효과적인 모델 기반 안드로이드 GUI 테스팅을 위한 동일 화면 비교 기법

백영민, 홍광의, 이정현, 배두환
KAIST 전산학부
{ymbaek, gwangui.hong, chlee, bae}@se.kaist.ac.kr

A Screen Comparison Technique for Effective Model-based Android GUI Testing

Youngmin Baek, Gwangui Hong, Cheonghyun Lee, Doo-Hwan Bae
School of Computing, KAIST

요 약

안드로이드 어플리케이션(앱)의 기능 검사와 크래쉬 탐지 등을 위해 다양한 GUI (Graphical User Interface) 테스팅 기법이 사용되고 있다. 그 중 모델 기반(Model-based)의 GUI 테스팅은 GUI 모델을 이용해 테스트 케이스를 생성하기 때문에, 기법의 유효성(Effectiveness)은 기반 모델의 정확도에 의존적이다. 따라서 모델 기반 기법의 유효성 향상을 위해서는 명확한 모델의 추상화 기준과 생성 기법을 통한 정확한 모델 생성이 필요하다. 본 연구에서는 정확하고 효율적인 모델과 테스트 케이스 생성을 위하여, GUI 상태 간 동일 여부를 정밀하게 구분하는 계층적 화면 비교 기법을 제안한다. 또한, 기존 연구의 기법과의 비교 실험을 통해 기존 기법보다 유효한 모델을 효율적으로 생성함을 확인함으로써, 모델 기반 안드로이드 GUI 테스팅의 성능 향상 가능성을 제시한다.

1. 서 론

GUI 테스팅(Graphical User Interface Testing)이란 화면을 구성하는 버튼, 체크 박스 등의 GUI 요소들을 클릭, 롱 클릭과 같은 사용자 이벤트를 통해 실행하여, 프로그램이 올바른 행위의 수행 여부를 검증하는 기법이다[1]. 최근 들어 모바일 어플리케이션(앱) 수의 기하급수적 증가로 인해 신뢰성 있는 앱의 중요성이 증가하면서, GUI 테스팅 기법을 이용한 모바일 앱 검증 역시 다양하게 시도되고 있다. 특히 안드로이드 OS가 모바일 플랫폼 시장에서 큰 규모를 차지하면서, 안드로이드 앱을 대상으로 하는 다양한 GUI 테스팅 기법이 연구되고 있다[2][3][4].

본 연구는 여러 GUI 테스팅 기법 중 소스 코드 정보가 없는 안드로이드 앱을 대상으로 한 모델 기반(Model-based) GUI 테스팅에 초점을 둔다. 모델 기반 GUI 테스팅은 앱의 행위를 반영한 GUI 모델로부터 테스트 케이스를 생성하고, 이를 테스트 대상 앱(Application under test)에 적용하여 행위를 검사하는 기법을 말한다. 이는 랜덤 기법으로는 어려운 특정 기능 검사를 위한 유의미한 테스트 케이스 생성이 가능하고, 오류 발생 경로 추적이 용이하다는 장점이 있다[5][6]. 그러나, 모델 기반 기법의 성능은 테스트 케이스 생성을 위한 기반 모델에 의존적이기 때문에 정확하고 효율적인 모델을 생성하기 위해 많은 주의를 기울여야 한다.

첫째, 모델의 추상화 수준이 높아 생성한 모델이 접근 가능한 화면(GUI 상태)이나 사용자 이벤트를 철저하게 반영하지 못할 수 있다. 이러한 경우, 테스트 케이스를 통해 버그가 존재하는 화면(GUI 상태)과 이벤트를 검사할 수 없어 테스트 케이스의 유효성(Effectiveness)을 떨어뜨릴 수 있다. 이러한 문제는 액티비티(Activity)를 기반으로 GUI 상태를 구분하여 모델을 생성하는 연구/도구에서 확인할 수 있는데[7][8], 최근의 많은 앱이 그림 1과 같이 사용자 편의를 위해 하나의 액티비티 상에서 동적으로 여러 화면을 구성할 수 있도록 개발되면서, 액티비티는 모델의 GUI

* 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. R0126-15-1101, (SW 스타랩) 모델 기반의 초대형 복잡 시스템 분석 및 검증 SW 개발)

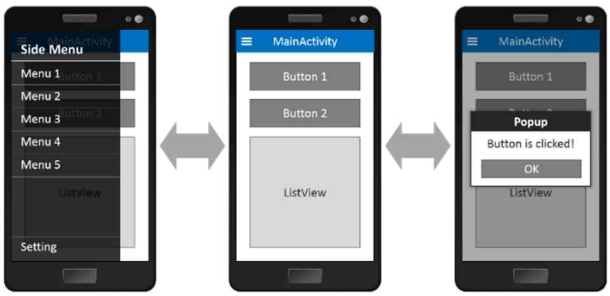


그림 1. 하나의 액티비티가 다수의 GUI 상태를 갖는 예

상태 구분의 기준이 되지 못하고 있다. 반대로 지나치게 구체적인 정보를 모델의 상태 구분 기준으로 사용할 경우[3], 모델이 동일한 행위를 갖는 화면들을 중복해서 표현하여 모델 및 테스트 케이스의 생성 효율을 떨어뜨릴 수 있는 문제를 가진다.

따라서, 모델을 정확하고 효율적으로 생성하기 위해서는 GUI 상태를 정밀하게 구분할 수 있는 기준과 기법이 필요하다. 본 논문에서는 이러한 문제를 해결하기 위한 모델 생성 기법에 초점을 두고, 정밀한 화면 비교 기법을 제안함으로써 모델 생성의 정확도를 향상시키고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 GUI 모델을 정의하고, 3장에서 모델 생성 기법과 화면 비교 기법을 소개한다. 4장에서는 기법을 이용해 생성한 모델을 기법을 분석하며, 5장에서 결론 및 향후 연구를 기술한다.

2. GUI 모델의 정의

모델 기반 GUI 테스팅의 핵심 요소는 앱의 행위를 상태 다이어그램으로 표현한 모델이며, 본 연구에서는 GUI 모델을 다음과 같이 정의한다:

> 독립적인 화면 정보를 담고 있는 GUI 상태(State)를 ***화면 노드 (Screen Node)***로 표현하고, 사용자 이벤트 실행을 통한 전이(Transition)를 ***이벤트 엣지(EventEdge)***로 표현한 그래프(Graph)

여기서 화면 노드와 이벤트 엣지를 통해 테스트 대상 앱의 모든 화면과 사용자 이벤트를 표현할 때, GUI 모델(이하 GUI 그래프)이 앱의 행위

를 충분히 반영한다고 볼 수 있다. 또한 모델은 각 화면 노드를 통해 독립적인 GUI 상태 정보를 표현해야 하는데, 이는 GUI 그래프가 중복 없이 구별되는 기능을 갖는 최소한의 화면 노드로 표현되는 것을 의미한다.

3. 화면(GUI 상태) 비교 기법

3.1 GUI 그래프 생성

소스 코드 정보가 없는 앱에 대해 GUI 그래프를 생성하기 위해서는 실행 중인 대상 앱으로부터 정보를 추출하여 분석하는 역공학적 접근이 필요하다. 기존 연구에서는 이를 위해 사용자 이벤트를 자동적으로 생성하여 테스트 대상 앱을 탐색하며 화면 정보를 수집하는 *크롤링(Crawling) 기법*을 사용했으며, 본 연구에서도 마찬가지로 자동 생성된 사용자 이벤트를 통해 위젯(Layout, Button, EditText 등)을 실행하며 앱을 탐색하는 크롤링을 통해 GUI 그래프를 생성한다.

그래프 생성 알고리즘은 다음과 같다. 우선 테스트 대상 앱을 실행하여 초기 화면의 정보를 수집하여 화면 노드를 생성하고, 실행 가능한 위젯에 대한 사용자 이벤트를 생성하여 *이벤트 큐(Event Queue)*에 저장한다. 이후 GUI 그래프는 "이벤트 큐에 저장되어 있던 사용자 이벤트를 꺼내서 실행 → 이벤트 발생 후 화면(S')의 추출 및 비교/분석 → 비교 결과에 따라 (a)기존에 탐색한 화면일 경우 이벤트 엣지만을 연결, 혹은 (b)새로운 화면일 경우 화면 노드 추가 및 이벤트 엣지 연결, 그리고 S'의 실행 가능한 사용자 이벤트를 이벤트 큐에 추가"의 크롤링 과정을 반복하며 생성된다. 알고리즘은 더 이상 큐에 남은 이벤트가 없을 때까지 대상 앱의 GUI 구조를 반복적으로 탐색/식별하며 그래프를 확장해 나가다가, 최종적으로 생성을 마친 그래프를 반환하고 종료한다.

3.2 화면 정보의 추출

본 기법은 PC와 안드로이드 기기 간 통신을 위해 ADB(Android Debug Bridge)를 통한 명령 전달 환경을 구축하고, 구글의 테스트 지원 라이브러리인 UIAutomator를 이용해 대상 기기의 화면 정보를 추출한다. UIAutomator는 기기의 화면을 분석하여 패키지(Package) 이름, 액티비티 이름, 그리고 화면 구성 위젯의 계층 구조를 제공하는데, 이를 통해 해당 화면을 구성하는 위젯 정보를 얻을 수 있다. 또한, UIAutomator는 각 위젯에 대해 다음과 같이 자세한 속성값을 제공한다:

```
<node bounds="[160,246][560,458]" selected="false" password="false"
long-clickable="false" scrollable="false" focused="false" focusable="true"
enabled="true" clickable="true" checked="false" checkable="false"
content-desc="" package="com.se.toyapp" class="android.widget.Button"
resource-id="" text="A Button" index="0"/>          ■: 이벤트 실행 가능 여부 속성
```

이를 통해 화면을 이루는 각 위젯의 정적인 정보(class, resource-id, text 등)와 이벤트 실행 가능 여부(clickable, long-clickable 등)에 대한 정보를 얻을 수 있다. 따라서, UIAutomator를 이용해 얻은 데이터를 바탕으로 대상 기기의 실행 화면을 구성하는 위젯과 각 위젯의 이벤트 속성 정보를 추출해 낼 수 있다.

3.3 화면 비교 기법

3.1에서 제안한 알고리즘은 S'의 화면 비교 결과에 따라 그래프 업데이트를 결정하기 때문에, 화면 비교 기법의 정확도에 따라 다른 형태의 그래프를 생성하게 된다. 따라서, 화면(GUI 상태)을 명확하게 구별할 수 있는 비교 기법이 중요한데, 이를 위해 본 연구에서 제안하는 기법은 그림 2와 같다. 본 기법은 사용자 이벤트 실행 후의 화면(S')을 GUI 그래프 상의 탐색했던 화면과 다섯 단계의 계층적인 비교를 통해 화면을 판단하고 결과를 반환한다.

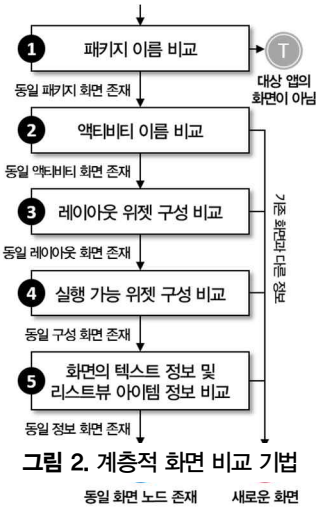
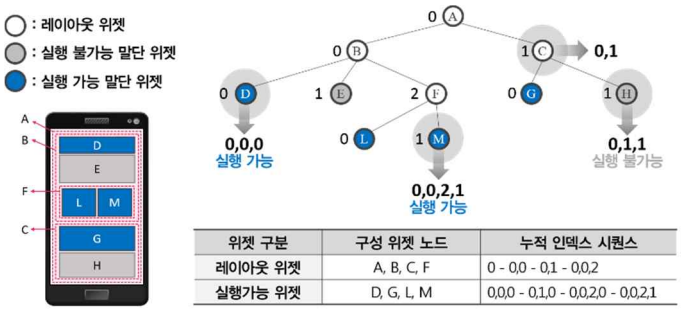


그림 2. 계층적 화면 비교 기법

(1) 패키지, 액티비티 이름의 비교

1, 2단계는 사용자 이벤트 실행 후 화면(S')으로부터 UIAutomator를 통해 추출한 패키지와 액티비티 이름을 이용해 비교한다. 1단계에서는 S'와 초기 화면의 패키지 이름 비교를 통해 대상 앱이 기기의 최상위에서 실행되고 있는지 확인한다. 만약 일치하지 않은 경우, 앱이 실행 중 사용자 이벤트에 의해 종료되었거나 제 3의 앱(3rd-party app)으로 전환되었다고 판단한다. 2단계에서는 GUI 그래프의 모든 노드들 중 S'과 같은 액티비티 이름을 갖는 것이 있는지 비교하여, 모든 노드에서 S'의 액티비티 이름이 존재하지 않을 경우 새로운 화면으로 판단한다.

(2) 화면 구성 위젯의 비교



위젯 구분	구성 위젯 노드	누적 인덱스 시퀀스
레이아웃 위젯	A, B, C, F	0 - 0,0 - 0,1 - 0,0,2
실행가능 위젯	D, G, L, M	0,0,0 - 0,1,0 - 0,0,2,0 - 0,0,2,1

그림 3. UIAutomator를 통해 추출한 위젯 트리(Tree)와 위젯 구성의 예

3단계와 4단계의 비교는 레이아웃 구조와 실행 가능한 위젯 구성이 같은 화면 노드가 있는지 확인한다. 이러한 특정 속성을 갖고 있는 위젯 구성 간 비교를 위해서 UIAutomator가 제공하는 위젯의 계층 구조를 이용한다. 3.2에서 설명한 화면으로부터 추출한 구조 정보는 그림 3과 같이 위젯 노드 간 부모-자식 혹은 형제 관계를 계층적으로 나타낸 트리(Tree) 형태로 표현된다. 계층 구조는 위젯의 속성값 중 부모 위젯 노드의 몇 번째 자식인지를 나타내는 정수 값인 index를 통해 표현되며, 각 위젯은 루트 노드로부터 내려오면서 인덱스를 누적한 고유의 인덱스 시퀀스로 나타낼 수 있다(예. 그림 3의 위젯 M의 인덱스 시퀀스 = 0,0,2,1). 이 때, 말단(Leaf) 노드를 제외한 부모 위젯들의 차례로 누적된 인덱스 시퀀스를 레이아웃 구성을 나타내는 값으로 이용하고, 이벤트 관련 속성이 true인 말단 위젯 노드의 누적된 인덱스 시퀀스를 실행 가능 위젯 구성을 나타내는 값으로 이용한다. 이를 이용해 3, 4단계의 위젯 구성 정보를 비교함으로써 액티비티 수준보다 구체적인 GUI 상태 구분이 가능하다.

(3) 텍스트 정보와 ListView 아이템 비교

마지막 5단계 비교는 위젯 구성까지 동일한 화면을 대상으로, 텍스트 정보만을 통해 구별할 수 있는 화면(예. Gmail 앱의 '받은 편지함'과

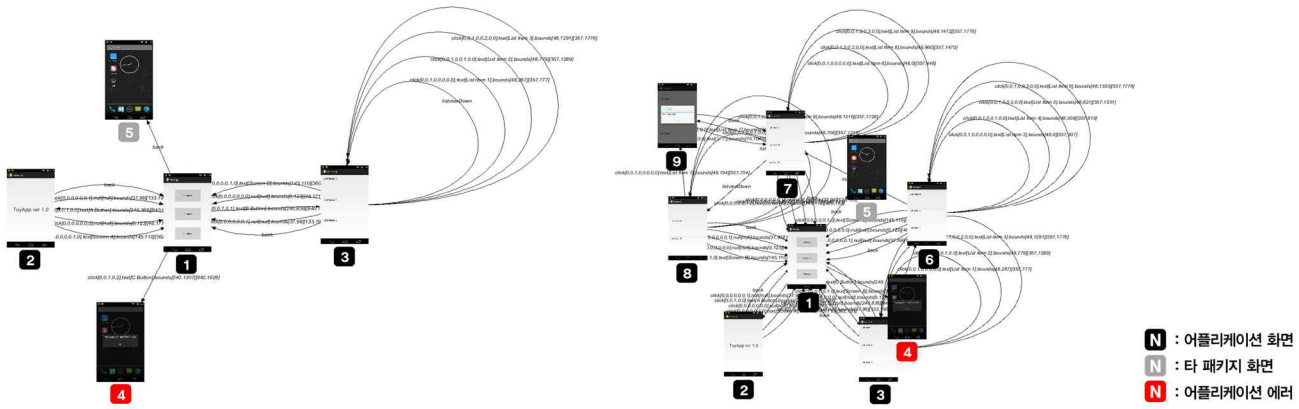


그림 4-(a). 액티비티 기반으로 생성된 ToyApp의 GUI 그래프

그림 4-(b). 제안 기법을 이용해 생성된 ToyApp의 GUI 그래프

그림 4. UIAutomator를 통해 추출한 위젯 트리(Tree)와 위젯 구성

'보낸 편지함')이나 ListView의 스크롤 후 화면을 구별하기 위해 수행한다. 해당 정보를 비교하기 위해 위젯의 text와 content-desc 속성을 이용해 텍스트 정보로 이용하며, ListView의 첫 번째 아이템의 text와 content-desc값을 비교하여 스크롤 여부를 판단한다. 이는 GUI 구조뿐만 아니라 내용적인 측면까지 고려한 것으로써, 독립적인 기능을 수행할 수 있는 화면을 최대한 구분 지을 수 있다. 최종적으로 본 기법은 5단계까지의 비교에서 모든 정보가 동일한 화면을 발견할 경우, 이를 기존에 탐색했던 화면으로 취급하고 해당 화면의 번호(id)를 반환한다.

4. 실험 결과

3장에서 제안한 화면 비교 기법을 적용한 모델 생성을 기존 기법과 비교 실험해 보기 위해, 본 연구에서는 실험 대상 안드로이드 앱으로 이용할 ToyApp을 개발했다. 이를 대상으로, 기존 액티비티 기반의 모델 생성 결과와 본 연구의 계층적 비교 기법을 사용해 생성된 GUI 그래프를 비교했는데, 해당 비교 결과는 그림 4에서 확인할 수 있다†.

비교 결과로부터 크게 두 가지 분석을 얻을 수 있다. 첫째, 본 기법을 통해 대상 앱의 모든 행위를 반영할 수 있는 정확한 GUI 그래프 생성이 가능했다. 본 기법은 향상된 비교 기법을 통해 기존 액티비티 기반 기법에서는 수집하지 못했던 화면과 사용자 이벤트 정보를 구별하여 GUI 그래프에 표현할 수 있었다. 즉, 본 기법의 모델을 이용해 생성한 테스트 케이스는 기존 기법보다 기능 검사 측면에서 보다 많은 앱의 행위를 커버(Cover)할 수 있어 유효성 향상의 가능성을 가진다는 것을 의미한다.

둘째, 생성된 GUI 그래프의 각 화면 노드는 중복 없이 구별되는 기능을 갖는 최소한의 노드로 대상 앱의 행위를 표현할 수 있었다. 즉, 각 화면 노드가 중복 없이 사용자 이벤트의 집합을 가짐으로써, 모든 이벤트가 유의미한 테스트 케이스 생성에 이용될 수 있다는 것을 의미한다. 따라서 제안한 기법은 구체적이면서도 합리적인 화면 비교 기준을 통해 모델의 상태 공간(State space)의 낭비 없이 앱의 행위를 잘 반영할 수 있는 기법임을 보였다.

5. 결론 및 향후 연구

모델 기반 GUI 테스팅의 유효성은 생성된 모델에 크게 의존적이다. 본 논문에서는 모델 기반 안드로이드 GUI 테스팅의 유효성을 높이기 위한 방안으로 계층적 화면 비교 기법을 통한 모델 생성을 제안하였다. 또한 실험을 통해 제안된 기법이 기존의 기법보다 정확한 모델을 생성함을 보임으로써 모델 기반 테스팅 기법의 성능 향상 가능성을 보여주었다.

향후 연구로는, 본 논문에서 제안한 기법에 의해 생성된 모델을 기반으로 테스트 케이스를 자동 생성하고, 테스팅 수행까지 가능한 통합된 자동화 안드로이드 GUI 테스팅 엔진 개발을 계획하고 있다. 또한, 다양한 상용 안드로이드 앱을 대상으로 본 기법의 실용성에 대해서도 평가하고자 한다.

† 자세한 실험 결과(그래프)는 http://se.kaist.ac.kr/ymbaek/kcc-experiment/ 에서 확인 가능

참 고 문 헌

[1] G. Bae, G. Rothermel, and D.H. Bae, "Comparing Model-based and Dynamic Event-Extraction Based GUI Testing Techniques: An Empirical Study", The Journal of Systems and Software, 2014.

[2] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. D. Carmine, A. M. Memon, "Using GUI ripping for automated testing of Android Applications," In Proceedings of 27th IEE/ACM International Conference on Automated Software Engineering, ASE 2012, pp. 258-261, 2012.

[3] W. Yang, M. R. Prasad, T. Xie, "A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications," FASE 2013, LNCS 7793, pp. 250-265, 2013.

[4] A. Machiry, R. Tahiliani, M. Naik, "Dynodroid: an input generation system for Android apps," In proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pp. 224-234, 2013.

[5] S. Methong, "Model-based Automated GUI Testing For Android Web Application Frameworks," ICBEM 2012, vol. 42, pp. 106-110, 2012.

[6] D. Amalfitano, A. R. Fasolino, P. Tramontana, "A GUI Crawling-based technique for Android Mobile Application Testing," Software Testing, Verification and Validation Workshops (ICSTW) 2011, pp. 252-261, 2011.

[7] SuperMonkey(https://github.com/testobject/supermonkey), 2014.

[8] 신원, 박두호, 장천현, "효율적인 안드로이드 애플리케이션 테스트를 위한 테스트 케이스 설계 방안," 정보과학회논문지: 소프트웨어 및 응용 제 40권 제 10호, pp. 575-581, 2013.