KIPS Tr. Software and Data Eng.
Vol.5, No.8 pp.351~360 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.8.351

Design Patterns for Mitigating Incompatibility of Context Acquisition Schemes for IoT Devices

Hyun Jung La[†] · Ku Hwan An[††] · Soo Dong Kim[†††]

ABSTRACT

Sensors equipped in Internet-of-Thing (IoT) devices are used to measure the surrounding contexts, and IoT applications analyze the contexts to infer situations and provide situation-specific smart services. There are different context acquisition schemes including pulling, pushing, and broadcasting. Most IoT devices support only one of the schemes. Hence, there can be an incompatible issue on data acquisition schemes between applications and devices, and consequently it could result in an increased development cost and inefficiency on application maintenance. This paper presents design patterns which can effectively remedy the incompatibility problem. By applying the patterns, IoT applications with incompatibility can be systematically and effectively developed. And, also its maintainability is expected to increase.

Keywords : Internet-of-Thing, Context Acquisition, Incompatibility, Mitigation, Design Patterns

사물인터넷 컨텍스트 획득 비호환성 중재를 위한 디자인 패턴

라 현 정[†] · 안 구 환[††] · 김 수 동[†††]

요 약

사물인터넷의 센서는 주변 컨텍스트를 수집하는데 사용되며, 애플리케이션은 이를 분석하여 상황을 인지하거나 예측한 후 상황에 특화된 스마트 서비스를 제공하게 된다. 사물인터넷 컨텍스트를 확보(Acquisition)하는데 풀링, 푸싱 등 방식들이 있는데, 대부분의 사물인터넷 장비는 특정한 하나의 방식만을 지원한다. 따라서, 애플리케이션이 필요로 하는 컨텍스트 획득 방식과 장비가 지원하는 획득 방식이 일치하지 않을 수 있으며 이 경우 개발 노력의 증가, 향후 유지보수 비효율성등의 문제가 발생한다. 본 논문에서는 애플리케이션과 장비 간의 컨텍스트 획득 방식 비호환성을 효과적으로 해결하기 위한 디자인 패턴(Design Pattern)들을 제시한다. 제시된 패턴들을 적용하면, 비호환성이 발생하는 사물인터넷 애플리케이션을 보다 체계적이며, 효과적으로 개발할 수 있고, 나아가 향후 유지보수의 효율성도 증가된다.

키워드 : 사물인터넷, 컨텍스트 획득, 비호환성, 중재, 디자인 패턴

1. 서 론

사물인터넷(Internet of Things, IoT) 장비는 네트워크 연결 기반의 장비로서 센서(Sensor)와 액츄에이터(Actuator)로 구성되어 있으며, 장비 유형에 특화된 기능을 제공한다[1, 2]. 사물인터넷 애플리케이션은 센서로 사용자나 그 주변 환경에 대한 정보를 수집하여 분석하고, 분석 결과에 따라 상황에 맞는 기능을 제공한다.

사물인터넷 컨텍스트를 수집하는 방식은 전형적인 풀링(Pulling) 이외에도 푸싱(Pushing), 통지-획득(Notify-and-Fetch) 등 다양하다[3]. 대부분의 사물인터넷 장비는 이 중에서 한 종류의 방식만 지원한다. 예를 들면, 사람의 뇌파를 측정할 수 있는 Mindwave는 푸싱 방식으로 뇌파가 측정될 때마다 이 정보를 전송한다.

사물인터넷 컨텍스트 획득의 기술적인 어려움은 사물인터넷 애플리케이션이 필요로 하는 컨텍스트 획득 방식과 사용되는 사물인터넷 장비가 제공하는 컨텍스트 획득 방식이 일치하지 않는 경우에 있다. 즉 필요로 하는 획득 방식과 제공되는 획득 방식의 비호환성(Incompatibility) 문제가 발생한다. 이 경우, 개발자가 컨텍스트 획득 방식에 종속적으로 애플리케이션을 설계 구현하게 되면, 향후 사용되는 사물인터넷 장비가 변경이 되면 이에 따라 애플리케이션도 수정해야 하는 문제가 발생한다.

※ 이 논문은 서울산업진흥원의 지원을 받는 서울시 산학연 협력사업(PA140034)과 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단(2015R1A2A2A01004078)의 지원을 받아 수행한 연구임.
† 종신회원 : ㈜스마티랩 대표
†† 준 회 원 : 숭실대학교 컴퓨터학과 석사과정
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
Manuscript Received : January 4, 2016
First Revision : April 15, 2016
Accepted : April 20, 2016
* Corresponding Author : Soo Dong Kim(sdkim777@gmail.com)

본 논문에서는 필요로 하는 컨텍스트 획득 방식과 제공되는 컨텍스트 획득 방식의 비호환성을 해결하기 위한 디자인 패턴(Design Pattern)들을 제시한다. 획득 방식의 비호환성 문제를 본 논문에서 제시되는 패턴을 적용하여 애플리케이션을 개발하면 다음과 같은 기술적 장점을 기대할 수 있다.

- 장점 1) 사물인터넷 애플리케이션 개발 시 컨텍스트 획득 방식의 비호환성 문제를 효과적으로 해결할 수 있다.
- 장점 2) 컨텍스트 획득 방식이 변경되어도 애플리케이션 수정없이 적용되는 디자인 패턴의 중재자 클래스만 수정하면 된다.
- 장점 3) 본 논문에서 제시된 패턴을 이용하여 플랫폼 형태로 개발할 경우, 여러 애플리케이션들이 플랫폼을 통해 획득된 컨텍스트를 공유할 수 있다. 이에 따라 배터리와 네트워크 등의 자원 사용을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 제 3장에서는 사물인터넷 장비의 컨텍스트 획득 기법들을 분류하고 애플리케이션과의 비호환성 문제를 분석한다. 제 4장과 5장에서는 이러한 비호환성 문제를 해결하기 위한 대표적인 두 개의 디자인 패턴을 제시한다. 각 패턴은 구조적 뷰, 주요 알고리즘, 동적 뷰, 적용 사례로 상세히 설명된다.

2. 관련 연구

Niculescu의 연구에서는 기존 센서 네트워크 분야에서 사용되는 세 가지 센서 통신 패러다임을 분류하고, 각 패터다임별 적합한 적용 환경에 대한 가이드라인을 제시한다[3]. 그러나 이 연구에서 제시한 가이드라인은 각 통신 패러다임 간의 호환성 등을 포함하지 않는다.

Perera의 연구에서는 스마트 시티에서의 컨텍스트 수집 및 제공을 위한 4계층의 센서 데이터 수집 서비스(Sensing as a Service) 모델을 제시한다[4]. 이 중, 컨텍스트 수집 계층에 해당하는 센서 및 센서 소유자 계층은 센서들을 분류하고, 데이터를 관리한다. 그러나 해당 논문은 컨텍스트 수집하는 방법에 대해서는 자세히 다루지 않는다.

Chen의 연구에서는 RFID와 Zigbee 센서, GPS 장비, 온도 센서, 조도 센서 등 다양한 장비들로부터 컨텍스트 중계(Context Broker) 컴포넌트를 활용하여 컨텍스트를 수집하고 이를 사물인터넷 애플리케이션에 제공하는 플랫폼을 제안한다[5]. 해당 플랫폼은 수집된 컨텍스트를 온톨로지(Ontology)를 활용하여 표현하고 이를 개방화된 게이트웨이 서비스 계획(Open Service Gateway initiative, OSGi) 기반의 미들웨어를 통해 사물인터넷 애플리케이션이 원하는 컨텍스트를 제공한다. 그러나 해당 연구는 컨텍스트 수집 시 네트워크 이질성만 고려하였으며 컨텍스트의 효율적인 제공에 좀더 초점을 맞추고 있다.

Hong의 연구에서는 자원 지향 아키텍처와 서비스 지향 아키텍처를 기반으로 한 이질적인 사물인터넷 장비를 위한 최적의 아키텍처를 제안한다[6]. 그리고, 이질적인 장비 연결을 위해 동일한 접근을 제공하는 미들웨어 프레임워크를 구성하였다. 그러나 제안된 프레임워크는 RESTful을 활용하여 센서 데이터를 수집한다고만 되어있을 뿐, 이를 자세히 다루지는 않는다.

그 외에 사물인터넷 장비 간의 이질성을 해결하여 컨텍스트를 획득하고 제공하는 프레임워크에 대한 연구들이 있다[7-11]. 그러나 이러한 연구들은 이질적인 장비들과의 상호연동을 통한 장비의 기능성 제공과 장비 연결 관점에서의 네트워크 이질성만을 다루며 컨텍스트 획득 방식에 대한 비호환성 이슈는 다루지 않는다.

3. 사물인터넷 컨텍스트 획득 기법의 종류 및 비호환성

3.1 사물인터넷 컨텍스트 획득 기법의 종류

데이터 및 컨텍스트 획득 방법에 대한 표준 분류 체계가 아직은 없다. 데이터 획득 방법 관련 문헌[3, 12]과 현재 사용 가능한 사물인터넷 장비들을 조사하여, 대표적인 데이터 획득 기법 종류를 Fig. 1과 같이 도출하였다.

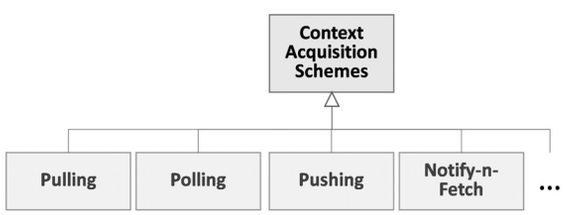


Fig. 1. Types of Acquisition Schemes

풀링(Pulling) 방법은 사물인터넷 애플리케이션의 데이터 요청에 따라 사물인터넷 장비가 요청된 데이터를 전달하는 방식이다.

폴링(Polling) 방법은 풀링 방법과 유사하지만 사물인터넷 애플리케이션이 주기적으로 사물인터넷 장비에게 데이터 요청을 보낸다는 것이 다르다. 즉 풀링은 일회성 요청인데 반해, 폴링은 정해진 일정대로 주기적으로 데이터를 요청한다.

푸싱(Pushing) 방법은 Publish/Subscribe 방식으로도 불리며 사물인터넷 애플리케이션이 사물인터넷 장비에 구독을 요청하면 장비는 구독을 요청한 애플리케이션에게 특정 이벤트 발생 때마다 수집한 데이터를 전달하는 방식이다.

통지-획득(Notify-n-Fetch) 방법은 푸싱 방법과 유사하지만 특정 이벤트 발생 시 사물인터넷 장비는 구독을 요청한 사물인터넷 애플리케이션에게 바로 데이터를 보내지 않고 해당 이벤트에 대한 정보만 전달한다. 이후 이벤트 정보를 전달받은 애플리케이션은 원하는 시점에 해당 이벤트에 대한 컨텍스트 정보를 요청하여 장비로부터 전달받는다.

각 획득 방법들 간에는 사물인터넷 애플리케이션이 데이터를 필요로 하는 시점, 사물인터넷 애플리케이션이 새롭게 생성된 데이터를 전송하는 시점, 데이터를 전송하는 절차

등에서 상당한 차이점이 있다. 예를 들면, 풀링 방법과 통지-획득 방법 간에는 데이터를 전송하는 절차와 데이터를 요청하는 방법이 다르다.

3.2 획득 기법 간 비호환성 이슈

컨텍스트 획득 기법 간 비호환성 문제는 사물인터넷 애플리케이션이 요구하는 컨텍스트 획득 기법과 사물인터넷 장비가 제공하는 컨텍스트 획득 기법이 불일치할 때 발생한다.

이 비호환성 문제를 해결하지 못한다면, 사물인터넷 애플리케이션은 요구사항을 완전히 충족시키는 소수의 장비만 상호작용하며, 요구사항과 일치하는 장비가 없다면 사물인터넷 애플리케이션은 그 기능성을 사용자에게 제공할 수 없게 된다. 예를 들어, 사물인터넷 애플리케이션은 풀링 방식으로 데이터를 획득하여 사용자에게 기능성을 제공해야 하지만, 현재 가용한 사물인터넷 장비가 푸싱 방식만을 지원하여 센서 데이터를 반환하는 경우, 해당 장비를 그대로 사용할 수는 없게 되고, 이를 대체할 수 있는 다른 장비를 찾지 못한다면 사물인터넷 애플리케이션 개발이 어려워지게 된다. 그러므로 사물인터넷 애플리케이션 개발 시 비호환성 문제를 해결해야 한다.

Table 1은 사물인터넷 애플리케이션과 사물인터넷 장비 간에 발생할 수 있는 비호환성 종류들을 보여준다. 이 테이블은 풀링, 푸싱, 통지-획득의 세 가지 방식에 비호환성만 다루고 있으나, 다른 컨텍스트 획득 방식도 유사하게 비호환성을 도출할 수 있다. 세 가지 데이터 획득 방법 간의 비호환성 종류는 Table 1과 같이 총 6 가지로 나타나며, 마찬가지로 각 비호환성 종류에 따라 중재자 종류도 6 가지로 나타난다.

비호환성(Incompatibility)열에 기입된 A-2-B 표기법은 사물인터넷 애플리케이션은 A 방법으로 데이터를 얻기를 원하지만, 사물인터넷 장비는 B 방법만을 지원하는 경우에 발생하는 비호환성 종류를 의미한다. 어떤 중재자 유형(Mitigator Type)을 선택할 지는 사물인터넷 애플리케이션의 컨텍스트 획득 유형 (A)와 사물인터넷 장비의 컨텍스트 획득 유형 (B)에 의해 테이블의 중재자 유형(Mitigator Type)열에 기입된 바와 같이 결정된다. 예를 들어, Pull-2-Push 비호환성 유형은 사물인터넷 애플리케이션이 풀링 방식으로 데이터를 획득하길 원하지만, 사물인터넷 장비는 푸싱 방식만으로 센서 데이터를 전달하는 경우에 발생하며, 이 때 푸싱 방법은 풀링 방법으로 변환되어야 한다.

Table 1. Incompatibility of Data Acquisition Schemes

Scheme Required by IoT Application	Scheme Supported by IoT Device	Incompati-bility	Mitigator Type
Pulling	Pulling	-	-
	Pushing	*Pull-2-Push*	*Pull-2-Push Mitigator*
	Notify-n-Fetch	*Pull-2-Notify*	*Pull-2-Notify Mitigator*
Pushing	Pushing	-	
	Pulling	*Push-2-Pull*	*Push-2-Pull Mitigator*
	Notify-n-Fetch	*Push-2-Notify*	*Push-2-Notify Mitigator*
Notify-n-Fetch	Notify-n-Fetch	-	
	Pulling	*Notify-2-Pull*	*Notify-2-Pull Mitigator*
	Pushing	*Notify-2-Push*	*Notify-2-Push Mitigator*

4. Pull-2-Push 이질성 중재 패턴

4.1 패턴의 구조적 뷰

이 패턴은 풀링 방식을 요구하는 사물인터넷 애플리케이션이 푸싱 방식을 지원하는 사물인터넷 장비를 사용할 경우에 나타나는 비호환성을 해결한다. 이 패턴에서 가장 핵심적인 부분은 사물인터넷 장비 부분과 사물인터넷 애플리케이션 부분을 수정하지 않고, 사물인터넷 장비로부터 값이 변경될 때마다 전달되는 데이터를 사물인터넷 애플리케이션이 원할 때마다 전달하도록 수정하는 것이다. 이를 위해, 사물인터넷 장비와 사물인터넷 애플리케이션 부분에 데이터 획득 방식을 변경하는 중재 클래스인 Pull-2-PushMitigator 클래스를 Fig. 2와 같이 위치시킨다. 사물인터넷 장비는 푸싱 방식으로 데이터를 전달하기 때문에, 관찰자(Observer) 패턴[13]을 이용하여 Pull-2-PushMitigator 클래스를 설계한다.

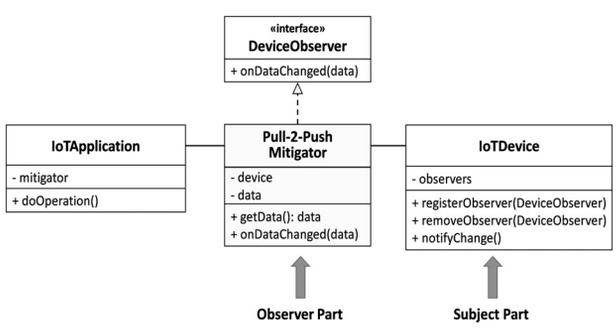


Fig. 2. Class Diagram for Mitigating Pull-2-Push Incompatibility

IoTApplication 클래스는 사물인터넷 장비부터 획득된 데이터를 이용하여 사용자에게 서비스를 제공하는 클래스이다.

IoTDevice 클래스는 수집된 데이터를 전달하는 클래스로, 푸싱 방식으로 새로 획득된 데이터를 전달하기 때문에 구독 신청 및 구독 해지를 수행하는 메소드와 새로운 값이 획득될 때마다 실행되는 메소드를 포함한다. 사물인터넷 장비마다 다른 메소드 시그니처로 이 기능들을 제공할 것이며, 위의 그림에서는 registerObserver( ), removeObserver( ), notifyChange( ) 메소드들이 각 기능을 수행한다.

Pull-2-PushMitigator 클래스는 Pull-2-Push 이질성을 중재하는 역할을 수행하며, 사물인터넷 애플리케이션을 대신하여 푸싱 방식으로 전달된 데이터를 전송 받기 위한 메소

드인 onDataChanged( )와 사물인터넷 애플리케이션에게 풀링 방식으로 데이터를 전달하기 위한 메소드인 getData( )를 포함하고 있다. 그리고 최신 변경된 데이터를 저장하는 변수인 data를 가지고 있다. Pull-2-PushMitigator 클래스는 사물인터넷 장비에서 변경된 값을 자동적으로 전달받기 위해서 onDataChanged( ) 콜백 (Callback) 메소드만을 가지고 있는 DeviceObserver 인터페이스를 구현해야 하며, 이 인터페이스 이름과 메소드 시그니처 역시 사물인터넷 장비 제조자쪽에서 제공되는 이름으로 바뀔 수 있다.

4.2 패턴의 주요 알고리즘

Pull-2-Push 이질성을 중재하기 위해 Pull-2-PushMitigator 클래스는 IoTDevice 클래스로부터 지속적으로 데이터를 받아 이를 data 변수에 저장해야 한다. List 1은 값이 변경될 때마다 호출되는 notifyChange( ) 메소드의 내부 알고리즘을 보여준다. 이 알고리즘은 관찰자 패턴의 알고리즘에서 변경된 값을 입력 매개변수로 전달할 수 있도록 수정한 것이다.

List 1. Algorithm of notifyChange( ) method defined in 'IoT Device Class'

```
1  void notifyChange( ) {
2      for   all o in observers {
3       o.onDataChanged(data)
4      }
5  }
```

3번째 줄에서와 같이 자신을 구독한 DeviceObserver 인터페이스의 메소드를 구현한 모든 클래스에 onDataChanged( ) 메소드의 입력 매개변수로 변경된 값을 전달한다. DeviceObserver 인터페이스를 구현한 Pull-2-PushMitigator 클래스는 onDataChanged( ) 메소드로 전달된 값을 추후에 IoTApplication 클래스가 요청할 때 전달하기 위해 이를 data 변수에 저장한다.

4.3 패턴의 동적 뷰

Fig. 3은 Pull-2-Push 이질성 중재 패턴에 포함되는 클래스들 간의 상호작용 관계를 보여줌으로써, 비호환성 문제가 어떻게 해결될 수 있는지를 표현하고 있다.

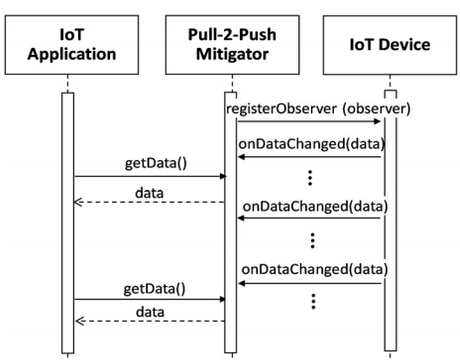


Fig. 3. Sequence Diagram for Mitigating Pull-2-Push Incompatibility

Pull-2-PushMitigator 클래스는 IoTApplication 클래스와 IoTDevice 클래스사이의 에이전트 역할을 수행한다. 먼저 Pull-2-PushMitigator 클래스는 IoTDevice 클래스에 registerObserver( ) 메소드를 호출하여 IoTApplication 클래스 대신에 IoTDevice 클래스에 구독을 신청한다. 이후 Pull-2-PushMitigator 클래스는 onDataChanged( ) 메소드를 통해 IoTDevice 클래스에 새롭게 생성된 데이터를 전달 받고, 이를 data 변수에 임시 저장한다. IoTApplication 클래스는 필요한 시점에 getData( ) 메소드를 통해 Pull-2-PushMitigator 클래스로부터 IoTDevice 클래스의 최근 데이터를 얻어서 기능을 수행한다.

이렇듯, Pull-2-PushMitigator 클래스가 IoTApplication 클래스를 대신하여 계속적으로 변경된 데이터를 획득하고, IoTApplication 클래스가 필요한 시점에 최근 데이터를 얻어갈 수 있도록 getData( ) 메소드를 제공하고 있기 때문에, Pull-2-Push 비호환성 문제가 해결될 수 있는 것이다.

이 경우 Pull-2-PushMitigator 클래스가 계속적으로 변경된 데이터를 받아와 이를 data 변수에 업데이트하는 오버헤드가 발생할 수 있다. 예를 들어 아래의 Fig. 4와 같이 사물인터넷 애플리케이션의 풀링 요청이 n번 있었으며 사물인터넷 장비의 푸싱으로 인한 데이터 업데이트가 m번 있었을 경우 아래의 상황에 따라 오버헤드가 발생할 수 있다.

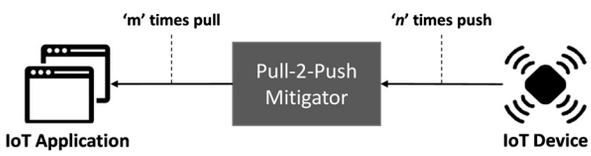


Fig. 4. An Illustration of Mitigating Pull-2-Push Incompatibility

- n > m일 경우 오히려 오버헤드는 감소한다.
- n < m 일 경우 비호환성 중재에 따른 오버헤드가 발생한다.

후자의 경우 오버헤드를 개선하기 위한 전략으로 Pull-2-PushMitigator 클래스를 서버와 같이 분리된 장소에 구현하여 사물인터넷 애플리케이션 성능이 영향을 받지 않도록 구현하는 등의 방법이 있다.

4.4 패턴의 적용 사례

본 절에서는 Pull-2-Push 이질성 중재 패턴을 적용하여 뇌파 센서를 이용한 집중도 분석 안드로이드 애플리케이션 (AttentionAnalyzer) 개발 적용 사례를 설명한다. 사용되는 사물인터넷 장비는 NeuroSky의 Mindwave이며 측정된 뇌파는 파워 스펙트럼 분석 기법[14]을 사용하여 분석한다.

AttentionAnalyzer는 사용자의 분석 요청에 따라 풀링 방식으로 획득한 뇌파를 분석한다. 이에 반해 NeuroSky에서 제공하는 Mindwave 안드로이드 API는 푸싱 방식의 컨텍스트 획득 구조로 되어 있다[15]. 아래의 List 2와 List 3은 NeuroSky에서 제공하는 개발 가이드 문서 중 일부분이다.

List 2. Setting up the TGDevice

```
1  btAdapter =   BluetoothAdapter.getDefaultAdapter();
2  if(btAdapter   != null){
3   tgDevice    = new TGDevice(btAdapter, handler);
4   }
5  }
```

List 3. Handling Data Receipt

```
1  private final Handler handler = new Handler(){
2  public void   handleMessage(Message msg){
3  switch(msg.what){
4  case   TGDevice.MSG_STATE_CHANGE:
5  …
6  …
```

NeuroSky는 자사 제품을 하나의 클래스로 관리할 목적으로 TGDevice라는 클래스를 제공한다. List 2의 3번째 줄을 보면 TGDevice 클래스 객체 생성시 handler 객체를 등록한다. 이 handler 객체는 List 3의 핸들러 구현 가이드 라인을 통해 구현된 것으로, 2번째 줄의 handleMessage( )에서 센서 데이터를 푸싱 방식으로 전달 받는다.

이와 같이 AttentionAnalyzer와 Mindwave 간에는 Pull-2-Push 이질성이 발생한다. 이를 위해 Pull-2-Push 이질성 중재 패턴을 적용하여 Fig. 5와 같이 설계하였다.

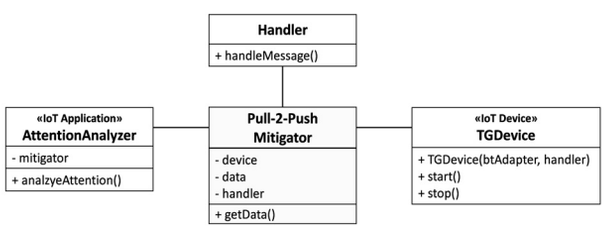


Fig. 5. Result of Applying Design Pattern for Mitigating Pull-2-Push Incompatibility

Pull-2-PushMitigator 클래스는 NeuroSky에서 제공하는 TGDevice 클래스의 객체와 제공된 개발 가이드라인을 준수하여 구현한 핸들러 객체를 가지고 있으며, 가장 최근의 수집된 컨텍스트를 애플리케이션에 제공하기 위한 변수 data를 갖는다. 또한 이 클래스는 AttentionAnalyzer 클래스에 풀링 방식을 지원하기 위한 getData( ) 메소드를 포함한다.

IoTDevice 클래스인 TGDevice 클래스는 객체 생성 시 핸들러 객체를 등록하며, 컨텍스트 수집의 시작과 중지를 위한 start( )과 stop( ) 메소드를 포함한다.

AttentionAnalyzer 클래스는 풀링 방식으로 컨텍스트를 수집하기 위해 Pull-2-PushMitigator 클래스 객체를 가지고 있으며 집중도를 분석하기 위한 analyzeAttention( ) 메소드를 가지고 있다. 4.1절의 구조적 뷰와 비교하여 DeviceObserver 대신 핸들러를 사용하는 등 부분적인 변화는 있지만 전체적인 구조는 동일하다.

List 4는 Pull-2-PushMitigator 클래스의 코드 일부이다.

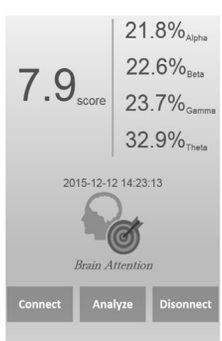
List 4. Implementation of Pull2PushMitigator Class

```
1  public class Pull2PushMitigator   {
2     TGDevice device;
3     BluetoothAdapter btAdpater;
4     …
5     public Pull2PushMitigator   (){
6        …
7        tgDevice = new TGDevice(btAdapter, this.handler);
8        tgDevice.connect(false)
9        tgDevice.start()
10    }
11    public Object getData(){
12       return data;
13    }
14    private Handler handler = new Handler(){
15       @Override
16       public void handleMessage(Message msg){
17       …
18          if(msg.what == TGDevice.MSG_EEG_POWER){
19             data = msg.obj;
20             …
25 }
```

핸들러 객체는 14번째 줄과 같이 구현되어 있으며, 7번째 줄에서 TGDevice 클래스 객체를 생성시 등록한다. 이후 9번째에서 start( ) 메소드를 호출함으로써 핸들러 객체를 통해 컨텍스트를 정기적으로 전달받는다.

AttentionAnalyzer 클래스의 분석 대상인 뇌파 데이터는 19번째 줄에서 data에 저장한다. 이후 11번째 줄의 getData( ) 메소드 호출 시 data를 반환하여 이를 사용하는 AttentionAnalyzer 클래스가 가장 최근의 수집된 컨텍스트를 전달받을 수 있도록 한다. Fig. 6은 Pull-2-PushMitigator 클래스 사용하는 AttentionAnalyzer의 화면과 화면의 'Analyze' 버튼과 연결되어 있는 소스 코드이다.

(a)



(b)

```
public void analyze(View v){
    TGEegPower eeg = (TGEegPower)mitigator.getData();
    JSONObject result;
    result = analyzeAttention(eeg);
    try {
        score.setText(String.valueOf(result.getDouble("score")));
        context.setText(result.getString("summary"));
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Fig. 6. Application Screen Dump (a) and Source Code for 'Analyze' Button (b)

Fig. 6(a)의 화면을 보면 AttentionAnalyzer는 3개의 버튼과 분석 결과를 보여주는 부분으로 구성되어 있다. 이중 중앙에 위치한 버튼 'Analyze'를 누르면 Pull-2-PushMitigator 클래스를 통해 연결된 Mindwave로부터 수집된 컨텍스트를 가져와 집중도를 분석한 후 분석 결과를 화면에 보여준다. Fig. 6(b)의 'Analyze'버튼 코드를 보면 Pull-2-PushMitigator 클래스의 객체 mitigator의 getData( ) 메소드를 통해 뇌파 데이터를 전달 받아 analyzeAttention( ) 메소드를 사용하여 뇌파를 분석한 후 결과를 화면에 보여주도록 구현되어 있다.

이와 같이 Pull-2-Push 이질성 중재 패턴을 사용하여, AttentionAnalyzer 클래스는 TGDevice 클래스를 수정하지 않고 Pull-2-PushMitigator 클래스를 통해 폴링 방식으로 컨텍스트를 전달받을 수 있게 되었다.

5. Notify-2-Pull 이질성 중재 패턴

5.1 패턴의 구조적 뷰

이 패턴은 장비로부터 폴링 방식으로 전달되는 센서 데이터를 사물인터넷 애플리케이션에게 통지-획득 방식으로 전달해야 하는 Notify-2-Pull 이질성 문제가 발생할 때 적용된다. 이 중재 패턴에서 가장 핵심적인 부분은 데이터 요청 시에만 새로운 값을 전달하는 장비로부터 사물인터넷 애플리케이션이 데이터 값을 수시로 받을 수 있도록 수정하는 것이다. 이를 위해, 사물인터넷 장비와 사물인터넷 애플리케이션 부분에 데이터 획득 방식을 변경하는 중재 클래스인 Notify2PullMitigator 클래스를 Fig. 7과 같이 위치시킨다. 이 외에 Notify2PullMitigator 클래스를 구독하는 애플리케이션이 구현해야 하는 DeviceObserver 인터페이스 및 Notify2PullMitigator 클래스가 정기적으로 사물인터넷 장비로부터 새 값을 읽어오는 역할을 하는 DataAcquisitionThread를 추가한다. 이전 장의 Pull-2-Push 중재 패턴과 같이 사물인터넷 장비가 변경될 경우 Notify2PullMitigator 클래스만 변경하면 됨으로 IoTApplication 클래스는 장비 변경에 영향을 받지 않는다.

Notify2PullMitigator 클래스는 Notify-2-Pull 이질성을 중재하는 역할을 수행하며, DataAcquisitionThread를 사용하여 사물인터넷 장비로부터 정기적으로 데이터를 읽어온다. 또한

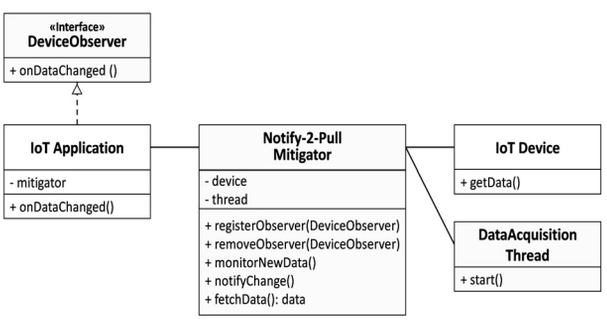


Fig. 7. Class Diagram for Mitigating Notify-2-Pull Incompatibility

이 클래스는 DeviceObserver 등록 및 해제를 위한 메소드 registerObserver( )와 removeObserver( )를 가지고 있으며 통지-획득 방식을 지원하기 위한 monitorNewData( ), notifyChange( ), fetchData( )들을 가지고 있다.

IoTApplication 클래스는 통지-획득 방식으로 컨텍스트를 전달받기 위해 DeviceObserver 인터페이스를 구현한다. IoTDevice 클래스의 getData( ) 메소드는 폴링 방식으로 컨텍스트를 획득하기 위한 것으로 마찬가지로 메소드 시그니처 역시 사물인터넷 장비 제조자에 따라 다를 수 있다.

5.2 패턴의 주요 알고리즘

Notify-2-Pull 이질성을 중재하기 위해 우선 정기적으로 IoTDevice 클래스에게 데이터를 요청해야 한다. List 5는 사물인터넷 장비에게 주기적으로 데이터를 요청하기 위한 DataAcquisitionThread의 run( ) 메소드 내부 알고리즘을 보여준다.

List 5. Algorithm of run() method defined in 'DataAcquisitionThread'

```
1  run(){
2    while(True){
3      data = device.getData()
4      check(data)
5      period   = estimatePeriod(data)
6      time.sleep(period)
7    }
8  }
```

3번째 줄에서 IoTDevice 클래스의 객체 device에 getData( ) 메소드를 호출하여 IoTDevice 클래스의 데이터를 읽어오며, 4번째 줄에서 변경된 컨텍스트를 IoTApplication 클래스에게 알리기 위해 check( ) 콜백 메소드를 호출한다. 그리고, 5번째 줄에서 주기적으로 데이터를 요청하는 오버헤드를 줄이기 위해 데이터 요청 주기를 조절하는 메소드를 호출한다. 이 메소드는 센서 값 변화 경향과 반복성 등을 분석하여 다음 데이터가 변화할 시점을 추정하여 적합한 주기를 반환한다. 조절된 주기만큼 기다린 후 DeviceObserver가 해지 될 때까지 위의 과정을 반복한다.

List 6은 check( ) 콜백 메소드의 내부 알고리즘을 보여준다.

List 6. Algorithm of check() method defined in 'Notify2PullMitigator'

```
1  void check(data){
2    if(isValueChanged(data)){
3      event = Event(Event.ValueChanged, timestamp)
4      dataQueue[event] = data
5      notifyChange(event)
6    }
7    …
8  }
```

컨텍스트 변경 감지는 2번째 줄의 isValueChanged( ) 메소드를 통해 이루어지며, 변경이 감지되면 3번째 줄에서 이벤트에 대한 정보를 지닌 이벤트 객체를 생성한다. 이후 fetchData( ) 요청에 응답하기 위해 이 이벤트에 해당하는 데이터를 4번째 줄과 같이 생성된 이벤트 객체를 키로 사용하여 dataQueue 저장소에 저장된다. 이후 notifyChange( ) 메소드를 통해 생성된 event 객체를 IoTApplication 클래스에 전달한다.

List 7은 IoTApplication 클래스가 선택적으로 데이터를 전달받기 위한 fetchData( )의 내부 알고리즘을 보여준다.

List 7. Algorithm of fetchData() method defined in 'Notify2PullMitigator'

```
1  data fetchData(event){
2     data = dataQueue[event]
3     return data
```

IoTApplication 클래스는 전달된 이벤트 객체를 통해 그 이벤트에 해당하는 데이터를 fetchData( ) 메소드를 호출하여 가져올 수 있다. 2번째 줄과 같이 키로 등록된 이벤트 객체를 통해 저장된 데이터를 읽어와 이를 3번째 줄에서 반환한다. 이런 방법으로 IoTApplication 클래스는 전달된 이벤트 정보를 바탕으로 선택적으로 데이터를 가져오거나 원하는 시간에 이전 이벤트에 대한 데이터를 가져올 수 있다.

5.3 패턴의 동적 뷰

Fig. 8은 Notify-2-Pull 이질성 중재 패턴에 포함되는 클래스들 간의 상호작용 관계를 보여준다.

Notify2PullMitigator 클래스는 IoTDevice 클래스 객체를 가지고 있으며, DeviceObserver 인터페이스가 등록된 후 monitorNewData() 메소드가 호출되면 DataAcquisitionThread 객체를 생성하여 이를 통해 IoTDevice 클래스에 지속적으로 데이터를 요청한다. 이후 DataAcquisitionThread 객체를 통해 전달된 데이터의 센서 값 변경 등 이벤트를 검사하고 어떤 이벤트가 발생했을 경우 이 이벤트에 대한 정보를 담은 이벤트 객체를 만들어 DeviceObserver 인터페이스를 통해 IoTApplication 클래스에 전달한다. IoTApplication 클래스 해당 이벤트에 대한 데이터를 얻고자 할 경우 Notify2PullMitigator 클래스로부터 전달된 event 객체를 매개변수로 하여 fetchData( ) 메소드를 호출하여 데이터를 획득한다.

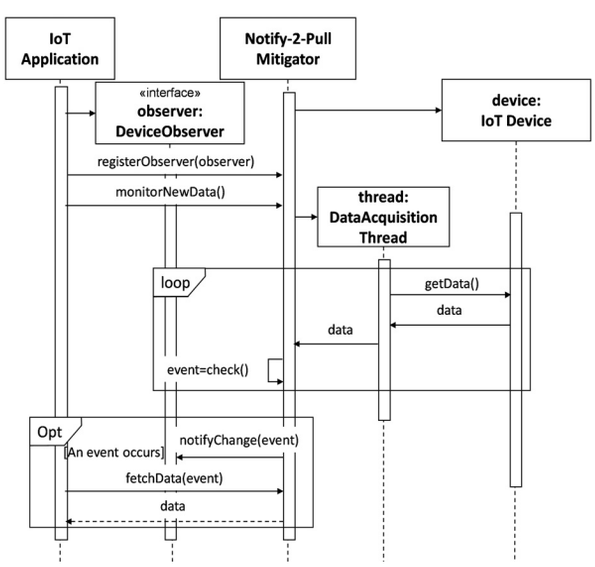


Fig. 8. Sequence Diagram for Mitigating Notify-2-Pull Incompatibility

5.4 패턴의 적용 사례

본 절에서는 Notify-2-Pull 패턴의 근전도(Electromyogram, EMG)센서를 이용한 근긴장, 근육 피로도 정보를 분석하는 EMGAnalyzer 애플리케이션 개발 적용 사례를 설명한다. 사용되는 사물인터넷 장비는 Coocking Hacks의 e-Health Sensor Platform Complete Kit 중 EMG센서와 Raspberry Pi 2가 사용된다[16].

EMGAnalyzer는 일정 시간 동안 측정된 근전도 정보를 RMS (Root Mean Square), Median Frequency 분석 기법[17, 18]을 통해 근긴장, 근육 피로도 정보를 분석하여 이를 사용자에게 보여준다. 두 분석 기법 모두 일정시간 측정된 근전도 정보를 이용하기 때문에, 통지-획득 방식으로 데이터를 수집하는 것이 효율적이다. 이에 반해 Raspberry Pi용 e-Health Kit 라이브러리는 풀링 방식의 컨텍스트 획득 구조로 되어 있다[16]. List 8은 근전도 센서 값을 읽어오는 부분(3번째 줄)을 보여준다.

List 8. e-Health Library for Raspberry Pi

```
1  int eHealthClass::getEMG(void){
2     int analog0;
3     analog0=analogRead(0);
4     return analog0;
5  }
```

이와 같이 EMGAnalyzer와 e-Health Sensor Platform Complete Kit의 EMG 센서 간에는 Notify-2-Pull 이질성이 발생한다. 이를 위해 Notify-2-Pull 이질성 중재 패턴을 적용하여 Fig. 9와 같이 클래스 다이어그램을 설계하였다.

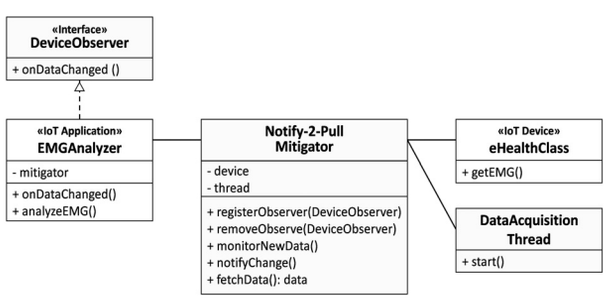


Fig. 9. Result of Applying Design Pattern for Mitigating Notify-2-Pull Incompatibility

IoTDevice 클래스 부분에 eHealthClass 클래스가 위치하여 데이터를 요청하는 메소드의 시그네처가 변한 것과 IoTApplication 클래스 부분에 EMGAnalyzer 클래스가 위치한 것 이외에는 5.1의 구조적 뷰와 동일한 구조이다.

이와 같은 구조로 구현된 Notify2PullMitigator 클래스와 DataAcquisitionThread는 아래의 List 9와 같다.

List 9. Implementation of Notify2PullMitigator Class and DataAcquisitionThread

```
1  class Notify2PullMitigator:
2      …
3      def registerObserver(self, observer):
4          self.observer = observer
5      …
6      def monitorNewData(self):
7          DataAcquisitionThread(self.device, self.check).start()
8      …
9      def check(self, data):
10         if isValueChanged(data):
11             event = Event(Event.ValueChanged, timestamp)
12             self.dataQueue[event] = data
13             notifyChange(event)
14         …
15     def fetchData(self, event):
16         data = self.dataQueue[event]
17         return data
18
19 class DataAcquisitionThread(Threading.Thread):
20     …
21     def run(self):
22         while(True):
23             data = device.getEMG()
24             self.check(data)
25             period = estimatePeriod(data)
26             time.sleep(period)
```

7번째 줄에서 Notify2PullMitigator 클래스가 가지고 있는 device 객체와 DataAcquisitionThread로부터 데이터를 받을 콜백 메소드인 check()를 등록하여 start() 메소드를 통해 주기적인 데이터 요청을 시작한다. DataAcquisitionThread는 23번째 줄과 같이 getEMG() 메소드를 호출하여 데이터를 읽어오며 이를 등록된 콜백 메소드에 전달한다.

9번째 줄의 check()함수는 컨텍스트 변경 여부를 확인하여 변경이 발생했을 경우 이벤트 객체를 생성하여 12번째 줄과 같이 생성된 객체를 키로 하여 데이터를 저장하고 이벤트 객체를 DeviceObserver 인터페이스를 등록한 EMGAnalyzer 클래스에 전달한다. 이후 15번째 줄의 fetchData() 메소드가 호출되면 매개변수로 전달된 이벤트 객체를 키로 사용하여 데이터를 찾아 이를 반환하다.

Fig. 10은 EMGAnalyzer의 DeviceObserver 인터페이스의 onDataChanged( ) 메소드 구현을 보여준다.

```
def onDataChanged(self, event):
    if self.start is None:
        self.start = event.timestamp()
        event_list = []
    event_list.append(event)
    print(event.toString())

    if event.timestamp() - self.start > self.three_minute:
        self.mitigator.removeObserver(self.observer)
        emg_list = []
        for event in event_list:
            emg_list.append(self.mitigator.fetchData(event))
            result = self.analyzeEMG(emg_list)
            print(result)
```

Fig. 10. Source Implementing DeviceObserver Interface of EMGAnalyzer

onDataChanged( ) 메소드는 Notify2PullMitigator 클래스로부터 이벤트 객체를 받아와 3분 동안 근전도를 측정하고, fetchData( ) 메소드를 호출하여 획득된 데이터를 분석하도록 구현되어 있다. Fig. 11은 EMGAnalyzer 실행 화면을 보여준다.

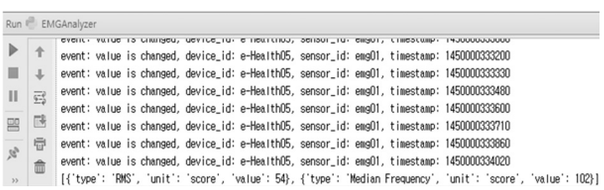


Fig. 11. Execution Result Screen Dump of EMGAnalyzer

이와 같이 Notify2Pull 이질성 중재 패턴을 사용하여, EMGAnalyzer는 eHealthClass를 수정하지 않고 Notify2 PullMitigator 클래스를 통해 통지-획득 방식으로 컨텍스트를 전달받을 수 있게 되었다.

6. 평 가

제 4장과 5장에서 제시한 적용 사례에서, 본 논문이 제시한 컨텍스트 비호환성 중재 패턴을 통해 각각의 사물인터넷 애플리케이션이 원하는 컨텍스트 획득 방식을 지원하는 것을 보았다. 특히 IoTDevice 클래스 변경이 사물인터넷 애플리케이션에 영향을 미치지 않음을 알 수 있다.

또한 컨텍스트 비호환성 중재 패턴을 확장하여 플랫폼화 할 시 다음과 같은 기대 효과를 얻을 수 있다.

- 하나의 사물인터넷 장비를 사용하여 다양한 컨텍스트 획득 패턴의 사물인터넷 애플리케이션을 지원할 수 있다.
- 여러 사물인터넷 애플리케이션이 하나의 사물인터넷 장비를 공유함으로써 배터리 및 네트워크 등의 자원을 최소화할 수 있다.

Fig. 12는 본 논문에서 제시한 중재 패턴을 확장하여 플랫폼화 하였을 시 다양한 유형의 사물인터넷 애플리케이션들이 하나의 장비를 공유하는 것을 나타낸 것이다.

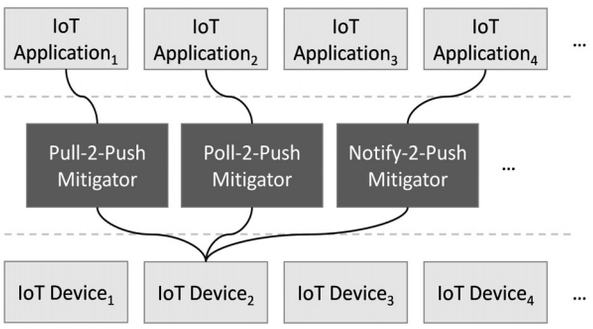


Fig. 12. An Illustration for Sharing an IoT Device with Different Types of IoT Application

IoT Application1은 풀링 방식, IoT Application2는 폴링 방식, IoT Application4는 통지-획득 방식의 사물인터넷 애플리케이션이며 하나의 푸싱 방식을 지원하는 사물인터넷 장비 IoT Device2를 공유하고 있다. 각각의 사물인터넷 애플리케이션은 비호환성 유형에 맞는 중재 클래스들을 사용하여 하나의 사물인터넷 장비로부터 각각의 컨텍스트 획득 유형에 맞게 데이터를 수집할 수 있다.

또한 위의 경우 한번 데이터 수집을 통해 여러 사물인터넷 애플리케이션이 데이터를 공유함으로써 배터리 및 네트워크 등의 자원을 절약할 수 있다.

7. 결 론

현재 가용한 사물인터넷 장비는 풀링, 푸싱 등 여러 컨텍스트 획득 방법 중 특정한 하나의 방식만을 지원한다. 따라서, 애플리케이션이 필요로 하는 컨텍스트 획득 방식과 장비가 지원하는 획득 방식이 일치하지 않는 경우에 개발 노력의 증가, 향후 유지보수 비효율성 등의 문제가 발생한다.

이를 효과적으로 해결하기 위해, 본 논문에서는 애플리케이션과 장비간의 컨텍스트 획득방식 비호환성을 효과적으로 해결하기 위한 디자인 패턴들을 제시하였다. 먼저, 대표적인 사물인터넷 컨텍스트 회득 방법을 기술하고, 이들간 6가지 발생 가능한 비호환성 이슈를 도출하였다. 도출된 6가지 중 Pull-2-Push 이질성과 Notify-n-Pull 이질성 문제를 해결하는 디자인 패턴을 구조적 뷰, 알고리즘, 동적 뷰, 적용 사례 관점으로 기술하였다.

본 논문에서 제시하지 않은 이질성 문제는 이 두 가지 패턴을 활용하여 쉽게 구현할 수 있다. 예를 들어, Pull-2-Notify 이질성은 Pull-2-Push 이질성 중재 패턴에 속한 Pull-2-PushMitigator 클래스의 onDataChanged (data) 오퍼레이션을 onDataChanged( )로 변경하고, 내부 알고리즘에서 fetch 방식으로 값을 읽어오도록 변경하면 된다.

제시된 패턴들을 적용하면, 비호환성이 발생하는 사물인터넷 애플리케이션을 보다 체계적이며, 효과적으로 개발할 수 있고, 나아가 향후 유지보수의 효율성도 증가될 것으로 기대된다.

References

[1] S. Haller, S. Karnouskos, and C. Schroth, “The Internet of Things in an Enterprise Context,” *FUTURE INTERNET - FIS 2008*, Vol.5468, pp.14-28, 2009.

[2] ITU Internet Reports 2005: The Internet of Things [Internet], https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf (downloaded 2016, Aug., 9).

[3] D. Niculescu, “Communication paradigms for sensor networks,” *IEEE Communications Magazine*, Vol.43, No.3, pp.116-122, 2005.

[4] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Sensing as a service model for smart cities supported by Internet of Things,” *Transactions on Emerging Telecommunications Technologies*, Vol.25, No.1, pp.81-93, 2014.

[5] Y.-S. Chen and Y.-R. Chen, “Context-oriented Data Acquisition and Integration Platform for Internet of Things,” in *Proceedings of 2012 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2012)*, pp. 103-108, Nov., 2012.

[6] Y. Hong, “A Resource-Oriented Middleware Framework for Heterogeneous Internet of Things,” in *Proceedings of 2012 International Conference on Cloud Computing and Service Computing (CSC 2012)*, pp.12-16, Nov., 2012.

[7] Z. Haibo, “A Framework to Enable Communication in Heterogeneous Environment for the Internet of Things,” *Journal of Computational Information Systems*, Vol.8, No.18, pp.7791-7798, 2012.

[8] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, “Smart Objects as Building Blocks for the Internet of Things,” *IEEE Internet Computing*, Vol.14, No.1, pp.44-51, Jan.-Feb., 2010.

[9] T. S. Lopez, D. C. Ranasinghe, M. Harrison, and D. McFarlane, “Adding Sense to the Internet of Things,” *Personal and Ubiquitous Computing*, Vol.16, No.3, pp. 291-308, Mar., 2012.

[10] A. Gomez-Goiri and D. Lopez-de-Ipina, “A Triple Space-based Semantic Distributed Middleware for Internet of Things,” in *Proceedings of the 10th International Conference on Current trends in Web Engineering (ICWE 2010), Lecture Notes in Computer Science (LNCS)*, Vol.6385, pp.447-458, Jul., 2010.

[11] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, “Smart Semantics Middleware for the Internet of Things,” in *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization (ICINCO 2008)*, pp.169-178, May, 2008.

[12] F. Buschmann, K. Henny, and D. S. Schmidt, “Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing,” Wiley, Apr., 2007.

[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley Professionals, Nov., 1994.
[14] LAXTHA Inc., [Post Analysis - Spectrum] Analyzing Power Spectrum of Brainwave Data [Internet], http://www.laxtha.com/SiteView.asp?x=7&y=32&z=33&infid=154.
[15] NeuroSky, Developer tools [Internet], http://store.neurosky.com/collections/developer-tools.
[16] Cooking hacks, E-health sensor platform V2.0 for Arduino and raspberry pi [Biometric / medical Applications] [Internet], https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical/.
[17] LAXTHA Inc., [Post Analysis - Muscular Contraction] Overview of EMG Amplitude Analysis Method (Muscular Contraction, Muscle Fatigue) [Internet], http://www.laxtha.com/SiteView.asp?x=7&y=46&z=41&infid=176.
[18] LAXTHA Inc., [Post Analysis - Muscle Fatigue] Overview of EMG Frequency Analysis Method (Muscle Fatigue) [Internet], http://www.laxtha.com/SiteView.asp?x=7&y=46&z=41&infid=177.

라 현 정

e-mail : hjla80@gmail.com
2003년 경희대학교 전자정보학부(학사)
2006년 숭실대학교 컴퓨터학과(석사)
2011년 숭실대학교 컴퓨터학과(박사)
2011년~2013년 숭실대학교 모바일 서비스 소프트웨어공학센터 연구교수
2013년~현 재 ㈜스마티랩 대표
관심분야 : 소프트웨어 아키텍처(Software Architecture), 모바일 클라우드 컴퓨팅(Mobile Cloud Computing), 사물 인터넷 컴퓨팅(Internet of Things Computing)

안 구 환

e-mail : usimebul@gmail.com
2014년 학점은행제 컴퓨터공학 전공(학사)
2015년~현 재 숭실대학교 컴퓨터학과 석사과정
관심분야 : 모바일 클라우드 컴퓨팅(Mobile Cloud Computing), 사물 인터넷 컴퓨팅(Internet of Things Computing)

김 수 동

e-mail : sdkim777@gmail.com
1984년 Northeast Missouri State University 전산학(학사)
1988년 The University of lowa 전산학(석사)
1991년 The University of lowa 전산학(박사)
1991년~1993년 한국통신 연구개발단 선임연구원
1994년~1995년 현대전자 소프트웨어연구소 책임연구원
1995년~현 재 숭실대학교 컴퓨터학부 교수
관심분야 : 객체지향 모델링(Object-Oriented Modeling), 소프트웨어 아키텍처(Software Architecture), 컨텍스트 인지 서비스(Context-Aware Service), 사물 인터넷 컴퓨팅(Internet of Things Computing)