Reverse Engineering Mobile Apps for Model Generation Using a Hybrid Approach

Ibrahim Anka Salihu, Rosziati Ibrahim
Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), 86400 Parit Raja, Batu Pahat, Johor, Malaysia.
rosziati@uthm.edu.my

***Abstract*—The popularity of mobile devices is ever increasing which led to rapid increase in the development of mobile applications. Model-Based testing can improve the quality of mobile application but the models are not always available or are of inadequate quality. Reverse engineering approaches are used to automatically generate model from the GUI of mobile applications for model-based testing. This paper proposes a hybrid approach for reverse engineering mobile applications which exploit the capabilities of both static and dynamic approaches while trying to maximize the quality of the generated models. The insight of this approach is to use static analysis on app's source to identify supported events. The generated events can be used to dynamically explore an app at run-time to generate a state model of the app's GUI. The preliminary results from our approach indicated that the technique can generate high quality models from android apps.**

***Index Terms*—GUI Model Generation; Mobile Apps; Model-based Testing; Reverse Engineering.**

I. Introduction

The growth in the number of computing devices has been dominated by smartphones and tablets in recent years [1]. Smartphones are used by many people for several computational tasks replacing personal and desktop computers. A recent study indicated that a total of 352 million smartphones (excluding china brands) were sold to users worldwide in the third quarter of 2015 and Android OS is leading with 84.7% [2]. The popularity of these devices lead to a rapid increase in the development of mobile applications to deal with different computational needs of their users [3]. The development of mobile applications has a significant impact from both economic and social perspectives. It has generated revenue of $4.5 billion in 2009, and a recent report estimated that the global applications business will be worth $77 billion by 2017 [4, 5].

Mobile apps are recently increasing in capacity, functionality, structure and behavior [6], hence becoming more and more complex [5]. They are now used not only for entertainment or social networking but also in safety and critical domains, such as payment systems, mobile government, m-health initiatives, etc. [7-9]. The widespread reliance on mobile apps in everyday life poses significant concern on apps quality such as correctness, performance and security [10-13]. Furthermore, the increased complexity has brought several challenges for the software engineering researchers such as understanding apps behavior and testing [1, 14, 15]. Therefore, there is a demand for software engineering techniques and tools to support program understanding, analysis and testing task for mobile apps [8, 16, 17].

Testing can play a significant role in improving the quality of software systems. Model-based testing is becoming increasingly popular among the software engineering community [18]. It is the automation of testing where the test execution is automatically derived from the model of a system under test (SUT). Model-based testing can enhance the creation of test scripts and test coverage of an application [19]. In order to benefit from model-based testing, there is a demand for techniques/tools to aid automated model generation to support analysis and testing tasks. However, building these models fully automatically for the Android apps remains challenging [20].

Several techniques have evolved to deal with the challenges in automated GUI model generation such as [14, 15, 20, 21]. However, the models generated by existing techniques are incomplete due inability to explore apps state extensively [17, 19]. There are numerous challenges which are associated with the nature of the Android platform (framework-based) where many of the app's behaviors reside in the Android framework. Another challenge is the limitation with the reverse engineering approaches (static/dynamic) used in exploring the apps state. To deal with these challenges, we proposed a hybrid approach that combines static and dynamic analysis to improve the exploration of application's state. Our proposed technique leverages the static analysis in GATOR which performs a control flow analysis on the source code of an app to generate control flow graph (CFG). The CFG will be used to extract all supported events which can be used for dynamic exploration.

II. Related Work

Automated model generation from software applications is specifically a reverse engineering process. Reverse engineering is an act of analyzing a software system to extract design and implementation information and abstracting the information in the form of a model for easy understanding [22]. There are two approaches for reverse engineering; Static approach and dynamic approach. The static approach performs analysis on the application's source code or binary code to extract information from an application [19]. It is particularly suitable for extracting information about the internal structure of a system and dependencies among structural elements [23]. Due to the object-oriented nature of GUI applications static analysis is not used widely in generating GUI model for testing. On the other hand, dynamic approach extracts information from an application by executing and analyzing its external behavior [19]. It is well suited for extracting the behavior of GUI applications [23]. The dynamic approach is widely used for reverse engineering GUI applications to generate models for testing an app.

Most techniques for automated model generation for mobile apps are based on dynamic approach due to its ability to deal with the dynamic behavior of GUI apps. One of the earliest technique is GUI ripping [24], an automated GUI model exploration technique for test case generation. GUI ripping generates model from an app by automatically executing and exploring the applications' GUI by opening all its windows and extracting all their widgets (GUI objects), with their properties, and values. It extracts both the structure and execution behavior of the GUIs as GUI tree and event flow graph respectively. The GUI ripper is implemented in GUITAR tool [25]. An extension of the tool has been proposed for the Android apps known as Android GUITAR [26]. GUITAR produces many false event sequences which may need to be weeded out later and it is not able to explore all the GUIs due to infeasible paths, such as when the visibility of some windows depends on other windows.

Amalfitano et al. [20] proposed GUI Ripper that was implemented in AndroidRipper tool for automated testing of Android apps. It is based on a crawler that automatically crawls an app GUI to generate test cases for regression test and crash testing. Due to the challenges of a fully automatic analysis to generate models from executing GUI, the Android Ripper tool is mainly designed to automatically traverse the application's UI to generate and execute test but not to develop a re-usable models of the app. Joorabchi and Mesbah [27] proposed ICRAWLER that is much similar to GUI Ripper and CRAWLJAX [28], dedicated to iOS applications. Some UI elements such as toolbar, slide bar, search bar, and advance gestures such as swiping pages and pinching are not supported by ICRAWLER. Android Automatic Testing Tool (A2T2) [29] is an extension of the crawling technique in [28] for the reverse engineering of android apps. The tool dynamically reverse engineers an app and automatically builds models representing the structure and behavior of the app's GUI. The models generated are GUI tree and state machine models respectively. The state machine model can be used for model-based testing. The tool can only extract a small set of widgets of an Android app.

ORBIT tool [15] integrated both static and dynamic analysis to generate a state model from android applications. It performs static analysis on the source code of an app to generate set of user actions supported by an app. It identifies listener objects and performs a backward slice to track the view ids of the GUI associated with the listeners. A dynamic crawler (built on top of Robotium) is used to fire actions on the GUI objects. This generates a state model that can be used for generating test cases. However, creating a list of actions that can be fired on the GUIs can lead to exploring merely a subset of GUI states.

Azim and Neamtiu [14] proposed Automatic Android App Explorer (A3T), a technique based on hybrid static and dynamic analysis that automatically explores an apps running on real phone or an emulator. The static analysis on A3E is specifically a data flow analysis (taint tracking) on the bytecode of an app to construct static activity transition graph (SATG) with nodes representing activities and edges showing the possible transitions between the activities (UI screens). It uses the SATG as input for automatic exploration of an app. The automatic explorer rips an app using Troyd tool which is based on Robotium to extract GUI elements which are used to fire events on an app. A3E is designed to automatically generate test cases that can be used to test an application but does not store re-usable models.

III. Discussion

The static and dynamic approaches for reverse engineering GUI applications have numerous strengths and weaknesses. The static approach is capable of retrieving more accurate and complete information from an application but the dynamic object-oriented nature of GUI applications can sometimes complicates the analysis, which makes it very difficult or even impossible to retrieve comprehensive information about the behavior of GUIs by just analyzing their source code [6, 29]. This is because access to some components depends on other components and some components are only reachable from a particular state. In addition, information about overlapping windows is not accessible using static analysis. On the other hand, the dynamic approach to reverse engineering could be easier in analyzing the dynamic behavior of GUI applications. However, the information extracted by pure dynamic approaches is incomplete which affects the quality of model generated [19, 30, 31]. The most challenging issue with any dynamic reverse engineering technique is the difficulty in selecting inputs that can be used in controlling the model exploration [15, 19, 30]. Another issue is the scalability that is associated with large amounts of data collected at run-time.

Table 1 shows the results of comparative analysis of mobile apps reverse engineering tools from our earlier work in [36]. This has presented the techniques used by the tools and the limitations with the current tools. As shown in the table most of the available techniques are based on dynamic approach using GUI ripper or crawler for the exploration. Though, dynamic approaches have shown good result in reverse engineering the behavior of GUI applications, the models generated by dynamic approach are incomplete. This highlights the limitation of using ripper or crawler to find actions/events and dynamically explore/crawl an application. On the other hand the hybrid static/dynamic approach can provide better result as it eliminates the problem of identifying events to fire. Several researchers believed that, using static analysis on source code to supply meaningful inputs for the dynamic exploration can ensure comprehensive coverage of GUI apps for the generation of complete and high quality models [1, 19, 32].

Table 1: Comparative analysis of mobile apps reverse engineering tools

Author	Tool	Technique	Approach	Advantage	Limitation
Atif Memon et al. (2003)	Android GUITAR	GUI Ripping	Dynamic analysis	Facilitates model generation for MBT	Information retrieve is Incomplete with several false states
Domenico Amalfitano et al. (2011)	A²T²	GUI Crawling	Dynamic analysis	Ability to recognize when two interface are equivalent	Manages only a subset of the possible Widgets
Amalfitano, D. et al. (2012)	Android Ripper	GUI Ripping	Dynamic analysis	Emphasis on reducing false event sequence in data extracted	It designed to detect bugs based on event traces, does not create re-usable models
Yang et al. (2013)	ORBIT	GUI Crawling	Static and Dynamic	Identify supported actions using static analysis	Order of event sequence is not controlled and no support for complex gesture
Azim, T. and I. Neamtiu. (2013)	A3T	Systematic Exploration using an explorer	Static and Dynamic	Identify all activities and their transition using static taint analysis	Some gesture events are not covered and it does not generate re-usable models

The recent tools based on hybrid approach are ORBIT and A3E but these tools are also not optimal. The static analysis in ORBIT does not capture the general flow of GUI objects and the behavior of callback request. However, the behavior of callback request can certainly modify GUI states, therefore it is essential to track them. In A3E the analysis does not capture menus/dialogs, it is also unclear how the analysis models arbitrary GUI objects and handlers associated with an activity. In conclusion, few techniques exist for automated model generation from mobile apps. Nonetheless, the quality of models generated by the techniques is still inadequate. Therefore, there is need to improve the quality of models generated by automated techniques for model-based testing of mobile apps and this is the goal of our work.

IV. Methodology

A. *Overview of Android Applications*

Android apps are developed in Java and compiled to .dex file which run on a special virtual machine called Dalvik Virtual Machine. They are distributed as apk files, as such the source code is not always available. An app comprises of one or several activities and the activity component present the visual interface screen (window) which a user can interact with. The Activity acts as a container for typical GUI elements (test boxes, check boxes etc). An activity instance can programmatically start another activity which is usually referred to as intent in the android framework. Two types of events are supported on android apps. User events that can be fired on the user interface objects which are implemented by event handlers and system events that are implemented by the lifecycle callback methods from the android framework or as a result of interrupt message from other system component. Android apps also support a wide range of user gestures such as long-press, double-taps, swiping, pinching etc. by using the Gesture Detector class in the framework. Furthermore, there are dialogs and fragments which are added to the view of a running app automatically. Their behavior is determined and controlled by the parent activity that they are attached to. They affect the lifecycle and state of events in Android. As Android programming model is based on callback methods, therefore, analyzing them will guarantee understanding and exploration of more app's state.

B. *Proposed approach*

The aim of this work is to improve the quality of models generated from the GUI of Android applications. Our approach is based on static analysis of application's GUI source code for the extraction of information and used the information for the dynamic exploration of the app at run-time to generate a model. Figure 1 presents an overview of the propose approach.

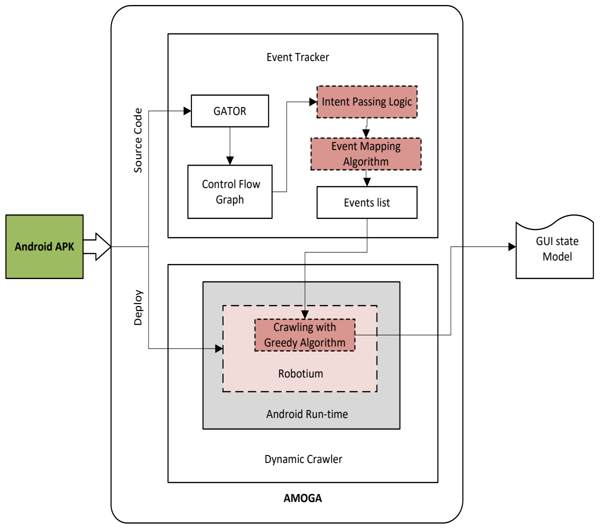

Figure 1: The proposed framework

In the first stage, we leverage GATOR [33], a static analysis tool to analyze the source code of an app. It is suitable for control flow analysis of traditional java applications and Android apps [13]. This is due the observation that life cycle callback define major changes to the visible states of activities, dialogues, and menus, and to the possible runtime events. The tool constructs callback control flow graph from app's source code. Our event mapping algorithm uses the generated graph as input to generate a list of all events (user and system events) supported by an application.

The event mapping algorithm traverse the graph to identify all edges in the graph in line 3 and for all edges it will backtrack to identify the source of event in line 5 of the algorithm. For all the sources (nodes), the algorithm extracts the triggers of events in line 7 and further extracts the views and ids of the views in line 8-9 and adds it to the event list. The output (list of events) from the algorithm will be used as input for the dynamic exploration in second stage of our approach. The second stage involves run-time exploration of an app to records it's states using Robotium framework [34] which is specifically designed for the testing of Android applications. Robotium uses crawler to identify and fire GUI events on a running application. GUI crawlers usually used Depth First Search (DFS) or Breadth First Search algorithms. However due to the limitation with DFS and BFS algorithms in exploring graphs such as selecting which path to explore and when the processing time is limited [35], we proposed greedy algorithm to improve the exploration. The algorithm will read the events list from the event mapping algorithm and dynamically exercise all events on the running app to explore the states of events.

Algorithm 1. Event mapping

```
Input: CCFG $g$ = (N,E)
Output: eventsList
1  Procedure EventMapping(g)
2      EventsList $el$ ← EdgesInGraph()
3      Edge ← getAllEdges(g)
4      for all edges $E \in$ EdgeSet do
5          eventHandler $h$ ← getSourceOfEdge($E$)
6          foreach $h \in$ evenHanlerSet do
7              t ← find trigger(h)
8                  v ← getwidget(t)
9                  id ← getParameter(v)
10                 $el$.add(E, id)
11         end
12     end
13 end
```

V. Results

Our approach will be validated on a number of different mobile applications from the Google play store. Good candidates to be assessed by the approach are Android native apps. In our experiments, we are using apps that are used by previous techniques [14, 15] for the analysis and the result will be compared with the results from existing techniques. We generated control flow graph of an application using GATOR. The event mapping algorithm is implemented on the graph to traverse and extract all events with their ids. We have generated a comprehensive list of events supported by an application which is represented as a queue. This will further be used as input for the run-time exploration in the dynamic analysis stage. The results from the static analysis stage indicated that the approach is promising for the generation high quality models.

VI. Conclusion

The popularity of model-based testing is ever increasing nowadays. In view of this, generation of high quality models from apps is necessary to support test automation for mobile apps. In this paper we have proposed a static/dynamic hybrid approach for the reverse engineering of mobile apps that has the potential to explore and app to generate high quality model that can be used for model-based testing of mobile apps. At this level we have extracted a comprehensive list of events from an app using static analysis on the source code. Our next state is to dynamically run these events on an app to automatically explore its states transition.

Acknowledgement

We would like to acknowledge the support from UTHM in undertaking the research, under the Graduate Research Incentive Grants (GIPS), Vote U308, Universiti Tun Hussein Onn Malaysia.

References

[1] Yang, S., et al. Static control-flow analysis of user-driven callbacks in Android applications. in International Conference on Software Engineering (ICSE). 2015.

[2] Gartner, I., "Worldwide Smartphone Sales", June, 2015. http://www.gartner.com/newsroom/id/3061917.

[3] Nayebi, F., J.-M. Desharnais, and A. Abran. The state of the art of mobile application usability evaluation. in CCECE. 2012.

[4] Gartner, I., Mobile Apps Will Be a Vehicle for Cognizant Computing, June 2015. http://www.gartner.com/newsroom/id/2654115.

[5] Minelli, R. and M. Lanza. Software Analytics for Mobile Applications-Insights & Lessons Learned. in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on Software Maintenance and Reengineering. 2013.

[6] Islam, R., R. Islam, and T. Mazumder, Mobile application and its global impact. International Journal of Engineering & Technology (IJEST), 2010.

[7] Wasserman, A.I. Software engineering issues for mobile application development. in Proceedings of the FSE/SDP workshop on Future of software engineering research. 2010. ACM.

[8] Muccini, H., A. Di Francesco, and P. Esposito. Software testing of mobile applications: Challenges and future research directions. in 7th International Workshop on Automation of Software Test (AST),. 2012. IEEE.

[9] Payet, É. and F. Spoto, Static analysis of Android programs. Information and Software Technology, 2012. 54(11): p. 1192-1201.

[10] Hu, C. and I. Neamtiu. Automating GUI testing for Android applications. in Proceedings of the 6th International Workshop on Automation of Software Test. 2011. ACM.

[11] Bhattacharya, P., et al. An empirical analysis of bug reports and bug fixing in open source android apps. in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on. 2013. IEEE.

[12] Enck, W., et al., TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 2014. 32(2): p. 5.

[13] Rountev, A. and D. Yan, Static Reference Analysis for GUI Objects in Android Software, in Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. 2014, ACM: Orlando, FL, USA. p. 143-153.

[14] Azim, T. and I. Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. in Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications. 2013. ACM.

[15] Yang, W., M.R. Prasad, and T. Xie, A grey-box approach for automated GUI-model generation of mobile applications, in Fundamental Approaches to Software Engineering. 2013, Springer. p. 250-265.

[16] Dehlinger, J. and J. Dixon. Mobile application software engineering: Challenges and research directions. in Workshop on Mobile Software Engineering. 2011.

[17] Janicki, M., M. Katara, and T. Pääkkönen, Obstacles and opportunities in deploying model-based GUI testing of mobile software: a survey. Software Testing, Verification and Reliability, 2012. 22(5): p. 313-341.

[18] Young, M., Software testing and analysis: process, principles, and techniques. 2008: John Wiley & Sons.

[19] Kull, A. Automatic GUI Model Generation: State of the Art. in Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on. 2012.

[20] Amalfitano, D., et al., Using GUI ripping for automated testing of Android applications, in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012, ACM: Essen, Germany. p. 258-261.

[21] Aho, P., et al. Automated Java GUI Modeling for Model-Based Testing Purposes. in Information Technology: New Generations (ITNG), 2011 Eighth International Conference on. 2011.

[22] Cipresso, T. and M. Stamp, Software Reverse Engineering, in Handbook of Information and Communication Security. 2010, Springer. p. 659-696.

[23] Grilo, A.P., A.R. Paiva, and J.P. Faria. Reverse engineering of GUI models for testing. in Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on. 2010.

[24] Memon, A., I. Banerjee, and A. Nagarajan. GUI ripping: Reverse engineering of graphical user interfaces for testing. in 10th Working Conference on Reverse Engineering (WCRE 2003). 2003. IEEE Computer Society.

[25] Nguyen, B., et al., GUITAR: an innovative tool for automated testing of GUI-driven software. Automated Software Engineering, 2014. 21(1): p. 65-105.

[26] Memon, A., AndroidGUITAR. http://sourceforge.net/apps/ mediawiki/ guitar/index.php?title=Android_GUITAR, 2011.

[27] Joorabchi, M.E. and A. Mesbah. Reverse engineering iOS mobile applications. in Reverse Engineering (WCRE), 2012 19th Working Conference on. 2012. IEEE.

[28] Mesbah, A., A. van Deursen, and S. Lenselink, Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes. ACM Trans. Web, 2012. 6(1): p. 1-30.

[29] Amalfitano, D., A.R. Fasolino, and P. Tramontana. A gui crawling-based technique for android mobile application testing. in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. 2011. IEEE.

[30] Silva, C.E. and J.C. Campos, Combining static and dynamic analysis for the reverse engineering of web applications, in Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems. 2013, ACM: London, United Kingdom. p. 107-112.

[31] Coimbra Morgado, I., A.C. Paiva, and J. Pascoal Faria, Dynamic Reverse Engineering of Graphical User Interfaces. International Journal On Advances in Software, 2012. 5(3 and 4): p. 224-236.

[32] Aho, P., T. Raty, and N. Menz. Dynamic reverse engineering of GUI models for testing. in Control, Decision and Information Technologies (CoDIT), 2013 International Conference on. 2013.

[33] GATOR: Program Analysis Toolkit For Android. web.cse.ohiostate.edu/presto/software/gator.

[34] GoogleCode, Robotium. http://code.google.com/p/robotium.

[35] Salva, S. and S.R. Zafimiharisoa, Model Reverse-engineering of Mobile Applications with Exploration Strategies. In Proceedings of the 9th International Conference on Software Engineering Advances (ICSEA), October 12-16, 2014, Nice, France., 2014.

[36] Salihu, I.A. and R. Ibrahim, Comparative Analysis of GUI Reverse Engineering Techniques, in Advanced Computer and Communication Engineering Technology. 2016, Springer. p. 295-305.