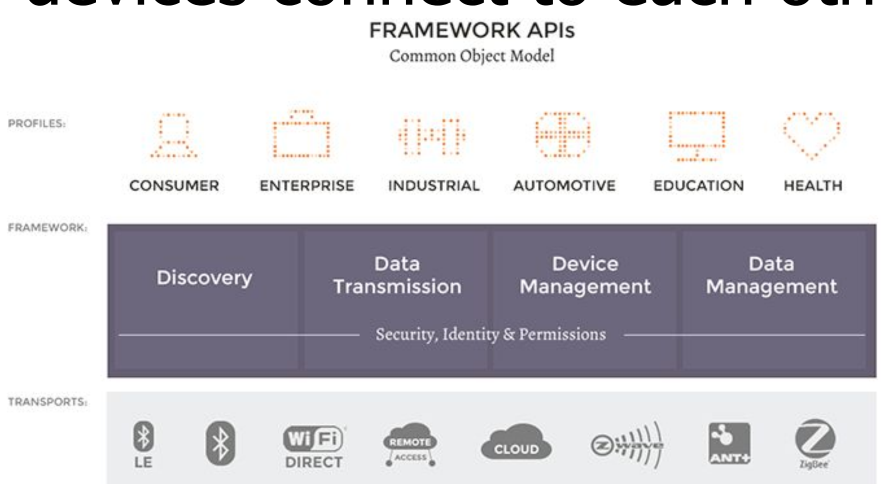
IOTIVITY

UNIT – III Part- 3

IoTivity

- An open source framework and SDK for building IoT applications
  - Linux Foundation Collaborative Project
  - Supported by Open Interconnect Consortium(OIC)
- Goal
  - A new standard by which billions of wired and wireless devices connect to each other and to the Internet



Layered Architecture for IoTivivty

Primitive Service

C++ API

C Stack

Security

Connectivity Abstraction

Multi-Bearer Support

IoTivity supports

- Bluetooth Low Energy

- Bluetooth EDR (Enhanced Data Rate) using RFCOMM

- Dual IPv4/v6 stack

- XMPP (remote access connectivity) hidden in the connectivity abstraction layer.

- built in by default support for other protocols can be added via primitive services.

Connectivity Abstraction

Provides a common platform for all bearers, with the following functionality:

- Transport-specific functionality that's specific to each bearer.
- Listen server to receive multicast packets for resource discovery.
- It also includes functionality to stop receiving resource discovery requests.
- Low-power devices can use this to publish their resources to a resource directory and avoid participating in active discovery to save energy.
- Send functionality for direct communication with other devices and communication across the network.
- Read functionality to get data through network interfaces.
- Specific network information, like addresses of the interfaces

Security

IoTivity security is provided at two layers: transport and application.

- The transport layer - security through the encryption of packets.
- IoTivity security relies on DTLS (Datagram Transport Layer Security) to provide packet to packet encryption.
- The application layer provides security through the use of an Access Control List (ACL) to control access to resources.

C Stack

- OIC representation and payload are defined in the C Stack.

  - capable of communicating directly with the security and connectivity abstraction layers.
- Payload handling via Concise Binary Object Representation – CBOR (RFC 7049).

  - it does not require the encoding of binary data as base64; data directly in binary format.

  - It can handle all JSON-related data.
- The protocol used in this layer is the Constrained Application Protocol – CoAP (RFC 7252).

C++ API

- This is a thin layer that exposes the C stack API to other modules.
- intended for the server and clients to use this API to write the IoT application on top of the C stack.
- used primarily on higher-end client devices such as mobile phones.
- to write IoT applications for the user to interact with.
- All of the client-side communication is defined in the C++ API.

IoTivity Features

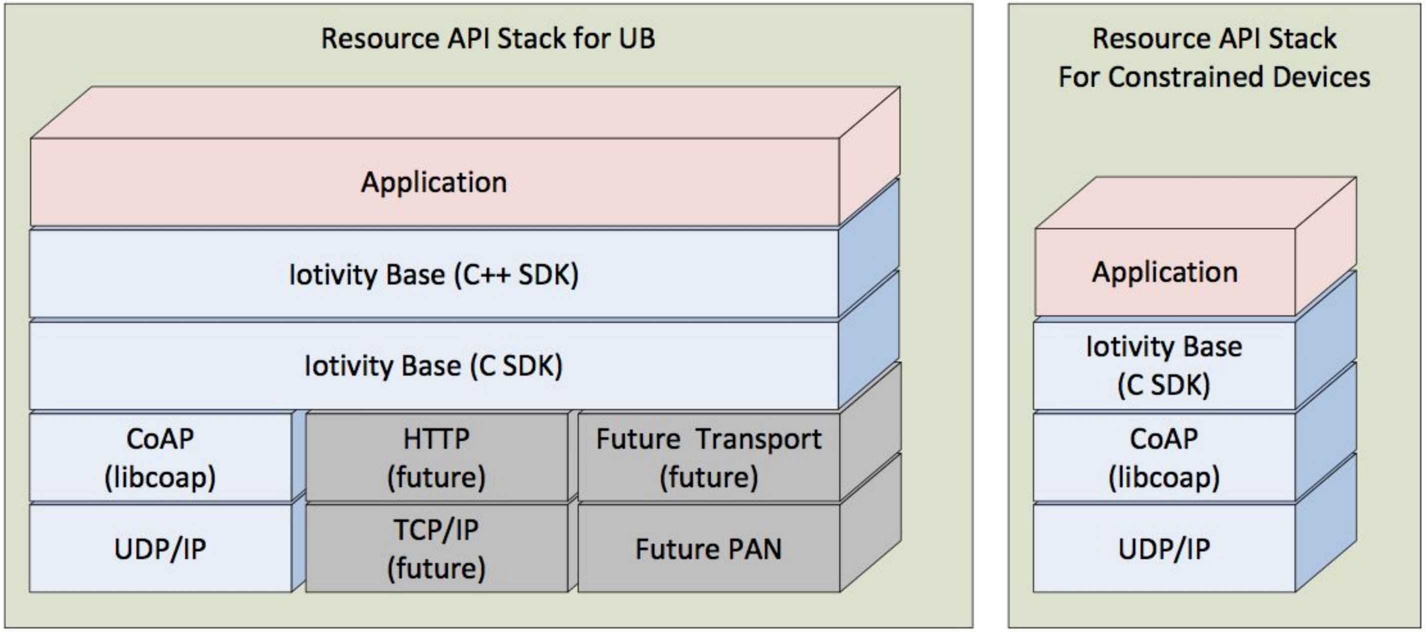
- Base
  - Discovery & Connectivity
  - Resource Management
- Services
  - Soft(Virtual) Sensor Manager
  - Protocol Plugin Manager
  - Things Manager
  - Notification Manager(Resource Offloading)
  - Control Manager(Smart Home Protocol)
- Target: Generic Linux, Tizen, Yocto, Android
- Source code is managed in Gerrit review server

IoTivity SW Stack

- IoTivity Services
  - Soft Sensor, Protocol Plugin, Things, Notification, Control Managers
- Resource API
  - based on OIC Resource Model
  - Interface between IoTivity service and base
- IoTivity Base
  - Abstract connectivity methods to Resource API
  - IoTivity base is included in service process and application process as a library

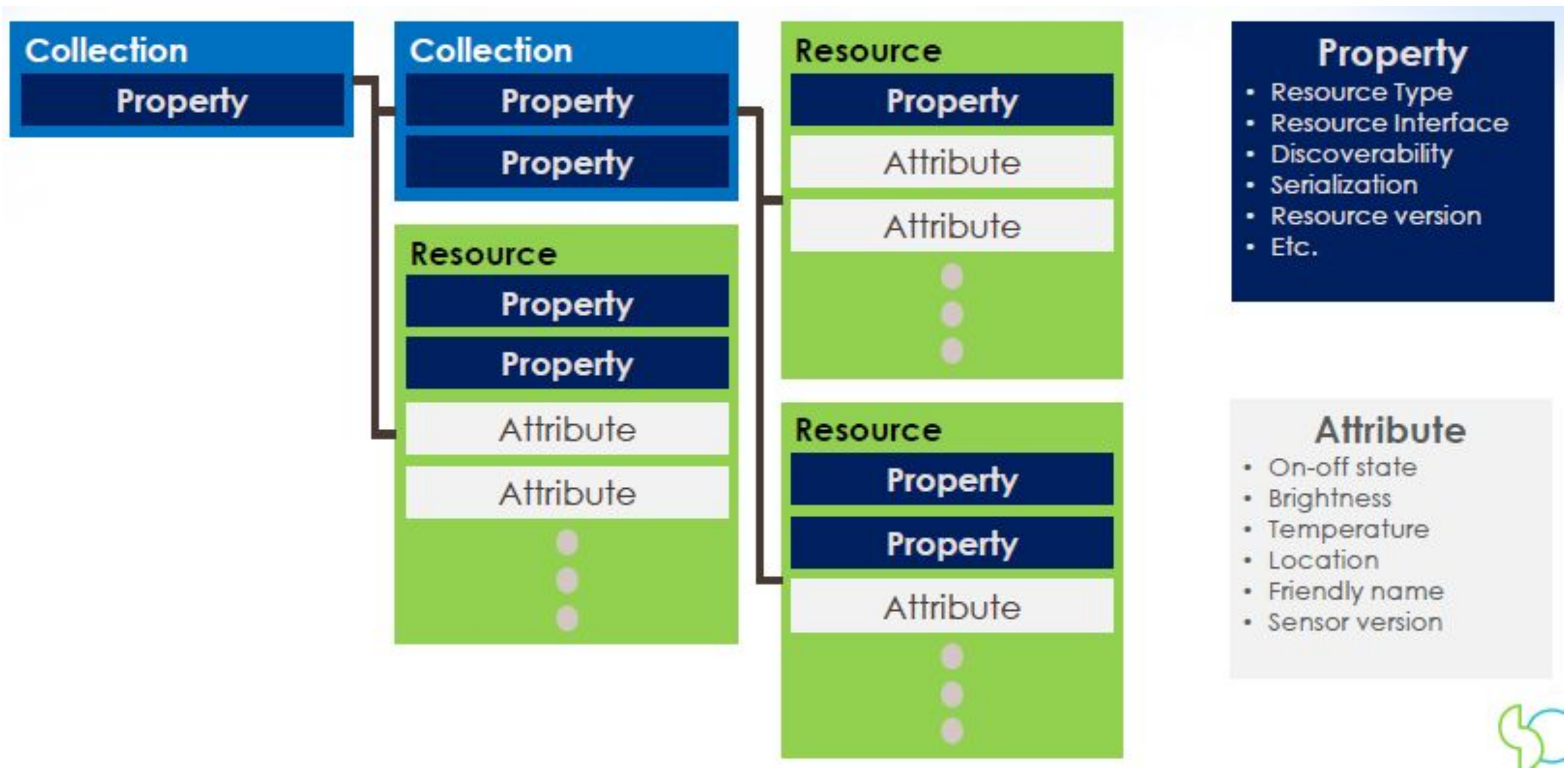
IoTivity SW Stack: Resource API

- Interface between service/app and base
- Support for constrained devices
  - Provide only C SDK, use only CoAP-UDP/IP protocol
  - ex. Arduino



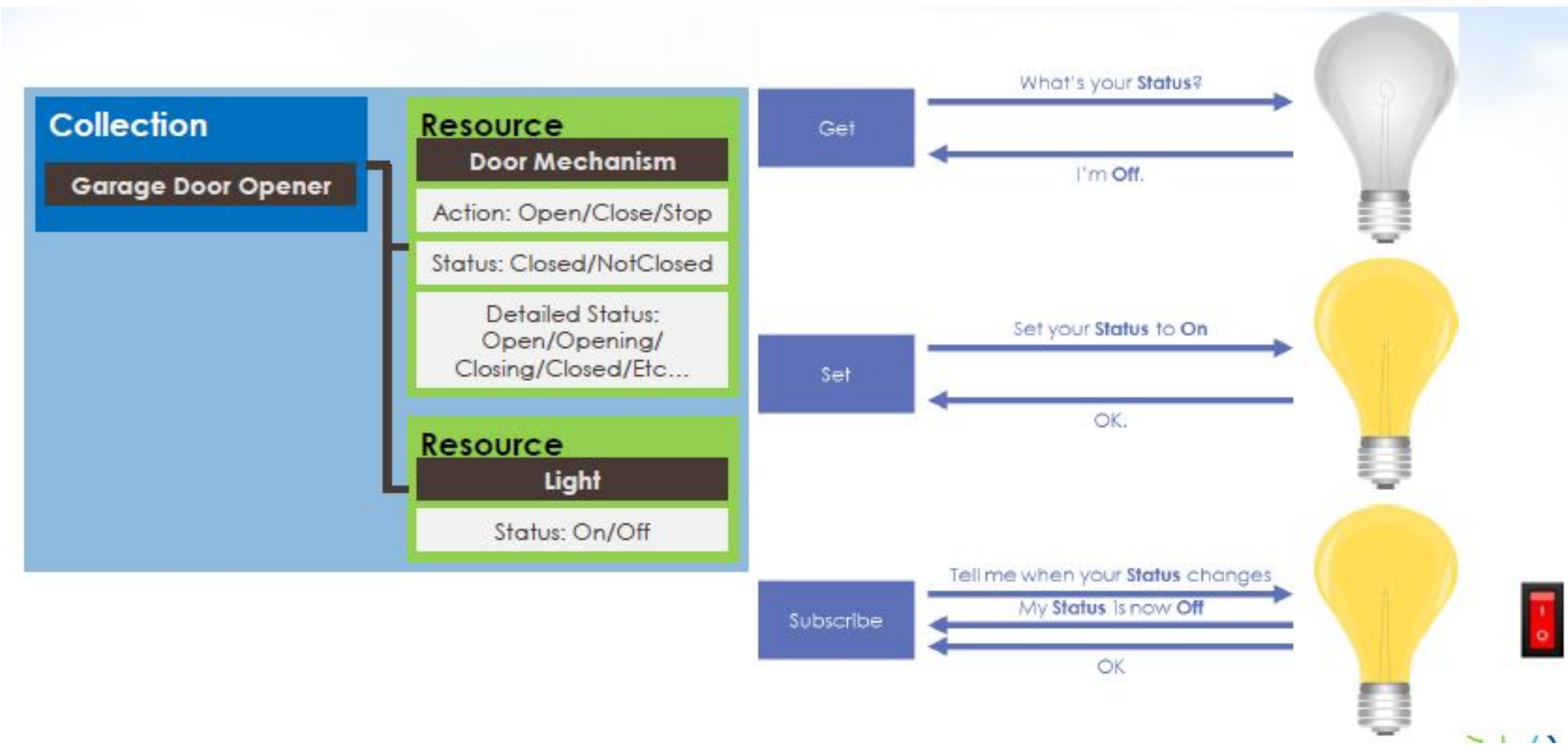
OIC Resource Model

- Collection – Resource – Property/Attribute



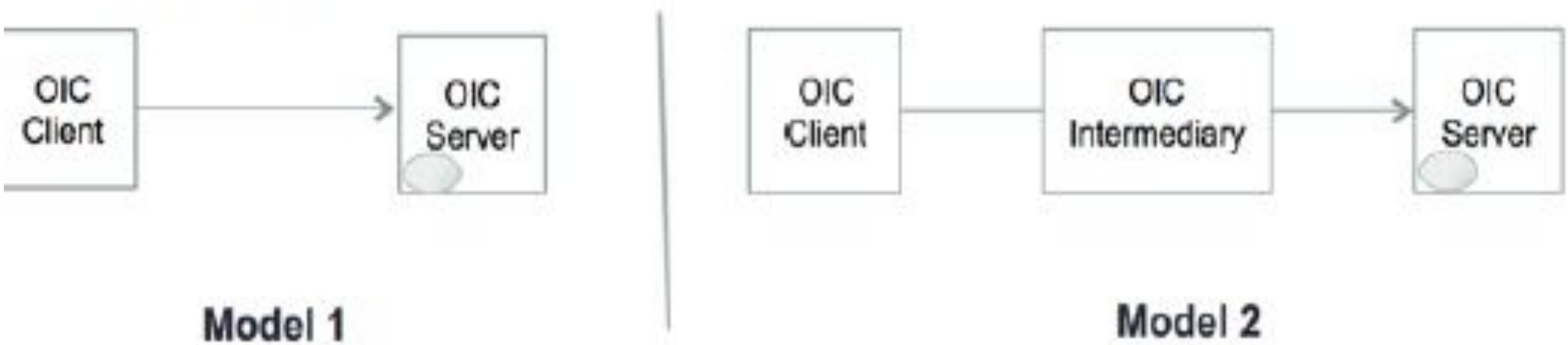
OIC Resource Model

- Behaviors on Resource Model
  - Finding a resource
  - Querying, setting and observing resource state

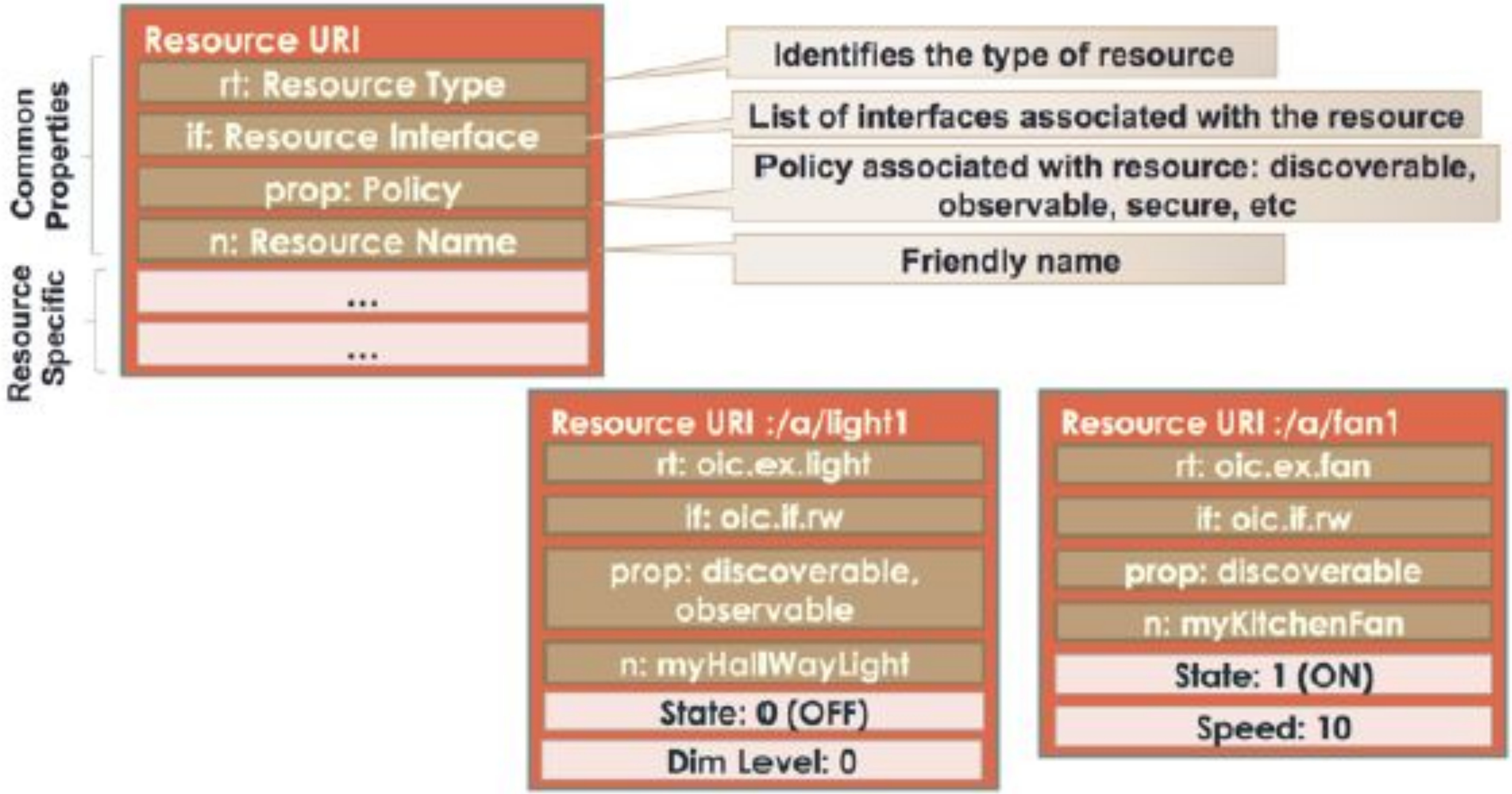


IoTivity resource Model

- RESTful design -> Things modeled as resources
- Server role: Exposes hosted resources
- Client role: Accesses resources on a server
- Intermediary role: Bridges messaging between client and server

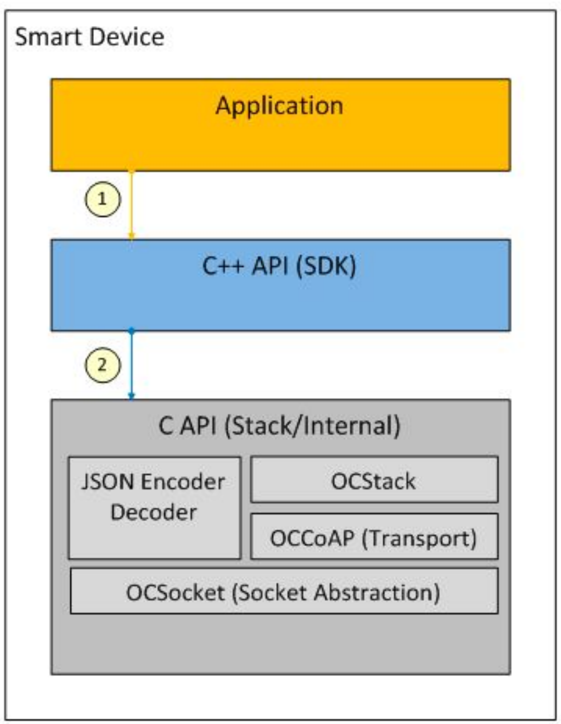


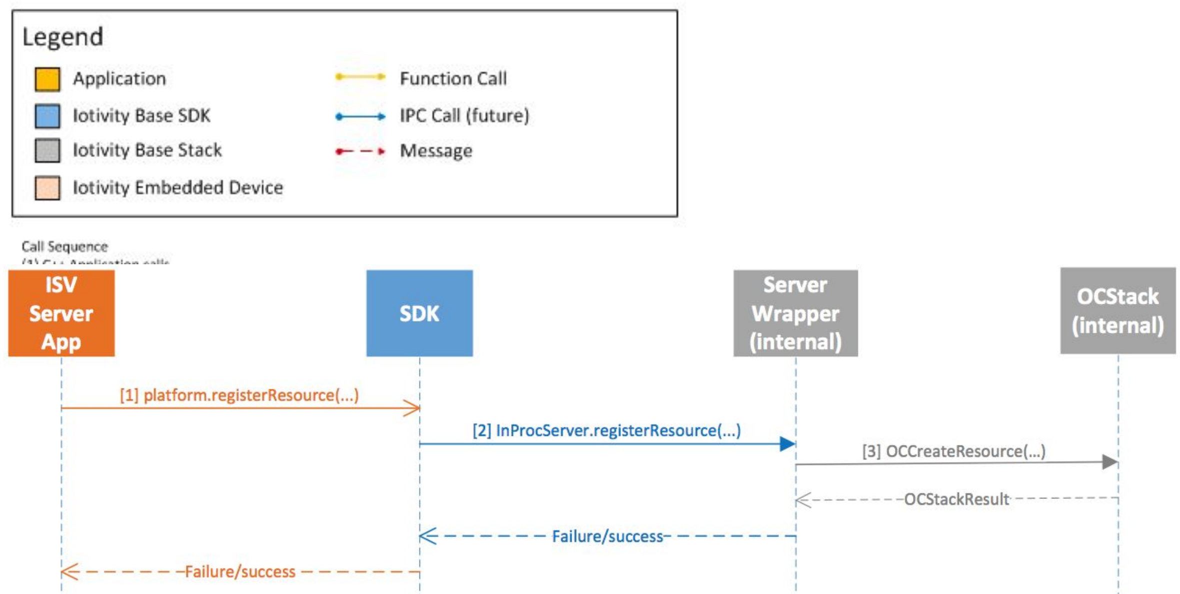
Conti….


Common Properties
Resource Specific
Resource URI
rt: Resource Type
if: Resource Interface
prop: Policy
n: Resource Name
...
...
Identifies the type of resource
List of interfaces associated with the resource
Policy associated with resource: discoverable, observable, secure, etc
Friendly name
Resource URI :/a/light1
rt: oic.ex.light
if: oic.if.rw
prop: discoverable, observable
n: myHallWayLight
State: 0 (OFF)
Dim Level: 0
Resource URI :/a/fan1
rt: oic.ex.fan
if: oic.if.rw
prop: discoverable
n: myKitchenFan
State: 1 (ON)
Speed: 10

IoTivity Base Flow

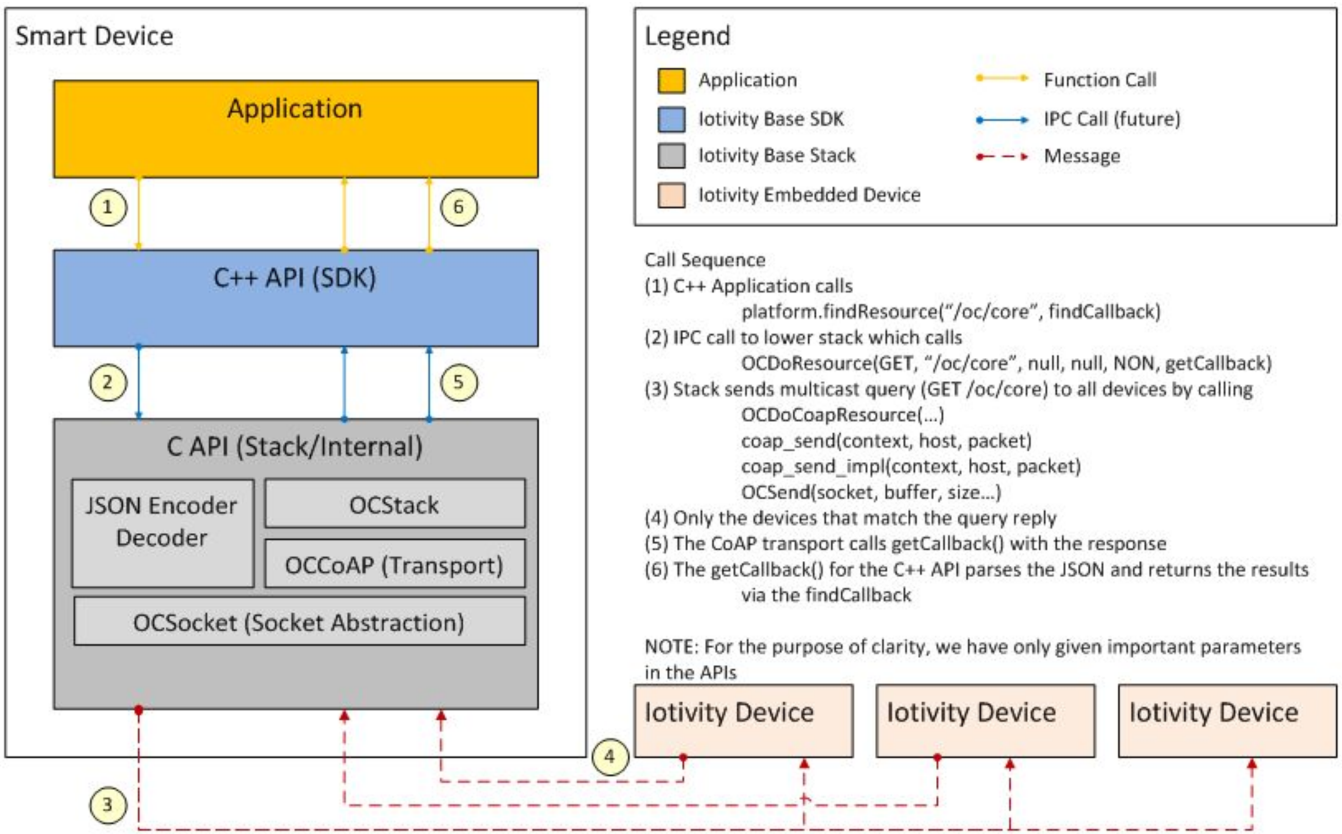
- Registering a Resource
  - Given a service running on port 5683 in a device at IP address 192.168.1.1,
  - If the application registers a resource with a URI path "/light/1",
  - The resulting fully qualified URI "oc://192.168.1.1:5683/light/1"





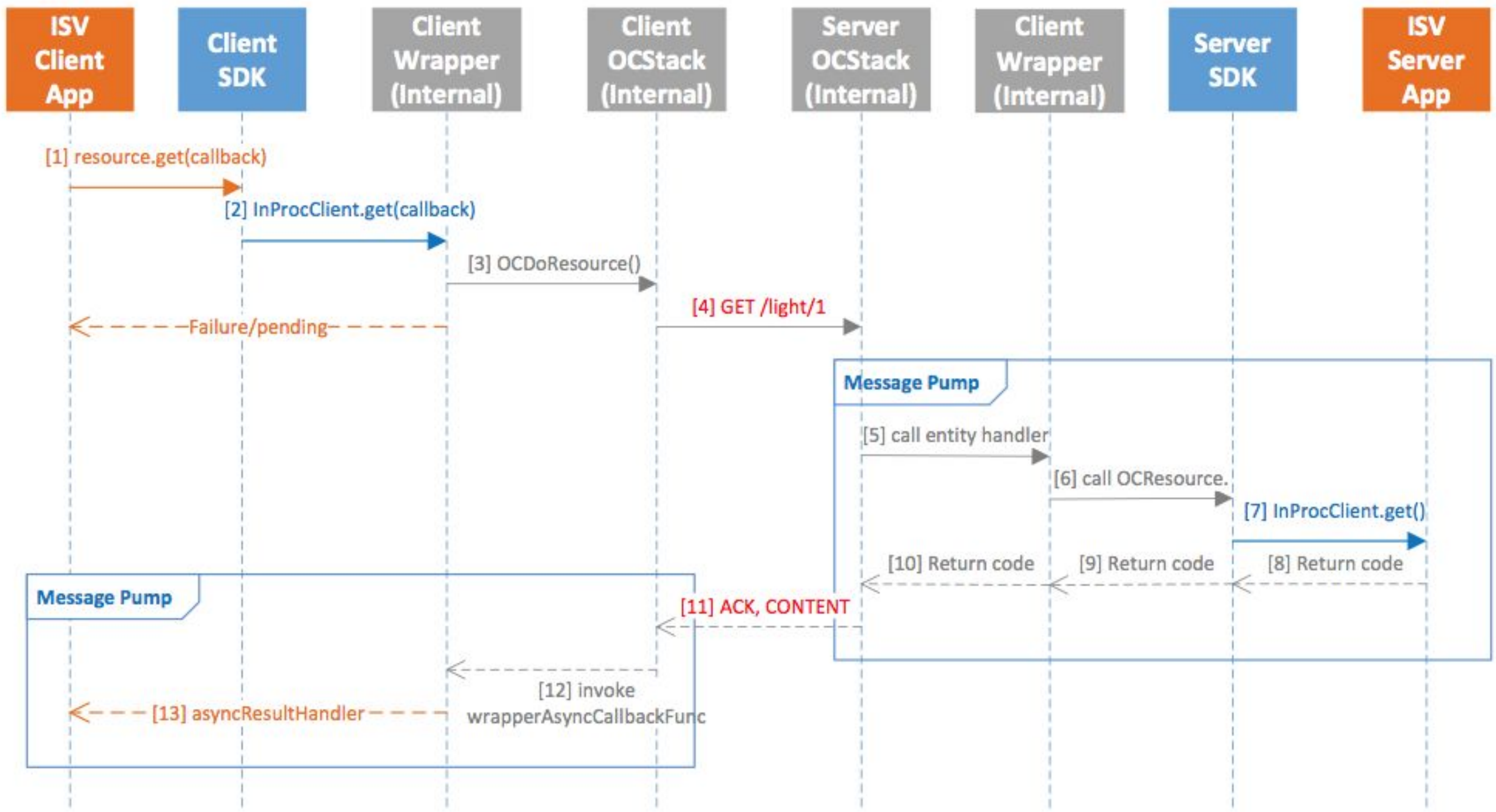
IoTivity Base Flow

- Finding a resource
  - returns all resources of given type on the network service



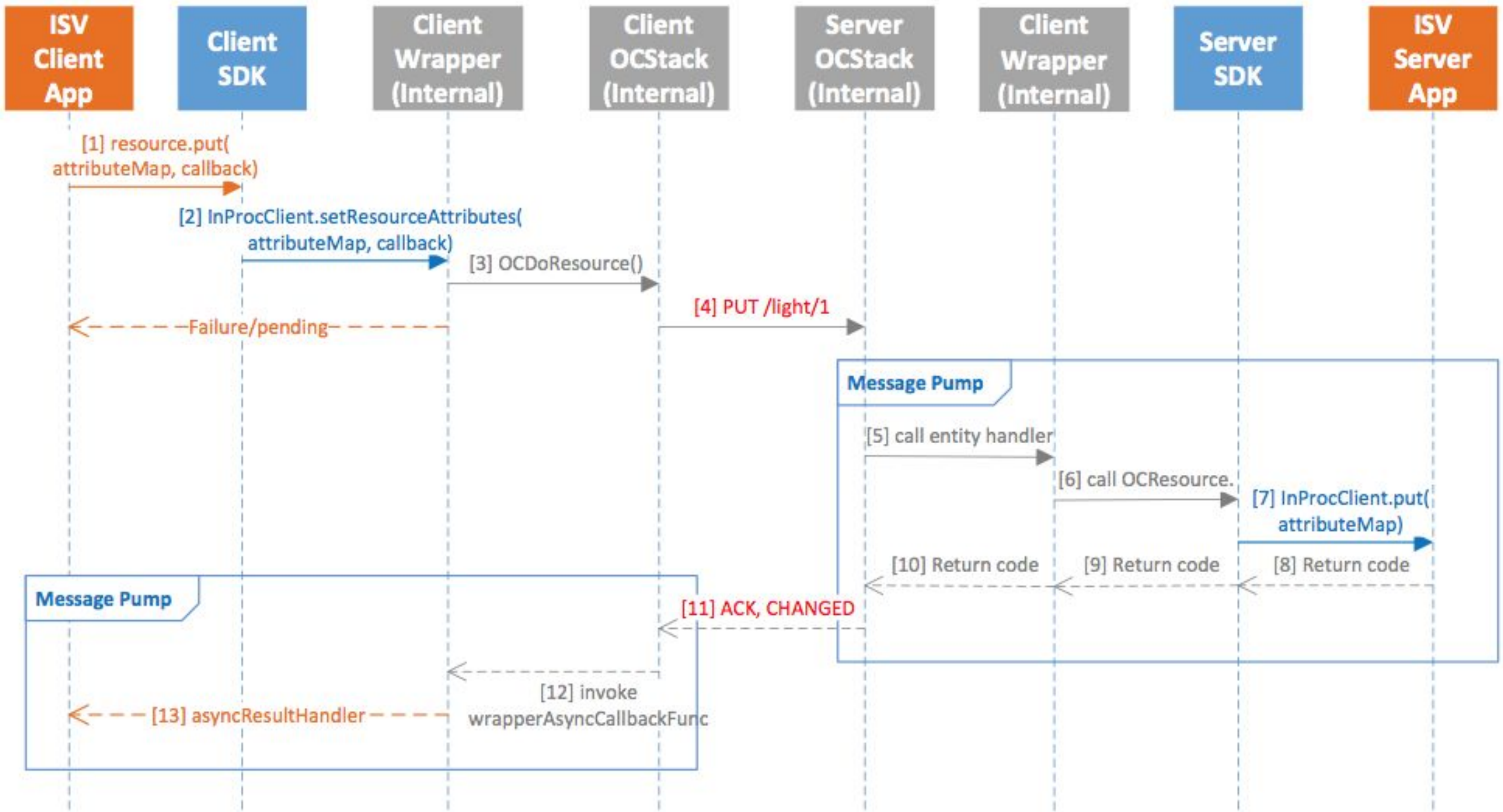
IoTivity Base Flow

- Querying resource state (GET)



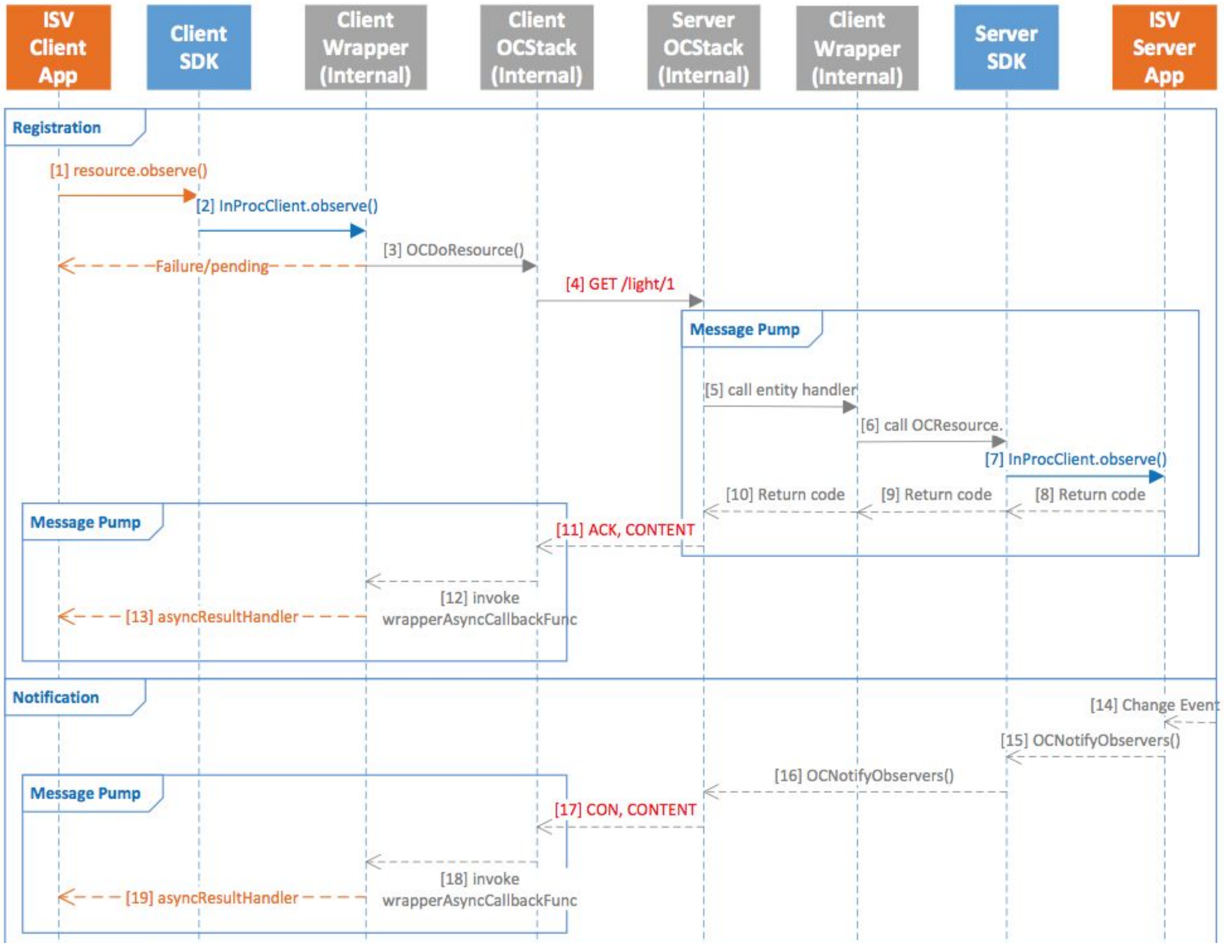
IoTivity Base Flow

- Setting resource state (PUT)



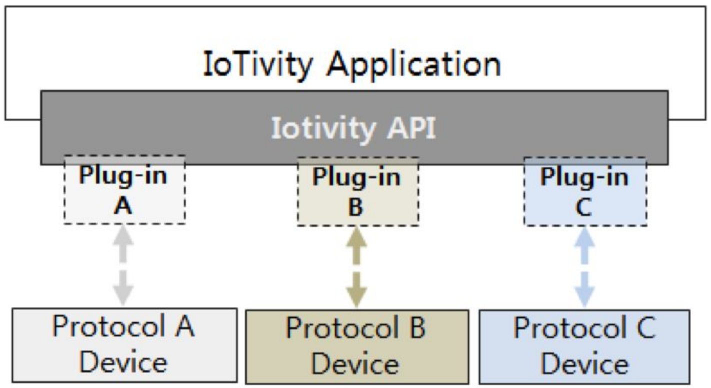
IoTivity Base Flow

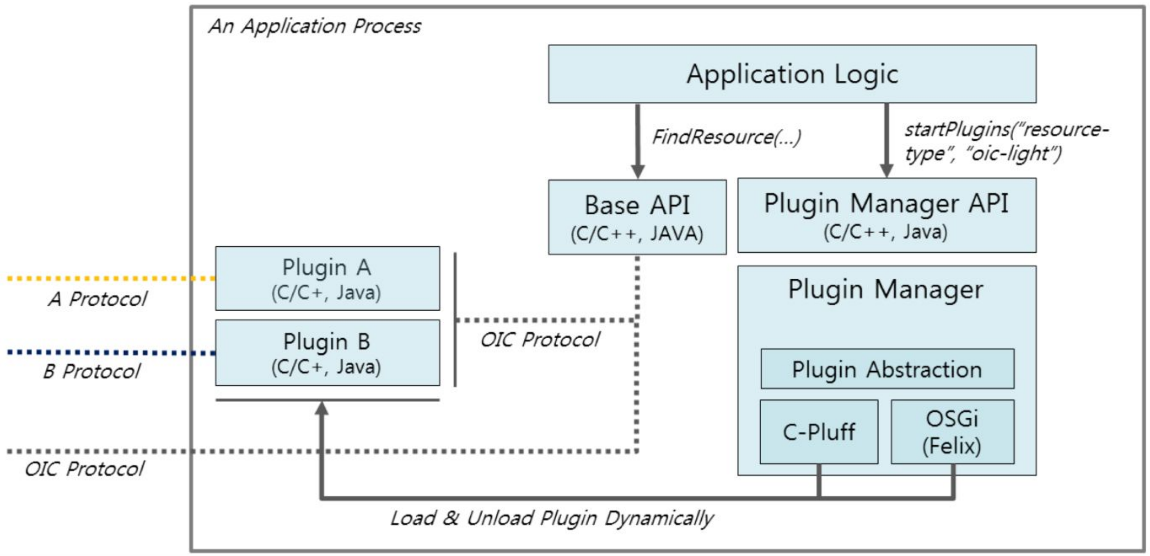
- Observing resource state



Service: Protocol Plug-in Manager

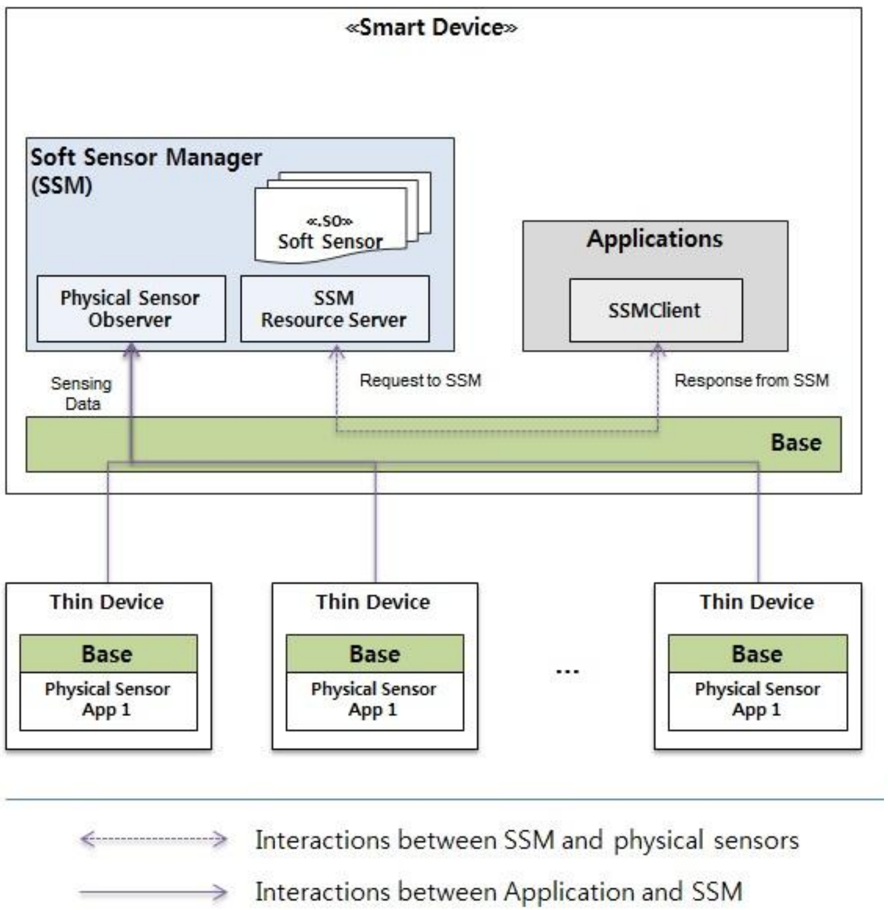
- Provides mechanism to represent non-OIC protocols within the OIC framework
- MQTT is supported as a protocol plug-in

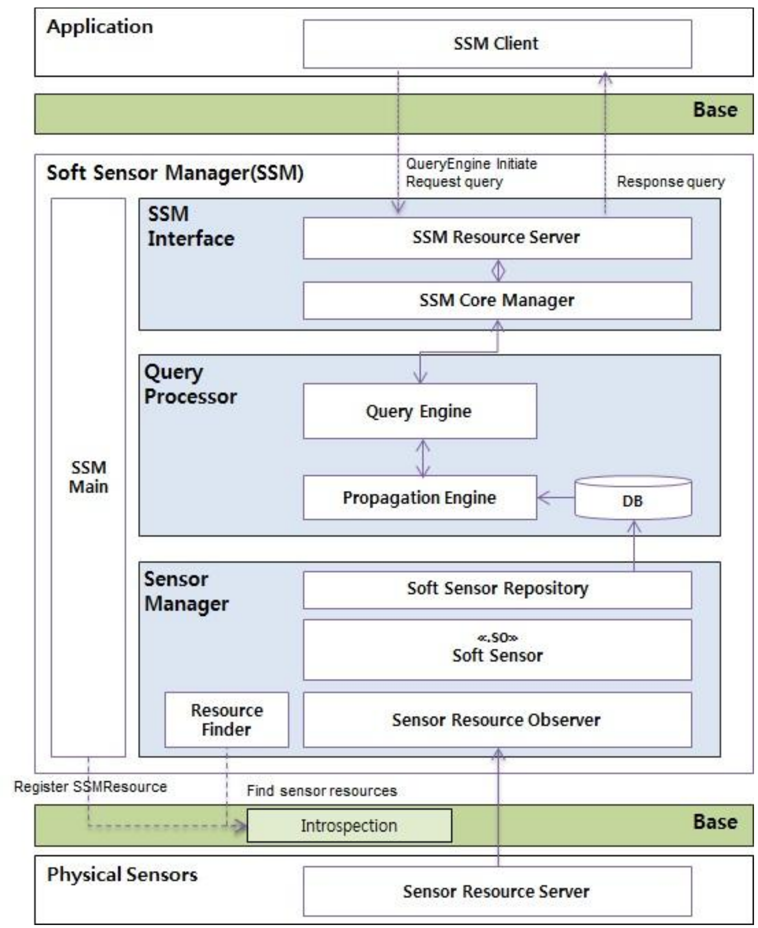




Service: Soft Sensor Manager

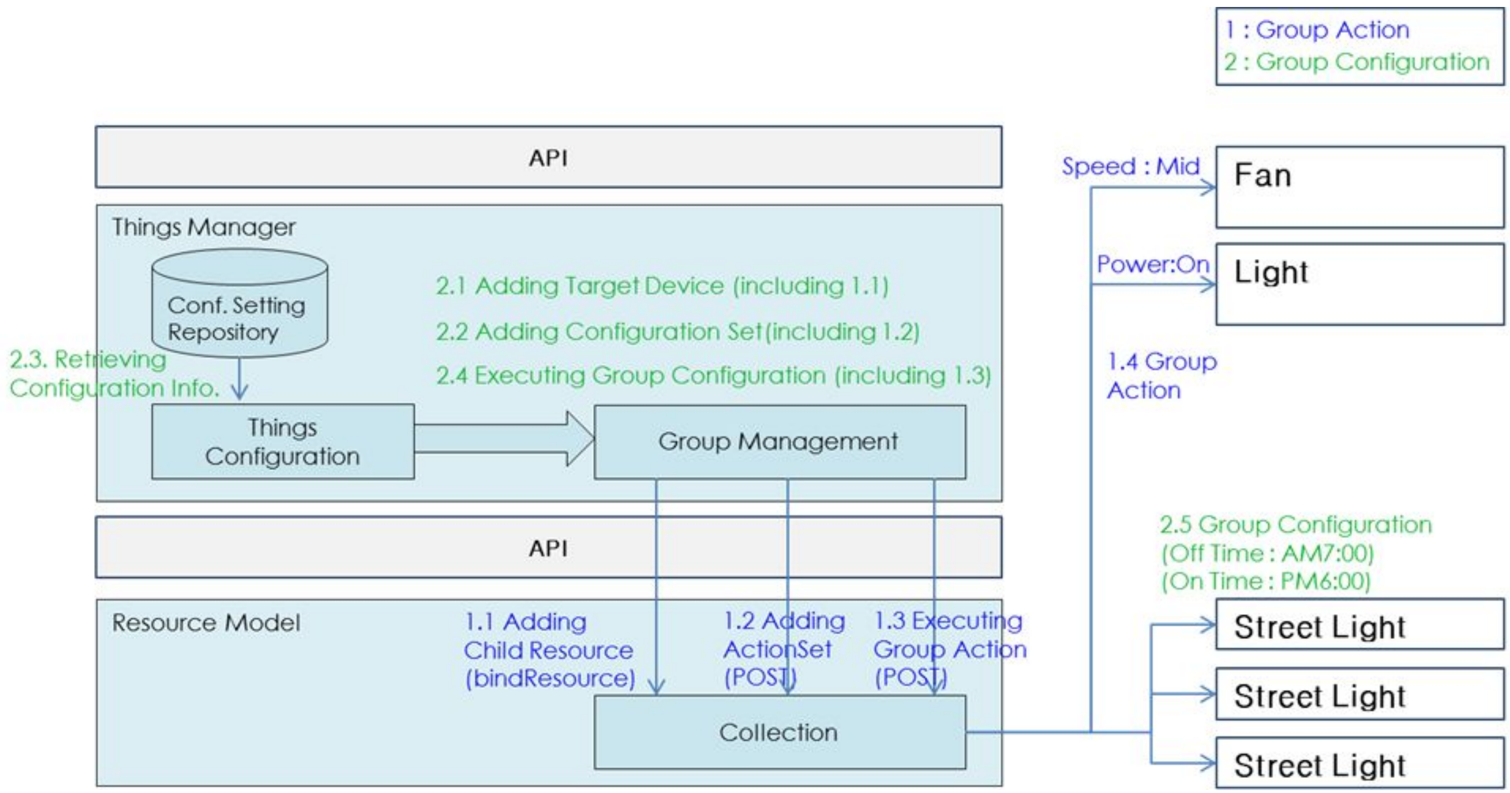
- Receive data from various sensors
- Query-based sensor data request





Service: Things Manager

- Group creation and finding appropriate resources in network



Control Manager

- Discover controlee devices
- Control controlee with Resource API
- Subscription/notification functionality
  - for monitoring the device operations or state changes
- Runs both as client and server