Journal of The Korea Society of Computer and Information
Vol. 24 No. 3, pp. 77-86, March 2019
www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.03.077

UniPy: A Unified Programming Language for MGC-based IoT Systems

Gayoung Kim*, Kwanghoon Choi*, Byeong-Mo Chang**

Abstract

The advent of Internet of Things (IoT) makes common nowadays computing environments involving programming not a single computer but several heterogeneous distributed computers together. Developing programs separately, one for each computer, increases programmer burden and testing all the programs become more complex. To address the challenge, this paper proposes an RPC-based unified programming language, UniPy, for development of MGC (eMbedded, Gateway, and Cloud) applications in IoT systems configured with popular computers such as Arduino, Raspberry Pi, and Web-based DB server. UniPy offers programmers a view of classes as locations and a very simple form of remote procedure call mechanism. Our UniPy compiler automatically splits a UniPy program into small pieces of the program at different locations supporting the necessary RPC mechanism. An advantage of UniPy programs is to permit programmers to write local codes the same as for a single computer requiring no extra knowledge due to having unified programming models, which is very different from the existing research works such as Fabryq and Ravel. Also, the structure of UniPy programs allows programmers to test them by directly executing them before splitting, which is a feature that has never been emphasized yet.

▸Keyword: Unified Programming Environment, Internet of Things, Distributed System

I. Introduction

사물인터넷은 실세계의 다양한 장치들과 컴퓨터를 통합하여 새로운 서비스를 제공하는 소프트웨어를 만들 수 있는 기반이다. 단일 컴퓨터에서 동작하는 기존의 소프트웨어 개발 방식과 달리 서로 다른 다양한 형태의 분산 컴퓨터들을 조화롭게 다루도록 사물인터넷 소프트웨어를 만들어야 한다.

사물인터넷의 사례인 스마트홈(Smart Home)은 일반적으로 4가지 중요 요소, 임베디드 장치, 게이트웨이, 클라우드 서버, 모바일 장치가 서로 연결된 시스템으로 구성되어 있다. 임베디드 장치는 센서 기능이 탑재되어 있어 예를 들어 온도 측정, 문의 개폐 여부 등을 감지한다. 게이트웨이는 임베디드 장치에서 감지한 센서 정보를 인터넷을 통해 클라우드 서버에 전달하는 역할을 담당한다. 클라우드 서버는 전달받은 정보를 데이터베이스에 축적한다. 사용자는 모바일 장치를 통해 클라우드 서버에 접속하여 집의 내부 상태를 확인하고 또한 게이트웨이를 통해서 임베디드 장치에 명령을 내려 방의 최소 온도를 설정하거나 문을 열고 잠근다.

이와 같은 분산 시스템에서 동작하는 사물인터넷 소프트웨어를 개발할 때 3가지 문제점이 있다. 첫째, 사물인터넷 시스템 장치들 간의 네트워크를 어떻게 구성할지, 둘째, 다양한 장치들에서 동작할 소프트웨어를 어떻게 개발할지, 셋째, 개발한 소프트웨어를 어떻게 배치하고 업데이트하며 나중에 유지 보수하는 문제가 있다.

이 논문에서는 사물인터넷 소프트웨어 개발의 두 번째 문제를 해결하고자 한다. 이 문제를 해결하기 위한 기존 연구에 두 가지 방법이 있다. 첫 번째는 서로 다른 장치들을 구분하지 않고 통신할

• First Author: Gayoung Kim, Corresponding Author: Kwanghoon Choi
*Gayoung Kim (kirayu15@gmail.com), Dept. Electronics and Computer Engineering, Chonnam National University
*Kwanghoon Choi (kwanghoon.choi@jnu.ac.kr), Dept. Electronics and Computer Engineering, Chonnam National University
**Byeong-Mo Chang (chang@sookmyung.ac.kr), Dept. of Computer Science, Sookmyung Women's University
• Received: 2018. 12. 27, Revised: 2019. 03. 04, Accepted: 2019. 03. 04.
• This study has been supported by the research grant (No.2018-0916) from Chonnam National University.

수 있는 표준 API(Application Programming Interface)나 프로토콜을 제공하는 프레임워크(framework)를 제안한 방법이다. 예를 들어, 아두이노 기반 사물인터넷 시스템 [1], PatRICIA [2], 분산 Node-Red [3], 사물인터넷 게이트웨이를 자유롭게 프로그래밍 할 수 있는 프레임워크 [4]와 같은 연구가 있다.

이러한 접근 방법은 표준화를 통해 여러 장치들의 다른 점을 동일하게 추상화한 인터페이스를 제공하는 장점이 있지만 여러 다른 장치들을 위한 인터페이스 표준화 과정이 쉽지 않다. 또한 각 장치에서 동작하는 프로그램을 각각 개발하고 테스트해야 하므로 단일 컴퓨터 상의 프로그램을 개발하는 것과 비교하면 더 많은 노력이 필요하다. 여러 장치에 분산된 프로그램들을 모아 테스트하는 작업은 더욱 복잡하다.

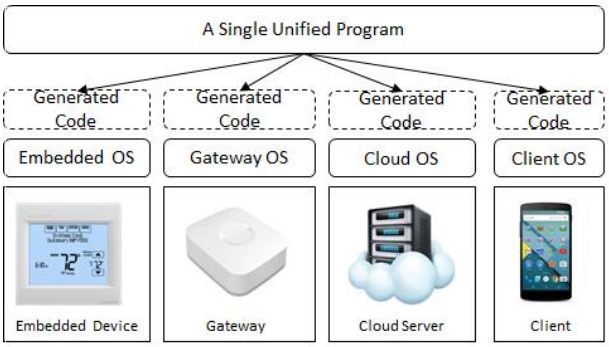

Fig. 1. Approach of Integrated Language for IoT Program

두 번째는 통합 프로그래밍언어를 사용하여 프로그래머는 하나의 통합 프로그램으로 작성하면 자동으로 각 장치를 위한 프로그램으로 분리컴파일 하는 방법이다. Fig. 1에서 이 방법을 보여준다. 통합 프로그래밍언어 사례로 Links [5]와 Lambda5 [6]는 클라이언트와 서버로 구성된 웹 시스템에서 동작하는 소프트웨어를 작성할 목적으로 설계되었다. Pleiades [7]는 무선 센서 네트워크를 위한 통합 프로그래밍언어이며, Fabryq [8]와 Ravel [9]은 스마트 홈 소프트웨어를 개발할 목적으로 설계된 통합 프로그래밍언어이다.

Fabryq [8]와 Ravel [9] 연구에서 각 언어를 사용하여 개발한 프로젝트를 소개하였다. Fabryq는 BLE (Bluetooth Low Energy)에 기반한 프로토콜 프록시 프로그래밍 모델을 제공하여 프로그래머는 마치 임베디드 장치가 직접 연결되어 있는 것처럼 간주하고 통합 프로그램을 작성할 수 있다. Ravel [9]는 3계층 사물인터넷 시스템에 대한 분산 모델-뷰-컨트롤러 모델을 제공한다.

두 번째 방법은 첫 번째 방법의 단점을 해결할 수 있는 장점을 지닌 반면에 각 연구에서 설계한 통합 프로그래밍언어로 통합 프로그램을 작성할 때 지나치게 이 언어에 의존적인 문제가 발생한다. 그 결과 새로운 장치를 사물인터넷 시스템에 도입하거나 새로운 통신 프로토콜을 사용할 때 기존 프로그램과 완벽하게 호환되며 확장하기 어렵다.

두 번째 방법에서 제안한 통합 프로그래밍언어를 사용하되 각 장치를 개별적으로 프로그래밍 하는 기존 방식을 그대로 활용할 수 있다면 가장 이상적일 것이다. 즉, 각 계층의 장치에서 동작하는 프로그램을 작성하는 기존 방식은 최대한 유지하면서 다른 계층의 장치에서 제공하는 함수를 동일한 계층의 장치에서 제공하는 함수를 호출하는 것과 동일한 방식으로 호출하면서 통합 프로그램을 작성하는 방법을 고려해보자. 이 방법은 프로그래머가 새로운 사물인터넷 프로그래밍 모델을 배우는 노력을 최소화하고 서로 다른 계층의 함수를 호출할 때 지원해야할 통신 방법을 컴파일러에서 제공함으로써 확장 가능한 통합 프로그래밍 모델이다.

본 논문에서 수행한 연구의 목적은 기존의 장치별 프로그래밍 모델을 최대한 유지하되 여러 장치들을 마치 하나의 장치와 같이 프로그래밍 할 수 있도록 최소한의 인터페이스를 추가한 통합 프로그래밍 모델을 설계하는 것이다.

이 목적을 달성하기 위하여 임베디드 장치, 게이트웨이, 클라우드 서버, 모바일로 구성된 사물인터넷 시스템에서 RPC 계산법(RPC calculus)을 기초로 설계한 통합 프로그래밍언어를 제안한다. RPC 계산법은 기존 람다 계산법(Lambda calculus)을 각 함수의 실행할 위치를 지정할 수 있다는 특징으로 확장한 것이다. RPC 계산법의 함수 실행 위치 지정을 활용하여 다른 계층의 장치에서 실행할 함수를 마치 동일 계층의 장치에서 실행할 함수처럼 호출하도록 프로그램을 작성할 수 있다. 이 연구에서 구현한 컴파일러를 활용하여 이렇게 작성한 통합 프로그램을 자동으로 4가지 장치에서 동작하는 프로그램으로 분리한다. 다른 계층의 함수를 호출하는 코드는 컴파일러에 의해 자동으로 적절한 통신 방법을 사용한 RPC 코드로 대체된다.

제안한 통합 프로그래밍언어 설계 방법에 대한 타당성을 확인하기 위하여 파이썬 언어를 확장하여 제안한 방법에 따라 통합 프로그래밍언어 UniPy(Unified programming language using Python)를 설계하고 컴파일러를 구현하였다. 평가에 사용할 사물인터넷 시스템은 대중성이 높은 다음의 장치들로 구성하였다. 임베디드 장치로 다양한 센서를 갖춘 아두이노를 선택하였고, 게이트웨이로 라즈베리 파이를 사용하였으며 클라우드로 데이터베이스 관리 시스템을 갖춘 데스크톱 컴퓨터를 갖추었다.

본 논문은 다음과 같이 구성한다. 2장에서 홈 모니터링 시스템의 예를 통해 연구 동기를 제시하고 통합 프로그래밍언어 UniPy의 설계 및 구현을 설명한다. 3장에서 이 논문에서 제안한 사물인터넷 통합프로그램 개발 방법의 장단점을 논의한다. 4장에서 기존 연구와 비교하고, 5장에서 결론 및 향후연구를 설명한다.

II. A Unified Programming Based on RPC Calculus for IoT Applications

사물인터넷 사례로 홈 모니터링 시스템을 먼저 소개하고 RPC 계산법에 기초한 통합 프로그래밍언어 설계를 제시한 다음 각 장치에서 동작하는 프로그램으로 분리하여 컴파일 하는 방법을 설명한다.

1. Motivation: Home Monitoring System

Fig. 2는 이 논문에서 고려하는 홈 모니터링 시스템을 보여준다. 이 시스템은 다양한 센서를 갖춘 아두이노 기반 임베디드 장치, 라즈베리 파이 기반 게이트웨이, 클라우드로 구성되어 있다. PC 데스크톱 기반 클라우드 서버에 MySQL 데이터베이스 관리 시스템을 갖추었고, 사용자는 자바 프로그램을 통해서 클라우드 서버의 정보를 읽거나 게이트웨이에 명령을 내려 임베디드 장치를 제어한다.

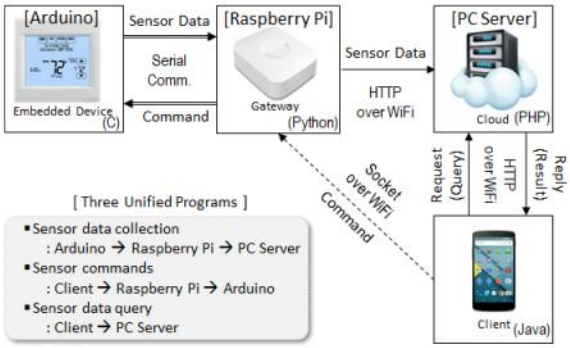

Fig. 2. Example of Home Monitoring System

홈 모니터링 시스템에 다음과 같은 센서를 사용한다. 2개의 마그네틱 센서를 두어 출입문과 창문의 개폐를 감지하고, 1개의 모션 센서로 문 앞에서 움직이는 물체가 있는지 확인하며, 문에 사용자가 직접 제어할 수 있는 카메라가 장착되어 있어 사진이나 라이브 영상을 실시간으로 보낸다. 이외에도 4x4 행렬 키패드와 16x2 엘씨디(LCD) 패널을 문에 두어 문을 열기 위해 비밀번호를 입력하고 메시지를 출력한다. 엘이디(LED)와 피에조(Piezo) 사운드 센서를 두어 침입자로 의심되는 경우 경고 빛과 소리를 낸다(Fig. 2).

홈 모니터링 사물인터넷 시스템의 각 구성 요소들은 서로 다른 통신 프로토콜을 사용한다. 아두이노 기반 임베디드 장치와 라즈베리 파이 기반 게이트웨이는 물리적으로 연결하여 시리얼 통신을 사용한다. 이 게이트웨이와 PC기반 클라우드 서버는 HTTP 프로토콜을 통해 웹 기반 인터페이스로 통신한다. 사용자는 게이트웨이와 마찬가지로 HTTP 기반 웹 인터페이스를 통해 축적된 센서 정보를 얻는다. 사용자가 직접 게이트웨이를 연결하는 방법도 가능한데 이때 TCP/IP 기반 소켓(Socket) 프로토콜을 사용한다. HTTP 프로토콜과 소켓 프로토콜은 모두 와이파이(Wi-Fi) 상에서 지원한다.

이와 같은 홈 모니터링 사물인터넷 시스템에서 각 장치별로 각각 프로그램을 개발하는 전통적인 프로그램 개발 방식은 적합하지 않다. 왜냐하면 이 시스템에서 의도한 서비스는 여러 장치들이 유기적으로 연결되어야만 완성할 수 있기 때문이다. 예를 들어, 홈 모니터링 시스템에서 각종 센서 데이터를 모으는 전형적인 사용 사례(Usecase)를 살펴보자. 문에 장착된 아두이노 기반 임베디드 장치의 마그네틱 센서에서 문을 열고 닫을 때마다 새로운 센서 데이터를 만든다. 시리얼 통신으로 연결된 라즈베리 파이 기반 게이트웨이로 이 데이터를 전달하면 무선 와이파이로 연결된 PC 클라우드 서버에 HTTP 웹 인터페이스를 통해 전달하고 궁극적으로 데이터베이스에 저장한다. 이렇게 한 가지 사용 사례에 3가지 장치가 관여하고 있다. 그러므로 이 사용 사례를 지원하기 위해서 각 장치별로 하나씩 3가지 프로그램을 따로 만들면 개발 노력도 더 필요하고 모두 모아 테스트하는 것도 복잡할 것이다.

전통적인 프로그램 개발 방법에 대한 대안으로 이 3가지 장치를 마치 하나의 연결된 장치로 가정하고 통합된 프로그램을 작성하는 것이다. Fabryq [8]와 Ravel [9]은 다른 계층을 통합된 뷰를 가지고 통합 프로그램을 작성할 수 있도록 별도의 추상화된 계층을 도입하였다. Fabryq에서는 BLE 프로토콜 프록시 모델을 도입하여 각 계층이 모두 BLE 프로토콜을 통해 통신하는 것처럼 추상화하였다. Ravel에서는 분산 MVC (Model-View-Controller) 모델로 각 계층별로 미리 정의된 템플릿을 제공한다. 이 템플릿을 활용하여 통합 프로그램을 작성하면 각 계층의 프로그램으로 자동 변환된다. 두 가지 통합 프로그래밍 모델은 전통적인 방법에 비해 장점을 갖지만, 각 모델에 맞게 추상화된 소프트웨어 계층이 다양한 장치들에 대해 항상 준비되어 있는 것은 아니다.

이상적으로 통합 프로그램을 작성하더라도 각 장치 개별로 프로그램을 작성하는 방식을 그대로 활용할 수 있다면 Fabryq나 Ravel과 같이 미리 추상화된 소프트웨어 계층을 준비하지 않아도 된다. 다만 두 계층 간의 데이터를 주고받는 통신 코드를 같은 계층에서 사용하는 함수를 호출하는 것처럼 간단하게 작성할 수 있어야 한다. 이러한 접근 방식은 장치별 프로그래밍 모델을 허용하면서도 여러 장치 간 연동을 마치 하나의 프로그램을 작성하듯이 작성할 수 있다. 계층 간 함수 호출은 컴파일러에 의해 자동으로 복잡한 원격 함수 호출 코드로 변환하도록 지원해야 한다.

2. UniPy: A Design and Implementation

이제 원격 함수 호출(RPC) 기반 통합 프로그래밍 모델을 제안한다. 먼저 RPC 계산법에 관한 이론을 살펴보고 홈 모니터링 사물인터넷 시스템에 어떻게 적용할지 설명한다.

2.1 The RPC Calculus

RPC 계산법, λrpc는 Cooper와 Wadler가 함수형 언어 계산 모델인 람다 계산법을 함수의 실행할 위치를 지정하는 특징을 추가하여 확장한 것이다[10]. 함수에 해당하는 람다 추상식(lambda abstraction)에 위치를 지정하는 주석을 붙인다. 이 주석의 위치에서만 이 함수를 실행할 수 있도록 실행 방법이 정의되어 있다. 이 계산법은 다계층 웹 프로그래밍 언어 Links를 설계할 때 이론적 기초로 사용하였다. 이 논문에서 관심 있는 사물인터넷 시스템의 목적에 따라 위치를 지정하는 주석으로 임베디드 장치를 지정하는 m, 게이트웨이를 지정하는 g, 클라우드 서버를 지정하는 c를 도입한다. 필요한 경우 더 많은 위치

를 지정하는 주석을 추가로 도입할 수 있다. Fig. 3에서 이러한 위치 주석을 갖는 λrpc의 구문(syntax)과 이 구문으로 작성된 프로그램의 실행 방법(semantics)을 보여준다.

Syntax

a, b	$::=$	$\mathbf{m} \mid \mathbf{g} \mid \mathbf{c}$	locations
L, M, N	$::=$	$L\ M \mid V$	terms
V, W	$::=$	$x \mid \lambda^a x.N$	values

Semantics

$$V \Downarrow_a V \quad \text{(Value)}$$

$$\frac{L \Downarrow_a \lambda^b x.N \quad M \Downarrow_a W \quad N\{W/x\} \Downarrow_b V}{L\ M \Downarrow_a V} \quad \text{(Beta)}$$

Fig. 3. The RPC Calculus for Three-tier IoT Systems

람다 계산법의 구문과 같이 함수 호출(lambda application) L M, 변수 x, 함수 정의(lambda abstraction) λ^ax.N의 구문으로 프로그램을 작성한다. 함수 정의에 위치를 나타내는 주석 a를 지정하는 점만 다르다. 이렇게 작성한 프로그램을 실행하는 방법(semantics)은 (Value)와 (Beta) 규칙으로 정의한다. 프로그램 M을 위치 a에서 실행한 결과 V를 내는 계산 과정(evaluation judgments)을 $M \Downarrow_a V$로 나타낸다. 이 계산 과정을 조합하여 두 가지 규칙을 정의한다. 특히 (Beta) 규칙이 중요하다. 함수 호출 L M을 실행하면 먼저 L을 동일한 위치 a에서 계산하여 b에서 실행할 함수 정의를 얻고, M을 위치 a에서 계산하여 W를 얻은 다음 위치 b에서 실 인자 W를 형식인자 x에 전달하고 (N{W/x}) 함수를 실행한다(Fig. 3).

λrpc에서는 프로그램 실행을 원하는 지정한 위치에서 시작한다. 예를 들어, 센서 데이터를 수집하는 사용사례에 대한 프로그램은 임베디드 장치에서 실행을 시작한다. 이 λrpc 프로그램은 다음과 같이 작성할 수 있다.

```
(λ^m gtw. gtw (read_sensor ()))
   [ (λ^g cld. λ^g sensor_value. cld sensor_value)
        (λ^c v. store_into_db v) ]
```

이것은 임베디드 장치, 게이트웨이, 클라우드 서버에서 실행할 3가지 함수를 마치 하나의 장치를 프로그래밍 하는 것처럼 작성한 통합 프로그램 예제이다.

이 통합 프로그램에서 임베디드 장치에서 제공하는 read_sensor() 함수로 센서 데이터를 읽어 변수 gtw가 가리키는 게이트웨이 함수를 호출하여 전달한다. 변수 gtw는 게이트웨이에서 실행할 함수 (λ^g cld. ...)가 된다. 그 다음 게이트웨이는 변수 cld가 가리키는 클라우드 함수를 호출하여 센서 데이터를 계속 전달한다. 변수 cld는 (λ^c v. ...)가 되는데 이 함수는 클라우드 서버에서 제공하는 stored_into_db 함수를 호출하여 데이터베이스에 이 센서 데이터를 저장한다.

이 통합 프로그램의 예제에서 볼 수 있듯이 각 장치에서 사용가능한 함수를 그대로 사용하되 서로 다른 위치에서 제공하는 함수를 호출하는 방법은 같은 위치에서 제공하는 함수를 호출하는 방법과 거의 비슷하다. 따라서 Fabryq이나 Ravel에서 제안한 바와 다르게 통합 프로그래밍 모델을 위한 별도로 새로 추가한 추상화된 인터페이스가 거의 필요하지 않음을 알 수 있다.

본 논문에서 RPC 계산법을 채택하여 제안하고자 하는 통합 프로그래밍언어 설계의 기초로 삼고 이 언어로 작성한 통합 프로그램을 각 장치별 프로그램으로 분리하여 코드를 생성하기 위해 이 계산법의 컴파일 방법[10]을 응용한다. RPC 계산법은 통합 프로그래밍 모델을 위한 다른 계산법들 [6,7,11]과 비교하여 매우 간결한 특징이 장점이다. 하지만 RPC 계산법은 클라이언트-서버 모델 기반 통합 웹 프로그래밍언어의 기초로 사용되어 Links [5]를 설계하는데 활용되었다. 따라서 RPC 계산법이 다양한 형태의 장치들을 연결한 사물인터넷 시스템에도 적용될 수 있을지 분명하지 않다. 이 논문은 사물인터넷 통합 프로그래밍언어 설계의 기초로 RPC 계산법을 활용하고자 한다. 사물인터넷 시스템은 웹을 위한 클라이언트-서버 모델과 비교하면 더 복잡한 구조를 가지고 있다. 이 논문에서 가정하는 사물인터넷 장치는 RPC 계산법의 고차원 함수(higher-order function)를 사용하지 않기 때문에, 이 특징은 제한한다.

사물인터넷 시스템을 위한 통합 프로그래밍언어에 RPC 계산법을 적용하여 구현하기 위해 3가지 주요 설계 문제를 해결해야 한다.

- 다양한 장치들에 대한 통합 프로그램을 작성하는데 사용할 통합 프로그래밍언어 설계
- 시리얼통신, HTTP, 소켓과 같은 통신 프로토콜 상에서 원격 함수 호출을 지원하는 방법
- 임베디드 장치 프로그램의 이벤트 루프와 같이 각 장치에 내재된 런타임 시스템을 고려한 구현

다음 절에서 이 설계 문제들을 해결하는 방법에 대해 하나씩 설명할 예정이다.

2.2 UniPy: A Unified Programming Language for IoT Applications

UniPy는 파이썬을 확장하여 만든 사물인터넷을 위한 통합 프로그래밍 언어다. 파이썬을 선택한 이유는 배우기 쉬운 가장 잘 알려진 프로그래밍언어 중 하나이기 때문이다.

UniPy 프로그램에서 클래스를 통해서 실행할 위치를 지정한다. 예를 들어, Fig. 4는 홈 모니터링 시스템에 대한 UniPy 프로그램의 클래스 다이어그램의 예를 보여준다.

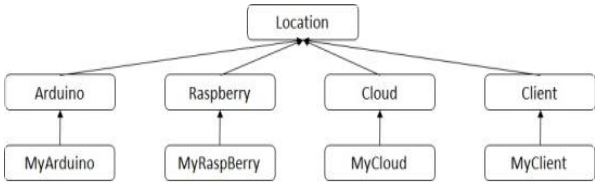

Fig. 4. UniPy Program Class Representing Location

UniPy 프로그램의 아두이노 기반 임베디드 장치에서 동작하는 코드를 작성하기 위해 Arduino 클래스를 상속받아 MyArduino 클래스를 만든다. 이런 방식으로 상속받은 부모 클래스에 따라 자식 클래스의 코드가 실행될 위치를 결정하도록 의미를 설계하였다. 사물인터넷 시스템에 둘 이상의 아두이노 기반 임베디드 장치를 사용하는 경우에는 동일한 부모 클래스를 상속받은 새로운 자식 클래스를 만들어 코드를 작성하면 된다. 라즈베리 파이 기반 게이트웨이, 클라우드, 클라이언트 장치에서 실행할 코드들도 동일한 방식으로 실행 위치를 지정한다(Fig. 4).

Fig. 5는 홈 모니터링 시스템에서 문에 장착된 마그네틱 센서로부터 문의 개폐 상태에 대한 센서 데이터를 모으는 UniPy 프로그램의 구체적인 예를 보여준다. 이 시나리오는 3가지 장치가 관여하여 동작하고 있기 때문에 3개의 클래스로 구성되어 있다.

각 장치의 코드는 해당 위치를 나타내는 부모 클래스를 상속받은 자식 클래스의 정적 메서드로 작성한다. 프로그래머가 작성한 각 장치 클래스 D는 3가지 종류의 함수들을 포함한다.

- 이 클래스의 지역 함수 m을 정의해서 상속받은 클래스가 나타내는 장치에서 실행한다. 보통의 함수를 호출하는 방법과 동일하게 m()으로 이 지역 함수를 호출한다.
- 장치의 운영체제 또는 런타임 시스템에서 제공하는 라이브러리 함수는 특별한 변경 없이 동일한 방식으로 자유롭게 사용한다.
- 다른 클래스 D'에서 정의된 원격 함수를 D'.m()으로 호출한다. 이 방법은 마치 객체지향 프로그래밍 언어에서 클래스 이름으로 정적 메서드를 호출하는 것과 유사하다.

예를 들어, Fig. 5에서 MyArduino 클래스에 readDoorSensor()는 이 장치에서 실행하는 지역함수이고, digitalRead()는 아두이노 운영체제에서 제공하는 지역 라이브러리 함수이며, MyRaspberry.reqSend()는 라즈베리 파이 기반 게이트웨이에서 실행할 함수를 지정한 원격 함수 호출이다.

Fig. 5의 예제 프로그램은 아두이노 운영체제에서 시작한다. 이 운영체제가 시작하면 setup() 함수를 호출한다. 이 함수에 아두이노와 라즈베리 파이를 연결하는 시리얼통신을 초기화하고 주기적으로 loop() 함수를 호출하여 문에 장착된 센서 데이터를 읽는다. 문의 상태가 달라질 때마다 아두이노 임베디드 시스템은 이 센서 데이터를 읽고, 닫힘 상태이면 'c'를 열림 상태이면 'o'를 라즈베리 파이 기반 게이트웨이에 보낸다. 이때 MyRaspberry.reqSend() 함수를 호출하면 게이트웨이에서 이 장치에 부착된 카메라를 통해 문 앞을 찍어 사진을 만든다. 그런 다음 MyCloud.recordDoorState() 함수를 호출해서 클라우드 서버에 센서 데이터와 사진을 보내고 시간 정보와 함께 이 서버에 저장한다.

```
class MyArduino(Arduino):
 _int_doorPin = 2
 _int_preState = LOW
 _int_postState = LOW

 def _void_setup():
   pinMode(doorPin, INPUT)
 def _void_loop():
   readDoorSensor()
 def _void_read_DoorSensor():
   postState = digitalRead(doorPin)
   if postState==HIGH and preState==LOW:
    MyRaspberry.reqSend('c')  # close
    preState = postState
   if postState==LOW and preState==HIGH:
    MyRaspberry.reqSend('o')  # open
    preState = postState

class MyRaspberry(Raspberry):
 def reqSend(val):
   if val== 'c' or val =='o':
    pic_bin = takePhoto()
    MyCloud.recordDoorState(val, pic_bin)

class MyCloud(Cloud):
 def recordDoorState(openclose, pic_bin):
   pic_loc = save_picture(pic_bin)
   time = getTime()
 # insert into
 #  doorStateTable
 #    (col_openclose, col_pic_loc, col_time)
 #  values (openclose, pic_loc, time)
```

Fig. 5. Example of Integrated Program Collecting Magnetic Sensor Information

예제 프로그램에서 UniPy 프로그래밍언어의 원격 함수 호출 방법은 지역 함수 호출과 매우 유사하지만 내부적으로 이 함수를 호출한 장치와 이 함수를 실행할 장치 사이에 통신을 지원해야 한다. 기본적으로 임의의 두 장치 간 통신 라이브러리를 미리 준비한다. 클래스 D에서 원격 함수 D'.m()을 호출할 때마다 이 통신 라이브러리를 사용하도록 컴파일러가 자동으로 코드를 생성하여 지원한다. 이 논문에서 다루는 홈 모니터링 시스템을 위해 Table 1에서 보여주는 목록의 통신 라이브러리를 준비하였다.

Table 1. A Catalogue of Communication Libraries for Connected Devices in the Home Monitoring System

From	To	Protocol
Arduino	Raspberry	Serial
Raspberry	Arduino	Serial
Raspberry	Cloud	HTTP
Client	Cloud	HTTP
Cloud	Client	HTTP
Client	Raspberry	Socket

UniPy 컴파일러는 컴파일 옵션으로 두 장치 간 통신 프로토콜 종류와 이 프로토콜의 세부 설정 방법을 받는다. 예를 들어, 이 논문 연구에서 개발한 UniPy 컴파일러 (unipyc)는 MyArduino로부터 MyRaspberry로의 통신 방법과 그 다음 MyCloud로의 통신 방법을 아래와 같이 컴파일 옵션을 받는다.

```
$ unipyc doorSensorCollection.py
  -MyArduino:MyRaspberry
   :Serial:baudrate=9600
  -MyRaspberry:MyCloud
   :Http:ipaddr=192.168.0.123
```

사물인터넷 시스템을 위한 새로운 장치를 추가하더라도 각 장치에서 사용할 수 있는 통신 프로토콜 목록은 보통 고정되어 있다. 따라서 UniPy 컴파일러에서 이 장치에서 실행할 원격 함수를 호출할 때 데이터 송수신에 필요한 통신 라이브러리를 각 장치 별로 준비할 수 있다. 클래스 D에서 원격 함수 D'.m()을 호출하는 경우,

- 이 원격 함수 호출 문장을 D 장치에서 D' 장치로 데이터를 보내는 통신 라이브러리를 컴파일 옵션에 따라 선택하고 함수 인자를 보내는 통신 문장들로 변환한다.
- 원격 함수 m의 정의에서 형식 인자를 제거하고 함수 시작에 이 인자들의 값을 받는 통신 라이브러리를 사용하는 문장들을 추가한다.

원격 함수 호출 관련 컴파일 방법에서 한 위치에 둘 이상의 원격 함수가 존재할 때 목적하는 함수를 선택하여 호출되도록 고려해야 한다. 각 장치 별로 통신을 주고받기 위한 별도의 디스패치(dispatch) 모듈이 있어서 호출되는 함수를 연결해주는 방식이다. 따라서 동일한 장치에 존재하는 원격 함수들을 구분할 수 있는 식별 번호를 정하고 해당 원격 함수를 호출할 때 이 식별 번호도 함께 전달하여 제대로 연결될 수 있도록 구성해야 한다. 원격 함수 문장을 컴파일 할 때 UniPy 컴파일러는 자동으로 각 원격 함수에 고유 식별자를 부여하고 디스패치 모듈에서 장치로 들어올 수 있는 모든 식별자를 구분하여 해당 함수로 연결할 수 있도록 구성하였다.

지금까지 설명한 UniPy의 특징과 구현 시 고려 사항들을 종합하여 컴파일 과정을 설명한다. 입력을 받은 UniPy 프로그램을 각 클래스로 나누어 해당 위치의 분리된 작은 프로그램들을 출력한다. 각 클래스 선언을 제거하고 모든 필드와 메서드 선언을 각 장치를 위한 프로그램에서의 전역 변수와 전역 함수로 변환한다.

앞에서 소개한 UniPy 컴파일 실행 예에서 doorSensorCollection.py를 입력 받아 3개의 작은 프로그램들을 생성한다. 각각 아두이노 장치에서 동작할 아두이노 C언어로 작성된 MyArduino.ino, 라즈베리 파이 장치에서 동작할 파이썬으로 작성된 MyRaspberry.py, 클라우드 서버에서 동작할 파이썬으로 작성된 MyCloud.py이다. Fig. 6에서 생성된 프로그램들을 보여준다.

첫째, Fig. 5의 UniPy 예제 프로그램에서 원격 함수를 호출하는 각 문장을 컴파일 옵션에서 지정한 통신 프로토콜의 통신 방법으로 데이터를 전송하는 문장으로 컴파일 변환한다. 예를 들어, 아두이노 장치에서 MyRaspberry.reqSend('c')를 실행하는 문장을 컴파일하면 시리얼 통신으로 전송하는 Serial.println 함수를 사용하는 문장 2개로 변환된다. 처음에 원격 함수 reqSend의 식별자 3을 보내고, 그 다음에 실제 인자 'c'를 보낸다. 컴파일 옵션 중 MyArduino:MyRaspberry:Serial:baudrate=9600은 아두이노 장치에서 시리얼 통신을 초기화할 때 필요한 설정 값을 지정한다.

```
/// MyArduino.ino ////////////////////////////////////////
int doorPin = 2
int preState = LOW;
int postState = LOW;
void setup() {
 Serial.begin(9600);
 pinMode(doorPin, INPUT);
}
void loop() {
 readDoorSensor();
}
void readDoorSensor() {
 postState = digitalRead(doorPin);
 if (postState==HIGH && preState==LOW) {
  Serial.println(3);
  Serial.println(99); /* 'c' */
  preState = postState;
 }
 if (postState==LOW && preStaet==HIGH) {
  Serial.println(3);
  Serial.println(111); /* 'o' */
  preState = postState;
 }
}
/// MyCloud.py ///////////////////////////////////////////
def _dispatch():
 funid = getCgiField('_funid')
 if (funid == 4):
    recordDoorState()
```

```
/// MyRaspberry.py ///////////////////////////////////////
url = 'http://192.168.0.123/common.php'
ser = Serial()

def _dispatch():
 _funid = ser.read(1)
 if (_funid == 3)
    reqSend()
def reqSend():
 val = ser.read(1)
 if (val == 99):  /* 'c' */
    pic_bin = takeAPhoto()
    req.request('POST', url,
     data={'_ARGSPRG' : 'MyCloud.py',
           '_funid' : '4',
           '_ARGS0' : 'c',
           '_ARGS1' : pic_bin})
 if (val == 111): /* 'o' */
    pic_bin = takeAPhoto()
    req.request('POST', url,
     data={'_ARGSPRG' : 'MyCloud.py',
           '_funid' : '4',
           '_ARGS0' : 'o',
           '_ARGS1' : pic_bin})
/// MyCloudy.py (Cont.) ////////////////////////////////
def recordDoorState():
 openclose = getCgiField('_ARGS0')
 pic_bin = getCgiField('_ARGS1')
 d_pic_loc = save_picture(pic_bin)

# insert into
#     doorStateTable(d_openclose, d_pic_loc)
#     values (openclose, d_pic_loc)
#     ...
```

Fig. 6. Automatically Generated Programs From the UniPy Program Example

Fig. 6의 setup() 함수에서 시리얼통신의 baudrate를 설정하는 문장을 확인할 수 있다. 라즈베리 파이 기반 게이트웨이는 시리얼 통신으로 수신자 역할을 한다. _dispatch() 함수는 아두이노 기반 임베디드 장치에서 시리얼통신을 통해 들어오는 모든 데이터를 받는다. _dispatch() 함수가 읽은 첫 번째 바이트는 함수 식별자로 reqSend 함수를 가리키고, 두 번째 바이트는 이 함수의 실인자로 문 개폐 상태에 대한 센서 값이다. 위 컴파일러 옵션은 라즈베리 파이 기반 게이트웨이에서도 유사하게 시리얼통신을 초기화할 때 ser=Serial() 문장에서 사용한다. 라즈베리 파이 게이트웨이에서 클라우드 서버로 데이터를 전달하는 과정에서도 동일한 방식으로 컴파일 된다. 다만 통신 프로토콜이 시리얼통신이 아닌 HTTP로 바뀌었다.

이와 같이 위치를 나타내는 방법으로 클래스를 사용하는 설계 아이디어는 매우 유용하다. 직관적으로 여러 다른 장치들을 통합하는 방법을 제공한다. 특히 파이썬을 확장하여 UniPy 프로그래밍언어를 설계할 때 위치를 지정하기 위한 별도의 키워드를 도입하는 것과 같은 확장을 요구하지 않는 장점도 있다. 여러 장치가 관여하는 사물인터넷 시스템에서 작동하는 프로그램을 작성하려면 프로그래머는 이 장치들 수만큼 클래스를 작성하면 된다.

또한 클래스를 장치의 위치를 지정하는 방법으로 사용하는 설계는 각 장치에서 제공하는 라이브러리 함수나 런타임 시스템과 같은 장치에 의존적인 사항을 추상화하기에 적합하다. UniPy 프로그램을 컴파일하기 위해서 기반 클래스들의 이름(Arduino, Raspberry, Cloud)만 참조하되 각 장치에서 제공하는 라이브러리 함수와 런타임 시스템을 사용하도록 컴파일러에서 코드를 생성하면 된다.

서로 다른 유형의 장치에 대한 차이점을 각 장치에 대한 컴파일 방법을 준비함으로써 해소한다. 예를 들어, MyArduino와 MyRaspberry에서 시리얼 통신을 초기화하고, MyCloud에서 HTTP기반 서버를 지원하기 위한 PHP 웹 서버 모듈을 준비해야 한다. 또한 MyArduino 파이썬 문장을 Arduino C 문장으로 변환할 때 이 변환을 쉽게 하기 위하여 특별한 이름 규칙에 따라 변수 이름과 함수 이름을 사용해야 한다. 그 이유는 파이썬 프로그램에서 변수 타입 선언을 요구하지 않지만 Arduino C 프로그램에서는 필요하기 때문이다. 이렇게 각 장치마다 컴파일러가가 고려해야할 사항들이 있고 각 유형의 장치마다 준비해놓았다. 따라서 UniPy 컴파일러는 기반 클래스에서 유추한 위치 문맥에 따라 각 장치에 의존적인 코드를 생성하도록 설계되었다.

이러한 설계 방식은 UniPy 프로그램 실행을 시뮬레이션하기에도 적합하다. 그 이유는 기반 클래스에 그 장치에서 제공하는 라이브러리 함수와 런타임 시스템을 시뮬레이션 하는 기능을 제공하면 UniPy 프로그램을 분리하지 않고 파이썬 해석기에서 직접 실행하고 테스트할 수 있다. 이러한 장점은 UniPy 프로그램을 컴파일해서 실제 장치에 배치하고 실행하기 전에 장치 간 연동 기능을 테스트하는데 적합하다.

III. Evaluation

제안한 UniPy 프로그래밍언어와 컴파일러를 평가하기 위하여 Fig. 2의 홈 모니터링 시스템에서 3가지 주요 시나리오로 구성된 3개의 UniPy 프로그램, 센서 데이터 수집 프로그램, 센서에 명령을 내리는 프로그램, 센서 데이터를 저장한 데이터베이스를 활용하는 프로그램을 개발하였다. 이 논문 연구에서 개발한 UniPy 컴파일러와 평가를 위해 개발한 3개의 UniPy 프로그램들, 홈 모니터링 시스템의 상세 명세를 다음 웹 사이트에서 확인할 수 있다.

https://github.com/kim-gayoung/unipy/

첫째, UniPy 프로그래밍언어를 통해 여러 장치들이 관여한 분산 프로그램을 간략하게 작성할 수 있음을 Fig. 5의 예제를 통하여 확인할 수 있었다. 홈 모니터링 시스템의 3가지 주요 시나리오에 대한 프로그램을 작성할 수 있었다. 임베디드 장치, 게이트웨이, 클라우드가 연동해서 하나의 시나리오를 구성하기 위해 프로그래머는 각 원격 장치들의 함수들을 차례로 호출하도록 프로그램을 작성하기만 하면 된다. UniPy 컴파일러가 자동으로 각 장치별로 분리된 프로그램들을 생성하고 원격 함수를 호출하는 통신 코드도 만들어준다. 게다가 원격 함수 호출 방법을 제외하면 나머지는 기존 장치를 직접 프로그래밍 하는 방식으로 동일하게 프로그램을 작성하면 되기 때문에 UniPy 프로그래밍을 쉽게 배울 수 있는 장점이 있다. 예를 들어, MyArduino 클래스에서 사용한 pinMode와 digitalRead는 모두 아두이노 라이브러리에서 제공하는 함수들이고, MyRaspberry 클래스에서 사용한 takePhoto는 라즈베리 파이에서 제공한 함수들을 가지고 작성한 것이다.

둘째, UniPy의 클래스 기반 프로그램 구조는 통합 프로그램을 테스트하기에 편리하다. 각 장치 위치를 나타내는 기반 클래스를 상속받아 작성한 클래스에서 사용하는 장치의 라이브러리 함수를 시뮬레이션하면 파이썬 환경에서 UniPy 프로그램을 직접 실행할 수 있다. Fig. 5의 예제를 클라우드 서버의 파이썬 환경에서 이 예제 프로그램을 실행하기 위해서 아두이노와 라즈베리 파이 장치에서 제공하는 다음 라이브러리 함수들을 시뮬레이션하면 된다.

- Arduino.pinMode()와 Arduino.digitalRead()
- MyArduino.setup() 함수를 호출하고 loop() 함수를 반복해서 호출할 드라이버 프로그램
- Raspberry.takePhoto()에서 사용한 라이브러리 함수

각 장치의 라이브러리 함수와 런타임 시스템을 시뮬레이션하는 모듈을 준비해놓으면 UniPy 프로그램을 미리 테스트할 수 있는 장점이 있다. 이러한 장점은 Fabryq [8]과 Ravel [9]와 같은 이전 연구에서 제공하지 못했던 특징이다.

요약하면, UniPy 프로그래밍언어와 컴파일러는 MGC(eMbedded, Gateway, Cloud) 유형의 사물인터넷에서 동작할 통합 프로그램을 작성할 때 유용한 도구임을 확인할 수 있었다.

UniPy 프로그래밍언어와 컴파일러 기반 방법은 2가지 제약사항이 있다. 첫째, 홈 모니터링 시스템과 같이 장치 간 연결 그래프가 고정된 사물인터넷이 아니라 동적으로 새로운 장치를 연결하고 해제하는 시나리오를 대응하지 못한다. 예를 들어 Fabryq [8]은 BLE 통신 프로토콜을 기반 연결 방법으로 삼아 사물인터넷의 장치들 연결 그래프가 동적으로 변하는 경우를 고려하였다.

둘째, UniPy 프로그래밍언어 설계의 기초로 도입했던 RPC 계산법은 비동기적 연결 방식을 사용하는 사물인터넷 구성하는데 한계가 있다. 원격 함수를 호출한 다음 그 결과가 반환되기까지 시간이 지체될 수 있으므로 보통 비동기적으로 콜백 함수를 도입해서 프로그램을 하는데 UniPy 프로그램은 그 결과가 돌아올 때까지 현재 장치의 실행을 중지하는 구조이다. 이 논문에서 고려한 3가지 시나리오의 경우 이러한 원격 함수의 동기 호출 방식이 문제가 되지 않았다. 하지만 예를 들어 임베디드 장치에서 얻은 센서 데이터를 게이트웨이를 거쳐 클라우드 서버까지 파이프라인 방식으로 전송하는 시나리오는 RPC 계산법으로 직접 표현하기 어렵다.

IV. Related Work

아두이노와 라즈베리 파이 각 장치에 대한 고급 프로그래밍 모델을 제안한 연구가 있다. Barbon은 아두이노와 같은 마이크로컨트롤러에서 동작하는 코드를 위한 소프트웨어 아키텍처로써 아두이노 서비스 인터페이스 프로그래밍 (ASIP) 모델을 제안하였다 [1]. 이 모델은 아두이노 장치와 이 장치의 사용자 사이에 메시지를 쉽게 주고 받을 수 있도록 텍스트 기반 프로토콜을 정의하였다. IBM에서 개발한 오픈소스 프로젝트 Node-RED(NR) [12]는 사물인터넷 서비스를 개발하기 위해 데이터 플로우(data-flow) 프로그래밍 모델을 사용한다. 라즈베리 파이와 같은 장치도 지원한다. 프로그램 작성을 위해 Node.js 상에서 자바스크립트 언어를 사용한다.

임베디드 장치, 게이트웨이, 클라우드, 3계층으로 구성된 사물인터넷 시스템을 대상으로 PatRICIA [2], IoTvity [13], Mobile Fog [14], DDF [15]와 같은 여러 고급 프로그래밍 모델을 설계한 연구가 있었다. 사물인터넷 시스템의 다양성, 복잡성, 크기로 인한 프로그램 개발의 어려움을 해결하고자 이러한 모델들을 제안하였다. 하지만 이 모델들 모두 개별 계층과 장치별로 따라 프로그램을 개발하는 형태이다.

IoTvity는 각종 장치, 응용프로그램, 서비스가 서로 연동할 수 있는 표준 모델로 OCF 프레임워크을 설계하였다. 이 프레임워크에 대한 참조 구현을 제공한다 [16].

PatRICIA [2]는 인텐트(intent)와 인텐트 범위(intent scope)라는 메시지 개념을 기반으로 클라우드 플랫폼에서 사물인터넷 프로그램을 작성하는데 활용할 수 있는 고급 프로그래밍 모델이다. 클라우드 상에서 동작하는 사물인터넷 프로그램을 쉽고 직관적으로 개발할 수 있도록 추상화 개념을 정의하였고 클라우드와 연동하는 다양한 시스템을 다룰 수 있도록 런타임 시스템을 제공한다.

포그(fog) 기반 사물인터넷 프로그램을 작성하기 위한 고급 프로그래밍 모델에 관한 연구도 있다. Mobile Fog [14]는 주변 컴퓨팅 장치를 이용하는 프로그램을 쉽게 작성할 수 있도록 도와주는 고급 프로그래밍 모델이다. 단순하고 추상화된 프로그래밍 방법을 제공하고 실행하면서 사용할 장치의 수를 프로그램에서 필요한 계산 양에 따라 동적으로 변경할 수 있다.

분산 데이터플로우(DDF, Distributed Data-Flow)는 포그 기반 사물인터넷 환경에서 빠르게 프로그램을 개발하고 컴퓨팅 자원을 제공하는 장치들에 분산된 자원들을 쉽게 관리할 수 있는 프로그래밍 모델이다 [15].

분산 노드레드(D-NR, Distributed Node-RED)는 기존 노드레드 (NR) 런타임 시스템을 확장하여 개발자로 하여금 포그 기반 사물인터넷을 위한 프로그램을 작성할 때 장치와 클라우드 간의 자원 관리를 쉽게 해주는 방법을 제공한다 [3].

이제까지 나열한 관련 연구에서 제안한 아이디어들을 무선 센서 네트워크를 프로그래밍하는 문제에 적용한 연구들도 많이 있다 [7,17,18,19]. 예를 들어, Pleiades 언어 [7]는 무선 센서 네트워크를 위해 추상화한 프로그래밍 방법을 제공한다.

사물인터넷 프로그램을 개발하기 위한 통합 프로그래밍 모델에 관한 연구가 있다. RPC 계산법 [10]은 기존 순차 함수형 프로그램에 대한 이론적 모델인 람다 계산법에 위치를 지정하는 특징을 추가하여 확장한 것이다. 이 특징을 활용하여 프로그래머는 각 함수에 실행할 위치를 지정할 수 있고, 컴파일러와 런타임 시스템에서 장치 간 통신 방법 등을 자동으로 지원한다. 이러한 특징은 원격 함수 호출(RPC, Remote Procedure Call) 방법이 내재된 언어를 설계할 때 참조할 수 있는 기초가 된다. 이 계산법에서 웹 클라이언트 서버 환경에서 맞도록 통합 프로그램을 분리하는 컴파일 방법도 제안하였다.

Fabryq [8]은 MGC(eMbedded-Gateway-Cloud) 구성의 사물인터넷 시스템을 위한 프로그램을 작성하기 위한 통합 프로그래밍 모델이다. 자바스크립트 언어를 사용하여 개발하며 임베디드 장치와 클라우드의 원격 함수를 쉽게 호출할 수 있는 추상화된 방법을 제공한다.

Ravel [9]는 모델-뷰-컨트롤러 아키텍처를 3계층 사물인터넷 시스템의 분산 환경으로 확장한 프로그래밍 모델을 제공한다. 특정 계층의 속성과 구현 방법을 포함하는 위치를 지정하는 방법을 사용할 수 있다.

V. Conclusions

본 논문에서 MGC 구성의 사물인터넷 시스템을 대상으로 통합 프로그램을 작성하기 위한 통합 프로그래밍언어 UniPy를 설계 및 구현하고 홈 모니터링 시스템을 대상으로 평가하였다. UniPy 언어를 사용하면 장치의 위치를 기반 클래스를 통해서 지정하고 해당 장치에서 동작할 코드를 상속받는 클래스를 작성하며 통합 프로그램을 작성할 수 있다. 다른 장치에서 실행할 원격 함수를 호출하는 방법은 동일한 장치에 있는 지역 함수를 호출하는 방법과 동일하여 여러 장치들을 연동하는 프로그램을 쉽게 작성할 수 있다. 이렇게 작성한 통합 프로그램은 UniPy 컴파일러에 의해 각 장치에서 실행할 프로그램들로 분리되고 장치 간 함수 호출에 대해서 통신이 이뤄질 수 있도록 코드를 자동으로 생성한다.

UniPy 프로그래밍언어를 활용하여 아두이노, 라즈베리 파이, Web기반 데이터베이스 서버를 이용한 홈 모니터링 시스템 환경에서 센서 데이터 수집, 센서 명령어, 센서 데이터 데이터베이스 활용의 3가지 시나리오에 대해 통합 프로그래밍 언어로 프로그램을 작성하였다. 통합 프로그래밍 방법은 프로그래머가 장치 위치를 나타내는 클래스 개념을 사용하고, 매우 간단한 방법으로 원격 함수를 호출을 할 수 있어 쉽게 사물인터넷 프로그램을 개발할 수 있다. 또한 분산된 장치에 개별적인 프로그램을 개발하고 이들을 모두 모아 테스트 했던 프로그래머들의 부담을 줄일 수 있을 것이다.

향후 더욱 다양한 장치와 통신 방법을 지원하도록 통합 사물인터넷 프로그래밍 언어, 컴파일러, 라이브러리를 확장하는 연구가 필요하다. 그리고 이를 활용하여 사물인터넷 프로그램을 개발한 경험을 축적하여 제안한 방법의 장점을 정량적으로 평가하는 연구를 수행할 계획이다.

REFERENCES

[1] G. Barbon, M. Margolis, F. Palumbo, F. Raimondi, and N. Weldin, “Taking Arduino to the Internet of Things: The ASIP Programming Model,” Computer Communications, Vol. 89-90, pp. 128-140, September 2016.

[2] S. Nastic, S. Sehic, M. Vögler, H. L. Truong, and S. Dustdar, “PatRICIA - A Novel Programming Model for IoT Application on Cloud Platforms,” Service-Oriented Computing and Applications (SOCA) 2013 IEEE 6th International Conference on, pp. 53-60, December 2013.

[3] M. Blackstock and R. Lea, “Toward a Distributed Dataflow Platform for the Web of Things(Distributed Node-Red),” Proceedings of the 5th International Workshop on Web of Things(WoT ’14), pp. 34-39, Cambridge, MA, USA, October 2014.

[4] S. Nastic, H. L. Truong, and S. Dustdar, “SDG-Pro: A Programming Framework for Software-defined IoT Cloud Gateways,” Journal of Internet Services and Applications, Vol. 6, No. 1, pp.1-17, 2015.

[5] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, “Links: Web Programming without Tiers,” Proceedings of the 5th International Conference on Formal Methods for Components and Objects (FMCO ’06), pp. 266-296, Amsterdam, The Netherlands, November 2006.

[6] T. Murphy VII, K. Crary, R. Harper, and F. Pfenning, “A Symmetric Modal Lambda Calculus for Distributed Computing,” Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS ’04), pp. 286-295, Washington DC, USA, July 2004.

[7] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, “Reliable and Efficient Programming Abstractions for Wireless Sensor Networks,” ACM SIGPLAN Notices - Proceedings of the 2007 PLDI conference, Vol. 42m No. 6, pp. 200-210, June 2007.

[8] W. McGrath, M. Etemadi, S. Roy, and B. Hartmann, “Fabryq: Using Phones as Gateways to Prototype Internet of Things Applications Using Web Scripting,” Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS ’15), Duisburg, Germany, pp. 164-173, June 2015.

[9] L. Riliskis, J. Hong, and P. Levis, “Ravel: Programming IoT Applications As Distributed Models, Views, and Controllers,” Proceedings of the 2015 International Workshop on Internet of Things Towards Applications (IoT-App ’15), pp. 1-6, November 2015.

[10] E. Cooper and P. Wadler, “The RPC Calculus,” Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP ’09), pp. 231-242, New York, USA, September 2009.

[11] M. Neubauer and P. Thiemann, “From Sequential Programs to Multi-tier Applications by Program Transformation,” Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages(POPL ’05), pp. 221-232, New York, USA, January 2005.

[12] IBM, Node-red, *https://nodered.org/*

[13] T. Dang, M. Tran, D. T. Le, and H. Choo, “On Evaluating IoTivity Cloud Platform,” International Conference on Computational Science and Its Applications (ICCSA 2017), Trieste, Italy, July 2017.

[14] K. Hong, D. J. Lillethun, U. Ramachandran, B. OttenWälder, and B. Koldhofe, “Mobile fog: A

Programming Model for Large-Scale Applications on the Internet of Things," Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing(MCC '13), pp. 15-20, Hong Kong, China, August 2013.

[15] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing IoT Applications in the Fog: A Distributed Dataflow Approach," In Proceedings - 2015 5th International Conference on the Internet of Things (IoT '15), pp. 155-162, Seoul, Korea, October 2015.

[16] O. C. Foundation, IoTivity, *https://www.iotivity.org/*

[17] B. Greenstein, E. Kohler, and D. Estrin, "A Sensor Network Application Construction Kit (SNACK)," Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04), pp. 69-80, Baltimor, USA, November 2004.

[18] R. Gummadi, O. Gnawli, and R. Govindan, "Macro-Programming Wireless Sensor Networks Using Kairos," Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '05), pp. 126-140, Marina del Rey, CA, USA, June 2005.

[19] Y. Tu, Y. Li, T. Chien, and P. H. Chou, "Ecocast: Interactive, Object-Oriented Macro-Programming for Networks of Ultra-Compact Wireless Sensor Nodes," Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN '11), pp. 366-377, Chicago, IL, USA, pp. 366-377, April 2011.

Authors

Gayoung Kim received the B.S. degree in Electronics and Computer Engineering from Chonnam National University, Korea, in 2018. She is currently a Master Student in the Software Languages and Systems Laboratory, Chonnam National University. She is interested in Software Engineering, Programming Language and Compiler.

Kwanghoon Choi received the B.S, M.S and Ph.D. degrees in Computer Science from KAIST, Korea, in 1994, 1996 and 2003, respectively. From 2006 to 2010, he joined a senior research engineer in LG electronics. From 2011 to 2016, he was a professor in the Department of Computer and Telecommunications Engineering Division, Yonsei University, Wonju. He is currently a professor in the Department of Electronics and Computer Engineering, Chonnam National University. His research interests are Programming Language, Compiler, Program Analysis and Software Engineering.

Byeong-Mo Chang received the B.S degree in Computer Engineering from Seoul National University, Korea, in 1988. He received the M.S and Ph.D. degrees in Computer Science from KAIST, Korea, in 1990 and 1994. He is currently a professor in the Department of Computer Science, Sookmyung Women's University. His research interests are in the area of Software Security, Programming Language and Program Analysis/Verification.