IoTware Framework
설명서

- 개정이력 -

버전	개정일자	개정내역	작성자
1.00	2020.10.22.	V1.0 작성	

목차

1. 개요 ····· 7
1.1. 목적 ····· 7
1.2. 범위 ····· 7
1.3. 용어 및 약어 ····· 7
1.3.1. 용어 ····· 7
1.3.2. 약어 ····· 8
2. 마이크로서비스 구조의 IoTWare 서비스 운용 프레임워크 개요 ····· 10
2.1. 초소형 IoTWare 마이크로서비스 구조 ····· 11
2.2. 태스크 ····· 11
2.3. 마이크로서비스 ····· 13
3. IoTWare 서비스 운용 프레임워크 구조 설계 ····· 14
3.1. IoTWare 서비스 인프라 연동 프레임워크 구조 설계 ····· 14
3.1.1. 서비스 인프라 연동 프레임워크 개요 ····· 14
3.1.2. 서비스 인프라 연동 프레임워크 주요 마이크로서비스 기능 ····· 14
3.1.3. 서비스 인프라 연동 프레임워크 구조 및 인터페이스 ····· 17
3.2. 서비스 미션 수행 프레임워크 구조 설계 ····· 22
3.2.1. 서비스 미션 수행 프레임워크 개요 ····· 22
3.2.2. 서비스 미션 수행 프레임워크 주요 마이크로서비스 기능 ····· 22
3.2.3. 서비스 미션 수행 프레임워크 구조 및 인터페이스 ····· 23
3.3. 동적 자율구성 기반 분산협업 마이크로서비스 구조 설계 ····· 26
3.3.1. 동적 자율구성 기반 분산협업 마이크로서비스 개요 ····· 26
3.3.2. 동적 자율구성 기반 분산협업 마이크로서비스 주요 기능 ····· 27
3.3.3. 동적 자율구성 기반 분산협업 마이크로서비스 구조 및 인터페이스 ····· 27
3.4. 사용자 마이크로서비스 구조 설계 ····· 28
3.4.1. 사용자 마이크로서비스 개요 ····· 28

3.4.2. 사용자 마이크로서비스 주요기능 ····· 29
3.4.3. 사용자 마이크로서비스 구조 및 인터페이스 ····· 29
3.5. IDE 인터페이스 구조 설계 ····· 30
3.5.1. 개요 ····· 30
3.5.2. IoT 디바이스 개발환경 플랫폼(IDE) 인터페이스 구조 ····· 31
3.6. IoTware 자원운용부 구조 및 기능 ····· 32
3.6.1. 전력 프로파일 관리 ····· 32
3.6.2. 최적상태 자율제어 및 전력 상태관리 ····· 34
4. OAL 구조 설명 ····· 38
4.1. 전체 Folder Tree ····· 38
4.2. OS 지원 ····· 39
4.2.1. 총 7종의 OS 지원 ····· 39
4.3. 네트워크 및 센서 드라이버 지원 ····· 54
4.3.1. 추상화 개요 ····· 54
4.3.2. 디바이스 등록 ····· 54
4.3.3. 네트웍 디바이스 드라이버 API 추상화 ····· 54
4.3.4. 센서 디바이스 드라이버 API 추상화 ····· 66
4.3.5. 인터페이스 추상화 API ····· 72

표 목차

[표 1] 태스크의 구성 요소 ····· 14
[표 2] 태스크 종류 ····· 15
[표 3] 인증 마이크로서비스 인터페이스 ····· 19
[표 4] 등록 마이크로서비스 인터페이스 ····· 20
[표 5] 태스크 오케스트레이션 연동 마이크로서비스 인터페이스 ····· 20
[표 6] 태스크 오프로딩 연동 마이크로서비스 인터페이스 ····· 21
[표 7] 초소형 IoTWare 플랫폼 연동 게이트웨이 인터페이스 ····· 22
[표 8] 기존 플랫폼 연동 게이트웨이 인터페이스 ····· 23
[표 9] 임무지향적 태스크 분해 재조합 마이크로서비스 주요 인터페이스 ····· 25
[표 10] 태스크 상태 관리 마이크로서비스 주요 인터페이스 ····· 26
[표 11] Power profile 생성 마이크로서비스 주요 인터페이스 ····· 27
[표 12] 동적 자율구성 기반 분산협업 마이크로서비스 주요 인터페이스 ····· 29
[표 13] 사용자 마이크로서비스 인터페이스 ····· 31
[표 14] 단위 태스크 인터페이스 ····· 32

그림 목차

[그림 1] 마이크로서비스 구조의 IoTWare 서비스 운용 프레임워크 구조 ····· 12
[그림 2] 초소형 IoTWare 마이크로서비스 구조 ····· 13
[그림 3] 태스크 구조 ····· 14
[그림 4] 마이크로서비스 구조 ····· 15
[그림 5] 인증 및 등록 마이크로서비스 ····· 16
[그림 6] 태스크 오케스트레이션 및 오프로딩 연동 마이크로서비스 ····· 17
[그림 7] 초소형 IoTWare 플랫폼 연동 게이트웨이 ····· 18
[그림 8] 표준 플랫폼 연동 게이트웨이 ····· 18
[그림 9] 인증 마이크로서비스 구조 ····· 19
[그림 10] 등록 마이크로서비스 구조 ····· 19
[그림 11] 태스크 오케스트레이션 연동 마이크로서비스 구조 ····· 20
[그림 12] 태스크 오프로딩 연동 마이크로서비스 구조 ····· 21
[그림 13] 초소형 IoTWare 플랫폼 연동 게이트웨이 구조 ····· 22
[그림 14] 표준 플랫폼 연동 게이트웨이 구조 ····· 22
[그림 15] 서비스 미션 수행 프레임워크 및 인터페이스 ····· 23
[그림 16] 임무지향적 태스크 분해 재조합 마이크로서비스 구조 ····· 25
[그림 17] 태스크 상태 관리 마이크로서비스 구조 ····· 25
[그림 18] Power profile 생성 마이크로서비스 구조 ····· 26
[그림 19] 서비스 동적 자율구성 기반 분산협업 구성도 ····· 27
[그림 20] 동적 자율구성 기반 분산협업 마이크로서비스 ····· 28
[그림 21] 동적 자율구성 기반 분산협업 마이크로서비스 구조 ····· 29
[그림 22] 사용자 마이크로서비스 ····· 30
[그림 23] 사용자 마이크로서비스 구조 ····· 31
[그림 24] 개발환경 플랫폼의 단위 태스크 인터페이스 구조 ····· 32
[그림 25] IDE에서 수행되는 코어프레임워크 재구성 절차 ····· 34

1. 개요

1.1. 목적

본 문서는 경량형 운영체제(OS), 센서·통신 등 펌웨어 롬(read only memory/ROM)에 저장되어 있는 소프트웨어 프로그램, 자원관리 및 저전력 관리 모듈 등을 제공하여, 초보 개발자도 쉽게 IoT 관련 과제를 통한 마이크로서비스를 만들고 이를 연결해 프로그램 완성이 가능할 수 있는 IoTware Framework 개발자 가이드 제공을 목적으로 한다.

1.2. 범위

본 문서에서는 IoT 서비스 인프라 연동 및 IoT 서비스 미션 수행 프레임워크와 자원운용 프레임워크의 구조에 대해 설명한다.

1.3. 용어 및 약어

1.3.1. 용어

번호	용어	설명
1	동적수행	디바이스내 수행되는 태스크를 runtime 동안 실행 또는 중지하는 기능
2	동적재구성	하나의 임무를 수행하는 마이크로서비스는 여러 개의 태스크들로 구성된다. 마이크로서비스내 주어진 임무의 변경을 위해 runtime 동안 태스크를 재구성하는 기능
3	마이크로서비스	애플리케이션을 느슨히 결합된 서비스의 모임으로 구조화하는 서비스 지향 아키텍처(SOA) 스타일의 일종인 소프트웨어 개발 기법
4	분산협업	네트워크 내에 있는 디바이스들이 하나의 서비스 또는 마이크로서비스를 수행하기 위해 태스크를 분장하여 수행하는 것
5	오프로딩	태스크 수행 주체를 디바이스에서 엣지 또는 클라우드로 변경하는 것으로, 급증하는 데이터 트래픽을 다른 네트워크로 분산하는 효과가 있음
6	엣지 컴퓨팅	중앙 서버가 모든 데이터를 처리하는 클라우드 컴퓨팅과 달리 네트워크 가장자리에서 데이터를 처리한다는 뜻이다. 사물인터넷(IoT) 기기가 본격적으로 보급되면서 데이터 량이 폭증했고, 이 때문에 클라우드 컴퓨팅이 한계에 부딪히게 됐는데, 이를 보완하기 위해 에지 컴퓨팅 기술이 개발됐다. 즉, 모든 데이터를 클라우드로 보내서 분석하는 대신, 중요한 데이터를 실시간으로 처리하기 위한 기술
7	제로 터치 매니지먼트	어떤 목표를 수행하는데 있어, 사람의 도움 없이 디바이스 인증 및 시스템 구성을 수행하는 것
8	클라우드	데이터를 인터넷과 연결된 중앙컴퓨터에 저장해서 인터넷에 접속하기만 하면 언제 어디서든 데이터를 이용할 수 있는 것

번호	용어	설명
9	태스크	운영체계가 제어하는 프로그램의 기본 단위를 말하는데, 운영체계를 설계할 때, 태스크를 어떻게 정의하느냐에 따라 단위 프로그램이란 전체 프로그램이 될 수도 있고, 또는 계속되는 프로그램의 호출이 될 수도 있음
10	태스크 오케스트레이션	비즈니스 프로세스 흐름에 따라 요구되는 태스크(마이크로서비스)를 가장 효율적으로 수행할 수 있도록 IoT 디바이스·엣지·포그·클라우드에 자율할당하고 엣지·포그에서 취합 및 실행 관리하는 기술
11	프레임워크	소프트웨어 어플리케이션이나 솔루션의 개발을 수월하게 하기 위해 소프트웨어의 구체적 기능들에 해당하는 부분의 설계와 구현을 재사용 가능하도록 협업화된 형태로 제공하는 소프트웨어 환경
12	플랫폼	여러 가지 기능들을 제공해주는 공통 실행환경

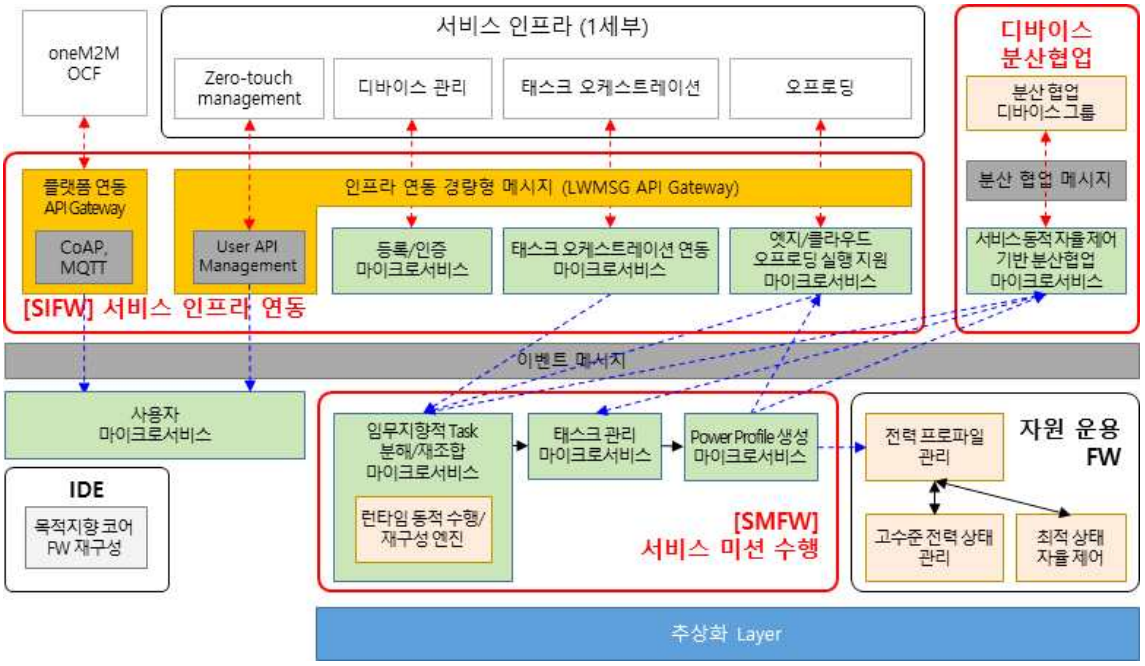
1.3.2. 약어

번호	약어	설명
1	API	Application Programming Interface
2	CoAP	Constrained Application Protocol
3	DCMI	Distributed Cooperation Message Interface
4	FWP	Framework Performance
5	IoT	Internet of Things
6	IPC	Inter-Process Communication
7	LWM2M	Lightweight M2M
8	MQTT	Message Queueing Telemetry Transport
9	OCF	Open Connectivity Foundation
10	PoC	Proof of Concept
11	REST	Representational State Transfer
12	SFR	System Function Requirements
13	SIR	System Interface Requirements
14	SNR	System Non-function Requirements
15	SRVI	Micro IoT platform Service Interface
16	STDI	Standard IoT platform Interface
17	UA	User Assumptions
18	UC	User Constraints

19	UFR	User Function Requirements
20	UNR	User Non-function Requirements

2. 마이크로서비스 구조의 IoTWare 서비스 운용 프레임워크 개요

초소형 IoT 디바이스를 위한 IoTWare 프레임워크는 표준 플랫폼(oneM2M, OCF)은 물론 서비스 인프라와 연동할 수 있는 서비스 인프라 연동 부분과 클라우드 또는 엣지에서 내려오는 미션을 수행하는 서비스 미션 수행 부분으로 나누어진다. 서비스 인프라 연동 부분에서는 마이크로서비스 제어 메시지를 이용하여 인프라에서 내려오는 명령(디바이스 관리, 태스크 오케스트레이션, 태스크 오프로딩 등)을 해석하여 인프라와 서비스 미션 수행 부분을 연동시킨다. 서비스 미션 수행 부분에서는 서비스의 명령에 따라 임무를 수행한다.



[그림 1] 마이크로서비스 구조의 IoTWare 서비스 운용 프레임워크 구조

IoTWare 서비스 운용 프레임워크는 다수의 태스크(Task)로 구성된 마이크로서비스(Microservice) 구조로 되어 있으며, 사용자 마이크로서비스는 임무에 따라 마이크로서비스내 태스크를 분해/재조합할 수 있다. 마이크로서비스 구조의 IoT 서비스 운용 프레임워크는 서비스 인프라 연동 프레임워크와 서비스 미션수행 프레임워크 2개의 프레임워크와 서비스 동적자율 구성을 수행하는 분산협업 마이크로서비스로 구성된다. 서비스 인프라 연동 프레임워크는 플랫폼 연동 API Gateway, 인프라 연동 마이크로서비스 제어 메시지 API Gateway와 4개의 마이크로서비스로 구성되며 다음과 같다.

- 등록 마이크로서비스
- 인증 마이크로서비스
- 태스크 오케스트레이션 연동 마이크로서비스
- 엣지/클라우드 오프로딩 실행지원 마이크로서비스

서비스 미션수행 프레임워크는 3개의 마이크로서비스로 구성되며 다음과 같다.

- 임무지향적 태스크 분해/재조합 마이크로서비스
- 사용자 태스크 관리 마이크로서비스
- Power profile 생성 마이크로서비스

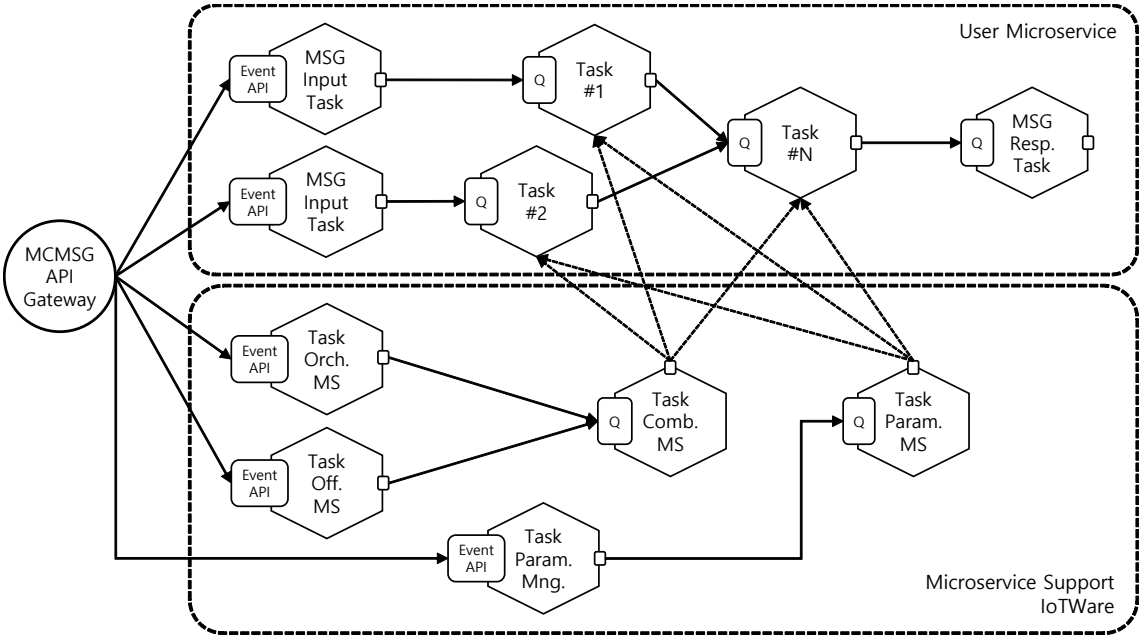
2.1절에서 2.3절은 본 구조설계서에서 사용된 태스크와 마이크로서비스에 대한 기본 개념 및 구조에 대해서 설명한다. 3장부터는 각 프레임워크 구조, 각 마이크로서비스와 각 태스크에 대한 내용, 주요기능, 구조 및 주요 인터페이스에 대해 설명한다.

2.1. 초소형 IoTWare 마이크로서비스 구조

마이크로서비스(Microservice)란 아키텍처이자 소프트웨어 작성을 위한 하나의 접근 방식으로, 애플리케이션을 상호 독립적인 최소 규모 구성 요소로 분할한다. 모든 요소를 하나의 애플리케이션에 구축하는 전통적인 모놀리식(Monolithic) 접근 방식 대신 마이크로서비스에서는 모든 요소가 분리되고 연동되어 모놀리식 애플리케이션과 동일한 역할을 완수한다. 이러한 각각의 구성 요소 또는 프로세스가 마이크로서비스이다. 소프트웨어 개발에 대한 이러한 접근 방식은 세분화, 경량화 그리고 다수의 애플리케이션 간에 유사한 프로세스를 공유하는 기능을 중요시한다.

초소형 IoTWare 서비스 운용 프레임워크는 사용자의 응용 소프트웨어가 마이크로서비스 단위로 동작할 수 있도록 기능을 제공한다. 이를 위해 사용자 응용 애플리케이션의 최소 단위는 마이크로서비스이며 초소형 IoTWare 인프라와의 연동을 통한 태스크 오케스트레이션과 태스크 오프로딩 지원을 위해 마이크로서비스는 다시 태스크 단위로 구성된다.

마이크로서비스 사이의 IPC(Inter-Process Communication)를 위하여 메시지 큐(Message Queue) 방식을 사용한다. 메시지 큐를 이용한 마이크로서비스 사이의 데이터를 이벤트(Event)로 정의한다.



[그림 2] 초소형 IoTWare 마이크로서비스 구조

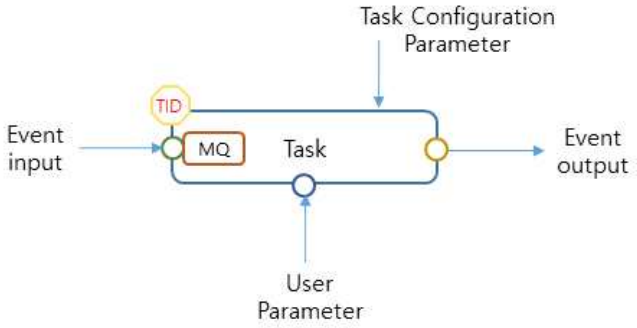
2.2. 태스크

태스크(Task)는 IoTWare 서비스 운용 프레임워크의 최소 단위이다. 태스크는 다른 태스크로부터 이벤트를 수신하는 이벤트 입력(Event input), 태스크의 결과를 다른 태스크로 송신하는 이벤트 출력(Event output) 및 태스크에 정의된 변수를 제어하는 사용자 전역 변수(User parameter), 태스크의 설정을 제어하는 태스크 구성 변수(Task configuration

parameter)로 구성된다.

[표 1] 태스크의 구성 요소

구성 요소	설명
Task ID	디바이스 전체 태스크 고유 ID
Event input	태스크 입력으로 태스크의 종류에 따라 결정. 입력 이벤트 메시지 큐와 연동
Input event message queue	태스크 입력으로부터 수신한 메시지 저장
User parameter	태스크에서 사용하는 사용자가 정의한 변수
Event output	태스크의 결과를 다른 태스크의 입력으로 출력
Task configuration parameter	태스크의 구성을 설정하는 변수. 태스크의 종류에 따라 정의



[그림 3] 태스크 구조

태스크의 종류에는 입력 태스크(Input task), 기능 태스크(function task), 출력 태스크(output task) 등이 있다.

입력 태스크는 시스템으로부터 이벤트를 수신하는 태스크로 마이크로서비스의 시작에 위치한다. 입력 태스크는 시스템으로부터 이벤트를 수신하기 때문에 다른 태스크로부터 연결할 이벤트 입력이 없다. 입력 이벤트의 형식은 입력 태스크의 종류에 따라 정의된다.

기능 태스크는 사용자 기능을 수행하는 태스크로 입력 태스크와 출력 태스크 또는 다른 기능 태스크 사이에 위치한다. 다른 태스크로부터의 입력 이벤트 수신과 다른 태스크로의 이벤트 출력이 모두 가능하다. 기능 태스크의 기능에 따라 마이크로서비스 외부로의 입출력을 수행하는 경우도 있다. 예를 들어 HTTP request 기능 태스크의 경우 HTTP 서버로 request를 요청하고 응답을 처리하는 기능 수행을 위해 프로토콜 송수신을 위한 마이크로서비스 외부 이벤트가 존재한다.

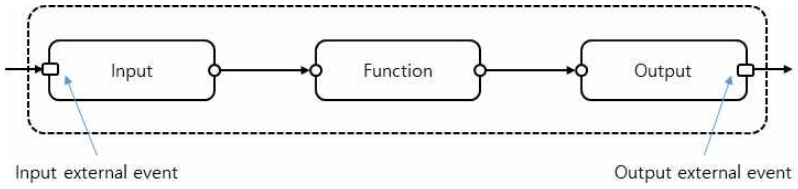
출력 태스크는 시스템으로 이벤트를 송신하는 태스크로 마이크로서비스의 끝에 위치한다. 출력 태스크는 시스템으로 이벤트를 송신하기 때문에 다른 태스크로 연결할 출력이 없다. 출력 이벤트의 형식은 출력 태스크의 종류에 따라 정의된다.

[표 2] 태스크 종류

구성 요소	설명	종류
Input task	시스템으로부터 이벤트 입력을 받는 태스크	Protocol (HTTP, CoAP, MQTT) receive External device (sensor) input
Function task	사용자의 기능이 개발되는 태스크	Protocol (HTTP, CoAP, MQTT) request Basic (delay, trigger, switch, change, range, split, join, sort) function
output task	태스크의 결과를 시스템으로 출력하는 태스크	Protocol (HTTP, CoAP, MQTT) send Basic output (debug, link) function

2.3. 마이크로서비스

마이크로서비스는 태스크의 조합으로 구성된다. 마이크로서비스에는 최소 1개 이상의 입력 태스크와 출력 태스크를 가진다.



[그림 4] 마이크로서비스 구조

입력 태스크의 입력 이벤트는 마이크로서비스의 입력 외부 이벤트(Input external event)가 되고 출력 태스크의 출력 이벤트는 출력 외부 이벤트(Output external event)가 된다. 마이크로서비스의 입출력 외부 이벤트는 태스크의 입출력 이벤트와 같은 원리로 동작하면 마이크로서비스 단위의 외부로부터의 입력과 외부로의 출력만을 의미한다.

3. IoTWare 서비스 운용 프레임워크 구조 설계

3.1. IoTWare 서비스 인프라 연동 프레임워크 구조 설계

3.1.1. 서비스 인프라 연동 프레임워크 개요

서비스 인프라 연동 프레임워크는 초소형 IoTWare 플랫폼과의 연동 기능을 제공하는 프레임워크이다. 초소형 IoTWare 플랫폼만 아니라 oneM2M, OCF, LWM2M과 같은 기존의 IoT 플랫폼과의 연동도 지원한다.

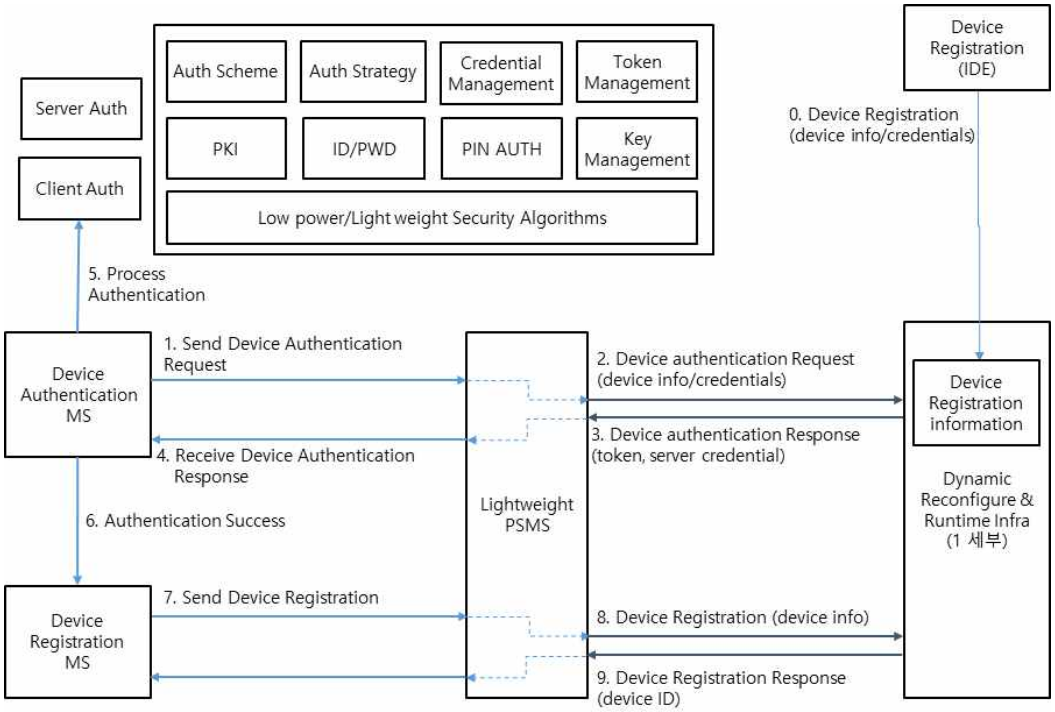
서비스 인프라 연동 프레임워크는 마이크로서비스 단위로 구성되어 있다. 주요 마이크로서비스에는 인증 마이크로서비스, 등록 마이크로서비스, 태스크 오케스트레이션 연동 마이크로서비스, 태스크 오프로딩 연동 마이크로서비스가 있다.

3.1.2. 서비스 인프라 연동 프레임워크 주요 마이크로서비스 기능

3.1.2.1. 인증 및 등록 마이크로서비스 기능

초소형 IoTWare 디바이스는 인증과 등록 과정을 통하여 IoTWare 서비스를 시작할 수 있다. 인증 과정을 통하여 IoTWare 서비스를 수행할 수 있음을 확인받고 등록 과정을 통하여 실질적인 서비스에 투입된다.

초소형 IoTWare 디바이스 개발자는 먼저 디바이스를 IoTWare 서비스에 디바이스를 등록하고 인증에 필요한 인증 정보를 획득한다. 획득한 인증 정보를 개발된 사용자 프로그램과 함께 디바이스에 설치한다. IoTWare 서비스에 디바이스가 투입되면 디바이스는 인증 정보를 바탕으로 IoTWare 서비스에 인증을 요청한다. 인증 마이크로서비스는 이러한 디바이스에 설치된 인증 정보를 바탕으로 IoTWare 서비스에 인증 과정을 수행하는 기능을 가진다. 인증 과정이 성공적으로 수행되면 등록 과정을 통하여 IoTWare 서비스에 투입된다.

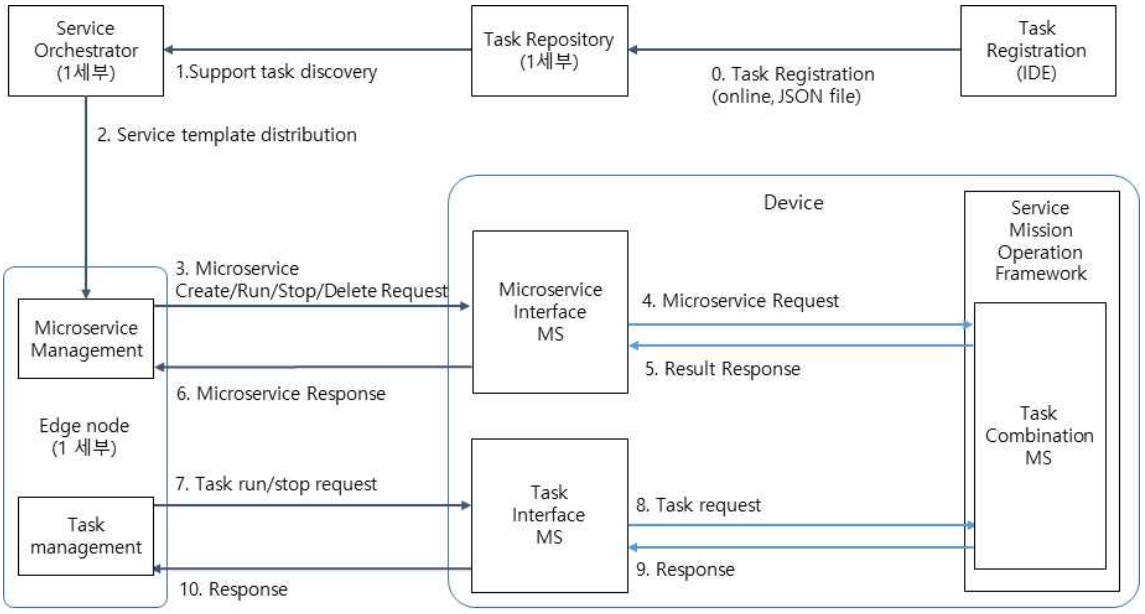


[그림 5] 인증 및 등록 마이크로서비스

3.1.2.2. 태스크 오케스트레이션 및 오프로딩 연동 마이크로서비스 기능

초소형 IoTWare 서비스 인프라 연동 프레임워크는 서비스 인프라의 태스크 오케스트레이션 서비스와 태스크 오프로딩 서비스를 제공하기 위하여 태스크 오케스트레이션 연동 기능과 태스크 오프로딩 연동 기능을 제공한다.

태스크 오케스트레이션 서비스는 개발자가 개발한 태스크와 마이크로서비스를 태스크 저장소(Task repository)에 등록하면서부터 시작된다. 서비스 오케스트레이터(Service orchestrator)는 개발자가 등록한 태스크와 마이크로서비스 정보를 바탕으로 태스크의 조합으로 구성된 마이크로서비스 템플릿(template)을 구성한다. 구성된 서비스 템플릿은 각각의 용도에 따라 클라우드와 엣지로 배포된다. 엣지에는 엣지 노드의 서비스 템플릿과 각각의 디바이스를 위한 서비스 템플릿이 함께 배포된다. 디바이스를 위한 마이크로서비스 템플릿은 마이크로서비스 인터페이스로 전달되고 다시 태스크 분해/재조합 태스크로 전달되어 태스크 조합을 통한 할당받은 마이크로서비스를 수행할 수 있게 된다. 마이크로서비스가 수행되는 동안에도 새로운 태스크가 수행되거나 기존의 태스크를 중지하는 등의 태스크 오프로딩 서비스를 수행할 수 있다.

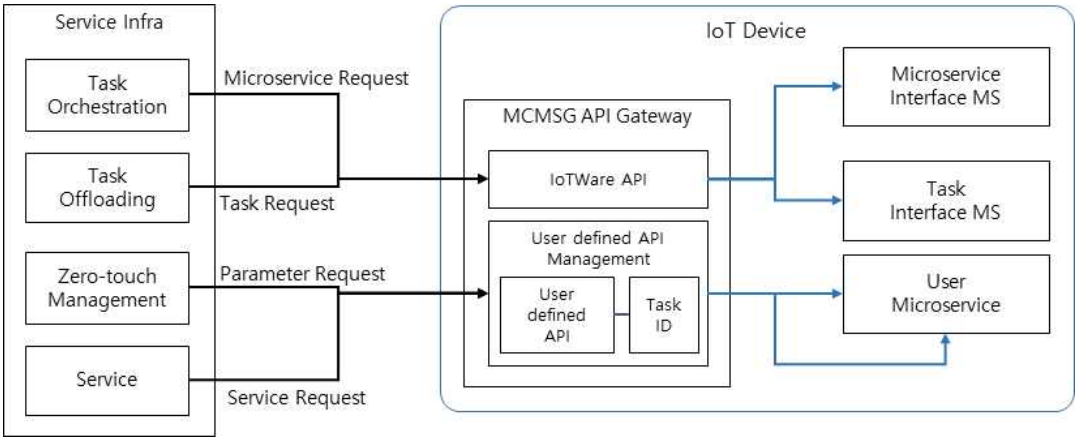


[그림 6] 태스크 오케스트레이션 및 오프로딩 연동 마이크로서비스

3.1.2.3. 초소형 IoTWare 플랫폼 연동 게이트웨이

초소형 IoTWare 플랫폼 연동 게이트웨이는 초소형 IoTWare 플랫폼 연동을 위하여 마이크로서비스 제어 메시지(Microservice Control Message, MCMSG)를 지원하여 IoTWare 서비스 인프라 연동 API와 마이크로서비스 제어 메시지 기반 사용자 마이크로서비스 API를 지원한다.

초소형 IoTWare API는 태스크 오케스트레이션, 태스크 오프로딩 등 초소형 IoTWare 플랫폼 서비스를 지원하는 인터페이스이다. 사용자 정의 API는 사용자가 마이크로서비스를 개발하면서 정의한 인터페이스로 사용자 정의 API와 사용자 마이크로서비스의 태스크 ID와 결합하여 서비스된다.



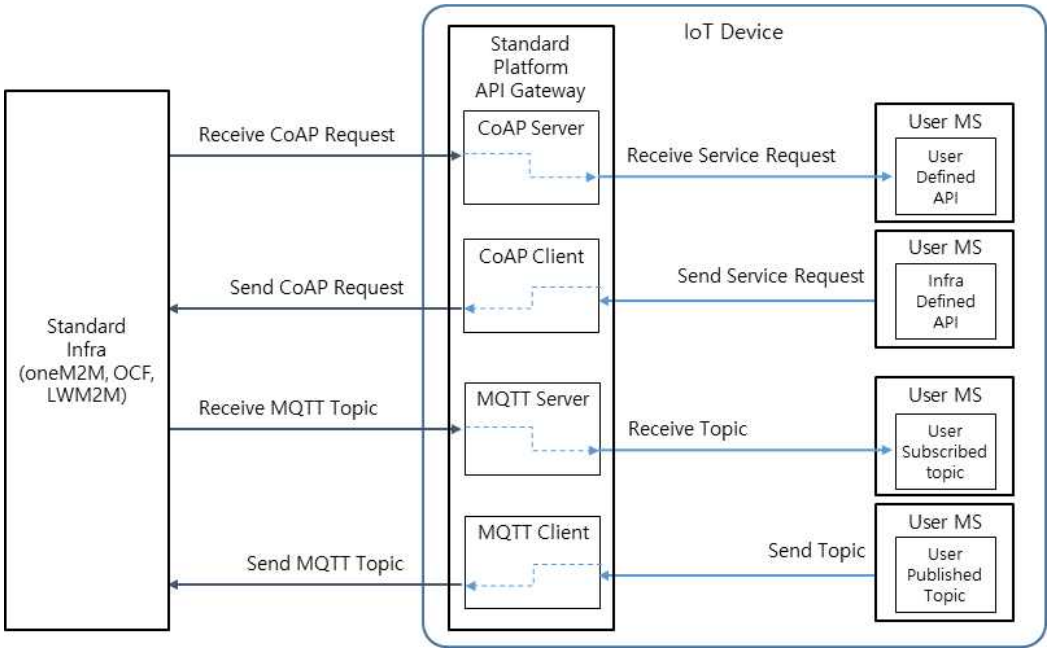
[그림 7] 초소형 IoTWare 플랫폼 연동 게이트웨이

3.1.2.4. 표준 플랫폼 연동 게이트웨이

표준 플랫폼 연동 게이트웨이는 oneM2M, OCF, LWM2M 등의 기존 플랫폼 연동을 위하여 CoAP와 MQTT Server 및 Client 기능을 제공한다.

CoAP(Constrained Application Protocol)는 RESTful 기반의 응용 데이터 전달 프로토콜로 oneM2M, OCF, LWM2M 등의 표준 프로토콜로 사용된다. UDP를 이용한 Request/Response 모델로 서버와 클라이언트로 구성된다.

MQTT(Message Queue for Telemetry Transport)는 M2M 또는 IoT 기기와 G/W의 연동을 위해 정의된 프로토콜이다. 토픽(topic)이라고 하는 메시지를 중계하는 브로커(broker)를 중심으로 토픽을 생산하는 발행자(publisher)와 토픽을 소비하는 구독자(subscriber)로 구성된다.

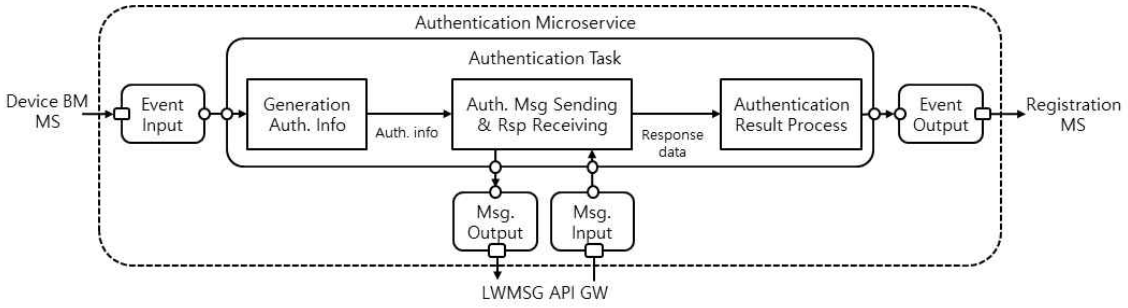


[그림 8] 표준 플랫폼 연동 게이트웨이

3.1.3. 서비스 인프라 연동 프레임워크 구조 및 인터페이스

3.1.3.1. 인증 마이크로서비스 구조 및 인터페이스

인증 마이크로서비스는 인증 및 등록 과정에서 IoTWare 서비스에 소속됨을 확인하는 과정을 수행한다. 디바이스가 네트워크 연결 등 최초 부팅이 성공적으로 이루어지면 본격적인 인증 과정을 수행한다. 디바이스의 부팅 정보를 수신한 인증 마이크로서비스는 인증을 위한 인증 정보를 생성하고 인증 메시지를 통해 인증을 수행한다. 인증 결과를 수신하면 결과에 따라 등록 과정을 수행하거나 재인증을 시도한다.



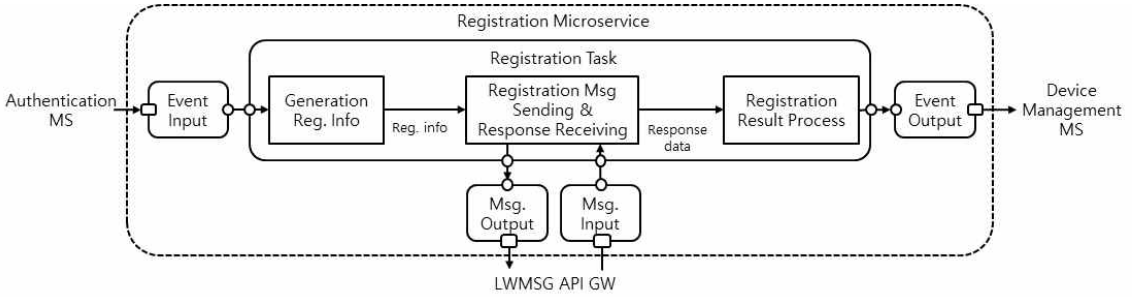
[그림 9] 인증 마이크로서비스 구조

[표 3] 인증 마이크로서비스 인터페이스

이름	입출력	설명
Device_Status	I	디바이스 상태 정보
Network_Status	I	네트워크 상태 정보
Auth_Result	O	디바이스 인증 결과 정보

3.1.3.2. 등록 마이크로서비스 구조 및 인터페이스

등록 마이크로서비스는 인증 및 등록 과정에서 IoTWare 서비스를 본격적으로 시작하기 위해 디바이스의 각종 정보를 제공하고 등록 정보를 수신한다.



[그림 10] 등록 마이크로서비스 구조

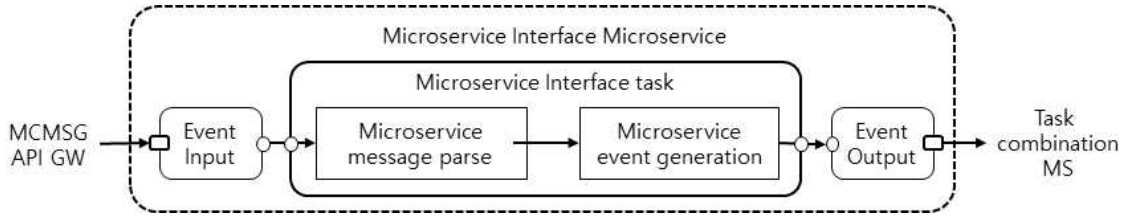
등록 마이크로서비스는 인증 마이크로서비스의 결과가 성공이었을 때 수행된다. 디바이스의 각종 정보를 포함하는 등록 정보를 생성하고 등록 메시지를 통하여 디바이스 등록 과정을 수행한다. 등록 결과에 따라 등록이 성공적으로 수행된 경우에는 사용자 태스크 실행을 준비하고 태스크 오케스트레이션 정보를 수신하기 위해 태스크 오케스트레이션 마이크로서비스를 실행한다. 태스크 오케스트레이션 기능을 제공하지 않으면 사용자 태스크를 바로 실행할 수 있다. 등록이 실패한 경우에는 재등록을 시도한다.

[표 4] 등록 마이크로서비스 인터페이스

이름	입출력	설명
Auth_Result	I	인증 결과 정보
Device_Info	I	디바이스 정보
Reg_Result	O	디바이스 등록 결과 정보
Device_ID	O	디바이스 고유 ID

3.1.3.3. 태스크 오케스트레이션 연동 마이크로서비스 구조 및 인터페이스

태스크 오케스트레이션 연동을 위한 마이크로서비스 인터페이스 마이크로서비스는 네트워크로부터 태스크 오케스트레이션을 위한 마이크로서비스 단위의 마이크로서비스 제어 메시지를 수신하면 시작된다. 수신된 메시지를 파싱하는 태스크와 파싱 결과에 따라 임무지향적 태스크 분해/재조합 태스크로 보낼 이벤트 메시지를 생성하는 태스크로 구성된다.



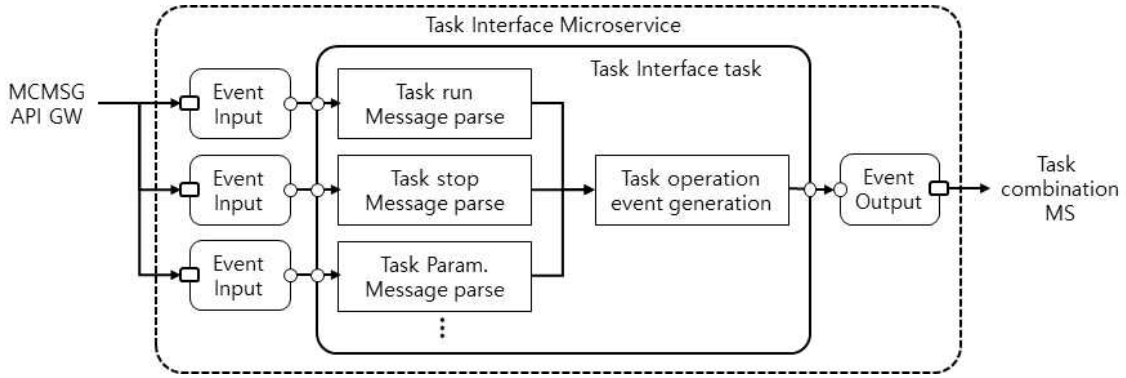
[그림 11] 태스크 오케스트레이션 연동 마이크로서비스 인터페이스

[표 5] 태스크 오케스트레이션 연동 마이크로서비스 인터페이스

이름	입출력	설명
Rcv_Msg	I	태스크 오케스트레이션 연동 API로 수신된 메시지
Task_ID	O	Enable 또는 Disable할 Task의 unique ID
Task_Enable	O	Task_ID를 갖는 Task를 Enable 또는 Disable 한다.
Conn_Task_ID1	O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Task_ID2	O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Enable	O	Conn_Task_ID1을 갖는 Task와 Conn_Task_ID2를 갖는 Task간 메시지 전달 관계를 enable 또는 disable 수행

3.1.3.4. 태스크 오프로딩 연동 마이크로서비스 구조 및 인터페이스

태스크 오프로딩 연동 마이크로서비스는 네트워크로부터 태스크 오프로딩을 위한 태스크 단위의 마이크로서비스 제어 메시지를 수신하면 시작된다. 태스크 오프로딩을 위한 마이크로서비스 제어 메시지에는 태스크 수행(Task run), 중지(Task stop), 일시정지(Task pause), 삭제(Task delete) 등이 있을 수 있다. 수신된 마이크로서비스 제어 메시지에 따라 각각을 파싱하는 태스크와 태스크 상태관리 태스크로 보낼 이벤트를 생성하는 태스크로 구성된다.



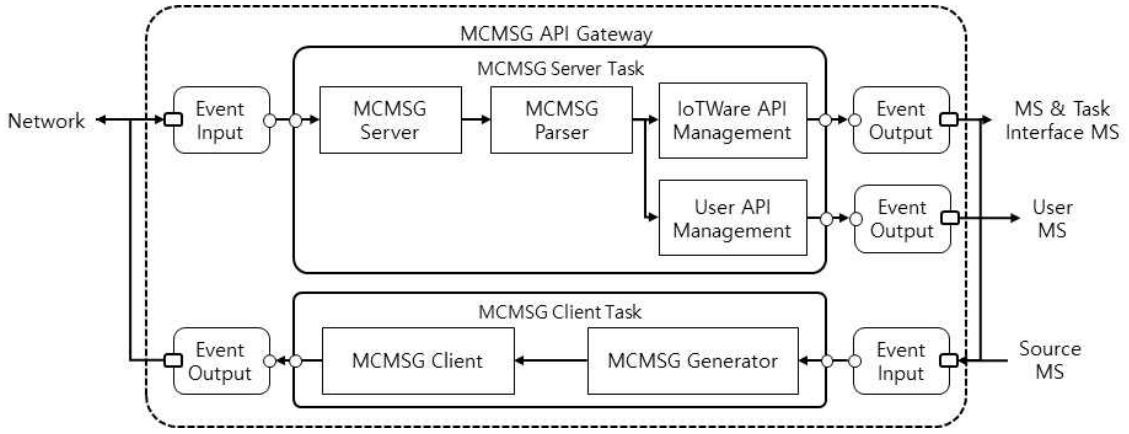
[그림 12] 태스크 오프로딩 연동 마이크로서비스 구조

[표 6] 태스크 오프로딩 연동 마이크로서비스 인터페이스

이름	입출력	설명
Rcv_Msg	I	태스크 오프로딩 연동 API로 수신된 메시지
Task_ID	O	Enable 또는 Disable할 Task의 unique ID
Task_Enable	O	Task_ID를 갖는 Task를 Enable 또는 Disable 한다.
Conn_Task_ID1	O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Task_ID2	O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Enable	O	Conn_Task_ID1을 갖는 Task와 Conn_Task_ID2를 갖는 Task간 메시지 전달 관계를 enable 또는 disable 수행

3.1.3.5. 초소형 IoTWare 플랫폼 연동 게이트웨이 구조 및 인터페이스

초소형 IoTWare 플랫폼 연동 게이트웨이는 마이크로서비스 제어 메시지 서버와 클라이언트로 구성된다. 마이크로서비스 제어 메시지 서버(MCMSG Server)는 마이크로서비스 제어 메시지 파서와 API와 해당 태스크의 ID를 관리하는 API-Task ID 관리 기능을 가진다. 마이크로서비스 구조의 사용자 응용 프로그램을 지원하기 위한 마이크로서비스 제어 메시지를 이용한 초소형 IoTWare 인프라 연동 API를 지원한다. 마이크로서비스 제어 메시지의 규격은 1세부에서 정의하는 마이크로서비스 제어 메시지 규격을 따른다.

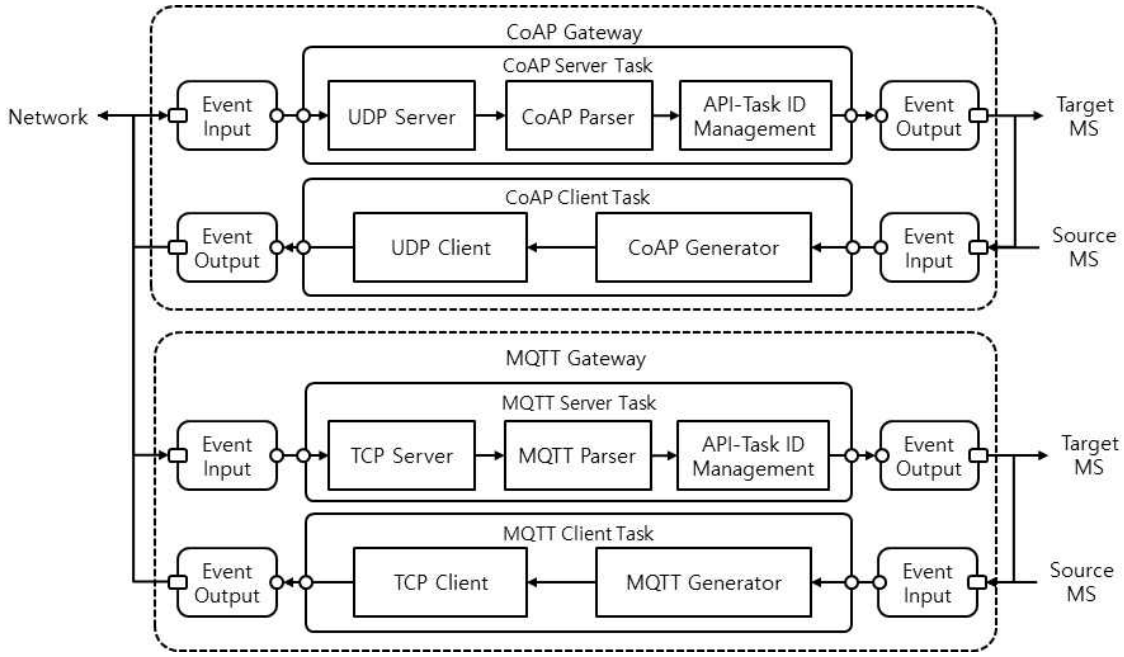


[그림 13] 초소형 IoTWare 플랫폼 연동 게이트웨이 구조

[표 7] 초소형 IoTWare 플랫폼 연동 게이트웨이 인터페이스

이름	입출력	설명
Rcv_Msg	I	플랫폼 표준 API로 수신된 메시지
Rcv_Event	I	전송할 데이터를 포함한 이벤트 메시지
Task_ID	O	메시지를 전달할 Task의 unique ID
Task_data	O	수신된 메시지의 Payload

3.1.3.6. 표준 플랫폼 연동 게이트웨이 구조 및 인터페이스



[그림 14] 표준 플랫폼 연동 게이트웨이 구조

표준 플랫폼 연동 게이트웨이는 CoAP 게이트웨이, MQTT 게이트웨이로 구성된다. 각각의 게이트웨이는 UDP, TCP와 같은 세션 단위로 관리를 지원하는 전송 계층이 필요하다. 전송 계층을 지원하는 디바이스에서는 동시에 사용 가능하며 지원하지 않는 디바이스에서는 전송 계층에 대응하는 기능이 필요하다.

[표 8] 기존 플랫폼 연동 게이트웨이 인터페이스

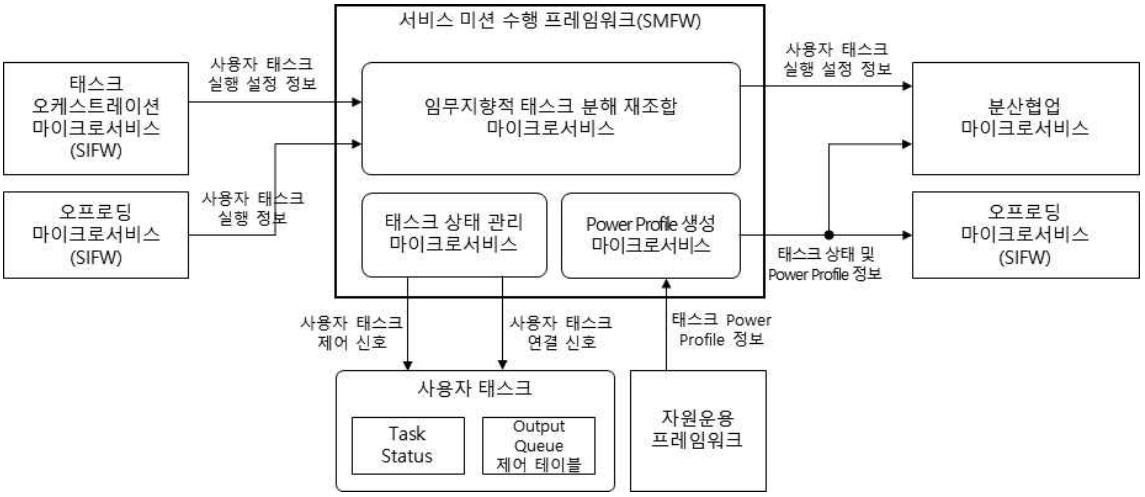
이름	입출력	설명
Rcv_Msg	I	플랫폼 표준 API로 수신된 메시지
Rcv_Event	I	전송할 데이터를 포함한 이벤트 메시지 - CoAP의 Code, URI, Payload 등 - MQTT의 topic 등
Task_ID	O	메시지를 전달할 Task의 unique ID
Task_data	O	수신된 메시지의 Payload

3.2. 서비스 미션 수행 프레임워크 구조 설계

3.2.1. 서비스 미션 수행 프레임워크 개요

서비스 미션 수행 프레임워크는 사용자 태스크 등 태스크의 상태 관리 기능을 수행하는 태스크 상태관리 블록, 클라우드 또는 엣지의 오프로딩에 따른 임무지향적 태스크 분해 및 재조합 기능을 수행하는 블록으로 구성된다. 서비스 미션 수행 프레임워크는 서비스 인프라 연동 프레임워크내 태스크 오케스트레이션 마이크로서비스와 태스크 오프로딩 마이크로서비스와의 태스크 제어 인터페이스가 존재한다. 태스크별 전력 프로파일 관련 정보를 자원운용 프레임워크에서 획득하여 분산협업 마이크로서비스로 전달하는 인터페이스가 존재한다.

3.2.2. 서비스 미션 수행 프레임워크 주요 마이크로서비스 기능



[그림 15] 서비스 미션 수행 프레임워크 및 인터페이스

서비스 미션 수행 프레임워크는 사용자 마이크로서비스내 태스크 상태 관리 기능을 수행한다. 서비스 인프라 연동 프레임워크의 태스크 오케스트레이션 마이크로서비스와 태스크 오프로딩에서 전달되는 사용자 태스크 실행 설정 정보와 사용자 태스크 정보를 전달받는다.

전달된 정보를 이용하여 사용자 태스크를 실행하거나 실행하지 않거나 설정한다.

사용자 태스크 실행 설정 정보와 사용자 태스크 정보를 이용하여 사용자 태스크를 바로 설정하지 않고, 분산협업 마이크로서비스에 이러한 정보들을 전달하게 되고, 분산협업 마이크로서비스내 서비스 동적 자율 구성 태스크는 자율 구성 클러스터내 어느 디바이스에서 해당 태스크를 수행하는 것이 적합한지 판단하여 해당 디바이스로 분산협업 마이크로서비스 제어 메시지를 이용하여 전달한다.

서비스 동적 자율 구성 태스크는 태스크 실행 여부 판단을 전력 프로파일 생성 마이크로서비스에서 전달된 태스크 상태 및 전력 프로파일 정보를 이용하여 종합적으로 판단하여, 어떤 디바이스에서 태스크를 수행하도록 설정한다. 태스크 상태 및 전력 프로파일 정보는 전력 프로파일 생성 마이크로서비스에서 태스크 상태 관리 마이크로서비스에서 전달된 태스크 정보와 자원 운용 프레임워크에서 전달된 태스크 전력 프로파일 정보를 이용하여 생성한다.

태스크 상태 관리 마이크로서비스는 분산협업 마이크로서비스내 서비스 동적 자율 구성 태스크에서 생성된 태스크 제어 정보를 이용하여 사용자 마이크로서비스내 사용자 로직 태스크에 태스크 제어 신호를 전달한다. 임무지향적 태스크 분해 및 재조합 마이크로서비스는 사용자 논리 태스크들의 출력 연결을 분해 및 재조합하여 사용자 마이크로서비스의 임무를 오케스트레이션 및 오프로딩이 이루어질 수 있도록 수행한다.

서비스 미션 수행 프레임워크에서 수행하는 주요 기능을 요약하면 다음과 같다.

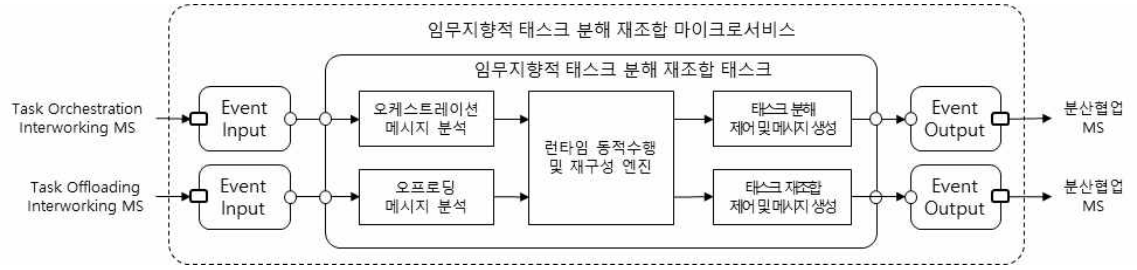
- 태스크 상태 관리 기능
- 태스크간 메시지 큐 연결 관리 기능
- 태스크 실행 및 중지 명령 수행 기능
- 태스크 분해 및 재조합 메시지 분석 기능
- 서비스 런타임 동적 재구성 기능
- 태스크별 사용 전력량 취합 및 메시지 전달 기능

3.2.3. 서비스 미션 수행 프레임워크 구조 및 인터페이스

3.2.3.1. 임무지향적 태스크 분해 재조합 마이크로서비스 구조 및 인터페이스

서비스 미션 수행 프레임워크는 임무지향적 태스크 분해 재조합 마이크로서비스, 태스크 상태 관리 마이크로서비스 및 전력 프로파일 생성 마이크로서비스로 구성된다.

임무지향적 태스크 분해 재조합 마이크로서비스는 서비스 인프라 연동 프레임워크에서 전달된 태스크 오케스트레이션 마이크로서비스 및 태스크 오프로딩 마이크로서비스 메시지를 분석하여, 태스크 분해 및 재조합을 위한 정보를 런타임 동적수행 및 재구성 엔진에서 분산협업 마이크로서비스에 전달하기 위한 태스크 제어 정보 메시지를 생성한다. 생성된 메시지를 이벤트 메시지 송수신 기능을 이용하여 분산협업 마이크로서비스로 전달한다.



[그림 16] 임무지향적 태스크 분해 재조합 마이크로서비스 구조

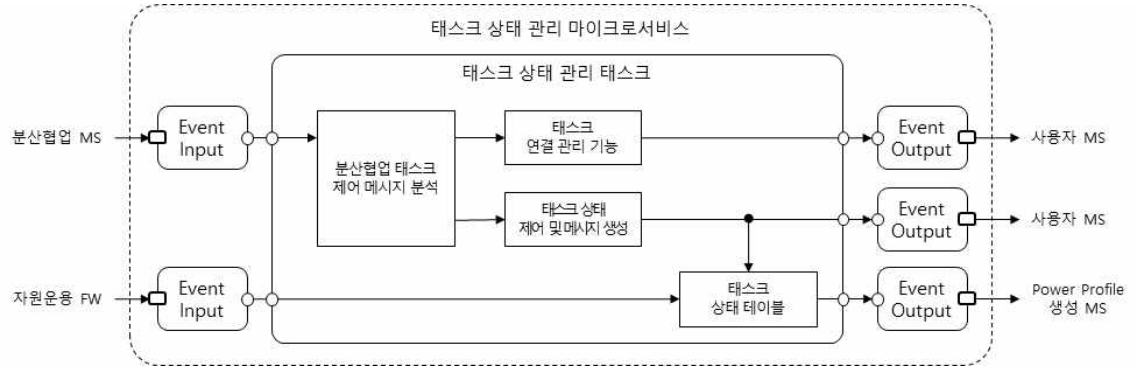
임무지향적 태스크 분해 재조합 마이크로서비스는 서비스 인프라 연동 프레임워크내 태스크 오케스트레이션 마이크로서비스와 태스크 오프로딩 마이크로서비스와의 태스크 제어 인터페이스가 존재한다. 디바이스별 태스크별 제어를 위한 메시지를 분산협업 마이크로서비스로 전달하는 인터페이스가 존재한다.

[표 9] 임무지향적 태스크 분해 재조합 마이크로서비스 주요 인터페이스

이름	입출력	설명
Task_ID	I/O	Enable 또는 Disable할 Task의 unique ID
Task_Enable	I/O	Task_ID를 갖는 Task를 Enable 또는 Disable 한다.
Conn_Task_ID1	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Task_ID2	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Enable	I/O	Conn_Task_ID1을 갖는 Task와 Conn_Task_ID2를 갖는 Task간 메시지 전달 관계를 enable 또는 disable 수행

3.2.3.2. 태스크 상태 관리 마이크로서비스 구조 및 인터페이스

태스크 상태관리 마이크로서비스는 분산협업 마이크로서비스에서 전달된 태스크 상태관리 메시지에서 획득한 사용자 마이크로서비스간 연결 관리 및 사용자 마이크로서비스내 사용자 태스크 동작 수행 여부 정보를 이용하여 사용자 태스크를 제어한다. 전력 프로파일 마이크로서비스에 현재 태스크 상태 정보를 전달하여, 태스크별 전력 프로파일을 분산협업 마이크로서비스에 전달할 수 있도록 한다.



[그림 17] 태스크 상태 관리 마이크로서비스 구조

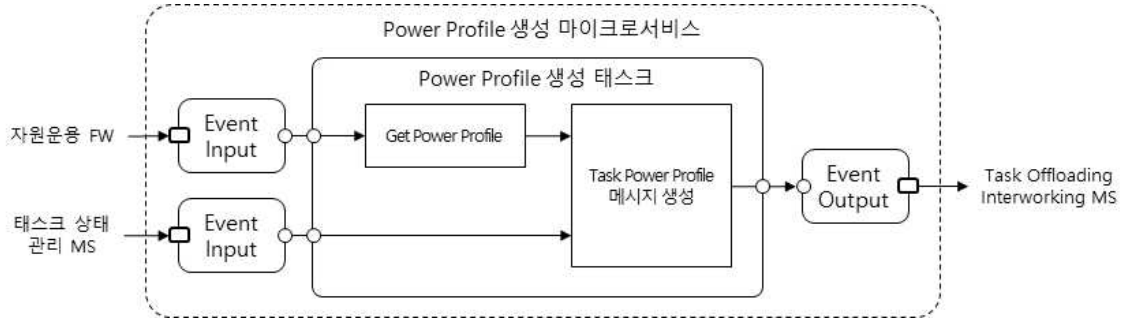
태스크 상태 관리 마이크로서비스는 분산협업 마이크로서비스 사이에 디바이스별 사용자 마이크로서비스내 사용자 논리 태스크 제어 인터페이스가 존재한다. 자원운용 프레임워크에서 전력 프로파일을 조사할 태스크 목록을 얻기 위해 전달되는 태스크 목록 요청 인터페이스가 존재한다. 태스크 상태 관리 마이크로서비스와 사용자 마이크로서비스 또는 사용자 논리 태스크간 디바이스별 태스크별 제어를 위한 인터페이스가 존재한다.

[표 10] 태스크 상태 관리 마이크로서비스 주요 인터페이스

이름	입출력	설명
Device_ID	I/O	제어 신호가 적용되는 디바이스 unique ID
User_MS_ID	I/O	제어 신호의 사용자 마이크로서비스 unique ID
Task_ID	I/O	Enable 또는 Disable할 Task의 unique ID
Task_Enable	I/O	Task_ID를 갖는 Task를 Enable 또는 Disable 한다.
Conn_Task_ID1	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Task_ID2	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Enable	I/O	Conn_Task_ID1을 갖는 Task와 Conn_Task_ID2를 갖는 Task간 메시지 전달 관계를 enable 또는 disable 수행
Task_Pwr_Profile	I/O	Task별 생성된 전력관련 profile을 태스크오프로딩 마이크로서비스로 전달하는 구조체

3.2.3.3. Power profile 생성 마이크로서비스 구조 및 인터페이스

전력 프로파일 생성 마이크로서비스는 자원 운용 프레임워크에서 전달된 태스크별 전력 프로파일과 태스크 상태를 이용하여 원하는 시간동안 디바이스가 동작하는 것을 도와주기 위해 태스크 오프로딩에 해당 정보를 전달하는 구조를 갖는다. 분산협업 마이크로서비스내 서비스 동적 자율 구성 태스크에서 디바이스별 태스크 할당 알고리즘 수행을 위한 정보를 전달하는 구조를 갖는다.



[그림 18] Power profile 생성 마이크로서비스 구조

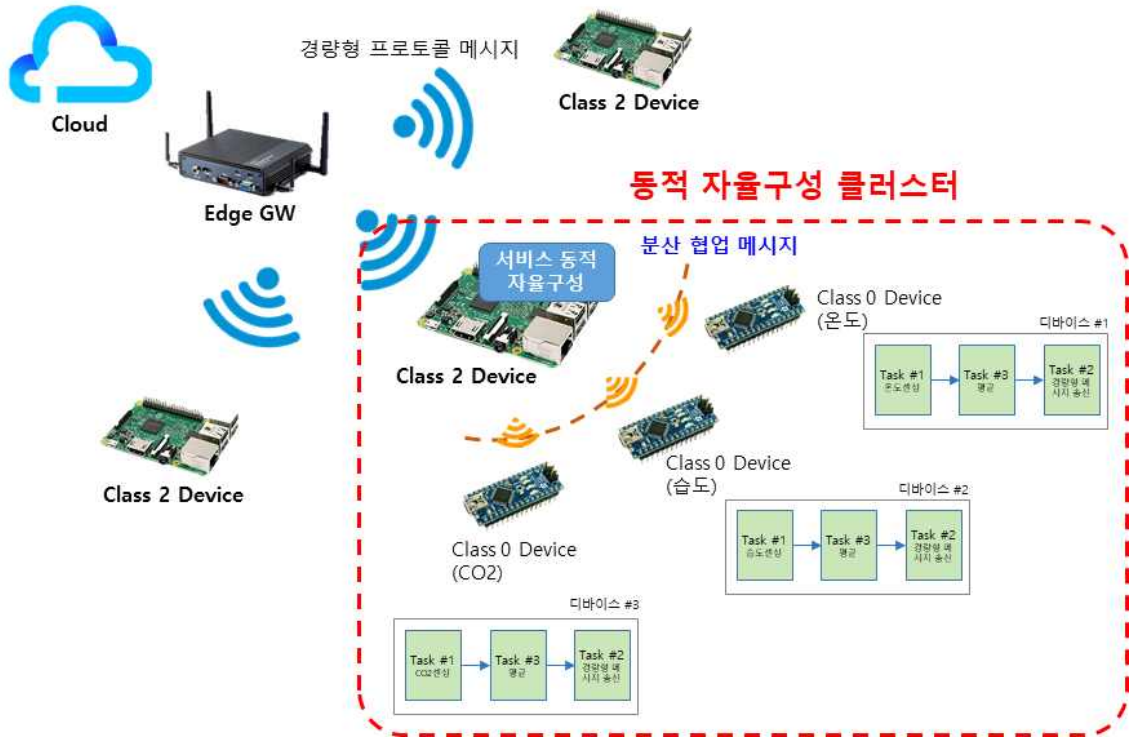
전력 프로파일 생성 마이크로서비스는 자원운용 프레임워크, 태스크 상태 관리 마이크로서비스 및 태스크 오프로딩 마이크로서비스와 인터페이스가 존재한다.

[표 11] Power profile 생성 마이크로서비스 주요 인터페이스

이름	입출력	설명
Device_ID	O	제어 신호가 적용되는 디바이스 unique ID
User_MS_ID	O	제어 신호의 사용자 마이크로서비스 unique ID
Task_ID	I	Enable 또는 Disable할 Task의 unique ID
Task_Pwr_Profile	I/O	Task별 생성된 전력관련 profile을 태스크오프로딩 마이크로서비스로 전달하는 구조체

3.3. 동적 자율구성 기반 분산협업 마이크로서비스 구조 설계

3.3.1. 동적 자율구성 기반 분산협업 마이크로서비스 개요



[그림 19] 서비스 동적 자율구성 기반 분산협업 구성도

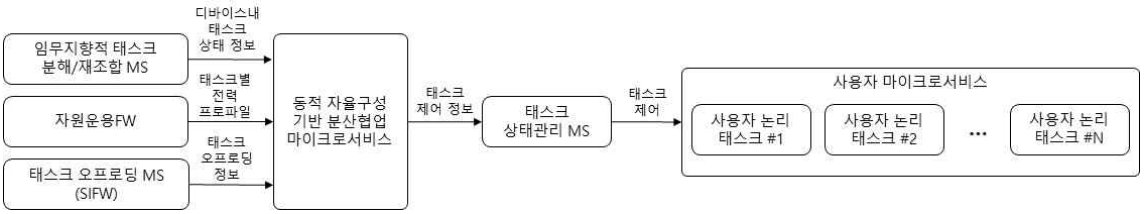
자율구성클러스터내 분산협업을 위한 망 토폴로지는 Class 2 Device 중심의 star topology로 구성된다. 자율구성클러스터내 디바이스간 메시지 교환은 분산 협업 메시지를 사용한다. 분산 협업 메시지는 broker를 이용한 publisher와 subscriber간 메시지 전송 방식을 사용한다. Star topology에서 broker 역할을 수행하는 디바이스는 class 2급의 gateway 역할을 수행하며, 1세부의 edged gateway에서 전달된 task orchestration, task offloading 및 zero touch management 등 마이크로서비스 제어 프로토콜 메시지에 따라 디바이스내 태스크를 수행하게 된다.

자율구성클러스터내 디바이스간 분산협업을 위해서는 Class 2 Device내에 서비스 동적 자율 제어 마이크로서비스가 정의되어 있어야한다. 서비스 동적 자율 제어 마이크로서비스는 자율구성클러스터내 디바이스별 태스크 상태, 태스크별 전력 소모 등을 수집하고, 이를 이용하여 분산협업을 위한 태스크 오프로딩 결정 및 제어를 수행한다.

3.3.2. 동적 자율구성 기반 분산협업 마이크로서비스 주요 기능

동적 자율구성 기반 분산협업 마이크로서비스는 이벤트 메시지 수신 태스크, 분산협업 메시지 송신 태스크 및 서비스 동적 자율제어 태스크로 구성되며 클러스터내 Class 2 Device에 위치한다. 분산협업 마이크로서비스는 태스크 오케스트레이션 마이크로서비스 및 태스크 오프로딩 마이크로서비스에서 수행 지시된 태스크를 클러스터내 어느 디바이스에서 수행할 것인지 결정하는 기능을 수행한다. 이러한 결정 기능은 분산협업 마이크로서비스내 서비스 동적 자율제어 태스크에서 수행한다.

동적 자율구성 기반 분산협업 마이크로서비스의 서비스 동적 자율제어 기능을 수행하기 위해, 클러스터내 디바이스의 태스크 종류를 취합하고 태스크별 전력 프로파일을 취합 비교하는 기능을 수행하여야 한다. 엣지와 통신이 원할히 이루어지지 않은 경우 또는 클러스터내에서 서비스를 수행하는 것이 적절한 경우 클러스터내 태스크 오프로딩 판단 제어하는 기능을 수행하여야 한다.



[그림 20] 동적 자율구성 기반 분산협업 마이크로서비스

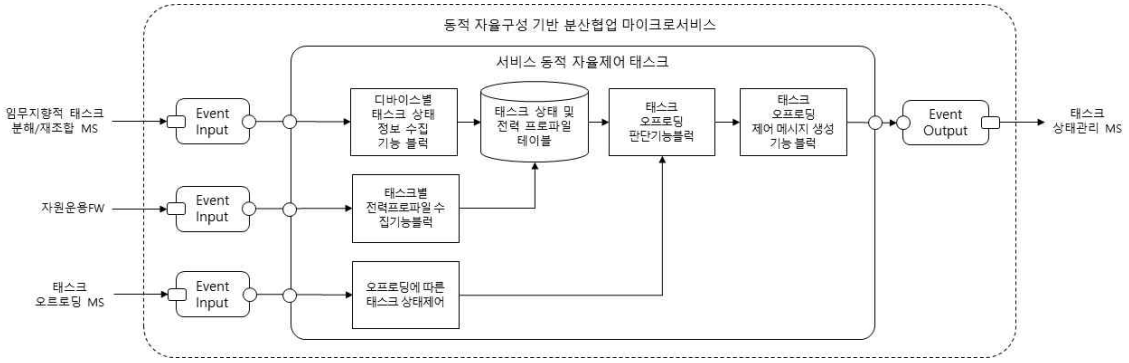
동적 자율구성 기반 분산협업 마이크로서비스가 수행하는 주요 기능은 다음과 같은 기능을 수행한다.

- 클러스터내 디바이스의 태스크 종류 취합 기능
- 클러스터내 태스크별 전력 프로파일 취합 기능
- Edged gateway와 통신 가능 여부 확인 기능
- 태스크 오프로딩에 따른 태스크 상태 및 연결 관리 제어 기능
- 클러스터내 디바이스간 태스크 오프로딩 판단 제어 기능
- 분산협업 메시지 송수신 기능
- 이벤트 메시지 송수신 기능

3.3.3. 동적 자율구성 기반 분산협업 마이크로서비스 구조 및 인터페이스

Class 2 Device내 동적 자율구성 기반 분산협업 마이크로서비스 구조는 이벤트 메시지 수신 태스크, 분산협업메시지 송수신 태스크 및 서비스 동적 자율제어 태스크 총 3개의 태스크로 구성된다. 서비스 동적 자율제어 태스크는 클러스터내 디바이스의 태스크 종류 취합 기능, 클러스터내 태스크별 전력 프로파일 취합 기능, Edged gateway와 통신 가능 여부 확인 기능 및

클러스터내 디바이스간 태스크 오프로딩 판단 제어 기능 등을 수행하는 블록으로 구성된다.



[그림 21] 동적 자율구성 기반 분산협업 마이크로서비스 구조

동적 자율구성 기반 분산협업 마이크로서비스는 다른 디바이스의 서비스 미션수행 프레임워크 및 자원운용 프레임워크와 분산협업 메시지를 통한 인터페이스, 서비스 미션수행 프레임워크의 이벤트 메시지 인터페이스가 존재한다. 분산협업 마이크로서비스 인터페이스 주요 파라미터를 나타내었다.

[표 12] 동적 자율구성 기반 분산협업 마이크로서비스 주요 인터페이스

이름	입출력	설명
Coop_Task_Pwr_Profile	I	Parameter를 설정하거나 얻어올 Task의 unique ID
Coop_Task_Status	I	값을 설정 또는 얻기 위한 Parameter의 ID
Coop_Task_ID	I	값을 설정하는 것인지 얻는 것인지 표시
Device_ID	O	태스크 제어를 받는 Device의 unique ID
Task_ID	I/O	Enable 또는 Disable할 Task의 unique ID
Task_Enable	I/O	Task_ID를 갖는 Task를 Enable 또는 Disable 한다.
Conn_Task_ID1	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Task_ID2	I/O	Connect 또는 Disconnect할 Task의 unique ID
Conn_Enable	I/O	Conn_Task_ID1을 갖는 Task와 Conn_Task_ID2를 갖는 Task 간 메시지 전달 관계를 enable 또는 disable 수행

3.4. 사용자 마이크로서비스 구조 설계

3.4.1. 사용자 마이크로서비스 개요

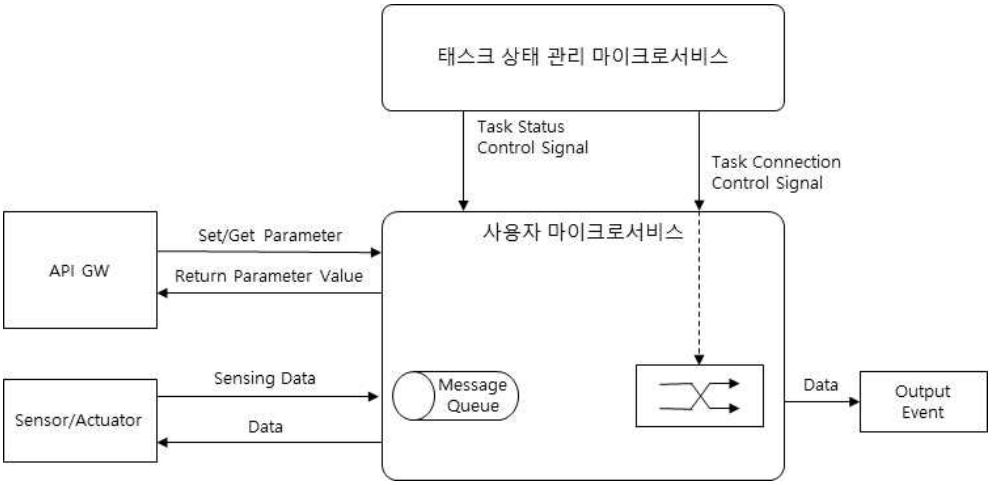
사용자 마이크로서비스는 디바이스에서 수행하는 고유 업무이며, 클라우드나 엣지에서 태스크 단위로 지정되어 수행된다. 사용자는 사용자 마이크로서비스 작성 시, 사용자 입력 태스크와 사용자 출력 태스크는 입출력으로 사용할 메시지를 선택하게 되며, 사용자 로직 태

스크의 logic function 블록만 코딩하게 된다.

3.4.2. 사용자 마이크로서비스 주요기능

사용자 마이크로서비스에서 수행하는 기능은 디바이스에 할당된 고유 업무를 수행한다. 사용자 마이크로서비스내 태스크 오프로딩 지원을 위해 태스크 실행 및 실행 중기 기능을 지원해야 한다. 태스크간 데이터 전달 연결을 관리하여 임무지향적 태스크 분해 및 재조립이 가능하도록 해야 한다. 사용자 논리 태스크를 수행하기 위해 필요한 파라미터를 설정하는 기능을 제공해야 한다.

사용자 논리 태스크 파라미터 설정은 API 게이트웨이를 통해 전달되며, 센서에서 전달되는 데이터는 메시지 큐를 통해 전달된다. 태스크 상태 관리 마이크로서비스에서 전달되는 사용자 논리 태스크 상태 제어를 위한 신호에 따라 태스크를 실행 또는 중지하며, 태스크 연결 제어 신호에 따라 출력 신호를 제어하는 기능을 제공한다.



[그림 22] 사용자 마이크로서비스

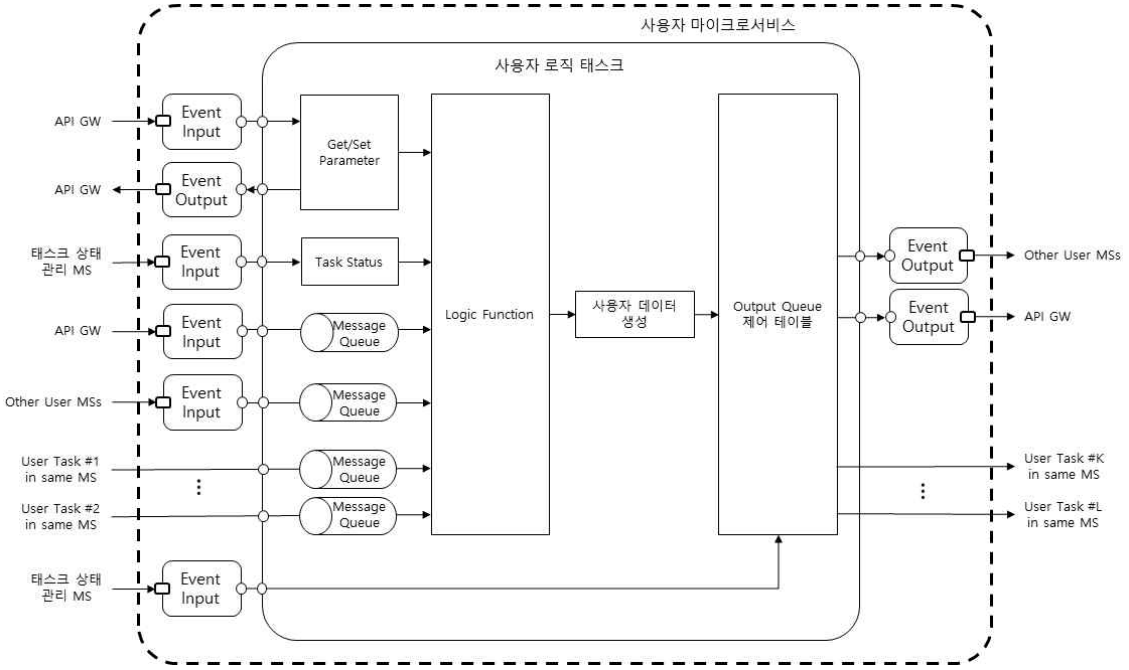
사용자 마이크로서비스에서 수행하는 기능은 다음과 같다.

- 메시지 입력 기능
- 출력 메시지 연결 제어에 따른 메시지 출력 기능
- 태스크 파라미터 설정 및 값 제공 기능
- 태스크 메시지 큐 관리 기능
- 태스크 실행 및 중지 기능
- 태스크 실행 및 중지 가능 여부 제공 기능
- 태스크 상태 설정 기능
- 태스크 상태에 따른 로직 함수 수행 기능

3.4.3. 사용자 마이크로서비스 구조 및 인터페이스

사용자 마이크로서비스는 사용자 입력 태스크, 사용자 출력 태스크 및 사용자 로직태스크로 구성된다. 사용자 입출력 태스크는 이벤트 메시지로 전달되는 메시지 송수신 기능을 수행한다. 사용자 로직 태스크는 디바이스에서 고유 업무를 수행하는 Logic function이 정의된

다. Logic function을 수행하기 위해 필요한 각 파라미터는 Get/Set Parameter 블록을 이용하여 설정된다. 사용자 로직태스크에 전달되는 데이터는 메시지 큐로 전달되며 logic function 수행 후 Output Queue 제어 테이블에서 지정한 태스크로 출력 데이터를 전달한다.



[그림 23] 사용자 마이크로서비스 구조

사용자 마이크로서비스는 서비스 인프라 연동 프레임워크내 zero touch management에서 전달되는 파라미터 설정 인터페이스가 존재한다. 또 다른 사용자 마이크로서비스와는 virtual layer에서 제공하는 이벤트 메시지 인터페이스가 존재한다.

[표 13] 사용자 마이크로서비스 인터페이스

이름	입출력	설명
Para_Task_ID	I	Parameter를 설정하거나 얻어올 Task의 unique ID
Para_ID	I	값을 설정 또는 얻기 위한 Parameter의 ID
Para_Get	I	값을 설정하는 것인지 얻는 것인지 표시
Para_Value	I	설정하는 Parameter 값

3.5. IDE 인터페이스 구조 설계

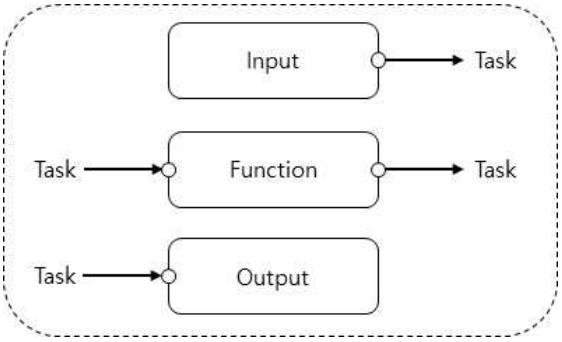
3.5.1. 개요

IoT 디바이스 개발환경 플랫폼(IDE)은 IoT 디바이스 또는 프레임워크에서 발생한 이벤트를 처리하기 위한 브라우저 기반 프로그래밍 (Drag&Drop) 환경이다. 즉, 흐름 기반 프로그래밍 도구(Flow-based programming tool)이다. 단위 태스크(Task) 기반의 박스를 나열하고, 단위

태스크를 서로 연결하고나 끊음으로서(네트워크 구성) 데이터의 흐름을 제어할 수 있다. 또한, 단위 태스크로 구성된 마이크로서비스를 이용하여 또 다른 서비스 설계가 가능하다.

3.5.2. IoT 디바이스 개발환경 플랫폼(IDE) 인터페이스 구조

IoT 디바이스 개발환경 플랫폼(IDE)은 태스크를 추상화한 박스와 정보(이벤트)의 흐름을 추상화한 와이어로 구성된다. 개발환경 플랫폼(IDE)에 사용되는 입력, 출력, 그리고 기능 단위 태스크는 [그림 24]과 같은 구조를 갖는다. 각각의 태스크는 입력과 출력 단자를 갖고 있고, 이벤트 메시지가 흐르게 된다.



[그림 24] 개발환경 플랫폼의 단위 태스크 인터페이스 구조

IoT 디바이스 개발환경 플랫폼(IDE)에 사용되는 입력, 출력, 그리고 기능 단위 태스크는 다음 표 11과 같은 세부 태스크들을 지원할 수 있다.

[표 14] 단위 태스크 인터페이스

인터페이스	태스크	비고
Input	MCMSG	마이크로서비스 제어 메시지 수신
	DCMSG	분산 협업 메시지 수신
	CoAP	CoAP 메시지 수신
	MQTT	MQTT 메시지 수신
	http	HTTP 메시지
	websocket	클라이언트와 서버간 메시지
	tcp	TCP 패킷 입력
	serial	시리얼 통신 메시지
Output	MCMSG	마이크로서비스 제어 메시지 송신
	DCMSG	분산 협업 메시지 송신
	CoAP	CoAP 메시지 송신
	MQTT	MQTT 메시지 송신
	http response	HTTP 응답 메시지
	websocket	클라이언트와 서버간 메시지
	tcp	TCP 패킷 출력
	serial	시리얼 통신 메시지 출력

	debug	디버깅 메시지 출력
Function	function	태스크 기능 부여 (센싱 주기 등)
	trigger	이벤트 트리거
	switch	사용자 조건에 따라 출력의 목적지 태스크를 결정
	change	사용자 조건에 따라 변경
	range	허용 범위 설정
	split	조건에 따른 다수 출력으로 분기
	join	다수 입력을 결합
	sort	조건에 따른 정렬
	MCMSG request	마이크로서비스 제어 메시지 요청
	DCMSG request	분산 협업 메시지 요청
	CoAP request	CoAP 메시지 요청
	MQTT request	MQTT 메시지 요청
	http request	HTTP 메시지 요청
	tcp request	TCP 메시지 요청
	xml	온라인 메시지 입력

3.6. IoTware 자원운용부 구조 및 기능

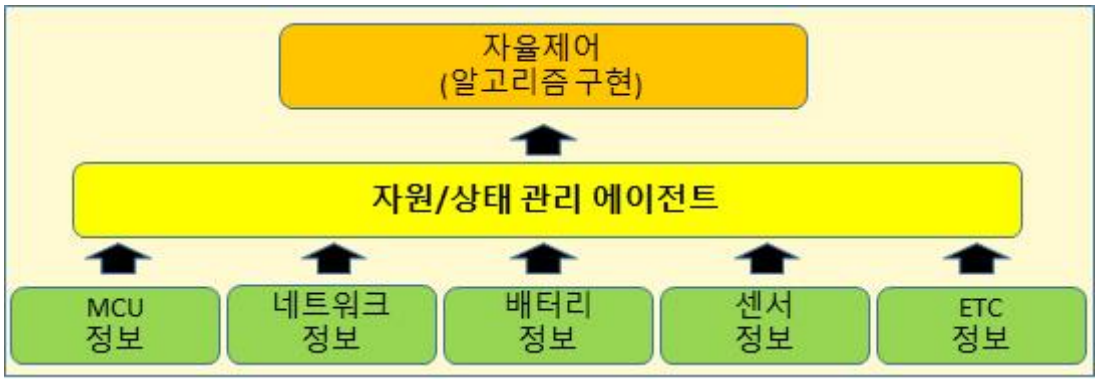


그림 자원 운용 FW 구성도

3.6.1. 전력 프로파일 관리

■ 자원/상태 관리 인터페이스 목록

항목	interface api	추상화 api
MCU 정보	get_mcu_info	iw_get_mcu_info
배터리 상태	get_battery_status	iw_get_battery_status
메모리 사용량 정보	get_memory_status	iw_get_memory_status
네트워크 상태	get_network_status	iw_get_network_status
센서 상태	get_sensor_status	iw_get_sensor_status

■ MCU 정보 반환 API

iw_error_t get_mcu_info(mcu_type_t *id)		
iw_error_t iw_get_mcu_info(mcu_type_t *id)		
기능	MCU 타입을 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	id (out)	MCU 타입 2 : MCU_TYPE_NRF52840 : nRF52840 1 : MCU_TYPE_STM32L476 : STM32L476 0 : MCU_TYPE_UNKNOWN : Unknown

■ 배터리 상태 반환 API

iw_error_t get_battery_status(int *percent,, int *time)		
iw_error_t iw_get_battery_status(int *percent,, int *time)		
기능	배터리 잔량(%) 정보 정보를 얻어 온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	percent (out)	배터리 잔량 (단위: %)
	time (out)	reserved

■ 메모리 사용량 정보 반환 API

iw_error_t get_memory_status(unsigned int *global_data, unsigned int *total_heap, unsigned int *free_heap)		
iw_error_t iw_get_memory_status(unsigned int *global_data, unsigned int *total_heap, unsigned int *free_heap)		
기능	메모리 사용량을 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	global_data (out)	global 데이타 사용량 (단위 : byte)
	total_heap (out)	총 메모리 힙 사이즈 (단위 : byte)
	free_heap (out)	사용 가능한 메모리 힙 사이즈 (단위 : byte)

■ 네트워크 상태 반환 API

iw_error_t get_network_status(unsigned char dev_id, int *connected, int *strength, wake_src_type_t *wake_src, net_pwr_mode_t *mode)		
iw_error_t iw_get_network_status(unsigned char dev_id, int *connected, int *strength, wake_src_type_t *wake_src, net_pwr_mode_t *mode)		
기능	해당 디바이스의 네트워크 연결 상태, 신호 세기, 동작 전력 모드 상태를 반환 온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	connected (out)	네트워크 연결 상태 0 : ap/gateway/host 등에 연결되어 있지 않음. 1 : ap/gateway/host 등에 연결되어 있음.
	strength (out)	네트워크 신호 세기 (단위 : dbm)
	wake_src (out)	wakeup 소스 0 – WAKEUP_SRC_NONE : 없음 1 – WAKEUP_SRC_INTERRUPT : 인터럽트 2 – WAKEUP_SRC_TIMER : 타이머
	mode (out)	네트워크 디바이스 전력 모드 0 - NET_PWR_RUN_MODE : 런 모드 1 - NET_PWR_SLEEP_MODE : 슬립 모드 2 - NET_PWR_LOWPOWER_MODE : 저전력 모드 3 - NET_PWR_AUTO_MODE : 자동 모드

■ 센서 상태 반환 API

iw_error_t get_sensor_status(unsigned char dev_id, wake_src_type_t *wake_src, sensor_pwr_mode_t *mode)		
iw_error_t iw_get_sensor_status(unsigned char dev_id, wake_src_type_t *wake_src, sensor_pwr_mode_t *mode)		
기능	해당 센서의 wakeup 소스와 전력 모드 정보를 가져온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서 디바이스 ID
	wake_src (out)	wakeup 소스 0 – WAKEUP_SRC_NONE : 없음 1 – WAKEUP_SRC_INTERRUPT : 인터럽트 2 – WAKEUP_SRC_TIMER : 타이머
	mode (out)	센서의 전력 모드 0 - SENSOR_PWR_RUN_MODE : 런 모드 1 - SENSOR_PWR_SLEEP_MODE : 슬립 모드 2 – SENSOR_PWR_OFF_MODE : 파워 오프 모드 3 – SENSOR_PWR_AUTO_MODE : 오토 모드

3.6.2. 최적상태 자율제어 및 전력 상태관리

■ 자율제어/전력관리 기본 개념

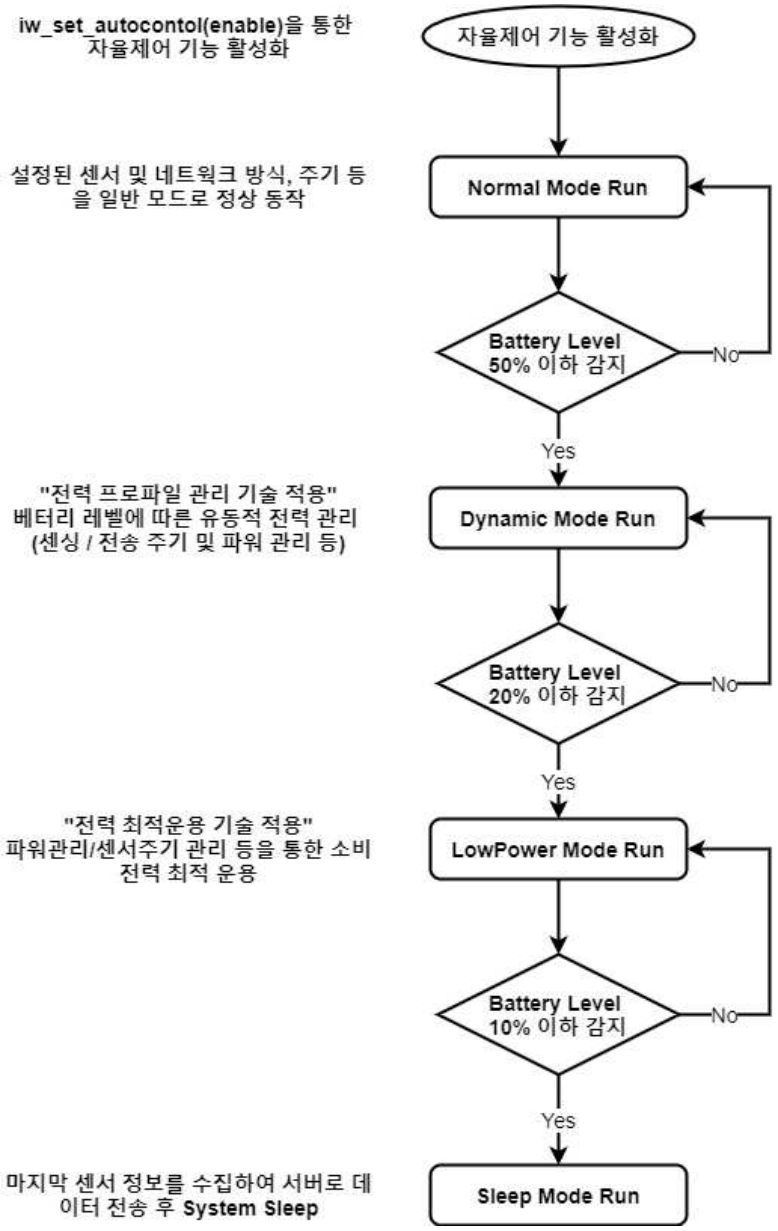


그림 26 최적상태 자율제어를 통한 전력관리 흐름도

기능	설명	기타
자율제어	- 자율 제어 API (autocontrol)를 통해 기능 활성화 - 활성화 시 전력모드는 배터리 상태에 따라 자동으로 변경됨. - 일반 모드: 배터리 50% 이상 감지 시 동작 - 유동적 관리 모드: 배터리 50% 이하 감지 시 동작 - 최적 관리 모드: 배터리 20% 이하 감지 시 동작 - 슬립 모드: 배터리 10% 이하 감지 시 동작 - 비활성화 시 전력 모드는 일반모드로 복귀하며 설정 권한을 전력 모드 API로 넘겨줌.	초기값:enable 모드 변경을 위한 배터리 용량 TH는 수정 가능.
전력관리	- 전력 모드 API (pwr_mode)를 통해 설정 - 일반 모드: 센싱 주기 및 전력 관리를 설정한 상태로 운용 - 유동적 관리 모드: 센싱 및 전송 주기를 배터리 잔량과 비례식을 통해 유기적으로 조절하며 운용 - 최적 관리 모드: 센싱 및 전송 주기를 초저전력 운용하며, 일반모드 대비 30% 이상의 전력감소 효과. - 슬립모드 : 최종 센싱값을 마지막으로 전송하며, 슬립 모드로 진입.	자율제어 함수 활성화 시 자동 관리 비활성화 시 API를 통해 수동관리

■ 자율제어/전력관리 인터페이스 목록

항목	interface api	추상화 api
자율제어 동작 설정	set_autocontrol	iw_set_autocontrol
자율제어 동작 상태	get_autocontrol	iw_get_autocontrol
시스템 전력 모드 설정	set_pwrmode	iw_set_pwrmode
시스템 전력 모드 상태	get_pwrmode	iw_get_pwrmode

■ 자율 제어 동작 설정 API

iw_error_t set_autocontrol(auto_control_t enable)		
iw_error_t iw_set_autocontrol(auto_control_t enable)		
기능	자율 제어 기능을 활성화/비활성화 시킨다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	enable (out)	활성화 상태 0 : 비활성화 (AUTOCONTROL_OFF) 1 : 활성화 (AUTOCONTROL_ON)

■ 자율 제어 동작 상태 API

iw_error_t get_autocontrol(auto_control_t *enable)		
iw_error_t iw_get_autocontrol(auto_control_t *enable)		
기능	자율 제어 기능 활성화 상태를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	enable (out)	활성화 상태 0 : 비활성화 (AUTOCONTROL_OFF) 1 : 활성화 (AUTOCONTROL_ON)

■ 시스템 전력모드 설정 API

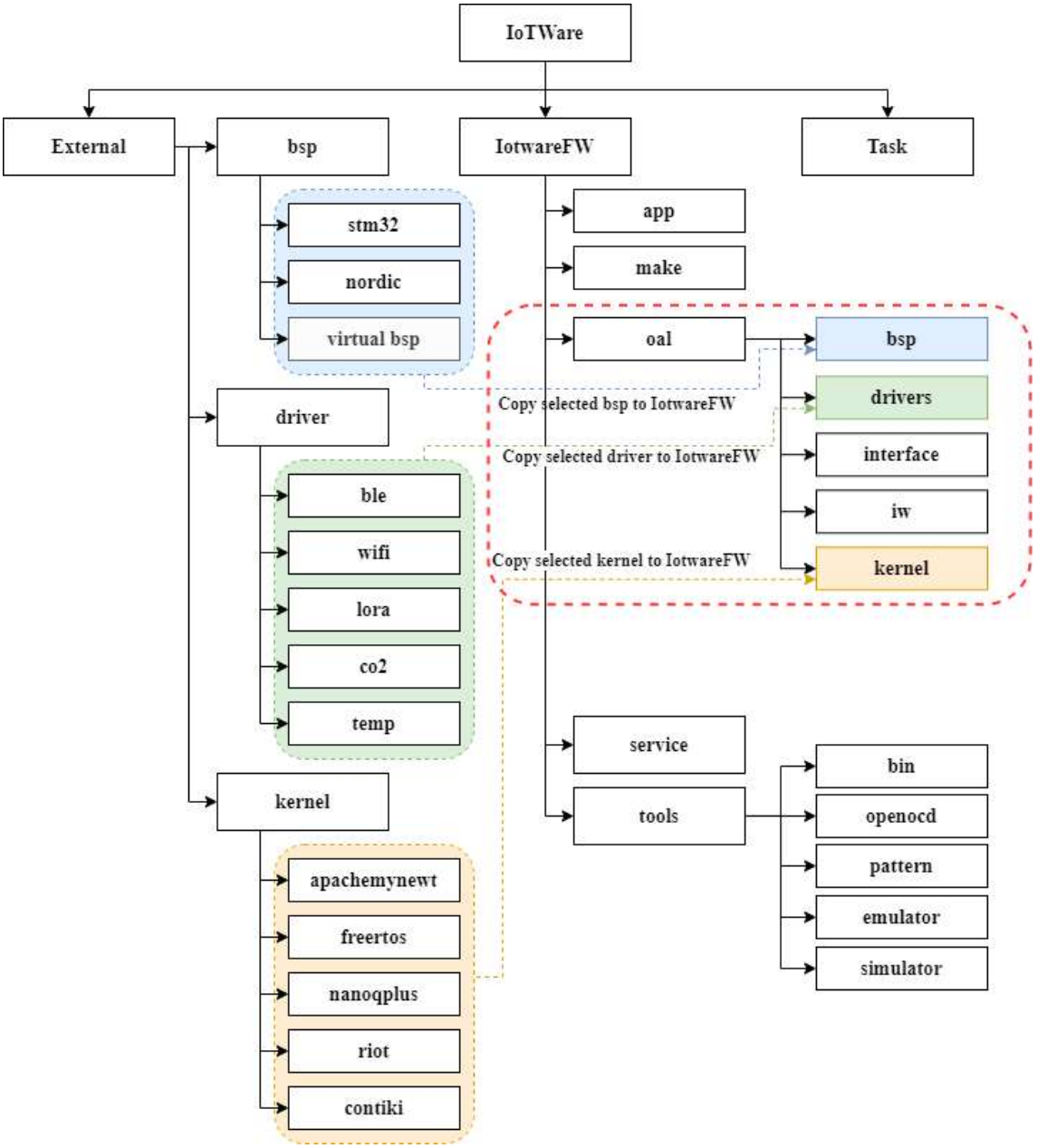
iw_error_t set_pwrmode(power_mode_t pwr_mode)		
iw_error_t iw_set_pwrmode(power_mode_t pwr_mode)		
기능	시스템 전력 모드를 설정한다. ※ 본 기능은 자율 제어 기능이 비활성화 시에만 사용자 설정이 유효하며, 자율 제어 활성화시에는 시스템 상태에 따라 자동으로 변경되며 운용된다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pwr_mode (in)	0 – POW_RUN_MODE : 일반 동작모드 (전력관리 OFF) 1 – POW_DYNAMIC_MODE : 유동적 전력관리 모드 2 – POW_LOW_MODE : 전력 최적 운용 3 – POW_SLEEP_MODE : 슬립 모드 (일반동작 OFF)

■ 시스템 전력모드 반환 API

iw_error_t get_pwrmode(power_mode_t *pwr_mode)		
iw_error_t iw_get_pwrmode(power_mode_t *pwr_mode)		
기능	시스템 전력 모드 정보를 가져온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pwr_mode (out)	0 – POW_RUN_MODE : 일반 동작모드 (전력관리 OFF) 1 – POW_DYNAMIC_MODE : 유동적 전력관리 모드 2 – POW_LOW_MODE : 전력 최적 운용 3 – POW_SLEEP_MODE : 슬립 모드 (일반동작 OFF)

4. OAL 구조 설명

4.1. 전체 Folder Tree



자원 운용 FW 폴더 구조			
External	bsp 와 kerne, driver 총 3개의 폴더로 구성되어 있고, Library DB 에 포함되는 소스 폴더임. 사용자가 IDE를 통해 선택한 bsp, OS, driver 등이 IotwareFW/oal 해당 폴더 아래로 복사되어 운용됨.		
	bsp	bsp 소스	
		stm32	stm32l476 관련 소스
		nordic	nRF52840 관련 소스
		virtual bsp	Emulator용 bsp 관련 소스
	kernel	kernel 소스	
		freertos-8.2.1	freertos-8.2.1 관련 소스 (nordic bsp와 연동 사용)
		freertos-8.2.3	freertos-8.2.3 관련 소스 (virtual bsp와 연동 사용)
		freertos-9.0.0	freertos-9.0.0 관련 소스 (stm32 bsp와 연동 사용)
		nanoqplus	nanoqplus 관련 소스
		riot	riot 관련 소스
		apachemynewt	apachemynewt 관련 소스
		contiki	contiki 관련 소스
	driver	network-ble	ble network 관련 device driver 소스
		network-wifi	wifi network 관련 device driver 소스
		network-lora	lora network 관련 device driver 소스
		sensor-co2	co2 sensor 관련 device driver 소스
		sensor-temp/humi	temperature / humidity sensor 관련 device driver 소스
IotwareFW	kernel/bsp/fw 소스 폴더와 빌드 관련 파일 포함.		
	app	경량형 메시지 소스 포함.	
	make	빌드 시 필요한 파일	
	oal	OAL 소스	
		bsp	bsp 소스 (mcu 관련 소스, external/bsp 폴더로부터 copy)
		drivers	device driver 소스 (external/driver 폴더로부터 copy)
		interface	Task 나 App 프레임에서 사용할 함수 모음
		iw	추상화 소스
		kernel	커널 소스 (os 관련 소스, external/kernel 폴더로부터 copy)
	out	빌드 후 자동 생성되는 파일 및 펌웨어 바이너리	
	service	저전력, 자율 제어 소스	
	tools	fusing, debugging, emulator, simulator 등의 도구 파일	
Task	Task DB 에 포함되는 폴더이고, 마이크로 서비스 FW 소스.		

4.2. OS 지원

4.2.1. 총 7종의 OS 지원

4.2.1.1. 지원목록

- NanoQplus, FreeRTOS, RIOT, ApacheMynewt, Contiki 5종 추상화 (현재)
- Zephyr OS, ARM mbed 2종 추상화 (추가 예정)

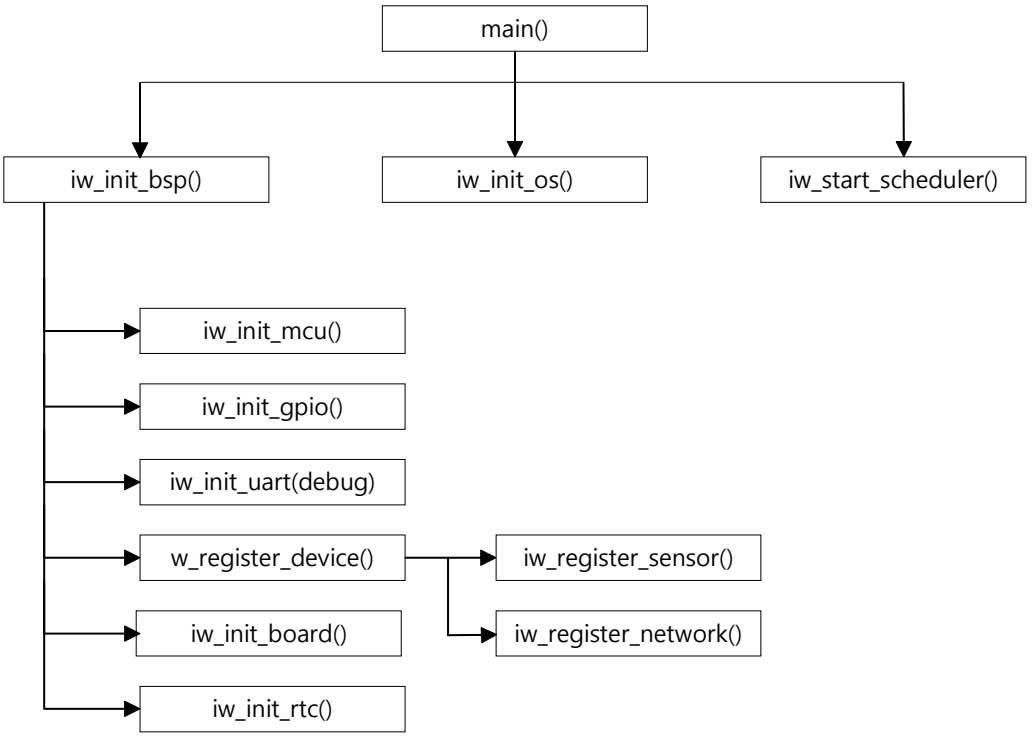
■ 경량형 OS가 제공하는 공통 기능의 OAL 함수를 추상화하여 어플리케이션 개발자는 OS가 변경되어도 기존 코드를 그대로 사용할 수 있도록 설계.

4.2.1.2. 추상화 OS (5종) 버전 정보

추상화에 사용한 OS 5종의 버전과 MCU 정보는 다음과 같다.

OS	버전	MCU	
NanoQplus	3.0.0 (2015.01.06.)	STM32L476	nRF52840
FreeRTOS	8.2.1 (nordic)		
	8.2.3 (virtual bsp)		
	9.0.0 (SMT32)		
RIOT	2018.04 Release		
ApacheMynewt	1.8.0		
Cotiki	3.1		

4.2.1.3. OS 초기화 순서



- iw_init_bsp 함수 : bsp 의존적인 hw 초기화 작업을 주로 진행하며, 호출되는 각 함수의 기능은 아래와 같다.
  - ✓ iw_init_mcu() : mcu 초기화
  - ✓ iw_init_gpio() : mcu의 gpio 기능 enable
  - ✓ iw_init_board() : 보드에 맞게 gpio 설정
  - ✓ iw_init_uart(debug) : debug uart port 초기화

- ✓ iw_register_device() : 사용할 센서, 네트워크, 배터리, LED등의 디바이스를 위한 추가정보 등록 및 MCU peripheral 초기화 및 device driver 맵핑
- ✓ iw_init_rtc() : rtc 초기화 및 초기값 설정

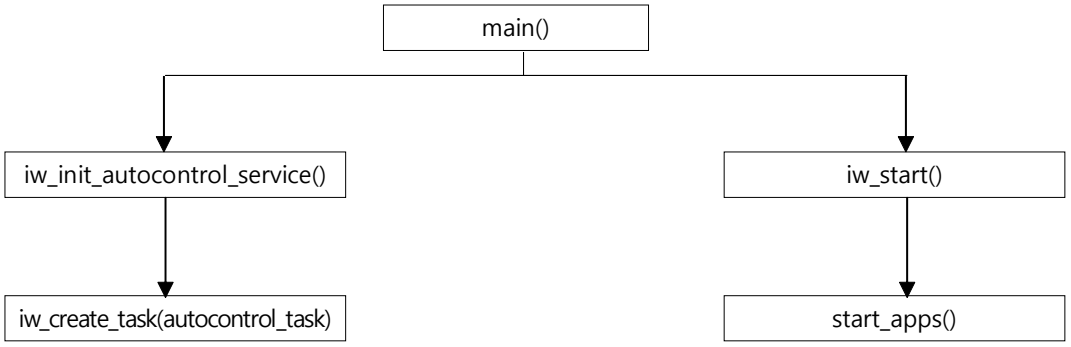
- iw_init_os 함수 : os 관련 초기화 작업을 진행하며, 선택된 kernel 폴더 아래 oal 함수와 연동된다.

/oal/iw/iw_oal_init.c	⇒	/oal/kernel/kernel_oal.c
iw_init_os()		kernel_init_os()

- iw_start_scheduler 함수 : 선택된 os에서 지원하는 scheduler 기능을 기동한다.

/oal/iw/iw_oal_init.c	⇒	/oal/kernel/kernel_oal.c
iw_start_scheduler()		kernel_start_scheduler()

- iw_init_service 함수



- iw_create_task(autocontrol_task) : 자율 제어 서비스 태스크 실행
- start_apps() : application 이나 외부 task 레벨에서 사용할 태스크를 등록하는 함수 실행

4.2.1.4. OAL API 추상화 함수 계층 구조

Application or Task API	Call	Abstraction API	Call	Kernel API
/oal/interface	⇒	/oal/iw	⇒	/oal/kernel
interface_oal.c		iw_oal_fifo.c iw_oal_init.c iw_oal_list.c iw_oal_mem.c iw_oal_mutex.c iw_oal_queue.c iw_oal_sem.c iw_oal_task.c iw_oal_time.c		kernel_oal.c

4.2.1.5. OAL API 추상화

■ task API : 경량형 운영체제의 멀티 타스킹 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 태스크 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_task.c 파일에 함수 정의됨.

■ task API 목록

	interface api	추상화 api
task API list	create_task	iw_create_task
	delete_task	iw_delete_task
	suspend_task	iw_suspend_task
	resume_task	iw_resume_task

■ task 생성 API

iw_task_t create_task(iw_task_fn_t fn,const char *name, unsigned int stack_size, void *arg, unsigned int priority, void *ext)		
iw_task_t iw_create_task(iw_task_fn_t fn,const char *name, unsigned int stack_size, void *arg, unsigned int priority, void *ext)		
기능	task를 생성한다.	
반환 값	생성된 task 의 핸들 값.	
매개 변수	fn (in)	task로 동작할 함수 포인터
	name (in)	task 이름
	arg (in)	시작 함수 인자
	stack_size (in)	스택 사이즈 stack_size * int size 만큼이 스택 사이즈로 할당됨. ex)int size 가 4바이트인 경우, stack_size를 0x100 으로 설정하면, 0x400 (1Kbytes) 로 스택 사이즈가 할당됨.
	priority (in)	task 우선 순위 *freeRTOS 의 경우 0 ~ 7 사이의 값 설정 가능. 0이 우선순위가 가장 낮고, 7이 가장 높음.

■ task 삭제 API

void delete_task(iw_task_t task)		
void iw_delete_task(iw_task_t task)		
기능	task를 삭제한다.	
반환 값	없음.	
매개 변수	task (in)	삭제할 task ID

■ task 중지 API

void suspend_task(iw_task_t task)		
void iw_suspend_task(iw_task_t task)		
기능	실행 중인 task를 중지한다.	
반환 값	없음.	
매개 변수	task (in)	중지할 task ID

■ task 재개 API

void resume_task(iw_task_t task)		
void iw_resume_task(iw_task_t task)		
기능	중지 상태에 있는 task를 재개한다.	
반환 값	없음.	
매개 변수	task	재개할 task ID

■ task 핸들 반환 API

iw_error_t get_task_handle(iw_task_t *handle)		
iw_error_t iw_get_task_handle(iw_task_t *handle)		
기능	현(current) task의 핸들 값을 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	handle [out]	task 의 핸들 값.

■ task 스택 정보 반환 API

iw_error_t get_task_free_stack(iw_task_t handle, unsigned int *free_stack)		
iw_error_t iw_get_task_free_stack(iw_task_t handle, unsigned int *free_stack)		
기능	태스크의 사용 가능한 스택 사이즈를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	handle [in]	task 의 핸들 값.
	free_stack [out]	사용가능한 스택 사이즈.

※ get_task_free_stack()을 호출을 통해 create_thread() 시에 설정하는 stack size 값을 적절히 조절하여 메모리의 낭비를 막을 수 있다.

※ task 핸들 값은 create_thread() 호출 후 리턴 값이나 get_task_handle()을 통해 얻을 수 있다.

■ time (tick) API : 경량형 운영체제의 time 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 time 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_time.c 파일에 함수 정의됨.

■ time (tick) API 목록

time API list	interface api	추상화 api
	get_tick_count	iw_get_tick_count
	get_time_ms	iw_get_time_ms
	delay_us	iw_delay_us
	delay_ms	iw_delay_ms
	sleep	iw_sleep

■ tick 반환 API

iw_tick_t get_tick_count(void)		
iw_tick_t iw_get_tick_count(void)		
기능	스케줄러 시작 이후부터 발생한 tick 카운트 값을 반환한다.	
반환 값	tick 카운트 값	
매개 변수	void	없음

■ ms 시간 반환 API

unsigned int get_time_ms(void)		
unsigned int iw_get_time_ms(void)		
기능	스케줄러 시작 이후부터 발생한 ms 단위 시간 정보 반환한다.	
반환 값	ms 단위 시간 정보	
매개 변수	void	없음

■ 시간 지연 (us) API

void delay_us(unsigned int us)		
void iw_delay_us(unsigned int us)		
기능	설정된 us 단위 시간만큼 프로그램을 지연시킨다.	
반환 값	없음.	
매개 변수	us (in)	대기할 시간 (us)

■ 시간 지연 (ms) API

void delay_ms(unsigned int ms)		
void iw_delay_ms(unsigned int ms)		
기능	설정된 ms 단위 시간만큼 프로그램을 지연시킨다.	
반환 값	없음.	
매개 변수	ms (in)	대기할 시간 (ms)

■ 시간 대기 (ms) API

void sleep(unsigned int ms)		
void iw_sleep(unsigned int ms)		
기능	설정된 ms 단위 시간만큼 대기한다.	
반환 값	없음.	
매개 변수	ms (in)	대기할 시간 (ms)

1.1.1.1.

■ time (rtc) API : 경량형 운영체제의 rtc 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 time (rtc) 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_bsp.c 파일에 함수 정의됨.

■ time (rtc) API 목록

rtc API list	interface api	추상화 api
	(1)	iw_init_rtc
	get_rtc_time	iw_get_rtc_time
	set_rtc_time	iw_set_rtc_time

※외부 task나 app에서는 사용하지 않아 인터페이스 함수는 정의하지 않음.

■ 날짜 및 시간 정보 구조체 정의

```
typedef struct
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t min;
    uint8_t sec;
} DATE_TIME_T;
```

■ 현재 날짜 및 시간 정보 반환 API

bool get_rtc_time(DATE_TIME_T *dt)		
bool iw_get_rtc_time(DATE_TIME_T *dt)		
기능	현재 시간을 반환한다.	
반환 값	성공하면 true, 실패하면 false를 반환한다.	
매개 변수	dt (out)	날짜, 시간 정보

■ 현재 날짜 및 시간 정보 설정 API

bool set_rtc_time(DATE_TIME_T *dt)		
bool iw_set_rtc_time(DATE_TIME_T *dt)		
기능	현재 시간을 설정한다.	
반환 값	성공하면 true, 실패하면 false를 반환한다.	
매개 변수	dt (in)	날짜, 시간 정보

■ memory API : 경량형 운영체제의 memory 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 memory 관련 기능을 수행할 수 있는 API 제공.
※ /IotwareFW/iw/iw_oal_mem.c 파일에 함수 정의됨.

■ memory API 목록

memory API list	interface api	추상화 api
	alloc_mem	iw_malloc
	free_mem	iw_free
	get_memory_status	iw_get_memory_status

■ memory 할당 API

void *alloc_mem(unsigned int size)		
void *iw_alloc(unsigned int size)		
기능	메모리를 할당하여 주소값을 반환한다.	
반환 값	할당받은 메모리의 주소	
매개 변수	size (in)	할당받을 메모리의 바이트 수

■ memory 해제 API

void free_mem(void *ptr)		
void iw_free(void *ptr)		
기능	할당받은 메모리를 해제한다.	
반환 값	없음	
매개 변수	ptr (in)	해제할 메모리의 주소. alloc_mem()을 통해 받은 주소값을 넣어준다.

■ memory 사용량 정보 반환 API

iw_error_t get_memory_status(unsigned int *global_data, unsigned int *total_heap, unsigned int *free_heap)		
iw_error_t iw_get_memory_status(unsigned int *global_data, unsigned int *total_heap, unsigned int *free_heap)		
기능	메모리 사용량을 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	global_data (out)	global 데이타 사용량 (단위 : byte)
	total_heap (out)	총 메모리 힙 사이즈 (단위 : byte)
	free_heap (out)	사용 가능한 메모리 힙 사이즈 (단위 : byte)

※ ₩ow₩iw₩iw_sys.c 에 함수 정의.

■ message queue API : 경량형 운영체제의 message queue 관련 기능 분석을 기반

으로 공통으로 사용 가능한 구성 요소를 유 추하여 message queue 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_queue.c 파일에 함수 정의됨.

■ nanoQplus의 message queue 추상화 구현

■ nanoQplus의 경우 queue의 아이템 사이즈가 'UINT32'로 고정되어 사이즈 변경이 불가한 상태.

■ create_queue 함수 호출시 'item_size'에 맞는 동작을 구현하기 위해 함수를 새로 작성함.

■ nanoQplus가 지원하는 함수에 'item_size'가 적용되도록 소스 변경.

	원 함수	수정 함수
소스 파일 위치	₩kernel₩nos₩kernel₩msgq.c	₩kernel₩nanoqpls_oal.c
queue 생성	msgq_create	msgq_create_ex
queue 제거	msgq_destroy	msgq_destroy_ex
queue 전송	msgq_send	msgq_send_ex
queue 수신	msgq_recv	msgq_recv_ex

■ message queue API 목록

	interface api	추상화 api
queue API list	create_queue	iw_create_queue
	delete_queue	iw_delete_queue
	send_queue	iw_send_queue
	recv_queue	iw_recv_queue

■ message queue 생성 API

iw_queue_t create_queue(unsigned int q_size, unsigned int item_size)		
iw_queue_t iw_create_queue(unsigned int q_size, unsigned int item_size)		
기능	메시지 큐를 생성한다.	
반환 값	할당받은 큐의 핸들 값	
매개 변수	q_size (in)	사용할 최대 아이템 수
	item_size (in)	한 개 아이템의 바이트 수

■ message queue 해제 API

void delete_queue(iw_queue_t q);		
void iw_delete_queue(iw_queue_t q)		
기능	할당받은 큐를 삭제한다.	
반환 값	없음	
매개 변수	q (in)	할당받은 큐의 핸들 값

■ message 전송 API

iw_error_t send_queue(iw_queue_t q, void *ptr, unsigned int tick, iw_task_t receive_task)		
iw_error_t iw_send_queue(iw_queue_t q, void *ptr, unsigned int tick, oal_task_t receive_task)		
기능	큐에 아이템을 전송한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	q (in)	아이템을 전송할 큐의 핸들 값
	ptr (in)	아이템의 주소
	tick (in)	큐가 full 일 경우, 사용 가능할 때까지 기다리는 최대 시간. 단위는 tick. 0으로 설정하면 full 일 경우 바로 리턴.
	receive_task (in)	메시지를 받을 태스크의 ID. (NanoQplus와 RIOT에서 기능을 수행하기 위한 매개 변수.)

■ message 수신 API

iw_error_t recv_queue(iw_queue_t q, void *ptr, int block)		
iw_error_t iw_recv_queue(iw_queue_t q, void *ptr, int block)		
기능	큐로부터 아이템을 수신받는다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	q (in)	아이템을 전송한 큐의 핸들 값
	ptr (out)	아이템을 복사할 버퍼의 주소
	block (in)	0 : 수신된 메시지가 있으면 데이터를 복사한 후 IW_SUCCESS를 리턴하고, 메시지가 없으면 IW_FAIL을 리턴한다. 메시지 수신 대기를 하지 않는다. 1 : 메시지가 수신되기를 기다렸다가 수신되면 리턴한다.

■ semaphore API : 경량형 운영체제의 semaphore 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 memory 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_sem.c 파일에 함수 정의됨.

■ semaphore API 목록

semaphore API list	interface api	추상화 api
	create_sem	iw_create_sem
	delete_sem	iw_delete_sem
	unlock_sem	iw_unlock_sem
	lock_sem	iw_lock_sem

※ NanoQplus는 semaphore 함수를 지원하지 않음.

■ semaphore 생성 API

iw_error_t create_sem(iw_sem_t *sem, unsigned int max_count, unsigned int init_count)		
iw_error_t iw_create_sem(iw_sem_t *sem, unsigned int max_count, unsigned int init_count)		
기능	자원의 한정적 사용을 위한 세마포어를 생성한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	sem (out)	생성된 세마포어 핸들 값
	max_count (in)	최대 자원 공유 개수
	init_count (in)	초기 자원 공유 값

■ semaphore 삭제 API

void delete_sem(iw_sem_t sem)		
void iw_delete_sem(iw_sem_t sem)		
기능	세마포어를 삭제한다.	
반환 값	없음	
매개 변수	sem (in)	삭제할 세마포어의 핸들 값 create_sem()을 통해 받은 핸들 값을 넣어준다.

■ semaphore 획득 API

iw_error_t lock_sem(iw_sem_t sem, unsigned int tick)		
iw_error_t iw_lock_sem(iw_sem_t sem, unsigned int tick)		
기능	자원을 액세스하기 위해 세마포어를 획득한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	sem (in)	획득할 세마포어의 핸들 값
	tick (in)	세마포어를 획득하기 위한 최대 시간

■ semaphore 해제 API

iw_error_t unlock_sem(iw_sem_t sem)		
iw_error_t iw_unlock_sem(iw_sem_t sem)		
기능	세마포어를 해제(반환)한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	sem (in)	반환할 세마포어의 핸들 값

■ mutex API : 경량형 운영체제의 mutex 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 mutex 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_mutex.c 파일에 함수 정의됨.

■ mutex API 목록

mutex API list	interface api	추상화 api
	create_mutex	iw_create_mutex
	delete_mutex	iw_delete_mutex
	unlock_mutex	iw_unlock_mutex
	lock_mutex	iw_lock_mutex

■ mutex 생성 API

iw_error_t create_mutex(iw_mutex_t *mutex)		
iw_error_t create_mutex(iw_mutex_t *mutex)		
기능	자원의 독점적 사용을 위한 뮤텍스를 생성한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	mutex (out)	생성된 뮤텍스 핸들 값

■ mutex 삭제 API

void delete_mutex(iw_mutex_t mutex)		
void iw_delete_mutex(iw_mutex_t mutex)		
기능	세마포어를 삭제한다.	
반환 값	없음	
매개 변수	mutex (in)	삭제할 뮤텍스 핸들 값 create_mutex()를 통해 받은 핸들 값을 넣어준다.

■ mutex 획득 API

iw_error_t lock_mutex(iw_mutex_t mutex)		
iw_error_t iw_lock_mutex(iw_mutex_t mutex)		
기능	자원을 독점하기 위해 뮤텍스를 획득한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	mutex (in)	획득할 뮤텍스 핸들 값

■ mutex 해제 API

iw_error_t unlock_mutex(iw_mutex_t mutex)		
iw_error_t iw_unlock_mutex(iw_mutex_t mutex)		
기능	뮤텍스를 해제(반환)한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	mutex (in)	반환할 뮤텍스 핸들 값

■ fifo API : 경량형 운영체제의 fifo 관련 기능 분석을 기반으로 공통으로 사용 가능

한 구성 요소를 유추하여 list 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_fifo.c 파일에 함수 정의됨.

■ fifo 구조체

iw_fifo_t	
volatile uint32_t put_pos	다음 데이터를 저장할 주소
volatile uint32_t get_pos	읽을 데이터의 주소
uint32_t block_size	한 개의 data의 크기
uint32_t block_count	최대 저장 가능한 data 개수
uint8_t *block_pdata	fifo 의 첫 data 주소

■ fifo API 목록

	interface api	추상화 api
fifo API list	※	iw_fifo_init
	※	iw_fifo_put
	※	iw_fifo_get
	※	iw_fifo_count
	※	iw_fifo_flush

※ not implemented.

■ fifo 초기화 API

iw_error_t iw_fifo_init(iw_fifo_t *pfifo, void *pdata, unsigned int size, unsigned int count)		
기능	fifo 를 초기화한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pfifo (in)	초기화 할 fifo 주소
	pdata	데이터를 저장할 주소
	size	한 개 데이터의 크기
	count	저장할 최대 데이터 수

■ fifo 데이터 저장 API

iw_error_t iw_fifo_put(iw_fifo_t *pfifo, void *pdata)		
기능	fifo에 데이터를 저장한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pfifo (in)	fifo 주소
	pdata (in)	저장할 데이터 주소

■ fifo 데이터 반환 API

iw_error_t iw_fifo_get(iw_fifo_t *pfifo, void *pdata)		
기능	fifo에서 데이터를 가져온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pfifo (in)	fifo 주소
	pdata (out)	가져올 데이터 주소

■ fifo 데이터 반환 API

unsigned int iw_fifo_count(iw_fifo_t *pfifo)		
기능	fifo에서 읽어야 할 데이터 수를 반환한다.	
반환 값	읽어야 할 데이터 수	
매개 변수	pfifo (in)	fifo 주소

■ fifo flush API

iw_error_t iw_fifo_flush(iw_fifo_t *pfifo)		
기능	fifo를 flush한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	pfifo (in)	fifo 주소

※ fifo 구조체 변수 중 put_pos 값을 get_pos값에 넣는다. (모든 저장된 데이터를 읽은 것처럼 동작하게 됨.)

■ list API : 경량형 운영체제의 list 관련 기능 분석을 기반으로 공통으로 사용 가능한 구성 요소를 유추하여 list 관련 기능을 수행할 수 있는 API 제공.

※ /IotwareFW/iw/iw_oal_list.c 파일에 함수 정의됨.

■ list node 구조체

iw_list_node_t	
struct iw_list_node *next	다음 노드 엔트리
struct iw_list_node *previous	이전 노드 엔트리
void *pData	데이터 주소

■ list 구조체

iw_list_t	
iw_list_node_t *head	리스트의 헤드 엔트리
unsigned char nodeNum	리스트의 노드 개수

■ list API 목록

	interface api	추상화 api
list API list	init_list	iw_init_list
	add_list_node	iw_add_list_node
	remove_list_node	iw_remove_list_node

■ list 초기화 API

iw_error_t init_list(iw_list_t *list)		
iw_error_t iw_init_list(iw_list_t *list)		
기능	리스트를 초기화한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	list (in)	초기화 할 리스트 주소

■ list 노드 추가 API

iw_error_t add_list_node(iw_list_t *list, iw_list_node_t *node)		
iw_error_t iw_add_list_node(iw_list_t *list, iw_list_node_t *node)		
기능	리스트에 노드를 추가한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	list (in)	추가할 리스트 주소
	node (in)	추가할 노드 주소

■ list 노드 제거 API

iw_error_t remove_list_node(iw_list_t *list, iw_list_node_t *node)		
iw_error_t iw_remove_list_node(iw_list_t *list, iw_list_node_t *node)		
기능	리스트에서 노드를 제거한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	list (in)	리스트 주소
	node (in)	제거할 노드 주소

■ list macro

IW_SET_LIST_NODE_DATA(node, pData)	노드의 데이터를 설정한다.
IW_GET_LIST_NODE_DATA(node)	노드의 데이터를 반환한다.
IW_GET_HEAD(list)	헤드의 엔트리를 반환한다.
IW_GET_NEXT(node)	다음 노드의 엔트리를 반환한다.
IW_GET_PREVIOUS(node)	이전 노드의 엔트리를 반환한다.
IW_GET_LIST_NODE_NUM(list)	리스트의 노드 개수를 반환한다.

4.3. 네트워크 및 센서 드라이버 지원

4.3.1. 추상화 개요

4.3.1.1. 통신(네트워크) 디바이스 추상화 특징

■ 네트워크 디바이스의 공통 기능을 중심으로 추상화를 진행.

■ 어플리케이션 개발자가 네트워크 디바이스의 종류와는 무관하게, 추상화 함수를 통해 직관적인 디바이스 제어가 가능하도록 설계.

■ 네트워크 디바이스가 변경되어도 코드의 재사용이 가능.

4.3.1.2. 센서 추상화 특징

■ 센서의 공통 기능을 중심으로 추상화.

■ 어플리케이션 개발자가 센서의 종류와는 무관하게, 추상화 함수를 통해 센서를 제어하고, 센서 값을 읽을 수 있도록 설계.

■ 센서가 변경되어도 코드의 재사용이 가능.

4.3.1.3. 인터페이스 추상화 기술

■ 네트워크, 센서 디바이스 및 그 외 디바이스와 MCU와의 인터페이스를 추상화하여 디바이스의 인터페이스 변경 및 제어를 용이하게 할 수 있도록 설계.

■ gpio, uart, i2c, SPI, ADC 등을 인터페이스 추상화 함수로 구현.

4.3.1.4. 그 외 디바이스 추상화 기술

■ 어플리케이션에서 LED 의 제어를 손쉽게 할 수 있도록 LED 기능을 추상화 함수로 구현.

4.3.2. 디바이스 등록

디바이스 등록을 통해, 보드 상태에 맞는 디바이스를 등록하고, 추가 정보를 줄 수 있다.

4.3.2.1. iw_regiseter_device 함수

■ ₩IoTWare₩IotwareFW₩os₩iw₩iw_dev.c 파일에 정의

■ 함수 내에 원하는 디바이스의 등록 함수를 추가할 수 있다.

디바이스	함 수
센서	**iw_error_t iw_register_sensor_(sensor_type_t dev_type, const char *name, if_type_t if_type, uint32_t if_num, void *if_arg, sensor_driver_t *dev_driver)**
네트워크	**iw_error_t iw_register_network_(net_type_t dev_type, const char *name, if_type_t if_type, uint32_t if_num, void *if_arg, net_driver_t *dev_driver)**
LED	iw_error_t iw_register_led(led_color_t color, unsigned char bank, unsigned char gpio)
BATTERY	void iw_register_battery(if_type_t type, if_id_t port, unsigned char bank, unsigned char gpio)

■ 위의 함수를 통해 센서, 네트워크, LED, BATTERY 디바이스를 등록할 수 있다.
■ 함수 호출시 디바이스 타입, 인터페이스 타입과 번호, 드라이버 이름 등의 정보를 입력한다.

■ 네트워크 추가 정보 등록 함수 목록

정보 추가	함수
서버	void iw_register_server_info(char *server_address, unsigned int port)
Wi-Fi ap	void iw_register_wifi_info(char *ssid, char *pw)
LoRa abp	iw_error_t iw_register_lora_abp_info(char *dev_addr, char *nwks_key, char *apps_key)
LoRa otaa	iw_error_t iw_register_lora_otaa_info(char *dev_eui, char *app_eui, char *app_key)
LoRa band	iw_error_t iw_register_lora_band_info(char *band)

■ 추가 정보 등록 함수를 통해 Wi-Fi의 ssid 와 password, LoRa의 abp 나 otaa 정보 및 밴드 정보를 입력할 수 있다.

4.3.3. 네트웍 디바이스 드라이버 API 추상화

4.3.3.1. IoTware 보드의 네트워크 디바이스 정보

Network device	chip	특이 사항
Wi-Fi	esp8266	
LoRa	rak811	firmware ver.2.0 과 ver.3.0 모두 지원하도록 드라이버 설계.

4.3.3.2. 네트워크 디바이스 타입 구조체 정의

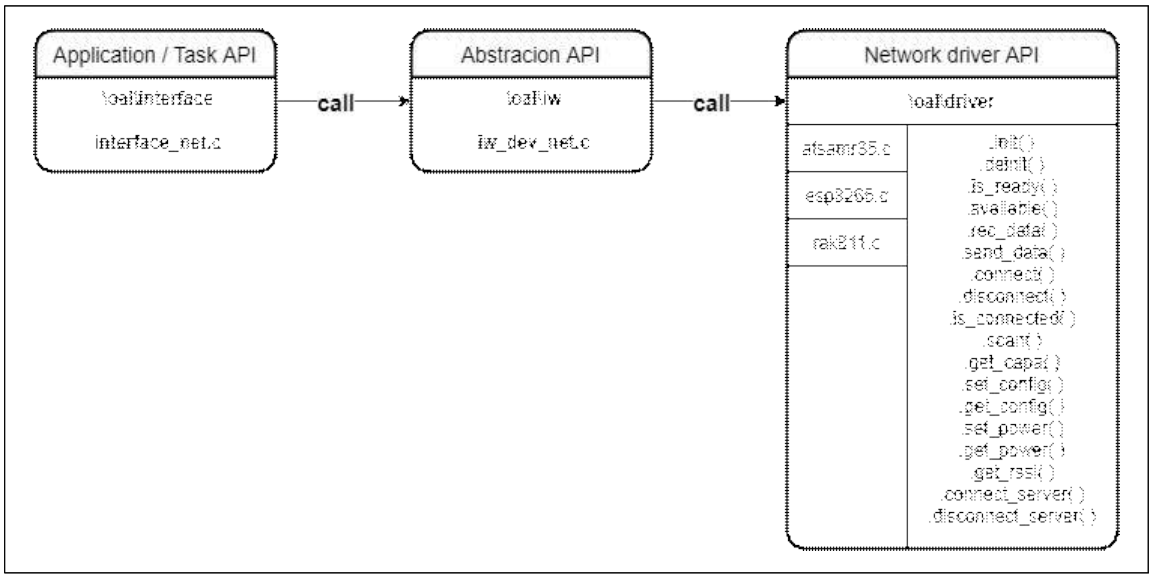
초소형 IoT에서 흔히 사용되고 있는 Wi-Fi, BLE, RoLa 타입을 정의하였다.

```
typedef enum{
        NET_DEV_TYPE_NONE       = 0,
        NET_DEV_TYPE_WIFI       = (1 << 0),
        NET_DEV_TYPE_BLE        = (1 << 1),
        NET_DEV_TYPE_LORA       = (1 << 2),
}net_type_t;
```

4.3.3.3. 네트웍 디바이스 드라이버 함수 정의 (net_func_t)

```
typedef struct{
        iw_error_t (*init)(void *param);
        iw_error_t (*deinit)(void *param);
        iw_error_t (*is_ready)(void *param, dev_init_status_t *status);
        iw_error_t (*available)( void *param, unsigned int *len);
        iw_error_t (*recv_data)( void *param, char *buff, unsigned int *len);
        iw_error_t (*send_data)( void *param, char *buff, unsigned int len);
        int (*connect)(void *param, unsigned int timeout);
        iw_error_t (*disconnect)(void *param);
        iw_error_t (*is_connected)(void *param, int *connected);
        int (*scan)(void * param, int *rssi);
        iw_error_t (*get_capa)(void *param, unsigned char *wake_src);
        iw_error_t (*set_config)(void *param, wake_src_type_t wake_src, void * config);
        iw_error_t (*get_config)(void *param, wake_src_type_t *wake_src, void * config);
        iw_error_t (*set_power)(void *param, net_pwr_mode_t pwr_mode);
        iw_error_t (*get_power)(void *param, net_pwr_mode_t *pwr_mode);
        iw_error_t (*get_rssi)(void *param, int *rssi);
        int (*connect_server)(void *param, char *address, int port, unsigned int timeout);
        int (*disconnect_server)(void *param);
}net_func_t;
```

4.3.3.4. 네트워크 디바이스 드라이버 API 추상화 함수 계층 구조



- 네트워크 드라이버추상화 함수 호출 흐름도 -

※ 개발자가 네트워크 드라이버 추가 가능.

4.3.3.5. 네트워크 디바이스 드라이버 관리 함수 목록

여러 네트워크 디바이스를 관리하기 위한 함수 목록이다.

interface api	추상화 api
none (주1)	iw_register_network
get_registered_net_num	iw_get_registered_net_num
get_all_networks_info	iw_get_all_networks_info
get_network_info	iw_get_network_info
get_network_devid	iw_get_network_devid
get_network_dev_id_by_type	iw_get_network_dev_id_by_type
get_network_dev_id_by_name	iw_get_network_dev_id_by_name

(주1) 외부 태스크나 app에서 사용하지 않아 인터페이스 함수는 정의하지 않음.

■ 네트웍 디바이스 등록 API

iw_error_t iw_register_network (net_type_t dev_type, const char *name, if_type_t if_type, uint32_t if_num, void *if_arg, net_driver_t *dev_driver)		
기능	사용할 네트워크 디바이스를 등록한다.	
반환 값	0 : IW_SUCCESS : 성공 -1 : IW_FAIL : 실패	
매개 변수	type (in)	네트워크 디바이스의 타입
	name (in)	디바이스 이름 *디바이스 드라이버 이름과 일치하여야 한다.
	if_type (in)	MCU와의 인터페이스 타입 1 : I2C 2 : UART 3 : SPI 4 : GPIO 5 : ADC
	if_num (in)	MCU와의 인터페이스 ID 0번부터 시작한다.
	arg (in)	devive 별 확장 option (Reserved)
	dev_driver (in)	네트워크 디바이스 드라이버를 등록

■ 네트웍 디바이스 개수 반환 API

iw_error_t get_registered_nw_num(unsigned char *num)		
iw_error_t iw_get_registered_nw_num(unsigned char *num)		
기능	등록된 네트워크 디바이스의 수를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	num (out)	등록된 네트워크 디바이스의 수

※ 등록된 모든 네트워크 디바이스 수를 알 수 있다.

■ 등록된 모든 네트웍 디바이스 정보 반환 API

iw_error_t get_all_networks_info(registered_net_info_t *info)			
iw_error_t iw_get_all_networks_info(registered_net_info_t *info)			
기능	등록된 모든 네트워크 디바이스의 정보를 반환한다.		
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패		
매개 변수	registered_net_info_t (in)	unsigned char dev_id	네트워크 디바이스 ID
		net_type_t type	네트워크 디바이스 타입
		unsigned char name[]	네트워크 디바이스 이름

※ 등록된 네트워크 디바이스 정보를 모두 반환하기 때문에 'get_registered_nw_num()' 함수를 통해 등록된 디바이스 개수를 확인하고 'info' 버퍼를 설정하여 함수를 호출해야한다.

■ 네트웍 디바이스 정보 반환 API

iw_error_t get_network_info(net_type_t type, char *name, registered_net_info_t *info)			
iiw_error_t iw_get_network_info(net_type_t type, char *name, registered_net_info_t *info)			
기능	type 정보로 해당 네트워크 디바이스 정보를 반환한다.		
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패		
매개 변수	type (in)	네트워크 디바이스 타입	
	name (in)	네트워크 디바이스 이름 동일한 디바이스 타입이 2개 이상 존재하는 경우 'name'과 일치하는 디바이스의 정보 반환.	
	registered_net_info_t (out)	unsigned char dev_id	네트워크 디바이스 ID
		net_type_t type	네트워크 디바이스 타입
		unsigned char name[]	네트워크 디바이스 이름

■ 네트웍 디바이스 ID 반환 API

iw_error_t get_network_devid(net_type_t type, char *name, unsigned char *devid)		
iw_error_t iw_get_network_devid(net_type_t type, char *name, unsigned char *devid)		
기능	type 과 name 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	type (in)	네트워크 디바이스 타입
	name (in)	네트워크 디바이스 이름
	devid (out)	네트워크 디바이스 ID

※ 동일한 디바이스 타입이 2개 이상 존재한다면 'type'과 'name'을 명시해 주어야 하고, 존재하지 않는다면 'name'을 NULL 로 설정하고 호출 가능.

■ 네트워크 디바이스 ID 반환 API

iw_error_t get_network_dev_id_by_type(net_type_t type, uint8_t *dev_id)		
iw_error_t iw_get_network_dev_id_by_type(net_type_t type, uint8_t *dev_id)		
기능	type 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	type (in)	네트워크 디바이스 타입
	devid (out)	네트워크 디바이스 ID

■ 네트워크 디바이스 ID 반환 API

iw_error_t get_network_dev_id_by_name(char *name, uint8_t *dev_id)		
iw_error_t iw_get_network_dev_id_by_name(char *name, uint8_t *dev_id)		
기능	name 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	name (in)	네트워크 디바이스 이름
	devid (out)	네트워크 디바이스 ID

4.3.3.6. 네트워크 디바이스 드라이버 API 목록

네트워크 디바이스들을 제어하기 위한 추상화 API 목록은 다음과 같다.

	interface api	추상화 api
network driver API list	init_network	iw_init_network
	deinit_network	iw_deinit_network
	is_ready_network	iw_is_ready_network
	available_network	iw_available_network
	recv_data_network	iw_recv_data_network
	send_data_network	iw_send_data_network
	connect_network	iw_connect_network
	disconnect_network	iw_disconnect_network
	is_connected_network	iw_is_connected_network
	scan_network	iw_scan_network
	get_capa_network	iw_get_capa_network
	set_config_network	iw_set_config_network
	get_config_network	iw_get_config_network
	set_pwrmode_network	iw_set_pwrmode_network
	get_pwrmode_network	iw_get_pwrmode_network
	get_rssi_network	iw_get_rssi_network
	connect_server	iw_connect_server
	disconnect_server	iw_disconnect_server

※ 외부 태스크나 app에서 사용하지 않아 인터페이스 함수는 정의하지 않음.

■ 네트워크 디바이스 드라이버 초기화 API

iw_error_t init_network(unsigned char dev_id)		
iw_error_t iw_init_network(unsigned char dev_id)		
기능	해당 네트워크디바이스의 드라이버를 초기화한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	초기화 할 네트워크 디바이스 ID

■ 네트워크 디바이스 드라이버 해제 API

iw_error_t deinit_network(unsigned char dev_id)		
iw_error_t iw_deinit_network(unsigned char dev_id)		
기능	해당 네트워크 디바이스 드라이버를 해제한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	해제할 네트워크 디바이스 ID

■ 네트워크 디바이스 드라이버 초기화 완료 정보 반환 API

iw_error_t is_ready_network(unsigned char dev_id, dev_init_status_t *status)		
iw_error_t iw_is_ready_network(unsigned char dev_id, dev_init_status_t *status)		
기능	해당 네트워크 디바이스 드라이버 초기화가 완료되었는지 상태를 반환한다..	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	status (out)	0 - DEV_NOT_INITIALIZED : 드라이버 초기화 미완료. 1 - DEV_INITIALIZED : 드라이버 초기화 완료. 2 - DEV_INIT_ERROR : 드라이버 초기화 실패.

■ 네트워크 수신 데이터 길이 반환 API

iw_error_t available_network(unsigned char dev_id, unsigned int *len)		
iw_error_t iw_available_network(unsigned char dev_id, unsigned int *len)		
기능	수신 받은 데이터 길이를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	len (out)	수신 받은 데이터 길이.

■ 네트워크 수신 데이터 반환 API

iw_error_t recv_data_network(unsigned char dev_id, char *buf, unsigned int *len)		
iw_error_t iw_recv_data_network(unsigned char dev_id, char *buf, unsigned int *len)		
기능	네트워크 디바이스의 데이터를 수신한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	buf (out)	수신 받을 데이터 버퍼 주소
	len (in/out)	(in)수신 가능한 버퍼 길이 (out)수신 받은 버퍼 길이

※ read 버퍼의 길이는 512 바이트로 설정되어 있다. 수신 데이터가 512가 넘어가는 경우 지원되지 않는다.

■ 네트워크 데이터 송신 API

iw_error_t send_data_network(unsigned char dev_id, char *buf, unsigned int len)		
iw_error_t iw_send_data_network(unsigned char dev_id, char *buf, unsigned int len)		
기능	네트워크 디바이스로 데이터를 송신한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	buf (in)	송신할 데이터 버퍼 주소
	len (in)	송신할 데이터 바이트 수

■ 네트워크 연결 API

int connect_network(unsigned char dev_id, unsigned int timeout)		
int iw_connect_network(unsigned char dev_id, unsigned int timeout)		
기능	네트워크에 연결한다. 디바이스 타입에 따라 AP, gateway, host 등에 연결한다.	
반환 값	0 - NET_SUCCESS : 연결 성공 -1 - NET_FAIL : 연결 실패 -2 - NET_TIMEOUT : 타임 아웃 -3 - NET_AP_NOT_FOUND : AP를 찾을 수 없는 경우.(wifi에 한함.)	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	timeout (in)	타임 아웃 (단위: ms)

■ 네트워크 연결 해제 API

iw_error_t disconnect_network(unsigned char dev_id)		
iw_error_t iw_disconnect_network(unsigned char dev_id)		
기능	네트웍 연결을 해제한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID

■ 네트워크 연결 상태 반환 API

iw_error_t is_connected_network(unsigned char dev_id, int *connected)		
iw_error_t iw_is_connected_network(unsigned char dev_id, int *connected)		
기능	네트웍 연결 상태를 반환한다. AP, gateway, host 등에 연결이 되어 있는지에 관한 정보이다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	connected (out)	연결 상태 0 - NET_DISCONNECTED : 연결되어 있지 않음. 1 - NET_CONNECTED : 연결. 2 - NET_GOT_IP : IP 얻어 옴.

■ 네트워크 스캔 API

int scan_network(unsigned char dev_id, int *rssi)		
int iw_scan_network(unsigned char dev_id, int *rssi)		
기능	0 - NET_SUCCESS : 스캔 성공 -1 - NET_FAIL : 스캔 실패 -2 - NET_TIMEOUT : 타임 아웃 -3 - NET_AP_NOT_FOUND : AP를 찾을 수 없는 경우.(wifi에 한함.)	
반환 값	0 - IW_SUCCESS : 성공 1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	rssi (out)	신호 세기 (dbm) *wifi 의 경우 미리 지정한 SSID 의 신호 세기를 얻어 온다.

■ 네트워크 능력 반환 API

iw_error_t get_capa_network(unsigned char dev_id, wake_src_type_t *wake_src)		
iw_error_t iw_get_capa_network(unsigned char dev_id, wake_src_type_t *wake_src)		
기능	네트워크 디바이스가 지원하는 wakeup 소스를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	wake_src (out)	지원하는 wakeup 소스 0 - WAKEUP_SRC_NONE : 없음. 1 - WAKEUP_SRC_INTERRUPT : 인터럽트 2 - WAKEUP_SRC_TIMER : 타이머 인터럽트와 타이머는 디바이스에 따라 모두 지원될 수도 있다. 타이머는 모든 디바이스에 적용 가능하고, 인터럽트는 디바이스가 지원해야 가능함.

■ 네트워크 설정 API

iw_error_t set_config_network(unsigned char dev_id, wake_src_type_t wake_src, void *config)		
iw_error_t iw_set_config_network(unsigned char dev_id, wake_src_type_t wake_src, void *config)		
기능	네트워크 디바이스의 wakeup 소스를 설정한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	wake_src (in)	설정할 wakeup 소스 0 - WAKEUP_SRC_NONE : 없음. 1 - WAKEUP_SRC_INTERRUPT : 인터럽트 2 - WAKEUP_SRC_TIMER : 타이머
	config (in)	reserved

※ get_capa_network()를 호출하여 디바이스가 지원하는 wakeup 소스를 확인 후 설정.

■ 네트워크 설정 반환 API

iw_error_t get_config_network(unsigned char dev_id, wake_src_type_t *wake_src, void *config)		
iw_error_t iw_get_config_network(unsigned char dev_id, wake_src_type_t *wake_src, void *config)		
기능	설정된 wakeup 소스를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	wake_src (out)	설정된 wakeup 소스 0 - WAKEUP_SRC_NONE : 없음. 1 - WAKEUP_SRC_INTERRUPT : 인터럽트 2 - WAKEUP_SRC_TIMER : 타이머
	config (out)	reserved

■ 네트워크 전력 모드 설정 API

iw_error_t set_pwrmode_network(unsigned char dev_id, net_pwr_mode_t pwr_mode)		
iw_error_t iw_set_pwrmode_network(unsigned char dev_id, net_pwr_mode_t pwr_mode)		
기능	네트워크 디바이스의 전력 모드를 설정한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	pwr_mode (in)	설정할 전력 모드 0 - NET_PWR_RUN_MODE : 런 모드 1 – NET_PWR_SLEEP_MODE : 슬립 모드 2 – NET_PWR_LOWPOWER_MODE : 저전력 모드 3 - NET_PWR_AUTO_MODE

■ 네트워크 전력 모드 반환 API

iw_error_t get_pwrmode_network(unsigned char dev_id, net_pwr_mode_t *pwr_mode)		
iw_error_t iw_get_pwrmode_network(unsigned char dev_id, net_pwr_mode_t *pwr_mode)		
기능	네트워크 디바이스의 현재 전력 모드를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크디바이스 ID
	pwr_mode (out)	현재 전력 모드 0 - NET_PWR_RUN_MODE : 런 모드 1 – NET_PWR_SLEEP_MODE : 슬립 모드 2 – NET_PWR_LOWPOWER_MODE : 저전력 모드 3 - NET_PWR_AUTO_MODE

■ 네트워크 신호세기 반환 API

iw_error_t get_rssi_network(unsigned char dev_id, int *rssi)		
iw_error_t get_rssi_network(unsigned char dev_id, int *rssi)		
기능	네트워크 디바이스의 신호세기를 반환한다. wifi 의 경우 미리 설정된 SSID의 신호 세기를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	rssi (out)	신호세기 (단위: dbm)

※ 현재 rssi 값을 가져오는 것이 아니라 마지막 'scan' 명령시 'rssi' 값을 반환한다. 함수 호출시 at command를 통해 현재 rssi 값을 가져오도록 변경 예정이다.

■ 네트워크 서버 연결 API

int connect_server(unsigned char dev_id, char *address, int port, unsigned int timeout)		
int iw_connect_server(unsigned char dev_id, char *address, int port, unsigned int timeout)		
기능	서버에 연결한다.	
반환 값	0 - NET_SUCCESS : 연결 성공 -1 - NET_FAIL : 연결 실패 -2 - NET_TIMEOUT : 타임아웃	
매개 변수	dev_id (in)	네트워크 디바이스 ID
	address (in)	서버 주소
	port (in)	포트 번호
	timeout (in)	연결시 타임 아웃 (단위: ms)

■ 네트워크 서버 연결 종료 API

int disconnect_server(unsigned char dev_id)		
int iw_disconnect_server(unsigned char dev_id)		
기능	서버와의 연결을 종료한다.	
반환 값	0 - NET_SUCCESS : 연결 성공 -1 - NET_FAIL : 연결 실패 -2 - NET_TIMEOUT : 타임아웃	
매개 변수	dev_id (in)	네트워크 디바이스 ID

4.3.3.7. 네트웍 디바이스 추가 정보 등록 API

WiFi나 LoRa에 연결시 추가 정보가 필요하여 만들어진 함수이다. WiFi의 경우 ssid와 pasword를 등록할 수 있고, LoRa의 경우 apb, otaa 정보 등을 등록할 수 있다.

■ WiFi 정보 등록 API

void iw_register_wifi_info(char *ssid, char *pw)		
기능	WiFi 연결에 필요한 정보를 등록한다.	
반환 값	없음.	
매개 변수	ssid	연결할 SSID (최대 길이 : 32bytes)
	pw	연결할 SSID 의 암호 (최대 길이 : 32bytes)

■ WiFi 정보 반환 API

iw_error_t iw_get_wifi_info(wifi_ap_info_t *ap)			
기능	등록된 Wifi 연결 정보를 가져온다.		
반환 값	IW_SUCCESS (0) : 성공 IW_FAIL (-1) : 실패		
매개 변수	ap	ssid	등록된 SSID
		pw	등록된 암호

```
typedef struct{
        char ssid[MAX_NAME_STR_SIZE];
        char pw[MAX_NAME_STR_SIZE];
}wifi_ap_info_t;
```

■ LoRa ABP 정보 등록 API

iw_error_t iw_register_lora_abp_info(char *dev_addr, char *nwks_key, char *apps_key)		
기능	abp 모드 gateway 연결에 필요한 정보를 등록한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_addr	network device address
	nwks_key	network session key
	apps_key	application session key

■ LoRa OTAA 정보 등록 API

iw_error_t iw_register_lora_otaa_info(char *dev_eui, char *app_eui, char *app_key)		
기능	otaa 모드 gateway 연결에 필요한 정보를 등록한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_eui	globally unique device identifier
	app_eui	global application ID
	app_key	AES-128 root key specific to the end-device

■ LoRa 정보 반환 API

iw_error_t iw_get_lora_info(lora_join_info_t *join)			
기능	gateway 에 연결할 정보를 반환한다.		
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패		
매개 변수	join	lora_join_mode_t join_mode	연결 모드 1 - JOIN_ABP : abp 모드 2 - JOIN_OTAA : otaa 모드
		char dev_addr[]	network device address
		char dev_eui[]	globally unique device identifier
		char nwks_key[]	network session key
		char app_eui[]	global application ID
		char apps_key[]	application session key
		char app_key[]	AES-128 root key specific to the end-device

4.3.4. 센서 디바이스 드라이버 API 추상화

4.3.4.1. 센서 타입 정의

초소형 IoT에서 흔히 사용되고 있는 온도, 이산화탄소, 습도, 가속도, 심박, 심전도, 각속도 센서를 정의하였다.

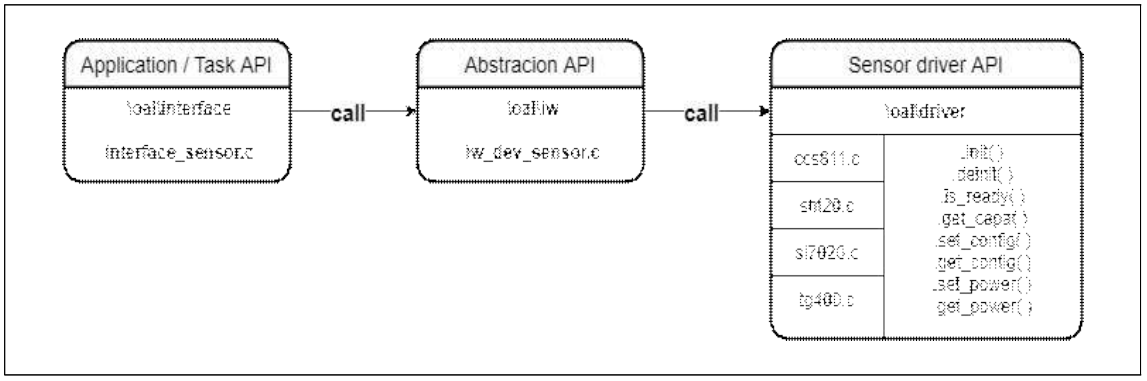
```
typedef enum
{
    SENSOR_TYPE_NONE        = 0,
    SENSOR_TYPE_TMP         = (1 << 0), //온도
    SENSOR_TYPE_CO2         = (1 << 1), //이산화탄소
    SENSOR_TYPE_HUM         = (1 << 2), //습도
    SENSOR_TYPE_ACC         = (1 << 3), //가속도
    SENSOR_TYPE_PPG         = (1 << 4), //심박
    SENSOR_TYPE_ECG         = (1 << 5), //심전도
    SENSOR_TYPE_GYR         = (1 << 6), //각속도
}sensor_type_t;
```

※ 한 개의 디바이스가 여러 타입의 센서를 지원할 수 있다.

4.3.4.2. 센서 드라이버 함수 정의

```
typedef struct{
        iw_error_t (*init)(void *param);
        iw_error_t (*deinit)(void *param);
        iw_error_t (*is_ready)(void *param, dev_init_status_t *status);
        iw_error_t (*read)(void *param, sensor_type_t type, void *buff, unsigned int len);
        iw_error_t (*get_capa)(void *param, wake_src_type_t *wake_src);
        iw_error_t (*set_config)(void *param, wake_src_type_t wake_src, void * config);
        iw_error_t (*get_config)(void *param, wake_src_type_t *wake_src, void * config);
        iw_error_t (*set_power)(void *param, sensor_pwr_mode_t power_mode);
        iw_error_t (*get_power)(void *param, sensor_pwr_mode_t *power_mode);
}sensor_func_t;
```

4.3.4.3. 센서 API 추상화 함수 계층 구조



※ 개발자가 센서 드라이버 추가 가능.

4.3.4.4. 센서 디바이스 드라이버 관리 함수 목록

interface api	추상화 api
none (주1)	iw_register_sensor
get_registered_sensor_num	iw_get_registered_sensor_num
get_all_sensors_info	iw_get_all_sensors_info
get_sensor_info	iw_get_sensor_info
get_sensor_devid	iw_get_sensor_devid
get_sensor_dev_id_by_type	iw_get_sensor_dev_id_by_type
get_sensor_dev_id_by_name	iw_get_sensor_dev_id_by_name

(주1) 태스크나 app에서 사용하지 않아 인터페이스 함수는 정의하지 않음.

■ 센서 디바이스 등록 API

iw_error_t iw_register_sensor_(sensor_type_t dev_type, const char *name, if_type_t if_type, uint32_t if_num, void *if_arg, sensor_driver_t *dev_driver)		
기능	사용할 센서 디바이스를 등록한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	type (in)	네트워크 디바이스의 타입
	name (in)	디바이스 이름 *디바이스 드라이버 이름과 일치하여야 한다.
	if_type (in)	MCU와의 인터페이스 타입 1 : I2C 2 : UART 3 : SPI 4 : GPIO 5 : ADC
	if_num (in)	MCU와의 인터페이스 ID 0번부터 시작한다.
	arg (in)	devive 별 확장 option (Reserved)
	dev_driver (in)	네트워크 디바이스 드라이버를 등록

■ 센서 디바이스 개수 반환 API

iw_error_t get_registered_sensor_num(unsigned char *num)		
iw_error_t iw_get_registered_sensor_num(unsigned char *num)		
기능	등록된 센서 디바이스의 수를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	num (out)	등록된 센서 디바이스의 수

■ 등록된 모든 센서 정보 반환 API

iw_error_t get_all_sensors_info(registered_sensor_info_t *info)			
iw_error_t iw_get_all_sensors_info(registered_sensor_info_t *info)			
기능	등록된 모든 센서의 정보를 반환한다.		
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패		
매개 변수	registered_sensor_info_t (in)	unsigned char dev_id	센서의 디바이스 ID
		sensor_type_t	센서 타입
		unsigned char name[]	센서 이름 (최대 길이 : 20byte)

※ 등록된 모든 센서 정보를 반환하기 때문에 'get_registered_sensor_num()' 함수를 통해 등록된 디바이스 개수를 확인하여 'info' 버퍼를 설정하여 함수를 호출해야한다.

■ 센서 디바이스 정보 반환 API

iw_error_t get_sensor_info(net_type_t type, char *name, registered_sensor_info_t *info)			
iiw_error_t iw_get_sensor_info(net_type_t type, char *name, registered_sensor_info_t *info)			
기능	type 정보로 해당 센서 디바이스 정보를 반환한다.		
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패		
매개 변수	type (in)	센서 디바이스 타입	
	name (in)	센서 디바이스 이름 동일한 디바이스 타입이 2개 이상 존재하는 경우 'name'과 일치하는 디바이스의 정보 반환.	
	registered_sensor_info_t (out)	unsigned char dev_id	센서 디바이스 ID
		sensor_type_t type	센서 디바이스 타입
		unsigned char name[]	센서 디바이스 이름 (최대 길이 : 20byte)

■ 센서 디바이스 ID 반환 API

iw_error_t get_sensor_devid(sensor_type_t type, char *name, unsigned char *devid)		
iw_error_t iw_get_sensor_devid(sensor_type_t type, char *name, unsigned char *devid)		
기능	type 과 name 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	type (in)	센서 디바이스 타입
	name (in)	센서 디바이스 이름
	devid (out)	센서 디바이스 ID

※ 동일한 디바이스 타입이 2개 이상 존재한다면 'type'과 'name'을 명시해 주어야 하고, 존재하지 않는다면 'name'을 NULL 로 설정하고 호출 가능.

■ 센서 디바이스 ID 반환 API

iw_error_t get_sensor_dev_id_by_type(sensor_type_t type, uint8_t *dev_id)		
iw_error_t iw_get_sensor_dev_id_by_type(sensor_type_t type, uint8_t *dev_id)		
기능	type 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	type (in)	센서 디바이스 타입
	devid (out)	센서 디바이스 ID

■ 센서 디바이스 ID 반환 API

iw_error_t get_sensor_dev_id_by_name(char *name, uint8_t *dev_id)		
iw_error_t iw_get_sensor_dev_id_by_name(char *name, uint8_t *dev_id)		
기능	name 정보로 해당 네트워크 디바이스 ID를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	name (in)	센서 디바이스 이름
	devid (out)	센서 디바이스 ID

4.3.4.5. 센서 드라이버 추상화 API 목록

센서 디바이스들을 제어하기 위한 추상화 API 목록은 다음과 같다.

	interface api	추상화 api
sensor driver API list	init_sensor	iw_init_sensor
	deinit_sensor	iw_deinit_sensor
	is_ready_sensor	iw_is_ready_sensor
	read_sensor	iw_read_sensor
	get_capa_sensor	iw_get_capa_sensor
	set_config_sensor	iw_set_config_sensor
	get_config_sensor	iw_get_config_sensor
	set_pwrmode_sensor	iw_set_pwrmode_sensor
	get_pwrmode_sensor	iw_get_pwrmode_sensor

■ 센서 드라이버 초기화 API

iw_error_t init_sensor(unsigned char dev_id)		
iw_error_t iw_init_sensor(unsigned char dev_id)		
기능	해당 센서의 드라이버를 초기화한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	초기화 할 센서의 디바이스 ID

■ 센서 드라이버 해제 API

iw_error_t deinit_sensor(unsigned char dev_id)		
iw_error_t iw_deinit_sensor(unsigned char dev_id)		
기능	해당 센서 드라이버를 해제한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	해제할 센서의 디바이스 ID

■ 센서 드라이버 초기화 완료 정보 반환 API

iw_error_t is_ready_sensor(unsigned char dev_id, dev_init_status_t *status)		
iw_error_t iw_is_ready_sensor(unsigned char dev_id, dev_init_status_t *status)		
기능	해당 센서 디바이스 드라이버 초기화가 완료되었는지 상태를 반환한다..	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	status (out)	0 - DEV_NOT_INITIALIZED : 드라이버 초기화 미완료. 1 - DEV_INITIALIZED : 드라이버 초기화 완료. 2 - DEV_INIT_ERROR: 드라이버 초기화 에러 발생.

■ 센서 데이터 반환 API

iw_error_t read_sensor(unsigned char dev_id, sensor_type_t type, void *buff, unsigned int len)		
iw_error_t iw_read_sensor(unsigned char dev_id, sensor_type_t type, void *buff, unsigned int len)		
기능	센서의 데이터를 읽어 온다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	type (in)	센서의 타입 *센서 한 개의 디바이스가 2개 이상의 센서 기능을 제공하기도 한다. 이런 디바이스의 경우 어떤 타입의 센서 데이터가 필요한지에 대한 정보가 필요하다. 원하는 센서 타입을 'type' 변수에 정의해주면 된다. *한 가지 기능만을 제공하는 디바이스라면 이 매개변수는 의미가 없다.
	buff (out)	데이터를 읽어갈 버퍼 주소
	len	*사용되지 않고 있다. 추후 정의할 예정.

■ 센서 능력 반환 API

iw_error_t get_capa_sensor(unsigned char dev_id, wake_src_type_t *wake_src)		
iw_error_t iw_get_capa_sensor(unsigned char dev_id, wake_src_type_t *wake_src)		
기능	센서가 지원하는 wakeup 소스를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	wake_src (out)	지원하는 wakeup 소스 0 – WAKEUP_SRC_NONE : 없음 1 – WAKEUP_SRC_INTERRUPT : 인터럽트 2 – WAKEUP_SRC_TIMER : 타이머 인터럽트와 타이머는 디바이스에 따라 모두 지원될 수도 있다. 타이머는 모든 디바이스에 적용 가능하고, 인터럽트는 디바이스가 지원해야 가능함.

■ 센서 설정 API

iw_error_t set_config_sensor(unsigned char dev_id, wake_src_type_t wake_src, void *config)		
iw_error_t iw_set_config_sensor(unsigned char dev_id, wake_src_type_t wake_src, void *config)		
기능	센서의 wakeup 소스를 설정한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	wake_src (in)	설정할 wakeup 소스 0 – WAKEUP_SRC_NONE : 없음 1 – WAKEUP_SRC_INTERRUPT : 인터럽트 2 – WAKEUP_SRC_TIMER : 타이머
	config (in)	reserved

※ get_capa_sensor()를 호출하여 디바이스가 지원하는 wakeup 소스를 확인 후 설정.

■ 센서 설정 반환 API

iw_error_t get_config_sensor(unsigned char dev_id, wake_src_type_t *wake_src, void *config)		
iw_error_t iw_get_config_sensor(unsigned char dev_id, wake_src_type_t *wake_src, void *config)		
기능	설정된 wakeup 소스를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	wake_src (out)	설정된 wakeup 소스 0 – WAKEUP_SRC_NONE : 없음 1 – WAKEUP_SRC_INTERRUPT : 인터럽트 2 – WAKEUP_SRC_TIMER : 타이머
	config (out)	reserved

■ 센서 전력 모드 설정 API

iw_error_t set_pwrmode_sensor(unsigned char dev_id, sensor_pwr_mode_t pwr_mode)		
iw_error_t iw_set_pwrmode_sensor(unsigned char dev_id, sensor_pwr_mode_t pwr_mode)		
기능	센서의 전력 모드를 설정한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 디바이스 ID
	pwr_mode (in)	설정할 전력 모드 0 - SENSOR_PWR_RUN_MODE : 런 모드 1 - SENSOR_PWR_SLEEP_MODE : 슬립 모드 2 – SENSOR_PWR_OFF_MODE : 파워 오프 모드 3 – SENSOR_PWR_AUTO_MODE : 오토 모드

■ 센서 전력 모드 반환 API

iw_error_t get_pwrmode_sensor(unsigned char dev_id, sensor_pwr_mode_t *pwr_mode)		
iw_error_t iw_get_pwrmode_sensor(unsigned char dev_id, sensor_pwr_mode_t *pwr_mode)		
기능	센서의 현재 전력 모드를 반환한다.	
반환 값	0 - IW_SUCCESS : 성공 -1 - IW_FAIL : 실패	
매개 변수	dev_id (in)	센서의 device ID
	pwr_mode (out)	현재 전력 모드 0 - SENSOR_PWR_RUN_MODE : 런 모드 1 – SENSOR_PWR_SLEEP_M0.ODE : 슬립 모드 2 – SENSOR_PWR_OFF_MODE : 파워 오프 모드 3 – SENSOR_PWR_AUTO_MODE : 오토 모드

4.3.5. 인터페이스 추상화 API

네트워크 및 센서, LED 모듈과 MCU 간의 인터페이스를 정의하기 위해 주로 사용하는 uart, i2c, gpio 드라이버를 정의하였다.

4.3.5.1. gpio 인터페이스 드라이버

■ gpio 인터페이스 API 목록

	interface api	추상화 api
gpio API list	init_digital	iw_init_digital
	put_digital	iw_put_digital
	get_digital	iw_get_digital
	toggle_digital	iw_toggle_digital

■ gpio 동작 설정 API

void init_digital(uint8_t defined_pin, uint8_t mode)		
void iw_init_digital(uint8_t defined_pin, uint8_t mode)		
기능	gpio를 동작을 설정한다.	
반환 값	없음	
매개 변수	gpio (in)	설정할 gpio 번호 (bsp별 define / bsp_main.h)
	mode (in)	gpio input/output 관련 설정 typedef enum { DIGITAL_INPUT_NOPULL = 0, DIGITAL_INPUT_PULLUP, DIGITAL_INPUT_PULLDOWN, DIGITAL_OUTPUT_PUSHPULL }bsp_digital_mode;

■ gpio 출력 값 설정 API

void put_digital(uint8_t defined_pin, uint8_t value)		
void iw_put_digital(uint8_t defined_pin, uint8_t value)		
기능	gpio 의 출력 값을 설정한다.	
반환 값	없음	
매개 변수	gpio (in)	설정할 gpio 번호 (bsp별 define / bsp_main.h)
	val (in)	설정 값 typedef enum { DIGITAL_LOW = 0, DIGITAL_HIGH }bsp_digital_value;

■ gpio 값 반환 API

uint8_t get_digital(uint8_t defined_pin)		
uint8_t iw_get_digital(uint8_t defined_pin)		
기능	gpio 의 현재 값을 반환한다.	
반환 값	uint8_t value (gpio 값) typedef enum { DIGITAL_LOW = 0, DIGITAL_HIGH }bsp_digital_value;	
매개 변수	gpio (in)	설정할 gpio 번호 (bsp별 define / bsp_main.h)

■ gpio 값 반전 API

uint8_t toggle_digitaldefined_pin)		
uint8_t iw_toggle_digitaldefined_pin)		
기능	gpio 의 현재 값을 반전한다. (Low --> High / High --> Low)	
반환 값	없음	
매개 변수	gpio (in)	설정할 gpio 번호 (bsp별 define / bsp_main.h)

4.3.5.2. I2C 인터페이스 드라이버

■ I2C 인터페이스 API 목록

I2C API list	추상화 api
	int iw_init_i2c
	int iw_deinit_i2c
	int iw_transfer_i2c

■ I2C 드라이버 초기화 API

int iw_init_i2c(unsigned char port)		
기능	I2C 드라이버를 초기화한다.	
반환 값	없음	
매개 변수	port (in)	초기화할 I2C 포트 번호

■ I2C 드라이버 해제 API

int iw_deinit_i2c(unsigned char port)		
기능	I2C 드라이버를 해제한다.	
반환 값	없음	
매개 변수	port (in)	해제할 I2C 포트 번호

■ I2C 통신 API

int iw_transfer_i2c(iw_i2c_msg *msg, int num)			
기능	I2C를 통해 데이터를 전송하고 받는다.		
반환 값	없음		
매개 변수	msg (in/out)	unsigned char port	I2C 포트 번호
		unsigned char addr	I2C 통신할 디바이스 주소
		unsigned char tx_len	송신 데이터의 바이트 수
		unsigned char *tx_buf	송신 데이터 버퍼 주소
		unsigned char rx_len	수신 받을 데이터 바이트 수
		unsigned char *rx_buf	수신 받을 데이터 버퍼
	num (in)	전송할 데이터 수	

4.3.5.3. UART 인터페이스 드라이버

■ UART 인터페이스 API 목록

UART API list	추상화 api
	int iw_init_uart
	int iw_deinit_uart
	int iw_putc_uart
	int iw_getc_uart

■ UART 드라이버 초기화 API

int iw_init_uart(int port , int baudrate)		
기능	UART 드라이버를 초기화한다.	
반환 값	성공이면 '0', 실패면 '-1'	
매개 변수	port (in)	초기화할 UART 포트 번호
	baudrate (in)	통신 속도

■ UART 드라이버 해제 API

int iw_deinit_uart(int port)		
기능	UART 드라이버를 해제한다.	
반환 값	성공이면 '0', 실패면 '-1'	
매개 변수	port (in)	해제할 UART 포트 번호

■ UART 데이터 전송 API

unsigned int iw_putc_uart(int port, char data)		
기능	UART 를 통해 1byte 데이터를 전송한다.	
반환 값	성공이면 '0', 실패면 '-1'	
매개 변수	port (in)	UART 포트 번호
	data (in)	전송할 데이터

■ UART 데이터 수신 API

unsigned intiw_getc_uart(int port, char *data)		
기능	UART 를 통해 1byte 데이터를 수신한다.	
반환 값	성공이면 '0', 실패면 '-1'	
매개 변수	port (in)	UART 포트 번호
	data (out)	수신할 데이터 버퍼 주소