Journal of The Korea Society of Computer and Information
Vol. 21 No. 10, pp. 69-76. October 2016

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.10.069

A Real-time Remote Logging Model for Development of Location-Based Mobile Applications

Yun-seok Choi*

Abstract

In this paper, we propose a real-time remote logging model for development of android applications using LBS(Location based Service). The model has two major parts: Mobile Log Management Service and Remote Log Server. Mobile Log Management Service consists of the log collector and the remote log manager. The log collector is an aspect of AOP which can collect logs from the target application without modifications of source codes. The remote log manager has a background service component so that it can receive logs whenever the log collector captures logs from the target application. Remote Log Server communicates with Mobile Log Management Service by socket interface. Therefore, Remote Log Server can show logs in real-time. To validate the efficiency of the proposed model, we show a case study, and compare the model with other models.

▸ Keyword : LBS(Location Based Service), Android, Remote Logging, Aspect-Oriented Programming

I. Introduction

모바일 어플리케이션은 이동성을 기반으로 네트워크, GPS, NFC, 카메라, 그리고 다양한 센서 등을 활용하여 서비스를 제공할 수 있다. 모바일 하드웨어와 플랫폼의 발전을 통해 증강현실과 위치기반 서비스 등 모바일 특성을 활용한 서비스를 더욱 효과적으로 제공할 수 있는 기반이 만들어지고 있다. 한 예로 최근 등장한 모바일 게임은 증강현실과 위치기반 서비스, 그리고 센서 기술 기반 위에 컨텐츠를 제공하여 전 세계적으로 선풍적인 인기를 끌고 있다[1]. 실내에서의 위치도 비콘 등의 기술을 통해 정밀한 확인이 가능하게 되어 이를 사용한 다양한 모바일 어플리케이션의 등장을 기대할 수 있다[2]. 이와 같이 이동성을 토대로 다양한 기술들을 조합하여 서비스를 제공하는 모바일 어플리케이션 개발에는 실행과 관련한 개발자 관심 정보 및 오류 관련 정보 등을 원격 상에서 효율적으로 수집하고 분석할 수 있는 방법이 필수적이다.

로그는 어플리케이션 개발에 활용하는 기본 도구 중 하나로서 어플리케이션 실행 및 오류에 관한 다양한 정보를 획득할 수 있다. 안드로이드의 경우 플랫폼 자체에서 로그 확인 도구인 로그캣(logcat)을 제공하고 있어 시스템에서 제공하는 정보, 어플리케이션이 발생시킨 실행 관련 정보, 그리고 개발자가 직접 지정한 정보를 살펴볼 수 있다[3]. 로그는 간단하면서도 효율적이 도구이긴 하나 개발자가 직접 로그 확인을 지정할 경우 어플리케이션의 규모가 커질수록 해당 로그 코드의 변경 및 추적에 어려움이 있을 수 있다. 이러한 문제를 해결하기 위하여 관점지향 프로그래밍(AOP, Aspect-Oriented Programming)을 활용한 로깅 관련 연구가 진행되었다. 관점지향 프로그래밍은 시스템의 요구사항을 핵심관심사와 핵심관심사를 지원하는 횡단관심사로 분리하여 개발한 후 결합(weaving)을 통해 시스템을 구현하는 방법으로서 관심사의 분리에 효과적으로 적용할 수 있는 개발 방법이다[4]. 로깅은 대표적인 횡단관심사로서 모바일 어플리케이션에 관점지향 프로그래밍을 적용한 로깅을 활용하는 다양한 연구가 진행되고 있다. 그러나 실사용 환경에서 실행하는 모바일 어플리케이션의 로그를 원격 상에서 확인할 수 있는 방법에 대한 연구는 부족한 상황이다. 플랫폼 또는 오류 보고 서비스를 이용할 경우에는 원격에서 로그 확인이 가능하나 이 경우 오류와 관련한 정보만 확인이 가능하며 개발자가 지정한 로그는 확인할 수 없다. 또한 해당 정보는 실시간이

• 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. 10. 11, Revised: 2016. 10. 17, Accepted: 2016. 10. 25.

아닌 로그 획득 후 보고 형식으로 개발자에게 제공되고 있어 다양한 환경 변수 및 실행 상황에 대한 실시간 확인이 어려운 상황이다. 그러므로 위치 기반 서비스를 제공하는 모바일 어플리케이션을 실제 사용 환경에서 시험할 때 원격 상에서 실시간으로 로그를 획득할 수 있는 방법에 대한 연구가 필요한 상황이다. 이에 본 논문에서는 연구[5]를 확장하여 위치 기반 서비스를 제공하는 모바일 어플리케이션에서 생성하는 로그를 원격 상에서 실시간으로 확인할 수 있는 원격 로그 서비스 모델을 제안한다. 제안한 모델을 적용할 경우 관점지향 프로그래밍을 활용하므로 로깅 대상 어플리케이션에 대한 영향을 최소화할 수 있으며, 기존 방법에서 확인 가능한 오류 관련 로그뿐만 아니라 개발자가 지정한 로그를 원격 상에서 실시간으로 확인할 수 있다.

논문의 구성은 다음과 같다. 2장은 연구 배경 및 관련 연구를 살펴보며, 3장에서는 제안한 원격 로그 서비스 모델에 대하여 살펴본다. 4장에서는 적용사례 및 비교평가를 통해 제안한 모델의 효용성을 살펴보고 5장에서는 결론을 맺는다.

II. Preliminaries

1. Related works

1.1 안드로이드 어플리케이션의 로그

안드로이드 프레임워크는 개발자가 필요에 따라 로그를 확인할 수 있도록 Log 클래스를 제공하며, 개발도구인 안드로이드 스튜디오의 로그캣 모니터에서 개발자가 지정한 로그, 시스템에서 제공하는 로그, 그리고 오류 관련 스택 정보와 관련한 로그를 실시간으로 확인할 수 있다[3]. 기본적인 로깅 방법은 간단하게 사용할 수 있는 장점이 있으나 다음과 같은 사항을 고려하여야 한다. 첫째 산재한 로그 코드를 관리하기 위한 개발자의 추가적인 노력이 필요하다. 어플리케이션의 실행 흐름을 추적하기 위해 로그 코드를 삽입하였을 경우 로그 코드를 실행 흐름에 적합한 위치에 삽입하였는지 대한 확인이 필요하며, 로그 코드의 변경이 필요할 경우에는 해당 위치를 개발자가 직접 추적하고 작업하여야 한다[6]. 둘째 기본 로그는 개발도구와 모바일 기기를 연동한 상태일 경우에만 확인할 수 있으므로 이동성을 고려한 어플리케이션 시험 시에는 로그를 저장하는 등 로그 확인을 위한 별도의 작업이 필요하다. 더불어 개발도구와 연동한 상태의 기기에 대한 로그만 수집할 수 있으므로 다수의 모바일 기기에서 동시에 시험을 수행할 경우에는 로그 수집이 어려울 수 있다.

1.2 AOP 기반의 안드로이드 로깅

관점지향 프로그래밍은 시스템의 요구사항을 핵심관심사와 횡단관심사로 분류하여 개발한 후 컴파일 또는 실행 시에 결합하여 시스템의 기능을 완성하는 개발 방법으로서[4], 모듈의 응집도를 높이고 결합도를 낮추는 장점이 있다. 로깅은 횡단관심사로 식별할 수 있는 대표적인 기능으로서 관점지향 프로그래밍을 활용한 안드로이드 어플리케이션 로깅 방법에 대한 다양한 연구가 진행되었다. 연구[7]과 연구[8]은 관점지향 프로그래밍을 적용하여 어플리케이션의 오류를 재현하는 방법을 제시하였다. 관점지향 프로그래밍의 모듈 단위인 애스펙트에서 로깅을 수행하므로 로그 코드의 산재를 막고 변경 및 관리를 용이하게 수행할 수 있었으며, 수집한 로그를 사용하여 모바일 어플리케이션의 오류를 추적하고 재현할 수 있음을 확인하였다. 연구[5]는 로그 코드의 적용에 따른 대상 어플리케이션의 변경 및 실행시간 영향을 최소화하는 방법을 제시하였다. 관점지향 프로그래밍 기반의 로깅에 안드로이드 서비스 컴포넌트를 활용하여 로그 수집 시 대상 어플리케이션의 변경 및 로깅에 따른 실행시간 영향을 최소화하였다. 연구[5], [6], 그리고 [7]을 통해 관점지향 프로그래밍을 적용한 로깅의 유용성 및 효율성을 확인할 수 있었으나 위치기반 서비스 모바일 어플리케이션 개발에 적용하기 위해서는 원격 로깅에 대한 추가적인 연구가 필요한 상황이다.

1.3 안드로이드 오류의 원격 확인

안드로이드 어플리케이션의 경우 오류 정보를 배포 플랫폼 또는 별도의 오류 보고 서비스를 이용하여 원격 상에서 확인할 수 있다. 안드로이드 배포환경의 경우 배포 또는 시험 버전에 해당하는 어플리케이션을 배포한 후 오류 발생 시 사용자의 동의를 받아 관련 로그를 개발자가 확인할 수 있도록 전송하는 오류 보고 서비스를 제공한다[9]. 안드로이드 어플리케이션 시험 서비스인 fabric.io는 오류 보고를 확인하기 위한 서비스로서, 안드로이드 개발환경에 fabric 관련 플러그인을 설치하여 어플리케이션 개발 후 오류가 발생하면 웹 콘솔에서 관련 로그를 살펴볼 수 있도록 오류 관련 정보를 제공한다[10]. 각 방법은 원격 상에서 어플리케이션의 오류와 관련한 로그를 확인할 수 있으나 다음과 같은 제약점이 있다. 안드로이드 배포환경의 경우 오류의 전송 여부가 사용자에게 있으므로 오류 수집이 어려울 수 있다. fabric.io 서비스의 경우에는 사용자의 동의 여부와 상관없이 오류 정보를 확인할 수 있으나 두 방법 모두 오류 정보 이외에 개발자가 추가한 로그 정보는 확인할 수 없으므로 실행 및 오류와 관련한 추가적인 정보의 수집이 어려울 수 있다. 따라서 원격 상에서 개발자가 필요에 따라 수집하고자 하는 개발자 지정 로그 및 오류와 관련한 로그를 실시간으로 수집하고 확인하는 방법에 대한 추가적인 연구가 필요하다.

III. The Proposed Model

1. Remote Log Service Model

본 연구에서 제안하는 원격 로그 서비스 모델은 관점지향 기

반 로깅 모델을 적용하여 대상 어플리케이션에 대해 로그 코드 추가 없이 로깅을 수행하며, 어플리케이션 실행에 대한 영향을 최소화하기 위하여 수집한 로그의 제어 및 원격 전송은 대상 어플리케이션과 별도의 백그라운드 서비스에서 수행한다. 원격 서버에 대한 로그 전송은 실시간 확인을 위해 소켓 통신 기반으로 수행하며, 서버로 전송한 로그는 웹 기반 사용자 인터페이스에서 확인할 수 있도록 구성한다.

Fig. 1은 원격 로그 서비스 모델의 전체 아키텍처를 나타낸다. 각 구성요소는 필요에 따라 스테레오타입을 표시하여 구분한다[11][12]. 제안한 모델은 모바일 기기 상에서 대상 어플리케이션으로부터 로그를 수집하고 관리하는 모바일 로그 관리 서비스 모델과 모바일 기기로부터 전달받은 로그를 처리하는 원격 로그 서버 모델로 구성한다. 모바일 모델은 로그 수집기와 원격 로그 관리자로 구성한다. 로그 수집기는 관점지향 프로그래밍 모듈인 애스펙트로 구성하여 개발 시 로깅 대상 어플리케이션에 포함시킨다. 로그 수집기는 수집한 로그를 백그라운드에서 실행중인 원격 로그 관리자에게 전송한다. 원격 로그 관리자는 수집한 로그를 통신에 적합한 형태로 구성한 후 서버에 전송하며 전송 상태를 기록한다. 원격 로그 서버 모델은 로그를 실시간으로 수신하는 서버 소켓 관리자와 수신한 로그를 처리하는 원격 로그 서버 관리자, 그리고 로그 정보를 확인할 수 있는 로그 모니터로 구성한다.

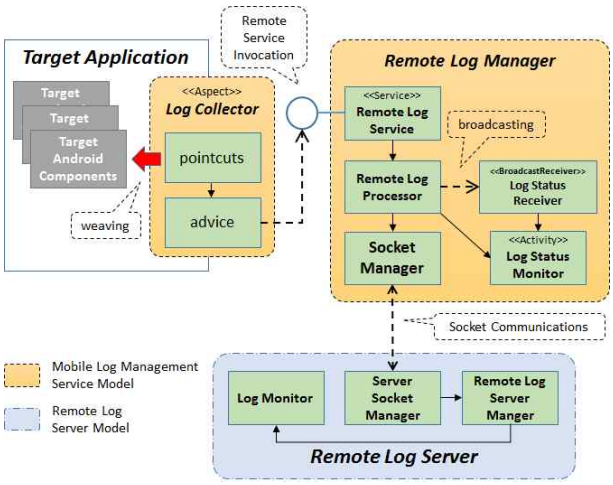

Fig. 1. The Architecture of Remote Log Service Model

2. Mobile Log Management Service Model

모바일 로그 관리 서비스 모델은 로그 수집 대상 어플리케이션으로부터 로그를 수집한 후 소켓 통신을 통해 원격 로그 서버에 전송하는 역할을 수행하며, 대상 어플리케이션과 결합하여 로그를 수집하는 로그 수집기와 수집한 로그를 관리하는 원격 로그 관리자로 구성한다. 로그 수집기는 관점지향 프로그래밍 모듈인 애스펙트로 구성하여 대상 어플리케이션에 포함시킨다. 원격 로그 관리자는 대상 어플리케이션과 별개의 독립적인 어플리케이션으로 구성하며 로그 수집기에서 수집한 로그를 백그라운드 실행 상태에서 수신, 원격 로그 서버와의 연결 관리, 그리고 로그의 전송 등을 담당한다.

2.1 Log Collector

로그 수집기는 대상 어플리케이션에서 개발자 지정 로그와 어플리케이션 실행 시 발생할 수 있는 오류 관련 로그를 수집한다. 로그 수집기는 애스펙트 모듈로 구성하여 개발 시에는 대상 어플리케이션과 물리적으로 분리되어 있으나 컴파일 또는 실행 시에 지정한 결합점(join point)에 결합된다. 그러므로 로그 코드의 추가로 인한 대상 어플리케이션의 변경 없이 로그 수집이 가능하다.

로그 수집기는 로그 수집을 위한 초기화 단계, 로그 수집 단계, 그리고 종료 단계를 수행하며, 각 단계를 수행하기 위해 대상 어플리케이션 특정한 실행 지점을 각 단계 실행을 위한 결합점으로 지정한다. 초기화 단계는 로그 수집기가 원격 로그 관리자와의 서비스 호출을 구성하는 단계로서 개발자가 지정한 로그 또는 오류가 발생할 수 있는 시점 이전에 수행하여야 한다. 안드로이드 어플리케이션은 각 구성요소 별로 생명주기에 따른 콜백 메소드를 제공하며[13], 각 메소드의 실행 시점은 개발자의 코드를 실행하기 이전이므로 로그 수집기의 초기화 단계 결합점으로 적합하다. 안드로이드 액티비티의 경우 객체 생성 시 최초 호출되는 onCreate() 또는 화면 표시 직전 단계인 onResume() 등을 로그 수집기의 초기화 시점으로 지정할 수 있다. 안드로이드 서비스를 로그 수집 대상으로 선정할 경우에는 onCreate() 또는 onStartCommand(), 그리고 onBind()를 초기화 시점으로 지정할 수 있으며, 안드로이드 방송수신자의 경우 onReceive()를 지정할 수 있다. 로그 수집 단계는 로깅 대상 어플리케이션에서 개발자가 지정하였거나 오류에 의해 발생하는 로그를 수집하는 단계로서 생성자 및 메소드의 실행 전, 실행 중, 실행 후 시점, 필드의 참조 또는 클래스 및 객체의 초기화, 그리고 예외처리 상황 등의 로그를 수집할 수 있다[6]. 로그 수집을 완료한 후에는 종료 단계를 통해 원격 로그 관리자로의 서비스 호출을 종료하여야 한다. 액티비티에 대한 로그를 수집하였을 경우 액티비티가 제거되는 시점인 onDestroy() 메소드 실행을 로그 수집 종료 시점으로 지정할 수 있다. Fig. 2는 로그 수집기의 수행 단계에 따라 AspectJ로 구성한 애스펙트 결합점의 예를 나타낸다.

```
(a) Initializing
pointcut init(Paramter params)
: execution (* package..AndroidComponent.method(..)) && target (params);

(b) Unbinding
pointcut unbindService(Paramter params)
: execution (* package..AndroidComponent.method(..)) && target (params);

(c) Capturing custom logs
pointcut captureCustomLog(Parameter params)
: call (access_specifier * root_package..*.target_method(..)) && args (params);

(d) Capturing exceptions
pointcut captureException()
: call (access_specifier * root_package.. AndroidComponent.*(..)) && !within (Aspect);
```

Fig. 2. examples of pointcuts for Log Collector

로그 수집 작업은 애스펙트의 충고(advice)에서 수행한다. 충고는 컴파일 또는 실행 시에 대상 어플리케이션 코드와 결합하므로 대상 어플리케이션의 실행시간을 증가시킬 수 있다. 제안한 모델에서는 로그 수집에 따른 대상 어플리케이션 실행시간 영향을 최소화하기 위하여 수집한 로그에서 필수 정보를 추출하는 것 이외에 별도의 가공 없이 서비스 호출을 통해 원격 로그 관리자에게 전달한다. Fig. 3은 AspectJ로 구성한 로그 수집기 충고의 예를 나타낸다.

```
(a) Custom Log Advice
after(Parameter params) : captureCustomLog(params){
   RemoteLogManagementService.sendLog(params);
}

(b) Exception Log Advice
after() throwing(Throwable info) : captureException(){
   // Extract error logs from Throwable Info.
   RemoteLogManagementService.sendLog(error_logs);
}
```

Fig. 3. Examples of Advices for Log Collector

2.2 Remote Log Manager

원격 로그 관리자는 로그 수집 대상 어플리케이션과 별개의 독립적인 안드로이드 어플리케이션으로서, 백그라운드 실행 상태에서 로그 수집기가 수집한 로그를 서비스 호출을 통해 전달받은 후 원격 로그 서버에 전달하는 역할을 수행한다. 원격 로그 관리자는 원격 로그 서비스, 원격 로그 처리기, 로그 상황 모니터, 그리고 소켓 통신 관리자로 구성한다. 이외에 백그라운드 상태에서 로그 수집 상태를 안드로이드 방송을 통해 수신하는 로그 상태 수신자를 추가할 수 있다. 원격 로그 서비스는 백그라운드 실행 상태에서 로그 수집기의 서비스 호출을 통해 로그를 전달받는다. 원격 로그 처리기는 전달 받은 로그에 수집 시간 등의 정보를 추가한 후 소켓 통신 관리자를 사용하여 원격 로그 서버에 전송하며, 전송 상황을 안드로이드 방송을 사용하여 방송한다. 로그 상황 모니터는 원격 로그 관리자의 실행 및 종료, 그리고 원격 로그 처리기의 방송 상황을 표시한다.

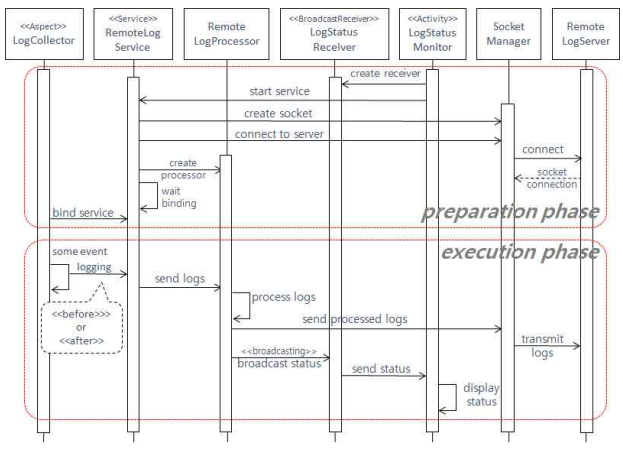

Fig. 4. Sequence Diagram of Remote Log Manager

원격 로그 관리자의 동작은 준비 단계와 로그 처리 수행 단계로 구분할 수 있다. 준비 단계에서는 원격 로그 서비스를 중심으로 로그 수집기와의 서비스 호출 설정, 원격 로그 서버와의 통신 연결 설정, 그리고 원격 로그 처리기의 초기화를 수행한다. 원격 로그 서비스는 원격 로그 서버와의 통신 소켓을 생성한 후 원격 로그 처리기에 전달하고 로그 수집기의 서비스 바인딩 요청을 대기한다. 바인딩 요청을 수신하면 원격 호출 메소드에 대한 스텁을 로그 수집기에 전달하여 서비스 호출을 수행할 수 있도록 준비한다. 로그 수집기는 스텁에 정의한 메소드를 통해 원격 로그 서비스의 서비스 호출이 가능하다. Fig. 4는 원격 로그 관리자의 준비 및 수행 단계 순차도를 나타낸다.

로그 처리 단계에서는 수집한 로그를 원격 로그 서버에 전송하는 작업을 수행한다. 로그 수집기는 대상 어플리케이션의 실행 시점 이전(<<before>>) 또는 이후(<<after>>)에 로그를 수집하며, 수집한 로그를 서비스 호출을 통해 백그라운드에서 실행 중인 원격 로그 서비스에 전달한다. 서비스는 이를 원격 로그 처리기에 전달하며, 원격 로그 처리기는 로그 사용에 필요한 부가정보를 추가한 후 소켓 관리자를 통해 원격 로그 서버에 전송한다. 백그라운드 실행 상태에서 전송 상황을 기록하기 위해 원격 로그 처리기는 로그 전송 상황 정보를 방송하며, 로그 상황 수신기는 해당 방송을 수신하여 로그 상황 모니터에 전달한다. 상황 정보는 원격 로그 관리자를 전면에서 실행 시 로그 상황 모니터를 통해 확인할 수 있다.

3. Remote Log Server Model

원격 로그 서버 모델은 소켓 통신 기반으로 로그 정보를 수신하는 서버 소켓 관리자와 로그를 처리하는 원격 로그 서버 관리자, 그리고 로그를 확인할 수 있는 로그 모니터로 구성한다. 서버 소켓 관리자는 복수의 모바일 기기 원격 로그 관리자와 연결할 수 있도록 구성하며, 각 원격 로그 관리자에서 전달한 로그는 로그 모니터를 통해 실시간으로 확인 가능하다. 모바일 기기의 원격 로그 관리자와 서버의 로그 모니터는 각각 소켓 통신을 사용하므로 양방향 통신이 가능하나 제안한 모델에서는 모바일 기기에서 수집한 로그를 로그 모니터에서 확인할 수 있도록 모바일 기기의 원격 로그 관리자는 로그 정보의 송신만을 수행하며, 로그 모니터는 수신하는 역할만 수행하도록 구성한다. 모바일 기기와의 연결은 복수 기기와의 연결을 수행하여야 하므로 소켓 세션의 집합으로 구성하며, 로그 모니터는 단일 클라이언트로 구성하고 필요에 따라 확장하도록 한다.

모바일 기기의 원격 로그 관리자와 원격 로그 서버의 로그 모니터는 동등한 소켓 클라이언트이므로 연결 시 구분을 위해 로그 모니터와의 연결을 선행하며, 로그 모니터와의 연결 설정 후 모바일 기기의 원격 로그 관리자와의 연결을 구성한다. 원격 로그 관리자는 하나 이상을 연결할 수 있으므로 소켓 연결 시 원격 로그 관리자가 실행 중인 모바일 기기 식별정보를 전송하며, 전송한 정보는 세션 아이디와 함께 서버에 저장하여 관리한다. Table 1은 안드로이드 운영체제의 Build를 사용하여 획득

할 수 있는 정보 중 원격 로그 관리자 식별을 위해 사용하는 정보를 나타낸다.

Table 1. Identifying information of Remote Log Manager

Fields	meaning
Build.MANUFACTURER	hardware manufacturer
Build.MODEL	hardware name
Build.DEVICE	Android code name
Build.ID	Android build no.
Build.SERIAL	hardware serial number

원격 로그 서버는 각 모바일 기기에서 전송한 로그에 대하여 세션 아이디를 확인한 후 초기 연결 설정 시에 전달받아 관리하고 있는 원격 로그 관리자 집합에서 세션 아이디에 해당하는 정보를 검색한다. 전송한 로그에 해당하는 원격 로그 관리자를 확인하면 저장 식별정보 중 모바일 기기별로 유일한 값을 제공하는 Build.SERIAL 정보를 로그에 추가하여 로그를 전송한 원격 로그 관리자 및 모바일 기기 식별에 사용한다. 로그 모니터는 로그를 전송한 대상의 식별 정보와 함께 로그를 실시간으로 확인할 수 있도록 제공한다.

IV. Evaluations

1. Case Study

본 절에서는 제안한 원격 로그 서비스 모델을 적용하여 사례 시스템을 개발하고 로그 수집 대상 어플리케이션에 적용하여 제안한 모델의 효용성을 확인한다.

원격 로그 서비스 모델의 각 구성요소는 각각 관점지향 프로그래밍, 안드로이드 서비스 컴포넌트, 그리고 웹소켓 등을 적용하여 구현하였다. 모바일 로그 관리 서비스 모델의 로그 수집기는 로깅 대상 어플리케이션에 결합하여 로그를 수집하도록 AspectJ의 이클립스 플러그인인 AJDT를 활용하여 애스펙트 모듈로 구현하였다[14].

```
public LogWebsocketManager(String url) {

    try {
        uri = new URI(url);
    } catch (URISyntaxException e) {
        e.printStackTrace();
        return;
    }

    websocket = new WebSocketClient(uri, new Draft_17()) {
        @Override
        public void onOpen(ServerHandshake serverHandshake) {
            Log.i(TAG, "Build.MANUFACTURER :" + Build.MANUFACTURER);
            websocket.send(ClientInfo.MANUFACTURER + ":" + Build.MANUFACTURER);
            Log.i(TAG, "Build.MODEL :" + Build.MODEL);
            websocket.send(ClientInfo.MODEL + ":" + Build.MODEL);
            Log.i(TAG, "Build.DEVICE :" + Build.DEVICE);
            websocket.send(ClientInfo.DEVICE + ":" + Build.DEVICE);
            Log.i(TAG, "Build.DEVICE_ID :" + Build.ID);
            websocket.send(ClientInfo.DEVICE_ID + ":" + Build.ID);
            Log.i(TAG, "Build.SERIAL :" + Build.SERIAL);
            websocket.send(ClientInfo.SERIAL + ":" + Build.SERIAL);
            Log.i(TAG, "opened");
            websocket.send("Hello from " + Build.MANUFACTURER + " " + Build.MODEL);
        }

        @Override
        public void onMessage(String message) {
            Log.i(TAG, "Received Message: " + message);
        }

        @Override
        public void onClose(int i, String s, boolean b) {
            Log.i(TAG, "Closed " + s);
        }

        @Override
        public void onError(Exception e) {
            Log.i(TAG, "Error " + e.getMessage());
        }
    };
}
```

Fig. 5. A part of the implementations of Remote Log Manager

원격 로그 관리자는 안드로이드 어플리케이션으로 구성하였으며, 세부 구성요소인 로그 상황 모니터와 원격 로그 처리기는 안드로이드 액티비티로, 원격 로그 서비스는 안드로이드 서비스를 활용하여 구현하였다. 원격 로그 관리자는 로그 수집 대상 어플리케이션과 독립적으로 실행하는 중 수집한 로그를 전달받아야 하므로 안드로이드 서비스의 원격 서비스 호출을 사용하여 구현하였다. 메시지 중심의 실시간 통신은 웹소켓이 적합하므로[15], 소켓 관리자는 웹소켓을 적용하여 구현하였다. Fig. 5는 원격 로그 관리자의 소켓 관리자 일부를 나타낸다.

원격 로그 서버는 복수의 모바일 기기 상에서 실행하는 원격 로그 관리자와의 연결을 위해 웹소켓으로 구성하였으며, 로그 모니터는 JSP를 사용하여 웹기반으로 구현하였다. 원격 로그 서버는 로그 모니터와 연결을 선행한 후 모바일 기기의 모바일 로그 관리자와 연결을 구성한다. 모바일 로그 관리자는 서버와 연결 시 모바일 로그 관리자가 식별 정보를 전송하며, 서버는 식별 정보를 저장한 후 로그 수신 시 식별 정보와 함께 로그 정보를 로그 모니터에 전달한다. Fig. 6은 원격 로그 서버 코드 일부분을 나타낸다.

```
@OnMessage
public void handleMessage(String message, Session session) throws IOException{
    String identifier = null;

    if (isServer) {
        identifier = "LOG Monitor";
        isServer = false;
    } else if (isClientConnecting) {
        String[] tmpStrings = message.split(":");
        int clientInfoType = Integer.valueOf(tmpStrings[0]).intValue();
        String clientInfoValue = tmpStrings[1];

        switch(clientInfoType) {
        case ClientInfo.MANUFACTURER:
            tmpClient.setManufacturer(clientInfoValue);
            break;
        case ClientInfo.MODEL:
            tmpClient.setModel(clientInfoValue);
            break;
        case ClientInfo.DEVICE:
            tmpClient.setDevice(clientInfoValue);
            break;
        case ClientInfo.DEVICE_ID:
            tmpClient.setDeviceId(clientInfoValue);
            break;
        case ClientInfo.SERIAL:
            tmpClient.setDeviceSerial(clientInfoValue);
            isClientConnecting = false;
            clientInfo.put(session.getId(), tmpClient);
            message = tmpClient.toString();
            identifier = "MOBILE Client";
            tmpClient = null;
            for (String key : clientInfo.keySet()) {
                log.debug(clientInfo.get(key));
            }
            break;
        }
    } else {
        identifier = clientInfo.get(session.getId()).toString();
    }
    if (tmpClient == null) {
        synchronized (serverMonitor) {
            serverMonitor.getBasicRemote().sendText(String.format("%-28s - %s", identifier, message));
        }
    }
}
```

Fig. 6. A part of implementations of Remote Log Server

제안한 원격 로그 서비스를 적용하여 위치기반 서비스 모바일 어플리케이션을 대상으로 로그 수집을 수행하였다. 로그 수집 대상 어플리케이션은 GPS로부터 현재 위치를 확인한 후 주변에 위치한 특정 관심사항의 세부정보를 Open API를 통해 조사하고 이를 지도에 표시한다. 복수의 기기에서의 실시간 로그 수집을 수행하기 위하여 두 대의 Nexus5와 한 대의 Nexus5X 기기에 로그 수집 대상 어플리케이션과 모바일 로그 관리자를 설치하였으며, Wi-Fi 무선 네트워크 환경에서 임의의 위치를 지정하기 위하여 가상 GPS 어플리케이션을 통해 위치정보를 수신하도록 구성하였다. 원격 로그 서버는 윈도우 환경에서 톰캣 서버를 사용하여 구성하였다. Fig. 7은 각각 기기에서 실행한 대상 어플리케이션을 나타내며, Fig. 8은 대상 어플리케이션으로부터 수집한 로그를 원격 로그 서버의 로그 모니터에서 확인한 결과를 나타낸다.

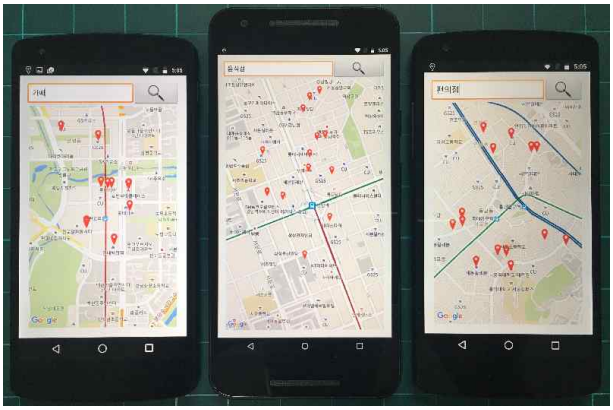

Fig. 7. executions of the target application

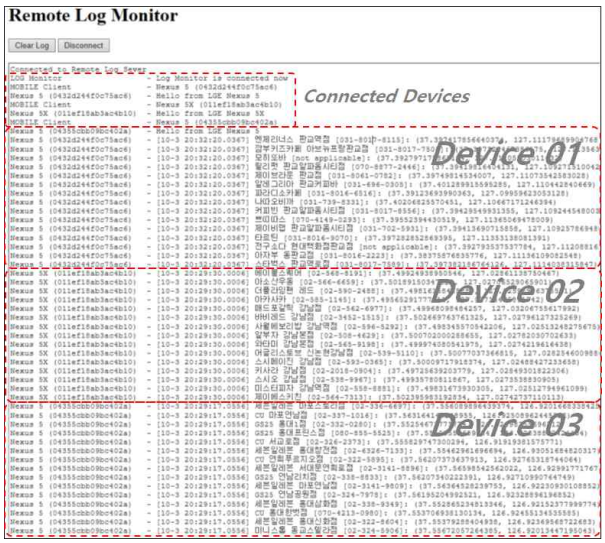

Fig. 8. The log monitor of Remote Log Server

적용사례를 통해 각 기기에서 대상 어플리케이션의 코드 및 설정 변경 없이 개발자가 지정한 로그를 수집할 수 있었고, 원격 로그 서버의 로그 모니터에서 실시간으로 확인이 가능함을 확인하였다.

대상 어플리케이션의 오류에 대한 로그 수집을 확인하기 위하여 대상 어플리케이션에 NullPointerException 오류를 삽입하여 실행하였다. Fig. 9는 오류가 발생한 대상 어플리케이션과 원격 로그 관리자에서 확인한 해당 오류 전송 상황, 그리고 이를 수신한 원격 로그 서버의 로그 모니터를 나타낸다. 로그 수집 대상 어플리케이션에서 발생한 오류의 경우에도 원격 로그 서버에서 실시간으로 확인할 수 있었으며, 이를 통해 오류에 대한 원격 분석이 가능함을 확인하였다.

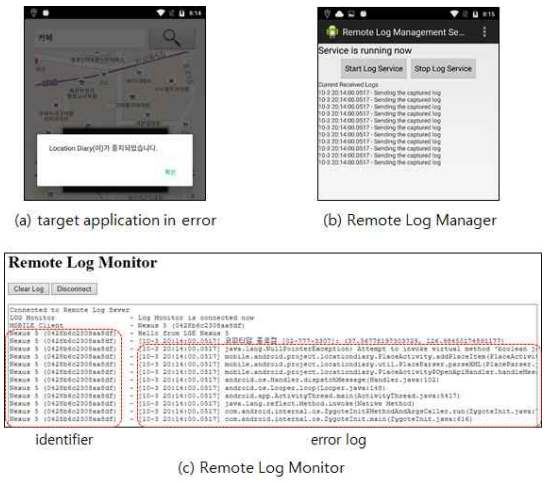

Fig. 9. Error logs processing

2. Evaluation and Comparison

2.1 Evaluation

위치기반 서비스 어플리케이션에 대한 로깅 실행 시의 로깅

실행시간 영향을 평가하기 위하여 파일로 로그를 저장하여 확인하는 방법, 로그 원격전송을 활용하는 방법, 그리고 제안한 모델을 적용한 방법을 각각 구현하여 실행시간에 대한 비교를 수행하였다. 적용사례에 사용한 어플리케이션을 로깅 대상 어플리케이션으로 사용하였으며, 로깅 항목 및 로깅 시간 측정 방법은 연구[5]와 동일한 방법을 적용하였다. 시험환경 구성은 다음과 같다. 모바일 기기는 Android 7.0 Nougat 버전의 Nexus 5X 기기를 사용하였으며, Wi-Fi 통신 상태에서 가상 GPS 정보를 기준으로 특정 위치에 대한 관심사 정보 검색을 수행하도록 구성하였다. Table 2는 각 방법 별 로깅에 따른 실행시간 측정 결과를 나타낸다.

Table 2. Execution time of logging

Type	sample	15	30	45
Logging with file	First	296 ms	470 ms	631 ms
	Second	254 ms	402 ms	713 ms
	Third	286 ms	483 ms	721 ms
	Average	**278.7 ms**	**451.7 ms**	**688.3 ms**
Remote logging in App.	First	191 ms	341 ms	411 ms
	Second	177 ms	297 ms	486 ms
	Third	179 ms	267 ms	479 ms
	Average	**182.3 ms**	**301.7 ms**	**458.7 ms**
Proposed model applied	First	105 ms	164 ms	224 ms
	Second	88 ms	155 ms	248 ms
	Third	86 ms	153 ms	231 ms
	Average	**93.0 ms**	**157.3 ms**	**234.3 ms**

샘플 개수 당 세 번의 로깅 실행시간을 측정하였으며, 측정결과 제안한 모델을 적용한 방법이 샘플 개수 및 수행 차수에 상관없이 여타의 방법에 비해 짧은 로깅 실행시간을 가짐을 확인할 수 있었다. 이는 제안한 모델이 로그 수집 단계에서 대상 어플리케이션의 실행시간에 대한 영향을 최소화 하도록 로그 수집 작업을 단순화하였고, 로그 정보에 대한 처리 및 원격 전송은 대상 어플리케이션과는 별도의 서비스에서 수행함에 따라 직접적인 로깅 수행시간을 감소시켰기 때문이다.

2.2 Comparison

적용 사례 및 평가에서 확인한 바와 같이 제안한 원격 로그 서비스 모델은 로깅 대상 어플리케이션의 변경 없이 로그 수집 및 실시간 원격 확인이 가능하였으며, 로그 수집에 따른 로깅 대상 어플리케이션의 실행시간에 대한 영향이 기존 방법에 비해 감소하였음을 확인할 수 있다. Table 3은 위치기반 서비스 모바일 어플리케이션에 적용할 수 있는 로깅 방법과 제안한 모델과의 비교를 나타낸다.

Table 3. Comparisons of log type

	Error reporting Service	Logging with file	Remote logging in App.	Proposed Model
Logging Type	remote	local	remote	remote
Writing logging code	not applicable	not easy	not easy	easy
logging information	error only	customized & error	customized & error	customized & error
Reporting Exception	easy	difficult	difficult	easy
real-time logging	partially available	not applicable	available	available
Influence of execution time	no influence	directly affect	directly affect	partially influence

실사용 환경에서 실행 중인 모바일 어플리케이션에서 로그를 수집하는 방법에는 오류 보고 서비스를 이용하는 방법과 기기에 로그 파일을 기록한 후 확인하는 방법, 원격 로깅을 대상 어플리케이션에 구현하는 방법, 그리고 본 논문에서 제안한 모델을 적용한 방법으로 구분할 수 있다. 안드로이드 배포 환경에서 제공하는 오류 보고나 fabric.io 등의 오류 보고 서비스를 이용할 경우 대상 어플리케이션의 변경 및 실행시간 영향 없이 로그를 수집할 수 있으나 실시간 확인이 아닌 보고 형태로 수집한 오류를 확인하여야 하며, 수집 정보도 오류 관련으로만 한정된다. 로그 파일을 이용할 경우에는 개발자가 지정한 로그를 확인할 수 있으나 로그 수집 후 파일을 별도로 확인하여야 하므로 실시간 확인이 어려우며, 로그 코드를 대상 어플리케이션에 포함시키므로 실행시간에 대한 영향이 발생한다. 더불어 오류 관련 로그를 기록하기 위해서는 별도의 오류 처리 코드 작성이 필요하며, 파일 처리를 위해 대상 어플리케이션의 설정 변경 등이 발생할 수 있다. 원격 로깅을 대상 어플리케이션에 직접 구현하였을 경우에는 개발자 로그와 오류 정보를 획득할 수 있고 실시간 로깅이 가능하나 파일 사용 방식과 유사하게 대상 어플리케이션의 변경 및 실행시간에 대한 영향이 발생할 수 있다. 제안한 모델을 활용할 경우 관점지향 프로그래밍을 적용하여 로그를 수집하므로 대상 어플리케이션의 변경 없이 개발자 지정 로그 및 오류 관련 로그의 자유로운 수집이 가능하며, 애스펙트로 로깅을 구현하므로 로그의 추적 및 변경이 용이하다. 더불어 로그의 원격 전송을 별도의 서비스에서 수행하여 대상 어플리케이션의 실행시간에 대한 영향을 감소시킬 수 있으며, 소켓 통신을 사용하므로 로그의 실시간 확인이 가능하다.

V. Conclusions

본 논문에서는 위치 기반 서비스를 활용하는 모바일 어플리케이션의 로깅을 위한 원격 로그 서비스 모델을 제안하였다. 제안한 모델은 관점지향 프로그래밍을 적용하여 로그 정보를 수

집하고, 백그라운드 실행 서비스에서 로그 처리 및 원격 전송을 수행하여 대상 어플리케이션의 변경 및 실행시간에 대한 영향을 최소화할 수 있었다. 또한 복수의 모바일 기기에서 수집한 개발자 지정 로그 및 오류 관련 정보를 실시간으로 원격 로그 서버에서 확인할 수 있었다. 제안한 모델을 적용할 경우 실사용 환경에서의 시험 및 다양한 안드로이드 기기를 사용한 동시 시험의 용이한 수행이 가능할 것으로 기대한다.

향후에는 대상 어플리케이션과 수집하고자 하는 로그의 유형에 따른 로그 수집기의 체계적인 설계에 대한 연구와 원격 로깅에 따른 보안 관련 연구가 필요하다. 더불어 제안한 모델을 원격 학습 및 디버깅 등 다양한 분야에 적용하는 방안에 대한 연구가 필요하다.

REFERENCES

[1] PokeMon GO, *http://pokemongo.nianticlabs.com*

[2] IT & Future Strategy, Vol.8, 12. 2014.

[3] Write and View Logs with Logcat, *https://developer.android.com/studio/debug/am-logcat.html*

[4] 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.

[5] Yun-seok Choi, "A Log Management Service Model based on AOP for Efficient Development of Android Applications", Journal of The Korea Society of Computer and Information, Vol. 21, No.3, pp.39-45, Mar. 2016.

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

[7] 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.

[8] 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.

[9] GooglePlaystore, *https://play.google.com/apps/publish*

[10] Fabric, *https://fabric.io/*

[11] 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.

[12] Minhyuk Ko, Yong-jin Seo, Bup-ki Min, Seunghak Kuk and Hyeon Soo Kim, "Extending UML Meta-model for Android Application", Computer and Information Science, 2012 IEEE/ACIS 11th International Conference on, IEEE, pp.669-674, May 2012

[13] Android Developers, *https://devloper.android.com*

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

[15] Websocket, *http://www.websocket.org/*

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 2009, 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.