Model-Based GUI Testing for HarmonyOS Apps

Yige Chen
Southern University of Science and Technology
Shenzhen, China
12011625@mail.sustech.edu.cn

Sinan Wang
Southern University of Science and Technology
Shenzhen, China
wangsn@mail.sustech.edu.cn

Yida Tao
Southern University of Science and Technology
Shenzhen, China
taoyd@sustech.edu.cn

Yepang Liu
Southern University of Science and Technology
Shenzhen, China
liuyp1@sustech.edu.cn

ABSTRACT

HarmonyOS is a new all-scenario operating system. As its software ecosystem rapidly expands, how to conduct automated testing of HarmonyOS apps to ensure app quality has become a crucial task. Model-based testing has been shown to be an effective method for automatic Android app GUI testing. Inspired by previous work, we in this work explore how to perform model-based testing for HarmonyOS apps. To characterize app behaviors, we first propose the *page transition graph* model, which is a directed graph describing transitions between various UI pages in a HarmonyOS app. We then devise a static analysis method to build page transition graphs from the source code of HarmonyOS apps. Leveraging the model, we implement a testing tool which can effectively perform systematic GUI exploration in HarmonyOS apps. We have evaluated our tool using 10 popular open-source HarmonyOS apps from GitHub and Gitee. Experimental results show that the extracted models are highly precise. Moreover, within the same time budget, model-based testing significantly improves the test coverage of HarmonyOS apps over a random baseline method. Our tool is open-sourced at **https://github.com/sqlab-sustech/HarmonyOS-App-Test** and a video demo is at **https://youtu.be/dgZWkHiBYbA**.

KEYWORDS

HarmonyOS, Model-Based Testing, Page Transition Graph

ACM Reference Format:
Yige Chen, Sinan Wang, Yida Tao, and Yepang Liu. 2024. Model-Based GUI Testing for HarmonyOS Apps. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695364

1 INTRODUCTION

With the widespread use of smart devices, the quality assurance of mobile apps have become crucial issues in software development. HarmonyOS, a newly-launched full-scenario distributed operating system, has quickly captured market share due to its powerful cross-platform capabilities. Over 4,000 apps and 800 million devices are now part of the HarmonyOS ecosystem. However, with the rapid expansion of the HarmonyOS app ecosystem, effectively ensuring app quality, especially in the field of automated testing for HarmonyOS apps, has become a key issue.

Currently, as HarmonyOS becomes more widespread, developers need effective tools and methods to ensure the quality of apps. However, due to the lack of relevant work on HarmonyOS app testing in both academia and industry, research in this area needs to draw on the experiences from Android app testing. In the field of Android automated testing, *model-based testing* is a widely used technique that leverages models to assist the execution and management of testing activities, which improves test coverage and promotes testing automation. In Android, these models are often referred to as window/activity transition graphs [11], which describe the transition relationships between different activities through certain UI events.

Inspired by previous research, in this paper, we explore how to perform model-based GUI testing for HarmonyOS apps. Due to the differences in app design principles, development languages and other aspects between Android and HarmonyOS, existing model-based testing methods for Android apps cannot be directly applied for HarmonyOS apps. To address this gap, we propose a page transition graph (PTG) model for capturing the behaviors of HarmonyOS apps. A PTG is a directed graph describing transitions between various HarmonyOS UI pages, which can be constructed through static analysis. Since HarmonyOS is primarily written in ArkTS, a JavaScript-compatible language, we build a PTG by traversing the abstract syntax tree (AST) and the call graph (CG) with existing JavaScript language toolchain. Then, we locate specific APIs to capture UI events related to page transitions. Based on the PTG, we further devise a general GUI testing tool for HarmonyOS apps by applying the automated testing framework *arkxtest* [4]. This tool can be easily extended to support various exploration strategies. As a proof of concept, we have implemented a depth-first search (DFS) strategy for demonstration and preliminary evaluation.

To show the effectiveness of our tool, we run it on a HarmonyOS phone emulator to test 10 popular open-source HarmonyOS apps. We first analyze the quality of constructed PTG models. It shows that the models do not contain any wrong nodes or edges (i.e., no false positives) but can be incomplete in some cases (i.e., there are false negatives in terms of node/edge identification) due to the

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695364

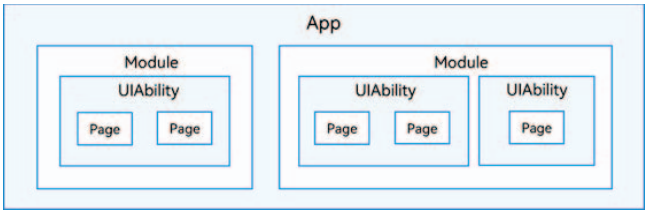

Figure 1: Stage model of HarmonyOS apps

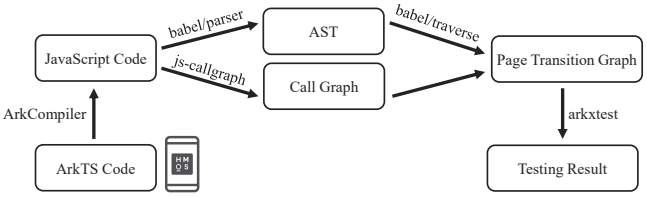

Figure 2: Overview of model-based GUI testing for HarmonyOS apps

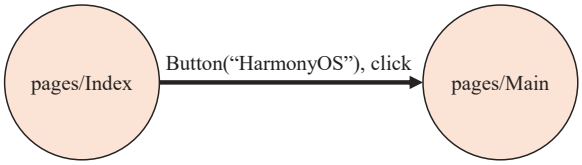

Figure 3: Page transition graph example

dynamic language features of JavaScript. We then assess the testing performance of our tool. The results show that, within the same time budget, model-based testing can significantly improve the test coverage of HarmonyOS apps compared to random testing, which is essentially unguided.

To summarize, our work makes a first attempt on effective model-based testing for HarmonyOS apps. We hope that by open-sourcing our tool, more researchers and practitioners can join and contribute to the exciting realm of HarmonyOS app testing.

2 BACKGROUND

2.1 HarmonyOS App Model

HarmonyOS employs a *stage model* for app development, whose general package structure is displayed in Figure 1. This model specifies that a HarmonyOS app can be developed in multiple modules, with each module containing multiple `UIAbility` components. `UIAbility` is a built-in class in the HarmonyOS app development framework, which is designed for creating UI pages. According to the system design of HarmonyOS, a `UIAbility` component corresponds to a unique task in the task list on the system. Each `UIAbility` component can contain multiple ArkUI pages, which can transit to each other using the built-in `router` APIs.

2.2 ArkUI

ArkUI is the UI development framework for building HarmonyOS apps. ArkUI comes with two development paradigms: ArkTS-based declarative development and JavaScript-compatible web-like development. The official documentation recommends using the declarative development paradigm for the latest HarmonyOS apps. The declarative developement paradigm uses ArkTS, a superset of the TypeScript language with declarative UI syntax, to develop HarmonyOS apps. ArkTS is compiled by the ArkCompiler to generate bytecode files, which can be executed on the Ark runtime.

3 APPROACH

3.1 Overview

Figure 2 shows the workflow of our proposed model-based testing tool. It consists of two major steps, static-analysis-based PTG construction and model-based testing automation. Since there is no static analysis tool for ArkTS now, we first obtain JavaScript code from ArkTS code using ArkCompiler. Then we apply static analysis to build PTGs. Specifically, we use JavaScript parser *babel.js* [2] to generate AST and use *js-callgraph* [1] to generate JavaScript CG. Based on the AST and the CG, we further analyze and obtain the transition relationships between ArkUI pages, which form the PTG model. Finally, we conduct model-based testing using the HarmonyOS testing framework *arkxtest*, based on the constructed PTG model.

3.2 Building Page Transition Graphs

PTG is a type of GUI model that represents page transitions. It is a directed graph $G = (V, E)$, where the nodes V represent different ArkUI pages, and the edges E represent possible page transitions between these nodes. Listing 1 shows a snippet of `UIAbility` program written in ArkTS, and Figure 3 is its corresponding PTG graph model. Specifically, the source code from Line 7 to 9 defines that clicking the button named "HarmonyOS" will navigate from the current page *pages/Index* to another page *pages/Main*. Such a page transition in the model corresponds to an edge from node *pages/Index* to the node *pages/Main* in Figure 3.

```
1  @Entry
2  @Component
3  struct Index {
4    build() {
5      Row() {
6        Column() {
7          Button('HarmonyOS').onClick(() => {
8            router.pushUrl({ url: 'pages/Main' });
9          })
10       }.width('100%')
11     }.height('100%')
12   }
13 }
```

Listing 1: An ArkTS code snippet from a HarmonyOS app

To build a PTG for a HarmonyOS app by static analysis, we first parse the JavaScript program compiled from ArkTS source code and get the AST representation of the program. Then, we identify some APIs related to page transitions such as `router.pushUrl`. We traverse the ASTs to locate such APIs and extract information of the ArkUI pages (e.g., page name) that the current page will transit to. Based on the specific location, we continuously inspect the outer functions to find out whether there are corresponding components and events. To handle possible nested function calls, we traverse the CG to retrieve function call chains. After getting all components

Algorithm 1 Model-based testing workflow

Input: page transition graph *PTG*

```
1:  repeat
2:    curPage ← router.getPage()
3:    componentInfo, event ← chooseEdge(PTG, curPage)
4:    components ← findComponentsBy(componentInfo)
5:    for each component of components do
6:      component.action(event)
7:      newPage ← router.getPage()
8:      if newPage != curPage then
9:        break
10:     end if
11:   end for
12: until time budget is exhausted
```

and UI events related to page transitions, we will obtain a PTG as illustrated in Figure 3.

3.3 Model-Based Testing

Based on the constructed PTG model, we leverage the HarmonyOS automated testing framework arkxtest to realize model-based testing. Algorithm 1 illustrates the workflow of our model-based testing approach, which uses the PTG generated in the previous step as the input. The algorithm starts from the initial ArkUI page loaded by current `UIAbility` component. First, we get the current page name (line 2). Next, we choose which edge on the PTG to execute, including the information of component and UI event, by using a specific strategy implemented by the `chooseEdge` method (line 3). Since the current page may contain many components with the same information, including type, text content and so on, we find all these components by calling the `findComponentsBy` method to filter components based on specific conditions and iterate through them (line 4). For each specific component, we execute the corresponding action (line 6). If the page transitions to a new page, we exit the current loop and proceed to the next edge selection (line 9). The steps above will repeat until the time budget is exhausted (line 12).

It is worth mentioning that our algorithm is generic and can support different testing strategies by overwriting the `chooseEdge` method. As a proof of concept, we have implemented a depth-first search strategy for demonstration.

4 EXPERIMENT

4.1 Subject HarmonyOS Apps

We selected 10 popular open-source HarmonyOS apps from GitHub[1] and Gitee[2] for our experiment. Table 1 shows the basic information of these apps. The apps' repository links are listed on our tool publicity repository. We selected these apps because they have multiple ArkUI pages with transitions between them, which is suitable for evaluating the performance of a model-based technique.

[1]https://github.com/
[2]https://gitee.com/

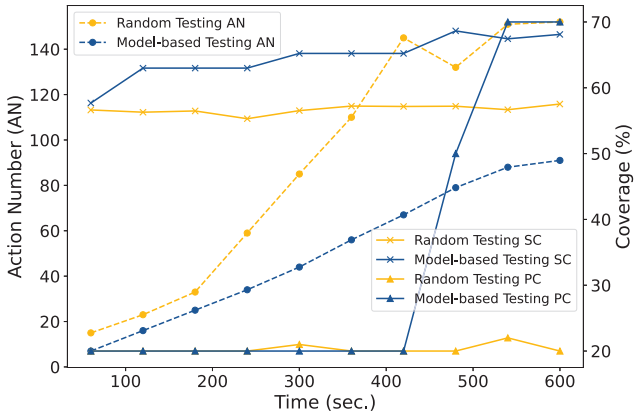

Figure 4: Comparison between random testing and model-based testing

4.2 Evaluation on Model Quality

To evaluate the quality of the constructed PTG models, we used the following two metrics:

- *Precision* (*Prec.*): The number of correct edges detected (*TP*) / Total number of detected edges (*TP* + *FP*). *Prec.* measures the ratio of edges that exist in the app among all the detected edges.
- *False Negative Rate* (*FNR*): The number of missing edges (*FN*) / Total number of actual edges (*TP* + *FN*). *FNR* measures the proportion of edges that are missing from the constructed model but actually exist in the app.

Table 1 shows the model qualities for the subject apps. As can be seen, the models obtained using our analysis process do not have false positives but do have a certain number of false negatives. For example, the model of HarmoneyOpenEye app has a false negative rate 43.75%. The primary reason for the false negatives is related to language issues. Currently, there is no static program analysis tool for ArkTS, and we can perform static analysis only on the JavaScript code generated from ArkTS. JavaScript, being a dynamic language, poses challenges in analyzing certain type information of variables. For instance, using anonymous functions as function parameters can reduce the accuracy of the function call graph generation, which in turn affects model construction.

4.3 Model-Based Testing Versus Random Testing

To evaluate the effectiveness of model-based testing, we compared model-based testing with random testing by adopting several additional metrics. First, we counted the number of actions executed within the same time, denoted as Action Number (*AN*). Then, we measured a series of coverage metrics, including statement coverage (*SC*), which can be obtained via the testing reports produced by the instrumentation tool *nyc*[3]. Since we build the PTG based on ArkUI pages, we added another metric called page coverage (*PC*), which is calculated as the number of ArkUI pages accessed during testing divided by the total number of ArkUI pages.

We compared our model-based testing tool with a random baseline method using the app HarmoneyOpenEye, which has the most

Table 1: The quality evaluation of models generated from HarmonyOS apps

HarmonyOS App	#Files	LOC	Star	Fork	#Pages	TP	FP	FN	Prec.	FNR
HarmoneyOpenEye	68	4,013	272	52	10	9	0	7	100.00%	43.75%
Harmony-arkts-movie-music-app-ui	39	4,994	24	9	9	10	0	0	100.00%	0.00%
Codelabs/MultiShopping	59	6,544	1,500	795	6	4	0	6	100.00%	60.00%
biandan-satge	61	16,877	3	3	6	8	0	0	100.00%	0.00%
Msea_HarmonyOS	17	900	32	5	6	2	0	0	100.00%	0.00%
Codelabs/MultiDeviceMusic	55	4,483	1,500	795	5	4	0	3	100.00%	42.86%
oh-bill	18	1,551	55	11	4	3	0	0	100.00%	0.00%
open_neteasy_cloud	9	896	137	35	3	2	0	0	100.00%	0.00%
homework-tasklist-v2	22	2,261	5	3	3	2	0	1	100.00%	33.33%
ArkTS-wphui1.0	20	2,158	15	7	2	2	0	0	100.00%	0.00%

number of ArkUI pages among our subject apps list. Due to the randomness during the test process, we ran both random testing and model-based testing by 10 times on the same HarmonyOS phone emulator with the same testing time. The experimental results were then averaged to reduce bias.

The results of random testing and model-based testing on HarmoneyOpenEye app are shown in Figure 4. Compared to random testing, model-based testing significantly improves all coverage metrics within the same time even with less action numbers. For example, when the total testing time comes to 600 seconds, compared to random testing, model-based testing improves the statement coverage by 18.39% (= (68.11% − 57.53%)/57.53%) and the page coverage from 20% to 70%, while using only nearly half the action numbers. The reason is that compared to random testing which randomly executes some UI events on some components, model-based testing leverages the constructed PTG to perform targeted transitions, allowing for faster coverage of multiple pages and thereby improving the coverage metrics.

5 RELATED WORK

HarmonyOS is currently in the early stages of ecosystem development. Therefore, there have been only a few existing studies on this emerging operating system. Li et al. [8] outlined a notable research landscape for HarmonyOS, identifying several potential research directions and possible challenges, including GUI modeling and testing. The authors pointed out that, unlike Android apps where a single `Activity` component corresponds to an XML layout file, the design principle of HarmonyOS apps encourages developers to use a single `Ability` component to display multiple ArkUI pages. Consequently, constructing the transition relationships between pages in HarmonyOS apps is more complex.

Model-based testing methods were extensively studied for Android apps. These methods construct a model that represents the behavioral space of the app under test and generate test inputs based on this model. *AndroidRipper* [5] and *MobiGUITAR* [6] build the model dynamically over the user interface. A^3E[7] constructs activity transition graphs with taint analysis and proposes a depth-first exploration, which is the strategy we implement based on the static model. Yang et al. [11, 12] applies static analysis to build window transition graphs as models. *PROMAL* [9] constructs window transition graphs using program analysis and maching learning. *Fastbot2* [10] incorporates reinforcement learning algorithms into the dynamic GUI exploration process to enhance testing efficiency.

6 CONCLUSION AND FUTURE WORK

In this work, we implemented a model-based GUI testing tool for mobile apps on HarmonyOS. Our experiment results demonstrate it can outperform a random baseline approach by a considerable margin. There are also several potential improvements to our model-based testing approach in the future. For example, HarmonyOS currently does not support obtaining GUI trees at runtime. Once this feature is provided, we will be able to build app models dynamically and apply more advanced and intelligent algorithms for GUI testing.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China (Grants No. 62202213 and 61932021). We would like to thank the anonymous reviewers for their constructive comments, which have significantly contributed to enhancing the quality of the paper.

REFERENCES

[1] 2013. js-callgraph. https://github.com/Persper/js-callgraph.
[2] 2014. babel.js. https://github.com/babel/babel.
[3] 2015. nyc. https://github.com/istanbuljs/nyc.
[4] 2022. arkxtest. https://gitee.com/openharmony/testfwk_arkxtest.
[5] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M Memon. 2012. Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 258–261.
[6] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, and Atif M Memon. 2014. MobiGUITAR: Automated model-based testing of mobile apps. *IEEE software* 32, 5 (2014), 53–59.
[7] Tanzirul Azim and Iulian Neamtiu. 2013. 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*. 641–660.
[8] Li Li, Xiang Gao, Hailong Sun, Chunming Hu, Xiaoyu Sun, Haoyu Wang, Haipeng Cai, Ting Su, Xiapu Luo, Tegawendé F Bissyandé, et al. 2023. Software Engineering for OpenHarmony: A Research Roadmap. *arXiv preprint arXiv:2311.01311* (2023).
[9] Changlin Liu, Hanlin Wang, Tianming Liu, Diandian Gu, Yun Ma, Haoyu Wang, and Xusheng Xiao. 2022. ProMal: precise window transition graphs for android via synergy of program analysis and machine learning. In *Proceedings of the 44th International Conference on Software Engineering*. 1755–1767.
[10] Zhengwei Lv, Chao Peng, Zhao Zhang, Ting Su, Kai Liu, and Ping Yang. 2022. Fastbot2: Reusable automated model-based gui testing for android enhanced by reinforcement learning. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–5.
[11] Shengqian Yang, Haowei Wu, Hailong Zhang, Yan Wang, Chandrasekar Swaminathan, Dacong Yan, and Atanas Rountev. 2018. Static window transition graphs for Android. *Automated Software Engineering* 25 (2018), 833–873.
[12] Shengqian Yang, Dacong Yan, Haowei Wu, Yan Wang, and Atanas Rountev. 2015. Static control-flow analysis of user-driven callbacks in Android applications. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 89–99.