A New Method for Automated GUI Modeling of Mobile Applications

Jing Xu[1](✉), Xiang Ding[1], Guanling Chen[1], Jill Drury[1], Linzhang Wang[2], and Xuandong Li[2]

[1] University of Massachusetts Lowell, Lowell, MA, USA
{jxu,xding,glchen,jdrury}@cs.uml.edu
[2] Nanjing University, Jiangsu, China
{lzwang,lxd}@nju.edu.cn

Abstract. It is often necessary to construct GUI models for automated testing of event-driven GUI applications, so test cases can be generated by traversing the GUI models systematically. It is, however, difficult to apply traditional modeling techniques directly for mobile platforms as common static models cannot reflect application behaviors under different contexts. To address these challenges, we propose a novel approach for automated GUI modeling of mobile applications and introduce our unique definition of GUI state equivalence, which can reduce state space and facilitate model merging. The proposed modeling method can already discover subtle implementation issues. Real-world case studies show that the proposed approach is effective for adaptive GUI modeling on the Android platform.

Keywords: Automated modeling · Mobile applications · Android · Contextual behaviors

1 Introduction

Mobile apps are playing an increasingly important role in ubiquitous computing environments and cost-effective solutions are urgently needed to support the testing of mobile apps. Mobile apps are a subset of the more general class of event-driven Graphical User Interface (GUI) applications [1] and are often context aware [2]. Traditional GUI modeling techniques typically assume a static model and fall short on context-aware apps that have dynamic behaviors and are not just driven by GUI events.

In this paper, we present a new approach to automatically model GUIs for mobile apps and we focus on the Android platform. We propose a *Coarse Grained GUI Model* (CGGM) based on state machines. CGGM only uses the unique

This work is supported partly by the National Science Foundation under Grant No. 1016823 and 1040725. Any opinions, findings, and conclusions expressed in this work are those of the authors.

I. Stojmenovic et al. (Eds.): MOBIQUITOUS 2013, LNICST 131, pp. 688–693, 2014.
DOI: 10.1007/978-3-319-11569-6_54

composition feature of a screen when identifying a GUI state. By not including a comprehensive set of properties of UI elements (thus called coarse grained), the GUI model is compact yet comprehensive; it also allows easy and meaningful aggregation of multiple models obtained through different runs, such as under different contextual situations.

2 Definition of Android Coarse-Grained GUI Model (CGGM)

To provide a cost-effective modeling solution for Android apps, the first major concern comes from the scaling issue of state machine models (the state explosion problem) [3]. To overcome that, we define the state of the GUI by emphasizing the group composition of GUI elements on the screen to identify a GUI state. The second concern is that most current GUI modeling techniques employ a static model. Therefore, they are unable to take into account context-based execution behaviors of GUIs. To tackle this problem, we perform multiple GUI rippings under different contexts. Taking advantage of our unique definition of GUI state equivalence, we can merge these models and obtain an adaptive model to include as many context-based execution behaviors as possible.

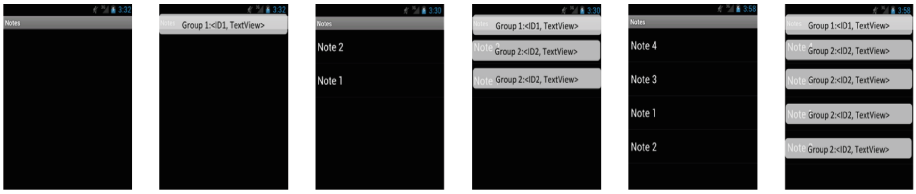
The level of model granularity is determined by how to define equivalence of two screen views. In previous research, the GUI state of a screen view (or a window for desktop apps) is usually defined as a set of triples $\{(w_i, p_j, v_k)\}$, where w_i is the widget in current window, p_j is the property of the widget and v_k is the value of the property and any change of any w_i, p_j or v_k would be considered to be a different state. In CGGM, each state represents a unique screen view that users interact with. We define the equivalence of two screen views in a way that if the composition of two screen views are the same, the two screen views are equivalent. To describe the composition of a screen view, we introduce the following definitions.

For Android app GUIs, *android:id* and *android:visibility* are two important properties of a View object on the screen. Android IDs are integer identities (IDs) associated with View objects and are used to find specific View objects within the hierarchy structure of a given GUI. If two GUI widgets on the screen share the same Android ID, they must also have similar behaviors. And we only deal with visible widgets that users could interact with.

Definition: A visible widget is a GUI widget that it is a View object, not a ViewGroup object, all its ancestors are visible and its visibility is set to VISIBLE.

Definition: Two widgets are ***siblings*** on a screen view if and only if they are both visible to users, of the same Android ID, of the same type (Android View type), and have the same structure of ancestors. Sibling widgets usually have common behaviors when interacting with users.

Definition: All siblings form a ***group*** on the screen view. The composition of a screen view is determined by widget groups. The most important property of a group is the homogeneity of widgets.



(a) screen1 (b) screen1 (c) screen2 (d) screen2 (e) screen3 (f) screen3

Fig. 1. Notepad screen views and their corresponding group composition

Definition: All sibling widgets in a group are of the same type, which is also the ***type of the group***.

Definition: A ***screen view*** is composed by a set of ***groups***.

Definition: The set of types of all groups on the screen describes the ***pattern*** of a screen view. As shown in Fig. 1(d), the pattern of screen view 1(c) is {TextView, TextView}.

Definition: If two groups from different screen views are in the same activity, of the same type and with the same Android ID for its elements, there is a ***mapping*** between these two groups.

As shown in Fig. 1(b), (d), (f), the first group of each screen view is a mapping to each other. They are actually the same widget on the screen but other parts of the screen view have changed.

Definition: Two ***equivalent screen views*** must be in the same activity, with the same number of groups, of the same pattern, and there is a one-to-one mapping for groups in the two screen views.

Definition: Two GUI states are equivalent if and only if the screen views they represent are equivalent.

3 Construction of Android Coarse-Grained GUI Model (CGGM)

Android mobile apps accept not only GUI events but also contextual events (such as network quality changes) [4]. To ensure the completeness of the model, we first get models by executing GUI ripping in different contexts, aiming at exploring varied execution behaviors caused by contextual changes. Then, models from different runs of GUI ripping are put together to examine inconsistent state transitions. Those inconsistencies can provide feedback to successive modeling process and are considered to be indicators of potential software deficiencies or context-based execution behaviors.

In each round, the GUI ripping process is driven by executing GUI events on widgets to invoke screen view changes or activity transitions. It would terminate normally when all scheduled GUI events are exercised according to the traversal

algorithm. However, three break conditions can also stop the current ripping process: (1) another app is invoked from the AUT (Application Under Test), since our toolkit uses Robotium APIs that cannot get GUI structural data from a second app; (2) the app crashes due to certain *actions*; or (3) the screen view is stuck for more than 5 min.

Between each round, we manipulate application context before ripping and perform inconsistency examination between models. Inconsistent behaviors of the same or sibling widgets in different models can help find incorrect event logic or contextual application behaviors. The application and system status change during different runs of GUI ripping, which could help discover more hidden states. Most importantly, the CGGM enables a relatively high percentage of equivalent states in different models, which makes the model merging process feasible and also meaningful. By merging GUI models from different contexts, we can actually construct a dynamic model which is open to revision.

4 Case Studies

In this section, we show two examples of using the proposed method to model Android apps - Nihao and Ohmage. Nihao is a personalized intelligent app launcher which dynamically recommends apps the user is most likely to launch [5]. Ohmage is an participatory sensing platform which supports mobile phone-based data capture [6,7]. Both apps are sensitive to contextual information such as location, app usage history, user's preference, and network conditions.

4.1 Nihao

For the main activity of Nihao, as shown in Fig. 2(a), 7 constant screen views that were not equivalent to each other were found during the process. One screen view was not captured and a final model consisting of 13 states is established, including 2 external states indicating a second app is invoked and 4 contextual states discovered by inconsistent state transitions under different contexts.

Nihao allows users to set the scope of apps and change ranking layout between Grid view and List view. Though the listed apps might change, the group composition of all the initial screen views are the same, thus they are all equivalent. As shown in Fig. 2, they all contain 10 groups in total where Group $1 \sim 7$ are static, Group 8 is a textview, Group 9 are app icons and Group 10 are app names. Even if users choose a different view (grid or list), the CGGM model can still adapt to such a significant UI change, as shown in Fig. 2(b) and (c).

4.2 Ohmage

We modeled major components of Ohmage, as shown in Table 1. Components were modeled separately and could be reassembled to establish a model for the whole app. We stayed logged in while performing GUI ripping with the purpose of getting a re-visitable initial state.

(a) (b) (c)

Fig. 2. Equivalent initial screen views of Nihao

Table 1. Inconsistent transitions in components

Components	States	Inconsistencies
Campaigns	23	6×2
Surveys	18	$2 \times 2 + 2 \times 3$
Response history	8	1×2
Upload queues	5	$1 \times 2 + 1 \times 3$
Help	3	0

As shown in Table 1, context-based transitions were counted for each component model. In the last column, $x \times y$ means that there are x places in the model that have y options of transitions depending on the context. Three types of sources have led to the varied transitions. First, the transition could be subject to updates of content presented. Such updates could be invoked by either actions triggered from GUIs or updates of sources on the ohmage server. Second, network conditions influenced execution behaviors. For example, if an "upload" event was triggered under a bad network condition, ohmage would present a dialogue with options "Retry Now, Retry Later, Delete" to deal with the situation. In good network conditions, such behaviors were hidden from GUIs. Finally, different exception handlers also resulted in various transitions.

5 Conclusions

The proposed method is proved to be an effective way to generate adaptive GUI models for Android mobile apps aware of contexts. The CGGM model could cover most of the functionalities of mobile apps and the results of model merging were also satisfying in terms of real context-based application behaviors discovered. The design of our ripping process, from the pure interaction perspective as of a real user's, could help find defective design of the layouts and support dynamically created UI elements.

References

1. Yang, W., Prasad, M.R., Xie, T.: A grey-box approach for automated GUI-model generation of mobile applications. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 250–265. Springer, Heidelberg (2013)
2. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Dartmouth College, Technical report TR2000-381, November 2000. ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.pdf
3. Graphical User Interface Testing Wiki Page. http://en.wikipedia.org/wiki/Graphical_user_interface_testing
4. Liu, Z., Gao, X., Long, X.: Adaptive random testing of mobile application. In: ICCET (2010)
5. Zhang, C., Ding, X., Chen, G., Huang, K., Ma, X., Yan, B.: Nihao: a predictive smartphone application launcher. In: Proceedings of MobiCase (2012)
6. Ramanathan, N., Alquaddoomi, F., Falaki, H., George, D., Hsieh, C., Jenkins, J., Ketcham, C., Longstaff, B., Ooms, J., Selsky, J., Tangmunarunkit, H., Estrin, D.: Ohmage: an open mobile system for activity and experience sampling. In: Proceedings of PervasiveHealth (2012)
7. Ohmage Homepage. http://ohmage.org/