Journal of The Korea Society of Computer and Information
Vol. 21 No. 2, pp. 25-32, February 2016
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.2.025

Design of a middleware for compound context-awareness on sensor-based mobile environments

Nak-Myoung Sung* and Yunseok Rhee**

Abstract

In this paper, we design a middleware for context-awareness which provides compound contexts from diverse sensors on a mobile device. Until now, most of context-aware application developers have taken responsibility for context processing from sensing data. Such application-level context processing causes heavily redundant data processing and leads to significant resource waste in energy as well as computing. In the proposed scheme, we define primitive and compound context map which consists of relavant sensors and features. Based on the context definition, each application demands a context of interest to the middleware, and thus similar context-aware applications inherently share context information and procesing within the middleware. We show that the proposed scheme significantly reduces the resource amounts of cpu, memory, and battery, and that the performance gain gets much more when multiple applications which need similar contexts are running.

▸ Keyword : context-awareness, compound context, sensor-based, mobile environment, middleware

I. Introduction

단말, 네트워크, 컴퓨팅 기술의 발전과 함께 스마트폰과 태블릿으로 대표되는 모바일 단말 기기들이 다양한 사용자층으로 확산되었다. 최근 이들 기기는 다양한 센서와 강력한 네트워킹 기능을 지원함으로써 간단한 수준의 서비스에서부터 점차 지능적이고 사용자에 특화된 서비스로 점차 영역을 확대하고 있다. 특히 다양한 센서로 부터 사용자의 행동은 물론 주변 상황정보를 분석하여 최적의 서비스를 지원하는 '상황인지(context awareness)' 기술은 사용자의 현재 위치, 시간, 주변에 있는 다른 사람이나 정보가전 기기들, 사용자의 행동 및 작업 이력 등과 같은 상황정보(contextual information)를 파악하고 분석하는 기술로서 실제로 여러 분야에서 현재 이용되고 있는 분야이다 [1-3]. 몇 가지 주요 기술로는 상황정보를 얻기 위한 센싱 기술과 상위 상황정보를 유도하고 추론하는 상황인식 추론 기술, 사용자와 애플리케이션 사이에 중간 매개체 역할을 하는 다양한 상황인지 미들웨어 기술이 제안되었다 [4-11].

상황인식 기술을 활용하여 개발된 개인 맞춤형 서비스 애플리케이션들은 각각의 애플리케이션들이 직접 모바일 시스템에서 제공하는 API를 활용하여 센서의 데이터들을 수집하고 상위 상황정보를 유도하고 추론하기 위한 상황인식 추론 알고리즘을 수행하는 구조를 가지고 있다.

그러나 위와 같은 애플리케이션 구조는 다음과 같은 문제를 가진다. 첫째, 동시에 동작하는 상황인식 애플리케이션의 개수가 증가할수록 각각의 애플리케이션 별로 상황정보를 얻기 위해 관련 API 호출 횟수가 점차 증가하고 같은 센서에 동시에 여러 개의 애플리케이션이 접근하는 문제가 발생할 수 있다. 둘째, 각각의 애플리케이션 별로 상위 상황정보를 유도하고 추론하기 위한 알고리즘을 수행해야 한다. 셋째, 앞에서 말한 센서 API 호출 횟수가 증가하고 각 애플리케이션이 별도로 자신만의 알고리즘을 수행함으로써 생기는 연산량 증가로 인해 배터리를 포함한 자원의 낭비

• First Author: Nak-Myoung Sung, Corresponding Author: Yunseok Rhee
*Nak-Myoung Sung(diesnm201@gmail.com), Korea Electronics Technology Institute
**Yunseok Rhee(rheeys@hufs.ac.kr), School of Computer and Electronic Systems Engineering, Hankuk Univ. of Foreign Studies
• Received: 2016. 01. 22, Revised: 2016. 01. 29, Accepted: 2016. 02. 15.
• This work was supported by Hankuk University of Foreign Studies Research Fund of 2013.

와 시스템 전체의 성능이 저하될 수 있다. 마지막으로 각 애플리케이션들을 개발하는 개발자들이 상황인식을 위한 알고리즘을 설계하기 때문에 개발 편의성이 떨어지게 된다.

이러한 문제들을 해결하기 위해 개별 애플리케이션 수준이 아닌 미들웨어에서 직접 기본적인 상황정보를 수집, 가공하고 상위 상황정보를 추론하여 애플리케이션으로 전달하고, 각 애플리케이션들 간에 복합적인 상황인지 정보를 공유하는 미들웨어 연구가 필요할 것으로 판단된다.

본 논문의 II 장에서는 상황인식 시스템의 관련 연구들을 소개하고, III 장에서는 본 연구에서 제안하는 기본적인 상황인식 정보와 이들을 결합한 복합 상황인식 정보의 정의 및 표현방법을 기술한다. 또한, 이를 지원하는 미들웨어의 기능과 구성, 제공 API를 소개한다. IV 장에서는 안드로이드 플랫폼 상에서 가속도 센서를 사용한 실험용 앱을 제작, 기존 방식과 성능을 비교 분석하고, V 장에서 결론을 맺는다.

II. Related Works

대부분의 상황 인지 관련 연구는 상황정보 모델링, 상황추론 엔진, 저장 및 서비스 제공 플랫폼 설계에 중점을 두어왔다[4-8]. 이들은 대부분 실제 센서 데이터를 수집하여 상황인식을 위한 서버로 전달하고 서버 내의 미들웨어에서 이미 정의된 상황 모델을 기반으로 상황 정보를 추론하고 그 결과를 다시 클라이언트로 제공하는 방식으로 설계되어 있다. 이와 같은 시스템들은 네트워크 부하, 모바일 단말기의 성능 제약, 소비 전력의 문제 등으로 현재의 모바일 시스템에 직접 적용하기에는 한계를 갖는다. 또한 센서 데이터 스트림으로부터 지속적으로 변화하는 상황을 처리하기에는 효율성 또한 낮은 문제를 야기하며, 이로 인해 최근에는 상황인식 서버-클라이언트 구조가 아닌 모바일 단말 미들웨어 방식에 대한 연구들이 다음과 같이 진행되고 있다.

MyHealthAssistant[9], Contory[10], ManySense[11] 등은 모바일 단말기 기반의 미들웨어를 제안하고 있다. 특히 MyHealthAssistant는 의료 애플리케이션을 지원할 목적으로, 각 애플리케이션이 필요로 하는 센서들의 정보를 직접 제공하는 Sensor Module과 전달 Channel을 설정하고, 이를 통해 각각의 애플리케이션이 원하는 형식의 원시 데이터(또는 일부 가공)를 전달한다. Event- Driven 방식으로 센서 데이터를 수집하는 Sensor Module Manager와 수집된 센서 데이터를 원하는 Application으로 전달하기 위한 Message Handler, 미들웨어에 대한 설정 및 모니터링, 보안을 각각 담당하는 Event Composer, System Monitor, Security Manager 등으로 구성되어 있다. Message Handler는 Sensor Module Manager로부터 센서 데이터를 수신하면 해당 센서 데이터를 필요로 하는 모든 애플리케이션으로 전달하는 일종의 Sensor Broker Model을 지원한다. 특히 이 모델은 안드로이드 플랫폼에서 제공하는 Broadcast Intents와 BroadcastReceivers를 활용하여 동시 Broadcasting함으로써 각 애플리케이션 별로 데이터를 전달하는 데이터 전송 overhead를 상당히 줄일 수 있다. 그러나, 이 방식은 센서 데이터 수준에서 애플리케이션들 간에 효과적인 공유는 가능하지만, 상위 상황정보를 추론하기 위한 상황인식 알고리즘은 개별 애플리케이션에서 처리되기 때문에 각 애플리케이션 수준에서 각각 상황인식 알고리즘을 탑재, 실행하게 되는 비효율성이 존재한다.

Contory[10] 역시 스마트폰 상의 미들웨어로서 다양한 context에 대한 처리까지 제공하기 위한 목적으로 설계되었다. BT, WiFi, 2G/3G 등 다양한 인터페이스 도메인별 센서 데이터를 수집할 수 있도록 Resources Monitor가 구성되어 있으며 실제 Context provisioning 및 reasoning을 수행하는 모듈을 통해 각 애플리케이션이 context를 요청할 수 있도록 query문을 정의하고 있으나 "AVG (temperature) > 25" (평균 온도 값이 25도 이상이 경우) 알림을 달라는 간단한 값 매핑 수준의 context만을 지원하고 상위 수준 및 복합 상황에 대한 처리를 지원하지 못한다. 한편, ManySense[11]는 상위 수준의 상황인식에 대한 처리를 위해 Adapter SDK를 제공함으로써 실제 애플리케이션이 필요한 상황인식 알고리즘을 개발자에게 구현하도록 하였으며 이는 상위 수준의 상황인식에 대한 처리는 가능하나 개발자의 개발편의성 및 현재 제공하는 추가 개발된 상황인식 알고리즘에 대한 정보를 애플리케이션이 사전에 아는 경우에만 활용 가능한 문제가 있다.

본 논문에서는 애플리케이션이 원하는 복합 상황을 등록할 수 있도록 인터페이스를 제공하고 각 상황을 도출하기 위한 상황정의맵(Context Definition Map) 개념을 정의하고, 이를 활용하여 자동적으로 상황 도출에 필요한 센서를 선택하여 데이터를 수집하고 수집된 데이터에 대한 상황인식 알고리즘 수행을 통해 애플리케이션으로 복합 상황정보를 제공할 수 있는 미들웨어를 제안한다.

III. The Proposed Scheme

센서 기반 복합 상황인식이란 그림 1과 같이 현실에서 실제 사용자 주변에 있는 모바일 단말에서 지원하는 센서를 포함한 여러 가지 종류의 이기종 센서들로부터 수집된 온도, 습도, 조도 등의 환경 정보와 심박수, 혈압 등의 신체 정보 및 가속도, 자이로 센서 등의 움직임 정보와 GPS 등의 위치 정보를 포함하는 모든 상황 정보를 토대로 각각의 기본이 되는, 예를 들어 따뜻한 날씨, 빠른 맥박, 빠른 움직임, 실외 등의 간단한 상황(Primitive Context)을 도출하고 이러한 Primitive Context들을 조합하여 복합 상황(Compound Context)을 추론하는 방식을 말한다.

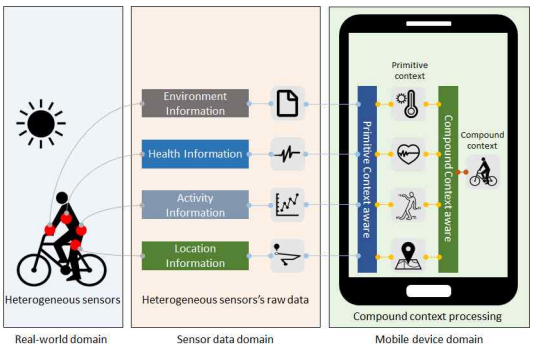


Fig. 1. Sensor-based Compound Context-awareness

3.1 상황 정의 맵

센서 기반 복합 상황인식을 위해서는 각 센서들과 센서들로부터 수집할 수 있는 상황 정보 상황 정보들로부터 추론할 수 있는 Primitive Context, Primitive Context들을 조합하여 추론할 수 있는 Compound Context들의 관계가 정의되어야 한다.

이러한 센서, 상황 정보, Primitive Context, Compound Context에 관한 관계를 그림 2와 같이 표현할 수 있다. 이 관계도에서 S(Sensor)는 상황 정보를 수집할 수 있는 센서를 의미한다.

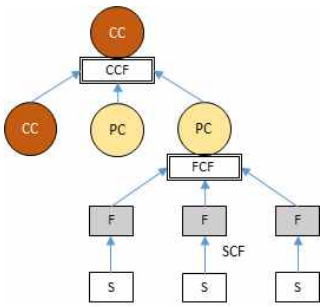


Fig. 2. Context Relation Diagram

SCF(Sensor Calculate Function)는 센서로부터 수집된 Raw 데이터를 상황 정보로 변환하기 위한 Function으로써 FFT 등의 시그널 프로세싱이나 Minimum, Maximum, Average 계산을 통한 실제 Digital value, continuous value, time series value를 추출하는 동작을 수행한다. F(Feature)는 SCF를 통해 추출된 상황 정보를 의미한다. SCF 함수는 단순한 조건 검색으로 부터 복잡한 신호처리 함수에 이르기까지 다양한 형태를 지원할 수 있으며, 본 미들웨어는 콜백(call-back) 방식을 통해 이들을 지원한다.

이와 같이 임의의 F를 추출하기 위한 SCF와 S를 연결하여

$$f_i = \{S_i , SCF_i \} \quad (1)$$

과 같이 표현할 수 있고 이를 Feature에 대한 상황 정의 맵이라 한다. 예를 들어 Activity 상황 정보 activity를 추출하기 위한 가속도 센서 Saccelerometer와 SCFaccelerometer를 식 (2)와 같이 표현할 수 있다.

$$activity = \{S_{accelerometer} , SCF_{accelerometer}\} \quad (2)$$

상황 정보가 추출되면 특정 상황 정보 F들을 조합하여 FCF(Feature Combination Function)을 통해 하나의 PC(Primitive Context)를 추출할 수 있다. 여기서 FCF는 F들 간의 unions, intersects, condition mapping 등의 동작을 수행하는 Function을 의미한다. 이렇게 임의의 PC를 추출하기 위한 FCF와 F들을 연결하여

$$PC_i = \{<f>_{required} , FCF_i \} \quad (3)$$

과 같이 표현할 수 있고 이를 Primitive Context로 정의한다. 예를 들어 fast movement 라는 Primitive Context PCfast movement를 추출하기 위한 상황 정보 activity와 FCFfast movement를 식 (4)와 같이 표현할 수 있다.

$$PC_{fast\ movement} = \{activity , FCF_{fast\ movement}\} \quad (4)$$

PC가 추출되면 이들을 조합하여 CCF(Context Combination Function)을 통해 하나의 CC(Compound Context)를 추출할 수 있다. 여기서 CCF는 PC들 간의 혹은 PC와 또 다른 CC간의 unions, intersects, condition mapping 등의 동작을 수행하는 Function을 의미한다. 이렇게 임의의 CC를 추출하기 위한 CCF와 PC 혹은 CC들을 연결하여

$$CC_i = \{<CC>_{required} , <PC>_{required} , CCF_i\} \quad (5)$$

과 같이 표현할 수 있고 이를 Compound Context에 대한 상황 정의 맵 이라 정의한다. 예를 들어 Jogging 이라는 Compound Context $CC_{jogging}$를 추출하기 위한 Primitive Context $PC_{fast\ movement}$, $PC_{use\ arm\ band}$와 $CCF_{jogging}$을 식 (6)과 같이 표현할 수 있다.

$$CC_{jogging} = \{PC_{fast\ movement} , PC_{use\ arm\ band} , CCF_{jogging}\} \quad (6)$$

위와 같이 Feature, Primitive Context, Compound Context에 대한 상황 정의 맵을 식 (1), (3), (5)와 같이 정의하였고 이를 표 1에 정리하였다. 상황 정의 맵을 통해 미들웨어는 애플리케이션이 복합 상황 CC에 대한 알림을 요청했을 시 해당 복합 상황 CC를 추론하기 위해 필요한 기본 상황 PC들을 파악할 수 있고, 각각의 기본 상황 PC들을 판단하기 위한 상황 정보 F들, 상황 정보 F들을 추출하기 위한 센서 S들을 찾을 수 있다.

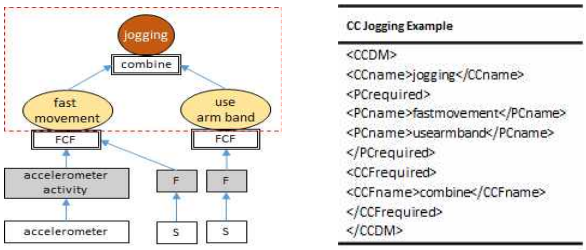
3.2 상황 정의 맵 표현

이와 같이 상황 정보 정의를 통해 도출된 상황 정의 맵을 실제로 미들웨어 및 애플리케이션이 활용하기 위해서는 특정한 형태로 표현되어야 한다. 계층적인 구조로 표현하고 인식 및 해석이 용이한 XML (Extensible Markup Language)를 활용하여 상황 정의 맵을 표현하도록 하며 그림 3과 같이 식 (6) Jogging이라는 CC의 상황 정의 관계도 (a)를 (b)와 같은 XML 형태로 표현할 수 있다.

Table 1. Context Definition Map

Category	Definition Map
Feature	*fi = {Si , SCFi}*
Primitive Context	*PCi = {<f>required , FCFi}*
Compound Context	*CCi = {<CC>required , <PC>required , CCFi}*

** < A >required : single or multiple required set of A*



(a) Context Relation Diagram (b) Context Representation

Fig. 3. Context Relation Diagram and Representation

3.3 Middleware API 및 구성

복합 상황인식 미들웨어는 애플리케이션을 위해 총 5개의 API를 제공한다. API는 표 3과 같이 정의되며 Compound Context 관련 API 2개와 Definition Map 관련 API 3개로 구성되어 있다. 각 API들은 기본적으로 요청을 위한 파라미터들을 가질 수 있으며 XML형태로 Query문을 구성할 수 있다. 리턴 형식 또한 3.2절의 상황 정의 맵 표현의 그림 3(b)와 같은 형태를 취하게 되어 애플리케이션 개발자들의 API 요청 구성 및 리턴에 대한 처리 구현이 편리하도록 구성된다.

앞서 정의한 상황 정의 맵(Context Definition Map)과 이를 활용한 애플리케이션 제공 API를 지원하기 위해 그림 4와 같은 상세 구조를 설계하였다. Middleware Coordinator에서는 애플리케이션으로 API를 제공하기 위한 Application Interface 모듈과 애플리케이션의 요청을 판단하기 위한 Application Request Handler 모듈, 애플리케이션이 원하는 복합 상황에 대한 알람을 위한 Context Broker 모듈을 가진다. Context Manager에는 실제 애플리케이션의 상황 요청을 처리하기 위한 Context Request Handler 모듈과 현재 등록된 상황 요청들을 모니터링하기 위한 Context Query Monitor 모듈, Sensor Manager로 사용하고자 하는 Sensor 리스트를 전송하는 Used Sensor Updater 모듈로 구성된다.

Context Instance Manager는 현재 등록된 복합 상황 요청(CC)에 대한 리스트를 관리하기 위한 CC query list 및 CC를 판단하기 위해 필요한 간단한 상황(PC)에 대한 리스트를 관리하기 위한 PC list, F의 리스트를 관리하기 위한 Feature list를 가지고 실제 데이터를 저장하는 구조를 가진다. Context Processor는 Sensor Manager로부터 상황 정보를 수신하기 위한 Sensor Data Receiver 모듈과 상황 정보를 통해 Context Instance Manager의 list들을 확인하여 복합 상황 판단 알고리즘을 수행하는 Context Processing Unit 모듈, 복합 상황 전달을 위한 Context Provider 모듈로 구성된다. 마지막으로 Sensor Manager는 Context Manager로부터 수신된 사용 센서 리스트를 활용하여 센서와 연결하는 Used Sensor Connection Manager와 실제 센서 데이터를 수집하는 Sensor Data Collector, 센서 데이터를 Context Processor로 전송하는 Sensor Data Provider로 구성된다.

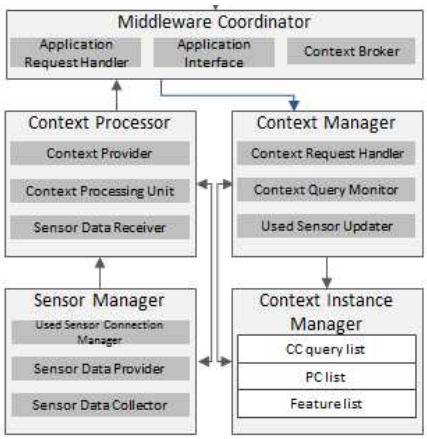


Fig. 4. Compound Context-awareness Middleware Architecture

IV. Experiments and Evaluation

본 장에서는 실제 구글의 Android OS를 활용한 스마트폰 단말 환경에서 복합 상황인식 미들웨어와 구현하고 복합 상황에 대한 알림을 등록하는 애플리케이션을 구현하여 기존 논문에서 제시한 미들웨어와 비교 분석을 통한 성능 평가를 수행하였다.

4.1 구현 환경

구현 하드웨어 환경은 표 2와 같이 Qualcomm사의 Snapdragon 800 Processor를 활용하고 2GB의 Memory를 탑재하고 있는 Google의 레퍼런스 스마트폰 Nexus5를 사용하였다. 소프트웨어 환경은 Nexus5에서 구동 가능한 현재 가장 최신 버전의 Android 5.0 Lollipop 운영체제를 사용하였고 Android Linux Vanilla kernel 3.4.0-g88fbc66 version에서 구현하였다.

Table 2. Experiment H/W specification

Model	Processor	Memory	Onboard sensor
Nexus 5	Qualcomm Snapdragon 800 MSM8974 (Quad core, 2260MHz, Krait 400)	2GB of LPDDR3-1600 RAM	Accelerometer, Magnetometer, Gyroscope, Proximity, Ambient light, Barometer, Gravity, GPS, … etc.

4.2 미들웨어 구현

기본적으로 Android에서 제공하는 background remote service로 미들웨어의 동작 환경을 구현하였다. Android의 background remote service는 운영체제 내에서 background로 동작하며 다른 service나 애플리케이션 들이 언제나 원하는 시점에 onBind를 통해 ServiceConnection을 수행하여 통신을 할 수 있도록 지원한다. 미들웨어의 총 5개의 컴포넌트는 Android에서 지원하는 AsyncTask로 동작하며 MessageQueue를 활용한 MessageHandling 방식으로 통신하도록 구현하였다.

상황정의맵(Context Definition Map)은 각각의 CCDM, PCDM, FDM에 해당하는 Data Class를 정의하였으며 CC query list, PC list, Feature list는 Object Type의 ArrayList를 활용하였다. CC query list는 CCDM Data Class를 하나의 인자로 갖는 ArrayList로, PC list는 PCDM Data Class를 하나의 인자로 갖는 ArrayList, Feature list는 FDM Data Class를 하나의 인자로 갖는 ArrayList로 구성되어 순차 검색 및 인덱스 직접 접근을 통한 각각의 내부 값을 읽어오거나 업데이트를 쉽게 할 수 있도록 구현하였다.

센서는 기본적으로 스마트폰에서 제공하는 센서에 대한 프로파일을 정의하고 각 센서를 사용하기 위한 요청이 들어올 경우 Android에서 제공하는 센서 관련 API를 호출하여 데이터를 수집하는 방식을 취했다.

Table 3. Compound Context-awareness Middleware API

Category	APIs	Parameter	Description
Compound Context	CCregistration	CC_{name}, Duration	API for an application to register a compound context of interest
	CCderegistration	CC_{name}	API for an application to deregister a compound context which it registered
Definition Map	DMlistget	Keyword	API to request a context definition map list provided by the middleware
	DMregistration	CC_{name}, $<CC>_{required}$, $<PC>_{required}$, CCF_{name}	API to register a new context definition map to the middleware
	DMderegistration	CC_{name}	API to deregister a context definition map from the middleware

애플리케이션 인터페이스 제공을 위해 IBinder를 통해 표 3에 정의된 5가지 인터페이스를 호출할 수 있도록 Method 형태로 구현하였으며 각 Method는 XML형태의 String을 인자로 가진다. 실제 onBind를 통한 ServiceConnection 시 onServiceConnected 상황에서 IBinder의 Method를 호출할 수 있다.

복합 상황 알림에 대한 처리는 Android에서 제공하는 broadcast Intent 및 Broadcast Receiver를 활용하였으며 실제 애플리케이션이 요구한 복합 상황에 해당될 때 해당 상황에 대한 Broadcast 채널로 한번에 Broadcasting 하는 방식으로 구현하였으며 이 방식은 동일한 복합 상황에 대한 알림을 여러 개의 애플리케이션으로 보낼 때 각각의 애플리케이션과 통신을 통해 여러 번의 과정을 거치지 않고 한번의 Broadcasting으로 처리할 수 있는 장점이 있다.

4.3 실험 벤치마크

이전의 상황인식 미들웨어들도 대부분 같은 상황의 공유에 대한 중요성을 언급하고 있다. 그 중 MyhealthAssistant[8]는 헬스케어를 위한 정보를 수집할 때 동일한 센서의 데이터를 여러 개의 애플리케이션이 공유하는 경우의 예를 들었으며 CoBrA[3], Gaia[4] 등의 시스템은 동일한 종류의 상황정보를 공유하는 여러 애플리케이션을 지원할 수 있도록 되어있다.

애플리케이션의 종류와 성격에 따라 원하는 상황정보는 다양할 수 있지만, 애플리케이션들이 동일한 종류의 상황정보를 공유하는 경우도 상당히 많이 발생한다. 예를 들면 운동과 관련된 애플리케이션으로 사용자가 jogging시 얼마나 많은 운동량이 발생하는지 측정하고자 하는 애플리케이션이 있을 수 있고, 헬스케어와 관련된 애플리케이션으로 사용자의 응급 상황을 판단하기 위해 사용자의 현재 상황이 필요한 애플리케이션의 경우도 똑같이 jogging 상태인지 수면 상태인지 현재 상황에 따라 다른 판단을 내릴 수 있기 때문에 상황을 확인해야만 정확한 판단을 내릴 수 있을 것이다. 또한 최근 관심이 증가하고 있는 도보 및 운동 상황을 지원하는 네비게이션 서비스 애플리케이션의 경우 jogging 이나 ride a bicycle 등의 상황정보를 통해 목적지까지의 이동경로를 지능적으로 판단하여 서비스를 할 수 있을 것이다. 위의 예와 같이 하나의 모바일 시스템 내에서 동일한 상황정보를 필요로 하는 다양한 애플리케이션이 존재할 수 있으며 그러한 경우가 빈번하게 발생할 수 있다.

본 논문에서는 이와 같이 동일한 상황정보가 공유되는 상황에서 미들웨어가 어느 정도의 성능을 보이는지 성능 측정을 위한 벤치마크를 구성하기 위해 미들웨어와 연동되는 애플리케이션을 구성하도록 했다.

애플리케이션의 기본 동작은 기본적으로 Android에서 제공하는 background activity를 활용하여 구현하였으며 일반 애플리케이션 방식으로 구현 시 홈 버튼을 통한 activity 전환 시 Android 운영체제에 의해 강제적으로 종료되는 문제를 해결하기 위해 종료 시 재실행이 되도록 했다. jogging이라는 복합 상황에 대한 알림을 요청하도록 구현하였으며 실제 jogging에 대한 알림이 올 경우 다른 동작을 수행하지는 않으며 단순히 로그만 기록하도록 했으며 미들웨어가 제공하는 IBinder의 인터

페이스를 활용하기 위해 미들웨어 remote service로 onBind를 통해 serviceConnection을 수행하도록 했고 onService-Connected 시 미들웨어가 제공하는 5개의 API 중 복합 상황 알림 등록에 대한 요청을 수행하도록 하였다. 앞서 말한 것처럼 jogging이라는 요청을 등록 API에 정의된 XML형태로 구성하여 보내도록 하였다. 복합 상황 알림을 수신하기 위해서 미들웨어와 동일하게 Android에서 제공하는 broadcast Intent 및 Broadcast- Receiver를 활용하여 구현하였으며 복합 상황과 같은 이름의 broadcast 채널을 활용하여 수신 대기하도록 하였다. 실제로 미들웨어에서 복합 상황에 대한 알림을 보내게 되면 BroadcastReceiver가 요청을 수신하고 해당 로그를 기록하도록 구현하였다.

4.4 성능 평가

모바일 환경에서의 복합 상황인식 미들웨어의 성능을 평가하기 위해서는 미들웨어가 사용하는 자원을 모니터링 해야 한다. 실제 애플리케이션이 미들웨어로 복합 상황에 대한 알림 요청을 할 때 미들웨어에서 사용하는 자원의 변화를 측정할 필요가 있으며 복합 상황에 대한 알림 요청을 등록한 애플리케이션이 늘어날 때의 측정 또한 필요하다.

자원 측정 항목은 각각 CPU 사용량, Memory 사용량, Battery 소모량에 대해 측정하도록 한다.

CPU 사용량의 경우 식 (7) 과 같이 모바일 시스템에서 수행된 전체 CPU 시간(CPU_{total}) 대비 대상 미들웨어와 애플리케이션이 활용하는 자원에 대해 % 단위로 환산한다.

$$CPU_{usage} = (CPU_{middleware} + CPU_{applications}) / CPU_{total} \quad (7)$$

Memory 사용량의 경우 식 (8) 와 같이 현재 사용 중인 Memory의 절대 값을 표현하며 단위는 MB(Mega Byte)로 환산한다.

$$Memory_{usage} = Memory_{middleware} + Memory_{applications} \quad (8)$$

Battery 소모량의 경우 식 (9) 와 같이 미들웨어와 애플리케이션 Battery 소모량을 전체 Battery 량 대비 비율(%)로 환산하며, CPU 시간을 참고하여, 단위 시간당 Battery 소비량을 다음과 같이 계산한다.

$$Battery_{usage/H} = (Battery_{middleware} + Battery_{applications}) * 1\ hour / (CPU_{middleware} + CPU_{applications}) \quad (9)$$

위의 측정 지표를 바탕으로 기존에 제안된 미들웨어 형태와 본 연구에서 제안한 복합 상황인식 미들웨어를 안드로이드 시스템에 구현하고, 이들을 활용하는 시뮬레이터 앱을 통해 성능을 측정 비교 분석하였다. 특히, 제안 시스템과 비교할 대상 시스템은 비교적 센서 데이터의 공유도가 높은 Sensor Broker 모델 (MyHealthAssistant 채택 모델) [8]로 선정했다. 따라서, Sensor Broker 모델과 본 연구에서 제안하는 Compound Context-aware (CCAM) 모델을 대상으로 상황인지 애플리케이션의 개수가 늘어남에 따라 CPU / Memory / Battery 사용량의 변화 추이를 비교하여 전체적인 성능을 측정했다. 당연히, 두 모델 간 형평성을 맞추고자 동일한 센서들을 활용하고 동일한 복합 상황을 처리한다. 또한, 애플리케이션은 가속도 센서를 사용하여 주기적으로 사용자의 동작 상황을 인지하며, 실제 소요 자원의 확장성에 대한 비교 판단을 명확히 하기 위해 이와 같은 유사 애플리케이션들이 복수 동작한다고 가정하였는데, 이는 현실 의료 및 스포츠, 감성모드, 게임 등에서 유사한 동작 상황인식이 반복적으로 수행될 것으로 예측된 상황이다.

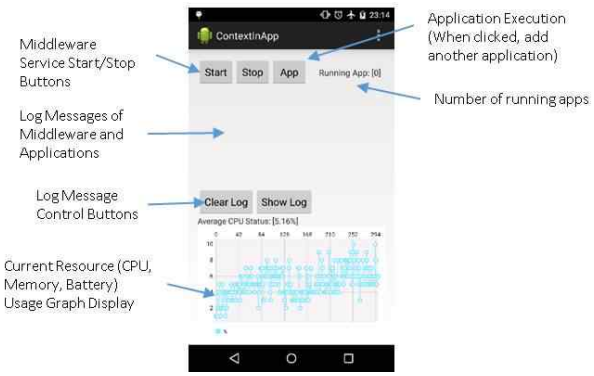


Fig. 5. Simulation App UI Layout

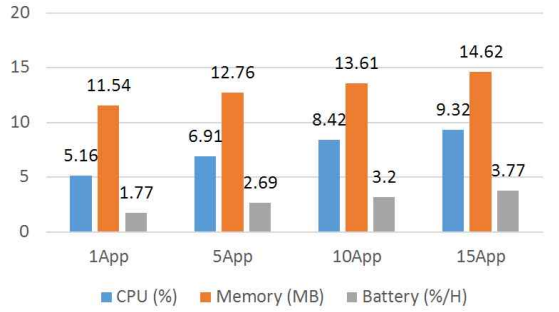
성능평가 시뮬레이터 앱은 Android 애플리케이션으로 구현하였으며 그림 5에서 보이는 앱 UI를 사용하여 미들웨어를 실행하고 중지, 애플리케이션 실행 및 추가, 미들웨어와 애플리케이션의 CPU / Memory / Battery 사용량 정보 모니터링, 그리고 측정 결과를 그래프로 출력하는 기능을 제공한다.

시뮬레이터 상에서 Start 버튼을 활용하여 미들웨어를 실행할 수 있으며 미들웨어가 실행중인 상황에서 App 버튼을 눌러 복합 상황 알림을 요청하는 애플리케이션을 실행하고 추가시킬 수 있다. 현재 실행중인 애플리케이션의 개수를 표시하고 미들웨어 및 애플리케이션 Log메시지를 표시하며 그래프를 드래그하여 CPU 이외에도 사용중인 Memory와 Battery 정보를 확인할 수 있다.

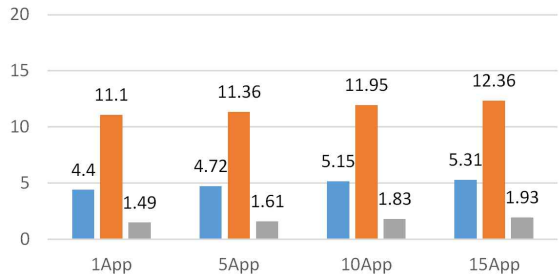
Fig. 6. Experiment Device and Environment

성능 측정을 위해 그림 6과 같이 Google Nexus5 레퍼런스 스마트폰을 암 밴드에 고정시킨 채로 jogging에 해당하는 행동을 반복 실험했다. 성능평가 시뮬레이터를 활용하여 Sensor Broker 모델과 Compound Context-aware Middleware(CCAM) 모델의 두 가지 모델 별 Application을 각

1개, 5개, 10개, 15개 씩 5분 간 동작시켜 총 Sensor Broker 모델 4가지, CCAM 모델 4가지 결과를 도출하도록 실험을 수행했다. 각각 애플리케이션이 1개, 5개, 10개, 15개일 때 Sensor Broker 모델의 성능 측정 결과는 그림 7(a)에 보이는 바와 같이 측정되었다. 1개의 애플리케이션이 동작할 때 미들웨어와 애플리케이션이 총 사용하는 리소스는 CPU 5.16%, Memory 11.54MB, Battery 1.77%/H로 측정되었지만, 애플리케이션의 개수가 5개, 10개, 15개로 늘어날수록 그림 13과 같이 리소스 사용량이 상당히 증가하는 것을 볼 수 있다.



(a) Sensor Broker Model



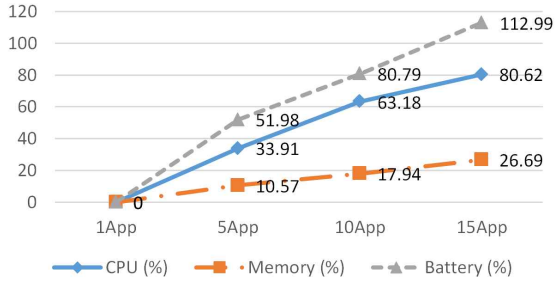
(b) CCAM Model

Fig. 7. Resource Usage with Multiple Apps

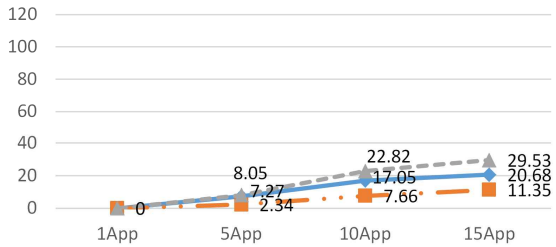
결과적으로 Sensor Broker 모델은 애플리케이션 개수가 늘어날수록 리소스 사용량이 상당히 증가하는 것을 볼 수 있으며 이는 Sensor Broker 특성 상 수집한 센싱 데이터에 대한 상황인식 알고리즘을 애플리케이션에서 직접 수행함으로써 애플리케이션 개수가 늘어남에 따라 동일한 연산이 중복되기 때문에 발생하는 결과라 볼 수 있다.

또한 Broadcasting 방식을 취하기는 하지만 매번 센싱 데이터가 수집될 때마다 애플리케이션으로 Broadcasting이 이루어지기 때문에 데이터 교환에 발생되는 리소스 또한 꾸준히 사용될 수밖에 없다.

각각 애플리케이션이 1개, 5개, 10개, 15개일 때 Compound Context-aware Middleware(CCAM) 모델의 성능 측정 결과는 그림 7(b)와 같이 측정되었다. 1개의 애플리케이션이 동작할 때 미들웨어와 애플리케이션이 총 사용하는 리소스는 CPU 4.4%, Memory 11.1MB, Battery 1.49%/H로 측정되어, 단일 애플리케이션의 자원 사용량은 비슷했지만, 애플리케이션의 개수가 5개, 10개, 15개로 늘어날수록 리소스 사용량이 소폭으로 증가하는 것을 볼 수 있었다. 이는 CCAM 특성 상 수집한 센싱 데이터에 대한 상황인식 알고리즘을 미들웨어가 직접 수행함으로써 애플리케이션 개수가 늘어나도 동일한 연산이 중복되지 않고 한번만 연산을 수행하기 때문에 발생하는 결과라 볼 수 있다. 또한 Sensor Broker 모델과는 다르게 동일한 형태의 Broadcasting 방식을 취하기는 하지만 복합 상황이 발생하지 않을 시에는 애플리케이션으로 Broadcasting이 이루어지기 않기 때문에 데이터 교환에 발생되는 리소스 또한 적게 사용된다고 분석된다.



(a) Sensor Broker Model



(b) CCAM Model

Fig. 8. Resource Scalability with Multiple Apps

그림 8은 동시 실행 애플리케이션의 증가에 따라 두 비교 모델의 자원 사용량의 변화를 보여준다. 즉, 단일 애플리케이션의 자원 사용량에 비하여 증가한 증가분을 백분위 비율(%)로 나타낸 것이다. 이 그래프를 통해, Sensor Broker 모델은 동시 실행 애플리케이션의 수에 선형적으로 자원 사용량이 증가하는 반면, CCAM 모델은 애플리케이션의 증가에 따른 변화가 매우 완만하므로, 더 많은 상황인지 애플리케이션의 동시 실행 환경에서도 우수한 확장성을 보일 것으로 기대된다.

V. Conclusions

본 논문에서는 애플리케이션 별 빈번한 센서 API를 호출하고 복합 상황인식 알고리즘을 수행함으로 인해 발생하는 연산량 증가 문제를 해결하기 위한 방안으로 미들웨어에서 대신 센서 데이터를 수집하고 복합 상황인식 알고리즘을 수행하는 방식을 설계하였다. 또한 애플리케이션 별 상황인식 알고리즘을 직접 개발자가 구현하는데 발생하는 개발 편의성 하락 문제를

해결하기 위해 미들웨어로 복합 상황 정보를 요청하는 API를 정의하였으며 상황정의맵(Context Definition Map)을 제안하여 복합 상황(Compound Context), 간단한 상황(Primitive Context), 상황 정보(Feature) 및 센서(Sensor) 까지의 관계도를 표현할 수 있도록 하여 개발자가 이를 활용해서 원하는 복합 상황을 추가로 요청하거나 새롭게 조합하여 등록할 수 있도록 지원할 수 있는 방안을 제시하였다. 또한 본 연구에서 제안된 미들웨어와 상황인지 시뮬레이션 앱을 실제 안드로이드 기반 모바일 단말기에 구현하고, 유사 기법과 이의 성능을 비교하였다. 기존에 연구된 상황인지 미들웨어들과 달리 미들웨어 수준에서 복합 상황인지 처리를 실행하고, 이를 다수의 유사 애플리케이션들에게 제공함으로써 컴퓨팅 자원의 효율성을 제고하였다. 특히, 제안된 기법은 모바일 기기의 배터리 성능을 효과적으로 개선하는 데 기여할 것으로 기대된다.

REFERENCE

[1] Korea Communication Agency, “Context Awareness Technology, Applications, and Future Trends", Broadcasting and Communication Technology Issues and Prospects, No. 7, pp. 1-10, Dec. 2013.

[2] M. Balduf, S. Dustdar, and F. Rosenberg, “A Survey of Context-aware Systems”, International Journal of Ad Hoc and Ubiquitous Computing, Vol. 2, Issue 4, pp. 263-277, June 2007.

[3] H. Chen, "An Intelligent Broker Architecture for Pervasive Context-Aware Systems", Doctorial Thesis of the University of Maryland, Baltimore County, Dec. 2004.

[4] S. Chetan, J. Al-Muhtadi, R. Campbell and M. Mickunas, "Mobile Gaia: A Middleware for Ad-hoc Pervasive Computing", IEEE Consumer Communications and Networking Conference, pp. 223-228, Jan. 2005.

[5] Hwa Young Jung, Jae Wook Park, Yong Kyu Lee, "A Context-Aware Treatment Guided System", Journal of The Korea Society of Computer and Information, Vol. 19, No. 1, pp. 141-148, Jan. 2014.

[6] Tao Gu, Xiao Hang Wang, Hung Keng Pung and Da Qing Zhang, "An Ontology-based Context Model in Intelligent Environments", Communication Networks and Distributed Systems Modeling and Simulation Conference, pp. 270-275, Jan. 2004.

[7] H. Kim, Y. Cho and S. Oh, "CAMUS: A middleware supporting context-aware services for network-based robots", IEEE Workshop on Advanced Robotics and Its Social Impacts, pp. 237-242, June 2005.

[8] Abhay Daftari, Nehal Mehta, Shubhanan Bakre and Xian-He Sun, "On Design Framework of Context Aware Embedded Systems", Monterey Workshop on Software Engineering for Embedded Systems, Sept. 2003.

[9] Christian Seeger, Kristof Van Laerhoven and Alejandro Buchmann. "MyHealthAssistant: An Event-driven Middleware for Multiple Medical Applications on a Smartphone-mediated Body Sensor Network". IEEE Journal of Biomedical and Health Informatics, Vol. 19, No. 2, pp. 752-760, March 2015.

[10] Oriana Riva. "Contory: a middleware for the provisioning of context information on smart phones". Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware. pp 219-239, Nov. 2006.

[11] Joonas Westlin and Teemu H. Laine. "ManySense: An Extensible and Accessible Middleware for Consumer-Oriented Heterogeneous Body Sensor Networks". International Journal of Distributed Sensor Networks, Vol. 2014, pp. 1-15, July 2014.

Authors

Nak-myoung Sung received the B.S. and M.S. degrees in Digital Information Engineering from Hankuk University of Foreign Studies, Korea in 2010 and 2015 respectively. His current research intere- sts include context-aware systems and Internet of Things(IoT).

Yunseok Rhee received the B.S. degree in Computer Science and Statistics from Seoul National University, Korea in 1984 and the M.S. degree in Information and Communication Engineering, and the Ph.D degree in Computer Science from KAIST, Korea in 1995 and 1999 respectively. He is currently a Professor in the School of Computer and Electronic Systems Engineering at Hankuk University of Foreign Studies. His current research interests include context-aware systems, embedded systems, and internet services.