Model Dressing for Automated Exploratory Testing

Mehmet Cagri Calpur[*], Sevgi Arca[†], Tansu Cagla Calpur[‡] and Cemal Yilmaz[§]
Faculty of Engineering and Natural Sciences
Sabanci University, Istanbul, Turkey
[*]mehmetcagri@sabanciuniv.edu, [†]sevgiarca@sabanciuniv.edu
[‡]tcagla@sabanciuniv.edu, [§]cyilmaz@sabanciuniv.edu

***Abstract*—Automation of software testing is a complex problem with multiple facets to be handled in sync to be viable. In this work we propose two novel concepts; *model dressing* for *automated exploratory testing*. Model dressing maps an application under test to a model created for the domain of the application. In its simplest form, the domain model defines the business tasks as well as the user actions that can be carried out from the perspective of the end-user. Automated exploratory testing leverages the domain knowledge as well as the experience gained from testing applications in the same domain to test another application.**

***Keywords*—Exploratory Testing, Model-Based Testing, Software Domain Analysis, Application Text Mining, Automated Testing, Domain Classification**

I. Introduction

Applications that run on mobile devices are usually GUI based thin-clients of varying complexities. User interactions are of physical nature focusing on using the application via interacting with the screen and the physical buttons on mobile devices. Consequently, systematic testing of mobile applications typically requires simulation of the actions that would normally be conducted by a regular user.

A regular user is a sentient human being, capable of intuitively determining how to interact with the application and find out what kind of data is required by the application. The main research question we have in this work is whether such knowledge can automatically be discovered, such that testing approaches, which have some domain knowledge, can be developed to efficiently and effectively test previously unseen applications of the domain in a way that can be regarded as *automated exploratory testing*.

II. Approach

To this end, the very first step is to automatically classify the domain of the application under test. Since determining the domain would let the automated testing agent be aware of the application context, it would help the agent narrow down both the input space and the test concerns.

Once the domain is automatically determined, the knowledge about the domain is represented in the form of a model. In its simplest form, this model defines both the business tasks and the set of actions that can be performed by an end-user from the perspective of the user. We, initially chose to express such a model using finite state machines. The model can further be partitioned according to the classification of resources that constitute the program and its sub-units, the activities. Figure 1 presents as an example a partial model for the airline ticket reservation domain.

Given a domain model and an application under test, the goal is then to automatically map the application under test to the model, so that the test cases that are generated using the domain model and that represent the knowledge of the business as well as development experts in the domain, can automatically be executed against the previously unseen application under test. We call the former part *model dressing* and the latter part *automated exploratory testing*. Note that model dressing lets an automated testing agent to use the test cases developed for testing an application for testing another application in the same domain. We believe that the proposed approach is of practical importance because with the increasing number of cloud-based testing services, it is quite vital for such services to 1) learn from testing each and every application and 2) to carry out the knowledge learned from testing one application to do a better job of testing another application.

The proposed approach requires to map the states, state parameters, and transitions in the model to, for example, screens, input fields on the screens, and the user actions, respectively. To this end, we have been developing data mining and machine learning-based approaches. In particular, we use the data available directly from outside the applications to perform the mapping. For example, we automatically collect all the natural language text descriptions appearing in an application, on a screen, or about an input field and analyze the collected data by using some natural language processing techniques to determine the domain of the application as well as to map the screens and input fields to the states and the state parameters of the domain model.

We use Android platform as our initial experimentation platform. The proposed approach, however, is independent of the hardware and software platforms used to develop the application under test. An android application consist of programs that are called activities. Even though the activities may rely on the output from other activities, the lifetime of activities are independent of each other. The application packages store information and resources for each activity and the activities as well as the relationships between activities are constructed with this information. Independent nature of these activities makes them suitable candidates for defining them as the states constituting the finite state model of the application. The configuration of the activity may change but

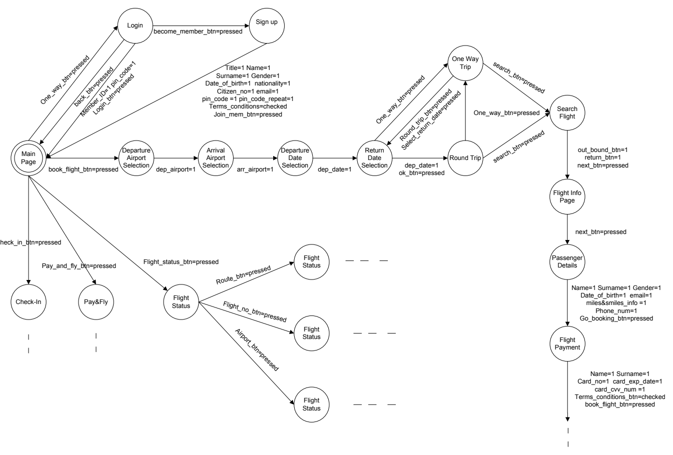

Fig. 1. The formal model of a commercial airliner mobile application.

the general business task handled by the activity generally stays the same. This means that an activity can be regarded as a ”Meta-State” and the sub-configurations of the activity as the sub/inner-states of the activity. The initial configuration upon reaching an activity can then be defined as the default state. Some action and input combinations may lead to transition to another state (activity), while others may not result in a state change. A transition can be defined as reaching a state after a collection of user input and actions on an activity. A transition may cause a self loop, which means, the state (activity) does not change after the set of transition actions. There can be two cases for such transition. The first case is the invalid transition action, which is a collection of input or action items that are not handled by the mobile application. In this case the system does not capture the invalid action and stays in stand by mode waiting for a legitimate action. The second case is the transitions between sub-classes. In this case the activity, which represents the meta-state, does not change but the configuration of the state change resulting in a different sub-state. The set of components that can be interacted is modified, which also updates the required transition function. On this platform, therefore, the proposed approach, for example, maps a state of the domain model to an activity, a state parameter to an input field defined by the activity, and a transition to a tap (or any user gesture, such as swipe).

III. Related Work

Zhu et. al. [1] conducted GUI similarity research on large number of android applications. The data source for the research is gathered from the XML resource files in the application packages. Text on the application GUI, information in XML resources and image files are used for detection of application similarity. The similarity metric is then used for plagiarism detection and insertion of malicious content.

He et. al. [2] processes xml files that define Java application GUIs for screen similarities in order to find out similar interfaces. The idea is to select test cases from dissimilar GUI formations to increase the diversity of test suite.

Hallenberg et. al [3] engages the need for domain knowledge for appropriate test case generation and proposes that there is a gap between the focus of quality assurance professionals testing focus, which is based on more technical problems of the application, an the business domain requirements. Their focus is on the GUI and the test actions that can be performed on the GUI is extracted from a program model.

Kowalczyk et. al. [4] inspect application behavior from the information that can be mined from the GUI and resource files of application package, as well as information from application store reviews and descriptions. The argument of the paper is that, when combined data gathered from these sources are sufficient identifying the behavior of the application.

IV. Conclusion

Pairing a domain model with processed textual information gathered from the application is the key to building an automated exploratory testing approach, mimicking domain experts. Currently, domain expert’s manual labor is required, leading to a loss of precious workforce time and proneness to human error.

References

[1] Jiawei Zhu, Zhengang Wu, Zhi Guan, and Zhong Chen, “Appearance similarity evaluation for android applications,” in *7th International Conference on Advanced Computational Intelligence*, March 27-29 2015.

[2] Zhi-Wei He, Cheng-Gang Bai, “Gui test case prioritization by state-coverage criterion,” in *IEEE/ACM 10th International Workshop on Automation of Software Test*, 2015.

[3] N. Hallenberg and P. L. Carlsen, “Declarative automated test,” in *Proceedings of the 7th International Workshop on Automation of Software Test*, ser. AST ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 96–102. [Online]. Available: http://dl.acm.org/citation.cfm?id=2663608.2663628

[4] E. Kowalczyk, A. M. Memon, and M. B. Cohen, “Piecing together app behavior from multiple artifacts: A case study,” in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2015, pp. 438–449.