KIPS Tr. Software and Data Eng.
Vol.5, No.7 pp.319~326 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.7.319

Automatic Detection of Usability Issues on Mobile Applications

Kyeong Wook Ma[†] · Sooyong Park[††] · Soojin Park[†††]

ABSTRACT

Given the attributes of mobile apps that shorten the time to make purchase decisions while enabling easy purchase cancellations, usability can be regarded to be a highly prioritized quality attribute among the diverse quality attributes that must be provided by mobile apps. With that backdrop, mobile app developers have been making great effort to minimize usability hampering elements that degrade the merchantability of apps in many ways. Most elements that hamper the convenience in use of mobile apps stem from those potential errors that occur when GUIs are designed. In our previous study, we have proposed a technique to analyze the usability of mobile apps using user behavior logs. We proposes a technique to detect usability hampering elements lying dormant in mobile apps' GUI models by expressing user behavior logs with finite state models, combining user behavior models extracted from multiple users, and comparing the combined user behavior model with the expected behavior model on which the designer's intention is reflected. In addition, to reduce the burden of the repeated test operations that have been conducted by existing developers to detect usability errors, the present paper also proposes a mobile usability error detection automation tool that enables automatic application of the proposed technique. The utility of the proposed technique and tool is being discussed through comparison between the GUI issue reports presented by actual open source app developers and the symptoms detected by the proposed technique.

Keywords : Mobile Application, GUI Usability, User Behavior Model, Automatic Detection of GUI Bad Symptom

모바일 앱에서의 사용자 행동 모델 기반 GUI 사용성 저해요소 검출 기법

마 경 욱[†] · 박 수 용[††] · 박 수 진[†††]

요 약

어플리케이션의 구매 결정 소요시간이 짧은 동시에 구매 취소 역시 간편한 모바일 앱의 속성을 고려했을 때, 사용 편리성은 모바일 앱이 제공해야 할 다양한 품질 요소들 중 상위의 우선순위를 가지는 요소라 할 수 있다. 이러한 배경에서 모바일 앱 개발자들은 앱의 상품성을 저하시키는 사용성 저해 요소를 여러 가지 측면에서 최소화시키는데 많은 노력을 기울이고 있다. 모바일 앱의 사용 편리성을 저해하는 대부분의 요소는 GUI 설계 시에 발생되는 잠재적인 오류들로부터 기인한다. 우리는 앞선 연구에서 사용자 행위 로그를 이용한 모바일 앱의 사용성 분석 기법을 제안한 바 있다. 본 논문에서는 앞선 연구 결과를 토대로 사용자 행위로그를 유한 상태 모델로 표현하고, 여러 명의 사용자로부터 추출된 사용자 행위모델을 병합하여 설계자의 의도가 반영된 설계 행위모델과 비교해 나감으로써, 체계적으로 모바일 앱의 GUI 모델상에 잠재된 사용성 저해 요소 검출해 내는 기법을 제안하고 있다. 또한 기존 개발자들이 사용성 오류 검출을 위해 행해왔던 반복적인 테스트 작업의 부담을 줄이기 위해, 본 논문에서는 제안된 기법의 자동화가 가능하도록 하는 사용성 오류검출 자동화 도구를 함께 제안하고 있다. 제안된 기법과 도구의 효용성은 실제 오픈 소스 앱 개발자들에 의해 제기된 GUI 이슈 리포트와 제안된 기법에 의해 검출된 이상징후들 간의 비교를 통해 논의하고 있다.

키워드 : 모바일 어플리케이션, GUI 사용성, 사용자 행동 모델, GUI 이상징후 자동 검출

1. 서 론

스마트폰과 태블릿을 지원하는 모바일 앱은 기하 급수적으로 그 숫자가 증가하고 있다. 따라서 동일한 기능을 제공하는 다수 개의 앱 중에서 사용자들은 품질에 따라 선택하여 사용할 수 있다. 모바일 앱의 품질요소로는 여러 가지가 있을 수 있으나 일반적으로 모바일 앱 사용자들은 특별한 매뉴얼 없이도 직관적으로 사용하기 쉬운 앱을 선호한다. 생활에서 사용되는 모바일 앱들 중에서 이러한 사용자의 요구사항을 정확히 파악하여 많은 수의 사용자를 확보하는 앱이 있는가 하면 그렇지 못한 사례들도 있다. 일례로 기차 좌석 예매 앱의 경우, 기차표 예매 시 예매하고자 하는 좌석 수를 지정하는 화면에서, 탑승할 어른과 어린이의 숫자

※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No. 2012M3C4A7033348).
† 준 회 원 : 서강대학교 컴퓨터공학과 석사과정
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
††† 종신회원 : 서강대학교 서강미래기술연구원 교수
Manuscript Received : April 26, 2016
Accepted : May 9, 2016
* Corresponding Author : Soojin Park(psjdream@sogang.ac.kr)

를 화면 왼쪽의 아이콘을 오른쪽으로 밀어서 지정하도록 화면이 설계되었다. 그러나 이러한 지정 방식은 직접 숫자를 패드를 통해 입력하는 기존 앱들의 화면설계와 상이하여 실제 많은 사용자들이 사용 후기에서 불편함을 표현하고 있다. 결과적으로 이러한 불편함은 해당 앱이 사용자들에 의해 2.4점(5점 만점)이라는 저조한 평가를 받고 있는 것과 무관하지 않다. 이와 같은 사례에서 모바일 앱의 전반적인 평가와 직결될 수 있는 모바일 앱의 화면 설계가 사용성 측면에서 문제점을 내포하고 있지 않은지를 평가하여 이를 해소시켜 나감으로써 모바일 앱의 사용성(Usability)을 향상시켜 나가야 할 필요성을 찾아볼 수 있다.

본 연구에서는 사용자들이 모바일 앱을 사용하는 행동 모델과 모바일 앱의 GUI 개발자들이 의도한 행동 모델간의 차이점을 분석하여 이로부터 미리 정의된 네 가지 유형의 사용성 저해 요소를 자동으로 검출해 내는 기법을 제안한다. 이러한 자동화된 모바일 앱의 사용성 저해요소 검출 도구의 적용을 통해 기존의 사용성 측정 방식이 가지는 객관성 결여 문제와 비용을 요하는 테스팅 부담을 일정 부분 해결할 수 있을 것으로 기대할 수 있다.

우리는 이미 선행연구[1]를 통해 사용자 로그 분석을 통한 사용성 분석 기법을 제시한 바 있다. 본 연구는 [1]의 연구결과를 바탕으로 하였으나 수집된 사용자 로그 정보로부터 유한상태기계(Finite State Machine) 모델 기반의 행동모델을 추출하는 방법을 제시하고 다수의 사용자들의 로그로부터 추출된 행동모델을 병합하는 방법을 제시함으로써 특정 모바일 어플리케이션의 대다수의 사용자가 느끼는 보편적인 사용성 측면의 문제점 분석이 자동으로 가능하다는 점에서 차별점을 찾을 수 있다.

본 논문의 이후 구성은 다음과 같다. 2장에서는 모바일 앱의 사용성 측정과 관련된 관련연구들을 살펴본 후, 3장에서는 본 논문에서 제안하고 있는 모바일 앱의 사용성 저해요소를 사용자 행동 모델 분석을 통해 자동 검출해 내기 위해 필요한 접근 방법을 설명한다. 4장에서는 사용성 저해요소 검출 도구의 효용성과 사용성 저해 요소가 실제 사용성에 미치는 영향을 측정하기 위해 설계된 실험의 결과를 제시하고 있다. 마지막으로, 5장에서는 본 연구의 결론과 향후 연구계획을 밝히고 있다.

2. 관련 연구

이전의 연구[2-4]에서는 시스템의 사용성을 측정하고 향상시키기 위한 방식으로 주로 사용자를 대상으로 설문조사를 수행하였다. 하지만 사용자 설문을 기초자료로 사용성을 평가하는 방식은 설문 문항의 구성방식, 응답자의 특성 등의 변수에 따라 그 결과가 상이할 수 있다는 문제점을 가지고 있다. A/B 테스트[5]는 사용자들에게 똑같은 페이지/내용을 제공하지만, 메시지의 구체적인 형식이나 디자인에 변형을 주어서 각자의 반응을 살피는 테스트 방식이다. 다양한 GUI 대안에 대한 사용자들의 반응을 얻기 위해서는 대안 숫자만큼의 버전으로 소프트웨어를 제작해야 되기 때문에 테스트에 소요되는 시간이 증가한다. 또한, GUI의 편리한 정도를 미학(Aesthetic)적 측면에서 평가하는 연구들도 존재한다[6]. 즉, 화면 내 뷰(View)들의 정렬 정도, 위젯 넓이/높이 등을 측정하여 사용자가 편안하다고 느끼는 정도를 가늠할 수 있다. 그러나 다양한 이기종(Heterogeneous) 기기를 지원해야 되는 모바일 앱의 특성상 해당 측정법으로는 기기의 해상도에 따라 GUI의 배치가 달라지므로 같은 앱일지라도 메트릭 수치가 변하여 정확한 사용성 측정이 어려운 한계점이 존재한다.

본 연구는 모바일 앱의 사용성의 문제점은 개발자가 예상한 사용자 행동 모델과 실제 사용자들의 사용 행동에서 기록된 사용자 행동모델간에 차이점이 발생하는 지점에 내재해 있다는 것을 정의하는 것으로부터 시작되었다[7]. 이러한 사용성 검증을 위한 기준을 제시함으로써 객관적인 메트릭을 확보할 수 있으며 개발자의 의도를 나타내는 모델과 사용자의 행동 모델 비교를 통한 자동화된 사용성 저해 요소 검출이 가능함에 따라, 기존 연구들의 문제점으로 지적된 자의적 해석 가능성과 고비용 문제를 완화시킬 수 있을 것으로 기대한다.

3. GUI 사용성 저해 요소 검출 도구 (RUID: Real-time Usability Issue Detector)

본 장에서는 제안하는 기법을 자세히 설명하기에 앞서 동작중인 모바일 앱으로부터 사용자 로그를 추출하는 메커니즘 및 제안된 기법을 구현하는 자동화 도구를 구성하는 주요 컴포넌트들과 컴포넌트들간의 협업을 통해 이뤄지는 모바일 앱 저해요소 자동 검출 과정에 대한 개요를 설명하고자 한다.

Fig. 1은 GUI 사용성 저해 요소 검출 도구(RUID: Real-time Usability Issue Detector)를 구성하는 각 컴포넌트를 도식화하고 있다. RUID는 Java로 구현되었으며 안드로이드 플랫폼에서 동작하며, Fig. 1에 기술된 RUID구성 컴포넌트별 주요 역할은 다음과 같다.

(a) Probe: Android에 내장되어 사용자로부터 발생한 터치를 기록하여 Touch log storage(Fig. 1의 (b))에 저장한다.

(b) Touch log storage: Android 매크로인 Monkeyrunner를 이용하여 사용자들의 터치 좌표, 타임스탬프, 제스처 종류를 수집한 후 Probe에서 생성된 로그와 타임스탬프 기반으로 합성된 로그를 저장한다.

(c) View storage: 검사의 대상이 될 GUI ID 정보를 획득해야 하기 위해 Android 도구인 Hierarchical viewer를 이용하여 현재 화면의 전체 GUI ID 값이 포함된 구조 정보를 XML형태로 저장한다.

(d) Behavior model generator: (b), (c)에서 수집한 로그를 바탕으로 사용자의 행동을 나타내는 모델을 생성한다.

(e) Bad symptom analyzer: 모바일 어플리케이션의 사용성 저해요소를 몇 가지 타입으로 정의하여 검출한다. 설계

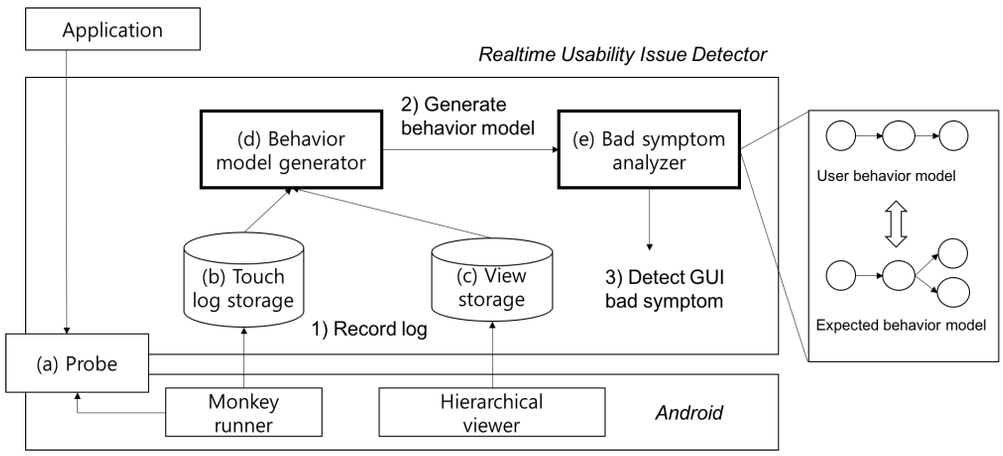

Fig. 1. RUID: Real-time Usability Issue Detector

시점에서 가정한 예측행동 모델과 실제 사용자로부터 수집된 로그를 기반으로 생성되는 실제행동 모델간의 차이점을 분석하여, 사용성을 저해하는 GUI 이상징후(Bad symptom) 유무를 판단한다.

4. 모바일 앱 GUI 이상징후 자동검출 기법

3장에서 소개된 RUID의 구성 컴포넌트들 중에서 실제 모바일 앱의 GUI 이상징후 자동 검출 기법의 구현부에 해당되는 컴포넌트는 Behavior model generator(Fig. 1의 (d))와 Bad symptom analyzer(Fig. 1의 (e))이다. Behavior model generator는 사용자의 터치로그에 기록된 정보를 기반으로 사용자 별 행동모델을 생성하고 이를 병합하여 하나의 모델을 생성하면, 하나로 통합된 사용자 행동모델(User Behavior Model)과 설계자의 GUI 설계 의도를 표현한 예상행동모델(Expected Behavior Model)을 비교하여 상이한 점들로부터 GUI 이상징후를 검출해 내는 것은 Bad symptom analyzer이다.

본 기법에서 GUI 이상징후 검출을 위해 비교 분석 대상이 되는 두 가지 모델인 사용자 행동 모델과 예상행동 모델을 생성하는 방식은 동일하다. 두 모델간에 상이한 점은 모델 생성의 소스가 되는 사용자 행위 로그를 발생시키는 주체가 GUI 설계자인지 실사용자인지에 따라 예상행동 모델과 사용자 행동모델이 구분된다. GUI 설계자가 자신의 디자인 의도에 따라 예상되는 시나리오에 따라 모바일 앱을 사용하면서 발생되는 행위 로그가 실제 사용자의 모바일 앱 사용 시 감지되는 행위 로그가 일치한다는 것은 GUI 설계자가 실사용자의 요구사항(Need)을 잘 파악하여 모바일 앱을 설계하였음을 의미한다. 반면, GUI 설계자의 행위로그로부터 추출된 예상행동모델과 실제 사용자 행동모델이 상이하다는 것은 설계자의 모바일 앱 GUI 설계가 사용자의 경험(User Experience)과는 다른 형태임을 의미한다. 이러한 두 모델간의 상이한 부분 중에서 우리는 몇 가지의 공통적인 패턴화 가능한 모바일 앱 GUI 이상징후 타입들을 식별해 낼 수 있다.

이와 같이 모바일 앱 상에 존재하는 사용성 저해요소는 Fig. 2에서 보이는 바와 같이, 크게 세 가지 단계를 거쳐 자동으로 검출 가능하다. 첫째, ① 행위로그로부터 의미 있는 정보들을 추출하여 행위모델을 생성하는 단계, 둘째, ② 다

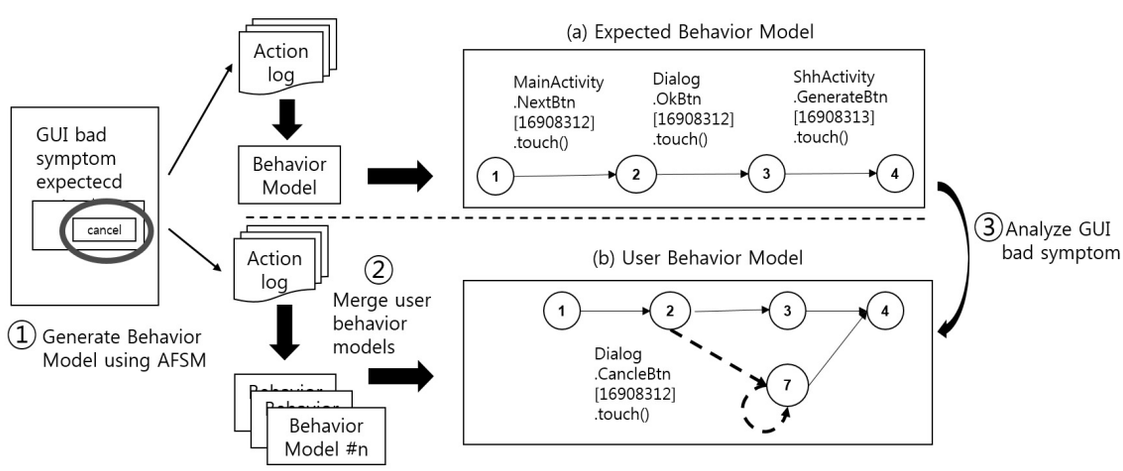

Fig. 2. Process of usability issue detection

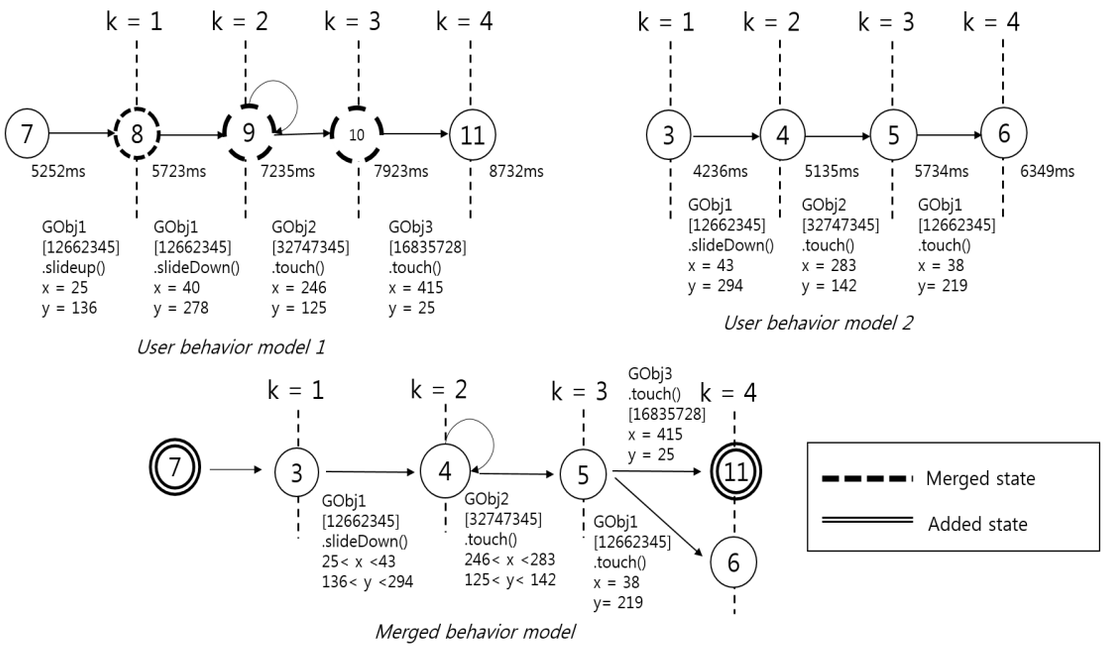

Fig. 3. Merging process of two user behavior model using Gk-tail algorithm. k=2, strong subsumption applied

수의 사용자 행위로그부터 생성된 다수의 행위모델을 하나의 모델로 병합하는 단계, 그리고 마지막으로 ③ 병합된 사용자 모델과 예상행동모델간의 비교를 통해 상이점을 분석하여 GUI 이상징후를 검출하는 단계이다. 각 단계에 대한 좀더 자세한 설명은 다음과 같다.

4.1 터치로그로부터 사용자 별 행동모델 생성

먼저, Fig. 2에서 보이듯이 ① 모바일 앱에서 발생하는 이벤트들은 기록되어 로그로 저장된다. 이때 기록의 단위는 하나의 서비스 시나리오이다. 서비스 시나리오란 특정 서비스 결과물을 제공 받기까지 행해지는 사용자와 모바일 앱 간의 상호행동을 시간 순으로 나열한 일련의 흐름을 의미한다.

사용자가 ①에서 저장된 로그는 개발자가 의도한 행동과 실제 사용자가 행하는 행동 간의 유사도 분석을 위한 유한 상태 다이어그램(Finite State Diagram)을 기반으로 하는 모델 형태로 전환된다.

사용자와 모바일 앱 간의 상호작용을 나타내기 위해서는 특정 모바일 앱의 상태를 표현하면서 사용자의 조작에 반응하여 다른 상태로 어떻게 전이되는지를 표현하는 정형화된 모델이 필요하다. 우리는 이전 연구[7]에서 모바일 앱과 사용자간의 상호작용을 모델링하기 위하여 GUI 구성요소의 ID, 터치 좌표의 x, y값, 사용자 이벤트 발생 시간을 나타내는 타임스탬프, GUI 화면 이미지 바이너리 값과 같은 요소들을 추가적으로 표현 가능하도록 기존의 모델을 설계하여 AFSM(Augmented Finite State Machine)이라 명명하였다. 어떠한 특정 GUI객체에 대해 가해지는 사용자의 터치의 방향성 및 형태를 파악할 수 있으며 각 이벤트 발생 시 타임스탬프를 기록함으로써 액션 간 소요시간 등을 분석하여 GUI에 잠재적으로 존재하는 문제점 파악에 도움을 줄 수 있다. Fig. 3의 상단에 있는 두 다이어그램은 서로 다른 두 명의 사용자로부터 수집된 행위로그를 기반으로 생성한 두 개의 서로 다른 AFSM 모델들이다. 각 전이(Transition)간에는 발생 이벤트의 명칭(slideup())과 해당 이벤트가 발생한 GUI 컨트롤의 ID(GObj1) 등이 표시되어 있으며 터치 좌표의 x, y값, 사용자 이벤트 발생 시간을 나타내는 타임스탬프, GUI 화면 이미지 바이너리 값 등이 추가적인 정보로 포함되어 있다.

4.2 사용자 행동 모델 병합

동일하게 설계된 모바일 앱이라 하더라도 사용자의 사용 방식은 개인별로 상이하다. 따라서 동일한 서비스 시나리오를 수행하는 사용자 행동 모델들은 행동 주체인 개인의 특성이 반영되어 모두 각기 다른 형태의 AFSM모델로 표현된다. 따라서 생성된 다수의 서로 다른 사용자 행동 모델은 부분적으로 서로 달라 보일지라도, 실제 포함하고 있는 행동 자체는 동치관계에 있을 수 있다. 각 사용자들이 아닌 전체 사용자들의 행동 대변하기 위해서는 예상행동모델과의 비교분석에 앞서 여러 개의 사용자 행동모델을 병합하여 하나의 모델로 통합하는 작업이 필요하다. FSM(Finite State Machine)들 간의 동치관계를 파악하여 여러 개의 FSM을 하나의 FSM으로 병합하는 알고리즘으로 Gk-tail 알고리즘[8]이 있다. Gk-tail 알고리즘의 경우, 특정 소프트웨어 시스템에 국한되지 않고 일반적인 FSM모델들을 하나로 병합 가능하다는 점에서 우리가 모바일 앱 사용자 행위모델 표현법으로 확장시킨 AFSM 의 병합에 적용 가능한 알고리즘으로 판단하여 선택하였다. Gk-tail알고리즘은 두 개의 서로

다른FSM(Finite State Model)모델을 비교하여 서로 동치인 상태전이(Transition) 부분을 찾아낸 후, k값이 지시하는 병합모델의 마지막 상태전이를 순차적으로 FSM에 추가해 나가면서 하나의 병합모델을 완성해 나간다. 제안된 AFSM 역시 기본적인 FSM의 확장된 표현방법이므로 Gk-tail 알고리즘 적용이 가능하다. Fig. 3은 두 명의 사용자로부터 수집된 로그를 기반으로 생성된 두 개의 사용자 행동모델을 Gk-tail 알고리즘을 적용하여 병합해 나가는 과정을 예시하고 있다. 먼저, 두 개의 AFSM의 상태전이를 비교해 나가면서 동일한 상태전이(동일한 GUI 객체에 동일한 이벤트 발생)를 검색해 나간다.

예를 들어, Fig. 3의 GOjb1.slideDown( ) 이벤트와 GObj2.touch( ) 이벤트가 두 AFSM모델(*사용자행동모델1*과 *사용자행동모델2*)에서 중복되어 검출되면, 동일한 상태 전이 흐름의 첫 번째 상태(State)의 k값을 1로 설정한다. Fig. 3의 *사용자행동모델1*의 ⑧번상태와 *사용자행동모델2*의 ③번상태가 각각 k값이 1로 설정되었음을 알 수 있다. k=1로 설정된 ⑧번과 ③번 상태에서 ⑨번, ④번 상태로의 전이를 일으키는 이벤트와 이벤트가 발생되는 객체가 일치할 경우, k값을 1씩 증가해 나가면서 다음 상태로 비교하면서 이동해 나간다. Fig. 3의 경우, 이러한 AFSM 모델 순회(Traverse)가 k=3까지 진행됨을 알 수 있다. k값 1에서 3까지 전이가 일치한 상태들(*사용자행동모델1*의 ⑦, ⑧, ⑨번 상태와 *사용자행동모델2*의 ③, ④, ⑤번 상태)은 동치상태로 단일한 하나의 상태로 각각 병합된다.

그러나 k=3 이후의 전이는 각 모델에서 서로 다른 이벤트 발생에 의한 전이들이다. 따라서, 서로 다른 전이에 의해 도달할 수 있는 상태인 *사용자 행동 모델1*의 ⑪번 상태와 *사용자 행동 모델2*의 ⑥번 상태는 하나로 병합할 수 없다. 이러한 서로 다른 상태는 병합된 모델의 ⑤번 상태에서 전이 가능한 서로 다른 상태로 각각 추가되며, ⑤번 상태에서 다음 상태로 ⑥번 상태와 ⑪번 상태 중 어떤 상태로 전이가 일어날 지는 발생하는 이벤트에 따라 결정된다.

이와 같이 Gk-tail 알고리즘을 사용자 행동 모델 병합에 반복적으로 적용함으로써 n명의 서로 다른 사용자들로부터 추출된 n개의 행동 모델을 하나의 행동모델로 나타낼 수 있다.

4.3 모바일 앱 사용성을 저해하는 이상징후 검출

사용자 행동 모델들이 하나로 통합되면 개발자의 설계의도를 포함하고 있는 예상행동모델과 사용자 행동모델을 비교 분석하여 사용성 저해요소를 검출한다. 단순히 두 모델간의 차이점이 발견된 부분을 앱 상에 존재하는 사용성 저해요소로 판별하지 않고 일정 임계치 이상의 사용자가 다른 형태의 행동을 보이는 경우를 GUI 이상징후로 판별한다.

우리는 자동검출이 가능한 모바일 앱 사용성을 저해하는 이상징후들의 타입을 정의하기 위하여 오픈소스 앱의 개발자 커뮤니티에 등록된 GUI 관련 이슈들을 분석하여 그룹화하였다. 그 중에서 사용자 행동모델과 예상행동모델간의 비교 분석을 통해, 자동 검출이 가능한 4가지 타입의 사용성 저해하는 요소들인 이상징후들을 다음과 같이 정의하였다.

이상징후 1. 예상치 못한 행동 흐름
이상징후 2. GUI 컨트롤에 구현되지 않은 이벤트 발생
이상징후 3. 반복적으로 수행되는 상태전이
이상징후 4. 예상 수행 시간 초과

이 같은 4 가지 타입의 이상징후들을 검출하기 위한 알고리즘을 의사코드로 나타내면 Fig. 4와 같다. AFSM 모델로 표현된 설계자의 의도를 표현한 행동모델인 예상행동모델과 여러 명의 사용자로부터 추출된 AFSM 모델들을 하나로 병합한 사용자행동모델을 각각 인자로 받아들여 두 모델들 간에 존재하는 차이를 이상징후 타입 별로 검출하기 위한 알고리즘이 각각 표현되어 있다.

```
GUI bad symptom 1
UNEXPECTED_GESTURE(AFSM ebm, AFSM ubm,
GUIObj target)
1: diff = get difference of ubm, ebm.
2:   loop diff[0…n]
3:     if diff[i].action not in ebm & GetRatio(diff[i]) > 0.4
3: return UNEXPECTED_GESTURE

GUI bad symptom 2
UNEXPECTED_SEQUENCE(AFSM ebm, AFSM ubm,
GUIObj target)
1: diff = get difference of ubm, ebm.
2:   loop diff[0…n]
3:    if diff[i - 1].destination != target.source &
      GetRatio(diff[i]) > 0.4
4:       return UNEXPECTED_SEQUENCE

GUI bad symptom 3
REPEATION_OF_GESTURE(AFSM ebm, AFSM ubm)
1: diff = get difference of ubm, ebm
2:    loopdiff[0…n]
3:       if diff[i-1].action()=diff[i].action()
4:    & GetRatio(diff[i]) > 0.4 & count(diff[i]) > TRHESHOLD
5: return REPEAT_GESTURE

GUI bad symptom 4
LONG_LATENCY(AFSM ebm, AFSM ubm)
1: equivalent = get equivalent transition of  ubm, ebm
2:   if sum of equivalent's latency > ebm's latency * 2
3:     return LONG_LATENCY
```

Fig. 4. GUI bad symptom detection pseudo-code

앞서 정의한 4가지 타입의 모바일 앱의 사용성을 저해하는 이상징후를 검출하기 위한 알고리즘을 간단한 예제 모바일 앱을 대상으로 설명하면 다음과 같다. 다음 Fig. 5는 오픈소스 원격 리모컨 컨트롤러 모바일 앱 중 하나인 MyRemocn으로 원격으로 조작 가능한 새로운 기기를 등록하고 리모콘의 설정을 수정하기까지 GUI 설계자가 예상한 행동모델과

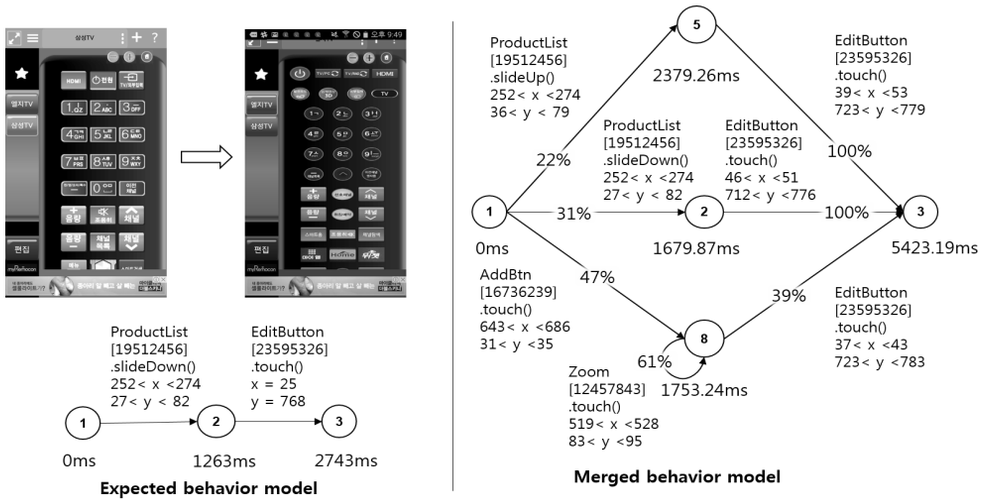

Fig. 5. Example of GUI bad symptom detection

실제 사용자 행동모델의 단편을 보여준다. Fig. 5의 좌측 예상행동모델은 ProductList 객체에서 slideDown 액션을 수행한 후, EditButton 객체에서 터치 액션을 수행하는 형태를 지니고 있다. 우측의 병합된 사용자 행동 모델은 4장에서 소개한 병합과정을 거쳐서 생성된 행동 모델이다. 각 상태 간 전이는 터치 좌표, 액션 정보를 포함하고 있다. 또한 각 상태마다 기록된 타임스탬프는 사용자 별 평균 타임스탬프를 기록하였다.

Fig. 5에 기술된 예상행동모델과 실제 사용자 행동 모델 간의 상이점과 각각의 상이점으로부터 감지 가능한 이상징후의 타입들은 다음과 같다.

이상징후 1. 예상치 못한 행동 흐름: GUI 설계자가 기대한 예상행동모델 상에서의 행동 흐름은 상태 ①부터 ③까지 순차적으로 진행되지만, 실제 사용자들의 행동 모델을 병합한 결과 22%의 사용자 행동모델에서 상태 ① → 상태 ⑤ → 상태 ③에 이르는 예상되지 않은 행동 흐름이 감지되었고, 또 다른 47%의 사용자 역시 상태 ① → 상태 ⑧ → 상태 ③으로의 예상되지 않은 행동 흐름을 보였다. 즉, 이벤트 로그 수집 대상인 전체 사용자의 69%가 Fig. 5의 상단에 정의된 설계자가 의도한 예상행위모델에는 정의되지 않은 행동흐름을 보였다. 실험적으로 60%의 사용자가 어플리케이션을 사용할 때, 사용성에 문제가 없다고 간주할 수 있다[2]. 즉, 40% 이상의 사용자들이 설계의도와 다른 행위를 보인다는 것은 사용자들이 해당 앱 사용에 불편함을 느끼고 있다는 것을 의미하므로 해결해야 할 사용성 이슈로 판단 가능하다.

5장의 실험 결과는 이상징후 판단의 임계값을 0.4로 설정한 결과이다. 전체 사용자 수의 40% 이상의 사용자로부터 추출된 행위모델에서 이상징후로 판별하였다.

이상징후 2. GUI 컨트롤에 구현되지 않은 이벤트 발생: Fig. 5의 예상행동모델에서 정의된 이벤트는 ProductList 객체에서 발생하는 slideDown( ) 액션과 EditButton 객체에서 발생하는 touch( ) 액션, 두 가지이다. 그러나 사용자 행동모델에서는 이미 이상징후 1에서 감지되었던 정의되지 않은 행동흐름 상의 전이를 유발하는 ProductList.slideUp( ), Zoom.touch( ) 등의 GUI 모델상에 포함되어 있지 않은 이벤트를 사용자가 발생시키고 있음을 알 수 있다. 이 같은 이상징후 타입이 감지될 경우, 사용자 행동모델에 포함된 이벤트에 대한 추가적인 구현을 고려해야 할 필요가 있다.

이상징후 3. 반복적으로 수행되는 상태전이: Fig. 5의 예제에서는 상태 ①에서 상태 ⑧로의 전이 후 상태 ⑧에서 Zoom.touch( ) 액션을 계속해서 발생시키는 사용자가 61%임을 알 수 있다. 이같이 사용자가 동일한 이벤트를 계속해서 발생시킨다는 것은 사용자가 기대하는 서비스가 모바일 앱에서 제대로 제공되고 있지 않음을 의미한다. 이와 같은 이상징후 3의 경우, 해결을 위해 단순한 GUI 설계의 조정 외에도 실제 모바일 앱 서비스 시나리오 상에 오류가 없는지 검토해야 할 필요가 있다.

이상징후 4. 예상 수행시간 초과: 예상 행동모델과 사용자 행동모델 중 동치인 상태간 전이 집합을 저장한다. 동치관계인 예상행동모델의 상태전이의 총 소요시간보다 사용자 행동 모델의 총 소요시간이 2배 이상 소요될 경우, 네 번째 타입인 예상 수행시간 초과 이상징후가 감지된 것으로 판단한다. Fig. 5의 예제 상에서는 상태 ① → 상태 ② → 상태 ③에 이르는 동치 전이 수행 시간이 예상행동모델의 경우 2743 ms인데 반해 실제 사용자들이 수행한 평균수행시간은 5423.19 ms이다. n개의 서로 다른 전이흐름을 통해 도달 가능한 상태의 타임 스탬프 값은 각 전이흐름을 통한 도달시간의 평균값 n개 중 최소값을 채택한다. 상태 ③에 도달 가능한 세 가지 전이 흐름을 통한 평균 도달시간 중 최소값인 5423.19 ms는 예상행동모델에서의 상태 ③까지의 도달 시간인 2743 ms의 2배를 초과하는 값이므로 이상징후 4가 감지된 것으로 판단 가능하다. 이상징후 4의 원인은 GUI 설계가

사용자의 입장에서 이해하기 어렵게 구성되어 있기 때문일 수도 있으며 이상징후 3과 마찬가지로 실제 구현상의 오류로 인한 서비스 지연이 원인이 될 수 있다.

5. 실 험

우리는 본 논문에서 제안한 기법의 검증을 위해, 몇 가지의 오픈소스 모바일 앱을 대상으로 4가지 타입의 사용성을 저해하는 이상징후들을 자동 검출하는 실험을 진행하였다. 다음은 실험에 대한 설계와 실험 결과 분석 내용이다.

GUI 사용성 평가 시 적정 실험 대상자의 숫자가 5명이라는 [9]의 권고안에 따라, 성별, 연령대, 앱 사용 능숙도 별로 유사한 프로파일을 가지는 5명의 사용자들을 각각 3그룹씩 구성하여 RUID가 탑재된 모바일 기기에서 실험 대상 앱을 사용하도록 하였다. 실험 대상 앱으로는 오픈소스 저장소에 오픈된지 1년 이상이며 100번 이상의 커밋이 기록된 앱 중에서 GUI 관련된 오류 이슈가 가장 많이 제기된 5개의 앱(Table 1 참조)을 선정하였다. 3개의 서로 다른 프로파일을 공유하는 사용자 그룹이 174개의 사용 시나리오를 선정된 앱 5개를 수행하도록 하였다. 그 결과 추출된 사용자 행동 로그를 기반으로 RUID를 적용하여 검출된 GUI의 사용성 저해 요소가 개발자들에 의해 제기된 사용성 관련 이슈와 비교하여 어느 정도 일치하는 지를 분석하였다.

사용성 저해요소 검출의 정밀도(Precision) 산출 공식은 모바일 앱의 GUI 사용성 저해 요소 찾아내야 하는 도메인의 특성을 반영하여 아래 Equation (1)과 같이 정의하였다. 제안된 RUID의 오류검출 정확도를 측정하는 것이 본 실험의 목적이므로, 재현율(Recall)은 평가 대상에서 제외하였다.

$$\text{정밀도} = TP / (TP + FP) \quad (1)$$

- True Positive (TP): 도구가 이상징후로 검출한 것 중, 앱에 등록된 GUI이슈에도 검출된 것
- False Positive(FP): 도구는 이상징후로 검출했으나 앱에 등록된 GUI 이슈에 언급되지 않은 것.

Table 1. Precision of GUI bad symptom detection by RUID

App name	TP	FP	Precision
IMSI	9	3	0.75
News-android	18	8	0.69
MyRemocon	8	4	0.66
open-key	5	2	0.71
Shhapp	4	2	0.67
Average			0.69

Table 1의 검증 결과에서 알 수 있듯이 RUID의 GUI 이상징후 자동 검출에 대한 정밀도는 평균 0.69이다. 이는 개발자들이 반복적인 테스트를 통해 식별하여 검출한 GUI 관련 이슈의 69% 정도를 RUID가 자동으로 검출해 낼 수 있음을 의미한다. 결과적으로 RUID적용할 경우, 수작업으로 사용성 오류를 검증하는데 소요되는 노력의 69%가 감소됨을 말한다.

실험은 3개의 그룹으로 나누어 진행하였으나, Table 1에서의 실험결과값을 그룹별로 비교하지 않은 이유는 3개 이상의 집단 간 평균을 구하는 기법으로 주로 사용되는 ANOVA[10]를 적용한 결과, 각 그룹간의 평균값의 차이가 0.318로 그룹별 특이사항이 감지되지 않았기 때문이다. 유의 확률이 0.05보다 크기 때문에 통계적으로 차이를 인정할 수 없으며 그룹간 평균의 차이는 의미가 없다고 할 수 있다.

두 번째로 측정한 메트릭은 RUID 적용을 통해 기대할 수 있는 모바일 앱의 품질 향상 정도이다. Table 2는 자동 검출된 모바일 앱의 사용성 저해 요소를 반영하여 수정함으로써 사용자들이 모바일 앱을 사용하여 대상 시나리오를 수행하는데 성공하는 확률이 얼마나 향상되는지, 시나리오를 수행하는 데 소요되는 시간은 얼마나 단축이 되는 지를 보여준다. Table 2의 실험 결과에 따르면 이상징후를 잠재적으로 가지고 있던 모바일 앱 버전에서의 사용자들의 시나리오 수행 성공률이 평균 58%를 보였으나 RUID가 자동 검출한 이상징후의 문제점을 파악하여 해결한 후의 모바일 앱 버전에서는 시나리오 수행 성공률이 평균 77%로 19% 가량 향상된 결과를 보였다. 또한, 이상징후를 검출하여 해결하기 이전 버전의 시나리오 별 평균 수행 소요시간은 5169ms인데 반해 이상징후 해결 이후 버전의 모바일 앱을 사용하여 시나리오를 수행한 평균 소요시간은 4071.4ms로 1097.6ms가 감소하였다.

이 같은 사용자의 앱 효용성 향상분에는 동일한 시나리오를 재수행함으로써 얻을 수 있는 학습효과가 일정 부분 포함되어 있겠으나 RUID의 이상징후 검출이 모바일 앱의 효용성 향상에 기여하고 있음을 입증하는 변화로 해석 가능하다.

Table 2. Comparison of efficiency: App with bad symptoms vs. App without bad symptoms

(1) 이상징후 수정 전/후 시나리오 성공률 비교

Application #Scenario	수정 전 성공률	수정 후 성공률	성공률 변화
IMSI#59	61 %	84 %	+23%
News-android#23	68 %	74 %	+6%
MyRemocon#157	37 %	86 %	+47%
open-key#54	71 %	88 %	+5%
Shhapp#47	54 %	66 %	+12%
Average	**58 %**	**77 %**	**+19%**

(2) 이상징후 수정 전/후 소요시간 비교

Application #Scenario	수정 전 소요시간	수정 후 소요시간	소요시간 변화(ms)
IMSI#59	4975 ms	3157 ms	-1818
News-android#23	5941 ms	5324 ms	-617
MyRemocon#157	7378 ms	5349 ms	-2029
open-key#54	3117 ms	3259 ms	-142
Shhapp#47	4434 ms	3268 ms	-806
Average	**5169 ms**	**4071.4 ms**	**- 1097.6**

6. 결 론

모바일 앱이 웹 어플리케이션과 비교하여 다양한 디바이스에 다양한 사용자 행동을 통해 서비스를 제공할 수 있다는 장점을 가지고 있으나 사용자의 의도를 정확히 파악하여 설계되지 못한 GUI는 사용자로 하여금 원하는 서비스를 제공받는 데 있어서 불편함을 초래할 수 있다. 이러한 문제를 해결하기 위하여 실제로 서비스 기능 개발시간과 비교하여 비교적 많은 개발 인력과 시간이 GUI 사용성 테스트에 할애되고 있다. 그러나 GUI 사용성 테스트 결과에 기록된 사용성 관련 이슈들 중 상당 부분은 몇 가지의 정형화된 검출 기법을 통해 자동 검출이 가능하다.

본 연구는 이와 같이 자동검출이 가능한 모바일 앱의 사용성 저해 요소의 타입을 식별하고, 사용자 로그 분석을 통해 사용성 저해요소들을 자동으로 검출해 내는 기법을 제안하였다.

제안된 자동 검출 기법 적용 결과 검출된 사용성 저해 징후들과 실제 개발자가 직접 테스트를 수행하여 발견한 사용성 관련 이슈 중 평균 69%가 일치하는 결과를 보였다. GUI 테스트에 소모되는 시간 비율이 전체 개발에 소모되는 시간 중 50-60%[11]인 것을 감안하면 제안된 기법의 적용을 통해 전체 개발 시간 단축을 이룰 수 있을 것이라 기대할 수 있다.

그러나 현재까지는 실험대상이 주로 예매, 일정관리, 뉴스 앱과 같은 유틸리티 성격의 모바일 앱에 국한되어 있다는 한계점이 있다. 향후 연구를 통해 제시한 기법의 적용 도메인을 넓혀 나가면서 동시에, 모바일 앱의 도메인마다의 조작 특성이나 GUI 구성 특징들을 반영하여 도메인 별 사용성 저해 요소 타입을 추가적으로 정의해 나감으로써 RUID의 사용성 저해요소 자동 검출의 정밀도를 향상시켜 나갈 계획이다.

References

[1] Y. Kim, J. Byun, S. Choi, S. Park, and S. Park, “A Usability Analysis Method of Mobile Application using User Behavior Logs,” *Journal of KIISE: Software and Applications*, Vol.39, No.2, pp.91-98, 2012.

[2] E. Dolstra, R. Vliegendhart, and J. Pouwelse, “Crowdsourcing GUI Tests,” in *Software Testing, Verification and Validation, 2013 IEEE Sixth International Conference*, pp.332-341, 2013.

[3] R. Gómez, D. Caballero, and J. Sevillano, “Heuristic Evaluation on Mobile Interfaces: A New Checklist,” *The Scientific World Journal*, pp.178-188, 2014.

[4] M. Zen. “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*, pp.183-186, 2013.

[5] S. Abrahao and E. Insfran, “Early Usability Evaluation in Model Driven Architecture Environments,” *Quality Software, International Conference on, Quality Software, International Conference*, pp.287-294, 2006.

[6] W. A. Munson and M. Gardner, “Standardizing Auditory Tests,” *Acoustical Society of America*, Vol.22, No.5, p.675, 2005.

[7] K. Ma and S. Park, “A Methodology of Usability Issue Detection for GUI Self-Adaptation,” in *Proceedings of the Korea Conference on Software Engineering*, pp.411-412, 2015.

[8] L. Mariani, F. Pastore, and M. Pezze, “Dynamic Analysis for Diagnosing Integration Faults,” *Software Engineering, IEEE Transactions*, Vol.37, No.4, pp.486-508, 2011.

[9] J. Nielson, Why you only need to test with 5 users [Internet], http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[10] R. G. O’Brien, “A General ANOVA Method for Robust Tests of Additive Models for Variances,” *Journal of the American Statistical Association*, Vol.74, No.13, pp.877-880, 1979.

[11] A. M. Memon, “A comprehensive framework for testing graphical user interfaces,” Ph.D. dissertation, University of Pittsburgh, 2001.

마 경 욱

e-mail : makw@sogang.ac.kr
2014년 숭실대학교 컴퓨터공학부(학사)
2014년～현 재 서강대학교 컴퓨터공학과 석사과정
관심분야 : Software architecture & Testing

박 수 용

e-mail : sypark@sogang.ac.kr
1986년 서강대학교 컴퓨터공학과(학사)
1988년 Florida State University, Computer and Information Science(석사)
1995년 George Mason University, Information Technology(박사)
1998년～현 재 서강대학교 컴퓨터공학과 교수
관심분야 : 요구공학, 동적 아키텍처

박 수 진

e-mail : psjdream@sogang.ac.kr
1995년 경북대학교 컴퓨터공학과(학사)
2000년 서강대학교 정보통신대학원(석사)
2008년 서강대학교 컴퓨터공학과(박사)
1995년～1999년 (주)LG-CNS 근무
2000년～2002년 한국래쇼날 소프트웨어 선임 컨설턴트
2010년～현 재 서강대학교 서강미래기술연구원 교수
관심분야 : 요구공학, 소프트웨어 아키텍처, 자가 적응형 소프트웨어