding selection strategy, the various initialization, selection made at each event. etc.
  - **ODEX_Apk.dex**: The optimized dex file that got installed on the emulator for the app.
  - **GenerelLogCatLogs.log**: File containing logcat entries which are not filtered by atleast one of the monitoring clients, could be useful in finding exceptions etc.
  - **GenerelKernelLogs.log**: File containing kernel messages which are not filtered by atleast one of the monitoring clients.
  - **ResultMonkeyScript.txt**: The script containing all the actions performed by Dynodroid on to the emulator, this provides some sort of reply mechanism.
  - **ManifestInfo.txt**: This file contains the information of the app from AndroidManifest.xml in human readable form.

- **RunStats**: This folder contains coverage report of the app, the folder Coverage1, Coverage2 etc contains intermediate reports and the folder FinalCoverageStats contains the final coverage report at the end of the testing.
- **packages.list**: This file contains the list of the packages that are present in the device after app installation.
- **MonitoringLogs**: This folder contains the logs of various monitoring clients ( each monitor the app for an activity) . Each of the below log files contain log entries along with the event which got triggered, for RandomMonkeyTesting these logs doesn't contain any events. These logs could be very useful especially for malware analysis.
  - **screenshots**: This folder contains png files of the widgets exercised and the corresponding screens.
  - **KernelNetwork_monitoring.log**: This file contains log entries for each of the major network event: open, listen and connect captured at kernel level.
  - **KernelFile_monitoring.log**: This file contains logs entries for each of the file opened only by the app under test, monitored at kernel level.
  - **OutboundUrl_monitoring.log**: This contains Urls being contacted by the app under test. This may not be the complete list of URLS contacted by the app.
  - **LOADLIB_monitoring.log**: The libraries being loaded by the app under test using JNI.
  - **SMSSend_monitoring.log**: The SMS messages being sent by the app under test.
  - **MethodsTrigerred.txt**: The methods of the app executed as part of the test run. This is very useful for apk apps , as emma coverage report cannot be generated for apks.
  - **DEXLOAD_monitoring.log**: The files which are loaded by using Dexload feature of android.
  - There are various other logs which are self explanatory by their name.

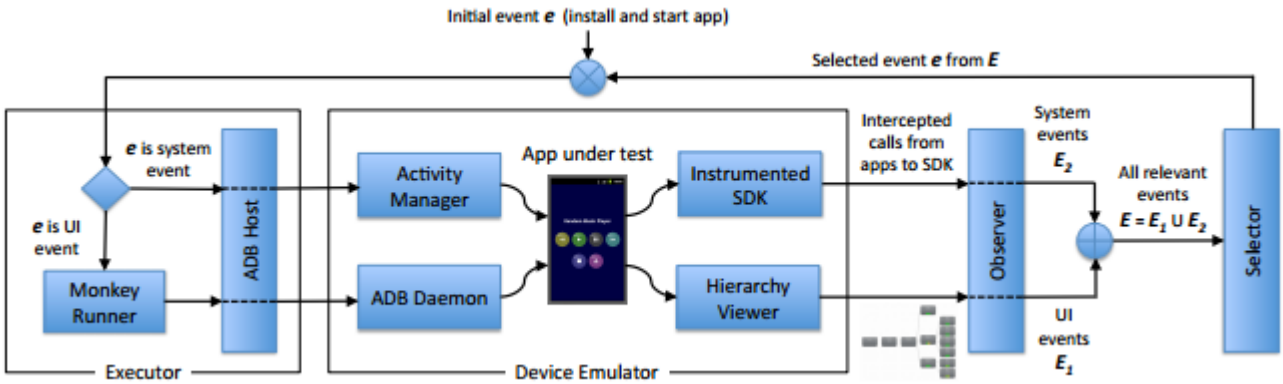
# Part III

# Guide for Developers

# Chapter 8

# Architecture of Dynodroid

This chapter presents the high-level architecture of Dynodroid, depicted below, and describes its key components.




**Dynodroid Properties**

All inputs to Dynodroid are specified by means of properties. Chapter 5 describes how to set properties and the meaning of each property that is recognized by Dynodroid.

**Executor**

The executor triggers a given event using the appropriate mechanism based on the event kind. It uses the Android Debug Bridge (adb) to send the event to an Android device emulator that is running the app under test. For UI events,the ADB host talks to the ADB daemon (adbd) on the emulator via the monkeyrunner tool. Note that this tool, which is used to send events to an emulator via an API, is unrelated to the Monkey fuzz testing tool, which runs in an adb shell directly on the emulator. For system events, the ADB host talks to the Activity Manager tool (am) on the emulator, which can send system events as intents to running apps.Chapter 9 describes the Executor in detail.

**Observer**

The observer computes the set of events that are relevant to the app under test in the current emulator state. It consists of two parts to handle the two kinds of events: the Hierarchy Viewer tool for UI events, and the instrumented framework (SDK) for system events. Hierarchy Viewer provides information about the apps GUI elements currently displayed on the devices screen. The observer computes the set of relevant UI events E1 from this information. The instrumented SDK provides information about broadcast receivers and system services for which the app is currently registered. The observer computes the set of relevant system events E2 from this information. It provides the set of all relevant events E = E1  E2 to the selector. Section 4 provides more details about the observer. Chapter 10 describes the Observer in detail.

**Selector**

The selector selects an event from E as the next event to be triggered. Dynodroid implements various selection strategies that one can choose from upfront, based on the selector configured for a test profile appropriate selection strategy is used to pick an event. Chapter 11 describes the Selector and selection strategy in detail in detail.

Dynodroid supports 2 modes of running, standalone mode(argument : ser) where we run Dynodroid as a standalone application and Java Remote Message Invocation(RMI) mode (argument : rmi) where we run Dynodroid as RMI server to accept requests to test an app. But currently RMI mode is not well tested.

Dynodroid is designed to be extensible, it uses interface based architecture. All Components are specified as Java interfaces, modifying a component involves implementing corresponding interface.

# Chapter 9

# Executor

The executor executes an event, chosen by the selector, on the emulator via the Android Debug Bridge (adb). It uses separate mechanisms for executing UI events and system events. UI events are triggered using the monkeyrunner tool. This tool is a wrapper around adb that provides an API to execute UI events. System events are triggered using the Activity Manager tool (am). Broadcast events of any type can be triggered using this tool provided the correct arguments are supplied.

## 9.1 Customizing Executor

An Executor should subclass edu.gatech.dynodroid.hierarchyHelper.DeviceActionPerformer.

```
public abstract class DeviceActionPerformer {

/***
 * This method is used to perform the given action on the given device
 * @param action target action to be performed
 * @param targetDevice device on which the action needs to be performed
 * @return true/false depending on whether performing action is sucessfull or not
 */
public abstract boolean performAction(IDeviceAction action,
ADevice targetDevice);

/***
 * used to set the working directory for the action performer
 * (this is not mandatory , performers need not have any working directory)
 * @param wd the resulting directory
 */
```

```
public void setWorkingDir(String wd){
}

/***
 * End the tracing (if this performer is doing any tracing)
 * @return true if tracing is ended
 */

public boolean endTracing(){
return false;
}

public TraceLogger getTraceLogger(){
return null;
}

}
```

New executor can be added by following the below steps:

1. Implement edu.gatech.dynodroid.hierarchyHelper.DeviceActionPerformer interface.
2. Modify the method edu.gatech.dynodroid.hierarchyHelper.DeviceActionPerformerFactory.getDeviceActionPerformer to create and return an object of the class.

Currently there is only one executor i.e edu.gatech.dynodroid.hierarchyHelper.MonkeyBasedActionPerformer, you can look into the implementation to know more details about this.
Using the new executor requires you to modify the callers of edu.gatech.dynodroid.hierarchyHelper.DeviceActionPerformerFactory.getDeviceActionPerformer to pass the correct type. Look into the constructor of class edu.gatech.dynodroid.testHarness.WidgetBasedTesting for more details.

# Chapter 10

# Observer

The observer computes the set of relevant events after an event is executed. We consider an event relevant if triggering it may result in executing code that is part of the app. The goal of the observer is to efficiently compute as small a set of relevant events as possible without missing any.

## 10.1 Customizing Observer

An Observer should subclass edu.gatech.dynodroid.hierarchyHelper.LayoutExtractor.

```
public abstract class LayoutExtractor {

/***
 * This method is used to setup Device and other data structures that will be required by
 * underlying layout extractor
 *
 * @return true/false on success and failure respectively
 */
public abstract boolean setupDevice();

/***
 * This method is used to get the current focused screen from the device
 * @param getFullDump this is used to specify whether a full dump of the screen
 * is required or not
 * full dump includes all properties of child view elements ,
 * their call backs and all the events
 * that are registered           by this application
 * @return ViewScreen object that contains the current screen
```

```
 */
public abstract ViewScreen getCurrentScreen(boolean getFullDump);

public abstract BufferedImage captureScreenShot(ViewElement targetWidget);

public abstract PsdFile captureCompleteScreenShot(ViewScreen targetScreen);

/***
 * This method is similar to getCurrentScreen but only addition is that this accepts a
   config file which could contain directions for how to handle the known widgets
 * @param getFullDump this is used to specify whether a full dump of the screen is
 * required or not
 * @param viewHandlingConfig the path to the file which contains the directions on how to
 * Exercise each view
 * @return ViewScreen object that contains the current screen
 */
public abstract ViewScreen getCurrentScreen(boolean getFullDump,String viewHandlingConfig);


}
```

New Observer can be added by following the below steps:

1. Subclass edu.gatech.dynodroid.hierarchyHelper.LayoutExtractor and implement all the virtual methods.
2. Modify the method edu.gatech.dynodroid.hierarchyHelper.LayoutExtractorFactory.getLayoutExtractor to create and return an object of the class.

Currently there is only one observer i.e edu.gatech.dynodroid.hierarchyHelper.ViewServerLayoutExtractor, you can look into the implementation to know more details about this.
Using the new observer requires you to modify the callers of edu.gatech.dynodroid.hierarchyHelper.LayoutExtractorFactory.getLayoutExtractor to pass the correct type. Look into the prepare method of class edu.gatech.dynodroid.testHarness.WidgetBasedTesting for more details.

# Chapter 11

# Selector

The selector selects an event for the executor to execute from the set of relevant events E computed by the observer. We implemented three different selection strategies in the selector, called Frequency, UniformRandom, and BiasedRandom. Please refer Dynodroid Paperfor more details.

## 11.1 Customizing Selector

A Selector should subclass edu.gatech.dynodroid.testHarness.WidgetSelectionStrategy.

```
public abstract class WidgetSelectionStrategy {

/***
 * This method gets the next widget to be excised and the corresponding action to
 * be performed
 * @param currScreen The current screen which the app is in according to the caller
 * @param lastPerformedAction The pair of ViewElement and the corresponding action
 * that was performed on the widget
 * @param resultOfLastOperation Result of last operation performed (true/false)
 * @return Pair of Widget and the action to be performed on it
 */
public abstract Pair<ViewElement,IDeviceAction>
getNextElementAction(ViewScreen currScreen,
Pair<ViewElement,IDeviceAction> lastPerformedAction,
boolean resultOfLastOperation);

/***
 * This method is used to notify the strategy about the new screen that popped up and which
```

```
 * according to the caller is not same as old screen
 * @param oldScreen the old screen which the app was in
 * @param newScreen the new screen which is the current screen of the App
 * @return the screen which should be considered as current screen.
 * Note: the return value can be same as old screen/new screen or any screen object
 * which the strategy thinks is same as new screen
 */
public abstract ViewScreen notifyNewScreen(ViewScreen oldScreen,ViewScreen newScreen);

/***
 * The method is used to notify the strategy about the initialization of the App
 * and its first screen
 * @param firstScreen the first screen of the app
 * @return true/false on sucess and failure respectively
 */
public abstract boolean initializeNewScreen(ViewScreen firstScreen);

/**
 * This method will compare the two screens provided
 * @param scr1  The first screen that needs to be compared
 * @param scr2 The 2nd screen that needs to be compared
 * @return true/false depending on whether they are same or not.
 */
public abstract boolean areScreensSame(ViewScreen scr1,ViewScreen scr2);

/***
 * This method is used to check if coverage need to be collected or not
 * @return true/false depending on whether the coverage is required or not
 */
public abstract boolean needDumpCoverage();

/***
 * This will do cleanup tasks
 */
public abstract void cleanUp();


public abstract void addNonUiDeviceAction(Pair<ViewElement,IDeviceAction> action);

public abstract void removeNonUiDeviceAction(Pair<ViewElement,IDeviceAction> action);

/***
 * This method indicates whether a new folder needs to be created for each
 * widget that needs to be excised
 *   *
 * @return true/false depending on whether a fresh directory is required for each widget
```

```
 */
public boolean needFreshDirectory() {
return false;
}

public boolean reStartStrategy(){
return true;
}

}
```

New Selector can be added by following the below steps:

1. Subclass edu.gatech.dynodroid.testHarness.WidgetSelectionStrategy and implement all the virtual methods.
2. Modify the method edu.gatech.dynodroid.testHarness.StrategyFactory.getWidgetSelectionStrategy to create and return an object of the class.

Currently as mentioned before there are 3 selectors i.e
edu.gatech.dynodroid.testHarness.WidgetRandomBiasSelection.WidgetRandomBiasBasedSelectionStrategy,
edu.gatech.dynodroid.testHarness.WidgetFrequencySelection.WidgetFrequencyBasedSelectionStrategy and
edu.gatech.dynodroid.testHarness.WidgetRandomSelection.WidgetRandomBasedSelectionStrategy you can look into the implementation to know more details about these.
Using the new selector requires you to modify the callers of
edu.gatech.dynodroid.testHarness.StrategyFactory.getWidgetSelectionStrategy to pass the correct type. Look into the getProperties method of class edu.gatech.dynodroid.testHarness.WidgetBasedTesting for more details.

# Chapter 12

# Debugging

Software always has bugs, Dynodroid is no exception. This section explains the steps you need to follow to setup debugging environment for Dynodroid.

1. Follow the steps in 4 and make sure that you can run Dynodroid.
2. Download and install Eclipse
3. Set up eclipse project:
   (a) Open eclipse
   (b) Create new project, lets call this dyno-droid
   (c) In the project explorer, import the folder <svn location>/src/edu to src folder of the project (i.e dyno-droid/src) Add dependencies:
      i. Right click on the project − > Properties − > Java Build Path − > Libraries − > Add external jars
      ii. Here select all the jar files under <svn location>/libs
      iii. Also, add ddms.jar,ddmlib.jar and ddmuilib.jar which are present under <sdk_install_dir>/tools/lib
4. Create a remote debugging configuration:
   (a) Run − > Debug Configurations − > Remote Java Application: Here click on New launch configuration (white box on upper left corner)
   (b) Select project as: dynodroid (Created in above step)
   (c) Name it as say: DDRemote
   (d) Click on close
5. Debugging:

```
cd directory_to_deploy
ant debug
```

You will get a debuggee waiting to be debugged like below:

```
Listening for transport dt_socket at address:8000
```

(a) Open eclipse and Put a break point.

```
Ex:
Open edu.gatech.dynodroid.master.MainHarness.java
Put a break point after main() at : if (args.length != 2)
```

(b) Run− >Debug Configurations− >Remote Java Application− >DDRemote(created in previous step)− >Debug

(c) You should now see, eclipse being properly attached. Enjoy debugging :)

# Chapter 13

# Modifying Android Sources

Dynodroid requires few modifications to android sources to be able to get all the information required by the observer 10. This chapter describes how to apply the required modifications to pristine copy of android sources.

**Note:** Dynodroid framework includes system.img, that was built using the modified android sources. This chapter is only for advanced users who are interested in the integrating Dynodroid to an existing system. so that they can apply these changes to the android sources and use Dynodroid in conjunction.

Following information is required to apply the patch.

```
1) Full path to the folder containing android sources to
which the modifications needs to be applied.
Lets call this android_sources_full_path

2) Patch file from the Dynodroid sources.
It will be under <dynodroid_sources_folder>/other/m3_android_sources_diff.patch.
Lets call this patch_file_full_path
```

## 13.1 Applying the patch

- **Browse to the folder containing android sources:**

```
cd android_sources_full_path
```

- **Apply the patch:**

```
patch -p10 < patch_file_full_path
```

## 13.2 Adding emma.jar to system.img

Dynodroid also produces coverage report when an app is provided with sources. This requires emma.jar to be present in the library path. The following modifications are required to add emma.jar to the system.img.

- **Browse to the folder containing Dynodroid sources:**

```
cd dynodroid_sources_folder/tools/systemImgModifier
```

- **Running the script:**

```
./modifysystemimg.sh system.img
This will create a new image file: newsystem.img which is the modified system.img
that needs to be used with Dynodroid.
```

# Chapter 14

# Known Issues

This chapter explains known non-reproducable issues with Dynodroid and workarounds for them.

1. Premature termination with NULLPTR_EXCEPTION: We have seen few cases where Dynodroid terminates prematurely with NULLPTR exception. I think this is because of the adb connection reset. We use adb to communicate with the emulator through sockets. adb can reset the connection, if it decides to do so. In those cases we can face this issue. But this happens very rarely. Generally rerunning the Dynodroid works most of the times.

If you can consistently reproduce any of these issue, please raise an issue in `http://code.google.com/p/dyno-droid/`with the details and we will look into them.