The Advantages and Disadvantages of Automated UI Testing on iOS Using Apple's UIAutomation Framework

Letter of Transmittal

Dear Constance,

This report explores the UIAutomation framework that Apple released with its initial release of iOS 4 until my experience with it ended with iOS 5. It is directly related to the work I did at <The Company> but it only mentions publicly available tools to ensure company procedures remain confidential. Many bugs and inadequacies are present with Apple's iOS UIAutomation and I've listed a few of the major bugs that I ran into in my course of working with the framework. This report is written as an overview of the tool, to help figure out whether it is worth using for a given project. My recommendations cannot be any more specific as usage of this tool is highly dependent on an individual project. I hope that this report will meet the guidelines required for a Co-op report.

Sincerely,
<Student Name>

The Advantages and Disadvantages of Automated UI Testing on iOS Using Apple's UIAutomation Framework

Summary

User interface or UI testing is crucial in creating an application in the iOS world. Testing UI is normally a boring and repetitive job and being able to automate it would greatly relieve much of the current overhead required to fully test an application. Apple attempted to create a tool (the UIAutomation Framework) to allow developers to automate the UI testing of their application. Whilst it performs this task adequately there is a significant investment in creating scripts to allow one to automate this process. This investment combined with the various issues the tool still has forces one to decide whether or not such an investment is worth it or whether the traditional manual testing is better suited to the task.

Table of Contents

Introduction

Apple's iOS is arguably the premier platform for smartphone applications. It brought the concept of the 'app' and the 'app store' to the masses. With over 500,000[1] applications in the app store one must ensure that their product is perfect in order to attract and keep users. The UI or user interface of the application is what every user will interact with and it is therefore crucial to ensure that the UI is as flawless and as user friendly as possible.

Since a user will most likely be interacting with the UI of an application on a device, it is necessary to test on a physical device as a simulation of the software running on a computer is insufficient. Often there are bugs that occur only on a simulator and not on a device or vice versa. In order to thoroughly test an application its UI functionality needs to be thoroughly tested on every variation of hardware it is expected to run on. This can often get very repetitive and costly as a tester must test the UI manually.

To solve this issue there have been many attempts to automate UI touches and gestures on iOS devices. However, many of these projects are individually run with little to no support and often break every time Apple releases an update to their device or firmware lineup. Furthermore, they often require one to embed code into the application to allow the framework to interact with it, creating code that is more complex and therefore increasing developer overhead.

To solve this problem Apple released the UIAutomation framework with iOS 4. This is part of Apple's XCode IDE and is run as a separate diagnostic application which Apple refers to as an 'Instrument'. It allows the user to simulate touches and gestures in the iOS simulator as well as a connected device. Gestures and touches are simulated by giving the UIAutomation instrument a JavaScript script file with instructions on what to do.

[1] http://www.apple.com/ca/iphone/from-the-app-store/

Advantages

The advantages of such a tool are very apparent. Instead of hiring multiple testers to test an application one can write a simple script to automate the task. When something unexpected occurs the UIAutomation framework can output a plist file containing the steps it took as well as screenshots, timestamps and error messages to help diagnose the problem. It is also very easy to pick up as the JavaScript is very simple and the calls straightforward. UIAutomation has evolved since its release and introduced fixes to a number of issues. Outlined below are some issues that have been addressed between its original release and the iteration released with iOS 5.

Fixed Issues

Creating scripts

Creating scripts with the original iOS 4 release was not as easy as it seemed. One needed to find the element name and hierarchy in the tree using the UIAElementTree call or accessibility tags. In order to use accessibility tags the developer must include them whilst programming the application introducing developer overhead. However an added benefit to this is that it is much easier for the application to be accessed by Apple's Voiceover software which allows visually challenged individuals to interact with iOS products.

Apple has since improved the creation of scripts greatly. One can tap on an item in the simulator to get its hierarchy and therefore cut down on UIAElementTree calls which can take a significant amount of time. Furthermore, Apple has introduced and Automator like feature which allows one to perform an action in the iOS Simulator and have the code to repeat the action automatically generated. Whilst this is generally decent, one still needs JavaScript knowledge to modify the script and address irregularities.

UIAElementTree Call

The UIAElementTree call is very crucial in allowing one to create scripts. In order to access any UI element in the application one needs the path to the object. To get the path one must use the UIAElementTree Call. Initially, the call returned a long line of seemingly unformatted text from which one had to discern the depth of the element and code the script appropriately.

With the iOS 5 version of the UIAutomation instrument the UIAElementTree call is greatly improved. It now returns an indented and formatted output which allows one to easily see the depth and path of a single element. This greatly reduces the time and effort needed to code a script. However, the UIAElementTree call is still fairly time consuming and depending on the number of elements on the screen can produce very complex output.

Running from command line

With the original release of the UIAutomation instrument the only way to run scripts was through the GUI interface. This requires one to be actively testing the product and whilst it eases the burden on a tester it is far from fully automated testing. Furthermore, output needs to be saved manually and this can be a time consuming and repetitively boring task. With the iOS 5 release one can now run the UIAutomation instrument from the terminal and specify the output directory (although currently the output directory flag is ignored and output is stored in the working directory) allowing scripts to be run autonomously.

Sleep timers

Sleep timers to wait for events such as fetching data were notoriously finicky with the original release. This issue has since been remedied.

Complex gestures

Complex gesture support such as 3 finger taps and 5 finger pinches has now been added. Previously only two finger scroll and tap was available.

Disadvantages

Whilst the UIAutomation instrument is a great tool and has been greatly improved since its release there are still a number of issues with the framework.

High initial overhead/investment

In order to utilize the framework one must put in the time and effort to create the script files. There is significant overhead in doing so and depending on the complexity of the application and the desired number of test cases one can spend weeks coding scripts to test the application. Often, manually testing can be faster depending on the project and whilst counterintuitive can often be cheaper.

Scripts break after minor changes

As development progresses and the applications UI elements change scripts need to be modified to ensure they still work. This isn't just the case for major UI changes but rather breakage occurs for very small changes such as moving a button a few pixels over. Having to fix these scripts introduces even more overhead and can often lead to uncertainty on why the script is failing. Is it due to the script or is something wrong with the application?

There is no way to verify it 'really' works

Simulating touches is not the same as actually touching a physical device. There can be quirks that allow one to tap an object which the UIAutomation instrument cannot access and vice versa. Furthermore, scripts break with minor changes leading one to doubt whether or not there is really a problem with the application.

Inadequate for custom interfaces

The UIAutomation framework works very well with stock UIElements. However, the majority of premier iOS applications utilize custom UI elements and the UIAutomation often has issues handling these. There is a larger overhead in trying to script for custom UI elements and scripts that address them are often very prone to breakage.

One cannot automate applications on previous versions of iOS

Often a client will demand that an application works with older deprecated versions of iOS. The UIAutomation framework is not compatible with these and one must resort to manual testing. Whilst this problem will disappear as more iterations of iOS are released it is still a significant problem today and leads one to question whether it is worth automating tests when they will need to be manually run anyway.

One cannot do anything outside the app

The UIAutomation framework does not allow one to do anything outside the application. If settings need to be changed from the settings application this is not currently possible. Furthermore, when attempting to test applications that utilize other applications the UIAutomation instrument fails spectacularly.

One cannot move back to start state

When testing it is crucial that one knows how to reproduce a bug. The UIAutomation instrument executes cases serially and remnants of previous cases remain, influencing the start state for any given case. This results in bugs being missed and limits the number of things one can test.

Conclusion

The UIAutomation framework is a step in the right direction. It brings the much needed ability to automate UI testing on the iOS platform. Whilst it offers the potential for cost and time saving the initial overhead is fairly high. For companies that build applications for other entities a typical project can last between 6 to 8 months and within this timeframe it is often impractical to utilize the UIAutomation framework. For projects lasting longer or applications created by the company that the company actively maintains it is often worth it to invest in UI automation. The UIAutomation framework is still fairly immature and does not allow for fine grain control. It lacks basic requirements for a test tool by not allowing one to reset the start state and as such is only good for augmenting current testing techniques. Overall, the decision to augment ones testing using the UIAutomation framework needs to be decided on a case by case basis and is very dependent on the time span and complexity of the given project.