논문 2014-09-01

안드로이드 어플리케이션의 결함 재생을 위한 액티버티 단위의 이벤트 캡춰 및 재생 기법

(Activity-based Event Capture and Replay Technique for Reproducing Crashes in Android Applications)

아자이쿠마르자, 이 우 진*
(Ajay Kumar Jha, Woo Jin Lee)

Abstract : As Android platform and its applications are widely used, it is necessary to assure the reliability and safety of Android applications. The capture and replay technique is widely used for reproducing the crashes in Java applications. In this paper, we adapt the context-based capture and replay technique to Android applications by considering Android activity structure and its GUI characteristics. Our capture and replay technique uses probe code insertion based on AspectJ for minimizing the original code change and for enhancing the efficiency and reusability of the probe codes. For proving the efficiency of our technique, we provide a case study and execution overhead comparisons with related works, in which our approach has merits in GUI oriented Android applications with relatively low execution overhead.

Keywords : Crash reproducing, Capture and replay, Android application, AspectJ

Ⅰ. 서론

최근 들어 스마트폰 보급이 급속히 증가함에 따라 안드로이드 기반의 어플리케이션의 사용이 급격하게 증가하고 있다. 또한 안드로이드 플랫폼은 스마트폰뿐만 아니라 자동차, 가전 등의 다양한 분야로 확대되고 있는 추세이다. 다양한 분야에 활용되는 안드로이드 어플리케이션에 내포되어 있는 오류들을 검출하여 어플리케이션의 안전성을 보장하는 것이 무엇보다 중요하다.

소프트웨어에 내포된 오류를 찾아내는 과정은 쉬운 일이 아니다. 소스코드 리뷰와 같이 정적으로 진행되거나 테스팅과 같이 실제 프로그램을 수행하면서 오류 존재 여부를 검사한다. 테스팅을 통해 프로그램이 제대로 동작하지 않는다는 사실을 알아내더라도 실제 어느 부분에 오류가 있는지를 알아내는 것은 쉽지 않다. 안드로이드 플랫폼이 사용되는 임베디드 소프트웨어의 경우, 프로그램 수행 중에는 다양한 비결정적 요소가 많아 크래시 상황의 재현이 쉽지 않다.

오류 발생 상황을 재현하는데 많이 쓰이는 기법으로는 컨텐트기반과 순서기반의 캡춰 및 재생(capture and replay) 기법이 연구되어 왔다. 컨텐트기반의 기법[1-5]은 발생하는 이벤트와 이와 연관된 데이터를 기록하고 기록된 정보를 바탕으로 재생하는 방법이다. 이러한 방법은 방대한 로그 정보를 저장하여야 하므로 시간적인 및 공간적인 오버헤드를 가진다. 최근 연구에서는 이러한 오버헤드를 최소화하는 방법들이 제안되고 있다. 순서기반의 방법[6-9]은 이벤트의 수행 순서를 저장하며 이를 기반으로 재생하는 기법이다. 순서기반 방법은 상대

*Corresponding Author (woojin@knu.ac.kr)
Received: 9 Aug. 2013, Revised: 13 Sep. 2013, 8 Oct. 2013, Accepted: 13 Oct. 2013.
A.K. Jha, W.J. Lee: Kyungpook National University.
※ 본 연구는 미래창조과학부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(SW.컴퓨팅)의 일환으로 수행되었으며[10041145, 자율군집을 지원하는 웰빙형 정보기기 내장 소프트웨어 플랫폼 개발] 또한 정보통신산업진흥원의 IT융합 고급인력과정 지원사업의 연구결과로 수행되었음 (NIPA-2013-H0401-13-1005).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2014.9.1.1

적으로 적은 양의 로그를 기록하지만 수행환경이나 입력 조건의 사소한 변화가 재생단계의 실제 수행 경로를 변경시킬 수 있다. 기존의 제시된 캡춰 및 재생 기법들은 자바 환경에서 동작하며 GUI 및 액티버티 중심으로 동작하는 안드로이드 특성을 제대로 반영하고 있는 방법은 없다.

이 연구에서는 안드로이드 어플리케이션의 크래시 상황을 효율적으로 재현하기 위해 안드로이드의 수행단위인 액티버티를 기반으로 컨텐트기반의 캡춰 및 재생 기법을 제안한다. 기존 연구들은 소스코드를 변경하여 이벤트를 기록하는 방법을 사용하였으나 이 연구에서는 관점지향 프로그래밍 기법을 지원하는 AspectJ를 활용하여 기존 소스 코드를 수정하지 않아도 된다. 안드로이드 어플리케이션은 GUI 중심으로 동작하므로 일반적인 이벤트 기록뿐만 아니라 GUI 이벤트들의 기록을 중점적으로 다룬다. 일반적으로 컨텐트 기반 기법은 로그의 정보 양이 방대하므로 체크포인트 탐지기를 사용하여 크래시와 연관된 액티버티들의 이벤트만을 기록한다. 그리고 제안된 방법의 효율성을 확인하기 위해 사례연구를 수행하여 기존 관련연구와의 수행시간 오버헤드를 비교한다.

논문의 구성은 다음과 같다. 먼저 제 2장에서 안드로이드 플랫폼의 특성에 대해서 살펴보고 관점지향 프로그래밍 기법인 AspectJ에 대해 살펴보고 관련연구들을 분석한다. 제 3장에서는 안드로이드 플랫폼에 특화된 이벤트 캡춰 및 재생 기법을 상세히 설명한다. 제 4장에서는 적용사례 분석 및 관련연구와의 비교 분석을 수행하고 마지막으로 제 5장에서는 결론 및 향후연구를 기술한다.

II. 연구배경

2.1 안드로이드 플랫폼

안드로이드 플랫폼[10]은 모바일 디바이스를 위해 고안된 리눅스 기반 운영체제이다. 안드로이드 어플리케이션은 액티버티(Activity), 서비스(Service), 컨텐트 제공자(Content Provider), 브로드케스트 수신기(Broadcast Receiver)의 4개의 컴포넌트로 구성되어 있다.

- 액티버티: 안드로이드 어플리케이션을 구성하는 가장 기본적인 빌딩블록으로 한 화면을 차지하는 유저 인터페이스를 화면에 표시하고 사용자의 입력을 처리한다. 하나의 어플리케이션은 각자의 화면을 표시하는 여러 액티버티로 구성이 되어 있다.
- 서비스: 오랫동안 수행되는 태스크나 원격 프로세스 업무 등을 백그라운드로 수행할 때 주로 사용되며 사용자 인터페이스는 없다. 서비스가 시작되면 호출자에게 결과값을 주지 않고 하나의 업무를 무한으로 수행할 수도 있다. 그러나 바운드된 서비스는 호출자에게 서비스와 연동되는 클라이언트 서버 인터페이스를 제공한다.
- 컨텐트 제공자: 공유되는 어플케이션 데이터 집합을 관리한다. 어플리케이션은 이를 통해 파일, SQLite 데이터베이스, 웹 또는 다른 스토리지 등의 데이터를 접근할 수 있다.
- 브로드케스트 수신기: 스크린 꺼짐, 밧데리 낮음, 화면 캡춰 등의 시스템에서 생성되는 브로드케스트 이벤트에 반응한다. 브로드케스트 이벤트는 시스템뿐만 아니라 어플리케이션에서도 생성될 수 있다.

2.2 AspectJ 프로그래밍 환경

AspectJ [11, 12]는 자바의 관점지향 프로그래밍(aspect-oriented programming) 기법을 지원한다. 관점지향 프로그래밍은 개발자가 여러 컴포넌트에 걸쳐 있는 횡단 관심사(cross-cutting concerns)을 독립적인 모듈로 개발하여 나중에 관련 컴포넌트에 끼워넣는 직조(weaving)과정을 거친다. 예를 들어, 로그, 보안 등과 같이 공통으로 사용되는 횡단 기능들을 독립 모듈인 횡단 관심사로 개발한 후에 연관된 부분에 자동으로 끼워 넣게 된다. 관점지향 프로그래밍의 주요구문에는 관심사가 주 프로그램의 어디에 횡단될 것인지를 나타내는 조인 포인트(join point), 언제 조인 포인트가 매치될 것인지를 알려주는 포인트컷(pointcut), 포인트컷이 활성화되면 수행되는 관심사 코드인 어드바이스(advice) 등이 있다.

2.3 관련연구

자바 언어의 컨텐트 기반 캡춰 및 재생기법으로 jRapture[1], ReCrash[2], SCARPE[3] 방법 등이 있다. jRapture 방법에서는 Java API 클래스를 변형하여 자바 어플리케이션과 하위 시스템간의 상호작용을 잡아낸다. 재생 단계에서는 캡춰 과정에서 저장된 입력 시퀀스와 동일하게 각각의 쓰레드를 수행한다. 이러한 방법에서는 메소드에서 접근 가능한 전역변수들을 캡춰하여야 하는데 이러한 전역변수 캡춰가 쉽지 않다[13]. 또한 직렬화 기법

(serialization)을 이용하여 객체 정보를 캡춰하는데, 이를 위해서는 직렬화 인터페이스를 구현하도록 모든 클래스를 변경하여야 한다는 단점이 있다.

ReCrash 방법[2]은 프로그램 수행시에 각 메소드의 실인자와 객체 인스턴스의 복사본과 함께 호출 스택의 복사본을 메모리상에 관리한다. 객체 인스턴스의 복사본은 힙 공간에 있는 원 객체를 참조한다. 프로그램의 오류가 발생하면, ReCrash는 호출 스택 복사본의 내용과 스택에서 참조되는 모든 힙 객체들을 직렬화한다. ReCrash 기법은 수행된 이벤트를 그대로 재생하지 않고, 호출 스택 복사본에 있는 역직렬화된 객체들의 각 메소드를 호출하여 후보 테스트군을 생성한다. 이러한 후보 테스트군을 수행하여 크래시 상황을 재현한다.

SCARPE 방법[3]은 사용자가 제공한 리스트를 바탕으로 관찰 클래스 집합을 구성하고 어플리케이션을 조금 변경하여 관찰 클래스와 나머지 시스템 간의 상호작용을 알아낸다. SCARPE 방법에서는 캡춰 단계에서 객체의 인스턴스를 파악하기 위해 객체 ID를 사용하여 객체 직렬화 문제가 발생하지 않는다. 재생 단계에서는 객체 ID를 이용하여 객체를 생성 및 검색한다.

이러한 jRapture, ReCrash, SCARPE 방식은 모두 자바 언어에 기반을 두고 있으며 안드로이드 어플리케이션에 적용하기 위해서는 안드로이드 플랫폼 특성에 맞게 수정하는 작업이 필요하다.

Ⅲ. 이벤트 캡쳐 및 재생 모델

안드로이드 어플리케이션의 이벤트를 캡춰하기 위해 캡춰 모듈을 그림 1과 같이 어플리케이션 프레임워크 위에 배치한다. 하지만 안드로이드 아키텍처의 어플리케이션 프레임워크나 다른 계층의 API를 이용하지 않으며 단지 어플리케이션 컴포넌트와 연동되어 동작한다. 안드로이드 어플리케이션은 액티버티 단위로 수행되며 캡춰 모듈도 주로 액티버티 상에서 동작한다.

3.1. 프로브 코드 삽입 단계

이 절에서는 자바의 관점지향 프로그래밍(Aspect Oriented Programming; AOP) 기법을 지원하는 AspectJ를 이용하여 프로브(probe) 코드를 삽입하여 안드로이드의 이벤트를 캡춰하는 기법을 다룬다.

기존의 캡춰 및 재생 기법들은 어플리케이션의

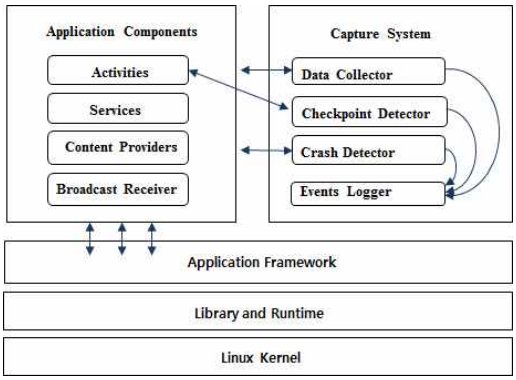

그림 1. 안드로이드 캡춰 시스템의 구조
Fig. 1 Structure of Android Capture System

소스코드에 직접 프로브 코드를 넣는 방법[2, 3, 7], API 수정 [1], 가상 머신 수정[14, 15], 운영체제 수정[16, 17] 등의 다양한 기법을 사용하고 있다. AOP 기반의 프로브 코드 삽입 기법은 어플리케이션 코드와 프로브 코드를 분리하여 관리함으로 기존 어플리케이션 코드를 수정할 필요가 없으며 프로브 코드를 재사용하여 사용할 수 있는 장점이 있다.

조인 포인트는 메소드 호출과 같이 개발자가 어플리케이션의 중간에 끼어들 수 있는 특정 포인트들을 나타낸다. 어드바이스에는 메소드 호출정보 저장과 같이 여러 컴포넌트에 공통으로 들어가는 관점들을 구현한 것이며 포인트컷은 조인 포인트 중에서 이러한 어드바이스가 호출되는 포인트를 정의한다. 또한 포인트컷을 통해 어플리케이션의 수행 정보를 넘겨받아 어드바이스의 바디 구현에 활용한다. 이 연구에서의 AspectJ를 이용한 안드로이드의 이벤트 캡춰 방법은 포인트컷에서 노출되는 이벤트 정보들을 어드바이스를 통해 뽑아내는 방식이다. 또한 AspectJ의 어드바이스는 어플리케이션과 무관하게 사용할 수 있으므로 다른 어플리케이션 적용 시에 재사용할 수 있다.

3.2 이벤트 캡춰 단계

이벤트 캡춰는 프로브 코드가 삽입된 어플리케이션의 수행과 함께 시작된다. 특정 이벤트가 발행하면 이벤트 정보와 관련된 데이터도 함께 저장한다. 또한 예외상황이 처리되지 않으면 예외상황 정보와 관련 데이터를 저장한다. 이 절에서는 AspectJ를 이용하여 이벤트 캡춰를 다루는 데이터 수집기 (Data Collector), 체크포인트 탐지기

```
pointcut methodExec():
   execution (* kidsmusicland..*(..))
   && !within(kidsmusicland.Trace);

pointcut methodCall():
   call (* kidsmusicland..*(..)) && !methodExec()
   && !within(kidsmusicland.Trace);

pointcut dataAccessedInt():
   get(int *) && !within(kidsmusicland.Trace);

pointcut dataModifiedInt(int newValue):
   set(int *) && args(newValue)
   && !within(kidsmusicland.Trace);

pointcut dataReturnedInt():
   call(* *.*(..)) && !methodCall() && !methodExec()
   && !within(kidsmusicland.Trace);
```

그림 2. 이벤트 캡춰를 위한 AspectJ 포인트컷
Fig. 2 AspectJ Pointcut for Event Capturing

(Checkpoint Detector), 크래시 탐지기 (Crash Detector) 컴포넌트의 구현에 대해 다룬다.

3.2.1 데이터 수집기

일반적으로 시스템의 행위는 상태와 이벤트에 의한 상태전이의 연속이다. 시스템의 크래시 상황을 탐지하기 위해서는 정상적인 상황뿐만 아니라 비정상적인 상황에서의 이벤트들을 수집하여야 한다. 메소드 호출은 시스템의 상태를 변화시키는 대표적인 경우로 파라미터의 값을 변경하거나, 사용된 애트리뷰트의 값을 변경하거나, 새로운 값을 리턴한다. GUI 어플리케이션의 경우 리소스 ID도 저장한다. 그림 2는 이러한 메소드 호출 관련 이벤트들을 캡춰하기 위한 AspectJ 포인트컷을 보여준다. 그림 2의 methodExec()와 methodCall() 포인트컷은 메소드 수행 및 메소드 호출 이벤트를 캡춰한다. 메소드 호출과 수행은 조금 차이가 있다[11, 12]. 메소드 호출은 메소드가 불려지는 시점을 나타내며 상황정보는 호출 클래스이며, 메소드 수행은 메소드 바디 수행 시점을 나타내며 상황정보는 수행중인 메소드 자신이다. 그림 3은 methodCall과 methodExec의 어드바이스 코드를 보여준다. 수집되는 정보로는 메소드 시그너처와 메소드의 파라미터 리스트 등이 있다. 그림 4는 메소드 데이터를 기록하는 dataAccessedInt(), dataModifiedInt(int), dataReturnedInt() 어드바이스를 보여준다. 기본 자료형의 정보는 쉽게 기록할 수 있지만 객체인 경우는 직렬화가 필요하며 이를 위해서는 소스 변경을 하여야 한다. 이 연구에서는 소스 코드를 변경하지 않으며 캡춰 오버헤드를 최소화하기 위해 객체에 포함된 기본 자료형만 기록한다. 안드로이드 어플리케이션에

```
before(): methodExec() {
  String mStart = "Method Execution => "
    + thisJoinPointStaticPart.getSignature().toString();
  tracelog.add(mStart);
  Object arguments[] = thisJoinPoint.getArgs();

  for(int i=0; i<arguments.length; i++) {
     Object argument = arguments[i];
     if(argument != null)
        if(argument instanceof View) {
           String res = "Resource->"
              + ((View) argument).getId());
           tracelog.add(res);
        } else {
           String args = "Args: "+ argument;
           tracelog.add(args);
        }
  }
}

before(): methodCall() {
  String imStart = "Method Call => "
    + thisJoinPointStaticPart.getSignature().toString();
  tracelog.add(imStart);
  Object arguments[] = thisJoinPoint.getArgs();

  for(int i=0; i<arguments.length; i++) {
     Object argument = arguments[i];
     if(argument != null) {
        String args = "Args: "+ argument;
        tracelog.add(args);
     }
  }
}
```

그림 3. AspectJ 메소드 관련 어드바이스
Fig. 3 AspectJ Advice for handling Methods

```
//capture value of the field (int type) being accessed
after() returning(int value): dataAccessedInt() {
   String gInt = thisJoinPointStaticPart.getSignature().
     toString()+ " = "+ value;
   tracelog.add(gInt);
}

//capture value of the modified field (int type)
before(int newValue): dataModifiedInt(newValue) {
   String sInt = thisJoinPointStaticPart.getSignature().
     toString()+ " = "+ newValue;
   tracelog.add(sInt);
}

//capture value of the returned value (int type)
after() returning(int retvalue): dataReturnedInt() {
   String mCall = thisJoinPointStaticPart.getSignature().
     toString()+ "->"+ retvalue;
   tracelog.add(mCall);
}
```

그림 4. 메소드의 데이터 관련 어드바이스
Fig. 4 Advice for handling Method Data

서는 GUI 컴포넌트도 중요한 구성요소이다. 안드로이드에서는 액티버티별로 GUI를 생성하는데, 각 GUI는 View와 ViewGroup으로 구성된다. GUI 이벤트가 생성되면 해당 행위를 처리하기 위해 콜백(callback) 함수가 수행된다. GUI 행위를 제대로 재현하기 위해서는 이벤트와 연관된 View 객체의 ID와 콜백 함수정보를 저장한다.

```
pointcut clearLog():
   call (* kidsmusicland.*.startActivity(..))
   && !within(kidsmusicland.Trace);

List<String> tracelog = new ArrayList<String>();

before(): clearLog() {
   tracelog.clear();
}
```

그림 5. 체크포인트의 포인트컷과 어드바이스
Fig. 5 Pointcuts and Advices for Checkpoint

3.2.2 체크포인트 탐지기

어플리케이션이 수행되는 동안 발생하는 모든 이벤트 정보들을 저장하는 것은 너무나 방대하기 때문에, 액티버티별로 체크포인트를 두어 크래시가 발생하였을 때 연관된 이벤트들만을 저장하는 것이 효과적이다. 체크포인트 탐지기는 크래시와 연관된 이벤트들을 저장하는 시작점을 결정한다. 일반적으로 안드로이드 액티버티는 리턴값을 넘겨주는 경우와 넘겨주지 않는 경우가 있다. 리턴값을 넘겨주는 경우는 호출 액티버티와 밀접하게 연동되어 동작하므로 호출 액티버티와 동일한 수행맥락으로 다룬다. 하지만 리턴값을 넘겨주지 않는 액티버티는 새로운 수행맥락이 시작되는 것으로 간주하여 체크포인트로 설정한다. 안드로이드 액티버티의 시작을 나타내는 startActiviy() 메소드가 불려지면 그림 5의 clearLog() 포인트컷과 어드바이스를 이용하여 기존 tracelog 정보를 삭제하고 새로운 수행맥락을 시작한다. 안드로이드 액티버티들이 순차적으로 불려지는 경우도 있지만 두 액티버티가 공존하는 경우도 있다. 예를들어, 한 액티버티가 일시정지(paused) 상태에서 다른 액티버티가 수행된 후에 처음 액티버티가 재시작되는 경우가 발생하는데, 이러한 경우 액티버티별로 저장 단위를 나뉘면 재생하는데 어려움이 있을 수 있다. 액티버티가 일시정지될 때 onPause()가 불려지므로 onPause() 이벤트 발생 이후 onDestory() 종료 이벤트가 발생되지 않고 다른 액티버티의 startActivity()가 불려지면 체크포인트를 새롭게 설정하지 않고 동일한 수행맥락에 계속 저장한다.

3.2.3 크래시 탐지기

크래시 탐지기는 실제 크래시가 발생되는 시점을 감지하는 것이다. 안드로이드에서 여러 예외상황이 발생하는데, 프로그램에서 예외상황이 처리되면 정상적으로 동작하지만 예외상황이 처리되지 않으면 크래시를 일으키며 프로그램이 종료한다. 크래시 탐지기는 처리되지 않는 예외상황을 감지하여 크래시

```
pointcut myException(Exception e): handler(Exception)
    && args(e) && !within(kidsmusicland.Trace);

pointcut unCaught():
    execution(* kidsmusicland..*(..))
    && !within(kidsmusicland.Trace);

before(Exception e): myException(e) {
    String exc = "Exception: "
      + thisJoinPointStaticPart.getSignature().toString();
    tracelog.add(exc);
}

after() throwing(Exception e): unCaught() {
    String unCut = "Uncaught Exception " +
    thisJoinPointStaticPart.getSignature().toString()+
    ":"+ e;
    tracelog.add(unCut);

    try {
        buf = new BufferedWriter(
                new FileWriter(fout, true));
        int len = tracelog.size();
        for(int i=0; i<len; i++) {
            buf.write(tracelog.get(i));
            buf.newLine();
        }
        buf.flush();
        buf.close();
    } catch(IOException ex) {
        ex.printStackTrace();
    }
}
```

그림 6. 예외상황 관련 포인트컷과 어드바이스
Fig. 6 Pointcuts and Advices for Exception

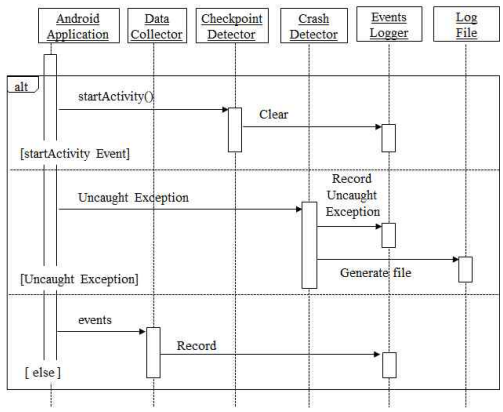

그림 7. 이벤트 캡춰 시나리오
Fig. 7 Event Capturing Scenarios

를 일으키는 이벤트 리스트를 로그 파일에 저장한다. 그림 6은 예외상황과 연관된 포인트컷과 어드바이스를 보여준다. 그림 7은 앞서 설명한 이벤트 처리 상황들을 정리하여 보여준다. 정상적인 이벤트와 예외상황은 이벤트 로그로 기록되며 만약 체크포인트 startActivity()가 수행되면 기록된 이벤트 로그를 삭제하고 새롭게 로그를 시작한다. 처리되지 않는 예외

```
Method Call =>
void kidsmusicland.ChildMusic.startActivity(Intent)
Args: Intent { cmp= kidsmusicland/.SelectSong (has
extras) }
Method Execution => void
kidsmusicland.SelectSong.onCreate(Bundle)
Method Call =>
boolean
kidsmusicland.SelectSongrequestWindowFeature(int)
Args: 7
Method Call =>
void kidsmusicland.SelectSong.setContentView(int)
Args: 2130903043
Method Call =>
Window kidsmusicland.SelectSong.getWindow()
void android.view.Window.setFeatureInt(int, int)
Method Call =>
Intent kidsmusicland.SelectSong.getIntent()
Bundle android.content.Intent.getExtras()
int android.os.Bundle.getInt(String)
int android.os.Bundle.getInt(String)->5
int kidsmusicland.SelectSong.index = 5
Method Call =>
View kidsmusicland.SelectSong.findViewById(int)
Args: 2131230741
int kidsmusicland.SelectSong.index = 5
Uncaught Exception void kidsmusicland.SelectSong.
onCreate(Bundle):java.lang.ArrayIndexOutOfBoundsEx
ception: length=5; index=5
```

그림 8. 로그 파일 예제
Fig. 8 Log File Sample

상황이 발생하면 예외상황 정보를 저장하고 로그파일을 생성한다.

3.3 크래시 재생 단계

로그파일에 저장된 이벤트를 바탕으로 크래시를 재생하기 위해서는 어플리케이션 소스 코드에 재생을 위한 프로브 코드를 추가하는 것이 필요하다. 로그파일의 첫번째 값은 액티버티와 연관된 정보이다. 예를 들어, 그림 8과 같이 void kidsmusicland.ChildMusic.startActivity (Intent)가 저장되어 있는데, 이는 kidsmusicland.ChildMusic 액티버티가 수행되었다는 것을 나타낸다.

두 번째 값은 startActivity()의 실인자를 나타내며 “Intent { flg = 0x2400000 cmp = SomeActivity / .AnotherActivity }”와 같은 형식이다. 이를 통해 AnotherActivity로 제어가 흘러감을 알 수 있다. 그림 8의 마지막 로그 부분에서 볼 수 있듯이 크래시 상황은 처리되지 않은 예외상황으로 나타난다. 재생 단계도 어플리케이션의 출력이 생성되는데, 재생단계의 출력과 로그에 기록된 출력을 비교하여 재생이 제대로 이루어지는지 판단할 수 있다. GUI 관련 이벤트도 유사한 방법으로 재생되며 GUI 이벤트의 경우 추가적으로 기록된 리소스 ID를 활용한다.

그림 9. KidsMusicLand 앱의 메인 화면
Fig. 9 Main Layout of KidsMusicLand App.

IV. 적용사례 및 비교분석

이 장에서는 논문에서 제시한 액티버티 기반 캡춰 및 재생 기법의 효용성을 확인하기 위해 적용사례와 관련연구와의 수행시간 오버헤드를 비교한다. 예제는 그림 9와 같은 메인화면을 가진, 3,457라인, 18개 액티버티, 21개 클래스로 구성된 비공개 테스트용 안드로이드 앱인 KidsMusicLand를 사용하였다. 사용된 컴퓨터와 적용환경은 인텔 코어 i3 3.10 GHz, 4 GB 램, Windows 7, Eclipse Indigo, Android 4.0.3, JDK 1.5과 AspectJ 1.6.12를 사용하였다.

4.1 관련 연구와의 기능별 비교

관련 연구들을 분석하여 크래시 재생에 필요한 이벤트 정보들을 분석하여 표 1과 같은 비교표를 얻었다. 표에서 √ 표시는 해당 항목이 포함됨을 나타내고 × 표시는 해당 항목이 포함되지 않음을 나타낸다. 표에서 알 수 있듯이, 제안하는 방법은 기존 관련 연구에서 사용하는 모든 캡춰 정보들을 포함하고 있으며 추가적으로 GUI 관련 이벤트를 포함하고 있다. SCARPE와 ReCrash 기법은 GUI 이벤트를 다루지 못하므로 GUI 중심의 안드로이드 어플리케이션에 적합하지 않으며 jRapture와 ReCrash 기법은 객체 직렬화를 통해 정보를 획득함으로 모든 클래스가 Serializable 인터페이스를 구현하여야 한다는 제약이 있다. 이 논문에서 제시한 방법은 안드로이드에 특화된 방법으로 GUI 이벤트를 포함한 전반적인 시스템 상황들을 캡춰할 수 있는 기능을 제공한다.

시스템의 크래시를 제대로 탐지하고 재생하는지

표 1. 관련연구와의 기능 비교
Table 1. Comparison with Related Works

	jRapture	SCARPE	ReCrash	Our
적용언어	Java	Java	Java	Android
메소드	√	√	√	√
파라미터	√	√	√	√
리턴값	√	√	√	√
전역사용	√	√	√	√
예외	√	√	√	√
객체	√	ID,Type	√	Type
GUI이벤트	√	×	×	√

확인하기 위해 예제의 TransportSound 액티버티에 IndexOutOfBoundsException, AnimalSound 액티버티에 IllegalArgumentException, PlaySong 액티버티에 NullPointerException 오류를 삽입하였다. 삽입된 모든 오류는 각각 2.45 KB, 6.41 KB, 2.24 KB로 생성하면서 정확하게 탐지되고 재생되었다.

4.2 수행시간 오버헤드 비교

이벤트 캡춰를 위해 추가된 프로브 코드의 효율성을 확인하기 위해 액티버티별 수행시간을 프로브 삽입 이전과 이후를 비교하였다. 안드로이드 어플리케이션은 GUI를 통한 사용자 입력이 많아 수행시간을 측정하기에 어려움이 있어, 사용자 입력에 의존하지 않고 Dalvik Debug Monitor Server (DDMS)를 이용하여 미리 설정된 입력값을 사용하였다. 표 2는 앞서 예제에 삽입한 3가지 오류의 탐지 및 재생 시나리오에서 수행된 8개의 액티버티에 대해 프로브 이전 평균 수행시간, 프로브 이후 최대 수행시간과 최대 오버헤드를 정리한 것이다. KidsMusicLand와 MakeSong의 최대 오버헤드가 56.8%와 44.5%로 조금 높은데 반해 나머지 액티버티는 전반적으로 높지 않다. 일반적으로 최대 오버헤드는 최악의 경우이며 평균값은 이보다 훨씬 낮다.

표 3에서는 자바 기반의 캡춰 및 재생 기법과 이 논문에서 제시한 방법에 대해 수행시간 평균 오버헤드를 비교하고 있다. 그림에 명시된 jRapture, SCARPE, ReCrash 방법의 평균 오버헤드 값은 각각의 논문에서 제공한 오버헤드의 구간 값을 그대로 나타낸 것이며[1-3], 본 연구의 오버헤드 값은 다양한 수행 시나리오를 적용하여 액티버티별로 오버헤드 값을 평균하여 4~11%의 구간 값을 얻었다. 표 2에서는 수행시간 오버헤드의 최대값을 기술하였으며 표 3에서는 수행시간 오버헤드의 액티버티

표 2. 액티버티별 수행시간 오버헤드
Table 2. Execution Overhead of Activities

Activity Name	Original Avg(ms)	Probed Max(ms)	Execution Overhead
KidsMusicLand	3,953	6,200	56.8 %
PlaySong	3,533	4,080	15.5 %
HelpMakeSong	2973	3,160	6.3 %
TransportSound	2080	2,240	7.7 %
AnimalSound	1,830	1,940	6.0 %
MakeSong	4,567	6,600	44.5 %
BirdSound	1,966	2,200	11.9 %
ChildMusic	2,440	2,640	8.2 %

표 3. 수행시간 오버헤드의 구간 값의 비교
Table 3. Execution Overhead Range Comparison

Methods	Range of Activities' Execution Overhead Averages
jRapture	0.2 ~ 1.0 %
SCARPE	30.0 ~ 50.0 %
ReCrash	13.0 ~ 60.0 %
Ours	4.0 ~11.0 %

별 평균을 구간 값으로 나타내었다. 이러한 구간 값은 SCARPE와 ReCrash 방법보다는 현저히 적으며 jRature보다는 상대적으로 큰 것을 알 수 있다. 하지만 jRapture의 경우는 자바의 API를 수정하여 적용한 것이라 수행시간 평균 오버헤드는 작지만 근본적으로 모든 클래스를 직렬화하여야 하는 소스코드 수정 단점이 있다.

V. 결론

이 논문에서는 안드로이드 어플리케이션에 발생하는 크래시 상황을 재현한 캡춰 및 재생 기법을 소개하였다. 안드로이드의 기본 구조인 액티버티를 기반으로 이벤트들을 기록하는 방식이며 안드로이드의 GUI 특성을 반영하여 GUI 이벤트들을 캡춰할 수 있다. 제안한 방법의 효율성을 보이기 위해 기존 연구와의 기능 비교 및 평균 수행시간 오버헤드 등을 비교하였다.

프로브 코드의 수행시간 오버헤드는 기능적인 관점에서는 영향을 미치지 않으나 병행적 특성은 영향을 받는다. 향후 연구로는 이러한 프로브 소스코드의 효과를 최소화하기 위해 수행시간 오버헤드를 최소화할 필요가 있으며 여러 쓰레드가 수행되는 병행적 상황에서의 크래시 상황의 재현에 대한 연구가 필요하다.

References

[1] J. Steven, P. Chandra, B. Fleck, A. Podgurski, “jRapture: A Capture/Replay Tool for Observation-Based Testing,” Proceedings of the International Symposium on Software Testing and Analysis, pp.158 - 167, 2000.

[2] S. Artzi, S. Kim, M.D. Ernst, “ReCrash: Making Software Failures Reproducible by Preserving Object States,” Proceedings of the European conference on Object-Oriented Programming, pp.542-565, 2008.

[3] S. Joshi, A. Orso, “SCARPE: A Technique and Tool for Selective Capture and Replay of Program Executions,” Proceedings of the ICSM, pp.234-243, 2007.

[4] J. Clause, A. Orso, “A Technique for Enabling and Supporting Debugging of Field Failures,” Proceedings of the International Conference on Software Engineering, pp.261-270, 2007.

[5] G. Altekar, I. Stoica. “ODR: Output-Deterministic Replay for Multicore Debugging.” Proceedings of the Symposium on Operating Systems Principles, pp.193-206, 2009.

[6] M. Roneee, K. Bosschere, “RecPlay: A Fully Integrated Practical Record/Replay System,” ACM Transactions on Computer Systems, Vol. 17, No. 2, pp.133-152, 1999.

[7] T.J. LeBlanc, J.M. Mellor-Crummey, “Debugging parallel programs with Instant Replay,” IEEE Transactions on Computers, Vol. C-36, No. 4, pp.471 - 482, 1987.

[8] A. Georges, M. Christiaens, M. Ronsse, K. Bosschere. “JaRec: A portable record/replay environment for multi-threaded Java applications,” Software: Practice and Experience, Vol. 34, No. 6, pp.523-547, 2004.

[9] J. Huang, P. Liu, C. Zhang, S. Kim. “CLAP: Concurrent Lightweight Crash Reproduction,” Technical Report. The HongKong University of Science and Technology, 2009.

[10] Android Developers. http://developer.android.com.

[11] Introduction to AspectJ. http://www.eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html.

[12] R. Miles, AspectJ Cookbook, O'Reilly Media, 2004.

[13] F. Cornelis, A. Georges, M. Christiaens, M. Ronsse, T. Ghesquiere, K. Bosschere. “A Taxonomy of Execution Replay Systems,” Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet, 2003.

[14] R. Konuru, H. Srinivasan, J.D. Choi, “Deterministic replay of distributed Java applications,” Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, pp.219-228, 2000.

[15] J.D. Choi, B. Alpern, T. Ngo, M. Sridharan, J. Vissides, “A perturbation free replay platform for cross-optimized multithreaded applications,” Proceedings of the International Symposium on Parallel and Distributed Processing, 2001.

[16] S. Narayanasamy, G. Pokam, B. Calder, “BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging,” Proceedings of the Annual International Symposium on Computer Architecture, pp.284-295, 2005.

[17] S.M. Srinivasan, S. Kandula, C.R. Andrews, Y. Zhou. “Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging,” Proceedings of the Annual conference on USENIX, pp.29-34, 2004.

저 자 소 개

아제이쿠마르자

2004년 Sikkim Manipal University 학사.
2013년 경북대학교 컴퓨터학부 석사.
현재, 경북대학교 IT대학 컴퓨터학부 박사과정

관심분야: Software Verification and Validation, Dynamic Software Testing 등.
Email:ajaykjha123@yahoo.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