Sejong Univ Smoke Map

Technical Report

Team SAYS

2017. 12. 08.

Team members:

Areum Lee
Seongwon Hyun
Seok Yoon

Contents

1 Introduction

1.1 Purpose

Our project aims to provide an IoT based smoke map for Sejong University Campus.

1.2 Motivation

Complaints about smokers who smoke outside of smoking zones have been ever existing within the campus. Many efforts were made to solve this problem, but they were unsuccessful. To tackle this problem, we decided to suggest a new solution.

1.3 Goals

To develop a successful smoke map service, we set the following goals:

- Detect smoke in non-smoking areas.
- If smoke is detected, alert the smoker via buzzer.
- Show the density of smoke on campus map.
- Provide access to the map through a mobile application.

1.4 Challenges

There were several challenges during this project:

- oneM2M had to be studied from scratch:
  None of the team members had experience with oneM2M.
- Late arrival of devices:
  We could not start a lot of tasks until we received the required devices.
- No one had experience with sensor and actuators
- Mobius server setting was difficult due to a lot of dependencies
- One our members left the team when we were 1 month into the project:
  So, we were a team of 3 members when others had 4-6 members.

2 Project Management

2.1 Schedule Management

Work Distribution

- Areum Lee : Project managem ent, SW design, Embedded device connections, Documentation
- Seongwon Hyun : Server development, Client development
- Seok Yoon : Smartphone application development

WBS

To Do				9월		10월					11월					12월
대분류	소분류	상세	담당	4	5	1	2	3	4	5	1	2	3	4	5	1
1. 설계 및 준비	1.1. 요구사항 분석	1.1.1. 요구사항 파악 및 해결책 도출	전원													
	1.2. 플랫폼 테스트 및 사전조사	1.2.1 Postman 환경구축 및 공부	전원													
		1.2.2 Mobius 서버 구축 및 동작 확인	현성원													
		1.2.3 &cube 클라이언트 구축 및 동작 확인	현성원													
		1.2.4 센서 데이터 형태 조사	전원													
		1.2.5 oneM2M 표준 API 조사	이아름													
		1.2.6 IoT 플랫폼 자료조사 (oneM2M)	현성원													
		1.2.7 아두이노&센서 연결 통신 실습	**이아름**, 현성원													
	1.3. 요구사항 및 설계 구체화	1.3.1. 요구사항 명세 정의	전원													
		1.3.2 Resource Tree 설계	이아름													
		1.3.3 아키텍처 설계	**이아름**,현성원													
		1.3.4 Call Flow 정의	이아름													
		1.3.5 모바일 어플리케이션 구성	윤석													
		1.3.6 요구사항 명세 작성	이아름													
		1.3.7 소프트웨어 개발 명세 작성	이아름													
		1.3.9 기자재 조사 및 신청	전원													
2. 개발	2.1. Mobius 서버	2.1.1 Resource Tree 생성	이아름													
		2.1.2 oneM2M Browser 도입	현성원													
	2.2. 라즈베리 파이 & things	2.2.1 센서 & 부저 schematic 공부	이아름													
		2.2.2 라즈베리파이 GPIO schematic 공부	이아름													
		2.2. 라즈베리파이 OS 설치 및 개발환경 구축	현성원													
		2.2. GPIO 사용을 위한 세팅 및 공부	**이아름**, 현성원													
		2.2. MCP 칩 연결	이아름													
		2.2.2 센서-라즈베리 연결	이아름													
		2.2.3 부저-라즈베리 연결	이아름													
		2.2.4 라즈베리파이 WPA2 Enterprise 공부	현성원													
		2.2.5 부저 trigger 코드 작성	현성원													
		2.2.6 센서 데이터 리시버 코드 작성	현성원													
		2.2.7 부저 비활성화 코드 작성	현성원													
		2.2.8 Mobius 서버 데이터 전송 코드 작성	현성원													
		2.2.9 코드 취합 및 스모크 디텍터 코드 작성	현성원													
		2.2.10 더미데이터 자동 생성기 코드 작성	현성원													
		2.2.11 Startup 스크립트 작성	현성원													
		2.2.12 코드 리뷰 및 버그 수정	현성원													
	2.3. App 개발	2.4.1 화면 구성 (Scene manager)	윤석													
		2.4.2 ppm 표시 영역 구현 (Object pool)	윤석													
		2.4.3 탐색 바 (Drawer Navigation)	윤석													
		2.4.4 ppm에 따른 색상 변경 기능 (Data manager)	윤석													
		2.4.5 Mobius 서버와 통신	**윤석**, 현성원													
3. 테스트 및 보완	3.1. Mobius	3.1.1 Resource 객체 생성 확인	이아름													
	3.2 라즈베리 파이 & things	3.2.1 센서 동작 테스트	이아름													
		3.2.2 ppm 값 전송 테스트	현성원													
		3.2.3 부저 동작 테스트	**이아름**, 현성원													
		3.2.4 값 변화에 따른 부저 반응 테스트	**현성원**, 이아름													
		3.2.5 더미데이터 전송 스크립트 테스트	현성원													
		3.2.6 부저 비활성화 스크립트 테스트	현성원													
		3.2.7 스모크 디텍터 기능 테스트	현성원													
		3.2.8 발표 전 환경 설정	전원													
	3.3 App	3.3.1 앱 더미 데이터 receive 테스트	현성원													
4. 마일스톤	4.1 1차 발표	4.1.1 요구사항 명세서 작성	이아름		★											
	4.2 설계명세서	4.2.1 설계명세서 과제물 작성	**이아름**, 현성원							★						
	4.3 Pitch 준비	4.3.1 최종 Technical Report 작성	이아름													
		4.3.2 Pitch 발표자료 작성														
		4.3.3 라즈베리파이 시연 영상 준비	**이아름**, 현성원													
		4.3.4 oneM2M Browser 시연 영상 준비	현성원													
		4.3.5 모바일 어플리케이션 시연 영상 준비	**윤석**, 현성원													
		4.3.6 발표 시연 리허설	전원													
		4.3.7 최종 발표	전원													★
		교수님 1차 온라인 미팅	전원							★						
		교수님 2차 오프라인 미팅	전원											★		

2.2 Collaboration Tools

The following collaboration tools were used:

- Github - for development
- Slack - for sharing project documents, code, and references.

3 Design specifications

3.1 Use Case

This clause briefly describes the use case from perspective of service being provided by the oneM2M platform. The physical device components are introduced in the current clause.

The described use case enables smoke monitoring through an android mobile application, which has access to a oneM2M service platform.

An overview of the use case is shown in Fig.1. The main components are as follows:

- Smoke detecting sensor
- Alarm actuator
- Raspberry Pi
- Mobius server
- Smartphone application

Each smoke detecting sensor and actuator are attached to a Raspberry Pi board, and are deployed in any place as needed. Each Raspberry Pi, in turn, is connected with Mobius server.

The smartphone hosts an application to retrieve and display data from the sensors.

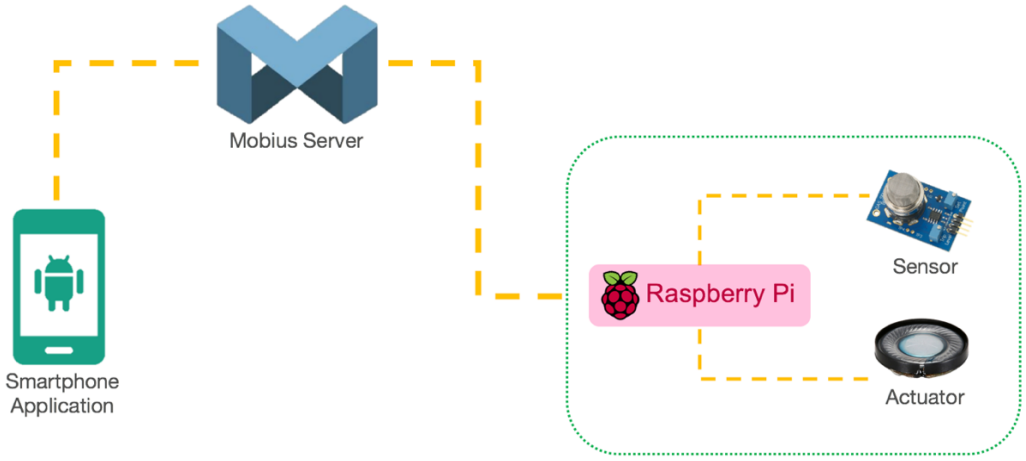


Fig. 1 Overview of smoke monitoring use case

3.2 Architecture

This clause describes the architecture of the implemented use case with components represented by the oneM2M entity roles. In this use case:

- IN-CSE (Infrastructure Node - Common Services Entity) is hosted in the Mobius server.
- All Raspberry Pi are grouped as containers under a single ADN-AE (Application Dedicated Node - Application Entity) named ADN-AE-SAYS.
- Sensors and actuators are sub containers under each Raspberry Pi container.
- The smartphone hosts an ADN-AE (Application Dedicated Node - Application Entity).

The architecture is shown in Fig.2.

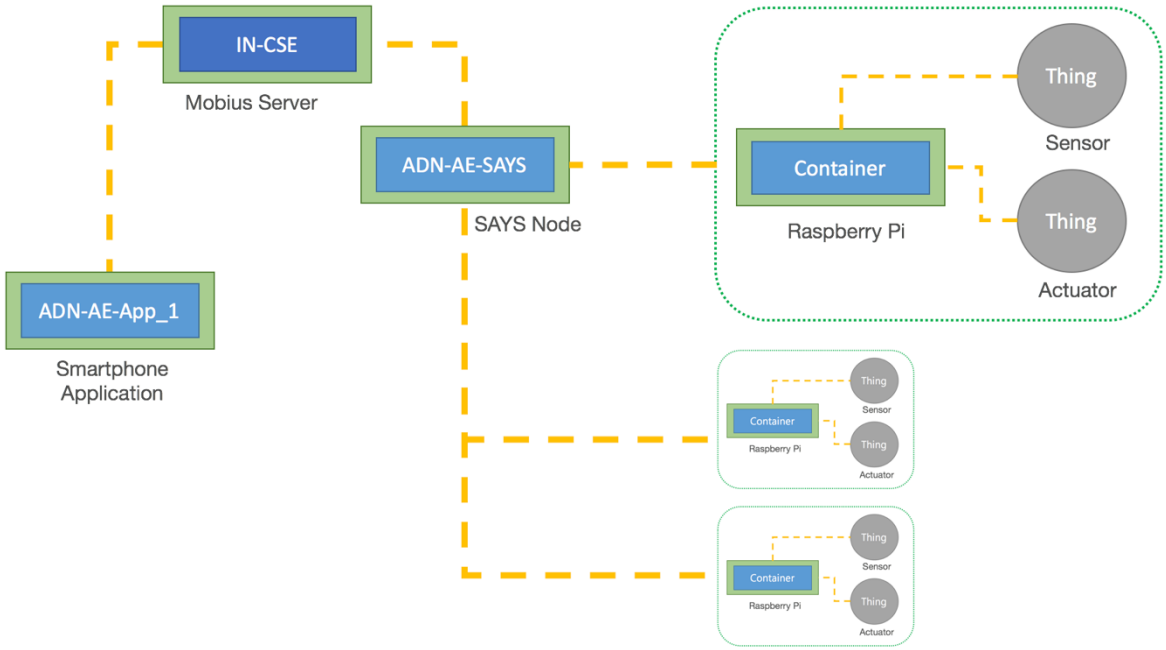


Fig. 2 oneM2M functional architecture of smoke monitoring use case

3.3 Resource Structure

The final resource structure is as shown in figure 3. We presumed that our service will not run on an independent server just for our service, but on a common platform for various other IoT services. For this reason, we created a separate AE node named ADN-AE-SAYS. This would make management and addition of IoT services easier on the Mobius server.

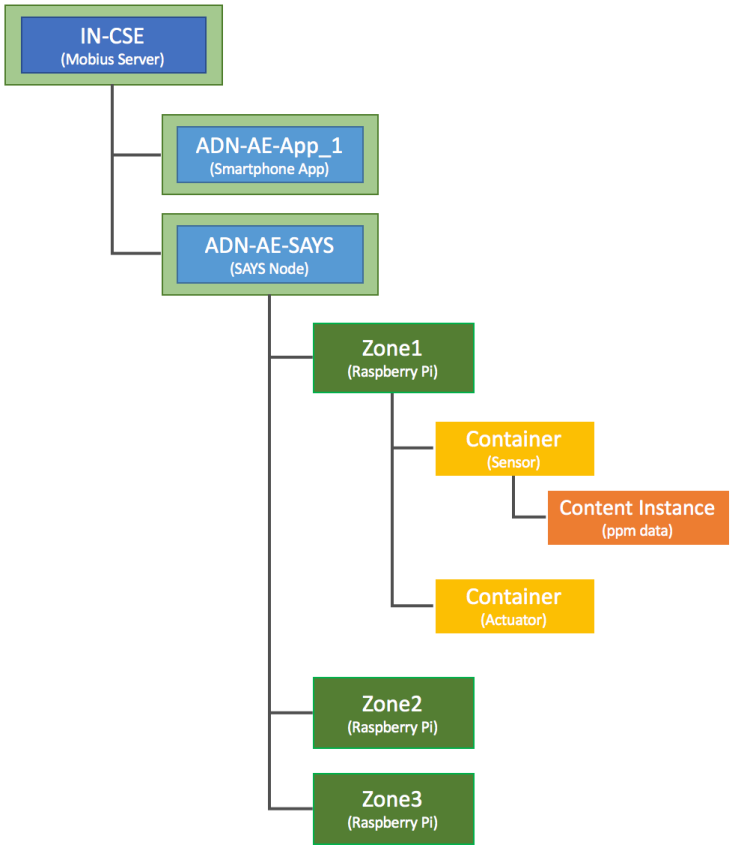


Fig. 3 Resource structure

4 Procedures

4.1 Introduction

Through our service, the user can use the smartphone application to monitor the sensor data. To realize these functions, the deployment of the oneM2M standard in the present use case requires procedures that are classified as follows:

- **Registration:** registration of AE.
- **Initial Resource Creation:** contains resource creation.
- **Sensor data POST:** post sensor data to the Mobius server.
- **Discovery and retrieval:** discovery and retrieval of sensor data through a smartphone application.

4.2 Call Flows

4.2.1 Registration

The first step is registration of Raspberry Pi and smartphone application. First, Raspberry Pi will register with oneM2M service platform. The smartphone applications can register with the oneM2M service platform anytime as needed.

Call flows regarding the registration phase depicted in figure 4 are ordered as follows:

1. Raspberry Pi (Container) registers into oneM2M service platform(IN-CSE)
2. Smartphone application (ADN-AE) registers into oneM2M service platform(IN-CSE)

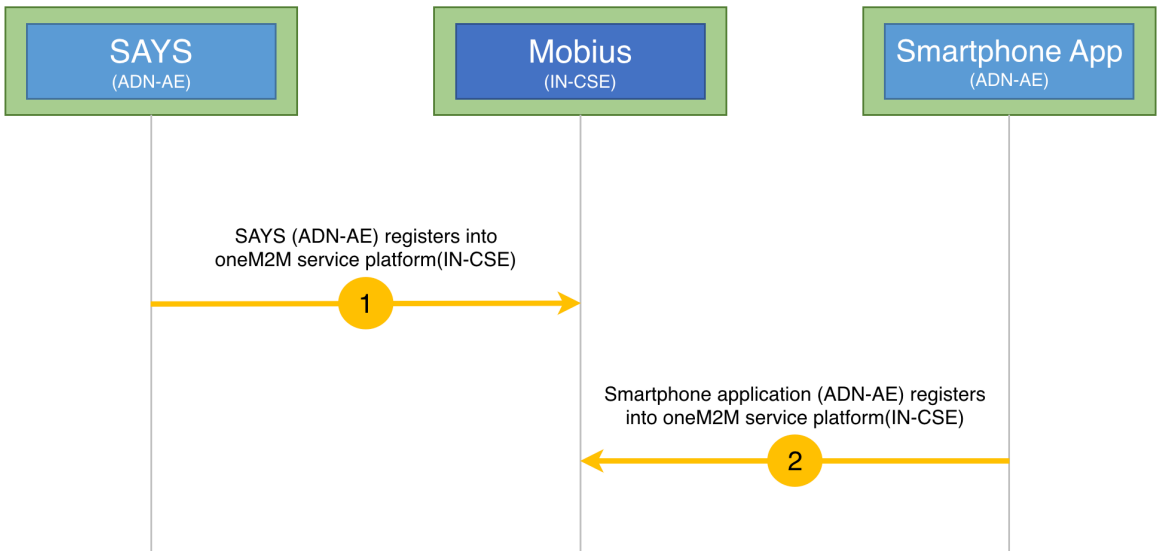


Fig. 4 Registration call flow

4.2.2 Initial Resource Creation

After registration, it is necessary to create container resource for each Raspberry Pi device. Call flows regarding the initial resource creation phase depicted in figure 5 are ordered as follows:

1. A container resource is created in the oneM2M service platform(IN-CSE) for each Raspberry Pi device under ADN-AE-SAYS.
2. Two sub container resources(sensor and actuator) are created under each Raspberry Pi container in the oneM2M service platform(IN-CSE).

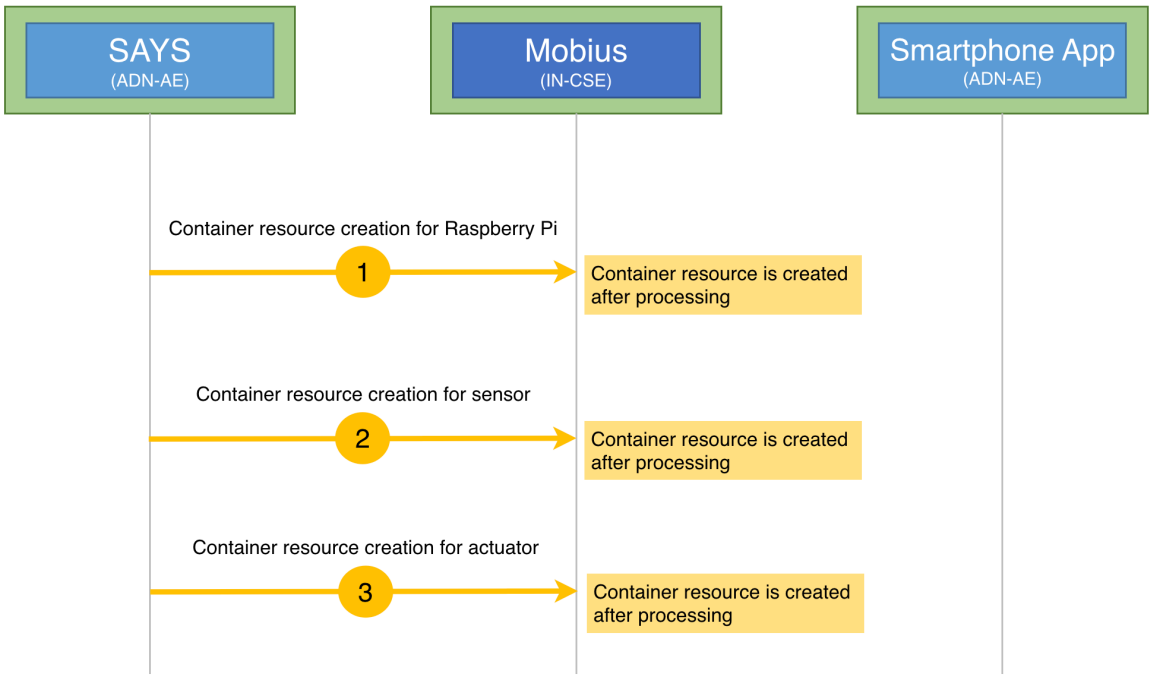


Fig. 5 Initial Resource Creation call flow

4.2.3 Sensor data post

The sensor has to send ppm data to the server, so that it can be accessed by the smartphone application. So, the Raspberry Pi container sends sensor data to oneM2M service platform(IN-CSE).

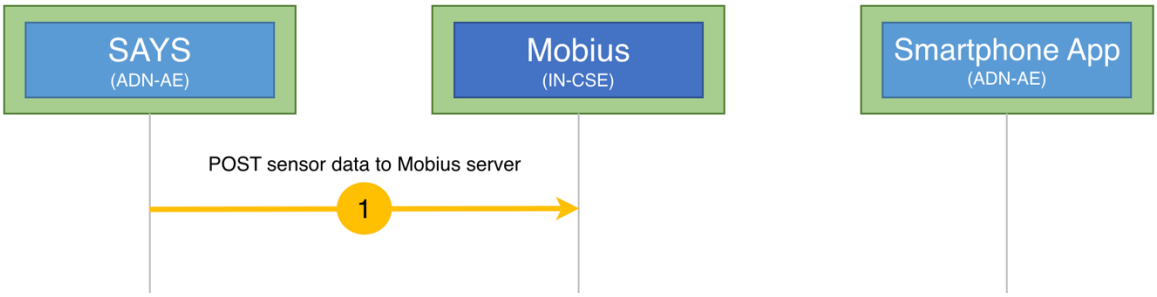


Fig. 6 Sensor data post call flow

4.2.4 Discovery and retrieval

The smartphone application needs ppm data so that it can show the changes in smoke concentration. For this, the application retrieves data every 5 seconds from the server. Call flows regarding the discovery and retrieval depicted in figure 7 are ordered as follows:

1. The smartphone application(ADN-AE) sends a request to oneM2M service platform(IN-CSE).

2. oneM2M service platform(IN-CSE) sends the ppm as a response to the Smartphone App(ADN-AE).

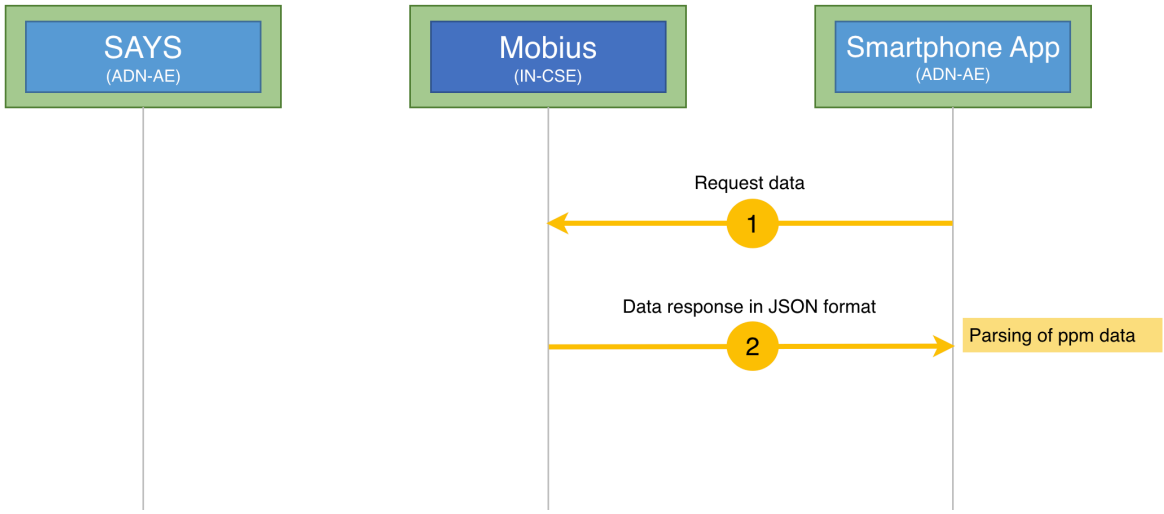


Fig. 7 Discovery and retrieval call flow

5 Implementation

5.1 Development Environment

Mobius server OS	Ubuntu 17.04
Node device OS	Raspbian
Smartphone application development tool	Unity
Server/Node programming	Pycharm, vim
Source code management	Git, Github

5.2 Mobius Server

5.2.1 Server installation

We installed 'Ubuntu 17.04' OS and dependencies(Node JS, Mysql, Mosquitto) on our server which has public IP address to use Mobius platform. And we pulled Mobius project from IoTKETI's Github repository (https://github.com/IoTKETI/Mobius) and configured the settings for the server as explained in the #Configuration column (https://github.com/IoTKETI/Mobius#configuration).

5.2.2 Mobius platform testing and initial registration

To check proper functioning of the Mobius server, Postman was used to test APIs of the server. Postman is a Chrome browser extension which can test APIs of servers. We downloaded Mobius_API_Release2.postman_collection.json file from IoTKETI's Github repository (https://github.com/IoTKETI/oneM2M-API-Testing) to test out Mobius platform.

After verifying the proper functioning of the Mobius server, we built a resource structure for our service by registration of entities using Postman.

Example (AE resource creation) :

```
<?xml version="1.0" encoding="UTF-8"?>
<m2m:ae mlns:m2m="http://www.onem2m.org/xml/protocols"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" rn="SaysNode">
<api>0.2.481.2.0001.001.000111</api>
<lbl>key1 key2</lbl>
<rr>true</rr>
</m2m:ae>
```

5.3 Node Device Setup

5.3.1 Components

To make a 'Smoke Detector', we decided to use the following components.

- Raspberry Pi 3 (Model B+)
- MQ-2 (Methane, Butane, LPG, smoke) sensor
- MH-FMD Buzzer
- MCP3002 Analog to Digital Converter

5.3.2 Raspberry Pi setup

We installed Raspbian OS on our Raspberry Pi3. Raspbian is one of many Linux operation systems, and is based on Ubuntu OS. So, we can use Python, Node JS, Apt-get. We used Python to interact with the Sensor and the Buzzer. We downloaded Raspbian Image from raspberrypi.org (https://www.raspberrypi.org/downloads/raspbian/) and copied image to SD card.

Copying an image to sd card:

sudo dd bs=1m if=./raspbian-jessie.img of=/dev/rdisk3

Then, we insert SD card to Raspberry Pi 3 and power up.

5.3.3 Wire connections

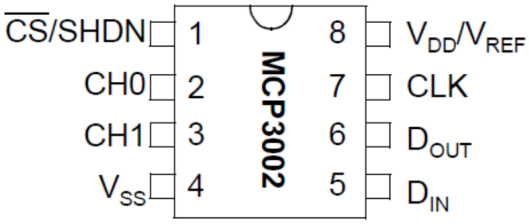
Raspberry Pi GPIO Pin information, MQ-2 Sensor datasheet, MCP3002 datasheet, and MH-FMD Buzzer datasheet were referenced for success connection of all devices.

First, before connecting the devices, connect a breadboard to the VSS and GND pin of Raspberry Pi. VSS and VDD/GND for all devices will be connected to the breadboard.

MCP3002 connection

- Pin 7 of MCP3002, which is the CLK pin is connected to GPIO 11 which is the CLK pin of Raspberry Pi.
- Connect MOSI pin of Raspberry Pi to DIN of MCP3002, and MISO pin of Raspberry Pi to DOUT of MCP3002.

Refer to the following schematic diagram:

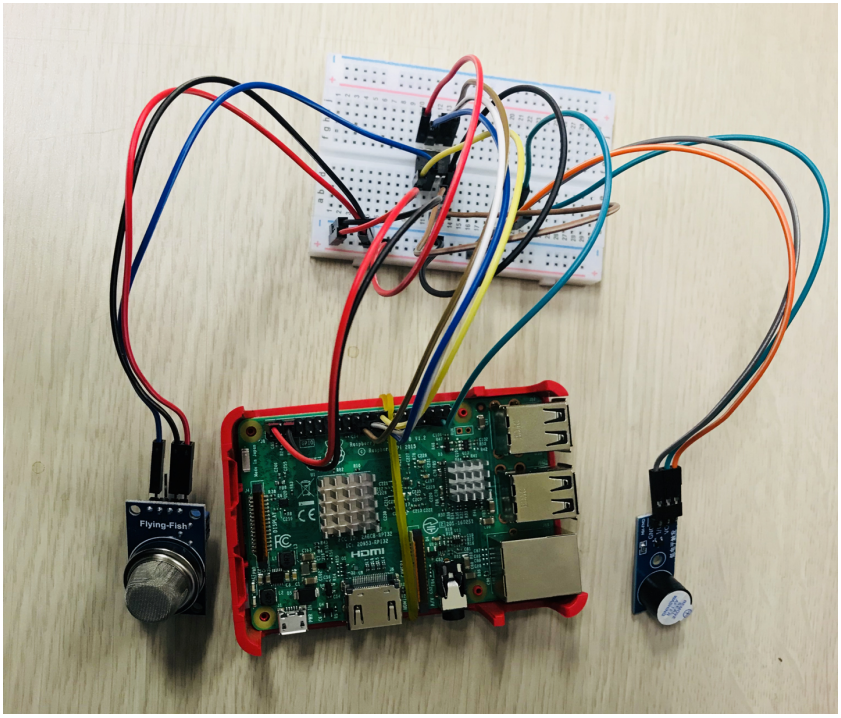


Sensor connection

- Connect the VCC and GND pins to the breadboard.
- The I/O signal pin is connected to pin 2 of the MCP3002. This is converted into digital signal and sent to Raspberry Pi by connecting pin 1 of MCP3002 to GPIO 8 of Raspberry Pi.

Buzzer connection

- Connect the VCC and GND pins to the breadboard.
- The I/O signal pin is connected to GPIO 19 of Raspberry Pi.



5.4 Node Device Programming

Sources can be found here: https://github.com/reum/IoT_SAYS/tree/master/RaspberryPi

5.4.1 Preparation

Before connecting the Sensor and Buzzer, the GPIO feature on Raspbian OS should be enabled. To enable it, Press the Raspberry Pi icon on top bar. And press the Preference and Raspberry Pi Configuration. Then Press the 'Interface' on new Windows, Check the 'enable check box' of SPI label. And we should install the requests library for Python. Requests is a library for interaction with server in HTTP(S). Command for installation : *sudo pip install requests*

5.4.2 Sensor programming

To receive the data from sensor(MQ-2) and analog converter(MCP3002), we used an example code from internet.

https://github.com/hackabletype/37-SensorsCode/blob/master/raspberrypi/ldr/botbook_mcp3002.py

This is an example code for receiving the data from sensor and analog converter by using botbook_mcp3002 module.

```
import botbook_mcp3002 as mcp #
def getSmokeLevel():
  smokeLevel= mcp.readAnalog()
  return smokeLevel
```

5.4.3 Buzzer programming

We enable the buzzer to detect the smoke or gas while Raspberry Pi is running. We found an example code for active buzzer here: https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-10-buzzer-module-sensor-kit-v2-0-for-b-plus.html

We wrote our buzzer enabling code by referring to the above example code. When we run the script, we enter the amount of time for the buzzer to stay active, and the buzzer will be enabled for that much of time.

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
import sys
```

```
BuzzerPin = 19 # Raspberry Pi Pin 17-GPIO 17

def setup():
    GPIO.setmode(GPIO.BCM) # Set GPIO Pin As Numbering
    GPIO.setup(BuzzerPin, GPIO.OUT)
#GPIO.output(BuzzerPin, GPIO.HIGH)

def on():
    GPIO.output(BuzzerPin, GPIO.LOW)
#GPIO.output(BuzzerPin, True)

def off():
    GPIO.output(BuzzerPin, GPIO.HIGH)
#GPIO.output(BuzzerPin, False)

def beep(Delay):
    time.sleep(Delay)
    GPIO.cleanup() # Release resource
    time.sleep(Delay)

if __name__ == '__main__': # Program start from here
    if len(sys.argv) != 2:
        exit()

        delay = sys.argv[1]
        delay = float(delay)

        setup()
    try:
        beep(delay);
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy()
will be executed.
    destroy()
```

5.4.4 Client programming

We should follow the oneM2M standard to send the data of sensor to the server. So, we referred the Postman and Mobius-API-Configuration.json to get header and body structure. Finally, we got the information to generate 'contentinstance' in container on platform.

- Header

```
X-M2M-RI: 12345
X-M2M-Origin: /0.2.481.1.21160310105204806
Content-Type: application/vnd.onem2m-res+xml; ty=4
Cache-Control: no-cache
```

- Body

```
<?xml version="1.0" encoding="UTF-8"?>
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<con> 0 ppm</con>
</m2m:cin>
```

A new ContentsInstace can be created by utilizing the above information and sending a POST method to the url of a sensor (ex. http://says.zeroday.me:7579/Mobius/SaysNode/Zone1/Sensor)

5.4.5 Smoke Detector programming

By combining the results of 5.3.1, 5.3.2, 5.3.3, and 5.3.4, we completed the development of out smoke detector.

- botbook_mcp3002.py : Provides function for interactacting with sensor.
- buzzer.py : Activate buzzer for N seconds
- buzzer_off.py : Deactivate buzzer
- smoke_detector.py : Smoke detector
- smoke_detector_tester.py : Send dummy ppm data to server
  - Raspberry Pi checks for smoke every 2 seconds
  - If the smoke detection level exceeds 150 ppm, the buzzer is enabled for 1 second. (The interval and condition for alarm can be modified!)

```
import time
import botbook_mcp3002 as mcp #
import os
import requests

INTERVAL = 2
TARGET_CT = "Zone1"

def header_parser(header_str):
    # Convert HEAD_str to dict.
    headers = {}
    header_str_lst = header_str.split("\n")
    for head in header_str_lst:
        if ":" in head:
            tmp = head.split(": ")
            key = tmp[0]
            value = "".join(tmp[1:])
            headers[key] = value
    return headers

# Send sensor's data to Mobius server
def send_data(target_ct ,ppm):
    header_str = """X-M2M-RI: 12345
        X-M2M-Origin: /0.2.481.1.21160310105204806
        Content-Type: application/vnd.onem2m-res+xml; ty=4
        Cache-Control: no-cache"""

    # Building payload to send server.
    POST = """<?xml version="1.0" encoding="UTF-8"?>
        <m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <con>"""+str(ppm)+""" ppm</con>
            </m2m:cin>"""

        # Parse header to interact with Mobius server
        headers = header_parser(header_str)
        try:
        # Send seonsor's data to Mobius server
```

```
        req =
requests.post("http://says.zeroday.me:7579/Mobius/SaysNode/Zone1/Sensor", POST,
headers=headers)
        if req.status_code == "200":
            # If success,
            return True
        else:
        return False
        except:
            print("[-] An error occured. Check your internet connection..")

def getSmokeLevel():
    smokeLevel= mcp.readAnalog()
    return smokeLevel

def main():
    while True:
        time.sleep(INTERVAL)
        smokeLevel = getSmokeLevel()
        print("[ ] Current smoke level is %i " % smokeLevel)
    send_data(TARGET_CT, smokeLevel)
        if smokeLevel > 150:
            print("[!] Smoke detected")
            # Active buzzer for n seconds.
            os.system("python /home/pi/Desktop/buzzer.py 1")

main()
```

5.4.6 Smoke detector setting

To initialize the sensor and execute the script automatically, we modified the boot script on Raspbian OS.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

service ssh restart
screen -dm python /home/pi/Desktop/buzzer_off.py &
sleep 1
screen -dm python /home/pi/Desktop/smoke_detector.py &
exit 0
```

This script has three commands.

1. Run SSH server
2. Deactivate the Buzzer
   - When we power up the Raspberry Pi, the buzzer sounds an alarm. So, we made a script for disabling it when the Raspberry Pi is booted.
3. Execute Smoke Detector

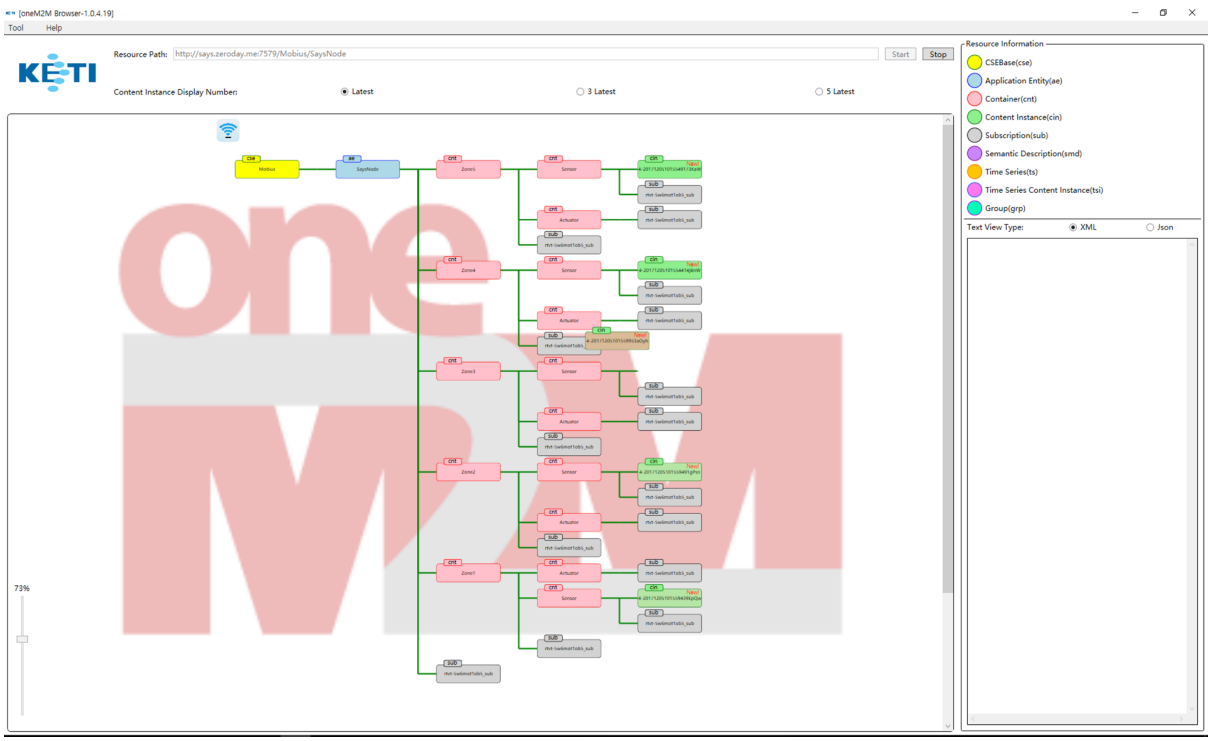
If we configure these three scripts to run right after booting, we can manage the Raspberry Pi anywhere and anytime. Also, the script will automatically run even after reboot.

5.5 oneM2M Browser

KETI distributes a program for monitoring the resource tree of a Mobius server in real time. To use this program, we should download the oneM2MBrowser project from IoTKETI's repository. https://github.com/IoTKETI/oneM2MBrowser

- Download the zip file: https://github.com/IoTKETI/oneM2MBrowser/blob/master/Installer/Installer_v1.0.4.19.zip
- Unzip and execute the installer
- Done!

We used the oneM2M Browser to easily check each entity in the resource structure. Run oneM2M Browser and enter the url of the entity you want to see. In our case, we can see the following screen on accessing this address: http://says.zeroday.me:7579/Mobius/SaysNode



5.6 Smartphone Application

Source code available here : https://github.com/reum/IoT_SAYS-MobileApp

5.6.1 UI design

Title Scene

The title scene which appears when the app is run

Campus map

The campus map is divided into 14 sections. We activated six sections - Gwanggaeto Building, Chapel, Yulgok-gwan, Yongduk-gwan, Student Union Building, and Library. Remaining sections can be activated later on.

The application gets ppm data from the server every 5 seconds, and immediately responds by displaying color on the campus map. The data manager stores the ppm data from the server, and processes the sensor data.

Viewing data for individual sections

By touching the menu icon, we can see the list of activated sections. Touching each button will move to each Section Scene, and display information for that individual section.

5.6.2 Programming

Data request from Mobius server

WWWForm Request was used to get data from the server. GetData() function requests for data from the Mobius server every 5 seconds, and receives it in JSON format. Then, the ppm data is parsed and stored. This stored data is constantly updated within the application.

```
// GET Request
public WWW GetData(string url)
{
    Dictionary<string, string> headers = new Dictionary<string, string>();
    headers.Add("Accept", "application/json");
    headers.Add("X-M2M-RI", "12345");
    headers.Add("X-M2M-Origin", "0.2.481.1.21160310105204806");
    headers.Add("Content-Type", "application/vnd.onem2m-res+json; ty=4");
    headers.Add("Cache-Control", "no-cache");
    Debug.Log("URL : " + Url);
```

```
        WWW www = new WWW(url, null, headers);
        StartCoroutine("dataEnumerator", www);
        return www;

    }
    IEnumerator dataEnumerator(WWW www)
    {
        yield return www;

        if (www.error == null)
        {
            Debug.Log("Data Submitted");
            Debug.Log("www Result: " + www.text);
            ProcessPlayer(www.text);
        }
        else
        {
            Debug.Log(www.error);
        }
    }
```

ppm concentration display

Depending on the value of ppm, the color is set to red, yellow, or green. The color will be red if the value exceeds 150, yellow if it is in the range of 70 and 150, and green if is equal or below 70.

```
    void CheckPPM()
    {
        if (ras_1_ppm >= 150)
        {
            render.color = new Color32(220, 10, 10, 150);
        }
        else if (ras_1_ppm < 150 && ras_1_ppm >= 70)
        {
            render.color = new Color32(200, 220, 10, 150);
        }
        else
        {
            render.color = new Color32(90, 220, 40, 150);
        }
    }
```

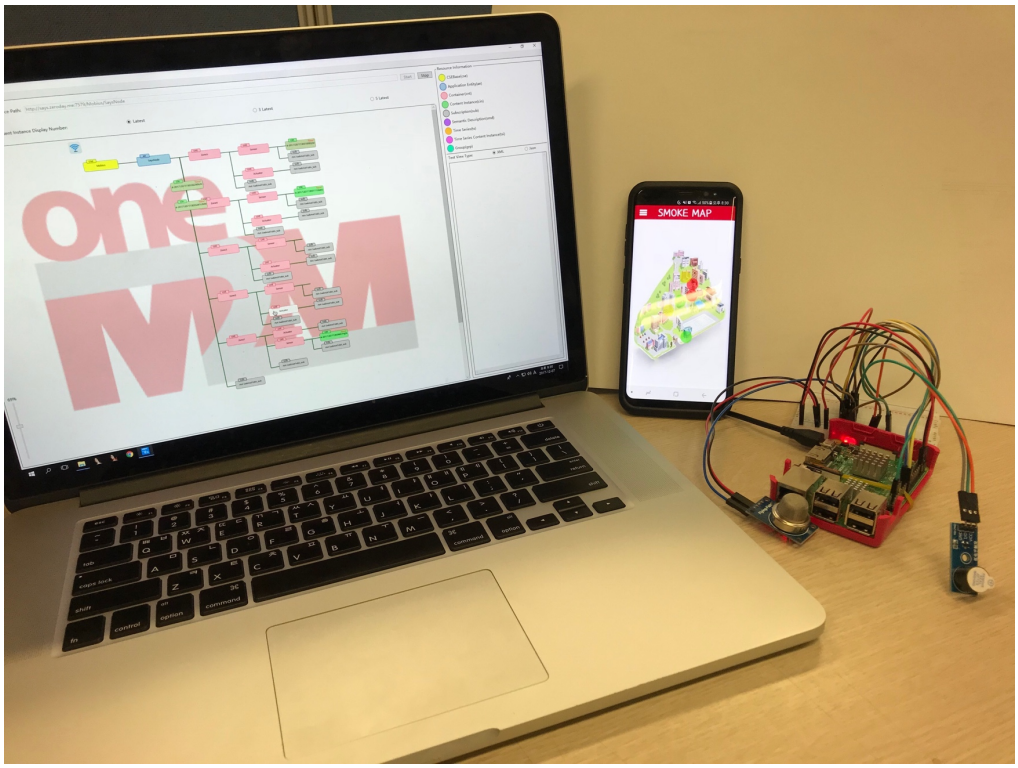
6 Final Output

The smoke map service functioned successfully, as described as follows:

- The resource tree along with the changes in content instance values were constantly being updated, which could be checked through oneM2M browser.
- The smartphone application retrieved data from the Mobius server and showed the changes in ppm concentration by changing color on the campus map.
- When the ppm value exceeded 150, the buzzer connected to the Raspberry Pi sound an alarm to alert the smoker.

Our demo can be seen here:

- OneM2M browser simulation
- Smartphone application demo
- Smoke detector demo
- Full simulation demo



7 Future Work

Ideas for further improvement are as follows:

- Make more Raspberry Pi nodes around the campus to get a more accurate smoke map.
- Implement function to provide the most smoke-free route in the campus.
- Implement function to export collected data as csv, excel, etc so that it can be used for various purposes such as non-smoking campaigns.

~ End of document ~

Project by Team SAYS:

Areum Lee,
Department of Computer and Information Security @ Sejong University

Seongwon Hyun,
Department of Computer and Information Security @ Sejong University

Seok Yoon,
Department of Physics @ Sejong University