Master's Thesis

Third-party Product Abstraction for Internet of Things Oriented Programming

Park HyunJoon (박 현 준)

Department of Computer Science and Engineering

Pohang University of Science and Technology

2016

사물인터넷 지향 프로그래밍을 위한 서드파티 제품 추상화

Third-party Product Abstraction for Internet of Things Oriented Programming

MCSE 박현준 Hyunjoon Park,
20142128 Third-party Product Abstraction for Internet of Things Oriented Programming
사물인터넷 지향 프로그래밍을 위한 서드 파티 제품 추상화
Department of Computer Science and Engineering, 2016, 108P,
Advisor: Hanjun Kim
Text in Korean.

ABSTRACT

In the era of Internet of Things (IoT), a number of products are connected to Internet, and provide a service interacting with other products. To develop an IoT service, programmers need to manage multiple third-party products in their IoT service application. However, since third-part product vendors provide different APIs for their products, managing multiple products is challenging for programmers. Computer scientists have tried to develop standard APIs for IoT products, but the standard APIs integrate only communication protocols, so programmers still need to know various features on underlying hardware. This work proposes third-party product abstraction that integrates the similar IoT product features into one API. The abstraction liberates programmers from underlying hardware and improve the productivity of IoT applications and promote the IoT industry. This work implements prototype abstract libraries and abstraction system for smartband, smart TV, and smart lightbulb services, and shows that the abstraction simplifies IoT programming without significant performance overheads.

Contents

1. 서론

일상적으로 이용되는 제품들이 인터넷에 연결되는 비중이 증가함에 따라[1], 인터넷을 통해 연결된 제품들이 하나의 서비스를 구축하여 제공하는 사물인터넷 어플리케이션들이 주목받고 있다[2]. 기존의 어플리케이션과 달리 사물인터넷 어플리케이션에서는 서드파티(third-party) 제품이 서비스의 기능에 대해 핵심적인 역할을 담당하고 있다. 예를 들면, 사용자의 건강 정보를 검진하고 관리하며 사용자에게 올바른 운동법이나 식이요법 등의 필요한 유의사항을 알려주는 '건강 정보 관리' 어플리케이션의 경우, 사용자의 신장, 체중 등의 신체 정보 및 걸음 수, 심박수, 수면 시간 및 품질 등의 건강 상태를 측정할 수 있는 센서가 필요하다. 건강 정보를 수집하기 위해, 어플리케이션 개발자가 직접 센서를 제작하는 경우도 있지만, 일반적으로 Apple Watch, Samsung Galaxy Gear, Fitbit 등 기존에 출시된 피트니스 트래커(Fitness Tracker) 서드파티 제품을 이용하는 경우가 대다수이다.

어플리케이션 개발에 있어, 서드파티 제품의 활용은 사물인터넷 서비스 제조비용을 낮추는 장점이 있지만, 어플리케이션을 특정 제품에 종속적으로 만드는 단점이 존재한다. 서드파티 제품을 사용하기 위해서는 제조업체에서 제공하는 제품, 펌웨어 및 개발툴을 이용하여 사물인터넷 어플리케이션을 개발해야 하므로, 하나의 어플리케이션은 하나의 서드파티 제품밖에

표 1 피트니스 트래커 제품군의 API Format과 Platform 비교

Model	Platform	API Format
Microsoft Band	iOS, Android, Windows Phone, OS X, Windows	BAND SDK
Jawbone Up24	iOS, Android	WEB API, UP SDK
Polar Electro M400	iOS, OS X, Windows	Polar AccessLink Service
Samsung Gear Fit	Android	TIZEN SDK
Fitbit Charge HR	iOS, Android, Windows Phone, OS X, Windows	WEB API, JAVA SDK
Fitbit Flex	iOS, Android, Windows Phone, OS X, Windows	WEB API, JAVA SDK
Fitbit Surge	iOS, Android, Windows Phone, OS X, Windows	WEB API, JAVA SDK
Xiaomi Mi Band	iOS, Android	Bluetooth Protocol

지원하지 못한다. 이를 해결하기 위해 제조업체들은 어플리케이션 프로그래머들이 자신들의 제품에 접근하고 제어할 수 있도록 open API 를 제공하고 있지만, API 포맷에 대한 표준이 통일되어 있지 않아 어플리케이션 프로그래머 입장에선 각 제품마다 다른 API 를 사용해야 한다는 문제를 여전히 안고 있다[3].

피트니스 트래커 제품군을 예로 살펴보자. 대표적인 피트니스 트래커의 제품군의 제품들이 제공하는 API 포맷의 경우 역시 각 제품마다 서로 다른 API 를 제공하고 있음을 확인할 수 있다(표 1). Microsoft Band, Jawbone Up24, Polar Electro M400, Samsung Gear Fit 의 경우 제조회사마다 내부 프로토콜을 SDK 로 묶어 제공하며, iOS, Android 등의 플랫폼에서 사용하도록 지시하고 있다. Fitbit 시리즈는 RESTful 등의 web API 를 이용하여 제품에 접근할 수 있도록 제공하고, Xiaomi Mi Band 는 Bluetooth 통신 프로토콜이 공개되어 Bluetooth 데이터 통신에 따라 기기를 제어할 수 있도록 제공하고 있다. 피트니스 트래커 제품들이 공통적으로 다양한 기능을 제공함에도 불구하고 건강 정보 관리 어플리케이션을 만들기 위해서는 어플리케이션의

표 2 피트니스 트래커 제품군의 기능 분류 및 비교

Model	Features						
	Activity	Calories	Distance	Elevation	Steps	Heart Rate	Sleep Time
Microsoft Band	O	O	O	O	O	O	O
Jawbone Up24	O	O	O	X	O	X	O
Polar Electro M400	O	O	O	O	O	X	O
Samsung Gear Fit	O	O	O	X	O	O	O
Fitbit Charge HR	O	O	O	O	O	O	O
Fitbit Flex	O	O	O	X	O	X	O
Fitbit Surge	O	O	O	O	O	O	O
Xiaomi Mi Band	O	O	O	X	O	X	O

각 기능마다 피트니스 트래커 제품군의 API 를 일일이 적용시켜야 하는 프로그래밍 상의 번거로움을 겪게 된다(표 2). 이처럼 현재 사물인터넷 어플리케이션 프로그래머가 겪고 있는 문제점을 정리하면 다음과 같다.

1) 비슷한 서비스를 제공하는 어플리케이션에도 불구하고, 어플리케이션이 지원하는 서드파티 제품이 하나 혹은 같은 제조업체의 제품만 지원함에 따라 어플리케이션의 사물인터넷 제품에 대한 적용 가능성(applicability)이 제한된다.

2) 제조업체에서 제공하는 API 가 통일되어 있지 않아, 생산성(productivity)이 결여된다.

3) 이미 존재하는 어플리케이션에 대해서 새롭게 생산되는 서드파티 제품에 대해 기능을 제공할 수 있는 확장성(expandability)이 부족하다.

위의 문제점을 해결하기 위해서, 이 연구는 사물인터넷 지향 프로그래밍을 위한 서드파티 제품 추상화 기술을 제안한다. 이 연구는 서드파티 제품들이 제공하는 공통적인 기능을 하나의 서비스 카테고리로 정의하고, 각 제품들에 대한 API 를 서비스 카테고리 차원에서 추상화하여

프로그래머 입장에서 하나의 통합 라이브러리를 형성해 서드파티 제품을 제어할 수 있게 하는 추상 라이브러리와 추상화 시스템을 제안한다. 추상 라이브러리(abstract library)는 제조업체 측의 제품 라이브러리와 계층적 관계를 가지며, 프로그래머가 추상 라이브러리만을 통해 제품 라이브러리를 접근할 수 있도록 한다. 추상 라이브러리에 명시된 추상 API 는 서비스 카테고리에서 지원하는 기능에 해당하며, 어플리케이션 프로그래머는 추상 라이브러리의 API를 호출하는 것으로 추상 라이브러리를 상속하는 실제 제품 라이브러리로부터 적합한 API를 호출하게 된다.

추상 라이브러리의 동작을 보조하기 위해, 추상화 시스템(abstraction system)은 서드파티 제품을 탐색하고 등록하며 추상 라이브러리와 제품 라이브러리를 연결하는 플러그 앤 플레이(plug-and-play) 기능을 지원한다. 추상 라이브러리 개발자에 의해 서비스 카테고리와 해당하는 API를 결정하며 이를 제조업체의 라이브러리, 즉 API 와 연결하기 때문에 제조업체 입장에서는 현재 추상화 시스템에서 제공하는 기능에 얽매이지 않고 자유롭게 제품의 기능을 추가, 변경할 수 있다.

이 연구의 핵심 가치를 정리하면 다음과 같다.

1) 계층적 구조에 따라 사물인터넷 서드파티에 대한 접근을 제품 라이브러리와 추상 라이브러리로 구분함에 따라서, 사물인터넷 어플리케이션 프로그래머는 서드파티 제품에 독립적으로 어플리케이션을 작성할 수 있다.

2) 추상화 시스템은 사용자의 사물인터넷 제품 내역을 반영할 수 있도록 실시간으로 추상 라이브러리와 해당 제품 라이브러리를 연결한다.

3) 스마트 밴드, 스마트 TV, 스마트 전구 서비스 카테고리에 대해서 추상 라이브러리 및 추상화 시스템의 프로토 타입을 구현하고 각각의 시스템을 분석하였다.

2. 관련 연구

이 연구는 사물인터넷 어플리케이션 프로그래머들이 서드파티 제품에 대한 접근성을 높이고 생산성을 높이기 위한 연구이다. 생산성 및 사물인터넷 연구는 크게 사물인터넷 표준화(IoT Standardization)와 사물인터넷 미들웨어(IoT Middleware) 두 가지 방향으로 진행되어 왔다.

2.1. 사물인터넷 표준화

사물인터넷의 표준화는 세 가지 방향에서 진행되어 왔다. 첫 번째는 프로그래밍 방법론에 대한 표준화로, 어플리케이션 프로그래머 측에서 어플리케이션에서 프로그램에 접근하는 API 에 대해서 규정하는 방법에 대해서 정의하고 있다. Alljoyn[4]에서는 Connected Lighting Project (CLP)와 Home Appliances & Entertainment (HAE) 서비스 프레임워크 프로젝트를 운영하여, Alljoyn 에 연결 가능한 제품들에 대해서 보편적인 API 를 지원하고 있다[5], [6]. Alljoyn 은 프로젝트에 참여할 제조업체들을 모집하여 CLP 는 전구 제품에 대해서, HAE 는 에어 컨디셔너, 공기 청정기, 공기 품질 모니터, 오븐, 냉장고, 로봇 청소기, 세탁기 및 TV 등의 가전 제품에 대해서 기능을 정의하고 각 기능에 대한 API 표준을 정의하고 있다. Alljoyn Core 와 Thin Client 가 동작하며 소프트웨어 상의 컴포넌트에 대해서 실제 사용가능한 제품을 연결해주는 역할을 지원하며, 이 연구와 유사한 기능을 제공한다. 단, CLP, HAE 는 표준 API 를 지정하고 있기 때문에,

프로젝트에 참여하는 제조업체 측에서 해당 표준에 맞추어 API 를 제공해야 하는 부담을 안겨준다.

두 번째 방향은 서비스 플랫폼에 대한 표준화로, M2M, 사물인터넷 서비스에 대해서 플랫폼을 회사가 소유하고, 사용자, 혹은 높은 수준의 프로그래머가 제품을 등록하고 플랫폼에서 제한하는 선에서 프로그래밍이 가능한 환경을 제공한다[7], [8]. 현재 상용화된 M2M/사물인터넷 플랫폼은 SmartThings[9], Xively[10], Everythng[11], ThingWorx[12], IoTivity[13] 등이 존재하며, 각 플랫폼은 사용자가 계정을 등록하고 서버에 등록된 클라우드를 바탕으로 클라우드에서 제공하는 서비스를 활용하도록 지원하고 있다. 쉽게 이용할 수 있는 반면에, 기능에 대해서 플랫폼이 제공하는 수준까지 밖에 프로그래밍을 할 수 없고 어플리케이션이 플랫폼에 종속된다는 한계점을 지닌다.

세 번째 방향은 서드파티 제품들의 네트워크 프로토콜에 대한 표준화이다. 서드파티 제품들은 Ethernet 을 시작으로 WIFI, Bluetooth, Zig-bee, Serial communication 등의 네트워크 프로토콜을 사용하는데, 네트워크 프로토콜에 맞추어 어플리케이션 프로토콜 또한 3GPP[14], IEEE 802.16 Wireless Broadcast Standards[15], ETSI[16], TIA TR-50 Committee 등 다양한 네트워크 표준들이 공존하고 있다. 이들은 내부 통신규약을 정의하지만 기능 및 서비스에 대한 규약이 없기 때문에 표준을 준수하더라도 어플리케이션 프로그래머 입장에선 여전히 같은 서비스에 대해 다른 비즈니스 프로토콜에 대해 다른 API 를 호출해야 한다.

2.2. 사물인터넷 미들웨어

사물인터넷 미들웨어는 사물인터넷에 참여하는 서드파티 제품에 대해서 서비스 컴포지션, 서비스 관리, 오브젝트 추상화 등의 단계로 나누는 모델과 이를 포괄하는 시스템을 의미한다[17-20]. 미들웨어에 대한 연구는 사물인터넷 제품들을 소프트웨어 컴포넌트로 취급하여 프로그래밍을 지원하지만, 제품들이 서드파티, 즉 프로그래밍 불가능한 제품이 아닌 프로그래밍 가능한 제품들에 대해 자바(JAVA)와 같은 하나의 플랫폼에서 통괄적으로 프로그래밍하는 환경을 제공하고 있다.

사물인터넷 게이트웨이 모델은 미들웨어 모델의 일종으로, 중앙집중적인 장치 연결 관리 모델이다[21], [22]. 게이트웨이(gateway)에 기기들이 플러그 인, 플러그 아웃하는 것을 관리하는 시스템으로, 주로 기기간에 서로 다른 네트워크 프로토콜을 관리하는 방법에 대해서 다루고 있다. 게이트웨이 모듈과 유사하게 사물인터넷 기기들이 웹에서 자유자재로 플러그 앤 플레이를 수행하도록 일종의 프록시(proxy) 역할을 하는 사물인터넷 웹 서비스 모델[23], [24]과 네트워크 프로토콜을 번역하는 모델들이[25], [26], [27] 있다. 이 연구는 사물인터넷 게이트웨이와 마찬가지로 플러그 앤 플레이를 지원하며 이종 네트워크간의 연결성을 보장한다. 하지만 게이트웨이 모델이 플러그 앤 플레이가 각 제품 단위로 이루어지는 데 반해, 이 연구의 추상화 시스템은 서비스 카테고리 단위로 플러그 앤 플레이를 지원한다. 즉, 기존 게이트웨이 모델과는 달리 같은 기능을 지원하는 제품에 대해서도 플러그 앤 플레이를 지원한다.

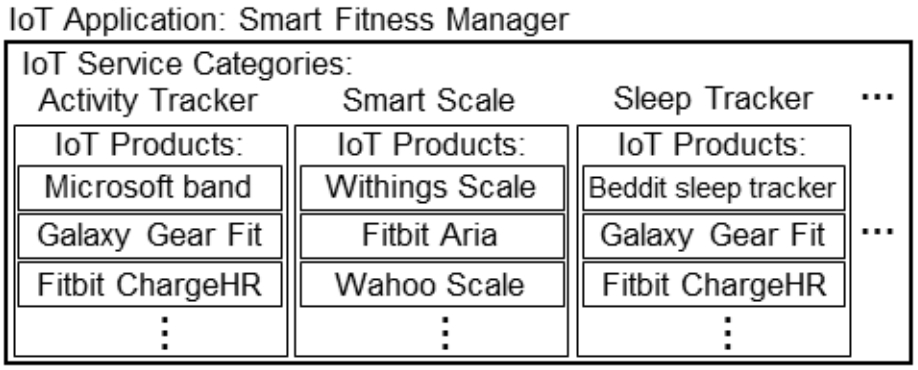
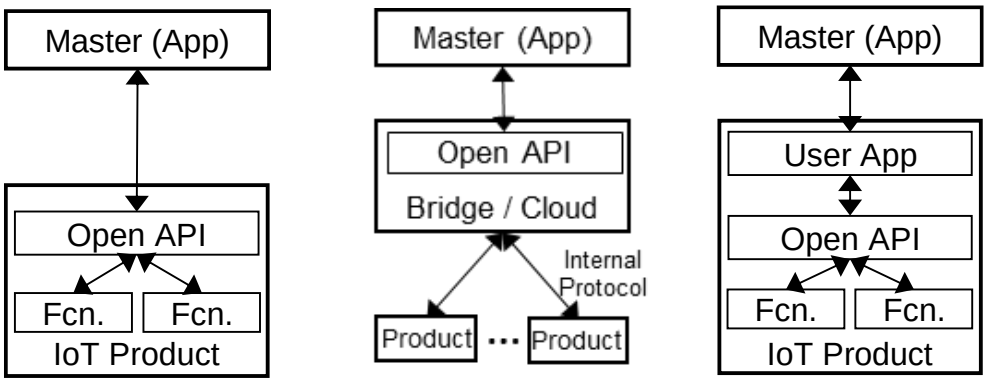


그림 1. 사물인터넷 어플리케이션: 사물인터넷 제품과 서비스 카테고리

3. 사물인터넷 서비스 개발의 어려움

사물인터넷 시대를 맞아 사물인터넷 제품으로 통칭되는 기기들은 다양한 기능을 내포하면서 사물인터넷 서비스를 제공하기 위해 복합적으로 상호작용하고 거대한 프로그램으로서 작용한다. 사물인터넷 제품들의 통합적인 상호작용은 다음과 같은 이유로 프로그램 개발을 어렵게 한다.

첫째, 하나의 사물인터넷 서비스는 여러 가지 서비스 카테고리로 기능에 따라 세밀하게 분류된 그룹이다. 예를 들면, 스마트 건강 관리 서비스는 활동 트래커(Activity Tracker), 스마트 저울(Smart Scale), 수면 트래커(Sleep Tracker) 등의 다양한 서비스 카테고리로부터 정보를 조합하여 사용자의 건강 상태를 검진한다(그림 1). 이 때 각 서비스 카테고리에는 해당 기능을 수행하는 서드파티 제품이 들어간다. 경우에 따라선 Galaxy Gear Fit 이나 Fitbit ChargeHR 처럼 하나의 제품이 다수의 서비스 카테고리에



(a) Direct Connection **(b) Indirect Connection** **(c) Programmable**

그림 2. 서드파티 제품의 다양한 환경

중복될 수도 있다. 스마트 건강 관리 서비스처럼 서비스 카테고리들이 복잡하게 구성될 경우, 프로그래머가 서비스 카테고리가 다루는 기능을 구분하는 데 어려움을 겪는다.

둘째, 같은 서비스 카테고리 내에서도 제품에 따라 다른 서비스를 제공할 수 있다. 예를 들면 같은 피트니스 트래커도 어떤 제품은 심박수를 측정할 수 있는 제품이 있는가 하면 그렇지 않은 제품도 있다(표 2). 즉 프로그래머는 제품에 따라서 다른 기능성에 대해서 세세하게 고려해가며 프로그램을 작성해야 한다.

셋째, 서드파티 제품의 API 기능의 다양성을 들 수 있다. IoT 제조업체마다 서로 다른 API를 제공하기 때문에 어플리케이션 프로그래머는 각 API 별로 별도의 처리를 해주어야 한다. 가령, 프로그래머가 스마트 건강 서비스를 제공하고자 할 때 Microsoft band와 Samsung Galaxy Gear Fit을 둘 다 지원한다고 한다면, 심박수를 구하는 `getHeart()` 함수를 작성할 때에도 현재 어플리케이션과 연동된 장치가 어떤 것인지 구하고 각 제품별로

케이스를 만들어 API 를 호출해야 한다(표 1). 이는 어플리케이션 개발을 더 어렵게 만드는 요인이다.

넷째, 같은 서비스 카테고리 안에 있는 제품이더라도 서드파티 제품마다 각기 다른 환경에서 동작할 수 있음을 고려해야 한다. 어플리케이션이 동작하는 주 장치인 마스터 장치가 서드파티 제품을 제어할 경우, 서드파티 제품의 환경에 따라서 접근 및 제어 방식이 달라질 수 있다. 서드파티 제품의 환경은 서드파티 제품이 마스터 장치에 직접 연결되어 API 호출이 제품으로 직접 전달되는 경우(그림 2(a)), 중도에 라우터, 허브 역할을 하는 브릿지나 클라우드를 걸쳐서 API 호출이 간접적으로 전달되는 경우(그림 2(b)), 서드파티 제품에 유저 어플케이션을 올릴 수 있어 유저가 만든 API 가 호출되는 경우(그림 2(c))로 나뉘게 된다. 피트니스 트래커를 예로 들면, Xiaomi Mi Band 의 경우 제품과 직접 통신하는 직접 연결 환경을, Fitbit ChargeHR 은 클라우드를 통해 간접 연결 환경을, Gear Fit 같은 경우 Tizen Application 을 Gear Fit 에 직접 올릴 수 있는 환경을 갖고 있다. 프로그래머는 API 호출 및 네트워크 통신에 있어서 제품에 직접 호출하는지 브릿지에 호출하는지 각 연결 환경에 대해서 고려해야 한다. 특히 프로그래밍이 가능한 제품의 경우 직접 어플리케이션을 작성하여 API 를 별도로 만들어야 하는 문제점이 있다.

마지막으로, 사물인터넷이 어플리케이션이 실행되는 환경은 전통적인 컴퓨팅 환경과는 달리 고정되어 있지 않다는 점이다. 서드파티 제품과 어플리케이션의 실행 주체인 마스터는 언제든지 이동 가능하며 서드파티 제품과의 연결은 언제든지 끊어지거나 연결되며, 여러 제품의 실행 환경이 중복될 수 있다. 가령, 걸음 수를 재는 프로그램은 피트니스 트래커가

감지될 경우 해당 장치로부터 센서 데이터를 가져오지만, 다른 장치로 유저가 바꿀 경우 해당 장치로부터 센서 데이터를 가져와야만 한다. 따라서 사물인터넷 어플리케이션이 실시간으로 사물인터넷 제품들을 추적해 이용가능한 제품에 연결해야 한다.

사물인터넷 어플리케이션에서 프로그래머에게 발생할 수 있는 문제점을 이상 다섯 가지로 정리하였다. 위 문제점을 해결하기 위해서는 본 연구가 제안하는 추상 라이브러리와 추상화 시스템은 다음과 같은 특징을 가져야 한다.

1) 추상 라이브러리는 서비스 카테고리를 분류하고, 각 서비스 카테고리가 지원하는 기능 및 제품에 대해서 정의한다. 프로그래머는 정의된 서비스 카테고리에 활용함에 따라 서비스 카테고리에 및 기능 정의에 대한 책임에서 해방된다.

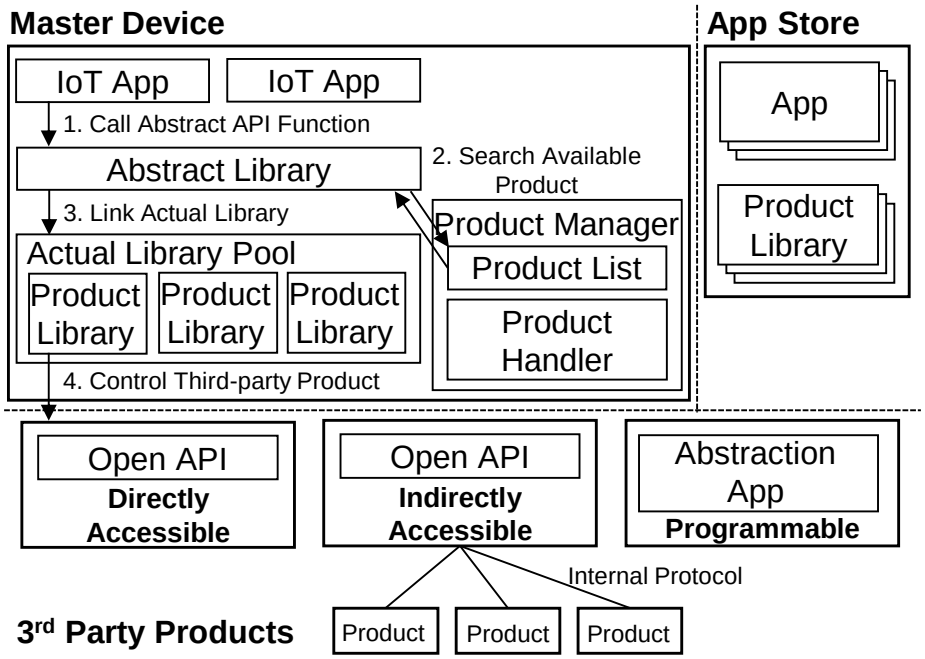
2) 추상 라이브러리는 서비스 카테고리에서 지원하지 않는 기능에 대해서 예외처리를 하고 이를 프로그래머에게 반환하여, 기능이 존재하지 않는 경우에 대해서 일괄적으로 처리할 수 있도록 지원한다.

3) 추상 라이브러리는 하나의 기능에 대해서 통일된 추상 API 를 제공하며, 프로그래머가 제품 API 들을 제어하지 않아도 추상 API 하나만으로 기능을 제어할 수 있도록 해야 한다. 추상화 시스템은 추상 API 와 제품 API 가 실시간으로 연결될 수 있도록 지원한다.

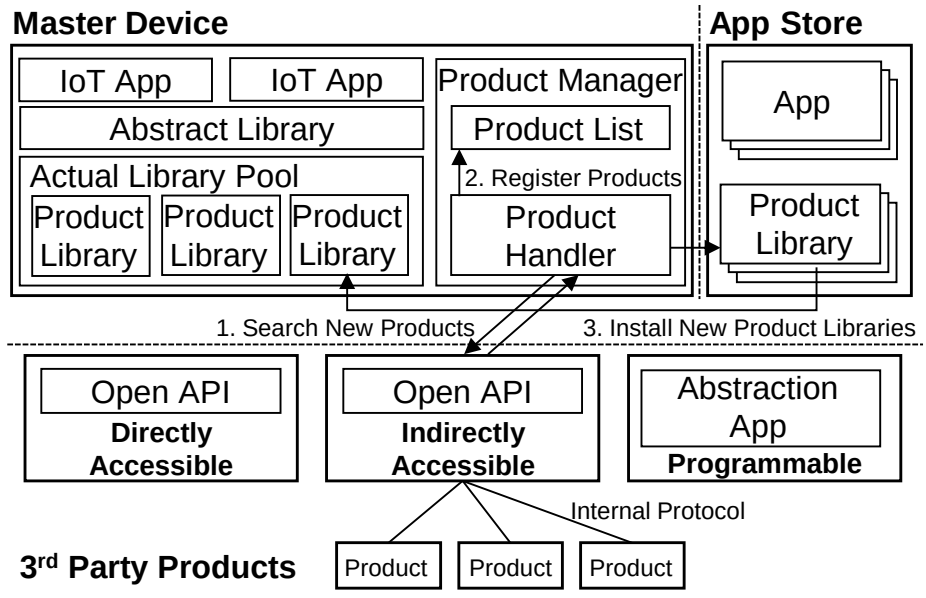
4) 추상화 시스템은 서드파티 제품의 환경에 대한 정보를 관리하며, 제품 연결 및 제품 API 에 대한 호출이 환경에 맞게 호출되도록 조정한다.

프로그래머가 직접 프로그램을 올릴 수 있는 서드파티 제품에 대해선 추상 라이브러리와 연결되는 제품 라이브러리를 제공한다.

5) 추상화 시스템은 서드파티 제품의 탐색 및 연결을 백그라운드에서 주기적으로 관리하며 플러그 앤 플레이를 지원한다.



(a) API 호출



(b) 제품 등록

그림 3. 서드파티 추상화 시스템 전체 구조도

4. 시스템 디자인

4 장은 전체 시스템 디자인에 대해 두 부분으로 나누어 설명한다. 4.1 장에서는 서드파티 제품이 런타임에서 어떻게 등록되고 동작하는지 추상화 시스템에 대해서 먼저 설명한다. 4.2 장에서는 추상 라이브러리의 구조 및 프로그래밍에 대해서 설명한다.

4.1. 추상화 시스템

추상화 시스템(Abstraction System)을 지원하는 어플리케이션의 도메인은 앱스토어(App Store), 마스터 장치(Master Device), 서드파티 제품(Third-Party Products)의 세 영역으로 분류된다(그림 3). 앱스토어는 어플리케이션을 배급하는 서비스 제공자로, 어플리케이션(App)과 함께 서드파티 제품에 대한 제품 라이브러리(Product Library)를 추상화 시스템의 요청에 따라 제공한다. 마스터 장치는 어플리케이션이 설치되는 장치로 사물인터넷 어플리케이션(IoT App), 추상 라이브러리(Abstract Library), 제품 라이브러리 풀(Actual Library Pool), 제품 관리자(Product Manager)를 갖고있다. 사물인터넷 어플리케이션은 어플리케이션 프로그래머가 작성한 프로그램으로, 사물인터넷 서비스에 대한 전체 명세를 담고 있다. 추상 라이브러리는 시스템에서 제공하는 프로그래머 측의 라이브러리로 서비스 카테고리의 기능을 추상 API 로 정의하여 API 의 묶음으로 이루어져 있다. 추상 API 는 프로그래머가 직접 호출하는 인터페이스로, 제품 라이브러리 풀에 설치된 서드파티 제품들의 제품 라이브러리의 제품 API 와 연결된다.

즉, 추상 라이브러리와 제품 라이브러리는 계층적 구조를 갖고 있으며, 추상 라이브러리가 지원하는 추상 API 에 대해 제품 라이브러리의 제품 API 가 구현체로서 상속받는 형식을 갖추고 있다. 제품 관리자는 현재 마스터 장치에 등록되어 있는 제품 리스트(Product List)와, 제품의 탐색 및 연결을 관리하는 제품 핸들러(Product Handler)로 구성되어 있다.

추상화 시스템은 API 호출과(그림 3(a)), 제품 등록을(그림 3(b)) 담당한다. API 호출은 어플리케이션에서 추상 라이브러리를 거쳐 실제 제품 API 를 호출하는 기능을 담당하고, 제품 등록은 주기적으로 마스터 장치에서 이용가능한 어플리케이션을 찾아 등록하는 역할을 한다.

4.1.1. API 호출

API 호출은 사물인터넷 어플리케이션에서 서드파티 제품을 제어하기 위해 추상 라이브러리의 API 를 호출하면서 시작한다(그림 3(a)). 처음에 추상 라이브러리에 API 호출이 도착했을 때에는, 현재 추상 라이브러리에 연결될 제품 라이브러리를 확인하고 존재하지 않을 경우 두 번째 단계로 이행한다. 두 번째 단계는 추상화 시스템은 현재 이용가능한 제품에 대해서 제품 관리자 안에 제품 리스트를 찾아보고, 현재 호출에 적합한 실제 제품을 선택한다. 만약 같은 카테고리 안에서 여러 제품이 존재한다면, 추상 라이브러리는 가장 최근에 사용한 제품을 선택한다. 프로그래머가 특정 제품을 사용하고자 할 때에는 API 호출에 제품 ID 를 같이 넘겨주어 해당 ID 를 가진 제품을 선택한다. 세 번째 단계는 추상 라이브러리가 두 번째 단계에서 넘겨받은 제품의 제품 라이브러리와 연결한다. 마지막으로 연결된 제품 라이브러리에서 추상 API 의 구현에 해당하는 제품 API 를 호출하여

표 3 제품 리스트 내의 제품 리스트

ID	Product	Service Category	Address	Auth.	Library	Connected	Last Connection
1	ChargeHR	Act. Tracker	192.168.11.3	"laa2.."	Fitbit_AT	Yes	7/11 15:04
1	ChargeHR	Slp. Tracker	192.168.11.3	"laa2.."	Fitbit_ST	Yes	7/11 15:04
2	Mi Band	Act. Tracker	88:0f:10:a0:90:8c	None	MI_AT	No	7/11 19:31
2	Mi Band	Slp. Tracker	88:0f:10:a0:90:8c	None	MI_ST	No	7/11 19:31

Algorithm 1: *Target Product Selection*

Data: *list* is a product list, *category* is the service category of the invoked API function
Result: *target* is an appropriate target product

```
1  let target = null;
2  let found = false;
3  foreach p ∈ list.getLists(category) do
4      if list.isConnected(p) then
5          if p.connect() then
6              target = p;
7              found = true;
8              break;
9          end
10         else
11             list.updateConnInfo(p, No);
12         end
13     end
14 end
15 if found then
16     list.moveFront(target);
17     list.updateAccessInfo(target, time);
18 end
19 return target;
```

최종적으로 제품을 제어한다. 시간 비용을 줄이기 위해서, 한 번 성공한 API 호출에 대해선 연결이 끊어질 때까지 두 번째 단계와 세 번째 단계를 생략한다.

두 번째 단계에서 적합한 타겟 제품 선택하는 것은 제품 리스트에 대한 반복 탐색에 의해 이루어진다(Algorithm 1). 선택하고자 하는 제품 p가 현재 제품 리스트에 들어있는지 검색하고, 존재하는 경우에 제품이 연결되어

있으면 타겟으로 선택하고, 연결되어 있지 않은 경우 리스트에서 p에 대한 연결 정보를 끊어졌다고 기록한다. 이 과정을 연결된 제품을 찾을 때까지 리스트에 있는 모든 제품에 대해서 반복한다. 타겟을 찾은 경우 해당 제품을 타겟으로 반환하고 타겟에 접근한 기록을 제품 리스트에 갱신한다(표 3).

4.1.2. 서드파티 제품 등록

추상화 시스템은 런타임에서 새로운 서드파티 제품에 대해서 탐색하고 탐색에 성공한 경우 제품을 시스템에 등록한다(그림 3(b)). 제품 탐색의 첫 번째 단계는 제품 관리자 안에 제품 핸들러가 주기적으로 새로운 제품에 대해서 탐색을 시도하는 것이다. 처음 시스템이 설치되었을 때 제품 라이브러리에는 추상 라이브러리가 지원하는 모든 제품에 대해, 제품을 탐색하는 `search()` API 가 설치되어 있다. 제품 핸들러는 각 제품들의 `search()` API 를 호출하며 해당 제품들을 탐색한다. 지정한 시간제한 안에 서드파티 제품이 탐색되지 않은 경우 실패로 간주하고 다음 탐색을 시행한다. 제품 탐색은 API 호출이 되었을 때 진행되는 것이 아니라, 백그라운드에서 주기적으로 진행한다. 즉 어플리케이션이 동작하지 않는 빈 시간에 제품을 미리 탐색하고 등록한다. 두 번째 단계는 제품 탐색에 성공한 경우 새로운 서드파티 제품에 대해서 제품 리스트에 등록한다. 제품을 등록할 때에는 제품의 이름, ID, 서비스 카테고리, 접근 주소, 암호화 정보, 라이브러리 종류, 연결 정보 등을 같이 기록한다(표 3). 세 번째 단계는 제품 리스트에 탐색한 제품을 등록하고 앱스토어에서 해당 제품의 라이브러리를 마스터 장치의 제품 라이브러리 풀로 가져와 설치한다. 처음부터 마스터 장치가

모든 제품 라이브러리를 갖고 있는 것이 아니라, 탐색에 성공한 경우에만 제품 라이브러리를 가져오는 것은 마스터 장치의 용량을 줄이기 위함이다.

표 4 ActTrcker**의 Function과 Event Signal**

Public Functions
public float ActTrcker::getDistance()
public int ActTrcker::getSteps()
public void ActTrcker::setCalIntake(float cal)
public float ActTrcker::getCalIntake()
public float ActTrcker::getCalBurned()
public float ActTrcker::getElevation()
public void ActTrcker::setExTag(int exType)
public int ActTrcker::getExTag()
public float ActTrcker::getHeartRate()

Event Signals	
STEP_CHANGED	Number of steps changed
HR_CHANGED	Heart rate changed

표 5 IoTProduct**의 Public Functions**

Method			Description
public	int	getID()	return id of the current product instance
public	IoTProduct	getInstance()	return the current product instance in the service category
public	IoTProduct	getInstance(id)	return the id product instance in the service category
public	IoTProduct	getNextInstance()	return the next available product instance in the service category
public	void	search()	search products in a category and update the product list
public	void	setLstner(int, Lstner)	set a listener for sig event

```
public class MainActivity extends Activity {
 ActivityTrackerType type;
 // ChargeHR
 OAuthService auth;
 Token accessToken;
 ...
 // GearFit
 HealthDataStore mStore;
 HealthDataStore.ConnectionListener mListener;
 ...

 protected void onCreate(Bundle savedInstanceState) {
  setContentView(R.layout.activity_main);
  ...
  type = getAvailableProduct();
  switch(type) {
   case ChargeHR:
    auth = new ServiceBuilder().provider(FitbitApi.class)
     .apiKey(userName).apiSecret(passwd).build();
    accessToken = auth.getAccessToken();
    ...
    break;
   case GearFit:
    HealthPermissionManager.getPermission();
    mStore = new HealthDataStore(mListener);
    mStore.connectService();
    ...
    break;
   case MiBand:
    ...
    break;
  }
  ...
 }

 protected float getHeartRate() {
  switch(type) {
   case ChargeHR:
    String urlGet = "https://api.fitbit.com/1/user/"
     + userName + "/activities/herat/date/today/1d.json";
    String result = HttpRequest(auth,accessToken,urlGet);
    return parseJSONvalue(result);
   case GearFit:
    String request = HealthConstants.HeartRate.HEART_RATE;
    return getHealthData(mStore, request);
    break;
   case MiBand:
    AlertDialog.Builder alert =
      new AlertDialog.Builder(MainActivity.this);
    alert.setMessage("There is no heart rate feature");
    alert.show();
    return 0;
  }
 }
 ...
}
```

그림 4. 스마트 건강 관리 시스템의 의사 코드 일부

```
1  public class MainActivity extends Activity {
2   ActTrcker actTrcker;
3   ...
4
5   protected void onCreate(Bundle savedInstanceState) {
6    setContentView(R.layout.activity_main);
7    ...
8    actTrcker = ActTrcker.getInstance();
9    ...
10   actTrcker.setLstner(STEP_CHANGED, new Lstner() {
11    public void run() {
12     int steps = getSteps();
13     ...
14    }
15   });
16   actTrcker.setLstner(HR_CHANGED, new Lstner() {
17    public void run() {
18     float rates = getHeartRate();
19     ...
20    }
21   });
22  }
23  ...
24  protected float getHeartRate() {
25   float rates = 0;
26   ActTrcker hrTracker = actTrcker;
27   while(rates == 0 && hrTracker != NULL) {
28    try{
29     rates = hrTrcker.getHeartRate();
30    } catch (NotSupportableFeatureException e) {
31     hrTrcker = getNextInstance();
32    }
33   }
34   if (rates == 0) {
35    AlertDialog.Builder alert =
36      new AlertDialog.Builder(MainActivity.this);
37    alert.setMessage("There is no heart rate feature");
38    alert.show();
39   }
40   return rates;
41  }
42  ...
43  protected void changeProduct(int ID) {
44   actTrcker = ActTrcker.getInstance(id);
45  }
46 }
```

그림 5. 그림 4의 스마트 건강 관리 시스템를 추상 라이브러리로 바꾼 일부

4.2. 추상 라이브러리(Abstract Library)

추상 라이브러리는 서드파티 제품의 다양한 기능을 통합하여 어플리케이션 프로그래머로 하여금 실제 제품의 상세를 추상화하고 추상 API 를 통해 간편하게 프로그래밍하도록 돕는다. 서드파티 제품에 대한 라이브러리는 프로그래머 측의 추상 라이브러리와 제품 측의 실제 라이브러리의 계층적 구조로 이루어진다. 4.2.1 장 에서는 각각 추상 라이브러리가 사용되는 예제를, 4.2.2 장에서는 라이브러리가 실제 제품을 제어하는 방식에 대하여 각각 설명한다.

4.2.1. 프로그래머 측의 추상 라이브러리

프로그래머 측의 추상 라이브러리는 어플리케이션 프로그래머에게 서비스 카테고리에서 제공하는 기능들에 대해 인터페이스를 제공한다. 예를 들어, `ActivityTracker` 추상 라이브러리는(표 4) 피트니스 트래커 제품군이 제공하는 모든 기능에 대한(표 2) API 함수를 제공한다. API 함수를 호출하는 것으로, 어플리케이션 프로그래머는 피트니스 트래커 제품에 대한 정보 상세 없이 기능들에 대해서 제어가 가능하다. 원래 프로그램과는 달리 (그림 4) 추상 라이브러리를 이용하여 새롭게 고쳐 쓴 프로그램에서는 (그림 5) 보안 토큰을 위한 특별한 코드나 JSON 코드를 작성하는 부분이 포함되지 않는다. 왜냐하면 새롭게 고쳐 쓴 코드의 추상 라이브러리 함수에 해당하는 `SmartBand` 생성자와 `getHeartRate` 이 제품의 상세를 숨겨주기 때문이다. 어플리케이션 프로그래머는 추상 라이브러리에서 제공해주는 추상 API 만 호출하면 되기 때문에, 제품 라이브러리의 버전이 올라가거나 새로운 제품이

어플리케이션에 추가되어도 어플리케이션 코드를 수정하지 않아도 된다. 즉, 제품 API 와 어플리케이션간의 종속성이 끊어짐을 확인할 수 있다.

IoTProduct 클래스는 추상 라이브러리에서 실제 제품 라이브러리에 접근할 수 있도록 연결하는 추상 클래스이다(표 5). 서비스 카테고리 클래스는 IoTProduct 클래스를 상속하며 IoTProduct 클래스의 인스턴스는 실제 그 제품의 컴포넌트와 일치한다. 즉, IoTProduct 클래스의 멤버 함수나 멤버 변수에 접근하는 것으로 서드파티 제품에 대한 제어를 할 수 있다, 또한 제품이 추상화 시스템에서 등록될 때 해당 제품에 대한 유일한 ID 를 부여받으므로, 추상 라이브러리는 같은 종류의 다른 제품에 대해서도 다른 ID 가 부여되어 각기 다른 제어를 수행할 수 있다. getID() 함수를 통해 재퓸 인스턴스의 ID 를 구할 수 있다. getInstance() 함수는 현재 IoTProduct 의 구현체에 해당하는 실제 제품에 대한 인스턴스를 반환하며 아직 IoTProduct 인스턴스와 실제 제품에 대한 연결이 없을시 인스턴스에 대해 서비스 카테고리의 제품들에 대한 연결을 시도하여 가장 먼저 반환되는 제품의 인스턴스를 갖는다. getInstance(id)는 제품 리스트에서 등록한 ID 에 해당하는 제품의 인스턴스를 반환한다. getInstance(id)가 스마트 전구같이 같은 종류의 제품에 대해서 독립적인 명령을 수행할 때 유용하게 사용할 수 있다. getNextInstance()는 현재 서비스 카테고리에 대해서 다음으로 탐색된 이용가능한 제품의 인스턴스를 반환한다. Search()는 해당 서비스 카테고리의 제품을 탐색하고 제품관리자의 제품 리스트를 업데이트한다. Search()는 일반적으로 프로그래머에 의해서 호출되지 않아도 백그라운드에서 지속적으로 호출되지만, 프로그래머가 원하는 시점에서 제품 리스트를 갱신할 수 있도록 함수를 지원한다. setLstner(int, Lstner)는 특정 이벤트 sig 에 대해서 리스너를 설치한다. 추상화 시스템은 sig

이벤트가 발생했을 때 리스너에 등록된 함수를 실행하며, 사물인터넷 어플리케이션은 즉각적으로 상황의 변화에 대해서 신호를 폴링(polling)하지 않고 반영할 수 있다.

추상 라이브러리를 이용하여 새로이 쓰여진 어플리케이션은(그림 5) 기존 버전의 어플리케이션과(그림 4) 비교해 액티비티 트레킹에 있어서 더 나은 기능을 제공함을 확인할 수 있다. 우선, 프로그래머가 setLstner를 이용하여 시그널을 설치하기 때문에, 심박수 변화에 대해서 이벤트 처리가 가능해짐을 확인할 수 있다(그림 5, Lines 10-21). 또한, 사물인터넷 어플리케이션은 제품이 어떤 기능에 대해서 지원하지 않는 경우에도 처리가 가능해진다. 가령, 액티피티 트레커가 심박수 측정을 지원하지 않는 경우, `getNextInstance()` 함수를 이용하여 다음 이용가능한 제품을 통해 대신 기능을 수행케 할 수 있다(그림 5, Lines 25-38). `getInstance(id)`를 이용하는 경우 사물인터넷 어플리케이션은 사용자에게 여러 제품 중 하나를 선택하도록 지정할 수 있다(그림 5, Lines 40-42).

```
public class ChargeHR extends ActTrcker {
 OAuthService auth;
 Token accessToken;
 ...

 public void connect(userName, passwd) {
  auth = new ServiceBuilder().provider(FitbitApi.class)
      .apiKey(userName).apiSecret(passwd).build();
  accessToken = auth.getAccessToken();
  ...
 }

 public float getHeartRate() {
  String urlGet = "https://api.fitbit.com/1/user/"
   + userName + "/activities/herat/date/today/1djson";
  String result = HttpRequest(auth,accessToken,urlGet);
  return parseJSONvalue(result);
 }
 ...
}
```

그림 6. 제품 라이브러리 예제: Fitbit Charge HR

```
public class Hue extends SmartBulb{
 ...
 public void changeColor(Color c) {
  httpRequest(...);
 }
 ...
}
```

```
public class PlainBulb extends SmartBulb{
 ...
 public void changeColor(Color c) {
  throw new NotSupportableFeatureException();
 }
 ...
}
```

```
public class MyPlainBulb extends PlainBulb{
 public void changeColor(Color c) {
  blink(5); // blink 5 seconds instead of changing color
 }
}
```

그림 7. 실제 라이브러리 함수의 세 가지 경우

4.2.2. 제품 측의 제품 라이브러리

제품 라이브러리는 서비스 카테고리의 기능에 대한 제품 API를 이용한 구현의 모음으로 하나의 제품 라이브러리는 하나의 제품에 대응한다. 서비스 카테고리를 기술하는 추상 라이브러리의 API는 추상 메소드로 존재하는데 그 구현이 제품 API의 호출과 연결된다. 예를 들어, 제품 라이브러리 **Fitbit ChargeHR** 는 추상 라이브러리 **ActTrcker** 클래스를 상속하며, 서비스 카테고리의 **ActTrcker** 의 기능에 해당하는 함수들에 대한 구현은 **Fitbit ChargeHR** 이 상속받아 실제 제품 API 를 호출한다(그림 6). 즉, 원래 프로그램(그림 4)에 존재하던 코드가 추상 라이브러리와 제품 라이브러리로 분리된 것을 확인할 수 있다. 그 결과 프로그래머는 서비스 카테고리에서 제공하는 추상 API만을 사용하여 어플리케이션을 제작할 수 있다.

제품 라이브러리를 구현할 때에는 세 가지 경우로 구분된다. 첫 번째는 제품이 기능을 지원하는 경우, 두 번째는 제품이 기능을 지원하지 않는 경우, 세 번째는 제품이 기능을 지원하지 않지만 대체 가능한 경우이다. **SmartBulb** 클래스의 예를 들어보자(그림 7). 추상 라이브러리의 **SmartBulb**를 상속 받는 Hue 제품 라이브러리의 경우 전구의 색을 바꾸는 **changeColor(Color c)**이 제품 API 차원에서 지원하는 첫 번째 경우이다. **PlainBulb** 제품은 색 변환을 지원하지 않으므로 예외상황을 throw 한다. 스마트 건강 관리 어플리케이션에서도 심박수를 지원하지 않는 제품의 경우 예외를 **NotSupportableFeatureException**을 throw하며 그에 대한 처리를 하는 것을 확인할 수 있다(그림 5, Lines 30-32). **MyPlainBulb** 의 제품 차원에서는 색 변경 기능을 제공하지 않지만 **blink()** 기능으로 대체한 경우이다.

5. 실험 및 분석

5.1. 실험 환경

추상 라이브러리와 추상화 시스템의 효용성을 이해하기 위해 측정해야 하는 성능은 다음과 같다. 추상 라이브러리의 핵심적인 목적은 프로그래머의 생산성이다. 추상 라이브러리를 이용했을 때와 그렇지 않을 때를 비교하여 프로그래머의 생산성의 변화를 살펴보아야 한다. 이를 위해, 같은 동작을 하는 사물인터넷 어플리케이션에 대해서 일반적인 어플리케이션과 추상 라이브러리를 사용한 어플리케이션에 대해 코드 라인 수(Lines of Code, LoC)를 비교하였다.

추상 라이브러리는 추상화 시스템의 보조를 통해서 동작하기 때문에 추상 라이브러리를 이용한 어플리케이션은 기존의 어플리케이션에 대해 시간 비용이 발생할 수 있다. 추상 라이브러리를 사용했을 때와 그렇지 않았을 때의 성능 차이를 비교하기 위해 두 가지 버전의 어플리케이션에서 지원하는 API 함수에 각각에 대해 API 호출로부터 결과가 반환될 때까지의 반응시간을 비교하였다.

나머지 두 가지 실험은 어플리케이션 탐색시간과 연결시간의 차이 및 하나의 서비스 카테고리에 대해서 등록가능한 제품 수가 늘어남에 따라 확장성(scalability) 비교 실험이다. 추상화 시스템의 제품 탐색 알고리즘이 모든 제품에 대해서 탐색에 성공할 때까지 탐색함에 따라서, 제품 수가 많아질수록 탐색 시간이 증가하는 추세를 보일 것이다. 실험은 등록가능한 제품 중 하나의 제품만 탐색 가능하게 하고, 제품탐색의 순서를 랜덤하게 섞어놓았다. 또한, 제품탐색의 순서를 결정하는 알고리즘 별로 등록가능한

제품 수가 늘어남에 따라 탐색의 평균적인 시간을 비교하여 제품 탐색 알고리즘의 확장성을 검증하였다.

프로토 타입에 대한 구현은 상용 제품으로 스마트 밴드(Smartband) Fitbit Charge HR, Galaxy Gear Fit, Xiaomi Mi Band 세 종류, 스마트 전구(Smart LightBulb) Philipse Hue, LIFX 두 종류, 스마트 TV(Smart TV)에 Chromecast, Amazon Fire TV, Tizen TV 세 종류에 대해서 총 세 개의 제품군의 여덟 개의 제품에 대해 구현하였다. 스마트밴드는 스마트 바디 오브젝트로, 항상 몸에 착용하며 지근거리에서 발생하는 통신에 대해서 시간 비용이 발생하는지 확인하는 것이 주 목표다. 스마트 전구는 같은 종류의 제품에 대해서 여러 개의 오브젝트를 제어할 수 있는지를 확인하는 것이 주 목표다. 스마트 TV 는 프로그래밍이 가능한 오브젝트에 대해서도 추상 라이브러리와 추상화 시스템이 잘 동작하는지 확인하기는 것이 주 목표다.

프로토 타입의 구현은 Android API 19(version 4.4 KitKat)을 기반으로 구현하였으며, 실험은 Google Nexus 7 에서 진행하였다.

5.2. 프로그램 개발 부담 경감 효과 분석

기존의 어플리케이션과 추상 라이브러리를 이용한 어플리케이션의 LoC 를 비교하였을 때 평균적으로 6.0 배 더 적었다(그림 8.). 기존의 여러 개의 제품을 지원하는 어플리케이션의 LoC 가 단일 어플리케이션의 LoC 의 합산과 비교했을 때 평균적으로 0.5 배 정도 적은데, 이는 같은 기능을 지원함에 따라서 단일 어플리케이션과 기존의 Total 어플리케이션 간에 중복되는

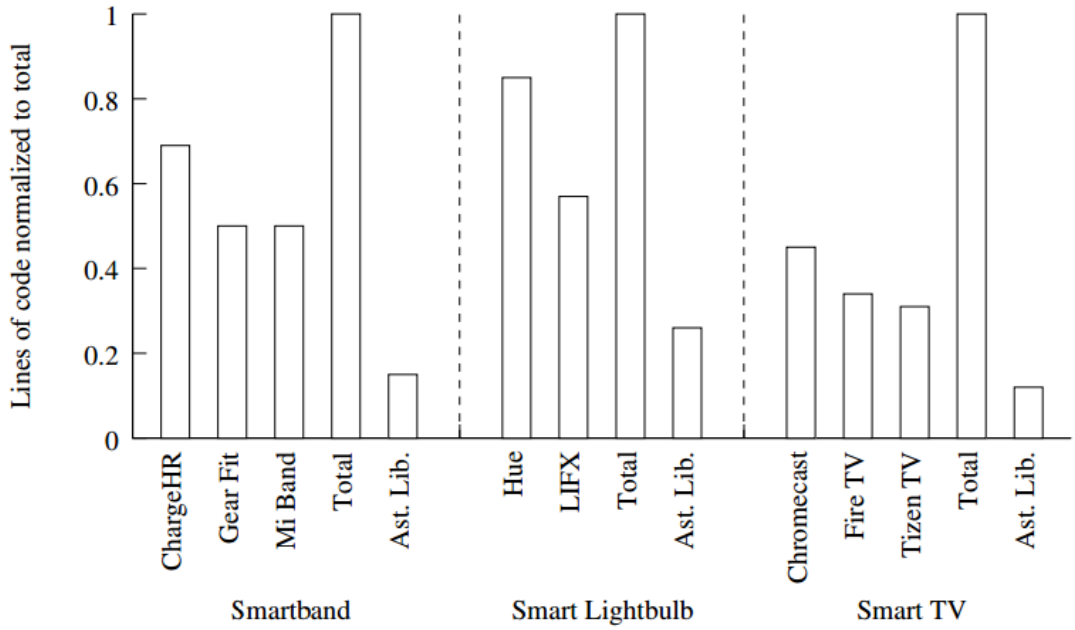


그림 8. 각 어플리케이션의 LoC 정규화 값 비교. Total은 제품 군에 포함된 모든 제품을 지원하는 기존 방식의 어플리케이션의 LoC, 정규화의 기준 값. Ast Lib.은 추상 라이브러리를 이용한 어플리케이션의 LoC.

코드가 존재하기 때문이다. 한 가지 주목할 점은, 추상 라이브러리를 사용한 어플리케이션이 단일 어플리케이션보다 LoC 가 줄어든 점인데, 단일 라이브러리 역시 실제 라이브러리 코드를 호출하는 반면, 추상 라이브러리의 경우 실제 라이브러리 코드에 대한 호출이 추상 라이브러리에 의해 가려지므로, 추상 API 에 대한 호출로 치환되기 때문에 줄어든 것이다.

5.3. 추상화 시스템 성능 분석

각 제품군의 기능에 대한 반응시간 비교는 유의미한 차이를 보이지 않았다(그림 9), (그림 10), (그림 11). 제품이 어플리케이션에 등록되고 난 후에 API 호출로부터 반응시간을 비교한 것으로, 추상화 시스템에서 제품

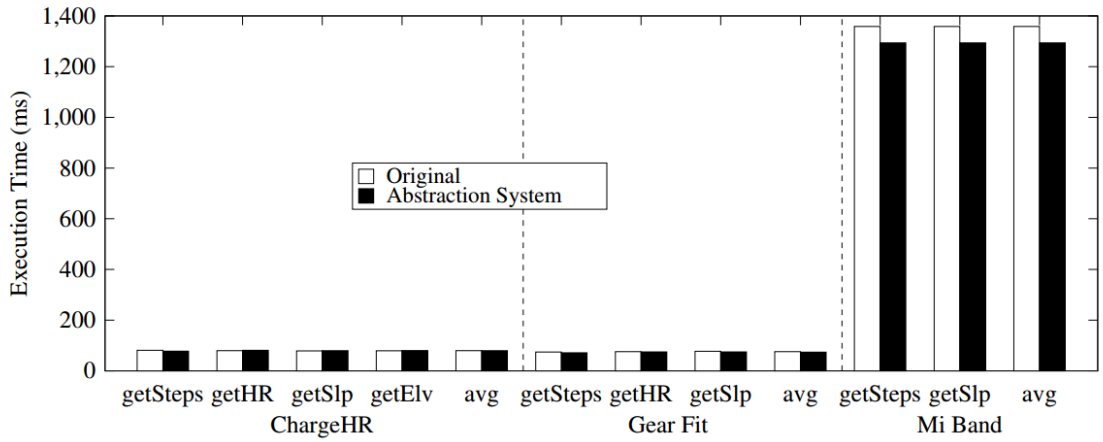


그림 9. 스마트 밴드의 각 기능에 대한 반응 시간 비교.

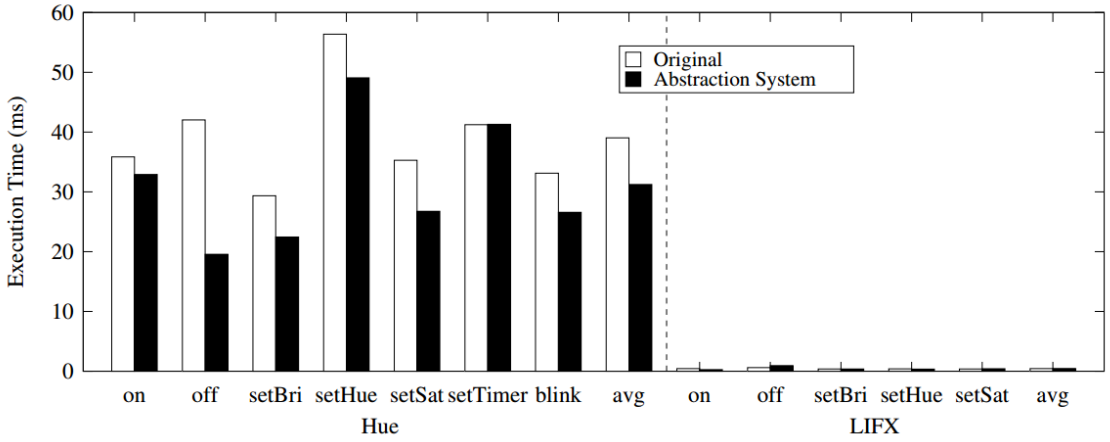


그림 10. 스마트 전구의 각 기능에 대한 반응 시간 비교

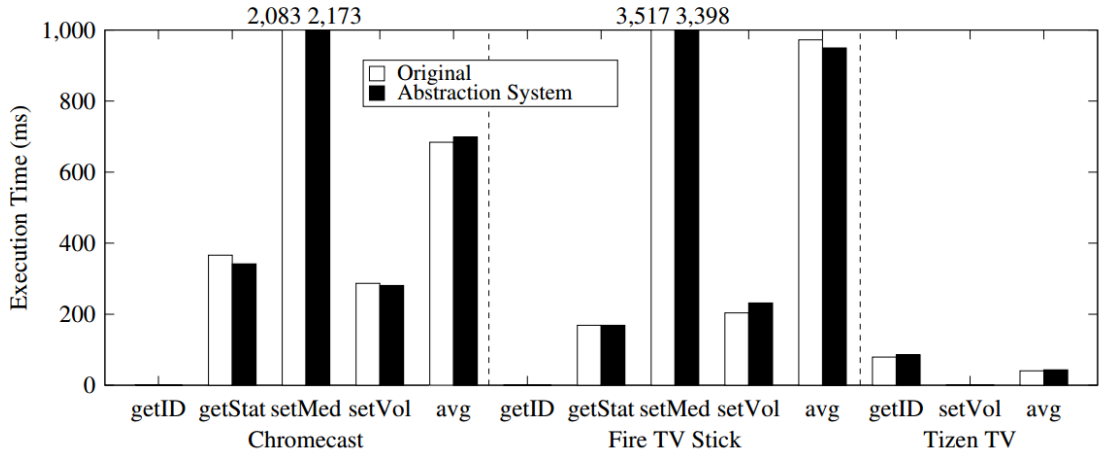


그림 11. 스마트 TV의 각 기능에 대한 반응시간 비교

실험을 반복하면서 횟수를 늘릴 경우, 4.1.1 장의 API 호출에서 두 번째 단계와 세 번째 단계가 생략됨에 따라서 추상화 시스템의 실질적인 API 호출 반응시간이 제품 API 호출 시간에 수렴함을 의미한다.

5.4. 추상화 시스템의 제품 탐색 성능 분석

각 제품군에 대한 탐색 및 연결 시간 비교는 단일 제품의 탐색 및 연결 시간에 비교해 스마트 밴드가 평균 9.3 초, 스마트 전구가 평균 2.5 초, 스마트 TV 가 평균 4.0 초 더 오래 걸린다(그림 12). 탐색 및 연결 시간 비교 실험은 각 서비스 카테고리에 대해서 하나의 제품만 존재한다고 가정하였을 때의 시간 비교이다. 현재 알고리즘은 타임 아웃에 걸릴 때까지 탐색을 시도하고, 이용 가능한 서드파티 제품이 발견될 때까지의 모든 제품을 탐색하기 때문에 탐색 시간이 증가한 것이다. 탐색 시간의 증가는 탐색에 대해 현재 단순한 알고리즘을 사용하기 때문에 발생한 문제로, 알고리즘 개선을 통해 탐색 시간을 줄일 수 있다. 프로파일링을 통해 가장 최근에

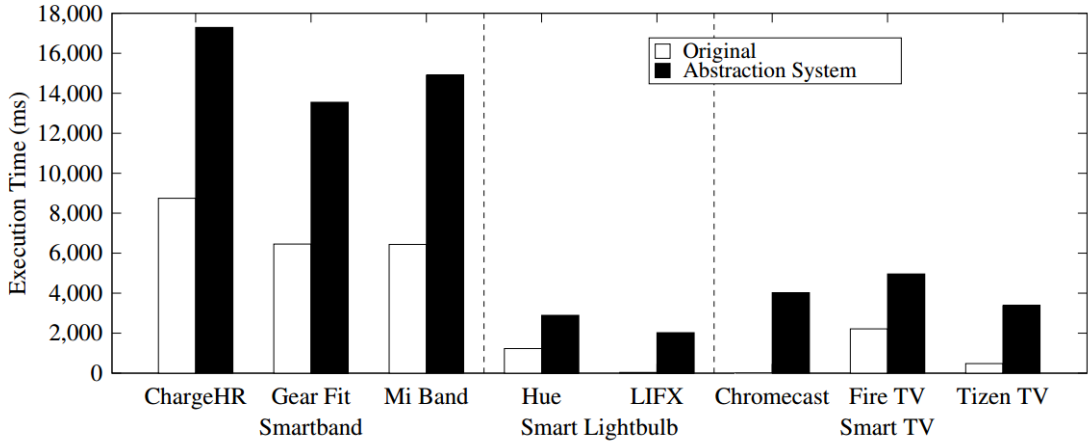


그림 12. 각 제품에 대한 탐색 및 연결 시간 비교 (한 제품에 대한 Timeout 5초, 스마트 밴드에 대해서는 Timeout 10초)

많이 사용한 서드파티 순으로 탐색 순서를 변경할 경우 일반적으로 탐색 시간이 줄 것을 기대한다. 또한, 타임아웃을 5 초로 동일하게 제한하여 실험하였는데, 스마트밴드의 경우 평균적으로 탐색시간이 5초보다 값이 컸기 때문에 값을 증가시킨 반면, 다른 두 제품군은 평균적인 탐색시간이 짧아 5 초의 타임아웃이 큰 시간 비용을 발생시켰다. 프로파일링을 통해서 제품들에 대한 평균적인 탐색 시간을 기록하고, 유동적으로 타임 아웃을 적용시킬 수 있는 경우 탐색의 시간 비용을 더 줄일 수 있다. 다만, 제품 탐색의 경우 어플리케이션 시작이나 API 요청 시점이 아닌, 백그라운드에서 사용되는 시간이기 때문에, 탐색 시간의 상당수는 백그라운드에서 대기(Idle) 시간에 잠식될 것을 기대한다.

탐색해야 하는 제품 수가 증가할 경우의 탐색의 시간 비용 증가와 확장성에 대해서도 비교하였다(그림 13). best case 는 첫 번째 시도로 탐색에 성공한 경우, worst case는 마지막 번째의 시도에 탐색에 성공한

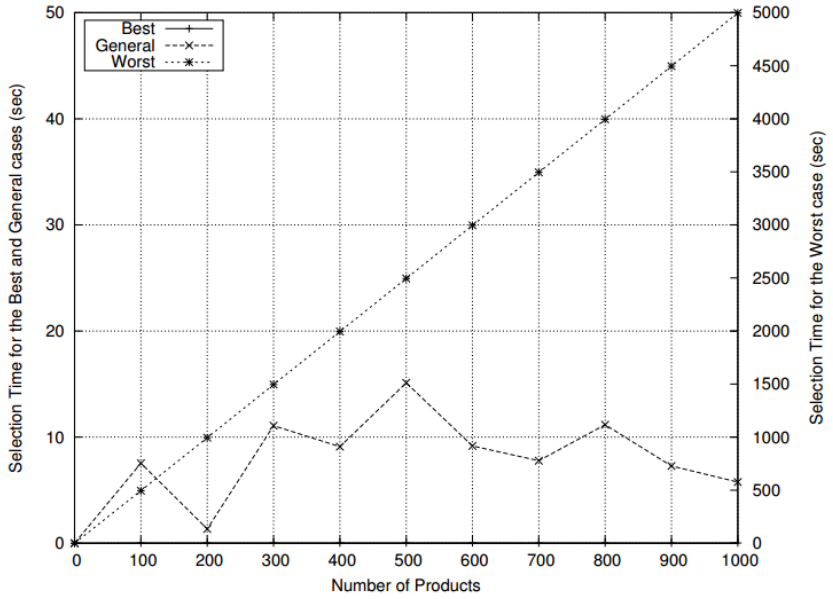


그림 13. 제품 수에 따른 제품 탐색에 소비한 시간 비교

경우로, general case 는 LRU 기법에 따라 최근에 연결한 서드파티 제품에 대해서 우선 순위를 높이고 실험한 결과다. 유저가 대부분 10 개의 제품에 대해서 사용한다는 가정을 두고 사용할 확률을 랜덤으로 놓고 실험한 결과 타임아웃 5 초에 대해서 제품 수가 증가하더라도 제품 탐색 시간이 크게 변하지 않는 것을 확인할 수 있다. 실제 사용 예와 비교하면 결과가 달라질 수 있으나, 일반적으로 서드파티 제품을 10 개 이상 사용하지는 않을 것으로 기대하며 알고리즘 적용 시 추상화 시스템은 확장성을 갖는다고 할 수 있다.

6. 향후 연구

추상 라이브러리 및 추상화 시스템에 대하여 향후 연구할 사항들은 다음과 같다. 우선, 확장성 실험에서 언급한 바와 같이 서드파티 제품의 탐색 알고리즘을 개선해야 한다. 문제의 단순화를 위하여 탐색 순서에 프로파일링을 통해 사용자가 최근에 사용한 제품에 대해서 우선순위를 제공하였지만, 제품의 특징 역시 우선순위에 영향을 미칠 수 있음을 고려해야 한다. 가령, 같은 피트니스 트래커 서비스 카테고리에서도 걸음수를 측정한다고 했을 때, 스마트폰에서 제공하는 센서보다 Fitbit 등의 스마트밴드의 센서가 더 정확할 수 있다. 심박수를 측정할 경우, Xiaomi Mi Band 의 경우 심박수를 측정하는 기능이 존재하지 않으므로 제품을 탐색할 때 탐색 범위에서 소거할 수 있다. 즉, 제공하고자 하는 기능에 따라서 연결하고자 하는 제품의 우선순위가 달라지는 점을 반영해야 한다. 가장 간단한 해결책은 추상 라이브러리 제작자가 우선순위 점수를 부여할 수 있도록 룰을 제공하는 것이다. 기능성의 유무 및 센서의 정확도와 품질, 기능 연관도 등을 점수를 매기는 요인으로 활용할 수 있다. 더 나아가서 해당 요인들을 바탕으로 머신러닝을 통해 우선순위를 선정할 수 있다.

상황 인지(context-aware) 요소를 추상 라이브러리에서 지원하는 것 역시 남은 연구 주제이다. 현재 추상 라이브러리에 같은 제품이 다수 존재할 때, 각 제품에 ID 를 부여하여 개별적인 컨트롤을 제공하고 있다. 하지만 ID 만으로는 프로그래머가 상황 정보를 반영하기에는 부족한 점이 있다. 가령, 전구 제품에 대해서 하나는 주방에 있고, 하나는 거실에 있을 때 프로그래머가 위치 정보를 반영하여 주방과 거실의 전구를 각각 컨트롤하고자 하여도, 추상화 시스템에 ID 정보만 등록되어 있으므로 이는

불가능하다. 즉, 프로그래머는 상황 정보를 사용하는데 있어 ID 정보만을 이용하도록 제한된다. 이를 해결하기 위하여, 추상 라이브러리와 추상화 시스템에 제품에 대한 상황 정보를 추가할 수 있다. 전구를 등록할 경우 전구의 위치 정보, 시간 정보, 개별 식별 정보 등 다양한 상황 정보를 등록하고, 추상 라이브러리를 통해 프로그래머가 제품의 인스턴스를 불러올 경우 상황 정보를 입력하여 해당 인스턴스에 접근할 수 있도록 한다. 이 때, 추상 라이브러리를 제작자가 라이브러리들을 정의해야 하며, 상황 정보에 대한 정의 및 규격화 문제를 우선적으로 해결해야 한다.

7. 결론

이 연구는 서드파티 제품의 추상화를 통해 제품 API 와 서비스 카테고리의 통합적 기능을 분리할 수 있음을 보였다. 추상 라이브러리는 프로그래머에게 제품의 상세를 생략하고도 서드파티 제품을 제어할 수 있도록 하면서, 사물인터넷 어플리케이션 프로그래밍을 쉽게 하고 프로그래머의 생산성을 높였다. 추상화 시스템은 서드파티 제품을 탐색하고 추상 라이브러리와 실시간으로 연결하여 서비스 카테고리 단위의 플러그 앤 플레이를 지원하였다. 이 연구는 스마트밴드, 스마트 전구, 스마트 TV 서비스 카테고리에 대해 각각 프로토타입을 구현함으로써 추상 라이브러리의 활용이 시간 비용의 소모 없이 실제 사물인터넷 어플리케이션 프로그래밍의 생산성을 높임을 보였다. 이 연구는 사물인터넷 시대에 있어, 사물인터넷 어플리케이션의 개발을 촉진하고 활성화하는데 이바지할 것이다.

REFERENCES

[1] National Intelligence Council, Disruptive Civil Technologies – Six Technologies with Potential Impacts on US Interests Out to 2025 –Conference Report CR 2008–07, April 2008, http://www.dni.gov/nic/NIC_home.html

[2] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D., "Context Aware Computing for The Internet of Things: A Survey," in Communications Surveys & Tutorials, IEEE , vol.16, no.1, pp.414–454, First Quarter 2014

[3] K. Chang, A. Soong, M. Tseng, and Z. Xiang, "Global Wireless Machine-to-Machine Standardization," IEEE Internet Computing, 2011

[4] "AllSeen Alliance," https://allseenalliance.org.

[5] "Home Appliances & Entertainment (HAE) Service Framework Project," https://wiki.allseenalliance.org/hae.

[6] "Connected Lighting Project," https://wiki.allseenalliance.org/tsc/connected lighting

[7] M. Castro, A. Jara, and A. Skarmeta, "An Analysis of M2M Patforms: Challenges and Opportunities for the Internet of Things," in Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012.

[8] J. Kim, J. Lee, J. Kim, and J. Yun, "M2M Service Platforms: Survey, Issues, and Enabling Technologies," IEEE Communications Surveys Tutorials, 2014.

[9] "SmartThings," http://www.smartthings.com/.

[10] "Xively," http://xively.com.

[11] "EVRYTHNG," https://evrythng.com/.

[12] "ThingWorx," http://www.thingworx.com/

[13] "IoTivity Project," https://www.iotivity.org.

[14] A. Damnjanovic, J. Montojo, Y. Wei, T. Ji, T. Luo, M. Vajapeyam, T. Yoo, O. Song, and D. Malladi, "A Survey on 3GPP Heterogeneous Networks," IEEE Wireless Communications, 2011.

[15] C. Cordeiro, K. Challapali, D. Birru, and N. Sai Shankar, "IEEE 802.22: the First Worldwide Wireless Standard Based on Cognitive Radios," in IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005.

[16] "M2M standards: Possible extensions for open API from ETSI," 2012

[17] "L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," Computer Networks, 2010

[18] S. de Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker, "SODA:Service Oriented Device Architecture," IEEE Pervasive Computing, 2006.

[19] J. Pasley, "How BPEL and SOA are Changing Web Services development", IEEE Internet Computing, 2005.

[20] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa, "SOA-Based Integration of the Internet of Things in Enterprise Services," in IEEE International Conference on Web Services, 2009.

[21] H. Chen, X. Jia, and H. Li, "A Brief Introduction to IoT Gateway", in IET International Conference on Communication Technology and Application (ICCTA 2011), 2011.

[22] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IoT Gateway: Bridging Wireless Sensor Networks into Internet of Things," in IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC), 2010.

[23] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H. Schmidtke, and M. Beigl, "Using Web Service Gateways and Code Generation for Sustainable IoT System Development," in Internet of Things (IoT), 2010.

[24] S. Datta, C. Bonnet, and N. Nikaein, "An IoT Gateway Centric Architecture to Provide Novel M2M Services," in IEEE World Forum on Internet of Things (WF-IoT), 2014.

[25] J. Nakazawa, H. Tokuda, W. Edwards, and U. Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems," in 26th IEEE International Conference on Distributed Computing Systems, 2006.

[26] H. Park, B. Kim, Y. Ko, and D. Lee, "InterX: A service interoperability gateway for heterogeneous smart objects," in IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011.

[27] H. Cerqueira Ferreira, E. Dias Canedo, and R. de Sousa, "IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and Arduino", in IEEE 9th International Conference on Wireless and Mobile Computing, Networking