ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

# IoT 컴퓨팅의 실용적 결함 관리 기법☆

# Practical Methods for Managing Faults in IoT Computing

박 춘 우[1] 김 수 동[1*]
Chun Woo Park Soo Dong Kim

**요 약**

최근 주목 받고 있는 정보기술분야 중 하나인 IoT(Internet of Things) 환경은 센서와 액츄에이터로 구성된 다양한 디바이스들이 네트워크에 연결되어 정보를 수집 및 공유하면서 상호작용이 가능한 컴퓨팅 환경이다. 하드웨어, 네트워크 기술의 발전으로 인한 IoT 디바이스의 보편화와 IoT 서비스에 대한 사회적인 요구의 증가에 따라, IoT 애플리케이션의 사용성이 증대될 것으로 기대하고 있다. 그러나 IoT 환경에서는 전통적인 소프트웨어 연구에서 다루어지지 않았던 다양한 종류의 결함이 발생할 수 있어, 신뢰성 있는 IoT 애플리케이션을 개발하는 데에 어려움이 있다. 이러한 문제를 해결하기 위하여, 본 논문에서는 IoT 결함을 분류하고, 분류된 결함들의 원인과 증상을 분석한다. 그리고 분석 결과를 기반으로 서비스 실행 시에 발생 할 수 있는 결함을 관리하기 위한 프로세스와 그 프로세스 안에서 수행되는 기법을 제안한다. 본 연구에서 제안하는 프로세스와 관리 기법을 활용하여 결함 관리가 필요한 IoT 애플리케이션의 개발 비용을 줄일 수 있다.

☞ 주제어 : 사물인터넷, IoT 컴퓨팅 환경, IoT 결함, 결함 관리 프로세스, 결함 관리 기법

**ABSTRACT**

Internet of Thing (IoT) computing is an environment where various devices with sensors and actuators are connect, and interact together to acquire contexts and provide useful services. With the advances of IoT technologies, its usability becomes an in important issue. However, there exist various types of faults in IoT computing which are not conventionally addressed in software research community. Providing reliable IoT services is challenging. In this paper, we present a hierarchy of IoT faults and analyze causes and symptoms of the faults. Based on the analysis, we define effective methods for managing IoT faults. We believe that our proposed framework for managing IoT faults can be utilized in reducing the development cost of IoT applications and enhancing the quality of the applications.

☞ keyword : Internet of Things (IoT), IoT Computing Environment, IoT Fault, Fault Management Process, Fault Management Method

## 1. 서 론

최근 주목 받고 있는 정보기술분야 중 하나인 IoT (Internet of Things) 환경은 센서와 액츄에이터로 구성된 다양한 디바이스들이 네트워크에 연결되어 정보를 수집 및 공유하면서 상호작용이 가능한 컴퓨팅 환경이다 [1]. IoT 디바이스들을 통해 수집된 다양한 정보들과 이들을 분석한 결과를 기반으로, 소비자들에게 보다 편리한 서비스를 제공해 줄 수 있다 [2]. 하드웨어, 네트워크 기술의 발전으로 인한 IoT 디바이스의 보편화와 IoT 서비스에 대한 사회적인 요구의 증가에 따라, IoT 컴퓨팅의 중요성이 더욱 커질 것으로 예상되고 있다 [3][4][5].

IoT 환경은 다수의 IoT 디바이스로 구성되며, 대부분의 IoT 디바이스들은 와이파이, 블루투스와 같은 무선 네트워크에 연결된다. 이러한 IoT 디바이스들은 주변 환경의 변화에 민감하게 반응한다. 이와 같이, 기존의 소프트웨어와는 다른 IoT 환경의 특성들로 인해 기존의 연구에서 다루어지지 않았던 다양한 종류의 결함이 발생 할 수 있다 [6]. 예를 들어, IoT 디바이스가 네트워크 범위 밖으로 이동하여 해당 디바이스와의 연결이 끊어 질 수 있다. 그리고 외부 환경 변화로 인하여 IoT 디바이스가 파손이 되거나 오작동을 할 가능성도 있다. 이러한 결함들을 해결하기 위한 연구들이 활발하게 진행 중에 있다 [7][8][9]. 그러나, 대부분의 연구들은 IoT 환경의 다양한 결함 중 일부의 결함만을 고려하거나, 구체적인 기법 설명이 미흡하다.

[1] Department of Computer Science, Soongsil University, Seoul, 156-743, Korea
* Corresponding author (sdkim777@gmail.com)
[Received 27 May 2015, Reviewed 2 June 2015, Accepted 7 July 2015]
☆ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2015R1A2A2A01004078).

이러한 문제를 해결하기 위하여 본 논문에서는 IoT 결함을 분류하고, 분류된 결함들의 원인과 증상을 분석한다. 그리고 분석 결과를 기반으로 서비스 실행 시에 발생 할 수 있는 결함을 관리하기 위한 프로세스를 제안한다. 본 연구에서 제안하는 프로세스와 관리 기법을 활용하여 결함 관리가 필요한 IoT 애플리케이션의 개발 비용을 줄일 수 있고, IoT 애플리케이션의 품질을 향상 시킬 수 있다.

본 논문의 3장에서는 IoT 환경에서 발생 할 수 있는 네 가지의 결함 종류를 나열하고 각 결함 종류에 대해 설명한다. 4장에서는 구분된 각 결함들을 야기시킬 수 있는 원인들을 설명한다. 5장에서는 결함들로 인해 발생할 수 있는 증상들에 대해 설명한다. 6장에서는 서비스 실행 시에 발생한 결함을 관리하기 위한 결함 관리 프로세스에 대해 설명하고, 프로세스 내에서 수행되는 기법들의 적용성을 검증한다. 그리고 7장에서는 본 논문에서 제안한 결함 관리 프로세스의 PoC (Proof-of-Concept) 구현과, 그 구현체를 이용한 실험 결과를 보여준다.

## 2. 관련 연구

Asim의 연구에서는 WSN (Wireless Sensor Network) 환경을 위한 결함 관리 메커니즘을 제안한다 [10]. 이 메커니즘은 WSN 환경을 다수의 cell로 분할하여 관리하는 cellular network 구조를 기반으로 하고 있다. 이 구조에서는 cell manager라는 노드가 하나의 cell에 있는 센서들을 관리한다. 그리고 cell manager는 이 연구에서 제안한 결함 감지와 결함 복구 기법을 이용하여 해당 cell에 있는 센서 연결에 대한 결함을 효과적으로 해결한다. 그러나 이 연구에서는 결함의 종류와 메커니즘간의 관계에 대해 기술하지 않고 있다. 그리고 이 연구는 다양한 결함 종류 중 네트워크 연결과 관련된 결함에만 초점을 맞추고 있다.

Ni의 연구에서는 신뢰성이 떨어지는 데이터에 의한 결함을 감지하기 위한 기법을 제안하고 있다 [11]. 이 기법은 HBST(Hierarchical Bayesian Spatio-Temporal) 기반의 수학적 모델링을 통해 신뢰성이 떨어지는 데이터를 감지하고, 이를 통해 신뢰성이 높은 센서를 선택한다. 이 기법을 통해 IoT 환경의 불안정한 네트워크로 인하여 IoT 서비스의 신뢰성이 저하되는 문제를 효과적으로 해결 할 수 있다. 그러나 이 연구는 다양한 결함 종류 중에서 비신뢰적인 데이터에 의한 결함만을 제한적으로 고려하고 있다.

Xu의 연구에서는 IoT 환경을 위한 지능적인 결함 예측 시스템을 제안한다 [12]. 4개의 레이어로 구성된 이 시스템은 센서를 모니터링 한 결과를 기반으로 결함의 유무를 예측하고, 예측된 결과에 대한 알맞은 피드백을 수행한다. 이 시스템을 이용한 주기적인 모니터링을 통해 IoT 환경에서 발생하는 결함을 실시간으로 감지하고, 결함에 대한 즉각적인 대응이 가능하다. 그러나 이 연구는 각 시스템 구성 레이어에 대한 구체적인 설명이 미흡하다.

Maalel 의 연구에서는 IoT 환경 내에서의 데이터 교환 프로토콜을 제안한다 [13]. 이 프로토콜은 전송 중 손실될 수 있는 패킷을 위한 복구 메커니즘과 에너지 효율적인 데이터 전송 메커니즘을 포함한다. 해당 프로토콜 전송을 활용하여, 신뢰성이 떨어지는 데이터로 인한 결함과 IoT 디바이스의 제한적인 자원에 의한 결함을 효과적으로 예방 할 수 있다. 그러나 이 연구에서는 제안된 기법의 실효성에 대한 검증이 부족하다. 그리고 이 연구는 네트워크와 관련된 결함 이외에 다른 결함들에 대한 고려가 부족하다.

이렇듯, 대부분의 IoT 결함 관리 연구에서는 IoT환경에서 발생할 수 있는 결함의 일부만을 고려하거나, 각 연구에서 제안하는 기법에 대한 설명이 상세하지 않다. 이에, 본 논문에서는 IoT 환경에서 발생 할 수 있는 다양한 종류의 결함을 분석한다. 그리고 이러한 결함들을 고려하여, 이들을 효과적으로 관리하기 위한 결함 감지 프로세스를 제안한다.

## 3. IoT 결함 타입 (Fault Types) 분류

IoT 환경은 전통적인 소프트웨어의 환경과 다른 특성을 지니고 있으므로, 순수 소프트웨어 연구에서 다루어지지 않았던 다양한 IoT 컴퓨팅의 결함이 발생 할 수 있다. 본 장에서는 IoT 컴퓨팅에서 발생할 수 있는 결함의 종류를 나열하고, 각 결함에 대해 설명한다.

### 3.1 IoT 서비스 결함

서비스 결함이란 IoT 애플리케이션이 제공하는 서비스의 내부적인 요소에 존재하는 결함을 의미한다 [14]. 이 결함은 IoT 애플리케이션의 부정확한 설계나 프로그래머의 실수에 의해 생성된 프로그래밍 코드에 의해 야기된다.

예를 들면, IoT 디바이스가 수집한 센서 데이터를 주

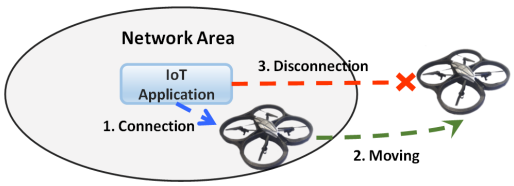
기적으로 수집하는 IoT 애플리케이션의 설계가 잘못 되어 있는 경우에는 IoT 디바이스와의 불필요한 연결이 자주 발생할 수 있다. 이러한 연결로 인해 IoT 애플리케이션이 운영되는 디바이스의 자원이 낭비되어 자원 부족 현상이 일어날 수 있다.

위와 같이, 서비스 결함은 IoT 애플리케이션 내부에 에러를 발생시킨다. 이러한 에러는 IoT 애플리케이션이 비정상적으로 종료되는 직접적인 원인이 될 수 있다.

### 3.2 IoT 디바이스 연결성 (Connectivity) 결함

연결성 결함이란 IoT 애플리케이션과 IoT 디바이스간의 연결 상태의 변화에 따른 결함을 의미한다. 이 결함은 주로 무선 네트워크에 연결된 IoT 디바이스의 위치가 변하면서 발생할 수 있다.

예를 들면, 웨어러블 디바이스나 AR.Drone 같이 스스로 움직일 수 있는 IoT 디바이스들의 위치는 IoT 애플리케이션과 연결된 상태에도 변할 수 있다. 그리고 대부분의 IoT 디바이스들은 Wi-Fi, Bluetooth와 같은 무선 네트워크를 기반으로 IoT 애플리케이션과 연결된다. 즉, IoT 디바이스의 위치가 무선 네트워크의 범위 밖으로 이동하여 (그림 1)와 같이 연결이 끊어질 수 있다.



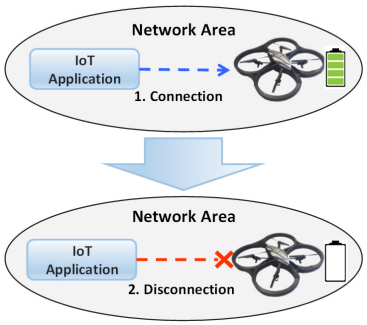

(그림 1) 디바이스의 위치 변화에 따른 연결성 결함의 예
(Figure 1) An example of connectivity faults by changes of device locations

이러한 연결성 결함으로 인해, 사용자의 의도와 상관없이 IoT 서비스가 종료되는 문제가 발생한다. 그리고 불안정한 연결을 통해서 비효율적인 메시지 교환이 이루어질 수 있다.

### 3.3 IoT 디바이스 하드웨어 결함

디바이스 하드웨어 결함이란 센서, 액츄에이터, 배터리와 같이 IoT 디바이스를 구성하는 하드웨어 컴포넌트 내에서 발생하는 결함을 의미한다 [15]. 이러한 하드웨어들은 소프트웨어와는 다르게 비영구적이라는 특성을 지니고 있으므로, 제한적인 자원의 고갈, 혹은 노후화로 인한 고장과 같은 결함이 발생 할 수 있다 [16][17][18].

예를 들면, IoT 디바이스의 구동에 따라 해당 디바이스의 배터리 자원 또한 지속적으로 소모된다. 만일, IoT 디바이스가 배터리 교체나 충전 없이 장시간 동안 사용되게 되면, (그림 2)과 같이 배터리가 고갈되는 문제가 발생한다.




(그림 2) 제한된 자원의 고갈에 따른 디바이스 하드웨어 결함의 예
(Figure 2) An example of device hardware faults by exhausting restricted resources

이러한 하드웨어의 결함으로 인해, 사용자가 원하는 디바이스를 사용하지 못하거나, 디바이스가 제공하는 기능을 효과적으로 활용 할 수 없는 문제가 발생할 수 있다.

### 3.4 IoT 운영 환경 (Environmental) 결함

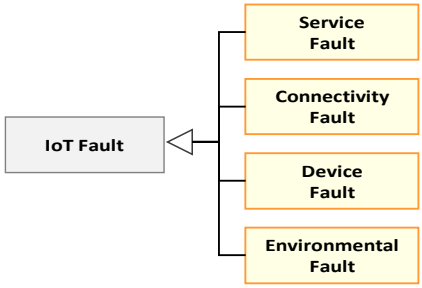
운영 환경 결함이란 IoT 디바이스가 운영되는 환경으로 인해 발생하는 결함을 의미한다. IoT 디바이스가 정상적으로 동작하지 못하는 환경에서 디바이스가 운영되는 경우, 해당 디바이스의 기능이 작동하지 않거나, 오작동할 수 있다[19].

예를 들면, AR.Drone과 같이 GPS 센서를 장착한 IoT 디바이스가 이동하여 운영 환경이 바뀜으로써 결함이 발생 할 수 있다. 만일 IoT 디바이스가 장애물이 많은 위치로 이동했을 경우에 GPS가 동작 하지 않거나, 정확한 위치 정보를 얻지 못할 수 있다.

위와 같이 디바이스의 기능이 동작하지 않는 문제가 발생하여 사용자가 원하는 IoT 서비스가 제공되지 않을 수 있다. 그리고, 디바이스의 오작동으로 인하여 사용자에게 잘못된 서비스가 제공 될 수 있다.

### 3.5 결함 타입의 정리

본 절에서는 위에서 설명한 결함들을 기반으로 IoT 환경에서 발생할 수 있는 결함을 분류한다. (그림 3)은 본 연구에서 제안하는 IoT 결함의 분류 체계를 보여준다. 본 논문에서는 위에서 설명한 네 가지의 결함으로 IoT 결함을 분류한다.

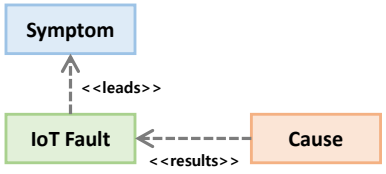



(그림 3) IoT 결함의 분류 체계
(Figure 3) Taxonomy of IoT Faults

서비스 결함(Service Fault) 은 IoT 애플리케이션이 제공하는 서비스의 내부에서 발생하는 결함을 말한다. 연결성 결함 (Connectivity Fault)는 애플리케이션과 디바이스간의 연결에 발생한 결함을 의미한다. 디바이스 하드웨어 결함 (Device Fault)는 애플리케이션이 사용하는 디바이스 자체의 결함을 가리킨다. 그리고 운영 환경에서의 결함(Environment Fault)는 디바이스가 운영되는 환경에서의 결함을 말한다.

## 4. IoT 결함의 원인 (Cause) 타입

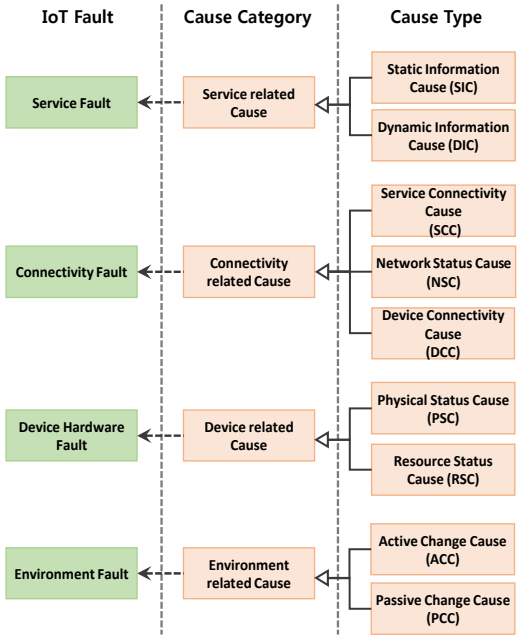
본 장에서는 IoT 결함의 원인과, 결함으로 인해 보여지는 증상에 대해 설명하여 IoT 결함을 분석한다. 원인과 IoT 결함, 그리고 증상간의 관계는 (그림 4) 와 같다.




(그림 4) IoT 결함, 원인, 증상간의 관계
(Figure 4) Relationships among IoT faults, causes, and symptoms

IoT 결함의 증상(Symptom)은 IoT 결함이 발생하였을 때의 상태와 정상 범위의 상태가 서로 다른 상황을 의미한다. 그리고 IoT 결함을 발생시키는 근본, 혹은 사건을 IoT 결함의 원인 (Cause)라고 한다.

IoT 결함에는 다양한 원인들이 존재한다. 예를 들어, 디바이스가 네트워크 범위 밖으로 이동하여 연결성 결함이 발생 할 수 있다. 그리고 운영환경의 자체변화로 인하여 운영환경 결함이 야기될 수 있다. 본 절에서는 IoT 결함을 야기시키는 원인의 종류를 구분하고, 각 원인들과 IoT 결함간의 관계를 분석한다. (그림 5)는 IoT 결함을 발생 시키는 원인들과 3장에서 분류한 IoT 결함들 간의 관계를 보여준다.




(그림 5) IoT 결함과 원인간의 관계
(Figure 5) Relationships between IoT faults and causes

정적 정보 원인(SIC): IoT 애플리케이션을 구성하는 컴포넌트나 컴포넌트간의 관계에 존재하는 문제를 의미한다. 애플리케이션 내부에 있는 메소드의 잘못된 구현이 본 원인의 예가 될 수 있다.

동적 정보 원인(DIC): 런타임 시에 애플리케이션의 상태, 혹은 컴포넌트들간의 상호작용 시에 발생하는 문제를 말한다. 예를 들어, 애플리케이션 실행 중에 메모리 누수로 인해 서비스 결함이 발생할 수 있다.

*서비스 연결성 원인 (SCC):* 애플리케이션이 디바이스와의 연결을 생성하지 못하는 문제를 가리킨다. 애플리케이션이 구동되는 노드가 네트워크 범위를 이탈하거나, 그 노드가 해당 네트워크 프로토콜을 지원하지 않는 경우가 본 원인의 예이다.

*네트워크 상태 원인 (NSC):* IoT 서비스와 디바이스간 연결의 기반이 되는 네트워크의 상태에 발생하는 문제를 말한다.

*디바이스 연결성 원인 (DCC):* 디바이스의 연결 상태에 발생하는 문제를 의미한다. 예를 들어, 디바이스의 위치가 네트워크 범위를 벗어나게 되면 디바이스의 연결성에 문제가 발생한다.

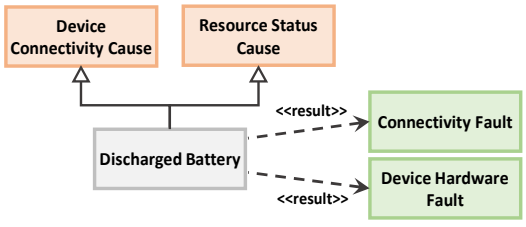
*물리적 상태 원인 (PSC):* 디바이스에 장착된 하드웨어 컴포넌트에 생긴 물리적인 문제를 의미한다. 대표적으로 충돌의 의한 파손, 하드웨어의 노후화와 같은 문제가 있다.

*자원 상태 원인 (RSC):* 디바이스 내부의 자원의 상태와 관련된 문제이다. 예를 들어, 디바이스의 배터리가 고갈되거나, 메모리가 고갈되는 문제들을 말한다.

*능동적 환경 변화 원인 (ACC):* 운영환경이 스스로 변화하여 생기는 문제들을 말한다. 날씨, 기온, 습도와 같은 요소들은 자체적으로 변화하여 디바이스가 정상적으로 운용되는 데에 어려움이 생길 수 있다.

*수동적 환경 변화 원인 (PCC):* 다른 요인으로 인해서 디바이스가 운영되는 환경이 비정상적으로 변하는 문제를 말한다. 예를 들면, GPS를 장착한 디바이스가 터널로 들어가게 되면, 그 디바이스의 운영 환경은 GPS가 정상적으로 동작 할 수 없는 환경으로 변하게 된다.

이 원인들은 3장에서 설명한 결함들을 동시에 야기시킬 수 있다. 예를 들어, 디바이스의 배터리가 고갈되어 디바이스와 서비스간의 연결이 끊어진 경우, IoT 결함과 그 원인은 (그림 6)와 같이 표현될 수 있다.

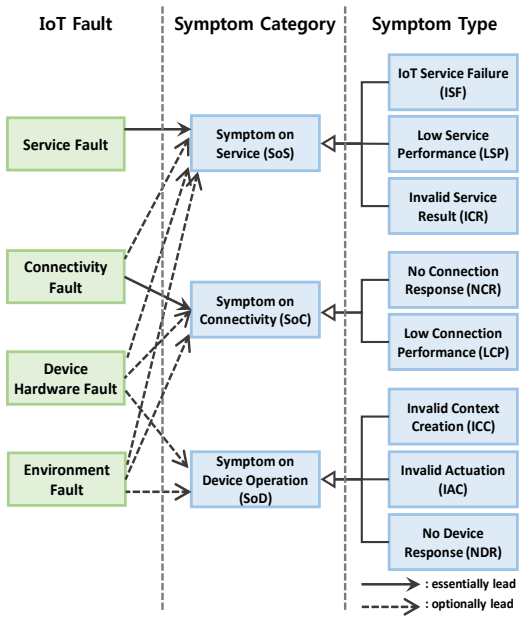



(그림 6) IoT 결함의 구체적인 원인의 예제
(Figure 6) An example of concrete causes of IoT faults

배터리는 디바이스가 갖는 자원 중 하나이므로, 배터리의 고갈은 RSC에 해당한다. 그리고 배터리가 고갈되어 디바이스가 연결 불가능한 상태로 바뀌었으므로 DCC에도 해당한다. 즉, 배터리의 고갈은 디바이스 하드웨어 결함과 연결성 결함을 야기시킨다.

## 5. IoT 결함의 증상 (Symptom) 타입

IoT 결함이 발생하면 그 결함에 수반되는 증상이 나타난다. 그리고 발생한 결함들을 다루기 위해서는 그 증상들을 감지해야 한다. 본 절에서는 각 결함들이 발생했을 때 나타나는 대표적인 증상들을 설명한다. (그림 7)는 3장에서 설명한 결함들이 수반하는 증상을 보여준다.




(그림 7) IoT 결함과 증상간의 관계
(Figure 7) Relationships between IoT faults and symptoms

**서비스에서의 증상 (SoS)**은 애플리케이션이 제공하는 서비스에서 발현되는 증상이다. 만일 이 증상이 발현되었다면, 이는 서비스 결함이 있다는 의미이다. 그리고 추가적으로 연결성 결함, 디바이스 하드웨어 결함, 운영환경 결함이 있을 수 있다. 이 증상은 아래의 세 가지의 세부 증상으로 구분된다.

*서비스 실패(ISF):* IoT 서비스가 실행 중에 강제로 종료되는 증상을 말한다.

*낮은 서비스 성능(LSP):* IoT 서비스가 결과를 반환하는데 까지 걸리는 시간이 오래 걸리는 증상이다.

*비 유효한 서비스 결과(ISR):* IoT 서비스가 정상 범위를 벗어나는 결과값을 반환하는 증상을 의미한다.

**연결성에서의 증상 (SoC)**은 IoT 서비스와 디바이스 간의 연결에 나타나는 증상이다. 이 증상은 연결성 결함이 있을 때 발현된다. 추가적으로 디바이스 하드웨어 결함이나 운영 환경 결함으로 인해 발현될 수 도 있다. 이 증상은 아래의 두 가지 증상으로 구분된다.

*연결 무응답(NCR):* IoT 서비스와 IoT 디바이스간의 연결이 파손되어 어떠한 요청도 디바이스에게 보낼 수 없는 증상을 말한다.

*낮은 연결 성능(LCP):* IoT 서비스와 디바이스간 통신에 지연이 발생하는 증상을 의미한다. 서비스가 디바이스가 획득한 컨텍스트를 늦게 받거나, 서비스의 명령이 디바이스에서 늦게 실행되는 것이 본 증상의 예가 된다.

**디바이스 구동에서의 증상 (SoD)**은 디바이스의 기능이 호출 될 때 발현되는 증상이다. 이 증상은 디바이스 하드웨어에 결함이 있거나, 운영 환경에 결함이 있을 경우에 생긴다. 이 증상은 아래의 세 가지의 세부 증상으로 구분된다.

*비 유효한 컨텍스트 생성(ICC):* IoT 디바이스가 정상 범위를 벗어나는 컨텍스트를 생성하는 증상을 나타낸다.

*비 유효한 액츄에이션(IAC):* 디바이스의 액츄에이션 기능이 오작동을 일으키는 증상을 말한다.

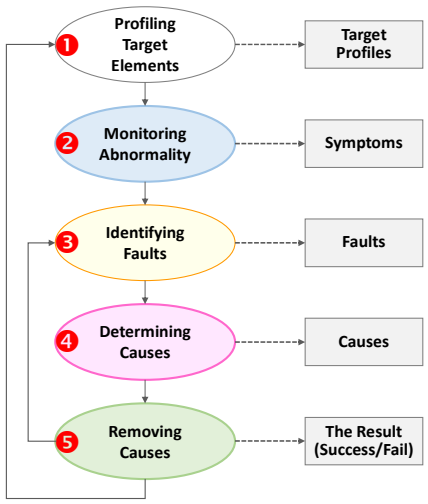
*디바이스 무응답 (NDR):* IoT 서비스가 IoT 디바이스와의 연결을 통해 디바이스의 기능을 호출했음에도 불구하고 디바이스가 응답하지 않는 증상을 의미한다.

4. 장에서 설명한 원인으로 인해 다양한 증상이 나타난다. 예를 들어 디바이스가 네트워크 영역 밖으로 이동하여 연결이 되지 않는 경우(DCC), 연결성 결함이 나타난다. 이 결함으로 인해 서비스와 디바이스간의 연결이 끊어지는 증상이 발생(NCR)한다. 그리고 그 서비스는 사용자가 원하는 결과를 반환(ISR)할 수 없게 된다.

# 6. IoT 결함 관리 프로세스 및 기법

## 6.1 결함 관리 프로세스

본 논문에서 제안하는 결함 관리 프로세스는(그림 8)과 같이 5개의 단계로 구성된다.




(그림 8) 결함 관리 프로세스
(Figure 8) Fault Management Process

단계 1에서는 결함 관리 대상인 항목들의 정보를 프로파일링 한다. 그 결과인 Target Profile은 IoT 서비스나 디바이스에 접근할 때 사용되는 정보를 포함한다. 단계 2에서는 IoT 서비스가 실행될 때 발생 할 수 있는 이상을 감지하여 그 증상을 반환한다. 단계 3에서는 반환된 증상을 이용하여 어떤 결함이 발생하였는지 식별한다. 단계 4에서는 식별된 결함을 야기시킨 원인의 범주를 판단한다. 단계 5에서는 판단된 원인을 제거하기 위한 작업이 수행된다. 만일 성공적으로 그 원인이 제거되었으면, 결함 관리 대상의 정보를 수정하기 위해 단계 1을 수행한다. 만일 원인을 제거하지 못했을 경우, 이는 다른 대상에 결함이 발생함을 의미하므로 그 결함을 식별하기 위한 단계 3을 수행한다.

## 6.2 단계 1. 대상 항목 프로파일링 (Profiling)

본 연구의 IoT 결함 관리 기법이 수행되기 위해서는 결함을 식별하기 위한 경계 값이나 교체될 수 있는 디바이스의 정보가 필요하다. 이러한 정보들을 이용하여 단계 1에서 서비스 프로파일과 디바이스 프로파일을 생성과 수정 작업을 수행한다. 서비스 프로파일에는 다음과 같은 정보가 기술된다.

- *Service ID*: IoT 서비스의 식별자
- *Utilized device*: IoT 서비스가 사용하는 디바이스의 식별자와 대체 가능한 디바이스의 식별자

- *Utilized device list: Utilized Device*의 목록
- *Valid range*: IoT 서비스의 반환 값의 정상 범위
- *Execution time threshold*: IoT 서비스의 최대 허용 응답 시간
- *Replaceable service ID*: 대체 가능한 IoT 서비스의 식별자

다음은 디바이스 프로파일에 포함되는 정보들이다.

- *Device ID:* IoT 디바이스의 식별자
- *Device name:* IoT 디바이스의 이름
- *Network info:* 디바이스가 지원하는 네트워크 프로토콜과 네트워크 주소
- *Network info list:* Network info의 목록
- *Sensor info:* 디바이스가 제공하는 센서 이름, 타입, 정상 호출시간의 경계 값, 정상 값의 범위
- *Actuator info:* 디바이스가 제공하는 액츄에이터 이름, 타입, 정상 호출시간의 경계 값
- *Sensor info list:* Sensor Info의 목록
- *Actuator info list:* Actuator Info의 목록

이러한 서비스 프로파일과 디바이스 프로파일에 명세된 정보들은 결함관리 프로세스 내에서 사용된다. 프로파일 정보를 사용하는 결함 관리 기법은 아래의 6.3절부터 6.6절에서 상세히 다룬다.

## 6.3 단계 2. 이상 감지 모니터링

본 기법은 호출될 서비스에서 발생한 예외를 인지하여 이상을 감지한다. 이러한 증상 인지를 위해서 서비스 프로파일과 디바이스 프로파일 정보가 이용된다. 각 증상과 프로파일의 정보간의 관계는 (표 1) 과 같다.

(표 1) 인지되는 증상과 프로파일 정보간의 관계
(Table 1) Components and detected symptoms

| Detected Sym]ptom | Profile | Information |
|---|---|---|
| LSP | Service profile | Execution time threshold |
| ISR | | Valid range |
| LCP | Device profile | Connection response time threshold |
| ICC | | Sensor info |
| NDR | | Sensor info<br>Actuator info |

서비스 프로파일의 정보는 낮은 성능의 서비스와 비유효한 서비스 결과를 인지하는 데에 사용 할 수 있다. 서비스의 수행 시간과 서비스 프로파일의 Execution Time Threshold를 비교하여 해당 서비스의 성능을 측정 할 수 있다. 그리고 서비스 수행 결과와 Valid Range간의 비교를 통해서도 결함 감지가 가능하다.

디바이스 프로파일의 정보를 이용하면, 서비스와 디바이스간의 연결 성능과, 비정상적인 컨텍스트 생성, 그리고 디바이스의 무응답을 인지할 수 있다. 디바이스 프로파일 정보 중에서, Sensor Info와 Actuator Info의 정상 호출시간의 경계 값을 활용하여 연결 상태와 해당 디바이스의 반응 유무를 판단 할 수 있다. 그리고 Sensor Info의 정상 값의 범위를 이용하여 해당 디바이스가 생성한 컨텍스트가 비정상적인지 판단이 가능하다.

서비스 실패와 연결 무응답 증상은 서비스 프로파일과 디바이스 프로파일을 사용하지 않고, 프로그래밍 언어의 메커니즘을 이용하여 인지가 가능하다. 예를 들어, Java 기반의 서비스가 연결이 불가능하여 IP를 할당 받지 못한 IoT 디바이스에 연결을 시도하게 되면 Connection Exception이 발생한다.

비 유효한 액츄에이터 증상의 경우에는 해당 액츄에이터의 동작 상태를 소프트웨어로 감지하는 데에는 한계가 있으므로 본 기법에서는 다루지 않는다.

이렇게, 프로파일 정보들과 프로그래밍 언어의 메커니즘을 이용하여 결함으로 인한 증상들을 파악할 수 있다. 리스트 1은 서비스 프로파일을 이용하여 서비스에서의 증상을 인지하는 알고리즘이다.

(리스트 1) 서비스에서의 결함 증상을 인지하기 위한 알고리즘
(List 1) An algorithm for recognizing symptom on service

```
input: N/A
output: N/A
01  Begin
02  IoTService srv = new IoTService();
03  try{
04    time startTime = Time.currentTime()
05    result = srv.execute()
06    time endTime = Time.currentTime()
07    if result is invalid :
08      throw new InvalidResultException()
```

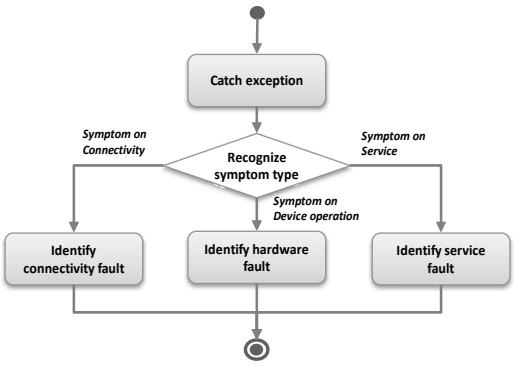
```
09      responseTime = endTime – startTime()
10      if responseTime > srv.maxReponseTime :
11          throw new TimeoutException()

12  } catch (ServiceFailureException){
13      throw new ServiceFailureException()
14  }
15  End
```

라인 05번에서는 사용자가 원하는 IoT 서비스를 호출한다. 만일, 그 서비스가 비정상적으로 종료되는 증상이 발생하면 라인 12번에서 이를 인지한 후, 예외를 발생시킨다. 서비스가 정상적으로 종료되면 라인 07번에서 서비스 결과값을 검사한다. 그 결과값이 유효하지 않다면 라인 08번에서 예외를 생성하여 외부로 반환한다. 라인 10번에서는 그 서비스의 수행시간과 경계 시간을 비교한다. 만일 경계 값보다 더 오래 걸렸다면 라인 11번에서 예외를 발생시킨다.

## 6.4 단계 3. 결함 타입 식별

단계 2에서 발생한 증상으로, 단계 3에서는 단계2에서 발생한 증상과 관련된 결함 타입을 식별한다. (그림 9)은 단계 3과에서 진행되는 결함 타입 식별의 순서를 보여준다.




(그림 9) 결함 타입을 식별하기 위한 업무 흐름
(Figure 9) Workflow for Identifying Fault types

단계 3에서는 단계 2에서 발생한 예외가 어떤 타입의 증상과 관련되어있는지 식별한다. 이 때, 판단되는 기준은 그 예외가 발생한 위치이다. 예를 들면, IoT 서비스 모듈에서 예외가 발생하였다면, Service Fault로 식별하고, 디바이스와의 연결을 의미하는 Connection 모듈에서 발생한 예외는 연결성에서의 증상로 인지하여 연결성 결함으로 식별된다.

## 6.5 단계 4. 원인 판단

본 단계에서는 단계 3에서 식별된 결함 타입과 4. 장에서 설명한 결함과 원인과의 관계를 이용하여 원인의 종류를 판단한다. 식별될 수 있는 각 결함의 종류와 각 결함을 발생 시킬 수 있는 원인들간의 관계 정보를 조회하여 결함을 발생시킨 원인을 판단할 수 있다. 그리고 추후에 IoT 환경을 관리할 때에 유용하게 사용할 수 있도록, 원인의 판단 결과를 로그로 저장한다.

## 6.6 단계 5. 원인 제거

본 단계에서는 판단된 원인에 따라서, 교체가 가능한 서비스, 디바이스와 교체하여 결함이 발생한 원인을 제거한다. 즉, 결함의 원인이 일어나지 않은 요소와 교체하여 IoT 애플리케이션의 결함 상태를 해결하는 단계이다. (표 2)는 판단된 원인과 대체되는 요소, 그리고 대체의 기준이 되는 서비스 프로파일 정보들간의 관계를 보여준다.

(표 2) 판단된 원인과 대체되는 요소간의 관계
(Table 2) Relationships between faults and replaced elements

| Determined Causes | Replaced Element | Information |
|---|---|---|
| Service related Cause | IoT service | Replaceable service |
| Connectivity related Cause | Network protocol | Network info |
| Device related Cause | IoT device | Utilized device |

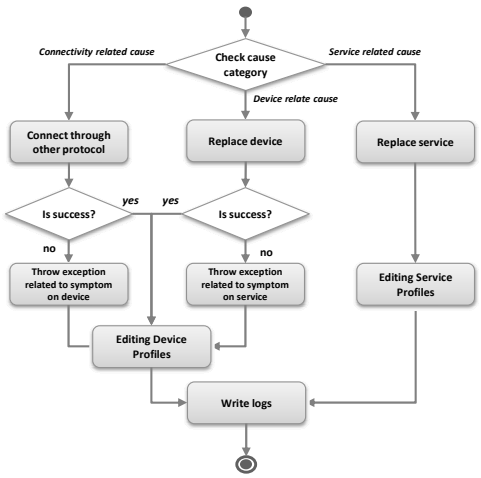
디바이스 관련 원인은 증상이 발현된 디바이스를 대신할 수 있는 다른 디바이스로 교체하여 제거가 가능하다. 즉, 디바이스 구동에서의 증상이 암시하는 결함의 원인인 디바이스 하드웨어와 자원에서 문제가 없고 정상적인 환경 내에서 구동되는 디바이스와 교체하여 결함의 원인을 제거하는 것이다. 이 때, 교체되는 디바이스는 서비스 프로파일 내의 Utilized device 정보에 포함된다.

만일 연결에서의 증상이 감지된 경우에는 연결된 디바이스의 프로파일 정보 중에서 Network info를 이용하여 디바이스가 제공하는 네트워크 프로토콜 중에서 교체 가

능한 프로토콜을 사용할 수 있다. 연결에 결함이 발생할 수 있는 네트워크 환경을 사용하지 않고, 디바이스가 지원하는 또 다른 네트워크 프로토콜을 이용하여 연결성 결함을 해결 할 수 있다.

서비스에 결함이 발생한 경우에는, 서비스 프로파일의 명시된 Replaceable Service ID가 가리키는 다른 서비스와의 교체로 해당 결함을 해결 할 수 있다. 이 때, response time 증상인 경우에는 정상적인 결과를 반환했으나, 서비스 수행 시간이 비정상적으로 오래 걸린 증상이므로, 사용자에게 그 결과를 반환한 후에 관리되는 서비스의 목록을 대체되는 서비스에 알맞게 수정한다.

이러한 해결 기법간에는 선행관계가 존재한다. 예를 들어, 연결성 결함을 해결하기 위한 네트워크 프로토콜 교체가 실패한 경우에는, 그 연결의 대상이 되는 디바이스의 교체가 일어난다. (그림 10)는 결함 해결 기법간의 선행 관계를 보여준다.




(그림 10) 결함 해결 기법간의 선행 관계
(Figure 10) Preceding relation among fault remedy methods

연결성 결함이 발생하였다면 네트워크 프로토콜을 교체한다. 만일, 교체된 네트워크 프로토콜을 기반으로 하는 연결에서도 결함이 존재한다면 디바이스 하드웨어나 환경에 결함이 있는 것으로 간주하고, 디바이스를 교체한다. 교체된 디바이스에도 결함이 있다면, 그 디바이스를 사용하는 서비스를 교체한다. 그리고 각 해결 기법이 성공적으로 끝난 후에는 발생되었던 결함과, 결함 해결 기법을 수행한 결과를 로그로 남긴다. 이러한 결함 해결 기법간의 선행관계를 통해, 해결 기법간의 중복 수행을 예방 할 수 있다. 그리고 작성된 로그를 IoT 환경 관리자가 참고하여 그 IoT 환경을 알맞게 바꿀 수 있다.

## 7. 사례연구를 통한 검증

### 7.1 결함 관리 프로세스 구현

본 연구에서 제안하는 결함관리 프로세스의 실효성을 검증하기 위해 Spheroball과 Arduino라는 두 개의 IoT 디바이스를 이용하여 아래와 같은 세 가지의 서비스를 구현하였다.

Brightness Sensing Service: Arduino를 통해 조도 값을 얻어오는 서비스이다.

Spheroball Dance Service: Spheroball을 무작위로 이동시키는 서비스이다.

Light Service: Spheroball과 Arduino에 부착된 LED를 켜는 서비스이다. 이 서비스는 각 디바이스 전용으로 두 가지 버전이 구현되었다.

두 가지 모델의 디바이스와 세 가지의 서비스를 기반으로 결함 관리 프로세스를 구현하였다. (리스트 2)는 Spheroball Dance Service 구현에 사용된 증상감지 단계의 구현체를 보여준다.

(리스트 2) 디바이스 하드웨어 결함을 다루기 위한 구현
(List 2) Implementation for handling device hardware fault

```
01  public void execute() {
02    Random random = new Random();

03    while (isExecuted) {
04      float direction = random.nextFloat();
05      float velocity = random.nextFloat();

06      try {
07        spheroball.move(direction, velocity);
08      } catch (BluetoothConnectionException e)
    { //Recognizing exception

            //Replacing a device
09        spheroball
         .setUrl(secondarySpheroballURL);
10          spheroball.connect();
11        }
12      }
13    }
```

Symptoms on Device가 발생하면, 그 증상과 관련된 예외가 라인 08번에서 감지된다. 그 후에 라인09번과 10번

을 통해서 결함이 있는 Spheroball은 교체 가능한 Spheroball로 교체된다.

### 7.2 실험 시나리오

결함 관리 프로세스의 실효성을 검증하기 위해 아래와 같은 세 가지의 시나리오가 수행되었다.

시나리오 #1: Brightness Sensing Service가 실행되던 중에 Wi-Fi 연결을 끊음으로써 Connectivity related Causes를 발생시켰다. 그 후에 Arduino가 USB Serial Port를 통해 연결되는지 확인한다.

시나리오 #2: Spheroball Dance Sensing Service가 실행되던 중에 Spheroball를 종료시킴므로써 Device related Causes를 발생시켰다. 그 후에 다른 Spheroball이 연결되는지 확인한다.

시나리오 #3: Arduino 전용의 Light Service를 특정 시점에 종료시킴으로써 Service related Causes를 발생시켰다. 그 후에 Spheroball 전용의 Light Service가 실행되는지 확인하였다.

### 7.3 실험 평가

시나리오 #1의 경우, Brightness Sensing Service에서 연결성 결함이 발견되었고, 이 결함을 해결하기 위해 (그림 11)의 (a)와 같이 USB 시리얼 포트를 통해 그 Arduino와 연결되었다.

시나리오 #2의 경우, Spheroball Dance Service에서 블루투스 연결 끊김에 대한 예외가 발견되었다. Spheroball은 블루투스 이외의 프로토콜은 지원하지 않기 때문에, 디바이스 결함으로 인지되었다. 그 후에, 이 서비스는 (그림 11)의 (b)와 같이 교체 가능한 Spheroball과 연결되었다.

```
Disconnected..
Connectivity Fault Occured!

Checking USB Serial Port
Connecting.....
Connection! (Serial Port: tty/0)
```

(a) Output of Scenario #1

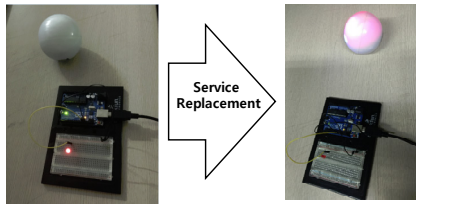
```
Disconnected..
Connectivity Fault Occured! (No Connection Response)

Executing Device Replacement
Replacable Device Found!
Connecting.....
Connection! (Addr: 00:06:66:44:62:8B)
```

(b) Output of Scenario #2

(그림 11) 실험 시나리오의 콘솔 출력
(Figure 11) Console Output of Experiment Scenarios

시나리오 #3의 경우, Arduino 전용의 Light Service가 원래의 시간보다 일찍 종료되었음을 확인하고 그 서비스를 호출한 컴포넌트에서 Spheroball 전용의 Light Service를 호출하였다. (그림 12)는 서비스 교체 전과 그 후의 모습을 보여준다.




(그림 12) Light Service가 교체된 실행 화면
(Figure 12) Screen dump for Light Service Replacement

## 8. 결 론

IoT 디바이스간의 협업을 통해 사용자에게 유용한 정보와 서비스를 제공하는 IoT 컴퓨팅 환경은 기존의 전통적인 소프트웨어 연구에서 다루어지지 않았던 결함들이 발생 할 수 있다. 그러나 이러한 결함들을 해결f하기 위한 연구들이 활발하게 진행 중에 있다. 그러나, 대부분의 연구들은 IoT 환경의 다양한 결함 중 일부의 결함만을 고려하거나, 구체적인 기법 설명이 미흡하다.

이러한 문제를 해결하고자, 본 논문에서는 결함 관리 프로세스와 결함 예방 기법을 제시하였다. 이를 위해, IoT 결함을 서비스 결함, 연결성 결함, 디바이스 하드웨어 결함, 운영환경 결함으로 나누었다. 그리고 각 결함들의 원인과 증상들을 분석하였다. 이러한 분석 결과를 기반으로 서비스 수행 중 발생한 결함을 관리하기 위한 결함 관리 프로세스와, 서비스 수행 전에 디바이스를 모니터링 하는 결함 예방기법을 제안하였다. 그리고 기법들에 대한 실험을 진행하여, 제안된 기법의 효과와 적용성을 확인하였다. 본 논문에서 제안한 결함 관리 기법을 통해 신뢰성이 요구되는 IoT 애플리케이션 개발의 비용 절감과 품질 향상을 기대할 수 있다.

향후의 연구에서는 ISO 25010에서 정의한 소프트웨어 품질 항목과 본 연구에서 제안한 기법들을 활용하여, IoT 애플리케이션의 신뢰성 향상을 위한 프레임워크를 다룰 예정이다.

## 참 고 문 헌 (Reference)


[1] Reed, D.A. Gannon, D.B., and Larus, J.R., "Imagining the Future: Thoughts on Computing", IEEE Computer, Vol.45, pp. 25 - 30, 2011.
http://doi.ieeecomputersociety.org/10.1109/MC.2011.327

[2] Sehgal, A., Perelman, V., Kuryla, S., Schönwälder, J, "Management of resource constrained devices in the internet of things", Communications Magazine, IEEE, 2012, 50.12: 144-149.
http://dx.doi.org/10.1109/MCOM.2012.6384464

[3] Paradis, L. and Han, Q., "A Survey of Fault Management in Wireless Sensor Networks", Journal of Network and Systems Management, Vol. 15, pp. 171-190, 2007.
http://dx.doi.org/10.1007/s10922-007-9062-0

[4] Zhu, Q., Wang, R., Chen, Q., Liu, Y., and Qin, W., "IoT Gateway: BridgingWireless Sensor Networks into Internet of Things", In Proceedings of 8th IEEE/IFIP Embedded and Ubiquitous Computing (EUC), pp.347-352, Hong Kong, December 2010.
http://dx.doi.org/10.1109/EUC.2010.58

[5] Sokullu, R. and Karaca, O., "Fault Management for Smart Wireless Sensor Networks", In Proceedings of 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), pp.382-387, Fkuoka, September, 2012.
http://dx.doi.org/10.1109/UIC-ATC.2012.159

[6] Yu, P., Jia, S., and Xi-yuan, P., "A self detection technique in fault management in WSN", In Proceedings of IEEE Instrumentation and Measurement Technology Conference (I2MTC), pp. 1-4, Binjiang, May 2011.
http://dx.doi.org/10.1109/IMTC.2011.5944171

[7] Shih, H.C., Ho, J.H., Liao, B.Y., and Pan, J.S., "Fault Node Recovery Algorithm for a Wireless Sensor Network", IEEE Sensors Journal, Vol. 13, pp. 2683-2689, 2013.
http://dx.doi.org/10.1109/JSEN.2013.2255591

[8] Liu, X., Cao, J., Bhuiyan, M., Lai, S., Wu, H., and Wang, G., "Fault tolerant WSN-based structural health monitoring", In Proceedings of IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), pp.37-48, Hong Kong, June 2011.
http://dx.doi.org/10.1109/DSN.2011.5958205

[9] Afsar, M.M., Yaghmaee Moghaddam, M.H., and Esmaeil, Z.K., "A Fault Tolerant Protocol for Wireless Sensor Networks", In Proceedings of Seventh International Conference on Mobile Ad-hoc and Sensor Networks (MSN), pp. 475-478, Beijing, December 2011.
http://dx.doi.org/10.1109/MSN.2011.5

[10] Asim, M., Mokhtar, H., and Merabti, M., "A self-managing fault management mechanism for wireless sensor networks", International Journal of Wireless & Mobile Networks (IJWMN), Vol.2, No.4, pp. 184-197, 2010.
http://arxiv.org/abs/1011.5072

[11] Ni, K. and Pottie, G., "Sensor network data fault detection with maximum a posteriori selection and bayesian modeling", ACM Transactions on Sensor Networks (TOSN), Vol. 8, no. 23, 2012.
http://dx.doi.org/10.1145/2240092.2240097

[12] Xu, X., Chen, T., and Minami, M., "Intelligent fault prediction system based on internet of things", Computers & Mathematics with Applications, Vol. 64, pp. 833-839, 2012.
http://dx.doi.org/10.1016/j.camwa.2011.12.049

[13] Maalel, N., Natalizio, E., Bouabdallah, A., Roux, P., and Kellil, M., "Reliability for Emergency Applications in Internet of Things", In Proceedings of IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), pp.361-366, May 2013.
http://dx.doi.org/10.1109/DCOSS.2013.40

[14] Silas, S., Ezra, K., and Rajsingh, E.B, "A novel fault tolerant service selection framework for pervasive computing", Human-centric Computing and Information Sciences, 2:5, 2012.
http://dx.doi.org/10.1186/2192-1962-2-5

[15] Lau, B., Ma, E, and Chow, T., "Probabilistic fault detector for Wireless Sensor Network", Expert Systems with Applications, Vol.41, pp. 3703-3711, 2013.
http://dx.doi.org/10.1016/j.eswa.2013.11.034

[16] Pop, F., Bessis, N., "Energy-Efficient and Fault Tolerant Methods for Message Delivery in Internet of Things", In Proceedings of 11th Roedunet International Conference (RoEduNet), pp.1-6, 2013.
http://dx.doi.org/10.1109/RoEduNet.2013.6511735

[17] Goudar, V. and Potkonjak, M., "Low-power semantic fault-detection in multi-sensory mobile health monitoring systems", In Proceedings of IEEE World Forum on Internet of Things (WF-IoT), pp.32-36, 2014. http://dx.doi.org/10.1109/WF-IoT.2014.6803112

[18] Redwan, H., Akele, G. and Kim, K.H., "Cluster-based failure detection and recovery scheme in wireless sensor network", In Proceedings of 6th Ubiquitous and Future Networks (ICUFN), pp.407-412, Shanghai, July 2014. http://dx.doi.org/10.1109/ICUFN.2014.6876824

[19] Munir, A., and Gordon-Ross, A. "Markov Modeling of Fault-Tolerant Wireless Sensor Networks", In Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN), pp. 1-6, Maui, August, 2014. http://dx.doi.org/10.1109/ICCCN.2011.6005768

## ◐ 저 자 소 개 ◑

**박 춘 우 (Chun Woo Park)**
2013년 숭실대학교 컴퓨터학부 졸업(학사)
2013년 ~ 2015년 숭실대학교 대학원 컴퓨터학과 석사과정
관심분야 : 사물 인터넷 컴퓨팅(Internet of Things Computing), 상황 인지 컴퓨팅 (Context-Aware Computing)
E-mail : cnsdnsla@gmail.com

**김 수 동 (Soo Dong Kim)**
1984년 Northeast Missouri State University 전산학과 졸업(학사)
1988년 The University of Iowa 전산학과 졸업(석사)
1991년 The University of Iowa 전산학과 졸업(박사)
1991년~1993년 한국통신 연구개발단 선임연구원
1994년~1995년 현대전자 소프트웨어연구소 책임연구원
1995년~현재 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)
E-mail : sdkim777@gmail.com