KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.10 pp.337~350 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.10.337

Unconventional Issues and Solutions in Developing IoT Applications

Hyun Jung La† · Soo Dong Kim††

ABSTRACT

Internet-of-Things(IoT) is the computing paradigm converged with different technologies, where diverse devices are connected via the wireless network, acquire environmental information from their equipped sensors, and are actuated. IoT applications provide smart services to users by interacting with multiple devices connected to the network. IoT devices provide the simple set of the information and also offer smart services by collaborating with other devices. That is, IoT applications always interact with IoT devices which are becoming very popular at a fast pace. However, due to this fact, developing IoT application results in unconventional technical challenges which have not been observed in typical software applications. Moreover, since IoT computing has its own characteristics which are distinguished from other former paradigms such as embedded computing and mobile computing, IoT applications also reveal their own technical challenges. Therefore, we analyze technical challenges occurring in developing IoT applications and present effective solutions to overcome the challenges. To verify identified issues and presented solutions, we present the result of performing a case study of developing an IoT application. Through the case study, we verify how the unconventional technical issues are raised in a real domain and analyze effectiveness of applying the solutions to the application.

Keywords : Internet-of-Thing, IoT Devices, Heterogeneity, Intrinsic Mobility, Physical Constraints, Technical Issues, Solutions

IoT 애플리케이션 개발에서 비전형적 이슈 및 솔루션

라 현 정† · 김 수 동††

요 약

사물 인터넷(Internet-of-Things, IoT)은 무선 인터넷을 기반으로 다양한 디바이스를 연결하고 센서를 통해 환경 정보를 획득하고 이를 기반으로 제어하는 여러 기술이 융합된 컴퓨팅 패러다임이다. IoT 환경에서 애플리케이션은 네트워크에 연결된 여러 디바이스들을 이용하여 사용자에게 유용한 정보와 편의를 제공할 수 있다. IoT 디바이스들은 단순한 정보를 제공하기도 하고, 다수의 디바이스들의 협업에 의한 서비스를 제공하기도 한다. 즉, IoT 애플리케이션은 빠르게 보급되고 있는 IoT 디바이스와 같이 상호작용한다. 이런 이유로 IoT 애플리케이션의 개발은 소프트웨어 기능만으로 구성된 소프트웨어 시스템 개발에는 나타나지 않는 비전형적인 기술적 이슈를 가지고 있다. 나아가 임베디드 컴퓨팅, 모바일 컴퓨팅 등의 패러다임과도 구별되기 때문에, 기존 형태의 시스템 개발에서 볼 수 없었던 이슈를 가지고 있다. 본 논문은 IoT 애플리케이션 개발에서 발생하는 기술적 어려움을 분석하고, 이들을 효과적으로 해결할 수 있는 기법들을 제시한다. 제시된 이슈들과 기법들을 검증하기 위하여, IoT 애플리케이션을 개발한 사례연구 결과를 보여준다. 이를 통하여, IoT 애플리케이션 개발의 비전형적인 기술적 이슈들이 구체적으로 어떻게 발생하여, 제시된 솔루션들이 어떻게 효과적으로 적용되었는지 분석한다.

키워드 : 사물 인터넷, IoT 디바이스, 이질성, 자체 이동성, 물리적 제약성, 기술적 이슈, 솔루션

1. 서 론

사물 인터넷(Internet-of-Things, IoT)은 무선 인터넷을 기반으로 다양한 디바이스를 연결하고 센서를 통해 환경 정보를 획득하고 이를 기반으로 제어하는 여러 기술이 융합된 컴퓨팅이다[1][2]. IoT는 기존 소프트웨어 패러다임인 임베디드(Embedded) 컴퓨팅, 모바일 컴퓨팅, 퍼베이시브(Pervasive) 컴퓨팅, 상황인지(Context-aware) 컴퓨팅으로부터 한 단계 진화한 형태라고 할 수 있다.

IoT 애플리케이션은 네트워크에 연결된 여러 IoT 디바이스들로부터 환경 정보를 획득하고 사용자에게 유용한 형태의 서비스를 제공하기 위해 관련 디바이스를 액추에이션(Actuation)한다. 이때, IoT 디바이스들은 단순한 정보를 제

※ 이 논문은 2012년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2012R1A1B3004130).
† 종신회원 : (주) 스마티랩 대표
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
Manuscript Received : September 4, 2014
Accepted : October 8, 2014
* Corresponding Author : Soo Dong Kim(sdkim777@gmail.com)

공하기도 하고, 다수의 디바이스들의 협업에 의한 서비스를 제공하기도 한다. 즉, IoT 애플리케이션은 IoT 디바이스들 간의 협업을 통해 서비스를 제공한다. 이는 순수 소프트웨어 기능 모듈만으로 구성된 기존 소프트웨어와는 구별되는 IoT 애플리케이션의 특징이다.

IoT 애플리케이션은 디바이스와 상호 협력한다는 점에서 임베디드 시스템 또는 모바일 애플리케이션과 유사하다. 그러나 IoT 애플리케이션은 임베디드 시스템이나 모바일 애플리케이션처럼 하드웨어 디바이스에 장착되어 해당 디바이스를 제어하는 기능을 수행하는 것 외에 별도의 PC나 디바이스에 설치되어 여러 분산된 디바이스로부터 값을 읽어오거나 액추에이션 기능을 수행할 수 있다. 그러므로 임베디드 또는 모바일 컴퓨팅과는 구별되는 특징을 가진다.

IoT 애플리케이션은 이질적인 여러 센서들과 상호작용한다는 점에서 퍼베이시브 컴퓨팅과 유사하다. 퍼베이시브 컴퓨팅에서 고려하는 디바이스는 주로 RFID 센서들로 네트워크 연결 능력을 반드시 가지고 있지 않고, 특정 장소에 고정되어 사용된다. 그러나 IoT 컴퓨팅에서는 네트워크 연결 능력을 가지는 다양한 형태의 디바이스를 고려하며, 다수의 디바이스는 이동성(Mobility)을 가지고 있으므로, 퍼베이시브 컴퓨팅 환경과는 구별되는 특징을 가진다.

IoT 애플리케이션은 주변 상황 정보를 기반으로 서비스를 제공한다는 점에서 상황인지 애플리케이션과 유사하다. 그러나 상황인지 컴퓨팅에서는 환경 정보를 얻어오기 위한 목적으로 센서들만 고려하는 반면에, IoT 애플리케이션은 디바이스를 액추에이션하는 기능도 포함한다.

이렇듯, IoT 애플리케이션은 순수 소프트웨어뿐 아니라 기존의 컴퓨팅 패러다임과 차별화되는 특성을 가지고 있기 때문에, 기존 형태의 애플리케이션을 개발하였을 때에는 관찰할 수 없었던 다양한 이슈들이 제기될 수 있다. IoT 개념은 최근에 소개된 개념이기 때문에, 이런 이슈들에 대한 언급은 일부 문헌에서 하고 있지만, 이에 대한 솔루션은 프레임워크 아키텍처 제시 또는 개념적인 수준의 기법 제시에 머무르고 있다.

그러므로 본 논문에서는 IoT 애플리케이션 개발 시에 발견할 수 있는 여러 이슈들을 언급하고, 이 이슈들을 효과적으로 해결할 수 있는 실용적인 솔루션을 제시한다. IoT 애플리케이션 개발과 관련된 이슈는 대부분 IoT 디바이스로부터 유도되기 때문에, IoT 디바이스의 특징인 이질성(Heterogeneity), 이동성, 물리적 제약성(Physical Constraints)으로부터 기존 형태의 소프트웨어에서 발견할 수 없었던 비전형적인 이슈들을 도출한다. 그리고 각 이슈별로 IoT 애플리케이션을 개발하면서 활용할 수 있는 실용적인 솔루션을 제안한다.

본 논문의 2절에서 IoT 연구 이슈를 다룬 문헌들을 조사 및 분석하고, 3절에서 IoT 디바이스의 세 가지 특징에서 발생할 수 있는 이슈를 분류한다. 그리고 4절에서는 각 이슈별로 효과적으로 적용할 수 있는 솔루션을 실용적인 수준으로 제시한다. 제안된 솔루션은 여러 IoT 애플리케이션 개발에 범용적으로 사용되기 위하여, IoT 애플리케이션 고유의 기능을 수행하는 모듈과 분리된 앱 에이전트(Agent) 설계에 영향을 미친다. 이는 대표적인 설계 원칙인 관점 분리 원칙(Separation of Concern)을 적용한 것으로, IoT 애플리케이션의 개발 복잡도를 줄이기 위한 방법이다. 마지막으로 5절에서는 도출한 이슈의 타당성과 제안된 솔루션의 적용 가능성 및 효용성을 검증하기 위하여, 지상용 디바이스 컨트롤러를 개발하는 사례연구를 수행한 결과를 보여준다. 본 논문을 통해 도출된 이슈와 제안된 솔루션을 이용하여 IoT 애플리케이션을 보다 쉽고 효과적으로 개발할 수 있고, 나아가 제안된 솔루션은 IoT 프레임워크 설계의 근간으로 활용될 수 있을 것이라고 기대한다.

2. 관련 연구

신기술인 IoT의 연구 방향을 제시하기 위해, IoT 애플리케이션 개발 시에 고려되어야 하는 이슈를 다룬 일부 논문들이 게재되었다.

Miorandi의 연구는 먼저 IoT 개념을 구현할 때 시스템 수준에서 제공되어야 하는 주요 기능으로서 디바이스 이질성 관리, 디바이스 확장성 관리, 디바이스 연결, 에너지 효율성 관리, 디바이스 이동성 추적, 데이터 시맨틱 관리, 보안 및 프라이버시 관리를 나열하였다[3]. 그리고 이와 관련하여 디바이스 식별 관련 이슈, 분산된 디바이스 관리 관련 이슈, 분산된 데이터 처리 관련 이슈, 보안 관련 이슈를 도출하고, 기존 연구 기법들이 해당 이슈들을 어느 정도 해결할 수 있는지를 기술하였다. 이 연구는 IoT 애플리케이션 개발과 관련하여 포괄적인 이슈를 제기하였지만, IoT 특성을 고려한 솔루션 제시가 필요하다.

Chen의 연구는 디바이스 초기화부터 사용자에게 유용한 서비스 제공에 이르기까지 IoT 애플리케이션이 수행해야 하는 전반적인 기능과 이와 관련된 이슈를 제기하였다[4]. 제기된 이슈에는 디바이스 설치 및 관리 노력 최소화, 디바이스의 제한된 전원, 수많은 디바이스 연결 관련 문제, 보안 및 프라이버시 이슈, 방대한 양의 효율적인 데이터 처리 이슈, 인터페이스 표준화 등이 있다. 그리고 이 이슈들을 해결하기 위해 저전력 센서 개발, 견고한 연결성, 자율적인 데이터 관리 및 분석 기법 등이 필요하다고 언급하였다. 이 연구는 IoT 애플리케이션 개발과 관련하여 포괄적인 이슈를 제기하였지만, 이에 대한 솔루션에 대해서는 방향만 제시하였다.

Khan의 연구는 IoT 애플리케이션을 위한 표준 아키텍처를 제시하고, IoT 애플리케이션 개발과 관련한 기술적 이슈를 제기하였다[5]. 제기된 이슈에는 IoT 디바이스의 네이밍 관리, 이질적인 디바이스 간의 상호 연동성 및 표준화, 정보 보안, 데이터 비밀 보장 및 암호화, 네트워크 보안, IoT 디바이스의 전원 제약성 등이 있다. 그러나 제기된 이슈들의 솔루션에 대한 언급은 없다.

Chaqfeh의 연구는 이기종 디바이스 간의 상호 연동성을 해결하는 IoT 미들웨어를 개발하는데 관련한 기술적 이슈를 제기하였다[6]. 이기종 디바이스에 대한 높은 상호 연동성,

많은 수의 디바이스를 관리할 수 있는 높은 확장성, 잘 설계된 API, 이동성 있는 디바이스 관리 및 요청에 맞게 적절한 디바이스 및 서비스 바인딩 등을 IoT 애플리케이션을 위한 미들웨어의 설계 이슈로 도출하였다. 그리고 시맨틱 웹 서비스 접근 방법, 센서 네트워크 접근 방법, 로봇 제어 접근 방법이 제기된 이슈를 효과적으로 해결할 수 있는지를 비교 분석하였다. 제기된 이슈들은 IoT 애플리케이션 개발에도 고려되어야 하는 주요한 것들이지만, 이에 대한 솔루션은 추상적으로 언급되어 있다.

Patel의 연구는 이기종의 여러 디바이스들 간의 상호작용을 보다 효과적으로 작성하는 것에 대한 어려움을 주요 이슈로 제기하고, 이를 해결하기 위해 도메인의 주요 개념들 간의 관계를 도식화한 도메인 모델(Domain Model)을 제안하였다[7]. 이 연구는 IoT 애플리케이션 개발과 관련된 이슈를 직접적으로 다루고 있지만, 그 이슈가 이기종 간의 복잡한 관계를 효과적으로 표현해야 한다는 것에 초점을 맞추고 있어 디바이스 제어와 관련하여 광범위한 이슈는 언급하고 있지 않다.

Singh의 연구는 IoT 애플리케이션 개발과 관련하여 네트워크에 연결된 디바이스들의 유일한 주소 체계와 여러 디바이스로부터 수집된 방대한 양의 데이터 처리 관련 이슈를 제기하였고, 이를 처리하기 위해 시맨틱 웹 기술을 기반으로 하는 아키텍처 모델을 제안하였다[8]. 제시된 이슈들은 많은 수의 IoT 디바이스가 참여하고 이로부터 방대한 양의 데이터가 수집될 필요가 있는 대규모 IoT 애플리케이션 개발에 관련된 것들이며, 제안된 솔루션은 아키텍처 수준에서 개략적으로 기술되고 있다.

이외에 Xu의 연구[9]와 같이 IoT 개념을 특정 도메인에 적용하여 애플리케이션을 개발하면서 당면한 이슈를 제기하고, 이에 대한 해결책을 제시하는 연구들도 있다. Xu의 연구는 IoT 기술 기반의 물류 관리 시스템 개발과 관련하여 디바이스 연결 및 데이터 저장 이슈 등을 제기하고, 이 문제를 해결하기 위한 아키텍처를 제시하였다.

현재까지 여러 연구에서 IoT 애플리케이션 개발과 관련하여 여러 이슈들을 제기하고 있지만, 다수의 디바이스들로 구성되고 방대한 양이 수집되는 대규모 IoT 애플리케이션 개발과 관련된 이슈들이 대부분이며, 이 이슈들은 소프트웨어 공학뿐 아니라 데이터베이스, 네트워크 등 다양한 컴퓨팅 분야와 관련된 것이다. 본 논문에서는 소프트웨어 공학 관점에서 IoT 애플리케이션 개발 시에 전형적인 소프트웨어 개발에서 관찰할 수 없는 이슈들을 위주로 정리한다. 그리고 도출한 이슈들을 해결할 수 있는 소프트웨어 공학 기법을 근간으로 한 솔루션을 구체적으로 제시한다.

3. IoT 애플리케이션 개발의 비전형적인 이슈

대부분의 IoT 애플리케이션은 하나 이상의 IoT 디바이스들과 상호작용하면서 사용자에게 서비스를 제공한다. 그러므로 전형적인 소프트웨어와는 달리 IoT 디바이스를 제어하면서 그 기능을 수행해야 하므로, IoT 디바이스 관리 기능이 반드시 고려되어야 한다. 그리고 IoT 디바이스는 임베디드 컴퓨팅, 모바일 컴퓨팅, 퍼베이시브 컴퓨팅, 상황인지 컴퓨팅에서 고려한 디바이스들과 차이점이 있다. 이에 따라 Fig. 1과 같이 IoT 애플리케이션 개발은 소프트웨어 시스템 개발에서는 발견되지 않는 비전형적인 기술적 이슈를 가지고 있다. IoT 애플리케이션 설계 시 발견할 수 있는 비전형적인 이슈는 모두 IoT 디바이스 사용으로 인해 생긴다. IoT 디바이스는 이질성, 이동성, 디바이스의 물리적 제약성 특징을 가지므로, 본 절에서는 이로부터 유도되는 관련 이슈들을 정의한다.

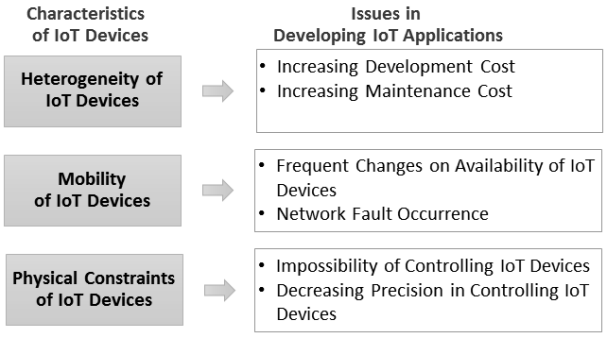

Fig. 1. Characteristics of IoT Devices and their Relevant Issues

3.1 이질성(Heterogeneity) 관련 이슈

IoT 애플리케이션은 여러 IoT 디바이스와 상호 연동한다. 이런 IoT 디바이스들은 다양한 벤더에서 제공되고 있고 IoT 디바이스 종류에 따라 각각 고유한 특징을 가지고 있으므로, IoT 디바이스들은 상당한 이질성을 가지고 있다. IoT 디바이스들이 가질 수 있는 이질성 종류를 정리하면, Table 1과 같다. 이는 현재 시중에 나와있는 여러 센서들, 스마트폰, 다양한 로봇 등을 포함한 IoT 디바이스들을 조사 분석하여 얻은 결과이다.

Table 1. Kinds of Heterogeneity on IoT Devices

Kinds of Heterogeneity	Possible Values
API	Different sets of APIs provided by IoT device vendors
Network Protocol	TCP, UDP, HTTP, Bluetooth, ZigBee, ANT, 6LoWPAN, DASH7, ONE-NET, Z-Wave, Wibree, WirelessHART, 802.15.4
Interface Paradigm	Request and Response, Publish and Subscribe, Polling

이질성 종류(Kinds of Heterogeneity)는 IoT 디바이스 벤더들이 IoT 디바이스를 제어하기 위해서 제공하는 SDK상에서 식별될 수 있는 차이점을 기반으로 도출한 것이다. 먼저, IoT 벤더들은 각각 다른 API로 디바이스를 제어하며, 온도 센서와 같이 동일한 종류의 IoT 디바이스라도 이 디바

이스로부터 값을 읽어오거나 특정 기능을 수행하는 API는 상이하다. 두 번째로, 각 디바이스마다 제공하는 네트워크 프로토콜이 다르다. 이는 하드웨어 스펙상의 차이에서 기인되지만, Wi-Fi만을 지원하거나 Bluetooth를 지원하는 등의 IoT 디바이스 연결을 가능하게 하는 네트워크 프로토콜에도 차이가 있다. 마지막으로, 각 디바이스로부터 센서값을 읽어오는 방식(Interface Paradigm)에 차이가 있다. 대부분 디바이스는 Request and Response 방식으로 사용자가 필요할 때마다 IoT 디바이스로부터 정보를 읽어오지만, 이외에도 특정한 정보가 필요하다는 요청을 IoT 디바이스에 등록하게 되면 해당 정보를 계속해서 보내주는 Publish and Subscribe 방식, 일정한 시간 간격으로 IoT 디바이스로부터 정보를 가져오는 Polling 방식도 지원하기도 한다. 각 디바이스마다 지원하는 인터페이스 패러다임 방식이 차이가 있을 뿐 아니라, 디바이스가 제공하는 방식과 목표 IoT 애플리케이션이 데이터 수집을 필요로 하는 방식 간에도 이질성이 존재할 수 있다. Table 1에 있는 내용 외에 IoT 디바이스가 제공하는 센서 및 액추에이터(Actuator) 종류, IoT 디바이스의 연산 능력 제공 여부 등과 같이 하드웨어 스펙상에서 유도되는 이질성이 있다. 그러나 이들은 IoT 애플리케이션 설계 시 크게 고려되지 않은 사항이므로, 표의 내용에서 제외되었다.

IoT 디바이스 이질성은 다음과 같이 IoT 애플리케이션 개발에 영향을 미친다.

- 다양한 IoT 디바이스 관리 기능이 부가적으로 필요하므로, IoT 애플리케이션 개발 복잡도가 높아짐.
  IoT 애플리케이션 개발 시에 IoT 디바이스의 다양한 이질성을 고려해서, 각 IoT 디바이스에 맞게 적절한 방식으로 IoT 디바이스를 제어해야 한다. IoT 애플리케이션은 요구사항 명세서에 기술된 요구사항을 만족시키기 위해 소프트웨어 기능성 위주의 설계 외에 IoT 디바이스 제어 기능도 설계되어야 한다. IoT 디바이스의 이질성으로 인해 IoT 애플리케이션에서 제어하는 IoT 디바이스 수가 증가할수록 디바이스 제어 기능 설계에 대한 오버헤드도 급증하게 된다. 그 결과로, 소프트웨어 기능 설계에 대한 노력보다 더 많은 노력을 필요로 하여 그 개발 부담이 역시 더욱더 높아지게 된다.
- IoT 디바이스 교체로 인한 IoT 애플리케이션 유지보수가 어려워짐.
  현재 다양한 종류의 IoT 디바이스들과 함께 이를 제어하는 SDK가 빠르게 출시되고 있다. 그리고 새로운 버전의 IoT 디바이스들이 출시될 때, 해당 디바이스를 보다 효과적으로 제어하기 위해 거의 새로운 형태의 SDK가 제공되기도 한다. 그러므로 기존 IoT 애플리케이션을 새롭게 출시된 IoT 디바이스와 상호 연동되도록 수정해야 하는 경우, 호환성 문제가 생겨 기존 애플리케이션의 코드를 상당수 수정해야 할 필요가 생기게 된다. 게다가, IoT 디바이스 제어 기능은 여러 소프트웨어 기능에서 모두 사용할 수 있으므로, 동일한 디바이스 제어 기능이 여러 곳에 산재되어 포함될 수 있어 유지보수가 더욱더 어려워질 수 있다.

3.2 자체 이동성(Intrinsic Mobility) 관련 이슈

IoT 디바이스가 기존 소프트웨어에서 제어하던 하드웨어와 가장 크게 다른 점은 IoT 디바이스가 갖는 이동성이다. IoT는 그 종류가 매우 다양하기 때문에, 이동성 종류도 다양하다. 첫째, 기존 하드웨어와 동일하게 이동이 되지 않는 IoT 디바이스가 있다. 둘째, 스마트폰처럼 사람이 디바이스를 가지고 이동함으로써 나타나는 이동성이 있고, 이를 “외부 요인에 의한 이동성(Extrinsic Mobility)”이라고 한다. 마지막으로, 모터와 같이 디바이스 자체가 이동할 수 있는 능력이 있어 자발적으로 이동할 수 있는 “자체 이동성(Intrinsic Mobility)”이 있다. 로봇, AR Drone[10], Sphero Ball[11] 등이 자체 이동성을 가진 IoT 디바이스의 대표적인 예이다. 이 중, 자체적 특성에 의한 이동성은 IoT 디바이스에서만 볼 수 있는 것으로, 사용자가 접근할 수 없는 장소로 이동하여 다양한 정보를 감지 및 전달할 수 있기 때문에 사용자에게 매우 유용한 서비스를 제공하는 데 주요한 역할을 한다.

IoT 디바이스의 이동성은 스마트 서비스(Smart Service) 제공에 핵심 역할을 하지만, Fig. 2와 같이 잠재적인 문제를 발생시킬 수 있는 주요 원인이기도 하다.

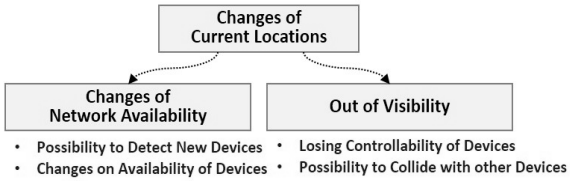

Fig. 2. Potential Faults caused by Mobility

먼저, IoT 디바이스의 이동성으로 인해 디바이스가 현재 있는 위치 또한 변경될 수 있다(Change of Network Availability). IoT 디바이스의 위치 변경은 신호 강도 및 네트워크 연결 여부와 같은 네트워크 연결 상태를 동적으로 변화시킬 수 있으며, 이로 인해 다음과 같이 네트워크 연결 관련 문제가 발생할 수 있다.

- IoT 애플리케이션은 새롭게 네트워크에 참여한 디바이스와 상호 연동할 수 없음.
  IoT 디바이스가 이동하면서 새로운 네트워크 범위에 진입할 수 있다. 그러나 IoT 애플리케이션은 이런 디바이스와 런타임에 상호 연동할 수 없다. 그러므로 IoT 애플리케이션은 현재 가용한 디바이스를 즉시 파악하여 최적의 디바이스와 상호작용할 수 있어야 한다.
- 기존에 연결하여 사용한 IoT 디바이스의 이용 가능 상태(Availability)가 수시로 변함.
  IoT 디바이스는 특정 네트워크 범위 내에 있는 경우에만 IoT 애플리케이션과 상호작용할 수 있다. 그러나

네트워크 범위는 제한되어 있고 IoT 디바이스는 네트워크 범위에 국한되지 않고 이동할 수 있으므로, 네트워크 상태가 Fig. 3과 같이 변한다. Fig. 3에서 구름 형태는 네트워크 영역을 나타내며, 네트워크 영역의 바탕색의 진하기 정도는 네트워크 신호 세기를 나타낸다. 예를 들어, Case #5는 네트워크 신호가 점점 약해지는 지역으로 이동하고 있는 상황을 보여준다. Case #1, Case #2와 같이 네트워크 범위 안팎을 이동하면서 특정 네트워크 범위를 벗어나거나 진입할 수 있다. 그리고 Case #3, Case #4, Case #5와 같이 다른 네트워크 영역으로 이동하면서 신호의 세기가 점점 약해지기 때문에, IoT 애플리케이션과의 네트워크 지연이 발생할 수 있다. 네트워크 신호 세기 및 범위 이탈 여부에 따라서 IoT 디바이스의 이용 가능성 상태 역시 변한다. 즉, IoT 디바이스가 불시에 사용할 수 없는 상태가 될 수 있으므로, 이 역시 관리되어야 한다.

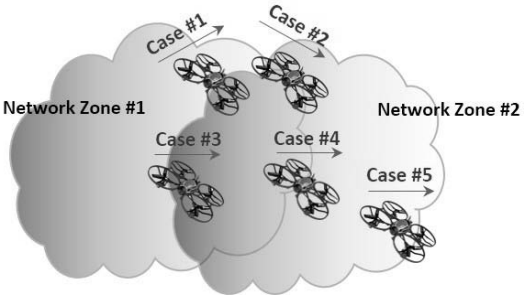

Fig. 3. Changes on Network Strengths according to Device Movement

Fig. 2에서 나타낸 것과 같이, 네트워크 연결 관련 문제 외에 IoT 디바이스가 사각지대로 이동하면서 발생하는 문제가 있다(Out of Visibility). 이는 사람이 IoT 디바이스 이동에 관여할 수 없으므로 발생하게 된다.

- 사용자의 가시성 범위를 벗어나서 제어권(Controllability)을 잃음.

  IoT 디바이스의 자체 이동성으로 인해서, 사용자가 장비의 현 상태를 시각적으로 확인하지 못하는 경우가 발생한다. AR Drone과 같은 경우는 정해진 목표 지점으로 경로를 만들어서 이동할 수 있지만, 사용자가 직접 제어해서 이동시킬 수도 있다. 후자의 경우에 AR Drone이 장애물 뒤로 이동하여 사용자의 가시성을 잃으면, 디바이스 제어를 할 수 없게 된다.

- 주변 장애물과 충돌로 인해 IoT 디바이스를 사용할 수 없게 됨.

  IoT 디바이스가 장애물에 막혀있는 경우와 같이 IoT 디바이스에 발생하는 일부 문제는 사람이 직접 해결할 수 있다. 그러나 가시성 범위를 벗어나게 되면 IoT 디바이스에 발생하는 문제를 즉시 처리할 수 없어 IoT 애플리케이션의 일부 기능을 활용할 수 없게 된다.

3.3 물리적 제약성(Physical Constraints) 관련 이슈

대부분의 IoT 디바이스는 작게 제작되어 있으며, 배터리 등 자원이 매우 제한적이다. Fig. 4는 IoT 디바이스 하드웨어 관련 제약성의 종류와 그로부터 유도되는 이슈를 보여준다.

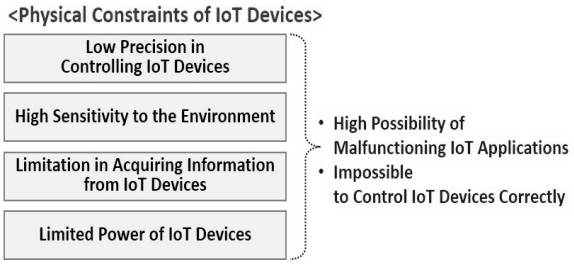

Fig. 4. Potential Issues caused by Physical Constraints

첫째, IoT 디바이스를 제어할 수 있는 API가 제공되어도 그 정교성(Precision)이 떨어지는 경우가 있다. 기존에는 정보 획득 수준의 디바이스들 위주로 제작되었고 액추에이션이 가능한 IoT 디바이스가 제작된 지 얼마 되지 않았다. 그러므로 IoT 디바이스의 하드웨어를 액추에이션 하는 부분의 정교함이 약간 떨어진다. 예를 들어, 공 형태를 띠고 스스로 이동할 수 있는 Sphero Ball[11]을 90° 회전하려고 할 때, 진행 속도에 따라 회전 오차가 발생하여 타원형 형태로 이동한다.

둘째, IoT 디바이스는 주변 환경에 매우 민감하게 제어된다. 일부 IoT 디바이스는 환경에 따라 액추에이션 기능을 정확하게 수행하지 못하는 경우가 발생한다. 예를 들어, AR Drone은 바람세기에 따라서 정해진 경로로 진행하지 못하며, Sphero Ball은 바닥에 저항이 있는 경우에 직진으로 이동을 못한다.

셋째, IoT 디바이스를 제어할 수 있는 API가 부족한 경우이다. IoT 디바이스를 제어할 수 있는 유일한 방법은 각 벤더에서 제공하는 API를 통하는 것이다. 그러나 일부 기능을 API로 제공하지 않기 때문에 IoT 디바이스를 제어하는 데 제약이 생긴다. 특히, IoT 디바이스의 현재 상태를 획득하는 일부 API가 제공되지 않는 경우가 있다. 예를 들어, Sphero Ball이 벽이나 장애물에 충돌한 경우, 이를 감지할 수 있는 API는 제공되지만, 충돌의 강도, 충돌로 인해 반사 이동의 각과 속도 등의 Sphero Ball 상태를 읽을 수 있는 기능은 제공되지 않는다. 따라서 충돌에 따른 IoT 장비의 제어를 위해서는 그 디바이스 자체의 상태 정보를 활용해서 이런 기능 오작동을 제어할 수 없게 된다.

마지막으로, 대부분의 IoT 디바이스는 제한된 배터리 전원을 사용한다. 그러므로 배터리 잔량은 IoT 디바이스 기능 수행에 중대한 영향을 끼친다. AR Drone의 경우, 배터리가

완전 충전되었을 경우에는 4개의 프로펠러 모터가 정상 속도를 유지하지만, 잔량이 50% 이하인 경우에는 일부 모터의 속도 저하로 원래 의도대로 이동하지 않는다. 게다가, 배터리가 방전되었을 경우에는 해당 IoT 디바이스를 사용할 수 없게 된다. 이런 특성들로 인해 IoT 애플리케이션 설계 시에 고려해야 하는 다음과 같은 이슈가 발생한다.

- IoT 애플리케이션 기능이 오작동(Malfunction)함.
  IoT 디바이스의 물리적 제약성으로 인해 디바이스가 의도한대로 작동하지 않을 수 있다. 이로 인해, IoT 애플리케이션 역시 기능 수행에 오류가 발생한다.
- IoT 디바이스를 제어할 수 없음.
  IoT 디바이스의 물리적 제약성 중 배터리 방전으로 인해 IoT 디바이스를 전혀 제어할 수 없는 경우가 발생한다. 이 경우 IoT 애플리케이션은 기능을 수행할 수 없게 되므로, IoT 디바이스의 배터리를 수시로 확인하여 방전 위험이 있는 디바이스와 상호 연동하지 않도록 설계되어야 한다.

4. 비전형적인 이슈에 대한 솔루션 공간

3절에서 언급한 이슈들은 IoT 애플리케이션 전 영역에 걸쳐서 발생하기 때문에, IoT 애플리케이션의 여러 오퍼레이션에서 IoT 디바이스 관련 이슈를 해결하는 기능이 위치한다. IoT 디바이스 관련 이슈를 해결하는 기능을 IoT 애플리케이션 고유의 기능과 같이 위치시키면, 개발 복잡도가 더욱더 높아지고 유지보수 역시 힘들어진다. 그러므로 IoT 디바이스와 관련한 이슈들을 관리하는 모듈을 별도로 설계하는 것이 효과적이다. 이 이슈들을 해결하는 기능들은 관점 지향 프로그래밍에서의 횡단관심사(Cross-cutting Concerns)와 같이 취급하여, 별도의 모듈로 구현할 것이다. 본 논문에서는 이슈를 해결하는 솔루션을 구현한 모듈을 앱 에이전트(App Agent)라고 한다. Fig. 5는 기존 방식과 에이전트를 이용한 방식의 IoT 애플리케이션 구현 형태의 차이를 보여준다.

Fig. 5의 상단 부분과 같이 기존에는 각 오퍼레이션마다 디바이스를 호출 및 제어하는 기능이 산재되어있는 형태로 구현되었다. 그러므로 동일한 디바이스를 제어하는 기능이 여러 오퍼레이션에 중복으로 위치하게 되었다. 그 결과로, 한 디바이스의 기능을 수정할 경우, 여러 오퍼레이션을 모두 수정해야 하는 어려움이 발생하였다.

그러나 본 논문에서는 Fig. 5의 하단 부분과 같이 별도의 에이전트에서 디바이스 호출로 인해 발생하는 이슈를 해결하며, IoT 애플리케이션은 에이전트와만 상호작용하도록 설계하도록 구현한다. 이렇게 함으로써 디바이스 사용으로 인한 이슈 해결이 한 모듈에서 이루어지고, 기존 방식의 개발 복잡도가 높아지고 유지보수가 어려워지는 문제점 역시 해결할 수 있다. 이 절에서 언급되는 솔루션은 앱 에이전트 설계와 관련이 있다.

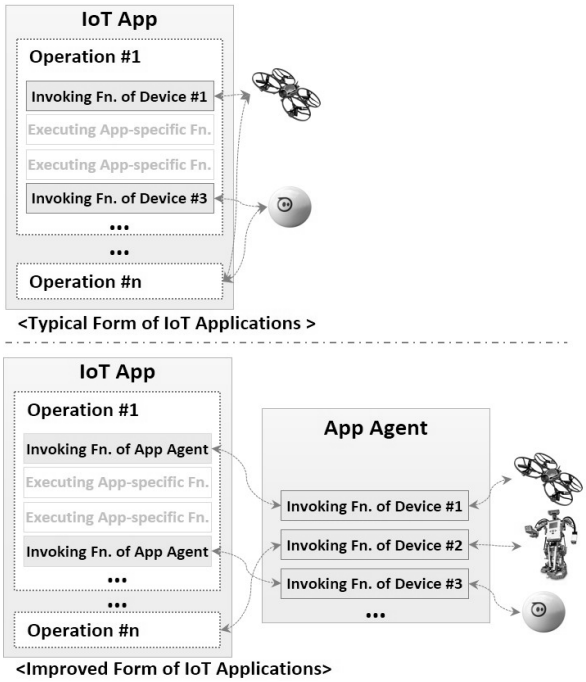

Fig. 5. Separation of Functionalities in IoT Applications

4.1 이질성 관련 이슈의 솔루션

이질성 관련 이슈를 해결하기 위한 가장 근본적인 방법은 IoT 애플리케이션에서 표준화된 인터페이스를 통해 기능을 호출하게 하는 것이다. 표준화된 인터페이스를 통하여 디바이스를 제어함으로써, IoT 애플리케이션은 IoT 디바이스 API의 변경, 네트워크 프로토콜 변경, 데이터 교환 방식의 변경에 영향을 덜 받는다. 대신에, IoT 디바이스 관리를 담당하는 앱 에이전트에서 변경에 따른 작업을 수행해야 하며, 앱 에이전트 역시 변경에 따라 코드를 적게 수정하도록 설계되어야 한다. 그리고 다른 이질성 종류에 따라 개발 비용과 유지보수 비용을 절감할 수 있도록 적절한 디자인 패턴을 활용하여 앱 에이전트를 설계한다.

이를 위해, 먼저 목표 IoT 애플리케이션 설계 시에 디바이스 제어와 관련하여 필요한 기능을 식별하고, 이를 앱 에이전트가 수행하도록 설계해야 한다. Fig. 6은 Table 1에서 살펴보았던 다양한 이질성을 고려한 IoT 애플리케이션 및 앱 에이전트 설계 프로세스를 보여준다.

첫 번째 단계에서 목표 IoT 애플리케이션과 상호작용해야 하는 IoT 디바이스 종류를 식별한다. 이때, 요구사항에 작성된 내용에 따라 AR Drone, Arduino 등 특정 IoT 디바이스를 식별할 수 있고, 온도 및 습도를 측정하고 세팅할 수 있는 디바이스와 같이 필요한 IoT 디바이스 종류만 선정할 수 있다.

두 번째 단계에서 IoT 애플리케이션에서 필요로 하는 IoT 디바이스 관련 기능을 정의한다. 이 기능은 IoT 디바이스가 제공하는 기능, IoT 디바이스를 관리하기 위해 사전에

Fig. 6. A 5-Step Process to Resolve Issues on Heterogeneity

수행되어야 하는 기능으로 분류된다. 전자인 IoT 디바이스가 제공하는 기능은 온도 측정, 습도 측정 등 IoT 애플리케이션에서 사용자에게 서비스를 제공하기 위해서 사용되는 것으로, 센싱 및 액추에이션 기능이 대부분이며 요구사항 명세서로부터 쉽게 도출될 수 있다. 후자로 언급된 IoT 디바이스를 관리하기 위해 필요한 기능은 IoT 애플리케이션 요청에 맞게 센싱 및 액추에이션 기능을 제대로 수행하기 위해 필요한 기능으로 디바이스 연결 및 오류 제어 등이 이에 속한다. 3절에서 언급한 다양한 이슈를 해결하는 기능 역시 이 분류에 속한다.

세 번째 단계에서 해당 IoT 애플리케이션에서 사용하는 디바이스 간의 이질성 종류를 파악한다. 목표 IoT 애플리케이션의 요구사항 및 사용되는 디바이스에 따라서 Table 1에서 언급된 3가지 이질성 중 일부만 도출될 수 있다. 첫 번째 단계와 두 번째 단계의 결과물을 이용하여 Table 2와 같은 테이블을 작성한다.

첫 번째 열은 이질성이 발생할 수 있는 항목들을 나열하고, 두 번째 열부터는 목표 IoT 애플리케이션에서 사용하는 디바이스에 대한 정보를 기술한다. API 이질성의 경우에는 두 번째 단계에서 식별한 기능들을 나열하고, 각 디바이스별로 이를 수행하기 위해 사용해야 하는 API 정보를 각 디바이스에 해당하는 열에 작성한다. 네트워크 프로토콜 이질성과 인터페이스 패러다임 이질성은 각 디바이스별로 지원하는 프로토콜 및 데이터 교환 방식을 작성한다. 마지막 열은 이질성이 발생하는지의 여부를 Y 또는 N로 작성한다.

네 번째 단계에서는 두 번째 단계에서 식별한 기능들에 대한 표준 인터페이스를 정의한다. 이때, 이 인터페이스는 디바이스들 간의 API 이질성을 해결하는 데 주요한 역할을 한다. 목표 IoT 애플리케이션의 모든 컴포넌트는 이 API를 통해 디바이스와 상호작용하도록 설계하여, IoT 애플리케이션은 이질성에 영향을 덜 받도록 한다.

마지막 단계에서는 이전 단계의 결과물을 모두 활용하여 앱 에이전트를 설계한다. 이때 각 이질성 종류별로 효과적으로 적용될 수 있는 디자인 패턴[12]을 적용하여 앱 에이전트를 설계하며, 그 종류는 Table 3과 같다. 이는 [13]의 연구 내용을 기반으로 확장한 것이며, 표에 기술된 패턴 외에 다른 패턴도 설계에 적용될 수 있다.

Table 2. A Template for Analyzing Heterogeneity

		Device 1	...	Device n	Heterogeneity (Y/N)
APIs for Requred Fn.	Fn. #1	getTemp()		readTemp()	Y
	...	...		...	...
	Fn. #n	...		...	...
Network Protocols		WiFi		WiFi	N
Interface Paradigm		Request /Reply		Publish /Subscribe	Y

Table 3. Design Patterns for Resolving Heterogeneities

Kinds of Heterogeneity	Applicable Design Patterns
API	Adapter Pattern
Network Protocol	Strategy Pattern, Template Method Pattern
Interface Paradigm	Proxy Pattern

API 이질성은 메시지 송신 객체와 수신 객체 간의 인터페이스 차이를 해결하는 데 활용되는 어댑터(Adapter) 패턴을 적용하여 해결할 수 있다. Fig. 7은 어댑터 패턴을 적용하여 설계된 앱 에이전트 설계의 일부를 보여준다.

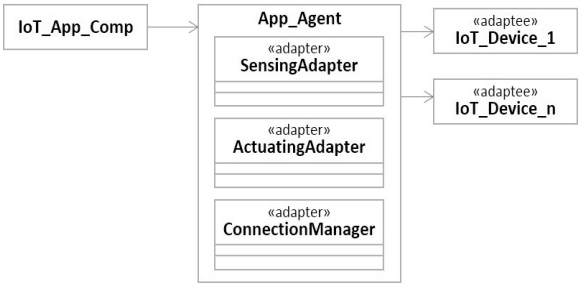

Fig. 7. Applying Adapter Pattern

앱 에이전트(App_Agent)를 구성하는 컴포넌트는 두 번째 단계에서 도출된 기능을 모두 수행할 수 있도록 도출되어야 한다. 일반적으로 IoT 애플리케이션은 IoT 디바이스로부터 센싱 및 액추에이션 기능 및 디바이스 연결 등의 기능을 필요로 하므로, 앱 에이전트는 Fig. 7과 같이 각각을 담당하는 컴포넌트인 SensingAdapter, ActuatingAdapter, ConnectionManager로 구성될 수 있다. 그리고 이 컴포넌트의 모든 오퍼레이션은 네 번째 단계에서 정의한 표준 인터페이스로 정의되어야 한다.

네트워크 프로토콜 이질성은 네트워크 프로토콜별로 준수해야 하는 절차대로 디바이스를 연결하도록 설계함으로써 해결될 수 있다. 전략(Strategy) 패턴 또는 템플릿 메소드(Template Method) 패턴은 동일 인터페이스로 다양한 알고리즘을 제공해야 할 때 활용될 수 있기 때문에, 네트워크 프로토콜 이질성 이슈를 해결하는 데 효과적으로 적용될 수 있다. Fig. 7의 ConnectionManager 컴포넌트 설계에 전략 패턴 또는 템플릿 메소드 패턴을 적용하여, 하위 클래스로 각 네트워크 프로토콜별 연결 컴포넌트를 위치시키고 각각

에 맞는 알고리즘을 구현하도록 한다. 이렇게 하면, 목표 IoT 애플리케이션의 컴포넌트는 표준화된 인터페이스를 통해 이질성을 가지는 다양한 디바이스를 연결하기 때문에, 네트워크 프로토콜 이질성에 영향을 덜 받는다.

마지막으로, 인터페이스 패러다임의 이질성은 목표 IoT 애플리케이션이 필요로 하는 방식에 맞게 이질적인 IoT 디바이스가 센서값을 전달할 수 있도록 설계함으로써 해결될 수 있다. IoT 애플리케이션은 인터페이스 패러다임 이질성을 고려하지 않고 센서값을 요청하지만, 중간에서 인터페이스 패러다임 이질성을 중재하여 센서값을 읽어야 하므로, 프록시(Proxy) 패턴을 적용한다. 특히, Fig. 8과 같이 IoT 애플리케이션과 IoT 디바이스 사이에 센서값을 임시로 저장하는 Device Context Pool을 위치시켜 인터페이스 패러다임 이질성을 중재하도록 한다. 그리고 이 설계는 Fig. 7의 SensingAdapter 부분에 적용한다.

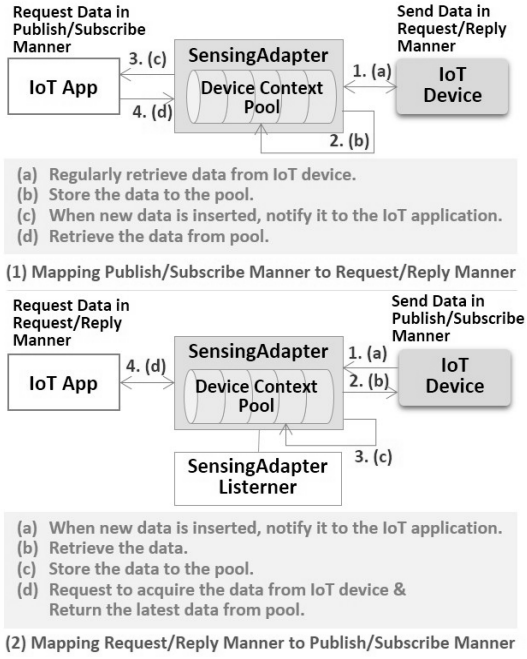

Fig. 8. Resolving Interface Paradigm with Proxy Pattern and Device Context Pool

Fig. 8의 상단 부분은 IoT 애플리케이션에서는 Publish/Subscribe 방식으로 데이터를 요청하지만, 상응하는 IoT 디바이스는 Request/Reply 방식으로만 센서값을 반환하는 경우를 보여준다. 이 경우, SensingAdapter는 사전에 정기적으로 IoT 디바이스로부터 센서값을 Request/Reply 방식으로 읽어와서(a 단계) 바로 이전에 저장된 값과 다를 경우에만 Pool에 저장한다(b 단계). Pool에 새로운 값이 저장되는 동시에 SensingAdapter는 IoT 애플리케이션에 새로운 값을 획득하였음을 알리고(c 단계), IoT 애플리케이션이 필요 시에 Pool로부터 획득된 값을 읽을 수 있도록 한다(d 단계).

Fig. 8의 하단 부분은 반대로, IoT 애플리케이션은 Request/Reply 방식으로 데이터를 요청하지만, IoT 디바이스는 Publish/Subscribe 방식으로만 센서값을 반환하는 경우를 보여준다. 이 경우, SensingAdapter는 리스너를 통해 IoT 디바이스로부터 Publish/Subscribe 방식으로 값을 읽어와서(a 단계와 b 단계), Pool에 값을 저장한다(c 단계). 그리고 IoT 애플리케이션에서 Request/Reply 방식으로 센서값을 요청할 때, IoT 디바이스 대신에 Pool로부터 최신 값을 반환한다(d 단계).

위와 같이 Device Context Pool과 프록시 패턴을 적용하여 SensingAdapter를 설계함으로써, IoT 애플리케이션은 IoT 디바이스의 인터페이스 패러다임에 영향을 받지 않고, 필요한 방식대로 센서값을 획득할 수 있게 된다.

4.2 자체 이동성 관련 이슈의 솔루션

자체 이동성으로 인한 이슈는 Active Device Pool을 이용한 방법과 디바이스가 제공하는 다른 센서를 이용하는 두 가지 방법에 의해 해결될 수 있다.

Active Device Pool을 이용한 디바이스 동적 발견: 자체 이동성에 따른 IoT 디바이스의 네트워크 연결성이 자주 변하는 문제와 제한된 가시성으로 인해 디바이스 제어의 어려움을 해결하기 위해서 IoT 애플리케이션은 수시로 이용 가능한 IoT 디바이스와 IoT 디바이스의 연결 상태를 파악해야 한다. 동시에 이런 사용 가능한 IoT 디바이스의 연결 상태 파악은 적시에 이루어져야 한다. 그러므로 앱 에이전트는 현재 가용한 디바이스를 확인하여 이에 대한 목록을 관리하는 기능과 디바이스의 연결 상태를 수시로 확인하는 기능을 제공해야 한다. 그리고 IoT 애플리케이션은 디바이스에 기능을 호출하기 이전에 에이전트로부터 가용한 디바이스 및 연결 상태에 대한 정보를 획득한 후 적절한 디바이스와 상호작용해야 한다. 이를 위해, [14][15]의 연구 결과를 확장하여, Fig. 9와 같이 앱 에이전트를 설계한다.

Pool Manager와 Active Device Pool은 자체 이동성을 관리하기 위해 필요한 컴포넌트 중 가장 핵심적인 역할을 한다. Active Device Pool은 현재 가용한 디바이스들의 프로파일 정보를 임시로 저장하며, IoT 애플리케이션을 통해 제어할 수 있는 디바이스를 런타임 시에 발견할 수 있도록 도와준다. Pool Manager는 새롭게 네트워크에 진입한 디바이스 정

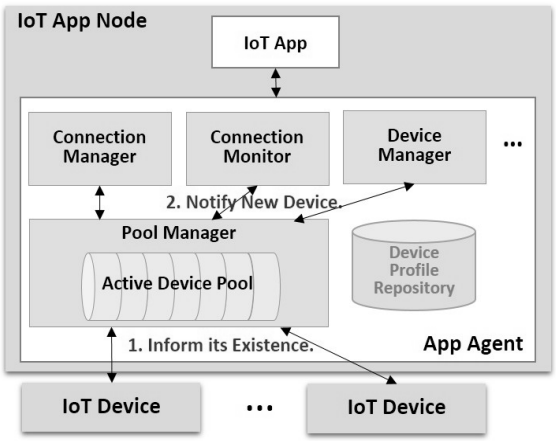

Fig. 9. Design of App Agent to Resolve Intrinsic Mobility

보를 Active Device Pool에 저장하며, 네트워크에 없는 디바이스 정보를 Active Device Pool에서 삭제하는 등의 Active Device Pool을 관리하는 역할을 한다. Connection Manager는 네트워크 프로토콜에 맞게 디바이스를 연결하는 작업을 수행하고, Connection Monitor는 Active Device Pool에 있는 각 디바이스들이 실제로 연결 가능한 상태인지를 수시로 체크한다. Device Manager는 Active Device Pool에 새롭게 저장된 디바이스 정보가 Device Profile Repository에 없으면 그 정보를 저장하는 역할을 한다. Device Profile Repository는 IoT 애플리케이션이 사용한 디바이스를 영구적으로 저장하며, 세션이 종료된 이후에도 해당 디바이스를 활용할 수 있게 한다.

Fig. 9는 새로운 디바이스가 사용 가능할 때의 동적 흐름 역시 보여준다. 먼저, IoT 디바이스는 Pool Manger를 통해 해당 디바이스가 가용 상태임을 알린다(1단계). 이때, IoT 디바이스 역시 자신의 상태를 알려주는 기능을 구현한 에이전트가 설치되어 있어야 한다. Pool Manager가 새로운 디바이스의 정보를 Active Device Pool에 저장하면, 새로운 디바이스가 가용한 상태임을 관련 컴포넌트에 즉시 알려야 한다(2단계). 이를 위해, Pool Manager와 Connection Manager, Connection Monitor, Device Manager는 옵서버(Observer) 패턴을 적용하여 설계된다. 그리하여, 새로운 디바이스가 Active Device Pool에 저장되면, 나머지 컴포넌트가 이를 인지하여 새로운 디바이스를 포함하여 각자 역할을 수행하도록 한다. Fig. 10은 옵서버 패턴을 적용한 설계 모델이다.

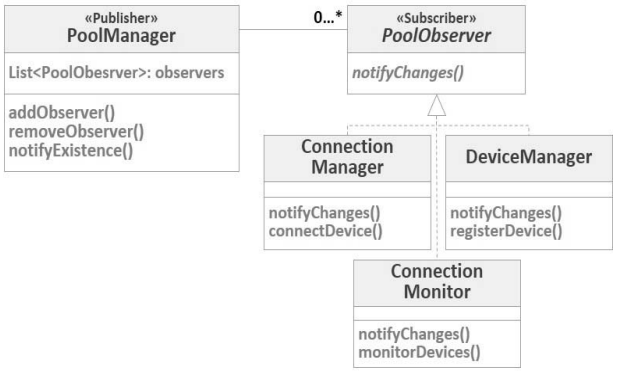

Fig. 10. Design to Publish New Device Existence with Observer Pattern

Pool Manager는 Publisher 역할을 하여 새로운 디바이스가 Active Device Pool에 저장되면 이 정보를 PoolObserver 인터페이스를 상속받은 Connection Manager, Connection Monitor, Device Manager에게 알려준다. 그리하여, Connection Manager는 새로운 디바이스를 필요 시에 연결할 수 있게 되고, Connection Monitor는 이 디바이스를 모니터링 대상으로 등록한다. 그리고 Device Manager는 새롭게 진입된 디바이스가 Device Profile Repository에 없으면 이 정보를 저장한다.

Active Device Pool과 Pool Manager를 이용하면 IoT 애플리케이션이 새롭게 진입한 디바이스를 자동으로 감지할 수 있지만, 디바이스의 이동성으로 연결 상태가 수시로 변경될 수 있다. 그러므로 IoT 애플리케이션이 필요한 디바이스를 연결하기 전에 디바이스의 연결 상태를 다시 한 번 확인할 필요가 있다. 그러므로 [15]의 연구 결과에서와 같이 Connection Manager는 디바이스 연결 전에 Ping 메시지를 전송하고 응답 메시지가 시간 내에 도착하는지 확인하는 알고리즘을 작성해야 한다.

디바이스의 카메라 기능 또는 주변 상황을 파악할 수 있는 컨텍스트 분석 기법을 활용한 가시성 확보: 자체 이동성과 관련된 또 다른 이슈인 제한된 가시성으로 인한 문제점은 사람이 직접 그 디바이스를 볼 수 있는 방법을 제공하지 않고는 해결하기 어렵다. AR Drone과 같은 디바이스는 자체적으로 비디오 촬영 기능이 있기 때문에 이를 이용하면 가시성 확보에 도움이 될 수 있지만, 모든 디바이스가 이런 기능을 제공하는 것은 아니다. 그러므로 사용하는 디바이스의 기능성 스펙을 이해하고, 가시성을 확보할 수 있는 방법을 고안해야 한다.

카메라가 부착된 디바이스의 경우 비디오 촬영 기능을 추가적으로 구현하여, 사용자가 카메라를 통해 디바이스를 제어할 수 있게 한다. 비디오 촬영 기능이 제공되지 않은 디바이스의 경우에는, 근접도 센서, 울트라소닉(Ultrasonic) 센서 등을 활용하여 주변 장애물 여부를 파악하도록 도움을 줄 수 있다. 이를 위해, 앱 에이전트는 디바이스의 장애물을 파악할 수 있는 센서로부터 획득되는 정보를 임시로 저장할 수 있는 큐를 가지도록 설계한다. 그리고 컨텍스트 분석 기법[16]을 활용하여 전방에 장애물이 있는지를 확인하는 기법을 구현하여, 장애물이 있으면 멈추거나 다른 방향으로 우회할 수 있도록 기능을 설계해야 한다. 그러나 주변 장애물을 피하는 기능을 제공하는 데 필요한 센서가 부착되어있지 않으면, 가시성을 확보하기가 어려우므로 일부 경우에만 해당 솔루션이 적용될 수 있다.

4.3 물리적 제약성 관련 이슈의 솔루션

물리적 제약성으로 인한 이슈는 크게 디바이스의 센서값을 조정하는 방법과 디바이스를 교체하는 방법으로 해결될 수 있다.

지속적인 센서값 모니터링을 통한 센서값 보정: 자체 디바이스 제어 API가 미성숙하거나 주변 환경에 민감하여 의도대로 움직이지 않는 IoT 디바이스를 제대로 동작시키기 위해서는 센서값 보정(Calibration)이 반드시 필요하다.

디바이스 API 자체의 결함이 있을 경우에는 먼저 해당 API에 대한 테스트를 거쳐 디바이스가 오작동하는 원인 및 오차 범위를 파악해야 한다. 앞서 Sphero Ball의 예제와 같이 90° 회전하려고 할 때, 회전 오차의 정도를 파악하여 타원형 형태로 이동하는지에 대한 이유를 분석한다. 그리고 다음과 같이 센서값을 보정하여 디바이스가 제대로 동작하도록 한다.

- 해당 디바이스 API를 테스트한 결과를 기반으로 오차 범위를 줄이도록 알고리즘 작성
- 문제가 되는 API 외에 다른 API에서 얻어온 값을 추가적으로 이용하여 센서값을 보정하는 알고리즘 작성

자원 모니터링 기반의 다른 디바이스 대체: 배터리 등의 자원 부족으로 디바이스가 제대로 동작하지 않는 경우를 대비하여 앱 에이전트는 지속적으로 IoT 디바이스 자원을 모니터링하는 기능을 구현해야 한다. 예를 들어, DeviceResourceMonitor 클래스를 앱 에이전트(App_Agent)에 추가한다. 그리고 앱 에이전트가 수행하는 다른 기능과 동시에 수행될 필요가 있으므로 별도의 쓰레드에서 동작할 수 있도록 구현한다. Table 4는 DeviceResourceMonitor가 목표 IoT 애플리케이션이 사용하는 디바이스의 배터리 잔량을 모니터링하는 기능에 대한 알고리즘이다.

Table 4. Algorithm for Monitoring Resources of IoT Devices

```
input: activeDevicePool
01  Begin
02  connectedDevices =
        activeDevicePool.keySetforConnectedDevices();
03  Iterator iter = connectedDevices.iterator();
04
05  while (iter.hasNext()) {
06    connectedDevice = iter.next();
07    batteryLevel = sensingAdapter.getBatteryStatus
    (connectedDevice);
08    If (batteryLevel < thresholdValue) {
09      notifyLowBattery();
10      replaceDevice(connectedDevice);
11    }
12  }
13  End
```

2번째 줄에서 효율적으로 모니터링하기 위해 모니터링 대상을 현재 IoT 애플리케이션과 네트워크로 연결되어있는 디바이스로 한정짓는다. 그러므로 Active Device Pool에 있는 디바이스 목록 중 현재 IoT 애플리케이션에 의해 사용되고 있는 디바이스 목록을 반환한다. 7번째 줄에서 SensingAdapter에 정의된 표준 인터페이스를 통해 IoT 디바이스의 배터리 상태를 확인한다. 이때, 배터리 외의 자원을 모니터링해야 한다면, 그에 해당하는 오퍼레이션을 호출하면 된다. 8~11번째 줄에서는 배터리가 임계치(Threshold Value) 이하로 감소하여 자원이 부족하다고 판단되었을 경우에만 실행된다. 여기서 임계치는 각 디바이스마다 다르므로, 이 역시 디바이스 테스트를 통해 각 디바이스가 오작동하게 되는 자원의 최저 임계치를 파악해야 한다. 자원이 부족한 경우 먼저 사용자에게 디바이스의 자원 부족을 Alert 메시지 형태로 알려주어, 사용자가 직접 배터리를 교환할 수 있게 한다. 그리고 Active Device Pool에 등록된 디바이스 중 대체 가능한 다른 디바이스로 교체한다. 다른 디바이스로의 교체는 Active Device Pool에 대체 가능한 디바이스가 있을 경우에만 성공적으로 수행된다.

5. IoT 애플리케이션 개발을 통한 사례연구

본 논문에서 제시된 이슈들과 솔루션 검증을 위해서, 다양한 지상용 IoT 디바이스를 제어할 수 있는 애플리케이션인 Ground IoT Controller(GIC) 안드로이드 앱을 개발하였다.

5.1 IoT 디바이스 제어 애플리케이션 개발 요구사항

GIC 앱은 지상용 IoT 디바이스를 두 가지 모드로 제어한다. 자동 운행(Auto Driving) 모드는 디바이스를 랜덤하게 생성된 값의 크기에 따라 이동 방향과 거리를 스스로 결정하고 이동하며, 수동 운행(Manual Driving) 모드는 사용자가 Fig. 11과 같이 안드로이드 화면에 나타난 제어 패널을 보고 직접 디바이스를 이동시킨다.

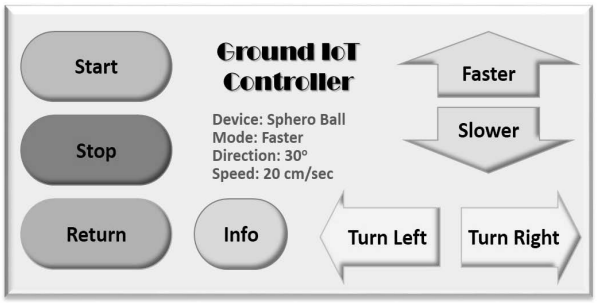

Fig. 11. Control Panel for Ground IoT Devices

제어 패널의 기능은 다음과 같다.

- Start 버튼: 기본 최소 속도로 디바이스를 이동시킨다.
- Stop 버튼: 디바이스를 현재 위치에서 멈추게 한다.
- Return 버튼: 디바이스를 출발 지점으로 귀환한다.
- Info 버튼: 디바이스의 세부 상태 정보를 나타낸다.
- Faster 버튼: 디바이스의 이동 속도를 초당 10cm만큼 높인다.
- Slower 버튼: 디바이스의 이동 속도를 초당 10cm만큼 낮춘다.
- Turn Left 버튼: 디바이스의 이동 방향을 왼쪽으로 5o만큼 바꾼다.
- Turn Right 버튼: 디바이스의 이동 방향을 오른쪽으로 5o만큼 바꾼다.

두 개 모드에 공통적으로 적용되는 기능은 다음과 같다.

- 디스플레이 패널에 디바이스의 현재 상태를 표시한다. 이때, IoT 디바이스 모델, 모드(Mode), 방향(Direction), 속도(Speed) 정보가 패널에 표시된다.
- 디바이스가 운행 중에 장애물을 만나서 충돌할 경우, 원래 출발 지점으로 자동으로 귀환한다.

5.2 적용 대상 IoT 디바이스

GIC 앱으로 제어할 지상용 IoT 디바이스로는 운행 방식이 전형적인 자동차와 다른 Sphero Ball과 Mindstorm nxt

[17]를 선정하였다. Sphero Ball은 2012년도 CES 전시회에 처음 소개된 이후 연구용 장비로 널리 사용되고 있으며, 레고(Lego)사가 개발한 Mindstome next는 다양한 물리적 형태를 구성할 수 있도록 하며 이동에 필요한 하드웨어 장치들을 포함하고 있다.

Sphero Ball: 당구공 크기의 이 디바이스는 ARM 프로세서 기반의 블루투스 네트워크를 지원하며, 자이로스코프(Gyroscope), 가속(Accelerometer)의 두 개 센서와 무선 충전 모터, LED 발광의 액추에이터들을 탑재하고 있다. 제어를 위한 API 세트는 Objective-C, 안드로이드, C# 등이 있는데, 본 개발에는 안드로이드 API를 사용하였다.

Mindstome nxt 2.0: 이 디바이스는 작은 벽돌 모양의 NXT Intelligent Brick AKA(Ciara)라는 허브 박스를 사용하는데, 이 장비는 네 개까지의 센서들로부터 입력을 받고, 세 개까지의 디바이스에 장착된 모터를 구동할 수 있다. 제어를 위한 API세트는 Java 언어이며, 블루투스 네트워크를 지원한다.

5.3 관찰된 이슈 및 적용된 솔루션

Fig. 12는 안드로이드 버전의 GIC 앱을 이용하여 IoT 디바이스 중 Sphero Ball을 제어하는 상황들을 보여준다.

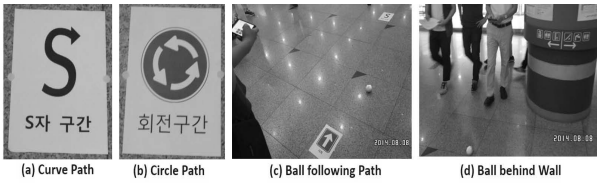

Fig. 12. Sphero Ball following various Paths

(a)는 IoT 디바이스를 S자 모양으로 이동시키는 상황이며, (b)는 일정한 반지름의 원을 회전하는 상황을 보여준다. (c)는 IoT 디바이스가 화살표 방향대로 이동하는 상황이며, (d)는 IoT 디바이스가 이동하다가 둥근 기둥을 사이에 두고 GIC 앱 스마트폰의 반대쪽에 위치해 있는 상황을 보여준다.

(a), (b), (c), (d)의 다양한 경로는 Turn Right, Turn Left, Faster, Slower 등 여러 기능의 조합으로 진행하였고, (d)의 경우만 기둥 뒤로 위치한 IoT 디바이스에 대해 가시성을 잃은 상황을 보여준다. 이 네 가지 상황에 맞춰 실행시키기 위한 GIC 앱을 설계하면서 관찰한 여러 이슈들은 다음과 같다.

1) IoT 디바이스 API 이질성 이슈

GIC 앱 개발에 IoT 디바이스 이질성은 Sphero Ball과 Mindstorm nxt 간에 지원되는 API와 이동 장치에서 발생하였다. Sphero Ball의 경우 안드로이드 등 여러 언어들의 API를 지원하며, Mindstorm nxt는 Java만을 지원한다. 공통이거나 호환이 되는 언어들이 없는 경우에는 API 이질성을 해결하기 위해 IPC(Inter-Process Communication) 등의 시스템 간 통신 솔루션을 해야 한다. GIC 앱의 경우는 안드로이드로 개발하였고, Sphero Ball과는 직접적인 연동이 가능했고, Mindstorm nxt는 앱 에이전트 안에서 Java 클래스를 정의하고 이 클래스가 해당 API를 호출하도록 하였다.

IoT 디바이스들 간의 이동 수단에 따른 이질성이 많이 발생한다. 본 개발에 사용된 두 개의 디바이스들도 공의 형태와 트랙 벨트를 사용하는 등 상당한 차이가 발생한다. 즉, Faster, Slower, Turn Right, Turn Left 등 앱에서 필요로 하는 기능성은 동일하지만, 이동 수단에 따른 메소드의 차이로 인해서 어댑터, 전략(Strategy), 템플릿 메소드(Template Method) 패턴을 적용하여 앱 에이전트를 설계하였다. 이 솔루션이 적용된 설계 모델은 Fig. 14의 클래스 다이어그램에서 확인할 수 있다.

2) 자체 이동성으로 인한 네트워크 연결성 이슈

Fig. 12에서 (d)의 경우, GIC 앱 스마트폰과 Sphero Ball 사이에 직선거리는 5미터 내외였으나, 반경 1미터 정도의 기둥이 가로막고 있어서, 블루투스 네트워크가 일시적으로 끊어지는 현상이 관찰되었다. 이 경우, Sphero Ball은 현재의 속도와 진행 방향대로 계속 진행하게 되므로, 원하는 경로를 따라가도록 통제하는 것이 어렵다. 나아가, 블루투스 네트워크를 자동으로 다시 페어링하는 데 몇 초간의 시간이 소요되었다.

이런 현상은 IoT 디바이스의 자체 이동성으로 인해, 네트워크 연결이 끊어지거나 불안정해지는 이슈이므로, 애플리케이션 설계 시 이런 오류를 자체적으로 해결하도록 Fault Tolerant한 설계를 하여야 한다. 본 개발에서는 ConnectionMonitor가 블루투스 네트워크 상태를 감지하고 자율적으로 재연결해주는 기능을 제공하도록 구현하였고, 이를 안드로이드 Service 클래스로 구현하여 배경 프로세스로 실행되도록 하였다. 그리고 ConnectionMonitor는 블루투스 연결을 잃은 IoT 디바이스를 Active Device Pool에서 삭제하도록 구현하였다. Sphero Ball을 제어하는 주 Activity 클래스는 이 Service 클래스와 직접 연동하지 않고 독립적으로 실행하도록 하여, 관점 분리의 원칙을 효과적으로 적용하였다.

3) 지상용 IoT 디바이스의 표면 상태 의존도 이슈

Fig. 12에서 (a)와 (b)의 경우 Sphero Ball이 회전을 하게 되는데, GIC 앱에서 정확한 회전 경로를 따라가기 위해서 현재 속도를 반영한 회전 각도를 계산한 후 Sphero Ball의 회전을 위해 void rotate(float heading) 메소드나 void drive(float heading, float velocity) 메소드를 실행시킨다. 이때 동일한 현재 속도와 동일한 희망 회전 각도로 이동을 시도할 때, 현재 위치의 지상 표면 상태에 따라서 실제 회전한 결과가 오차가 발생한 경우가 빈번했다. 이런 이슈가 발생한 원인은 Fig. 13과 같이 세 가지로 분석되었다.

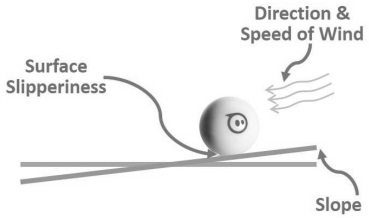

Fig. 13. Three Causes for Surface Dependency of Sphero Balls

- Sphero Ball이 있는 현재 위치의 표면 마찰력, 즉 미끄러운 정도(Surface Slipperiness): 마른 상태, 물에 젖은 상태, 기름 등 미끄러운 물질이 묻은 경우 등
- 현재 위치의 표면 기울기 상태(Slope): 진행하려는 방향에 비해서 표면의 기울기가 어떤 방향으로 몇 도 정도 기울어져 있는지의 영향
- 현재 위치의 바람의 방향과 속도(Wind Direction and Speed): 진행하려는 방향에 비해서 바람의 세기와 방향에 따른 영향

이러한 표면 상태 의존도 이슈에 대해서, 본 개발에서는 다음 두 가지의 소프트웨어적인 솔루션을 적용하였다.

- 정적 조정(Static Calibration) 장치: GIC 앱이 실행을 시작할 때, 약 10초간에 걸쳐 이동, 회전, 가속, 감속을 통한 표면 상태 및 바람세기 방향 등의 감지 작업을 한다. 감지된 데이터를 기반으로 위에서 설명한 세 가지 원인에 대한 이동 조정 계수(Adjustment Factors)를 설정한다.
- 동적 조정(Dynamic Calibration) 장치: 정적 조정 작업으로도 감지할 수 없는 표면 상태에 대해서 동적으로 표면 및 바람 상태를 감지하고 이를 통해 이동 조정 계수를 수정한다. Sphero Ball이 이동 지역이 넓거나 실외에서 이동할 경우에는 동적 조정이 더 자주 실행된다.

적용한 소프트웨어적 솔루션은 물리적 및 환경적인 의존도 이슈를 완벽하게 해결할 수는 없다. 이는 이러한 이슈들이 IoT 디바이스의 기계적인 기능 및 능력에 상당한 영향을 받기 때문이다. 그렇지만, 제한된 영역 내에서 IoT 디바이스 오작동 오차율을 줄일 수 있었다.

5.4 솔루션을 적용한 설계 모델

GIC 프로그램은 안드로이드 언어로 개발하였으며, 설계는 UML의 유즈케이스 다이어그램, 클래스 다이어그램, 배치(Deployment) 다이어그램을 사용하였다. Fig. 14는 앞서 언급한 솔루션들을 적용하여 설계한 앱 에이전트에 해당하는 클래스 다이어그램의 일부를 보여준다.

전략 패턴을 적용하여 Fig. 14의 왼쪽 상단에 있는 GIC Client 클래스는 우측의 Ground Device라는 추상클래스만

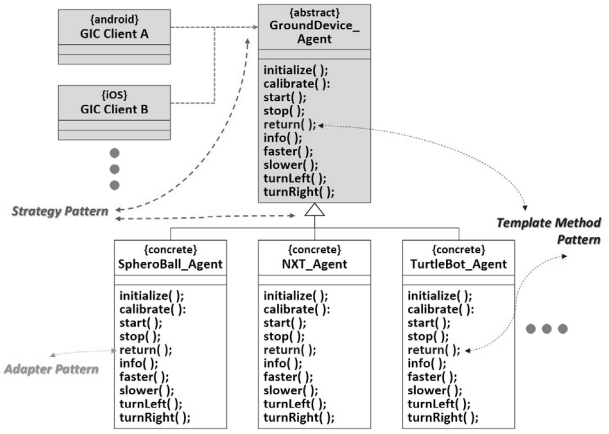

Fig. 14. Class Diagram applied with Strategy, Template Method, Adapter Patterns

사용하여 구현하였다. 실행 시에는 이 추상클래스 대신 디바이스별로 정의된 하위 클래스를 선택하고 그 객체 인스턴스(Instance)로 대치하여 사용한다. 그 결과 GIC Client는 수정 없이 다양한 지상용 IoT 디바이스들을 범용적으로 제어할 수 있다.

Fig. 14에서 Ground Device_Agent는 추상클래스로서 지상용 IoT 디바이스들의 공통 기능성을 제공하는 추상 메소드(Abstract Method)와 인터페이스를 정의하고 있다. 하위에 있는 Concrete 클래스는 지상용 IoT 디바이스를 제어하는 앱 에이전트를 나타내며, 상위 클래스에서 정의된 메소드를 그 디바이스의 규격과 기능에 맞도록 구현하고 있다. 이 구현에는 다음 세 가지 설계 방식을 적용하였다.

- 인터페이스에 대한 메소드 구현: 상위 클래스에서 순수 가상함수, 즉 메소드의 인터페이스만 정의된 경우, 각 디바이스에 맞도록 메소드 전체를 구현한다. 상위 클래스에 정의된 메소드는 Fig. 6의 프로세스를 통해 도출된 표준화된 형태의 인터페이스이다. 이렇게 함으로써, 다양한 종류의 IoT 디바이스를 표준화된 인터페이스를 통해 제어함으로써 GIC_Client는 여러 디바이스 간의 API 이질성으로부터 영향을 덜 받게 되었다.
- 메소드 재정의(Overriding)를 통한 구현: 상위 클래스에서 메소드에 대한 기본 기능이 구현되어 있으나, 각 디바이스별로 정의된 하위 클래스에서 전체나 일부 기능을 재정의한다.
- 템플릿 메소드 패턴에 따른 구현: 상위 클래스는 메소드의 전체 알고리즘을 정의하여 고정한 후, 알고리즘의 일부 스텝들만 하위 클래스에서 대리 실행(Delegation)을 요청하도록 구현한다. 예를 들면, Fig. 14에서 return() 메소드는 출발 위치로 귀환하는 알고리즘은 동일한데, 그중에서 일부 스텝은 디바이스별로 상이하다. 즉, 디바이스별로 현재 위치에서 출발 위치로 귀환 경로 계획(Return Route Planning) 스텝이 다르다. Sphero Ball의 경우 출발 지점에서부터 현재 위치까지

의 이동 명령들을 반대의 순서로 적용하면서 이동 오차 조정 작업을 병행해야 한다. Mindstorm nxt의 경우 설치되어있는 GPS 센서 등의 설치 여부에 따라 귀환 경로 계획이 가변적이며 트랙 벨트(Track Belt)에 이동함으로써 이동에 따른 오차는 고려하지 않아도 된다.

- 어댑터 패턴에 따른 구현: SpheroBall_Agent, NXT_Agent, Turtlebot_Agent의 모든 메소드는 템플릿 메소드 패턴의 적용으로 표준화된 메소드이다. 그러나 각 메소드 내부에서는 실제 디바이스 API를 호출함으로써, GIC_Client는 디바이스 종류별로 다른 API에 독립적으로 설계되었다.

이 밖에 Fig. 9와 같은 설계 모델로 ConnectionMonitor, Active Device Pool 등을 이용하여 네트워크 연결성 문제를 해결하였다. 그리고 Fig. 14의 SpheroBall_Agent 클래스의 움직임 관련 메소드인 start(), faster(), slower() 등의 메소드 내에서는 표면 상태를 고려하여 움직임의 오차율을 줄일 수 있도록 알고리즘을 구현하였다.

6. 결 론

IoT 환경에서 애플리케이션은 네트워크에 연결된 여러 디바이스들을 이용하여 사용자에게 유용한 정보와 편의를 제공할 수 있다. 소프트웨어 기능만으로 구성된 순수 소프트웨어와는 달리 IoT 애플리케이션은 빠르게 보급되고 있는 IoT 디바이스와 항상 같이 상호작용한다. 그리고 임베디드 시스템, 모바일 애플리케이션, 퍼베이시브 시스템과도 구별되는 특징을 가지고 있다. 그러므로 IoT 애플리케이션은 현재까지 개발된 애플리케이션과는 다른 특성을 가지고 있고, 이로 인해 소프트웨어 시스템 개발에는 나타나지 않는 비전형적인 기술적 이슈들이 발생한다.

본 논문에서는 기술적 복잡도도 높고 비전형적인 이슈들을 가지는 IoT 애플리케이션을 효과적으로 개발하기 위하여, IoT 애플리케이션 개발 시 발생하는 기술적 이슈를 IoT 디바이스의 특징인 이질성, 자체 이동성, 물리적 제약성의 세 가지 관점을 고려하여 도출하였다. 그리고 각 이슈들을 해결할 수 있는 기법들을 제시하였다. 제안된 솔루션은 여러 IoT 애플리케이션 개발에 범용적으로 사용되기 위하여, 디바이스 관련 문제만 별도로 담당하는 앱 에이전트 설계에 적용된다. 마지막으로, 제시된 이슈들과 기법들을 검증하기 위하여, 언급한 이슈를 가진 IoT 애플리케이션 개발에 제시된 기법들을 적용한 결과를 보여주었다. 본 논문을 통해 도출된 이슈와 제안된 솔루션을 이용하여 IoT 애플리케이션을 보다 쉽고 효과적으로 개발할 수 있고, 나아가 제안된 솔루션은 IoT 프레임워크 설계의 기반으로 활용될 것으로 기대한다.

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] International Telecommunication Union, ITU Internet Reports 2005., The Internet of Things, Nov., 2005., Available: http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf (downloaded 24, Dec., 2013.)

[3] D. Miorandi, S. Sicari, F.D. Pellegrini, and I. Chlamtac, “Internet of Things: Vision, Applications, and Research Challenges”, *Ad Hoc Networks*, Vol.10, No.7, pp.1497-1516, Sep., 2012.

[4] Y.K. Chen, “Challenges and Opportunities of Internet of Things”, *In Proceedings of the 17th Asia and South Pacitic Design Automation Conference(ASP-DAC 2012)*, pp.383-388, Jan., 2012.

[5] R. Khan, S.U. Khan, R. Zaheer, and S. Khan, “Future Internet: The Internet of Things Architecture, Possible Applications, and Key Challenges”, *In Proceedings of the 2012 10th International Conference on Frontiers of Information Technology(FIT 2012)*, pp.257-260, Dec., 2012.

[6] M.A. Chaqfeh, N. Mohamed, “Challenges in Middleware Solutions for the Internet of Things”, *In Proceedings of 2012 International Conference on Collaboration Technologies and System(CTS 2012)*, pp.21-26, May, 2012.

[7] P. Patel, A. Pathak, T. Teixeira, and V. Issarny, “Towards Application Development for the Internet of Things”, *In Proceedings of the 8th Middleware Doctoral Symposium (MDS 2011)*, Article No.5, Dec., 2011.

[8] D. Singh, G. Tripathi, and A.J. Jara, “A Survey of Internet-of-Things: Future Vision, Architecture, Challenges, and Services”, *In Proceedings of 2014 IEEE Forum on Internet of Things (WF-IoT 2014)*, pp.287-292, March, 2014.

[9] R. Xu, L. Yang, and S.H. Yang, “Architecture Design of Internet of Things in Logistics Management for Emergency Response”, *In Proceedings of 2013 IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things (iThings) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pp. 395-402, Aug., 2013.

[10] Parrpt AR Drone 2.0 [Internet], http://ardrone2.parrot.com/

[11] Sphero [Internet], http://www.gosphero.com/sphero-2-0/

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley, 1995.

[13] D.Y. Kim, H.J. La, and S.D. Kim, “A Framework for Effectively Managing Heterogeneity of IoT Devices”, *Journal of KIISE: Software and Applications*, Vol.41, No.5, pp.353-366, May, 2014.

[14] C.W. Park, S.H. Yang, H.J. La, and S.D. Kim, “A Method to Dynamically Connecting IoT Devices via Active Device Pool”, *In Proceedings of the 16th Korea Conference on Software Engineering(KCSE 2014)*, Vol.16, No.1, pp.165-166, February, 2014.

[15] H.J. La, C.W. Park, and S.D. Kim, “A Framework for Effectively Managing Dynamism of IoT Devices”, *Journal of KIISE: Software and Applications*, Vol.41, No.8, pp. 545-556, August, 2014.
[16] B.M. Albaker, N.A. Rahim, “Unmanned Aircraft Collision Detection and Resolution: Concept and Survey”, in *Proceedings of the 5th IEEE Conference on Industrial Electronics and Applications(ICIEA 2010)*, pp.248-253, June, 2010.
[17] Lego Mindstorm [Internet], http://mindstorms.lego.com

라 현 정

e-mail : hjla80@gmail.com
2003년 경희대학교 전자정보학부(학사)
2006년 숭실대학교 컴퓨터학과(공학석사)
2011년 숭실대학교 컴퓨터학과(공학박사)
2011년~2013년 숭실대학교 모바일 서비스 소프트웨어공학센터 연구교수
2013년~현 재 (주) 스마티랩 대표
관심분야 : 소프트웨어 아키텍처(Software Architecture), 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/1991년 The University of Iowa 전산학(석사/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)