Journal of The Korea Society of Computer and Information
Vol. 21 No. 3, pp. 39-45, March 2016
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.3.039

A Log Management Service Model based on AOP for Efficient Development of Android Applications

Yun-seok Choi*

Abstract

In this paper, we propose a log management service model for efficient developments of android applications. The proposed model consists of two major parts which are the log collector and the log manager service. The log collector can capture the log information of a target application without modifications, because the collector is composed by aspect-oriented programming. The collected logs are transformed to chunk of data, and the chunk of data is sent to the log management service. The log management service is an android service component and an independent application in another process. So, the log management service can reduce the workload of logging in the target application. Through a case study, we show that the proposed log management service model can reduce the log processing time compared to other models without modifications of a target application.

▸ Keyword : Android, Logging, Aspect-Oriented Programming, Android Service.

I. Introduction

모바일 플랫폼과 하드웨어의 급속한 발전으로 인해 다양한 모바일 어플리케이션이 등장하고 있다. 최근의 어플리케이션은 과거에 비해 진보한 사용자 인터페이스 구성요소와 함께 네트워크, NFC, 카메라, 그리고 GPS 등을 조합하여 멀티스레드 기반의 새로운 서비스를 제공하고 있다. 이와 같이 과거에 비해 규모와 복잡도가 증가한 모바일 어플리케이션 개발을 위해서는 효과적인 디버깅 도구가 필요하며, 로깅은 가장 기본적이면서도 효과적인 수단으로 사용되고 있다.

안드로이드의 경우 플랫폼 자체에서 로그를 제공하고 있으며, 개발자는 로그캣(Logcat) 도구를 사용하여 제공받은 로그를 확인할 수 있다[1]. 시스템에서 제공하는 로그 이외에 개발자가 어플리케이션의 흐름 또는 상태를 확인하기 위해 로그를 필요부분에 삽입하여 개발에 활용할 수 있다. 로그는 간단하면서도 효율적으로 디버깅 관련 정보를 제공할 수 있으나 로그 코드가 어플리케이션에 산재할 경우 어플리케이션의 응집도를 감소시킬 수 있다. 또한 로그 코드 자체의 변경이 필요할 경우 개발자가 로그 코드 삽입 지점을 일일이 관리하여야 하는 어려움이 있다. 관점지향 프로그래밍은 로그와 같이 시스템에 산재하는 기능을 횡단관심사로 식별하고 이를 애스펙트(aspect)라는 모듈로 구현하여 관심사의 분리를 효과적으로 적용할 수 있도록 도와준다[2]. 로깅을 애스펙트로 구현할 경우 로깅 대상의 변경 없이 필요한 정보를 얻을 수 있으며, 로그 모듈의 변경 또는 확장을 손쉽게 수행할 수 있다[3]. 그러나 실사용 환경에서의 시험 등으로 인해 로그를 기기에 저장하거나 원격으로 전송할 필요가 있을 경우에는 관점지향 프로그래밍을 적용한 로깅도 대상 어플리케이션의 변경을 야기하며, 수행시간에도 영향을 줄 수 있다. 이는 애스펙트로 구성한 로깅도 최종적으로는 어플리케이션의 일부분으로 변환되기 때문이다. 이에 본 논문에서는 대상 어플리케이션의 변경을 최소화하면서 로깅 수행의 영향을 최소화할 수 있는 로그 관리 서비스 모델을 제안한다. 제안한 모델은 관점지향 프로그래밍을 적용하여 로그를 수집하며, 수집한 로그는 별도의 프로세스 상에 동작하는 원격 서비스에서 처리한다.

논문의 구성은 다음과 같다. 2장에서는 연구 배경과 관련 연구를 살펴보며, 3장에서는 제안한 로그 서비스 관리 모델을 제시한다. 4장에서는 적용 사례 및 비교 연구를 통해 로그 서비스 관리 모델의 효용성을 살펴보고 5장에서 결론을 맺는다.

• First Author: Yun-seok Choi, Corresponding Author: Yun-seok Choi
*Yun-seok Choi(cooling@dongduk.ac.kr), Dept. of Computer Science, Dongduk Women's University.
• Received: 2016. 02. 15, Revised: 2016. 02. 19, Accepted: 2016. 03. 01.
• This work was supported by the Dongduk Women's University grant, 2013.

II. Preliminaries

1. Related works

1.1 관점지향 프로그래밍

관심사의 분리를 효과적으로 적용하기 위하여 제안된 관점지향 프로그래밍(Aspect-Oriented Programming, AOP)은 시스템의 목적을 달성하기 위한 기본적이고 대표적인 업무 요구사항을 핵심관심사로 식별하고, 핵심관심사를 지원하기 위하여 시스템 전체에 걸친 부가적인 요구사항을 횡단관심사로 식별하여 각 관심사를 분리하여 개발한 후 위빙(weaving) 과정을 통해 하나의 시스템으로 통합함으로써 시스템 설계와 구현의 복잡성을 감소시키는 개발방법이다[2]. 사용자 인증, 로깅, 데이터 풀링, 트랜잭션 관리 등은 대표적인 횡단관심사이며 관점지향 프로그래밍을 적용하여 애스펙트라는 새로운 모듈 단위로 구현할 수 있다[2][3]. 객체지향 개발방법을 중심으로 소프트웨어를 개발할 경우 핵심관심사는 클래스로 구현하고, 횡단관심사는 관점지향 개발방법을 적용하여 애스펙트로 구현한 후 관심사의 위빙 단계를 거쳐 최종 시스템을 개발한다. 애스펙트는 별도의 모듈로 구현되나 컴파일 등의 과정을 거쳐 위빙 대상의 일부로 포함된다. 즉, 애스펙트는 클래스의 인스턴스와 같이 실행 시에 개별적인 단위로 존재하지 않고 핵심관심사의 일부로서만 존재한다. 결과적으로 애스펙트는 결합 대상인 핵심관심사 모듈의 크기 및 수행시간을 증가시킬 수 있다.

1.2 안드로이드 어플리케이션의 서비스 컴포넌트

안드로이드 어플리케이션은 목적에 따라 기본 컴포넌트인 액티비티, 서비스, 브로드캐스트 리시버, 그리고 컨텐트 프로바이더로 구성할 수 있으며 이들 사이의 통신을 위해 인텐트를 사용한다[4]. 이 중 서비스는 장시간의 작업을 백그라운드에서 수행할 수 있는 컴포넌트로서 어플리케이션 내부의 백그라운드 작업을 수행하거나 외부의 컴포넌트로 동작하며, 원격 인터페이스를 통해 백그라운드 서비스를 제공할 수 있다. 어플리케이션에서 외부에 존재하는 백그라운드 서비스를 이용하고자 할 경우에는 AIDL(Android Interface Definition Language)[5]을 사용하여 원격 인터페이스를 정의하고 바인딩을 수행한 후 해당 서비스의 기능을 활용한다.

1.3 AOP를 활용한 안드로이드 로깅

로깅은 안드로이드 어플리케이션 개발 시 오류 검출, 사용자 입력 확인, 센서 정보 확인 등 디버깅을 위한 필수적인 도구이다[6][7][8]. 안드로이드는 운영체제 수준에서 로그를 제공하며 로그캣이라는 기본 제공 도구를 활용하여 시스템의 로그를 살펴볼 수 있다. 어플리케이션의 실행흐름이나 상태를 파악하고자 할 때에는 개발자가 작성한 로그 코드를 어플리케이션에 삽입하여 해당 정보를 확인할 수 있다.

로깅을 관점지향 프로그래밍을 적용하여 개발할 경우 대상 어플리케이션의 변경 없이 로깅을 수행할 수 있으며, 로깅 기능의 변경 시에도 애스펙트만을 변경하면 되므로 효율적인 로깅이 가능해진다. 안드로이드에서 로깅을 애스펙트로 구현하고자 할 경우 AspectJ의 플러그인을 활용하여 구현할 수 있다[9]. 이에 관점지향 프로그래밍을 적용한 로그를 활용한 다양한 연구가 진행되고 있다. 연구 [10]에서는 안드로이드 어플리케이션의 결함 재생을 위한 정보 획득 방법으로 관점지향 프로그래밍을 적용한 로그의 활용 방안을 제시하였다. 로깅을 애스펙트로 구현하여 적용하므로 기존 어플리케이션의 코드의 수정 없이 로깅을 수행할 수 있으며, 사용자 이벤트 캡처 시 이벤트 실행흐름과 관련 세부 정보를 획득하여 효과적인 오류 재생이 가능함을 확인하였다. 연구 [11]에서는 어플리케이션에 대한 사용자 조작을 효과적으로 추적하기 위해 관점지향 프로그래밍을 사용하여 구현한 로그 활용 방법을 제안하였다. 사용자 조작 흐름의 추적 정보 획득을 위해 애스펙트로 구현한 로그를 적용하고, 획득한 로그 정보를 통해 오류를 재현하는 방법을 제시하였다.

연구 [10], [11]을 통하여 애스펙트로 구현한 로깅의 유용성을 확인할 수 있으나 애스펙트를 활용한 로깅은 다음과 같은 보완점을 갖고 있다. 첫째, 로그의 영구보관 또는 원격전송이 필요할 경우 대상 어플리케이션의 변경이 불가피할 수 있다. 안드로이드는 시스템의 완전성과 사용자의 개인정보 보호를 위해 시스템 자원 활용 시 어플리케이션에서 퍼미션을 획득하여야 하며[4], 이에 따라 로그의 저장 또는 전송을 위해 대상 어플리케이션과 관련 없는 퍼미션을 추가해야 하는 상황이 발생할 수 있다. 둘째, 로깅에 따른 어플리케이션 수행시간의 증가가 발생할 수 있다. 애스펙트로 구현한 로깅은 대상 어플리케이션의 일부로 포함되므로 로깅 수행시간의 증가는 결과적으로 어플리케이션 수행시간 증가를 야기할 수 있다. 모바일 운영체제는 사용자 응답성을 중요시하므로 어플리케이션의 응답시간이 길어질 경우 ANR(Application Not Responding) 경고가 발생할 수도 있다. 따라서 대상 어플리케이션 수행에 대한 로깅의 영향을 감소시키는 방안이 필요하다.

III. The Proposed Model

1. Log Service Management Model

대상 어플리케이션의 변경 및 수행시간에 대한 영향을 최소화하기 위하여 본 연구에서 제안한 로그 서비스 관리 모델은 로그 수집기와 로그 관리 서비스로 구성한다. 로그 수집기는 대상 어플리케이션의 일부로 포함되어 로그를 수집하며, 수집한 여러 로그를 로그의 묶음으로 변환한 후 로그 관리 서비스에 전달한다. 로그 관리 서비스는 대상 어플리케이션과 독립적인 별개의 어플리케이션으로서 안드로이드 기본 컴포넌트인 서비스로 구성한 백그라운드 서비스이며, 전달받은 로그 정보를 처리하여 개발자에게 로그를 제공한다. Fig. 1 은 로그 서비스 관리 모델의 구성을 나타낸다.

로그 수집기는 로그 스니퍼(log sniffer), 로그 관리자, 그리

고 로그 관리 서비스를 호출하기 위한 원격 인터페이스로 구성한다. 로그 스니퍼는 로깅 대상 어플리케이션의 로깅 지점을 포착한다. 원격 로그 관리 서비스를 바인딩하는 시점과 해제하는 시점, 그리고 로그 수집 대상의 수행시점을 파악하여 로그 관리자에게 전달하며, 로그 관리자는 각 시점 별로 적절한 동작을 수행한다. 원격 호출 인터페이스는 로그 관리자가 어플리케이션 외부의 로그 관리 서비스에게 수집한 로그를 전달하기 위하여 사용한다.

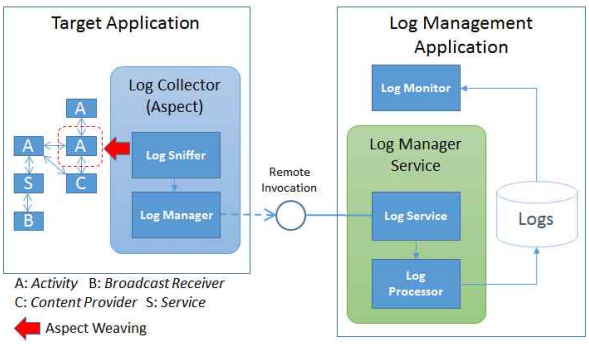

Fig. 1. Log Service Management Model

로그 관리 서비스는 안드로이드 서비스 컴포넌트로 구성하며 독립적인 어플리케이션으로 구성한다. 원격 호출 인터페스를 통해 로깅을 수행하기 위한 서비스 바인딩 요청이 들어오면 로그 기록을 준비한다. 그 후 로그 기록을 위한 인터페이스 호출을 전달 받으면 로그를 영구저장소에 기록하여 로그를 보존하는 등 개발자의 용도에 맞게 로그를 처리한다. 서비스 사용을 종료하여 바인딩 해제 요청을 전달받으면 바인딩을 해제한 후 새로운 서비스 요청을 대기한다.

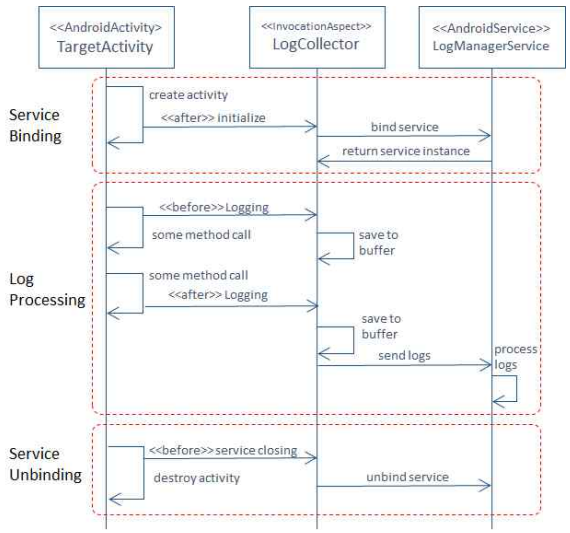

Fig. 2. Sequence Diagram of Log Management Service

Fig. 2 는 연구[12]의 UML의 애스펙트 표기법을 기반으로 구성한 로그 서비스의 바인딩, 기록할 로그 메시지의 전달, 그리고 로그 서비스 해제에 대한 순차도를 나타낸다.

UML 스테레오타입 <<InvocationAspect>>는 특정 메소드 호출과 결합한 애스펙트를 나타내며, 포인트컷의 사전/사후 결합은 <<before>>와 <<after>>로 표시하였다. 안드로이드의 기본 컴포넌트인 액티비티와 서비스를 구분하기 위하여 <<AndroidActivity>>와 <<AndroidService>>로 표기하였다.

2. Log Collector

로그 수집기는 로그를 수집할 대상 어플리케이션에서 로그를 수집하고 수집한 로그를 원격 인터페이스를 통해 로그 관리 서비스에 전달하는 역할을 수행한다. 로그 수집기는 로그 스니퍼, 로그 관리자, 그리고 원격 호출 인터페이스로 구성한다. 로그 수집을 담당하는 로그 스니퍼는 대상 어플리케이션의 변경을 최소화하기 위하여 관점지향 프로그래밍을 적용하여 구성한다. 로그 관리자는 로그 스니퍼에서 수집한 정보에 따라 로그 관리 서비스의 바인딩 또는 바인딩 해제를 수행하며, 로그를 기록할 경우에는 로그를 버퍼에 저장하여 보관하고, 버퍼의 용량을 초과하는 시점에 원격 호출 인터페이스를 호출하여 보관중인 로그를 로그 관리 서비스에 전달한다. 원격 호출 인터페이스는 외부 서비스를 호출하기 위하여 AIDL로 구현한 원격 인터페이스이다.

관점지향 프로그래밍을 적용하여 구성하는 로그 스니퍼는 로그 관리 서비스를 초기화하기 위한 지점, 로깅을 수행하는 지점, 그리고 로그 관리 서비스를 해제하기 위한 지점의 세 가지 포인트컷으로 구성한다. 포인트컷은 실행흐름이나 예외처리 정보 등의 확인이 필요할 경우 상황에 맞게 추가 구성할 수 있다. 다음 Fig. 3 은 로그 관리 서비스 바인딩/바인딩해제에 해당하는 로그 스니퍼의 바이딩/바인딩해제 부분의 포인트컷과 해당 부분의 동작을 정의한 어드바이스를 나타낸다.

```
pointcut init(Context context)
: execution (* mobile..PlaceActivity.onCreate(..))
    && target(context);

after(Context context) : init(context) {
    resultLogs = new StringBuilder();
    if (logService == null) {
        logService = new LogServiceManager(context);
    }
}

pointcut unbindService(Context context)
: execution (* mobile..PlaceActivity.onDestroy(..))
    && target(context);

before(Context context) : unbindService(context) {
    if (logService != null) {
        logService.unbindLogService(context);
        logService = null;
    }
}
```

Fig. 3. Binding/Unbinding part of Log Sniffer

안드로이드 어플리케이션은 일반적으로 단일 액티비티 또는 액티비티의 집합으로 구성하므로 액티비티의 시작과 종료 시점

에 로그 관리 서비스를 바인딩하고 해제하는 것이 적합하다. 액티비티는 실행 시 생명주기별로 정의하고 있는 생명주기 메소드가 호출되며, 생성 시에는 onCreate(), 소멸 시에는 onDestroy()가 호출된다[4]. 따라서 액티비티 별 로그를 관리하고자 할 경우에는 로그 스니퍼에서 로깅 대상 액티비티의 onCreate() 메소드 실행 시에 로그 관리 서비스를 바인딩하고 onDestroy() 메소드 실행 시에 바인딩 해제를 수행한다. 로그 관리 서비스 바인딩 시에는 안드로이드 어플리케이션의 문맥 정보가 필요할 수 있으므로 onCreate()을 수행한 후에 바인딩을 시작하며, 종료 시에는 onDestroy()를 수행하기 전에 바인딩 해제를 수행하도록 구성한다.

로그 관리자는 로그 스니퍼가 전달한 로그 정보와 원격 로그 서비스와의 연결을 관리한다. 로그 스니퍼가 액티비티의 생성을 감지하여 통보하면 원격 호출 인터페이스의 사용을 위해 서비스 바인딩을 수행한다. 정상적으로 바인딩을 수행한 후에는 로그 수신을 대기하며, 로그 스니퍼가 캡처 로그를 전달하면 이를 버퍼에 저장한다. 버퍼는 원격 호출 인터페이스에 대한 잦은 호출로 발생할 수 있는 부하의 최소화를 위해 사용한다. 일정량의 로그가 버퍼에 쌓이면 버퍼의 내용을 원격 호출에 적합한 형태의 데이터 유형에 기록한 후 원격 호출 인터페이스 호출을 사용하여 로그 관리 서비스에 전달한다. 로그 스니퍼가 로깅 대상이 되는 액티비티의 종료를 감지하면 원격 호출 서비스에 대한 바인딩을 해제하여 로그 관리를 종료한다.

원격 호출 인터페이스는 외부 프로세스에서 동작중인 안드로이드 서비스를 호출하기 위해 사용하며, AIDL을 사용하여 정의한다. AIDL은 독자적인 프로세스 상의 클라이언트와 서비스가 통신하기 위해 안드로이드에서 제공하는 인터페이스 정의 언어이다[5]. 로그 관리 서비스가 정의한 AIDL 기반의 인터페이스를 대상 어플리케이션의 개발 프로젝트엔 포함시키면, 안드로이드 SDK Tool은 인터페이스를 정의한 추상클래스를 자동으로 생성한다[5]. 다음 Fig. 4 는 로그 관리자의 원격 호출 인터페이스(a)와 스텁(b)의 예를 나타낸다.

```
interface ILogger {
    boolean putLogMessgage(in String str);
}
```

(a) Remote Interface of Log Manager (AIDL)

```
ILogger.Stub mBinder = new ILogger.Stub() {
    @Override
    public boolean putLogMessgage(String str)
            throws RemoteException {
        StringTokenizer st
        = new StringTokenizer
            (str, System.getProperty("line.separator"));
        while(st.hasMoreTokens()) {
            String log = "log: " + st.nextToken();
            processor.process(log);
        }
        return true;
    }
};
```

(b) Stub of Log Manager

Fig. 4. Remote Interface and Stub of Log Manager

3. Log Management Service

로그 관리 서비스는 안드로이드 서비스 컴포넌트 기반의 어플리케이션으로서 원격 호출 인터페이스를 통해 로그 처리를 제공한다. 클라이언트에 해당하는 로그 수집기가 원격 호출 인터페이스를 통해 전송하는 로그의 묶음을 분리하여 로그 정보를 획득하며, 획득한 로그 정보를 가공하여 개발자가 활용할 수 있는 형태로 제공한다. 로그 관리 서비스는 로깅 대상 어플리케이션과 별개의 어플리케이션으로 구성하며, 이에 따라 복수 개의 로그 수집기에서 전송하는 로그 정보의 처리도 가능하다.

로그 관리 서비스는 로그 서비스 요청을 처리하는 로그 서비스, 전달 받은 로그를 처리하는 로그 처리기, 그리고 로그 처리 상태 확인을 위한 로그 모니터로 구성한다. 이외에 AIDL을 사용하여 정의한 원격 호출 인터페이스를 포함한다. 로그 서비스는 안드로이드 컴포넌트인 서비스를 기반으로 구성한다. 원격 호출 인터페이스 상에서 정의한 스텁을 구현하고 서비스 컴포넌트의 onBind() 메소드에서 구현한 스텁을 반환하도록 구성한다. 로그 처리기는 로그 수집기로부터 전달받은 로그를 개발자가 활용 가능한 형태로 가공한 후 영구보존 또는 원격지에 전송하는 역할을 수행한다. 로그 모니터는 서비스에서 처리중인 로그를 살펴볼 수 있는 사용자 인터페이스를 지원하며, 안드로이드 컴포넌트인 액티비티로 구성한다.

로그 수집기에서 로그를 수집한 후 원격 호출 인터페이스를 통해 로그 관리 서비스에 로그의 묶음을 전달하면 로그 서비스는 로그의 묶음을 분리하여 단일 로그 정보로 변경한다. 안드로이드의 원격 호출 인터페이스는 자바의 기본 자료형과 문자열, 단일 자료형으로 구성한 컬렉션 객체 일부만을 전송할 수 있다. 따라서 본 연구에서는 가장 기본으로 활용할 수 있는 문자열을 사용한다. 필요에 따라서는 문자열 각각을 저장하는 컬렉션 객체를 활용할 수도 있다. 전달받은 로그의 묶음을 정해진 구분자에 의해 식별하여 분리한 후 로그 처리기에 전달한다. 로그 처리기는 로그 정보를 사용자 활용에 적합한 형태로 로그 정보를 가공한다. 실제 환경에서 로깅 대상 어플리케이션을 시험할 경우에는 로그 정보의 보관을 위해 로그 처리기에서 로그를 영구보관 처리할 수 있다. 다수의 기기에서 시험을 진행할 경우에는 수집한 로그 정보를 원격으로 전송하는 역할을 수행할 수 있다. 로그 관리 서비스에 의해 수집한 로그 정보는 로그 모니터를 사용하여 확인할 수 있도록 구성한다.

IV. The Case Study & Evaluations

1. The Case Study

제안한 로그 관리 서비스 모델의 효용성을 살펴보기 위하여 로깅 대상 어플리케이션을 개발하고, 로그 관리 서비스 모델을 적용한 로그 관리 서비스 어플리케이션을 구현하였다. 대상 어

플리케이션은 LBS 기반의 어플리케이션으로서 GPS를 사용하여 현재 위치정보를 확인한 후 지오코딩 정보를 획득하여 위치에 대한 세부정보를 조사한다. 기록한 위치의 정보는 Open API를 사용하여 해당 위치와 관련된 웹상의 정보를 확인할 수 있도록 검색 기능을 제공하며, 현재 위치 주변의 특정 관심사에 대한 정보를 검색하여 지도에서 확인할 수 있도록 구성하였다. Fig. 5 는 로그 관리 서비스의 대상 어플리케이션과 제안한 모델을 기반의 로그 관리 서비스 어플리케이션을 나타낸다.

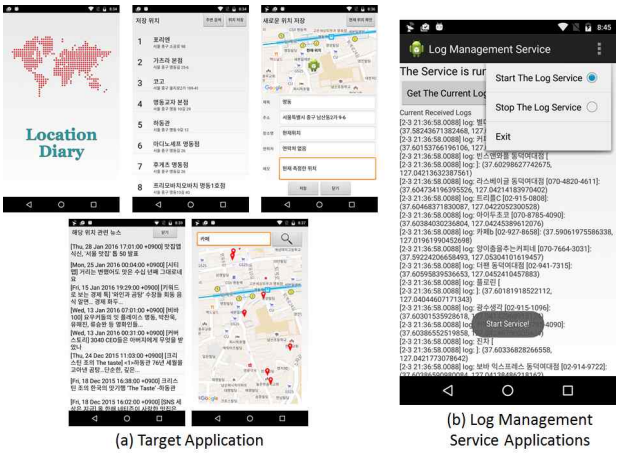

Fig. 5. Target Application and Log Management Service Application

대상 어플리케이션은 GPS와 네트워크를 이용하는 LBS 기반의 어플리케이션이므로 다양한 요소에 의한 오류가 발생할 수 있다. 실사용 환경에서의 시험 시 로그의 즉각적인 확인이 불가능할 수 있으므로 로그를 파일을 사용하여 기기에 보존하는 형태로 로그 관리 서비스를 구현하였다.

로깅의 내용은 대상 어플리케이션의 기능 중 현재 위치 정보 확인 후 주변의 특정 관심정보를 Open API를 사용하여 조사하고 조사결과를 지도에 표시하는 기능이 정상적으로 수행하는지 확인할 수 있도록 구성하였다. Open API를 통해 조사한 결과를 파싱한 후 위치 정보를 보관하는 객체에 기록하는 과정을 로깅하여 조사한 정보의 세부내용을 기록하였다.

로그 스니퍼는 관심 위치 표시를 담당하는 액티비티 생성 시 로그 서비스 매니저에게 로그 서비스 바인딩을 요청하고 소멸 시 바인딩 해제를 요청한다. 로깅 목표인 관심 장소를 기록하는 메소드 수행 시 해당 메소드를 포인트컷의 결합식으로 식별하여 로그 서비스 관리자에게 로깅 정보를 전달한다.

로그 서비스 관리자는 로그 스니퍼로부터 전달받은 관심장소를 기록한 로그를 로그버퍼에 분리자와 함께 추가하며, 로그버퍼의 용량을 초과하면 원격 호출 인터페이스를 통해 로그의 묶음을 전송하도록 구성하였다. 다음 Fig. 6 은 로그 서비스 관리자의 일부를 나타낸다.

```
public class LogServiceManager {

    public final static String TAG = "log_service";
    private final String LOGGER
    private final int MAX_LOGBUFFER = 256;
    private ILogger logger = null;
    private StringBuffer logBuffer = null;

    public LogServiceManager(Context context) {

    public boolean initLogService(Context context) {
        Intent intent = new Intent(LOGGER);
        boolean isBind
        = context.bindService(intent, serviceConn,
                              Context.BIND_AUTO_CREATE);
        return isBind;
    }

    public void unbindLogService(Context context) {
        Log.d(TAG, "Unbind Service");
        context.unbindService(serviceConn);
    }

    public void sendLog(String logMessage) {
        logBuffer.append(logMessage
                + System.getProperty("line.separator"));

        if (isCapacityOver(logBuffer)) {
            try {
                logger.putLogMessgage(logBuffer.toString());
                logBuffer = new StringBuffer();
            } catch (RemoteException e) {
                Log.e(TAG, "Service not found");
            }
        }
    }
```

Fig. 6. A part of Log Service Manager

로그 서비스는 원격 호출을 통해 전달받은 관심위치 정보 로그의 묶음을 분리자를 기준으로 분리한 후 로그 처리기에 전달한다. 로그 처리기는 대상 어플리케이션이 LBS 기반의 어플리케이션임을 감안하여 전달받은 로그를 외장메모리에 파일로 저장하였다. 다음 Fig. 7 은 로그 처리기가 저장한 로그 파일을 확인한 결과를 나타낸다.

```
[2-11 21:9:28.0521] log: 파스쿠찌 명동2호점 [02-776-8497]: (37.56408793465139, 126.9848511284075)
[2-11 21:9:28.0521] log: 카페엠셀브즈 [02-2266-5942]: (37.569191534088716, 126.98796519785786)
[2-11 21:9:28.0521] log: 드롭탑 명동점 [02-3789-7801]: (37.5646898483298, 126.98523131598029)
[2-11 21:9:28.0521] log: 고관한복촬영소 명동점 [02-318-8288]: (37.56139195326589, 126.98336672186578)
[2-11 21:9:28.0521] log: 할리스커피 명동점 [02-3789-8022]: (37.56282654144965, 126.98486496596423)
[2-11 21:9:28.0521] log: 카페마마스 3호시청점 [02-318-1230]: (37.56739195879878, 126.97937652192165)
[2-11 21:9:28.0521] log: 모리나리 [02-771-1808]: (37.55796808506492, 126.982810651293)
[2-11 21:9:28.0521] log: 고양이놀이터 명동점 [02-3789-2207]: (37.561986889989406, 126.98548085269667)
[2-11 21:9:28.0521] log: 가루 [02-776-3141]: (37.56295611940075, 126.98362443526973)
[2-11 21:9:28.0521] log: 카페이마 [02-2020-2088]: (37.56988921053745, 126.97768243311059)
[2-11 21:9:28.0521] log: 2Floor [02-319-9635]: (37.55843348119251, 126.98651825153947)
[2-11 21:9:28.0521] log: 해피레몬플러스 명동직영점 [070-4114-8103]: (37.56477269643533, 126.98488720973461)
[2-11 21:9:28.0521] log: 큐플러스롤카페 명동2호점 [02-3789-8110]: (37.561968871170215, 126.98548991085116)
[2-11 21:9:28.0521] log: 고양이다락방 명동본점 [02-318-3123]: (37.56328778433953, 126.984348748064)
[2-11 21:9:28.0521] log: 시실리 [02-779-1343]: (37.56291289717051, 126.98381006755616)
[2-11 21:9:28.0521] log: 반지대학 명동캠퍼스 [02-3789-6620]: (37.561110808242056, 126.9831268386891)
[2-11 21:9:28.0521] log: 팀벅와플 덕수궁직영점 [02-318-5202]: (37.56482537520031, 126.9764524633968)
[2-11 21:9:28.0521] log: 투썸플레이스 명동역점 [02-775-3216]: (37.560206540887, 126.98564870539924)
[2-11 21:9:28.0521] log: 브라운하우스커피 본점 [02-562-5175]: (37.559316530494854, 126.98716547751754)
[2-11 21:9:28.0521] log: 별다방미스리 인사동점 [02-739-0939]: (37.575173587728294, 126.98339988201602)
[2-11 21:9:28.0521] log: 팀리챗 [02-318-2464]: (37.56208762494434, 126.9841090503234)
[2-11 21:9:28.0521] log: 설빙 서울명동2호점 [02-774-7994]: (37.56302841675475, 126.98527691788773)
[2-11 21:9:28.0521] log: 탐앤탐스커피 명동역점 [02-774-8163]: (37.56095233596868, 126.98386481900687)
[2-11 21:9:28.0521] log: 크레이지브라운 엠플라자 [02-727-3235]: (37.562531053090445, 126.98516835905474)
[2-11 21:9:28.0521] log: 프린스 [02-752-7114]: (37.56067873805267, 126.98627790269269)
[2-11 21:9:28.0521] log: 제이홀릭 [02-3445-0126]: (37.562927348830186, 126.98407265286752)
[2-11 21:9:28.0521] log: 호미빙 명동본점 [02-318-0677]: (37.561694938458906, 126.98525001642477)
[2-11 21:9:28.0521] log: 스타벅스 명동점 [02-758-8134]: (37.561813873971715, 126.98528168421387)
[2-11 21:9:28.0521] log: 블리스앤블레스 명동점 [02-318-3773]: (37.558667727993964, 126.98640955823514)
[2-11 21:9:28.0521] log: 만나카페 [02-310-5031]: (37.560411274826706, 126.98077283444364)
```

Fig. 7. Result of logging

수행 결과 로그 관리 서비스는 대상 어플리케이션의 로깅 정보를 정확하게 기록할 수 있었으며, 관점지향 프로그래밍을 적용하였으므로 대상 어플리케이션의 코드 수정 없이 로깅을 수행할 수 있었다. 또 기록한 로그는 별개의 서비스에 의해 외장메모리에 정상적으로 기록됨을 확인할 수 있었으며, 로그모니

터를 통해 그 내용을 확인할 수 있었다.

2. Evaluations

제안한 모델의 효용성을 확인하기 위하여 로깅방법 별로 응답시간 비교를 수행하였다. 기본적인 로깅, 관점지향 프로그래밍을 적용한 로깅, 그리고 제안한 모델을 적용한 로깅을 각각 구현하여 적용사례와 동일한 로깅을 처리하도록 구성한 후 수행시간을 측정하였다. 수행시간의 측정은 시험 대상 어플리케이션의 기능 중 Open API가 반환하는 관심정보 개수(각 15개, 30개, 45개)별로 구분하여 측정하였으며, 각 개수별로 3회 반복 측정하여 평균값을 계산하였다. 시험 평가 환경은 안드로이드 Marshmallow 6.0.1 버전의 Nexus 5 기기에서 wi-fi 환경으로 구성하였으며 동일 위치의 시험을 위해 Mock-up GPS를 사용하였다. 측정 시 SD카드의 저장 파일을 초기화 한 후 반복을 진행할 때 로그를 추가하는 방식으로 시험을 진행하였다. 다음 Fig. 8 은 각 방법의 로그 개수별 로깅 평균 수행시간을 나타낸다.

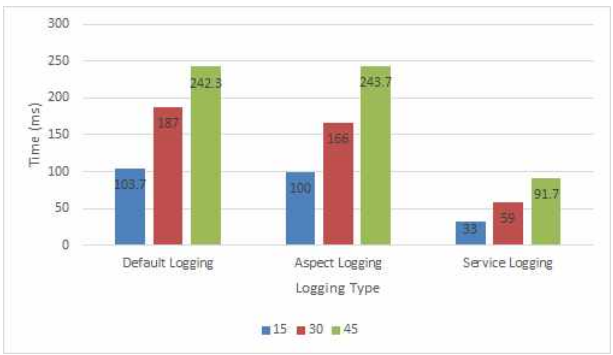

Fig. 8. Average Response Time of Logging

기본 로깅은 로그 수집 및 저장 코드가 대상 어플리케이션 내에 포함되어 있으므로 로깅의 수행시간은 대상 어플리케이션의 수행시간에 직접적으로 영향을 준다. 관점지향 프로그래밍을 적용한 로깅은 개발 시에는 로깅 모듈이 분리되어 있으나 컴파일 시에는 대상 어플리케이션에 포함되므로 기본 방식과 수행시간에 있어 큰 차이가 없음을 확인할 수 있다. 제안한 모델을 적용한 로깅은 로그 수집 후의 처리는 별개의 서비스에서 수행하므로 다른 두 방법과 대비하여 수행시간에 대한 영향이 적음을 확인할 수 있다.

비교 평가에서 살펴볼 수 있었던 바와 같이 제안 모델을 적용한 로깅은 대상 어플리케이션의 변경 없이 로깅을 수행할 수 있었으며, 수행시간에 있어서도 기존의 방법보다 빠른 처리가 가능함을 확인할 수 있다. 기존 방법들과 제안한 모델 사이의 비교는 Table 1에서 살펴볼 수 있다.

Table 1. Comparisons of Logging Types

Comparison \ Type	default logging	aspect logging	Proposed model used
detailed logging	Available	Available	Available
write logging code	not easy	easy	easy
logging modification	all position of logging	aspect only	aspect only
logging code influence	source code changing	configuration changing	no influence
logging time influence	directly affect	directly affect	only capture time

세 가지 로깅 방법 모두 상세한 정보를 획득할 수 있다. 그러나 로깅 대상을 지정할 때 기본 로깅의 경우에는 로깅 대상을 코드 상에서 직접 식별하고 로깅 코드를 추가해야 한다. 애스펙트 로깅과 제안한 모델 기반의 로깅 방법은 로깅 대상을 와일드카드 문자를 적용한 결합식을 사용하여 선택할 수 있으므로 효율적인 로깅 대상 지정이 가능하다. 로깅 대상이나 방법에 변경이 필요할 경우 기본 로깅은 해당하는 모든 로깅 코드를 추적하여 변경하여야하나 관점지향 프로그래밍을 적용한 방법들은 로깅을 처리하는 애스펙트만 변경하므로 작업 효율을 높일 수 있다. 기본 로깅의 경우 로깅 코드를 대상에 직접 추가하므로 대상의 변경이 발생하며 관점지향 프로그래밍을 적용하였을 경우에는 대상 코드를 변경하지 않는다. 그러나 관점지향 프로그래밍을 적용한 일반적인 방법은 로그의 저장 등이 필요할 경우 안드로이드의 특성상 대상 어플리케이션의 설정정보를 변경하여 권한을 추가해야 하는 상황이 발생한다. 제안한 모델을 적용한 방법의 경우 로그 처리를 별개의 서비스에서 처리하므로 대상 어플리케이션의 코드 및 설정정보에 대한 어떠한 변경도 발생하지 않는다. 로깅 수행시간의 경우 기본 로깅과 관점지향 프로그래밍을 적용한 로깅 모두 대상 어플리케이션에 포함되어 수행되므로 수행시간에 영향을 준다. 제안한 모델을 적용한 방법은 로그 수집과 관련한 동작 이외에는 별도의 서비스에서 처리하므로 수행시간에 대한 영향을 최소화 할 수 있다.

V. Conclusions

본 논문에서는 안드로이드 어플리케이션의 효율적인 로깅을 위한 로그 서비스 관리 모델을 제안하였다. 제안한 모델은 관점지향 프로그래밍의 장점을 수용하여 로깅 대상 어플리케이션에 대한 변경을 최소화할 수 있었으며, 로그 처리를 독립적인 서비스에서 수행하므로 대상 어플리케이션에 대한 로깅의 영향을 최소화할 수 있었다. 제안한 모델의 적용사례와 기존 방법과의 비교를 통해 효용성을 확인하였다.

제안한 모델을 활용할 경우 로깅을 활용한 다양한 서비스 확

장이 가능할 것이다. 이를 위해 향후에는 안드로이드 어플리케이션의 다양한 상황 정보를 효과적으로 캡처하기 위한 애스펙트 설계 연구가 필요하며, 다수의 기기에서 생성하는 로그를 원격 상에서 효율적으로 관리하는 방법에 대한 연구가 필요하다.

REFERENCES

[1] Reading and Writing Logs, http://developer.android.com/tools/debugging/debugging-log.html

[2] Kiczales G., Irwin J., Lamping J., Loingtier J.-M., Lopes C., Maeda C., and Mendhekar A., "Aspect-Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming(ECOOP'97), Springer-Verlag, Finland, pp.220-242, June 1997.

[3] Laddad R., AspectJ in Action, Manning Publications, 2005.

[4] Android Application Fundamentals, http://developer.android.com

[5] Android Interface Definition Language (AIDL), http://developer.android.com/guide/components/aidl.html

[6] Jinxin Liu, Hao Wu and Huabin Wang, "A detection method for malicious codes in Android apps", Wireless communications, Networking and Mobile Computing (WiCOM 2014), 10th International Conference on., pp.514-519, 2014.

[7] Hirabe, Y., Arakawa, Y. and Yasumoto, K., "Logging All the Touch Operations on Android:, Mobile Computing and Ubiquitous Networking (ICMU), 2014 7th International Conference on., pp.93-94, 2014.

[8] Won-Jae Yi, Sarkar, O., Mathavan, S. and Saniie, J., "Wearable Sensor Data Fusion for Remote Health Assesment and Fall Detection", Electro/Information Technology, 2014 IEEE International Conference on, pp.303-307, 2014.

[9] AJDT: AspectJ, http://www.eclipse.org/aspectj/

[10] Ajay Kumar Jha and Woo Jin Lee, "Activity-based Event Capture and Replay Technique for Reproducing Crashes in Android Applications", IEMEK J. Embed. Sys. Appl., Vol. 9, No.1, pp.1-9, Feb. 2014.

[11] Mun-Chan Kim and Choong-Kyo Jeong, "User Control Trace for Android Application Failure Analysis", Journal of KIIT., Vol. 13, No. 3, pp.73-83, Mar. 2015.

[12] Ballal, R., "Extending UML for Aspect Oriented Software Modeling", Computer Science and Information Engineering, 2009 WRI World Congress on, Vol. 7, pp.488-492, Mar. 2009.

Authors

Yun-Seok Choi, he received the BS, MS, and the Ph.D. degrees in computer science and engineering from Soongsil University in 1997, 1999, and 2008, respectively. He is an assistant professor in the Department of Computer Science, Dongduk Women's University. His recent interests focus on software architecture, mobile software and aspect-oriented programming.