Report on the modification of the BSc. FSM-Learner

Wesley van der Lee

May 8, 2017

Contents

1 Introduction

As the usage of software increases, software quality cannot be assured anymore with old testing standards. This has also been recognized by the company Bunq, who develops software for their mobile banking application. In collaboration with the Delft University of Technology, the bank supervised an entire team of Bachelor students to adapt state of the art testing principles on the mobile application. One of the applied methodologies is model validation, where a state machine of the application is learnt in order to validate whether this model comprises the specified and expected behavior. Apart from this, the learnt model can also be used to search for weaknesses in the system, by observing which transitions might direct the end user to unwanted systems.

As the project code has been published online[1], this application can be modified and used to learn other applications as well. In order to better understand the domain of model learning mobile applications, the source code has been modified to work again, and configured to work on other applications as well.

[1]`https://github.com/bunqcom/fsm-learner`

This report describes the steps that have been undertaken in order to learn the model of another mobile application: 9292[2]. First there will be discussed how the environment is configured where dependencies of the tool are installed. Next the required modifications for the source code is discussed. Lastly, the working tool is used on the 9292 application.

2 Configuring the Environment

Due to own experience and convenience, the driving environment is the following operating system: `Linux ubuntu 4.8.0-46-generic #49 16.04.1-Ubuntu x86_64`. Because the tool is written in Java, the software will run inside a Java Virtual Machine, and thus the tool will act the same on any operating system.

The following sections each describe what software is installed in what way.

2.1 Java

The project's source code is written in Java. In order to execute the source code, Java needs to be installed. Java JRE and JDK were installed with the following commands resepctively: `apt-get install default-jre` and `apt-get install default-jdk`

2.2 Appium

Reading from the Bachelor Report, the software heavily depends on the Appium tool to invoke actions on the mobile device. Appium can be installed through the `apt-get` functionality. After installing Appium this way, it was soon discovered that appium could not be initialized successfully. Appium should be installed via the Node.js package manager `npm`. To install `npm`, a brew-like installation is required, such as `linuxbrew`.

```
> ruby -e "\$(curl -fsSL https://raw.githubusercontent.com/
    Linuxbrew/install/master/install)"    # download and execute
    brew installer
> PATH="$HOME/.linuxbrew/bin:$PATH"

> brew install node       # get node.js
> npm install -g appium   # get appium
> npm install wd          # get appium client
```

Listing 1: Issued Commands to Install Appium

After the specified commands were executed, Appium was correctly installed and could be used to run Appium demo's.

2.3 Maven

To build and execute the Java source code, maven needs to be used. Maven could be successfully installed with `apt-get install maven`.

[2]https://play.google.com/store/apps/details?id=nl.negentwee&hl=en

2.4 Android Studio

After inspecting a piece of code which was missing from the original code, which will further be discussed in section [?]XXX, the `adb` utility is used by the fsm-learner. `Adb` is an utility within Android Studio and must therefore be installed in order to gain the advantages of `adb`.

```
$> wget https://dl.google.com/dl/android/studio/ide-zips/2.3.1.0/
    android-studio-ide-162.3871768-linux.zip  # download Android
    Studio
$> sudo unzip android-studio-ide-162.3871768-linux.zip -d /opt   $>
    unpack in /opt
$> PATH="/opt/android-studio/bin:\$PATH"  # add to path
```

Listing 2: Install Appium

The mobile device I used was an Oneplus X. The device was not accessible with `adb`, most likely because the Vendor Id for this specific device was not provided in the standard list of mobile device vendors [3]. This is likely due to the fact that OnePlus is not a globally supported phone. As a result, OnePlus devices are not recognized as MTP devices, which is needed for adb in order to invoke actions on the device.

Via the `lsusb` command, one can identify the vendor ID from this specific USB connected device, which is `2a70`. This is then used to declare the vendor ID for recognition by `adb` and `MTP`, after when the phone can be declared as a `MTP` device

```
# 1. Declare Vendor ID for ADB:
mkdir --parent ~/.android
echo "0x2a70" >> ~/.android/adb_usb.ini

# 2. Declare Vendor ID for MTP:
sudo wget --header='Accept-Encoding:none' -O /etc/udev/rules.d/51-
    android.rules https://raw.githubusercontent.com/
    NicolasBernaerts/ubuntu-scripts/master/android/51-android.rules
sudo chmod a+r /etc/udev/rules.d/51-android.rules

# 3. Declare the phone as a MTP device
sudo wget --header='Accept-Encoding:none' -O /etc/udev/rules.d/69-
    mtp.rules https://raw.githubusercontent.com/NicolasBernaerts/
    ubuntu-scripts/master/android/69-mtp.rules
sudo chmod a+r /etc/udev/rules.d/69-mtp.rules

# 4. Restart udev daemon and adb server:
sudo service udev restart
adb kill-server
adb start-server
```

After completing all steps to declare the Vendor ID to different services, the phone could be accessed through `adb`.

3 Modify Project Code

The project code could not be used as-is, due to changes in the environment and updates on other utilities. This section will be devoted to the how and why certain code fragments have been modified in order to make the tool work again.

[3] `https://developer.android.com/studio/run/device.html#VendorIds`

3.1 Missing Script

After observing the source code, one could immediately see that a file called `make_dump.sh` was invoked, but not present in the directory. I contacted the original authors of the project to ask them whether they could provide me with the script, and they did. The following is the missing script `make_dump.sh`.

```
args=("$@")
FILENAME=${args[0]}
adb shell uiautomator dump
echo ${FILENAME}
adb pull /storage/emulated/legacy/window_dump.xml alphabet/
    window_dumps/$FILENAME
```

Listing 3: make_dump.sh

The missing script in fact performs two system commands invoking adb: first it creates a screen dump with the uiautomator, then it downloads the dump from the device, to the project's subfolder. After inspecting the missing script, I found that these two arguments can be executed from the Java project, there was thus no reason to decouple these commands from the Java project.

```
 ...
 try {
  Runtime.getRuntime().exec("adb shell uiautomator dump").waitFor()
    ;
  Runtime.getRuntime().exec("adb pull /storage/emulated/legacy/
    window_dump.xml alphabet/window_dumps/" + fileName).waitFor();
  return 0;
}
...
```

Listing 4: Java implementation of make_dump.sh in method: Main.runAlphabetScript

When executing the program, it was discovered that the screenshots were not retrieved at all. After some debugging, it has been discovered that the `uiautomator` utility has changed since the source code was created. Instead of creating a screenshot file on the phone at `/storage/emulated/legacy/window_dump.xml`, it now creates a screenshot file at `/sdcard/window_dump.xml`. The following source code is applied to make this functionality work again:

```
Script(String fileName) {
  ...
  try {
    if(Runtime.getRuntime().exec("adb shell uiautomator dump").
    waitFor() != 0)
      return Runtime.getRuntime().exec("adb pull /sdcard/
    window_dump.xml alphabet/window_dumps/" + fileName).waitFor();
  } catch (IOException e) {
  ...
}
```

Listing 5: Final Java implementation of make_dump.sh in method: private int runAlphabet

3.2 Compilation Errors

Upon compilation after implementing the `uiautomator` utility, two classes do not compile any more. The classes `AndroidInstrumentator.java` and `IOSInstrumentator.java`.

Both classes do not compile because they contain statements like:

```java
waitFor.until(ExpectedConditions.elementToBeClickable(By.xpath(
    xpath)));
```

These statements can be changed to the following code, because the `ExpectedConditions` need to be cast to a `Function`-instance.:

```java
waitForLogin.until((Function<? super WebDriver, ?>)
    ExpectedConditions.elementToBeClickable(By.id("com.myApp.
    android:id/myBtn")));
```

3.3 Runtime Errors

When performing the learn action with the following command: `mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="learn"`, the following stack trace is generated:

```
java.lang.reflect.InvocationTargetException
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(
    NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(
    DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at com.bunq.main.Main.execute(Main.java:244)
  at com.bunq.main.Main.main(Main.java:315)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(
    NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(
    DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.codehaus.mojo.exec.ExecJavaMojo$1.run(ExecJavaMojo.java
    :282)
  at java.lang.Thread.run(Thread.java:745)
Caused by: java.lang.NoSuchMethodError: com.google.common.collect.
    ImmutableSet.toImmutableSet()Ljava/util/stream/Collector;
  at org.openqa.selenium.remote.ProtocolHandshake.
    streamW3CProtocolParameters(ProtocolHandshake.java:238)
  at org.openqa.selenium.remote.ProtocolHandshake.createSession(
    ProtocolHandshake.java:104)
  at org.openqa.selenium.remote.HttpCommandExecutor.execute(
    HttpCommandExecutor.java:141)
  at org.openqa.selenium.remote.RemoteWebDriver.execute(
    RemoteWebDriver.java:604)
  at io.appium.java_client.AppiumDriver.execute(AppiumDriver.java
    :180)
  at org.openqa.selenium.remote.RemoteWebDriver.startSession(
    RemoteWebDriver.java:244)
  at org.openqa.selenium.remote.RemoteWebDriver.<init>(
    RemoteWebDriver.java:131)
  at org.openqa.selenium.remote.RemoteWebDriver.<init>(
    RemoteWebDriver.java:158)
  at io.appium.java_client.AppiumDriver.<init>(AppiumDriver.java
    :109)
  at io.appium.java_client.android.AndroidDriver.<init>(
    AndroidDriver.java:39)
  at com.bunq.teacher.AndroidInstrumentator.startApp(
    AndroidInstrumentator.java:48)
  at com.bunq.teacher.FsmTeacher.start(FsmTeacher.java:36)
  at com.bunq.learner.SulAdapter.<init>(SulAdapter.java:51)
  at com.bunq.learner.FsmLearner.instantiateSuls(FsmLearner.java
    :162)
  at com.bunq.learner.FsmLearner.setUpLearner(FsmLearner.java:205)
  at com.bunq.main.Main.learn(Main.java:81)
  ... 12 more
  Suppressed: java.io.IOException: Incomplete document
    at com.google.gson.stream.JsonWriter.close(JsonWriter.java:527)
    at org.openqa.selenium.remote.ProtocolHandshake.createSession(
    ProtocolHandshake.java:121)
    ... 26 more
```

It appears that the class AndroidInstrumentator with method startApp causes the runtime error. The AndroidInstrumentator class takes care of all actions that can be executed on an Android device and depends heavily on Appium. Debugging the code, shows that the following statement within the AndroidInstrumentator class is responsible for the failure:

```
driver = new AndroidDriver(new URL("http://127.0.0.1:4723/wd/hub"),
     capabilities);
```

The corresponding exception that was raised, looked similar to another situation where an application called ErrorProne raised a similar stack trace [4]. This issue can be resolved by updating the file `pom.xml` to include the latest Guava version [5].

With updated Guava, another exception is raised at runtime:

```
Exception in AndroidInstrumentator:startApp
org.openqa.selenium.SessionNotCreatedException: Unable to create
    new remote session. desired capabilities = Capabilities [{
    appPackage=com.bunq.android, appActivity=com.bunq.android.ui.
    activity.MainActivity, noReset=false, newCommandTimeout=100,
    platformVersion=6.0.1, platformName=Android, deviceName=OnePlus
     X}], required capabilities = Capabilities [{}]
Build info: version: '3.3.1', revision: '5234b325d5', time: '
    2017-03-10 09:10:29 +0000'
System info: host: 'ubuntu', ip: '127.0.1.1', os.name: 'Linux', os.
    arch: 'amd64', os.version: '4.8.0-46-generic', java.version: '
    1.8.0_121'
Driver info: driver.version: AndroidDriver
  at org.openqa.selenium.remote.ProtocolHandshake.createSession(
    ProtocolHandshake.java:126)
  at org.openqa.selenium.remote.HttpCommandExecutor.execute(
    HttpCommandExecutor.java:141)
  at org.openqa.selenium.remote.RemoteWebDriver.execute(
    RemoteWebDriver.java:604)
  at io.appium.java_client.AppiumDriver.execute(AppiumDriver.java
    :180)
  at org.openqa.selenium.remote.RemoteWebDriver.startSession(
    RemoteWebDriver.java:244)
  at org.openqa.selenium.remote.RemoteWebDriver.<init>(
    RemoteWebDriver.java:131)
  at org.openqa.selenium.remote.RemoteWebDriver.<init>(
    RemoteWebDriver.java:158)
  at io.appium.java_client.AppiumDriver.<init>(AppiumDriver.java
    :109)
  at io.appium.java_client.android.AndroidDriver.<init>(
    AndroidDriver.java:39)
  at com.bunq.teacher.AndroidInstrumentator.startApp(
    AndroidInstrumentator.java:50)
```

[4]https://github.com/google/error-prone/issues/528
[5]`https://mvnrepository.com/artifact/com.google.guava/guava/21.0`

```
  at com.bunq.teacher.FsmTeacher.start(FsmTeacher.java:36)
  at com.bunq.learner.SulAdapter.<init>(SulAdapter.java:51)
  at com.bunq.learner.FsmLearner.instantiateSuls(FsmLearner.java
    :162)
  at com.bunq.learner.FsmLearner.setUpLearner(FsmLearner.java:205)
  at com.bunq.main.Main.learn(Main.java:81)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(
    NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(
    DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at com.bunq.main.Main.execute(Main.java:244)
  at com.bunq.main.Main.main(Main.java:315)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(
    NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(
    DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at org.codehaus.mojo.exec.ExecJavaMojo$1.run(ExecJavaMojo.java
    :282)
  at java.lang.Thread.run(Thread.java:745)
```

After an insane amount of Googling and discussing the stack trace with other people, one can derive that Selenium cannot send HTTP requests to the AndroidDriver because the ProtocolHandshake continuously fails. This is due to the fact that Guava 21.0 (which has just been set) and Appium's java-client 2.1.0 are incompatible in terms of HTTP session establishment, which was the root cause for the failing ProtocolHandshake as specified in the stack trace. Updating Appium's dependency configuration to the latest version: `5.0.0-BETA7`, overcame the problem. No stable Appium version (latest stable is Appium 4.1.2) was able to make a successful handshake for the HTTP session.

3.4 Modifications for Generics

The tool was designed specifically for the Bunq application. Because I do not have a Bunq (test) account, it was not suitable to learn a state machine of this application. Also creating a valid account to use as a test account, could be disruptive for Bunq's back-end. Hence the tool needs to be modified for generic application usages. Several functionalities need to be changed.

Login

The `login()` method is used by different methods like the reset methods and after the AndroidDriver is started. Since this login function is very specific to the Bunq application and not many applications require a login screen, the login-functionality has been overwritten by overriding the login-function as follows:

```java
// Does not login the user. Functions as a mock-up.
public String login(String pin) {
  return "logged_in";
}
```

Configurations

The class `androidConfig.properties` contains `appPackage=com.bunq.android` and `appActivity=com.bunq.android.ui.activity.MainActivity`. For this example, the 9292 app is learned. To retrieve the needed information, one can follow the following commands:

```
#> ~/Desktop$ adb shell pm path nl.negentwee
package:/data/app/nl.negentwee-1/base.apk
#> ~/Desktop$ adb pull /data/app/nl.negentwee-1/base.apk .
893 KB/s (4766922 bytes in 5.212s)
#> ~/Desktop$ ls
base.apk
#> ~/Desktop$ mv base.apk 9292.apk
#> ~/Desktop$ ls
9292.apk
#> ~/Desktop$ aapt dump badging 9292.apk | grep launchable-activity
launchable-activity: name='nl.negentwee.activities.StartupActivity'
      label='9292' icon=''
```

This gives the following information: `appPackage=nl.negentwee` and `appActivity=nl.negentwee.activities.StartupActivity`

4 Working with the tool

Now the tool has been correctly configured and modified, it can be used to learn the state model of an application. The end user can execute four types of commands through maven:

```
mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="learn"
```

This starts the learning process. Requires a valid alphabet.

```
mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="alphabet:create"
```

This action creates an alphabet. Guides the end user through the process of creating window dumps (works only for Android).

```
mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="alphabet:compose"
```

This action composes an alphabet from earlier created window dumps (.xml) or UI Automation dumps (.plist).

```
mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="alphabet:destroy"
```

This action removes an alphabet.

4.1 Application Run

To confirm whether the tool is working properly, an initial run is performed on the modified tool on the main screen of the 9292 application. It executes well and a description of the graph is outputted. This first round's logs are also put online [6].

The state machine returned by the application, is shown in Figure 1. This state machine is a slightly altered to ascertain user readability.

As can be seen in Figure 1, the application consices four different states, and three transitions. Almost one hundred transitions were excluded, again for readability, because they specified the "Element Not Found" transition, which causes the application to go to the floating state 1. Since these transitions were excluded, no transition is visible to state 1, which is the reason why it is a 'floating' state.

[6] `https://raw.githubusercontent.com/wesleyvanderlee/Thesis/master/Literature/BSc.%20FSM%20Learner/92%20main%20learn%20run.txt`

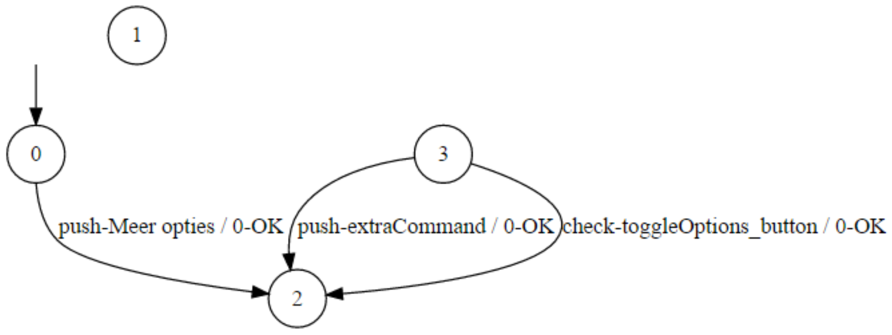

Figure 1: State Machine of application 9292