논문 2013-08-39

안드로이드 어플리케이션의 액티비티 라이프사이클 기반 GUI 테스팅 기법

(A GUI Testing Technique Based on Activity Lifecycle for Android Applications)

백 태 산, 이 우 진*
(Tae-San Baek, Woo Jin Lee)

Abstract : Most Android applications are being developed by one or a few person without profes sional testing team. In such a poor development environment, applications may contain severe err ors which may also affect the Android platform. In order to detect these errors, the behaviors of Android activities should be identified by considering user-defined lifecycle as well as the syste m-defined one. This paper proposes a method to generate test scenarios based on the both of u ser-defined and system-defined activity lifecycle of Android GUI applications. The test scenarios are generated by a state diagram extracted from the source code and are transformed into Jytho n scribes by using GUI layout information for automatically performing testing.

Keywords : Android, Android Activity lifecycle, GUI testing, Reverse engineering

Ⅰ. 서 론

안드로이드 플랫폼 기반의 스마트폰 보급률이 증가하고 안드로이드 OS활용범위가 사회 전반으로 확대되어 다양한 기기에 적용이 되고 있다. 안드로이드 어플리케이션은 비교적 간단한 인증절차로 등록 및 배포가 가능하여 다양한 기능의 안드로이드 어플리케이션이 하루가 다르게 출시되고 있다. 이러한 대부분의 어플리케이션들은 전문적인 검증팀 없이 소수의 인원으로 개발이 되고 있다. 제대로 검증되지 않은 어플리케이션은 개발과정에서 발견하지 못한 다양한 문제점들을 포함하고 있어 안정성, 신뢰성을 보장하여야 한다.

안드로이드 어플리케이션은 액티비티라는 GUI 단위로 동작하기 때문에 GUI 테스팅을 통해 안드로이드 환경에서만 발생할 수 있는 액티비티의 잘못된 생성과 소멸로 어플리케이션의 오동작 등의 오류를 검출할 수 있다[1]. 각 액티비티는 각각의 라이프사이클을 갖고 있고 액티비티의 생성, 대기, 소멸 상태를 개발자가 정의할 수 있다. 만약 개발자가 액티비티 라이프사이클 상태를 정의하지 않으면, 안드로이드 플랫폼에 정의된 상태로 관리가 된다. 그러므로 많은 개발자가 자신이 정의하지 않는 상태에 대해서는 제대로 신경을 쓰지 않는 경향이 있다. 이와 같은 액티비티 라이프사이클로 인해 안드로이드 어플리케이션의 액티비티 전환의 순서가 결정이 된다. 그리고 이를 바탕으로 한 시스템 행위 검증은 부분적으로 행해질 수밖에 없다. 제대로 된 시스템 행위 테스트를 위해서는 사용자 정의 상태뿐만 아니라 시스템 제공 상태들을 모두 포함한 모든 라이프사이클을 고려하여야 한다.

이 연구에서는 어플리케이션내의 모든 액티비티의 라이프사이클 정보를 고려하기 위해, 역공학을 통해 소스 코드로부터 액티비티별 사용자 정의 라이프사이클 정보를 상태도 형태로 먼저 추출하고 추가로 시스템 정의 부분을 보완한다. 보완된 상태도를 바탕으로 테스트 시나리오를 생성한다. 또한 테스트 시나리오 자동 실행을 위해 GUI 레이아웃

*Corresponding Author(woojin@knu.ac.kr)
Received: 21 June 2013, Revised: 25 July 2013, 29 Aug. 2013, Accepted: 4 Sep. 2013.
T.S. Baek, W.J. Lee: Kyungpook National University.
※ 본 논문은 2012학년도 경북대학교 전임교원 연구년 교수 연구비에 의하여 연구되었음.

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2013.8.6.319

구성요소 정보를 분석하여 Jython 기반의 테스트 스크립트 생성기법을 다룬다. 그리고 적용사례를 통해 제시한 방법의 적용가능성을 분석한다.

Ⅱ. 관련연구

일반적인 C 언어 또는 Java 언어 기반의 데스크탑 또는 모바일 소프트웨어는 main()이라는 하나의 진입점을 가지고 있지만, 안드로이드 어플리케이션의 진입점은 액티비티, 서비스, Broadcast Receiver, Content Provider의 4개의 컴포넌트 모두가 가능하다. 즉, 안드로이드는 메인함수와 같은 유일한 진입점이 따로 있는 것이 아니며 처음으로 생성되는 인스턴스 생성자가 실질적인 진입점 역할을 한다. 이와 함께 안드로이드는 키보드 또는 마우스 등과 같은 이벤트를 통해 동작하지 않고 터치 이벤트 기반으로 동작이 된다. 컴포넌트 중 액티비티는 안드로이드 어플리케이션 GUI를 구성하는 가장 기본적인 빌딩블록으로 한 화면을 차지하면서 뷰로 구성된 유저 인터페이스를 화면에 표시한다. 또한 사용자의 입력을 처리하는 역할을 수행하고 하나의 어플리케이션은 여러 개의 다른 화면을 구성하는 서로 다른 라이프사이클을 갖고 있는 액티비티들로 구성이 되어 있다. 이러한 안드로이드 어플리케이션의 여러 진입점, 액티비티 라이프사이클 특성으로 인해 기존의 C 언어 등의 테스팅 기법과 다른 관점에서 접근하여야 한다.

안드로이드 플랫폼은 어플리케이션 테스팅을 위해 JUnit[2], Monkey[3], MonkeyRunner[4]의 세 가지를 기본적으로 제공하고 있다. 안드로이드의 단위 테스트를 위해서는 JUnit을 지원한다. 다만 기존 JVM에서 수행하던 JUnit을 수행하는 것은 타겟에서 수행하는 것에 대한 보장을 하지 못하므로, Android Test Case Class를 별도로 제공한다. 또한, 단순한 API를 테스트하는 것 외에 Instrumentation 기반 테스팅 기법을 제공한다. 사용자의 행위에 대한 내용을 코드로 표현하여 기능 테스트를 한다. Monkey는 안드로이드의 무작위 이벤트를 타겟에 보내 에러가 발생하는지, 또는 어느 정도의 시간을 버티는지를 테스트하는 스트레스 테스트를 할 수 있다. MonkeyRunner는 액션 좌표값과 액션 이벤트를 Jython 기반 스크립트 형태의 코드를 제작하여 기능 테스트를 수행하는 도구이다.

JUnit을 이용한 단위 테스트에 사용되는 테스트 시나리오 생성에 대한 연구에는 안드로이드 어플리케이션의 GUI의 기본 단위인 하나의 액티비티에 대한 단위 테스트 방법을 제안하고 있다[5]. 이 방법은 하나의 액티비티에 대한 테스팅 방법으로 액티비티를 구성하는 xml 파일과 소스코드에서 추출한 정보를 기반으로 테스트 케이스를 생성하고 JUnit 테스트에 사용할 수 있는 테스트 코드를 생성한다. 이는 하나의 어플리케이션은 하나 이상의 액티비티로 구성되어 있어 모든 액티비티와 액티비티간의 이동에 대한 GUI 테스팅에 사용할 수 없다.

기존에 연구된 모바일 소프트웨어 GUI 테스팅 방법에는 명세 기반 테스팅, 베타 테스팅 그리고 RPB(Record Play Back) 기반 테스팅 등을 사용하고 있다[5]. GUI 명세를 활용하는 명세기반 테스팅은 간결한 방식으로 테스트 명세를 할 수 있지만 정확한 GUI 명세를 필요로 하는 단점을 갖고 있다. 베타 테스팅은 많은 수의 사용자를 대상으로 테스트 수행하는 방법으로 다양한 테스트 시나리오로 많은 오류를 발견 할 수 있으나 대상 선정에 어려움이 있다. 안드로이드 GUI 테스팅 연구에 가장 많이 사용이 되고 있는 RPB 기반 테스팅은 개발자가 직접 어플리케이션을 실행하여 발생되는 이벤트를 기록한 후 이를 테스트 시나리오로 활용하는 방식으로 개발자에 따라 주관적이고 어플리케이션에서 발생할 수 있는 모든 경로를 찾기 어렵고 위에서 언급한 안드로이드의 특성인 액티비티 라이프사이클[6]을 고려한 테스트 시나리오를 생성 할 수 없다. RPB 기반 테스팅을 이용한 기존 방법에는 Robotium[7], Sikuli[8] 등이 있다. 명세 기반 테스팅, 베타 테스팅 그리고 RPB(Record Play Back) 기반 테스팅 기법들은 테스트 스크립트의 수정이 불편하여 지속적으로 GUI가 변경이 되는 개발과정에 사용하기에는 불편하다는 문제점이 수반된다.

Ⅲ. 안드로이드 액티비티 기반의 테스트 시나리오 생성

본 논문에서는 안드로이드 어플리케이션 GUI 테스팅을 위한 테스트 시나리오 자동 생성을 위해, 소스코드에서 역공학 기법으로 단위 프로그램(함수, 액티비티 등)들 사이의 호출관계 정보로 안드로이드 액티비티 기반 상태도를 생성한다. 생성된 액티비티 기반의 상태도와 어플리케이션의 레이아웃 정보에서 GUI 테스팅에 사용할 수 있는 안드로이드 액티비티 기반의 테스트 시나리오를 생성한다(그림 1).

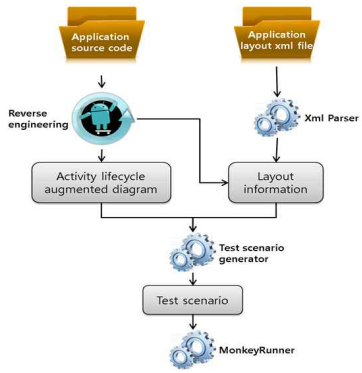


그림 1. 안드로이드 테스트 시나리오 생성 절차
Fig. 1 Generation process of Android test scenario

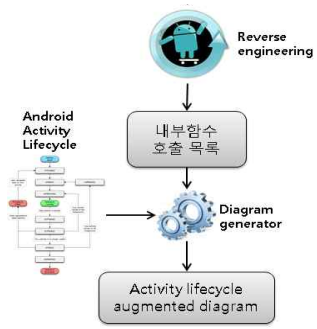


그림 2. 안드로이드 액티비티 기반 상태도 생성 절차
Fig. 2 Generation process of Android activity based state diagram

본 논문에서는 안드로이드 SDK에 포함되어 있는 샘플코드 중 메모장 어플리케이션에 대한 GUI 테스팅을 위한 테스트 시나리오를 생성하는 방법에 대해서 다룬다.

3.1 안드로이드 액티비티 기반 상태도 생성

안드로이드 GUI 테스팅에 사용할 테스트 시나리오 생성을 위해 안드로이드 액티비티의 특성인 라이프사이클을 고려한 액티비티 기반 상태도를 생성한다. 소스코드에서 역공학을 통해 생성된 액티비티 기반 상태도는 소스코드에 구현이 되어 있는 사용자 정의 상태만 고려되어 액티비티 완벽한 라이프사이클 정보를 포함하고 있지 않다.

안드로이드 액티비티 라이프사이클 정보는 하나의 액티비티가 전환 또는 시작할 때 액티비티의 생성, 소멸, 대기 상태를 결정 한다. 하지만 개발자는

표 1. 내부함수 호출 목록
Table 1. List of internal function calling

Class_Name	Function_name	Called Destination	Terminate Option
NoteList	OnCreate()	-	
NoteList	OnCreateContextMenu()	-	
NoteList	OnCreateOptionMenu()	-	
NoteList	OnListItemClick()	NoteEditor	
NoteList	OnContextItemselected()	NoteEditor	
NoteList	OnOptionItemselected()	NoteEditor	
NoteList	OnOptionItemselected()	TitleEditor	
TitleEditor	OnCreate()	-	
TitleEditor	OnResume()	-	
TitleEditor	OnPause()	-	
TitleEditor	OnClickListener().OnClick()	-	finish()
NoteEditor	OnCreate()	-	
NoteEditor	OnResume()	-	
NoteEditor	OnPause()	-	
NoteEditor	OnDraw()	-	
NoteEditor	OnPrepareoptionOnMenu()	-	
NoteEditor	OnOptiononItemselected().cancelNote()	NoteList	
NoteEditor	OnOptiononItemselected().SaveNote()	NoteList	finish()
NoteEditor	OnOptiononItemselected().deleteNote()	NoteList	finish()

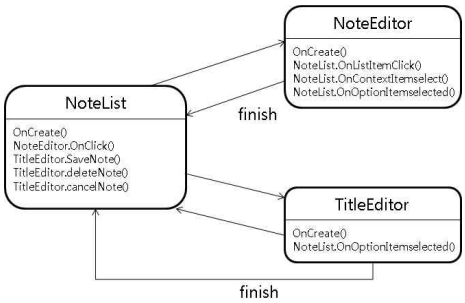


그림 3. 소스코드 기반 액티비티 상태도
Fig. 3 Source code based Activity state diagram

모든 액티비티의 상태를 결정하지 않고 관심이 있는 부분을 중심으로 구현한다. 이와 같이 소스코드에 구현되어 있지 않은 시스템 정의도 액티비티 전환의 순서에 영향을 주기 때문에 개발자 정의 부분과 함께 시스템 정의 부분을 함께 고려하여 상태도를 그림 2와 같이 생성한다.

3.1.1 소스코드 기반 액티비티 상태도 생성

기존에 존재하는 역공학 기술을 이용하여 안드로이드 메모장 샘플 소스코드에서 액티비티와 호출함수 정보로 구성된 내부함수의 호출 목록을 표 1과 같이 생성할 수 있다.

생성된 내부함수 호출 목록에서 다른 액티비티를 호출하는 함수와 자신의 라이프사이클과 관련이 있는 함수 정보를 하나의 상태로 정의하고 다른 액티비티 전이시에 추가적인 사항은 간선 라벨에 표시하

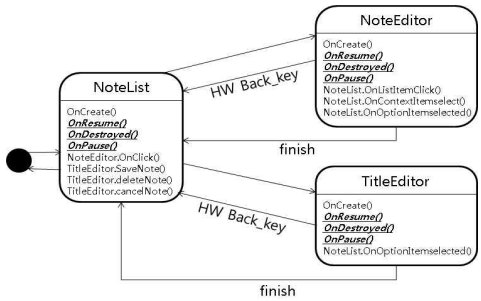


그림 4. 안드로이드 정의 행위를 고려한 상태도
Fig. 4 State diagram considering Android defined behaviors

여 안드로이드 소스코드 기반 액티비티 상태도를 그림 3과 같이 생성한다.

3.1.2 액티비티 특성을 고려한 상태도 생성

생성된 안드로이드 소스코드 기반의 액티비티 상태도는 액티비티의 특성인 라이프사이클 정보 중 소스코드에 구현이 되어 있는 일부분만 포함되어 있다. 각각의 액티비티들은 서로 다른 라이프사이클을 가지고 있고 소스코드에 구현이 되어 있지 않는 라이프사이클 정보에 따라 이전 액티비티로 이동하는 순서와 이동후의 액티비티의 상태가 달라진다. 위와 같은 이유로 액티비티의 라이프사이클의 소스코드에 구현이 되어 있지 않는 정보도 상태도 생성에 추가하여야 한다.

이를 위해 소스코드 기반 액티비티 상태도에 소스코드에서 구현되어 있지 않은 하드웨어 키 이벤트인 Back_key 이벤트에 따른 이전 액티비티로의 이동 간선과 라벨을 추가한다. 어플리케이션의 첫 번째 액티비티 또는 다른 액티비티에서 이전상태로 이동시 어플리케이션 실행 이전 상태로 돌아가기 위한 임의의 초기상태를 추가하여 안드로이드의 특성을 고려한 액티비티 상태도를 생성할 수 있다. 액티비티 라이프사이클은 소스코드에 개발자가 직접 구현할 수도 있고 구현을 하지 않아도 안드로이드 플랫폼에서 자체적으로 관리한다. 본 논문에서 이 두 부분을 개발자 정의 행위부분과 직접 구현하지 않은 부분인 안드로이드 정의 행위로 정의하여 사용한다. 테스트 시나리오를 생성하기 위해서 소스코드에 포함되어 있는 개발자 정의 행위뿐 아니라 구현되어 있지 않은 안드로이드 정의 행위의 라이프사이클도 고려하여 상태도를 그림 4와 같이 완성한다.

표 2. 사용자 정의 행위기반의 테스트 시나리오
Table 2. Test scenario for customer defined behaviors

ID	Test Scenario
TS1	init → NoteList.OnCreate() → NoteList.OnListItemClick() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnCreate()
TS2	init → NoteList.OnCreate() → NoteList.OnContextItemselect() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnCreate()
TS3	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnCreate()
TS4	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TetleEditor.save() → TitleEditor.finish() → NoteList.OnCreate()
TS5	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TitleEditor.deleteNote() → TitleEditor.finish() → NoteList.OnCreate()
TS6	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TitleEditor.cancelNote() → NoteList.OnCreate()

표 3. 시스템 정의 행위를 포함한 테스트 시나리오
Table 3. Test scenario considering system defined behaviors

ID	Test Scenario
TS1	init → NoteList.OnCreate() → NoteList.OnListItemClick() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnResume()
TS2	init → NoteList.OnCreate() → NoteList.OnContextItemselect() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnResume()
TS3	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → NoteEditor.OnCreate() → NoteEditor.OnClick() → NoteEditor.finish() → NoteList.OnResume()
TS4	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → NoteEditor.OnCreate() → HW Back_key → NoteList.OnResume()
TS5	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TitleEditor.save → TitleEditor.finish() → NoteList.OnResume()
TS6	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TitleEditor.deleteNote() → TitleEditor.finish() → NoteList.OnResume()
TS7	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → TitleEditor.cancelNote() → NoteList.OnResume()
TS8	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → HW Back_key → NoteList.OnResume()
TS9	NoteList.OnResume() → HW Back_key → NoteList.OnDestroyed() → init

3.2 액티비티 기반 테스트 시나리오 생성

생성된 안드로이드 액티비티 기반의 상태도에서 테스트 케이스를 생성하는 방법에는 DFS 알고리즘을 사용하는 Wang의 알고리즘[9], 인공지능의 적응형 에이전트를 이용한 Li의 알고리즘[10]과 메타데이타에 관한 정보를 교환하기 위한 표준방식인 XMI(extensible Metadata Interchage)를 이용하여 테스트 시나리오를 생성하는 방법인 Chandler 알고리즘[11] 등이 있다. 본 논문에서는 UML 다이어그램에서 DFS을 방법을 이용하여 비교적 빠른 속

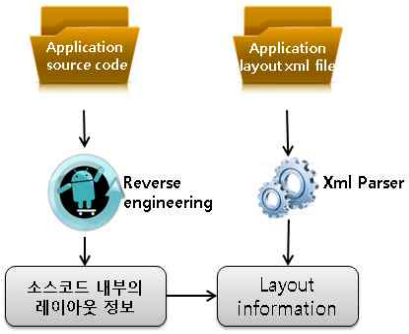


그림 5. 레이아웃 정보 생성 절차
Fig. 5 Generation process for layout information

도로 테스트 케이스를 도출하는 Wang의 알고리즘을 사용한다. 테스트 시나리오 생성 시에 GUI 테스팅에 사용하기 위하여 안드로이드 액티비티와 다른 액티비티를 호출하는 함수 정보를 중심으로 표 2와 같이 생성한다. 각 테스트 시나리오에서 이벤트는 “액티비티_이름.함수_이름” 형식에 맞춰 액티비티와 액티비티내의 함수 정보로 생성한다. 예를 들어, 표 2의 첫 번째 테스트 시나리오의 NoteList.OnCreate()와 같다.

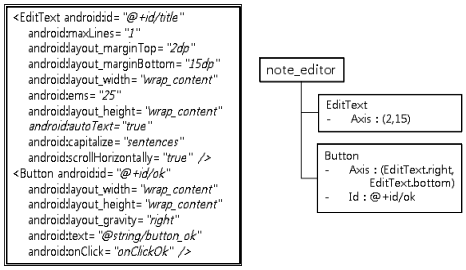
개발자 정의 부분만 고려한 테스트 시나리오는 소스코드에 포함되어 있지 않은 시스템 정의 부분을 고려하지 않아 모든 테스트 시나리오에서 NoteList 액티비티로 되돌아갈 때 NoteList.Oncreate() 상태로 돌아가게 된다. 하지만, NoteList 액티비티는 없어진 OnDestroyed() 상태가 아니라 OnPause() 상태였기 때문에 NoteList.OnResume() 상태로 돌아가야 한다. 이런 문제를 해결하기 위해 소스코드에 포함되지 않은 액티비티 라이프 사이클 정보와 함께 고려하여 테스트 시나리오를 표 3과 같이 생성한다.

3.3 GUI 레이아웃 기반 테스트 시나리오 변경

소스코드내의 액티비티 정보로 생성된 테스트 시나리오는 액티비티 이름과 호출함수로 표시되어 있어 레이아웃과 매칭하기 어렵기 때문에 테스트 시나리오가 정확한지를 확인하기 위해서는 실행을 해야만 가능하다. 테스트 시나리오 설계 시에 확인하기 위해서 어플리케이션 레이아웃 정보에서 액티비티 호출 이벤트 함수를 대체할 정보를 얻을 수 있다.

표 4. 레이아웃 정보를 포함한 테스트 시나리오
Table 4. Test scenario including layout information.

ID	Test Scenario
TS1	init → NoteList.OnCreate() → ListView.ListItem.TextView → NoteEditor.OnCreate() → note_editor.Button_ok → NoteEditor.finish() → NoteList.OnCreate()
TS2	init → NoteList.OnCreate() → NoteList.OnContextItemselect() → NoteEditor.OnCreate() → note_editor.Button_ok → NoteEditor.finish() → NoteList.OnCreate()
TS3	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → NoteEditor.OnCreate() → note_editor.Button_ok → NoteEditor.finish() → NoteList.OnCreate()
TS4	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → title_editor.Button_save → TitleEditor.finish() → NoteList.OnCreate()
TS5	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → title_editor.Button_delete → TitleEditor.finish() → NoteList.OnCreate()
TS6	init → NoteList.OnCreate() → NoteList.OnOptionItemselected() → TitleEditor.OnCreate() → title_editor.Button_cancel → NoteList.OnCreate()



(a) XML 파일 (b) 추출정보
(a) XML file (b) Layout

그림 6. 안드로이드 액티비티 레이아웃 구성 요소 정보
Fig. 6 Layout component for Android Activity

이미 기존에 xml 파일을 분석하여 레이아웃 정보를 추출하여 테스트 시나리오를 생성하는 방법들은 xml 파일 내에 모든 레이아웃 구성요소 정보가 존재한다는 전제조건이 있다. 안드로이드의 레이아웃 정보는 xml 파일 외부의 소스코드에도 존재하기 때문에 소스코드 내부의 구성요소 정보도 고려되어야 한다. 위와 같은 문제를 해결하기 위해 상태도 생성 시에 역공학을 통해 획득한 소스코드 내부에 존재하는 레이아웃 정보와 xml 파일을 파싱하여 획득한 레이아웃 구성 정보를 취합하여 사용하여 자동으로 레이아웃 정보를 포함한 테스트 시나리오를 그림 5와 같이 생성한다. 액티비티간의 이동과 관련되는 이벤트에는 버튼과 같이 레이아웃을 구성하는 xml 파일에 존재하는 부분과 하드웨어 메뉴키 또는

```
1. device = connectWithDevice()
2. # Installs the Android package. Notice that this method returns a boolean,
   so you can test
   # to see if the installation worked.
   #device.installPackage('C:\Users\Administrator\Desktop\Notepad-source-
   1.3\Notepad-source-1.3\notepad\NotePad\apk\NotePad.apk')
3. # sets a variable with the package's internal name
   package = 'org.openintents.notepad'

   # sets a variable with the name of an Activity in the package
   activity = 'org.openintents.notepad.NotePad'

   # sets the name of the component to start
   runComponent = package + '/' + activity

   # Runs the component
   device.startActivity(component=runComponent)
   takeSnapshot()

   device.touch(387,476, MonkeyDevice.DOWN_AND_UP)
4. #MonkeyRunner.sleep(0.5)
   takeSnapshot()

   device.touch(430,760, MonkeyDevice.DOWN_AND_UP)
   #MonkeyRunner.sleep(0.5)
   takeSnapshot()

   device.touch(430,760, MonkeyDevice.DOWN_AND_UP)
   #MonkeyRunner.sleep(0.5)
   takeSnapshot()

   device.touch(430,760, MonkeyDevice.DOWN_AND_UP)
   #MonkeyRunner.sleep(0.5)
   takeSnapshot()

   device.press('KEYCODE_BACK', MonkeyDevice.DOWN_AND_UP)

   MonkeyRunner.alert("Auto test is completed.", "END", "Exit")
```

그림 7.Monkeyrunner를 위한 생성된 스크립트
Fig. 7 Generation scripts for Monkeyrunner

long-press를 통해 표시되는 옵션메뉴와 같이 소스코드에 존재하는 부분으로 구분하여 표 4와 같이 변경한다.

각 액티비티는 그림 6(a)와 같이 레이아웃을 구성하는 xml 파일이 존재한다. xml 파일내에는 UI 컴포넌트와 ID, 위치 등과 같은 속성정보를 포함하고 있다. 이중 테스트 시나리오를 생성하기 위하여 이벤트를 발생시키는 UI 컴포넌트 정보과 각 컴포넌트의 ID와 위치정보를 추출하고 소스코드에서 역공학을 통해 동적으로 UI 컴포넌트를 구성하는 부분을 함께 고려한다.

DOM 파서를 이용하여 note_editor xml 파일을 Button, EditBox 등과 같은 UI 컴포넌트 정보와 각 속성정보를 그림 6(b)와 같이 추출하고 소스코드 내에서 구성 요소를 호출하는 함수 findViewById(R.id.ok)와 매칭하여 레이아웃 정보를 적용하여 테스트 시나리오 이벤트를 NoteEditor.OnClick()에서 레이아웃 기반의 note_editor.Button으로 변경한다.

IV. Monkeyrunner를 이용한 테스트 시나리오 실행

안드로이드 기반의 에뮬레이터 또는 타겟 디바이스에서 생성된 테스트 시나리오를 실행시키기 위

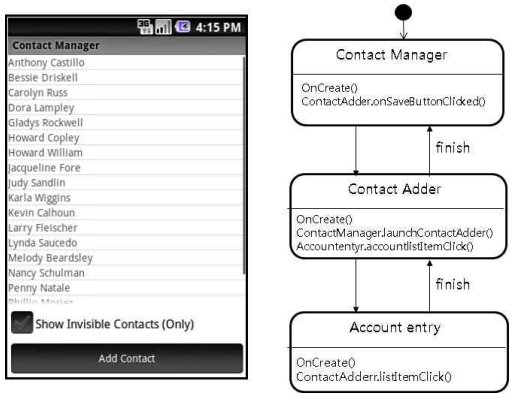


그림 8. Contact Manager 수행 화면 및 상태도
Fig. 8 Contact Manager snapshot and state diagram

한 도구가 필요하다. 본 논문에서는 테스트 시나리오를 실행시키기 위해 안드로이드 SDK에 포함되어 있는 스크립트 기반의 Monkeyrunner를 사용한다. Monkeyrunner를 사용하기 위해서는 생성된 테스트 시나리오를 실행 가능한 Jython 기반의 스크립트로 변환하여야 한다.

4.1 실행 가능한 Jython기반의 스크립트 생성

Monkeyrunner에서 테스트 시나리오를 사용하기 위해서는 Jython기반의 스크립트 파일로 변환 생성해야 한다. 본 논문에서는 스크립트 파일을 자동 생성하기 위하여 3절에서 테스트 시나리오를 레이아웃 정보를 고려하여 생성하였다. 스크립트 파일은 레이아웃의 좌표 값과 다음과 같은 4가지 제한사항을 고려하여 그림 7과 같이 생성한다.

1. 디바이스와 연결
2. 디바이스에 어플리케이션 패키지 설치
3. 디바이스에 패키지와 시작 액티비티 설정 및 시작
4. 좌표정보를 이용한 터치 이벤트 또는 디바이스에 전송 전 항상 2초간의 딜레이 타임을 삽입하여야 한다.

위의 4가지 제한사항을 고려하고 어플리케이션의 이벤트를 발생시키는 부분은 device.touch (x,y,MonkeyDevice.DOWN_AND_UP)와 같이 좌표 값을 이용하는 방법과 하드웨어 키 이벤트를 실행시키는 device.press ("KEYCODE_BACK", MonkeyDevice.DOWN_AND_UP)로 작성한다. 생성된

표 5. Contact Manager의 테스트 시나리오
Table 5. Test scenario for Contact Manager

ID	Test Scenario
TS1	init → ContactManager.OnCreate() → contact_manager.listitem → ContactAdder.OnCreate() → contact_adder.Button_save → OntactAdder.finish() → ContactManager.OnResume()
TS2	init → ContactManager.OnCreate() → contact_manager.listitem → ContactAdder.OnCreate() → contact_adder.listitem → AccountEntry.OnCreate() → account_entry.listitem → AccountEntry.finish() → ContactAdder.OnCreate() → ContactAdder.finish() → ContactManager.OnResume()
TS3	ContactManager.OnResume() → init

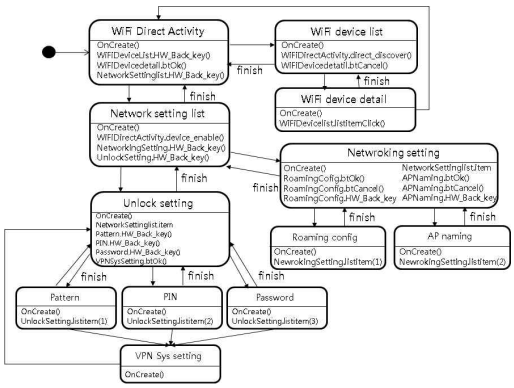


그림 9. WiFiDirectDemo 액티비티 기반 상태도
Fig. 9 Activity based state diagram for WiFiDirectDemo

스크립트 파일은 ADB(Android Debug Bridge)를 사용하여 Monkeyrunner와 연결하여 GUI 테스팅을 진행한다.

Ⅴ. 적용사례

본 논문에서 연구한 소스코드와 레이아웃 구성요소 정보 기반의 GUI 테스팅을 위한 테스트 시나리오 생성기법을 안드로이드 SDK에 포함되어 있는 샘플코드 중 주소관리 어플리케이션인 Contact Manager에도 적용하여 테스트 시나리오를 생성한다. Contact Manager 어플리케이션 소스코드에 대한 역공학을 통해 액티비티 정보를 추출하여 안드로이드 액티비티 기반 상태도를 생성한다(그림 8).

생성된 액티비티 기반 상태도에서 안드로이드 액티비티 기반의 테스트 시나리오를 생성하여 레이아웃 구성요소 정보를 취합하여 Contact Manager의 액티비티 라이프 사이클의 개발자 정의 부분과 시스템 정의 부분을 모두 고려한 상태도와 레이아웃 구성

표 6. WiFiDirectDemo의 라이프 사이클을 고려한 테스트 시나리오
Table 6. Test scenario considering lifecycle of WiFiDirectDemo

ID	Test Scenario
TS1	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_direct_discover → WiFiDevicelist.OnCreate() → WiFiDeviceListView.list → WiFiDevicedetail.OnCreate() → WiFiEdevicedetailView.btn_OK → WiFiDirectActivity.OnResume()
TS2	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_direct_discover → WiFiDevicelist.OnCreate() → WiFiDeviceListView.list → WiFiDevicedetail.OnCreate() → WiFiEdevicedetailView.btn_Cancel → WiFiDevicelist.OnResume() → WiFiDevicelist.HW_Back_key → WiFiDirectActivity.OnResume()
TS3	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → RoamingConfig.OnCreate() → RoamingConfig.btnOk → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS4	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → APnaming.OnCreate() → APnaming.btnOk → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS5	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → RoamingConfig.OnCreate() → RoamingConfig.btnCancel → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS6	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → APnaming.OnCreate() → APnaming.btnCancel → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS7	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → RoamingConfig.OnCreate() → RoamingConfig.HW_Back_key → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS8	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → NetworkingSetting.OnCreate() → NetworkingsetView.list → APnaming.OnCreate() → APnaming.HW_Back_key → NetworkingSetting.OnResume() → NetworkingSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS9	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → Pattern.OnCreate() → Pattern.unlock() → VPNSyssetting.OnCreate() → VPNSyssettingView.btnOk() → UnlockSetting.OnResume()
TS10	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → Pattern.OnCreate() → Pattern.HW_Back_key → UnlockSetting.OnResume()
TS11	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → PIN.OnCreate() → PIN.unlock() → VPNSyssetting.OnCreate() → VPNSyssettingView.btnOk() → UnlockSetting.OnResume()
TS12	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → PIN.OnCreate() → PIN.HW_Back_key → UnlockSetting.OnResume()
TS13	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → Password.OnCreate() → Password.unlock() → VPNSyssetting.OnCreate() → VPNSyssettingView.btnOk() → UnlockSetting.OnResume()
TS14	init → WiFiDirectActivity.OnCreate() → WiFiDirectView.btn_device_enable → NetworkSettinglist.OnCreate() → NetworkSettinglistView.list → UnlockSetting.OnCreate() → UnlockSetting.list → Password.OnCreate() → Password.HW_Back_key → UnlockSetting.OnResume()
TS15	UnlockSetting.OnResume() → UnlockSetting.HW_Back_key → NetworkSettinglist.OnResume()
TS16	NetworkSettinglist.OnResume() → NetworkSettinglist.HW_Back_key → WiFiDirectActivity.OnResume()
TS17	WiFiDirectActivity.OnResume() → WiFiDirectActivity.HW_Back_key → init

요소 정보를 이용하여 테스트 시나리오를 표 5와 같이 생성하였다.

추가적인 적용사례로 모바일 네트워크 및 WiFi설정을 지원하는 어플리케이션인 WiFiDirectDemo 어플리케이션에 대해 본 논문에서 제시한 테스트 시나리오 생성 방법을 적용하여 Contact Manager보다 많은 액티비티와 이동경로를 갖고 있어 큰 규모의 복잡한 어플리케이션에도 적용 가능성을 확인할 수 있다.

WiFiDirectDemo 어플리케이션 소스코드에 대한 역공학을 통해 액티비티 정보를 추출하여 그림 9와 같이 안드로이드 액티비티 기반의 상태도를 생성한다.

생성된 WiFiDirectDemo 상태도와 레이아웃 구성요소 정보를 취합하여 표 6과 같이 GUI 테스팅을 위한 테스트 시나리오를 생성하였다.

Ⅵ. 결론

본 논문에서는 가장 많이 사용되고 있는 안드로이드 플랫폼 기반의 어플리케이션의 GUI 테스팅을 위해 소스코드에서 역공학을 통해 안드로이드 액티비티 호출 정보를 추출하여 안드로이드 액티비티 특성을 고려하여 테스트 시나리오를 생성하는 방법에 대하여 연구하였다. 기존의 테스트 시나리오 생성 방법 연구는 소스코드만 고려하여 액티비티 라이프 사이클 정보 중 개발자 정의 부분만 고려하여 정확한 테스트 시나리오 생성이 되지 않는다. 하지만 본 논문에서 제안하는 방법인 개발자 정의 부분과 시스템 정의 부분을 함께 고려하고 실행 가능한 테스트 시나리오 스크립트 생성의 자동화를 위해 레이아웃 xml 파일과 소스코드에 구현이 되어 있는 레이아웃 정보를 함께 고려하여 테스트 시나리오를 생성한다. 이렇게 생성된 테스트 시나리오는 개발자가 신경을 쓰지 못하는 시스템 관리 부분까지 고려하여 커버리지가 향상된 GUI 테스팅을 수행할 수 있다.

References

[1] C. Hu, "Automating GUI Testing for Android Applications," Proceedings of International Workshop on Automation of Software Test. ACM, 2011.

[2] JUnit, http://www.junit.org/taxonomy/term/6.

[3] Monkey, http://developer.android.com/guide/developing/tools/monkey.html.

[4] Monkey UI/Application Excerceiser, http://developer.android.com/guide/developing/tools/monkey.html, May 2010.

[5] J.G. Lee, H.S. Kim, S.H. Kuk, D.W. Cho, "Record-Playback based Automatic test case generation for GUI test," Proceedings of the KIISE Korea Computer Congress, Vol. 34, No. 1(B), pp.96-100, 2007 (in Korean).

[6] Android activity lifecycle," http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle," 2010.

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

[8] Sikuli, http://sikuli.org/

[9] L. Wang, J. Yuan, X. Yu, J. Hu, X. Li, G. Zheng, "Generating Test Cases from UML Activity Diagram based on Gray-Box Method," Proceedings of Asia-Pacific Software Engineering Conference (APSEC'04), 2004.

[10] H. Li, C.P. Lam, "Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams," TestCom, LNCS 3502, pp.69-80, 2005.

[11] R. Chandler, C.P. Lam, H. Li, "An Automated Approach to Generating Usage Scenarios from UML Activity Diagrams," Proceedings of Asia-Pacific Software Engineering Conference, 2005.

저 자 소 개

백 태 산

2005년 위덕대학교 컴퓨터공학과 학사.
2008년 경북대학교 컴퓨터과학과 석사.
현재, 경북대학교 IT대학 컴퓨터학부 박사과정

관심분야: 임베디드 소프트웨어 테스팅, 소프트웨어 품질관리 등.
Email: sans79@gmail.com

이 우 진

1992년 경북대학교 전자계산학과 학사.
1994년 KAIST 전산학과 석사.
1999년 KAIST 전산학과 박사.
1999년~2002년 ETRI 선임연구원.
2002년~현재, 경북대학교 IT대학 컴퓨터학부 교수.

관심분야: 임베디드 소프트웨어 테스팅, 임베디드 소프트웨어 개발환경, 실시간 시스템 분석 등.
Email: woojin@knu.ac.kr