

# EFSM 기반의 사용성 저해 요소 실시간 검출 기법

마경욱\*, 박수진\*\*

\*서강대학교 컴퓨터공학과

\*\*서강대학교 미래기술연구원

e-mail : {shwarz, psjdream}@sogang.ac.kr

## EFSM based Real-Time Detection GUI Bad Symptom on Mobile Application

Kyeong-Wook Ma\*, Soo-Jin Park\*\*

\*Dept. of Computer Science, So-Gang University

\*\*Sogang Institute Advanced Technology, So-Gang University

### 요 약

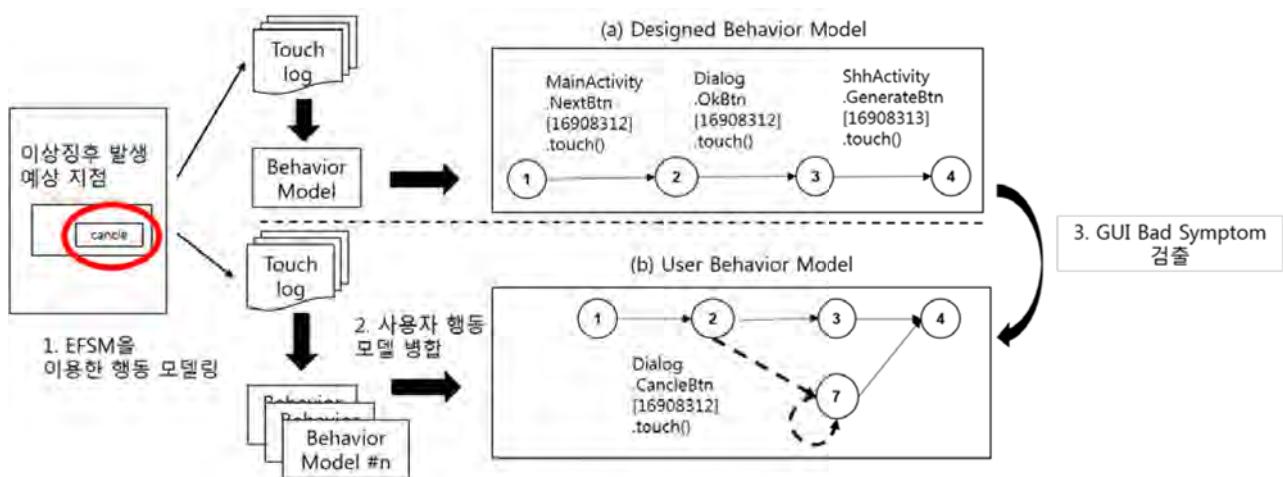
모바일 디바이스가 기존이 PC 시장규모를 압도하면서 기하급수적으로 많은 모바일 앱이 출시되고 있다. 수많은 모바일 앱 중에서 사용자들은 배우기 쉽고, 직관적으로 사용하기에 용이한 모바일 앱을 선택하려는 경향을 보인다. 그러나 모바일 앱 사용성의 중요성이 증대됨에도 불구하고 모바일 앱에서의 사용성에 대한 연구는 미미한 실정이다. 현 GUI 사용성 테스트는 많은 시간이 소요되고 주관적인 평가에 의존하는 단점이 존재한다. 본 논문은 사용자들이 앱을 조작하는 터치 입력 데이터를 기반하여 EFSM(Extended Finite State Machine)에 인자를 추가한 형태로 모델링 후 개발자의 예상 시나리오와 비교하여 GUI 사용성 저해요소를 검출하는 기법을 제안하고 있다. 이와 같은 반-자동화된 방법을 통해 모바일 앱 상에서의 사용성을 측정할 수 있다. 제안된 기법을 구현한 도구의 구조를 설명하고 사용성 저해 요소 검출하는 과정을 기술하였다.

### 1. 서론

그래픽 유저 인터페이스(Graphic User Interface: GUI)는 데스크탑이나 모바일과 같은 소프트웨어 시스템에서 중요한 역할을 수행하고 있다. GUI는 사용자들에게 보여지는 부분이기 때문에 GUI를 개발하고 테스트하는 것은 개발의 50-60%를 차지한다. 개발자들은 단위 테스트와 같은 자동화된 코드를 통해 GUI의 기능을 테스트 할 수 있다. GUI의 기능성 요구사항뿐만 아니라 사용자들에게 GUI가 사용이 용이한지, 효율적으로 기능을 수행할 수 있는지 역시 테스트의 대상이다. 이러한 비기능적 요구사항 중 사용자에게 효율성, 학습성 등을 제공하는 정도를 GUI 사용성(GUI Usability)이라 할 수 있다. GUI의 기능성 요구사항은 단위 테스트, 지속적 통합 도구 등을 통해 자동화가 가능하나 비기능 요구사항인 GUI 사용성은 사용자의 주관이 반영되고 실험 대상자에 따라 변하기도 하여 자동화가 어렵다. 그러므로 GUI 사용성을 테스트하기 위한 주된 작업은 수동으로 이뤄지고 있다. 많은 시간이 소요되는 GUI 사용성 테스트를 효율적으로 하기 위하여 사용자 터치를 기록 및 실행[1], 화면 캡쳐를 통한 테스트[2], 등의 연구가 진행되었다. 그러나 주관적 평가에 의존하거나 추가적인 카메라나 녹화된

영상을 처리하기 위한 기기가 필요하는 등 테스트에 많은 비용이 소모되는 등의 문제점이 있다.

이러한 문제점을 해결하기 위하여 이번 연구에서는 모바일 앱에서의 GUI 사용성을 실제 사용자들의 행동을 모델링 하여 정량적으로 측정하는 방안을 제안하고자 한다. 우리는 이전연구에서 GUI 사용성을 저해하는 요소를 GUI 이상징후(GUI Bad symptom)라 정의하였다. GUI 이상징후란 개발자가 예상한 사용자 행동과 실제 사용자 행동간의 차이에서 발생한다. 예로, ‘이메일 주소 입력’ 화면에서 사용자가 이메일을 입력할 것으로 예상했으나 40% 이상의 사용자들이 이전 화면으로 돌아가기 위해 다른 버튼을 여러 번 반복 조작할 때 ‘사용자 제스처 반복’, 이라는 이상징후가 발생한다. 본 논문에서는 GUI 사용성 저해요소 검출을 위해 사용자로부터 기록된 로그를 기반으로 모델링 후 GUI 사용성 저해 요소를 검출하는 기법을 기술하였다. 본 논문의 이하 구성은 다음과 같다. 먼저, 2 장에서는 모바일 앱에서의 GUI 테스팅에 대한 기존 연구들을 소개하고 있다. 3 장에서는 사용자로부터 수집한 로그를 기반한 모델링 과정을 기술하고 있다. 그 후 사용자로부터 얻은 행동 모델로부터 해당 GUI에 나타나는 이상징후를 예시를 소개하고 있다.



(그림 1) 실시간 GUI 이상징후 검출 과정

본 논문에서는 실제 개발자들이 GUI 문제로 간주하여 등록한 GUI 관련 이슈들을 바탕으로 이상 징후를 4 가지로 정의하고 그 검출과정을 모바일 앱 중 유저리티 도메인에서의 ‘사용자 제스처 반복’을 예시를 들어 기술하였다.

## 2. 관련 연구

효율적인 GUI 테스트 방법은 저-수준인 터치 기록/실행부터 고-수준인 모델 기반의 접근법이 존재한다. [1]의 연구는 모바일 디바이스의 터치 이벤트를 기록한 후 같은 화면에 이벤트를 실행하는 방법으로 GUI 기능성을 테스트하였다. 그러나 화면 해상도가 다양한 모바일 도메인에서는 실제 테스트를 통과함에도 불구하고 화면 크기가 다른 경우 해당 방법을 적용하는데 어려움이 있다. 모델 기반의 접근법은 화이트박스 방법론을 이용하여 GUI 내부 구조를 분석 후 테스트 케이스를 생성하는 방법을 사용한다. 아직 해결해야 될 문제점으로는 자동화된 방법은 GUI 가 시각적으로 문제가 없는지, 픽셀 단위 비교의 거짓 알람(False alarm) 등의 문제점을 해결해야 될 문제로 남아 있다. 이와 같은 기능성 테스트뿐만 아니라 GUI 사용성을 비교하는 테스팅 방법으로는 A/B 테스트가 존재한다. 그러나 사용자들의 반응을 얻기 위해서는 여러 버전으로 소프트웨어를 출시해야 되기 때문에 테스트에 소요되는 시간이 증가한다. 유사한 접근 방법으로 목표 질의 메트릭(Goal Question Metric)을 도입하여 사용자들에게 특정 시나리오를 수행할 때 소요시간, 인식성에 대한 설문조사를 통한 실험을 수행하였다[3]. 또한, 미학(Aesthetic)적 측정법[4]을 이용하여 화면 내 뷰(View)들의 정렬, 위젯 넓이/높이 등을 측정하여 GUI 사용성을 측정하였다. 그러나 화면 회전, 다양한 등이 지원되고 다양한 해상도를 지원해야 하는 모바일 앱의 경우 해당 측정법으로는 사용성을 정확히 측정하는 데 한계점이 존재한다.

우리는 선행 연구[5]에서 정의하였던 이상징후들을 사용성 저해 요소로 분류하고 이를 검출하기 위한 도구를 개발하였다. 이상징후 검출 도구는 사용자들의 행동을 모델링하여 개발자의 예상 행동 모델과 비교

하여 이상징후를 검출한다. 다음 3 장에서는 사용자들의 터치 모델을 어떠한 정보들을 수집하여 모델링 후 개발자가 예상한 행동 모델과 비교를 통해 이상징후를 검출하는 과정을 기술하고 있다.

## 3. GUI 이상징후 검출 기법

실사용자의 시스템 사용 행위를 기록한 로그 분석을 통해 사용자의 행위를 분석하기에 앞서, 모바일 앱 개발자는 GUI 설계 시 고려한 사용자의 예상 행위를 시나리오 단위로 기술한다. 사용자의 행위 시나리오란, 사용자가 특정 서비스를 제공받기 위해 거쳐 가야 할 앱의 대상 화면과 화면상에서 발생하는 행동모델을 의미한다. 개발자는 GUI 이상징후를 검출하기 위해 의논을 통해 앱의 전체 GUI구조에서 이상징후가 발생할 만한 GUI ID를 추출하여 해당 GUI를 조작하는 시나리오를 생성할 수 있다. 사용자들의 예상 시나리오를 생성하기 위해 개발자는 자신이 예상한 사용자의 행위대로 앱을 사용한다. 설계자가 행한 행위로부터 파생된 행동모델은 사용자로부터 얻은 로그파일을 분석할 때 대조 군으로 사용된다. 생성된 시나리오를 일반 사용자들에게 배포한 후, 사용자로부터 행위를 기록한 로그를 수집한다. 수집한 로그로부터 병합된 사용자 행동 모델을 생성한다. 예상 행동 모델과 사용자 행동 모델을 비교하여 분석을 실시한다.

개발자가 모바일 앱을 제작할 시 자신의 설계 의도대로 앱을 구동될 때 생성되는 행동 모델을 ‘예측 행동 모델’이라 한다. 사용자가 설계자의 의도를 전달받지 않은 상태에서 직관적으로 GUI를 이해한 후 앱을 사용할 때 생성되는 행동 모델을 ‘사용 행동 모델’이라 한다.

### 3-1. EFSM을 이용한 행동 모델링

소프트웨어 행동 모델을 나타내는 방법으로는 EFSM(Extended Finite State Machine)이 있다. EFSM은 기존의 FSM 이 상태와 상태간 전이에 대한 정보만을 표현할 수 있다는 한계를 극복하고자 추가적인 정보들을 인자(Parameter)로 만든 모델이다. EFSM은 소스

코드에 접근하지 않고도 동적으로 소프트웨어 행동 모델을 추출하는 특징을 지니고 있다. 이러한 특징으로 인해 소프트웨어 컴포넌트 간의 호출 관계를 모니터링하여 오류를 검출하고 분석하는데 이용된다[6]. 그러나 EFSM 소프트웨어 내부 컴포넌트간의 정보만을 표현하기 때문에 외부로부터 입력이 발생하는 GUI 컴포넌트의 행동 모델을 표현하기에는 부적절하다. 본 논문에서는 사용자들의 행동을 대표할 수 있는 모델을 만들기 위해 사용자들의 터치 시 x, y 좌표, 타임스탬프, GUI ID의 고유 값 정보들을 추가한 새로운 모델을 AFSM(Augmented Finite State Machine)이라고 명명하였다. 그림 1의 (a)는 터치 입력 정보를 모델링하는 예시를 제공하고 있다. 각 상태는 앱의 화면 상태의 바이너리 값을 의미하며 상태간 전이는 해당화면의 GUI(MainActivity.NextBtn)와 고유 ID(1690813)값, 액션의 종류(touch)로 표현된다. 사용자의 터치 행위의 방향에 따라 의미가 달라지는 경우, 터치 좌표까지 식별해야 한다. 또한, 터치 행위 발생시의 타임스탬프를 기록하여 다음 행위까지의 소요시간을 기록하고 있다.

### 3-2. 사용자 행동 모델 병합

하나의 AFSM 이 가진 상태의 개수가  $n$  개라 가정할 때, 상태 간 전이의 경우의 수를 고려하여 생성 가능한 행동 모델의 개수는  $n^{P_2}$  개이다. 그러나 이러한 행동 모델이 전부 다른 의도를 내포하고 있지는 않다. 즉, 다수의 사용자들은 개인별로 모바일 앱을 조작하는 특성을 가지고 있다. 따라서 사용자 행동 모델은 개인의 특성이 반영되어 모두 다른 형태일 수 있으며 이를 기반으로 한 AFSM 도 모두 다른 형태를 가질 수 있다. 따라서 각 사용자들이 아닌 전체 사용자들의 행동 모델을 대표하기 위해서는 각 사용자 별 AFSM 을 병합하는 과정이 필요하다. 병합 과정을 통해 서로 다른 형태의 행동 모델이지만 동일한 의도를 표현하고 있는지를 판단할 수 있다. 이러한 병합된 모델을 통해 사용성 저해 요소 검출의 성능을 향상시킬 수 있다. 이전 연구[5]에서는 각 사용자 별 행동 모델을 병합하는 방안을 제안하였다.

### 3-1. GUI 이상 징후 검출

실제 개발자들이 GUI 문제요소로 지적하는 요소를 정의하기 위해 Github 오픈소스 저장소에서 유필리티 모바일 앱 대상으로 등록된 GUI 이슈 174개 검색 결과를 바탕으로 총 4가지의 이상 징후로 분류하였다. 각 이상징후 별 정의는 다음과 같다.

1. 잘못된 GUI 위치: GUI의 위치가 부적절한 곳에 있어 사용자가 인식하기 어려움
2. 예상치 못한 사용자 제스처: 버튼, 리스트 등의 GUI에 사용자가 본래의 예상과 다르게 제스처 조작을 가하는 현상을 나타냄
3. 긴 지연시간: 사용자들에게 GUI가 정지한 것과 같이 인식됨
4. 사용자 제스처 반복: 개발자의 의도와 다르게 같은 GUI에 반복적 제스처로 조작 시도

이전 과정에서 병합된 사용자 행동 모델과 개발자가 예상한 행동 모델간의 비교를 통해 GUI 이상 징후를 검출할 수 있다.

\* Designed Behavior Model(DBM), User Behavior Model(UBM)

#### GetDifference(BM src, BM dst)

```

1: if src.size > dst.size
2:   return src.removeAll(dst)
3: else
4:   return dst.removeAll(src)

```

#### Analyze(DFM, UBM)

```

1: Transition diff = GetDifference(DBM, UBM)
2: for t in diff
3:   REPEAT = 4
4:   if t.getTarget() == DBM.GUI and
      t.getGesture() == DBM.gesture
5:     count = countSameTransaction(t, diff)
6:     if count > REPEAT and GetRatio(diff) > 0.4
7:       return REPEAT_GESTURE

```

#### GetRatio(t)

```

1: return t / wholeTransition.length()

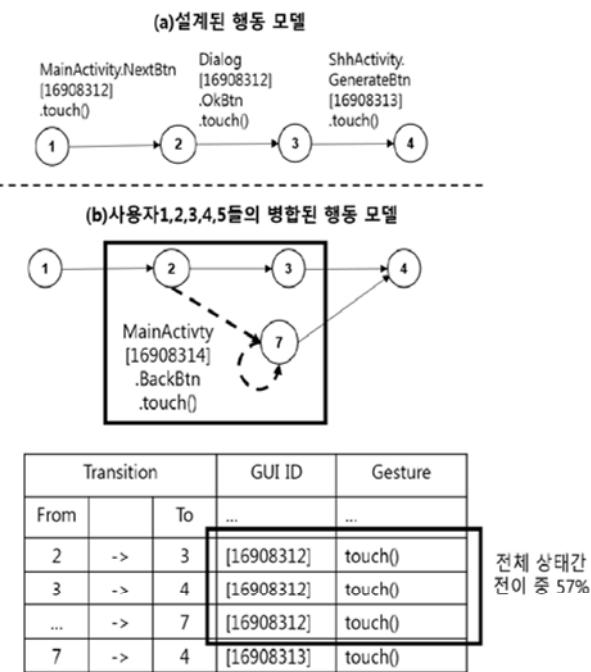
```

(그림 2) GUI 이상징후 검출 의사코드

그림 2 는 GUI 이상징후 중 ‘사용자 제스처 반복’ 이상징후를 검출하는 알고리즘의 의사 코드(Pseudo code)이다. 병합된 사용자 행동 모델과 예상 행동 모델과의 차이나는 부분 AFSM 집합을 추출한다 (Analyze 의 1 번 줄). 추출된 AFSM 에 포함된 모든 상태간 전이 중 예상되는 GUI 에 예상된 터치 제스처를 취한 상태간 전이 t 를 저장한다 (Analyze 의 2-4 번 줄). 해당 상태 간 전이가 전체 상태간 전이에서의 비율이 60% 이상이면서, 개발자가 설계한 반복 횟수를 초과하는 경우 ‘사용자 제스처 반복’ 이상징후가 발생하고 있다고 판단한다.

사용자 행동 모델 병합 시 추가되는 상태와 상태간 전이를 포함하는 전략으로 병합을 수행하였기 때문에 사용자 행동 모델이 비 결정적(non-deterministic) 형태로 생성된다. 이와 같이 다수의 사용자 의도를 나타내기 때문에 병합된 사용자 행동 모델과 완전 일치하지 않는 경우라고 해서 이상징후로 검출되는 것은 아니다. 사용자마다 모바일 앱 사용의 능숙도, 나이, 구동하는 모바일 디바이스의 해상도 등의 요소에 따라 사용자 행동 모델이 다르게 나타난다. GUI 사용성 시나리오 성공 기준은 해당 GUI 가 속한 시나리오를 수행할 때, 사용자의 60% 이상이 예상 시간 내 수행하는 것을 기준으로 한다[7]. GUI 이상징후 유무를 판별 기준은 예상 행동 모델과 사용 행동 모델이 차이 나는 상태와 상태간 전이가 전체 사용행동 모델 중 40% 이상인 경우 이상징후라 판별할 수 있다.

그림 3은 Shapp 앱에서 ‘이메일 주소 입력’ 시나리오에서 Dialog 에 GUI 이상징후가 발생하는지에 대한 이슈에서 추출한 ‘사용자 제스처 반복’ 이상징후이다. Dialog 에서 실제로 사용성 저해요소가 발생하는지



(그림 3) ‘사용자 제스처 반복’ 이상징후 검출 예시

검증하기 위해 개발자의 원래 의도대로 시나리오를 생성하였다. 개발자가 예상한 사용자 행동 모델은 MainActivity 화면에서 이메일 정보, 비밀번호를 입력하여 ShhActivity 화면으로 전환 후 GenerateBtn을 터치하는 것이다. 개발자는 이메일 입력 후 ShhActivity 화면으로 넘어갈 것을 예상하였으나 MainActivity에서의 로그인을 시도할 시 다이얼로그가 화면에 나타나면서 로그인을 수행하는데 필요한 정보를 가리게 되어 사용자의 인지 범위에서 벗어나게 된다. 그리하여 사용자는 MainActivity화면으로 다시 돌아가기 위해 CcancelBtn 버튼을 반복적으로 누르게 된다.

실제로 병합된 사용 행동 모델은 그림3의 (b)와 상태2 부터 상태7까지의 상태 전이가 발생하는 부분이 개발자가 예상한 행동 모델과 차이를 보인다. 차이가 발생한 부분 AFSM 집합을 추출한 후 포함된 상태 간 전이마다 터치 대상과 제스처를 예상 행동모델과 비교한다. 검색하는 행동 모델과 일치하는 경우, 전체 상태 간 전이 중의 비율을 구한다. 해당 상태 간 전이의 비율이 40%이상이고 개발자가 설계한 기준 제스처 반복횟수 4번을 초과하는 경우(그림2의 Analyze 2-5줄) ‘사용자 제스처 반복’이라고 판단하고 있다. 이와 같이 57% 이상의 사용자 행동은 MainActivity 화면에서 BackBtn의 touch() 제스처를 개발자가 설계한 기준치인 4번이상 수행하는 반복되는 형태를 보이고 있다. 개발자의 의도와는 달리 57%의 사용자가 실제 사용 시나리오와 달리 앱을 조작하고 있는 것이다. 이처럼 개발자의 예상과 달리 사용자가 반복되는 제스처를 행하는 것을 ‘사용자들의 반복된 제스처’ 이상징후라 하였다. 이러한 현상은 사용자가 특정 GUI를

반복적으로 조작하는 현상이 발생해 목표된 작업을 수행하는데 효율성을 저해하고 시간을 추가로 소모하게 하여 사용성을 저해시킨다.

#### 4. 결론

모바일 앱은 다양한 화면 크기, 제스처 등으로 인해 사용자들에게 혼란을 야기할 수 있다. 개발자들은 이러한 문제를 해결하기 위하여 GUI 사용성 테스트에 많은 시간을 할애한다. 본 연구에서는 개발자들의 효율적인 사용성 테스트를 위한 반-자동화된 GUI 사용성 저해요소 검출 기법을 제시하였다. 앱 도메인마다 조작 특성, GUI 구성이 다르므로 각 도메인 별 이상징후를 정의할 필요가 있다. 추후 연구에서는 이상징후가 실제로 사용성에 미치는 영향을 파악하고 검출한 이상 징후를 바탕으로 사용성 향상 기법을 제안하고자 한다.

- 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업 (No. 2012M3C4A7033348)과 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.10044457, 자율지능형 지식/기기 협업 프레임워크 기술 개발)

#### 참고문헌

- [1] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. 2013. RERAN: timing- and touch-sensitive record and replay for Android. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)
- [2] Ying-Dar Lin; Rojas, J.F.; Chu, E.T.-H.; Yuan-Cheng Lai, "On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices," in Software Engineering, (TSE'14), vol.40, no.10, pp.957-970, Oct. 1 2014
- [3] Azham Hussain and Elaine Ferneley. 2008. Usability metric for mobile application: a goal question metric (GQM) approach. In Proceedings of the 10th International Conference on Information Integration and Web-based Applications \& Services (iiWAS '08)
- [4] Mathieu Zen. 2013. Metric-based evaluation of graphical user interfaces: model, method, and software support. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '13). 183-186.
- [5] 마경욱, 박수진, “GUI 자가 적응을 위한 모바일 앱의 사용성 저해요소 자동 검출 기법,” 한국 소프트웨어공학 학술대회 논문집 제 17 권 제 1 호, p.411-p.412, 2015
- [6] Mariani, L.; Pastore, F.; Pezze, M., "Dynamic Analysis for Diagnosing Integration Faults," in Software Engineering, (TSE'14), vol.37, no.4, pp.486-508, July-Aug. 2011
- [7] Dolstra, E.; Vliegendaal, R.; Pouwelse, J., "Crowdsourcing GUI Tests," in Software Testing, Verification and Validation (ICST'2013), vol., no., pp.332-341, 18-22 March 2013
- [8] <http://www.nngroup.com/articles/why-you-only-need-5>