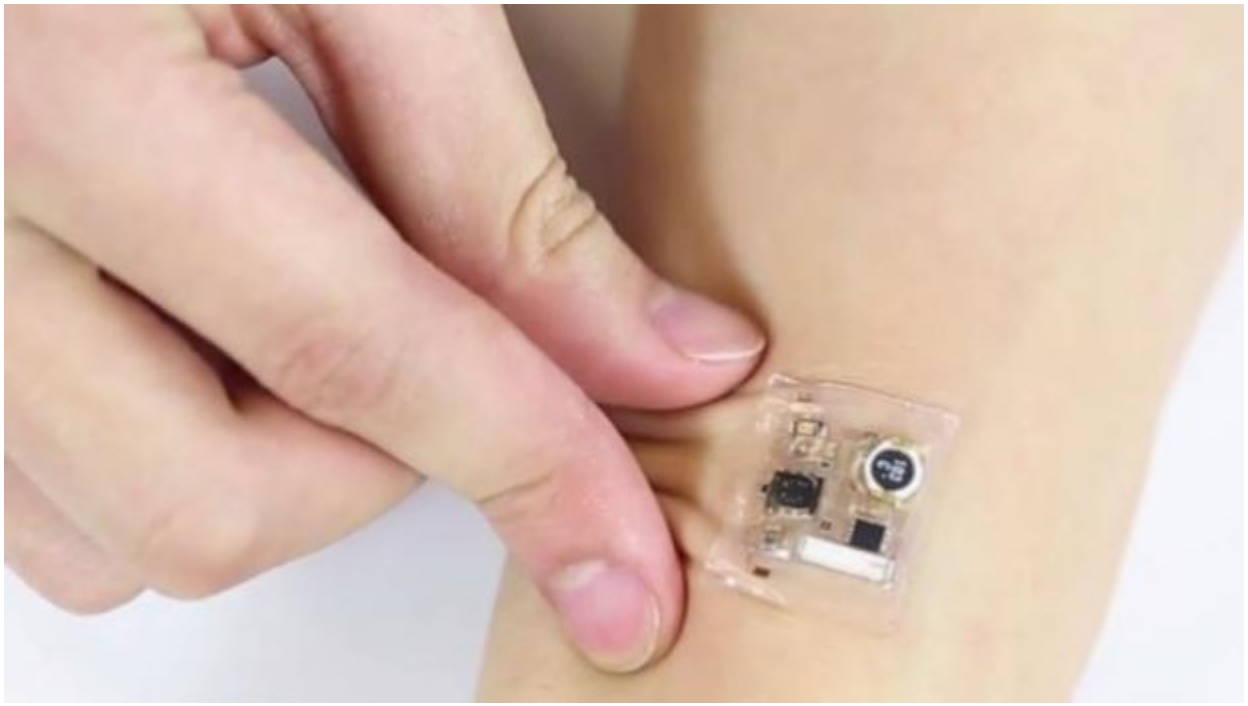
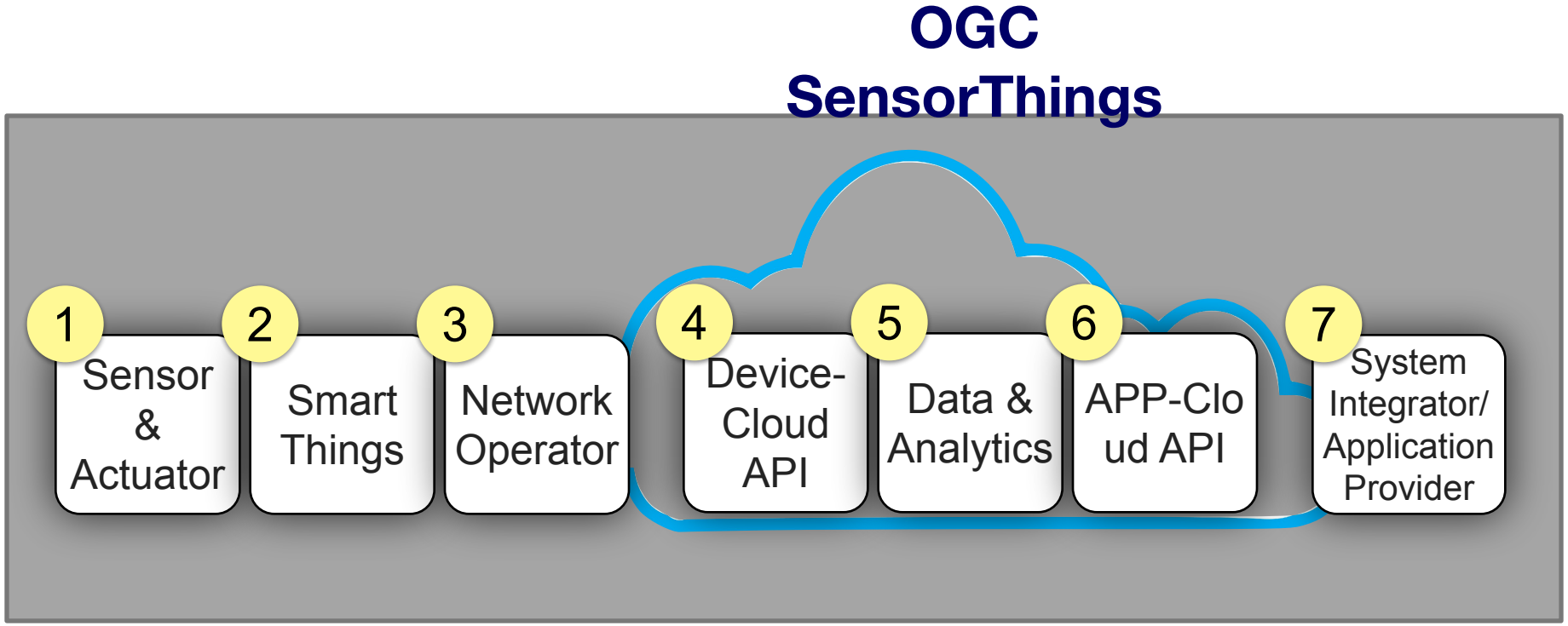
OGC®

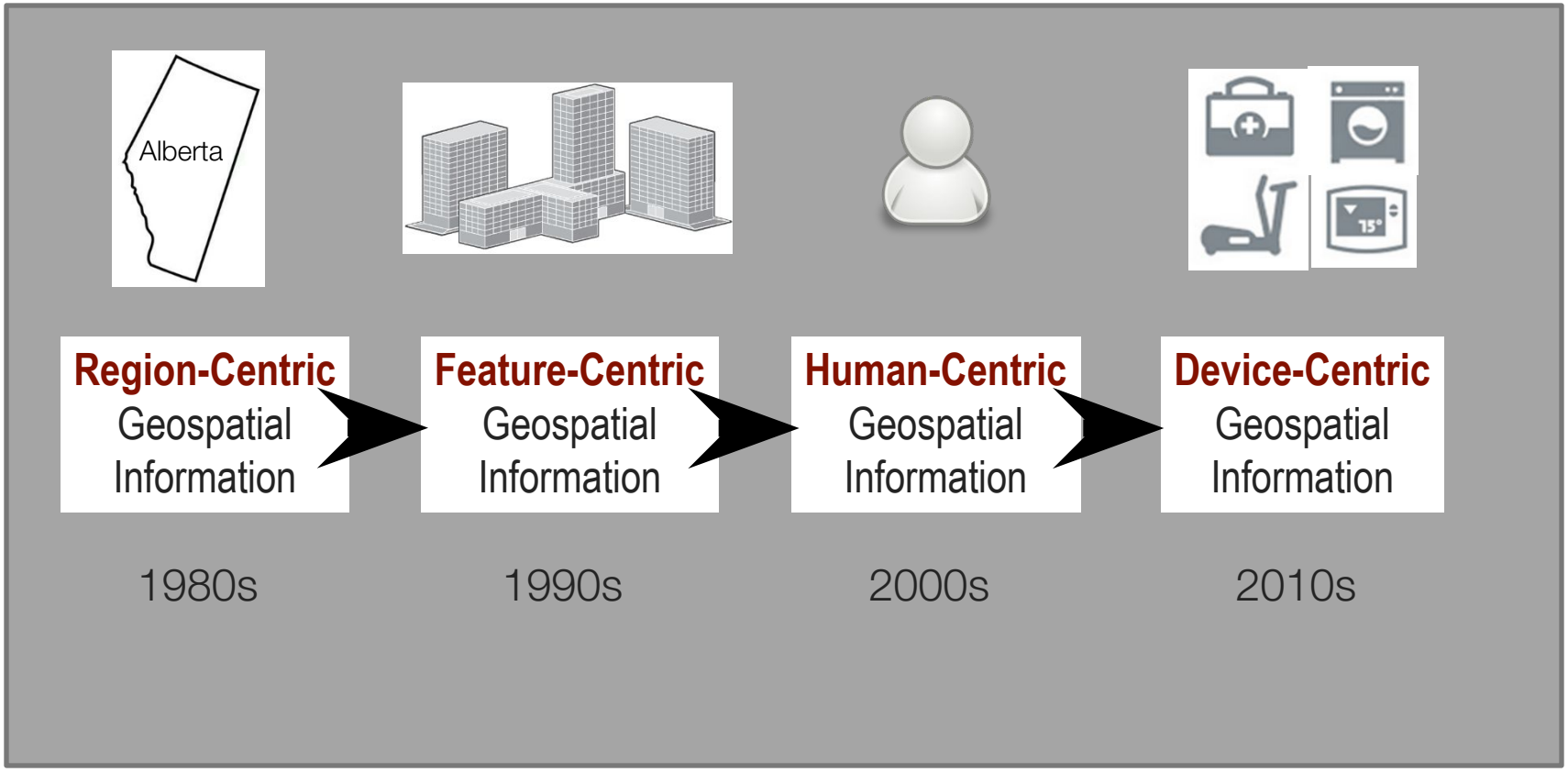
Every sensor in the ground
is in constant communication
with nearby relays.

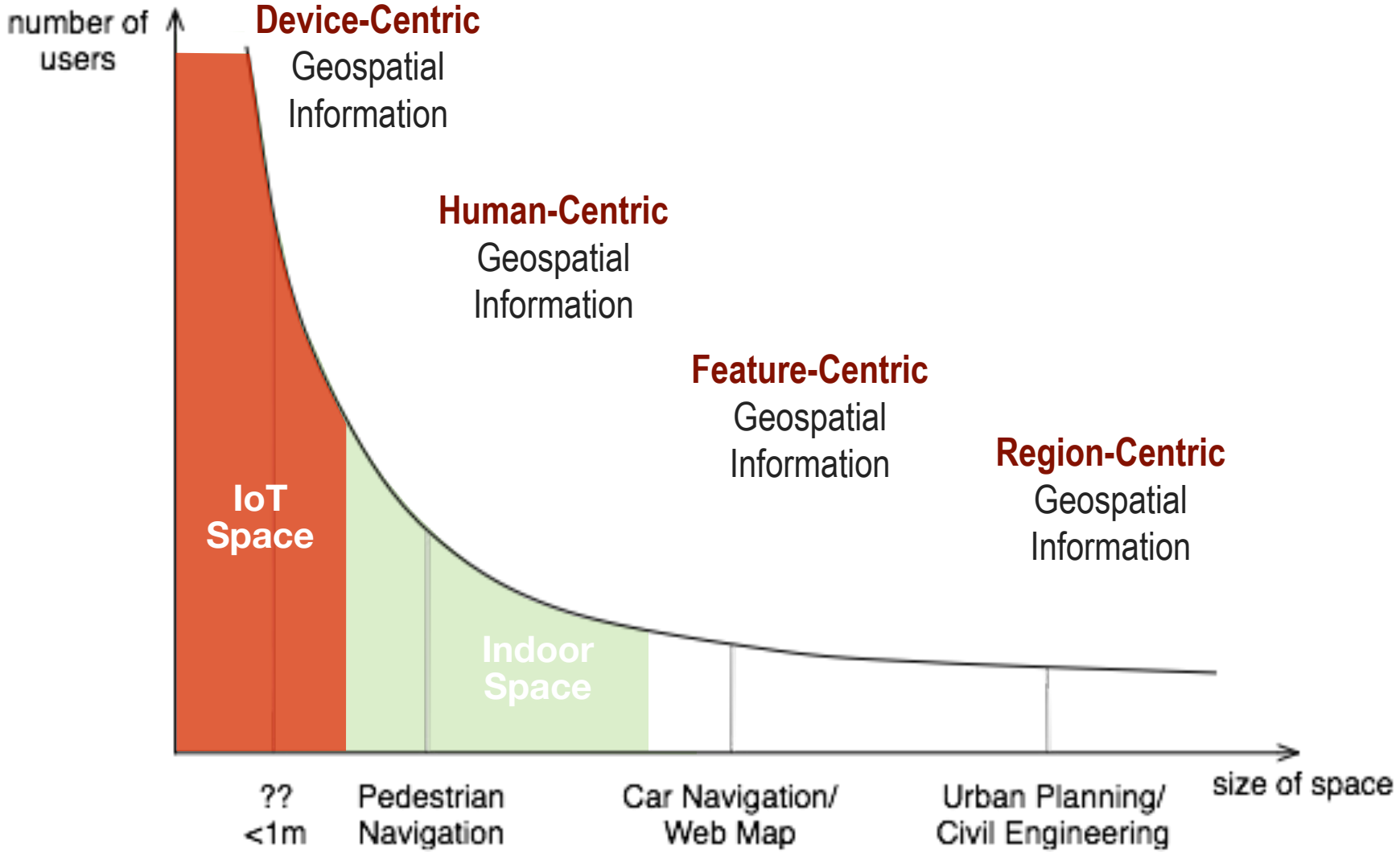


IoT Value Chain



Location Technology Evolution

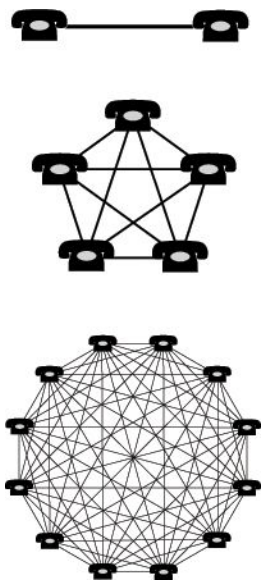




OGC®

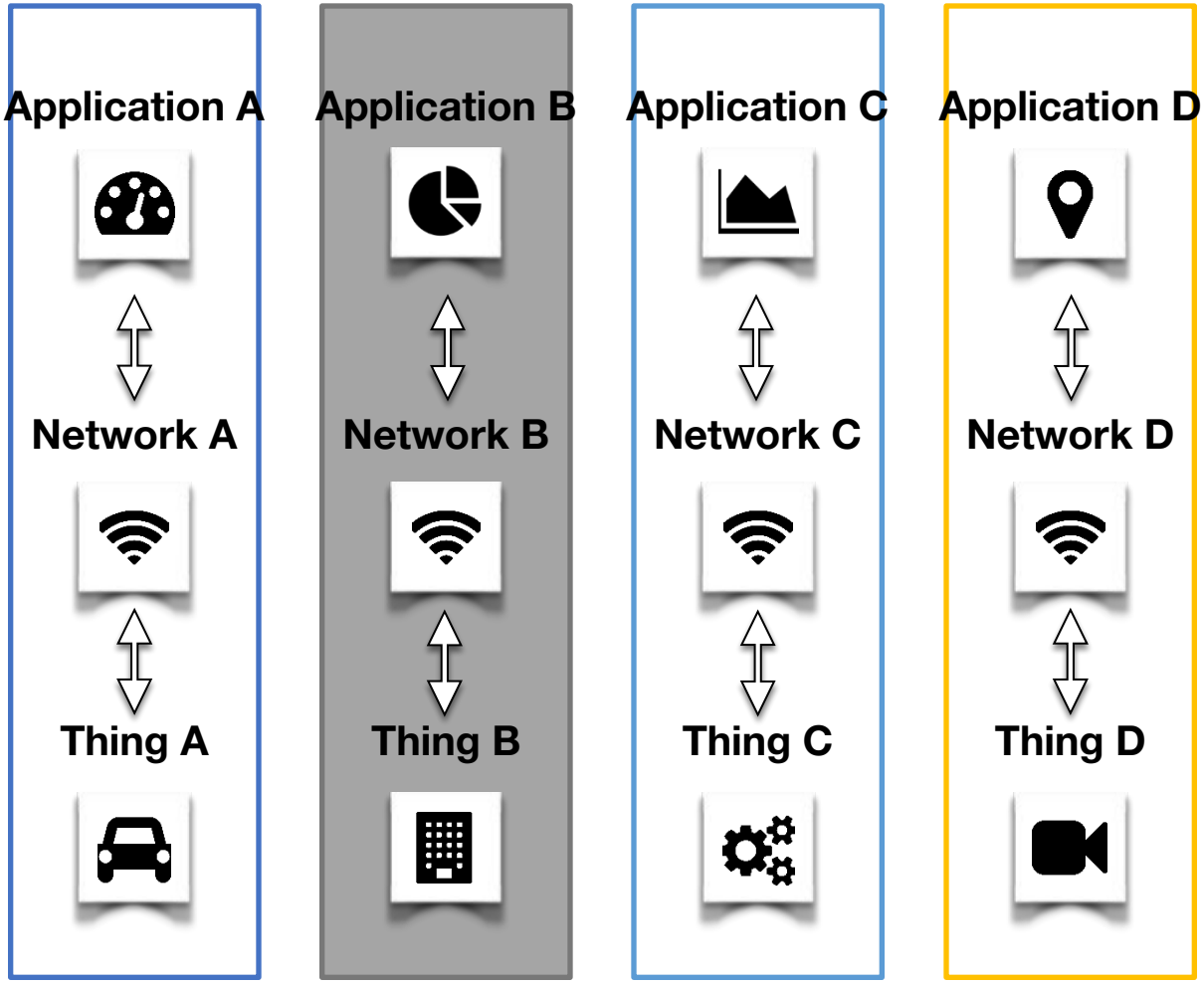
System of Systems

- The real potential of the Internet of Things



Network Effect:
The value of a network is proportional to the square of the number of users of the system (n^2).

Today's IoT Silos



*“ 77% of surveyed IoT experts claimed that **Interoperability** is the biggest challenge currently facing the Internet of Things ”*

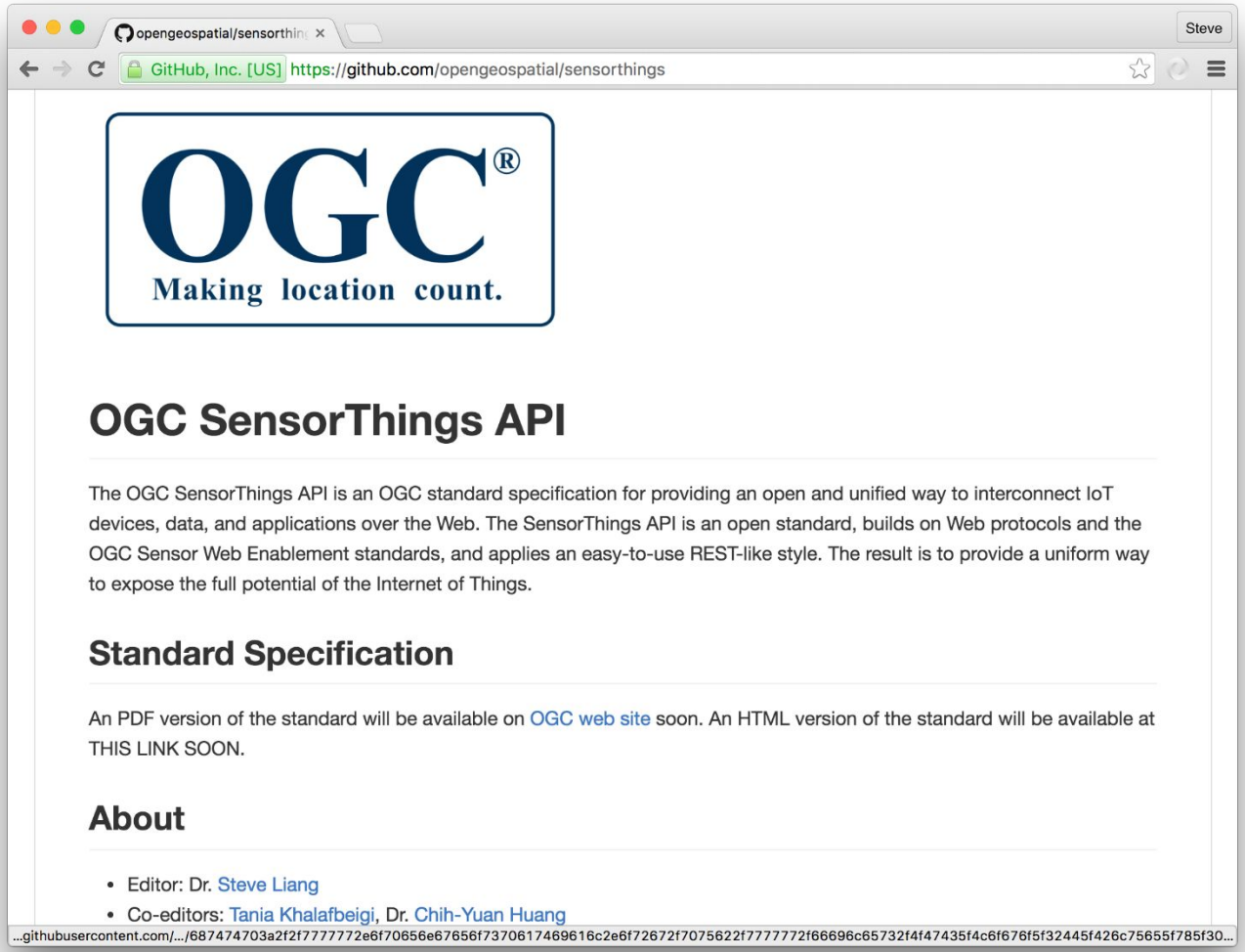
What is IoT Interoperability?

- Interoperability is the ability of two or more (IoT) systems or components to ***exchange*** information and to ***use*** the information that has been exchanged (IEEE) .

- Two components X and Y are interoperable if X can send requests R for services to Y, based on a ***mutual understanding*** of R by X and Y, and if Y can similarly ***return mutually understandable*** responses S to X (Brodie, 1993).

OGC SensorThings API

GitHub

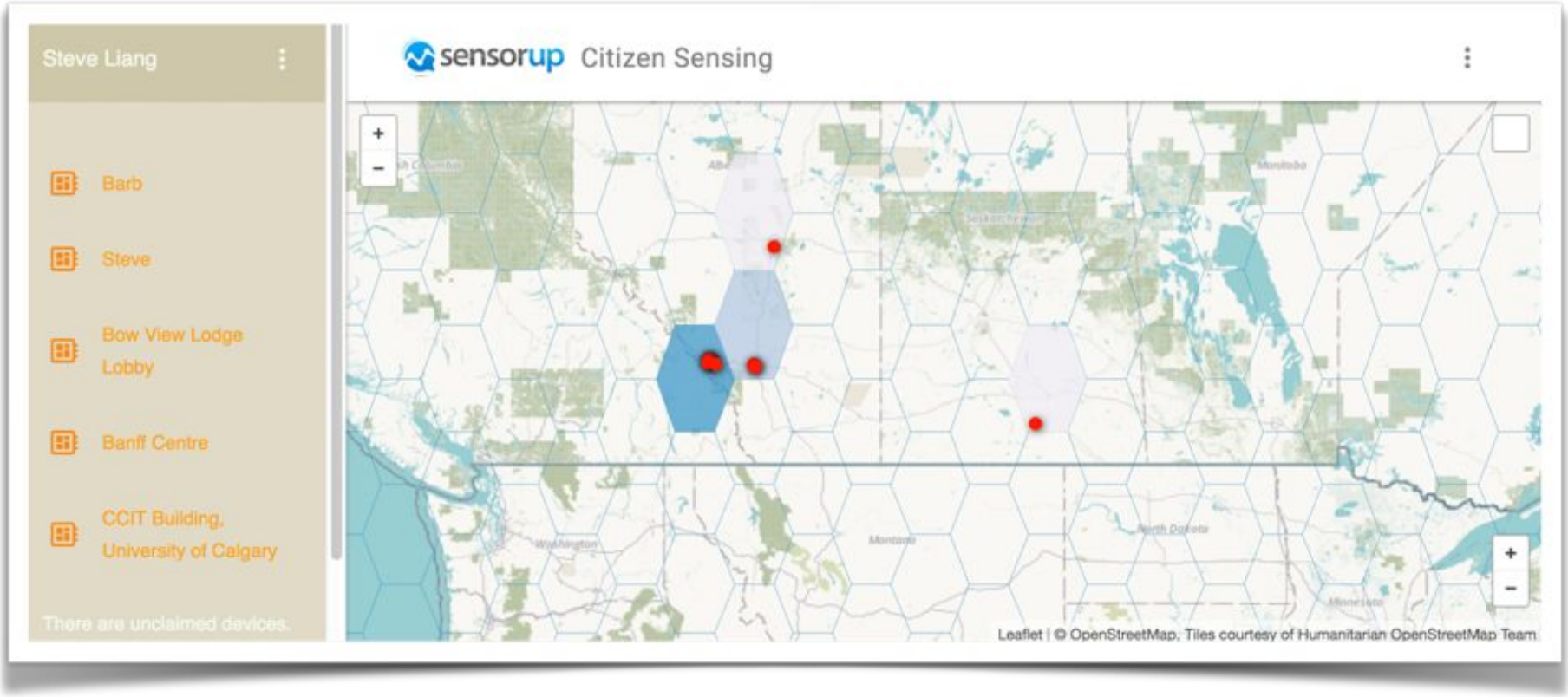


- "OGC specifications are [illegible] difficult to use…"

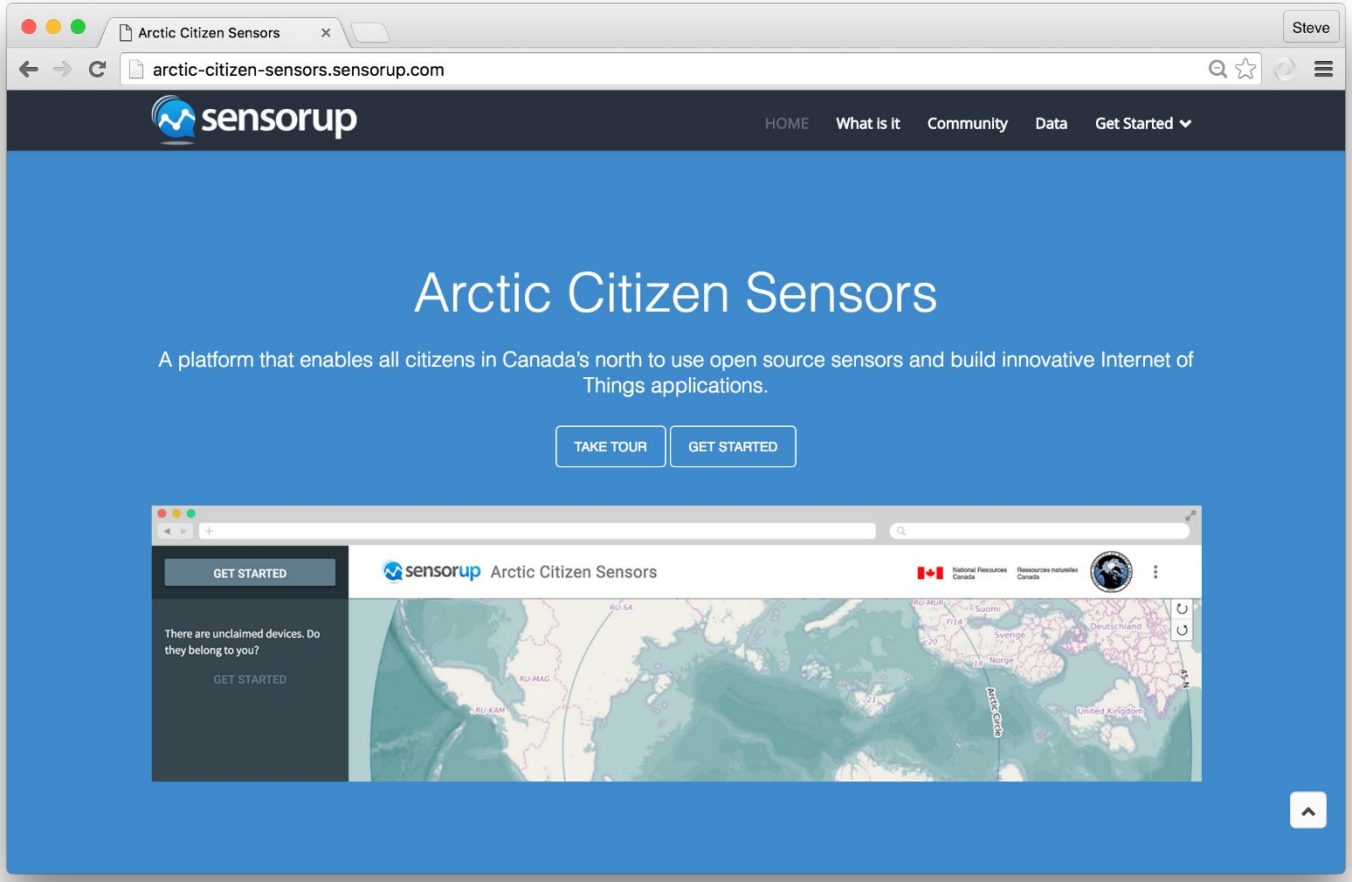
WRONG!

Someone at FOSS4G N.A.

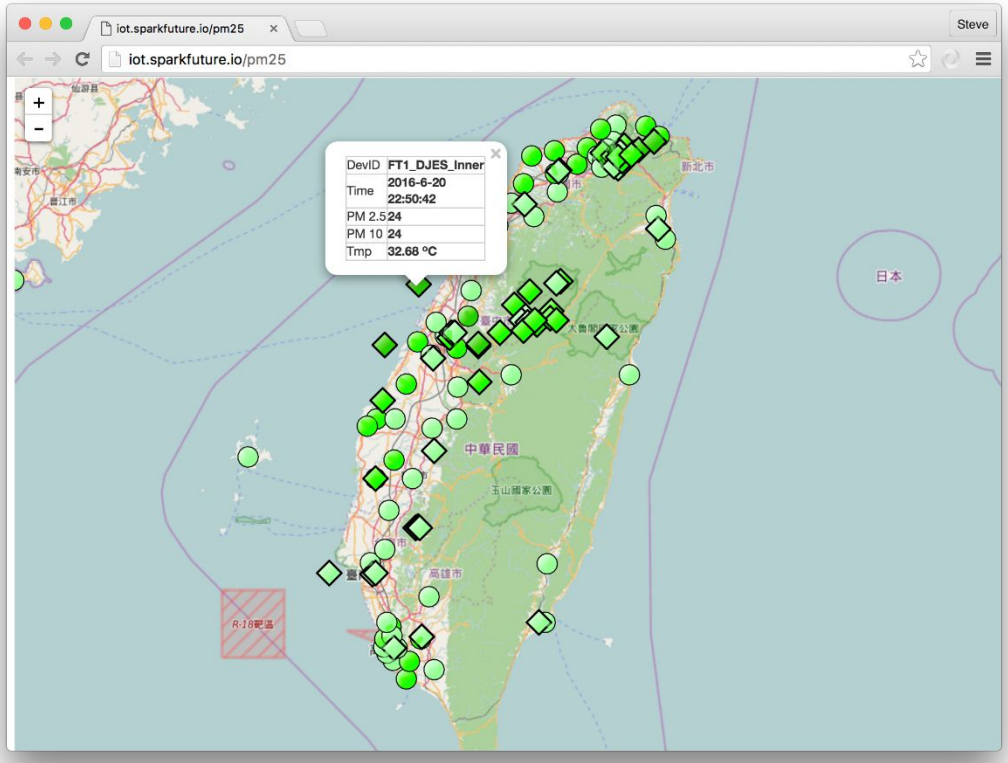
Case Study - Smart Citizen Sensors



Case Study - Arctic Citizen Sensors



Case Study - Citizen Sensing in Taiwan



Air Quality in Taiwan

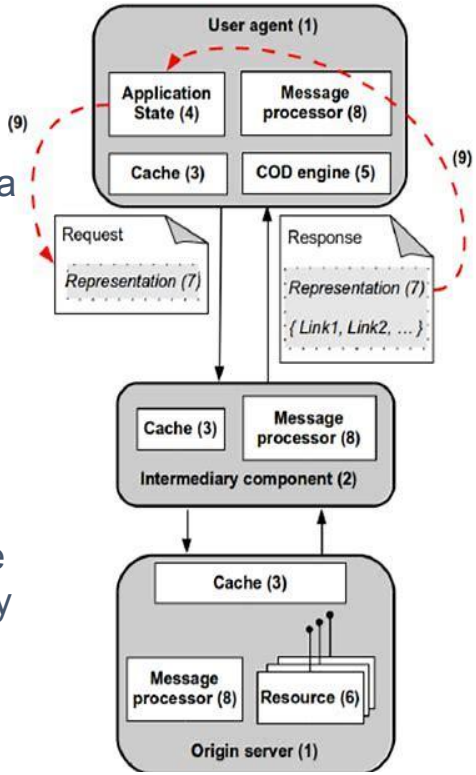
Working with the Maker community, and using SensorThings API to provide more than 500 near real-time air quality sensors

SensorThings API

- An open, geospatial-enabled and unified way to interconnect the Internet of Things (IoT) devices, data, and applications over the Web
- REST Principles
- CREATE, READ, UPDATE, and DELETE (*i.e.*, HTTP POST, GET, PATCH, and DELETE) IoT data and metadata
- Efficient JSON encoding
- MQTT (Message Queuing Telemetry Transport) protocol
- Follows flexible OASIS OData protocol and URL conventions

REST APIs follow six design principles

- **Client-server Separation:** The application which is requesting the resource is called the client, and the application which has the resource is called the server. When the client requests a request to the server, the server sends a response to the client. The server can't initiate a request to the client. In a RESTful API, the client and server are always kept independent of each other. This ensures that both the client and the server can be scaled independently.
- **Stateless:** In a RESTful API, each request needs to contain the data that is necessary to process it. Servers aren't allowed to store any data related to the client. No session or authentication state is stored on the server. If the client requires authentication, then the client needs to authenticate itself before sending a request to the server.
- **Cacheable:** In REST APIs, the resources should be able to cache themselves either on the client or on the server. When a client requests a resource from the server, the response from the server will contain the information on whether the resource can be cached or not and for how long. The main idea of caching is to improve the performance of the client by reducing the bandwidth required to load the resource.
- **Layered System:** In REST APIs, there can be multiple intermediaries between the client and the server. It isn't always necessarily true that the client connects directly to the server and requests a resource. There can be multiple systems in between them that are responsible for handling security, traffic, balancing the load, redirection, etc. The client or the server doesn't have any information about how many systems are in between them.
- **Uniform Interface:** All requests and responses in a REST API should follow a common protocol. This allows the applications to evolve independently. The client and server can interact with each other in a single language irrespective of the architecture that they are based upon.



Differences between SOS and SensorThingsAPI

- OGC SensorThings API can interoperate with SOS at both the **data level** and **service interface** level.

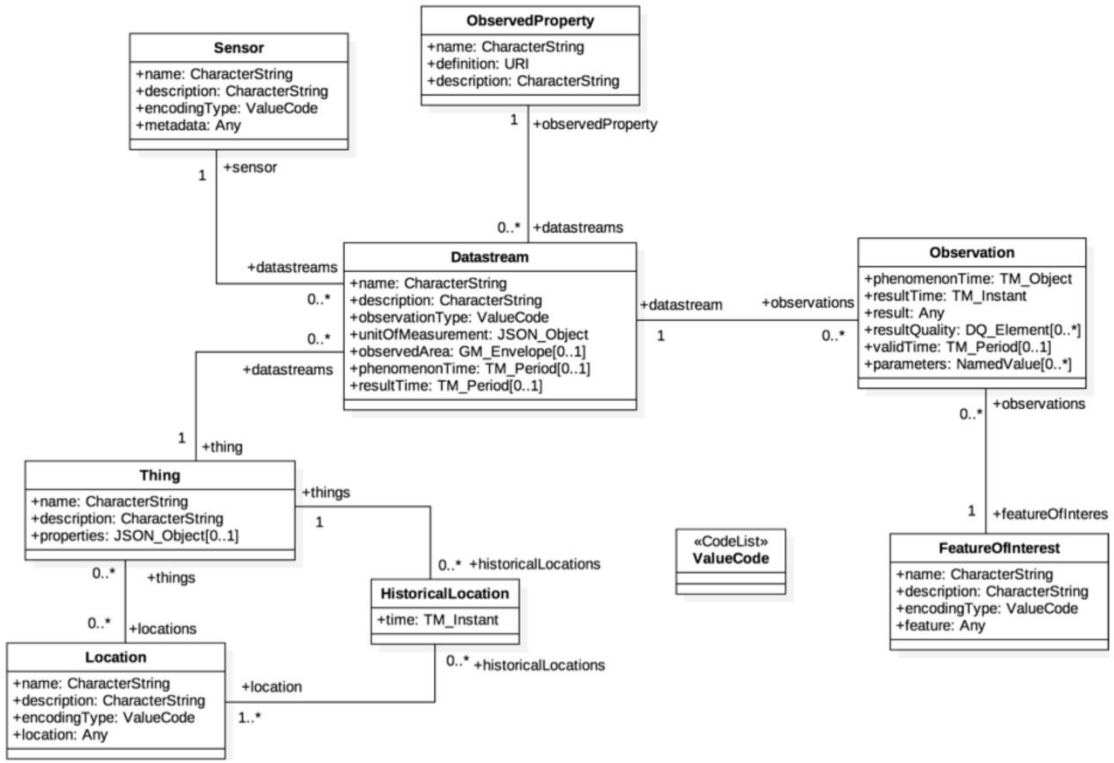
	OGC SensorThings API	SOS
Encoding	JSON	XML
Architectural Style	Resource Oriented Architecture	Service Oriented Architecture
Binding	REST	SOAP
Insert new Sensors and Observations	HTTP POST	SOS specific interface: RegisterSensor() and InsertObservation()
Deleting Existing Sensors	HTTP DELETE	SOS specific interface: DeleteSensor()
Pagination	$top/$skip/$nextLink	Not supported
Pub/Sub Support	MQTT and SensorThings MQTT Extension	Not supported
Updating Properties of Existing Sensors or Observations	HTTP PATCH and JSON PATCH	Not supported
Deleting Existing Observations	HTTP DELETE	Not supported
Linked Data Support	JSON-LD	Not supported

Supporting **real-time sensing applications**

Better developer experience

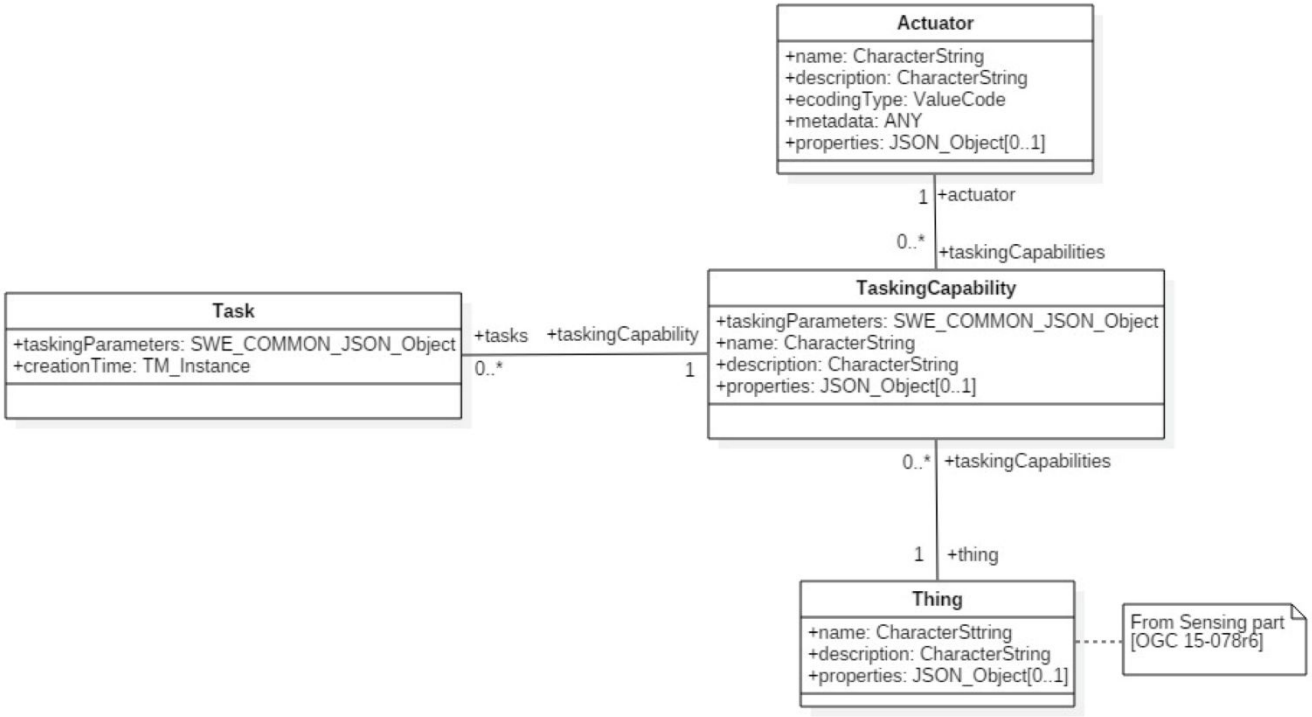
Part 1: Sensing Part

- Provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems.
- Designed based on the ISO/OGC Observation and Measurement (O&M) model



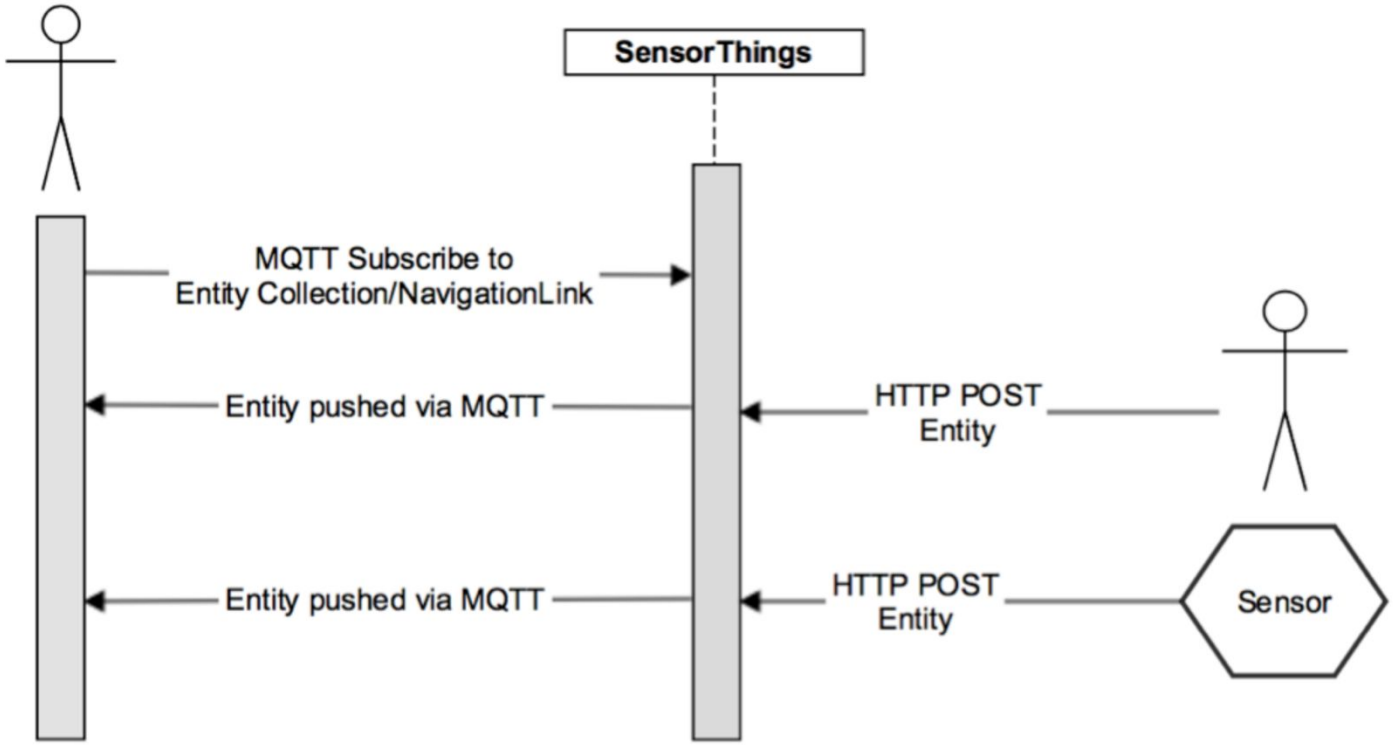
Part 2: Tasking Part

- Provides a standard way for parameterizing - also called tasking - of taskable IoT devices, such as individual sensors and actuators



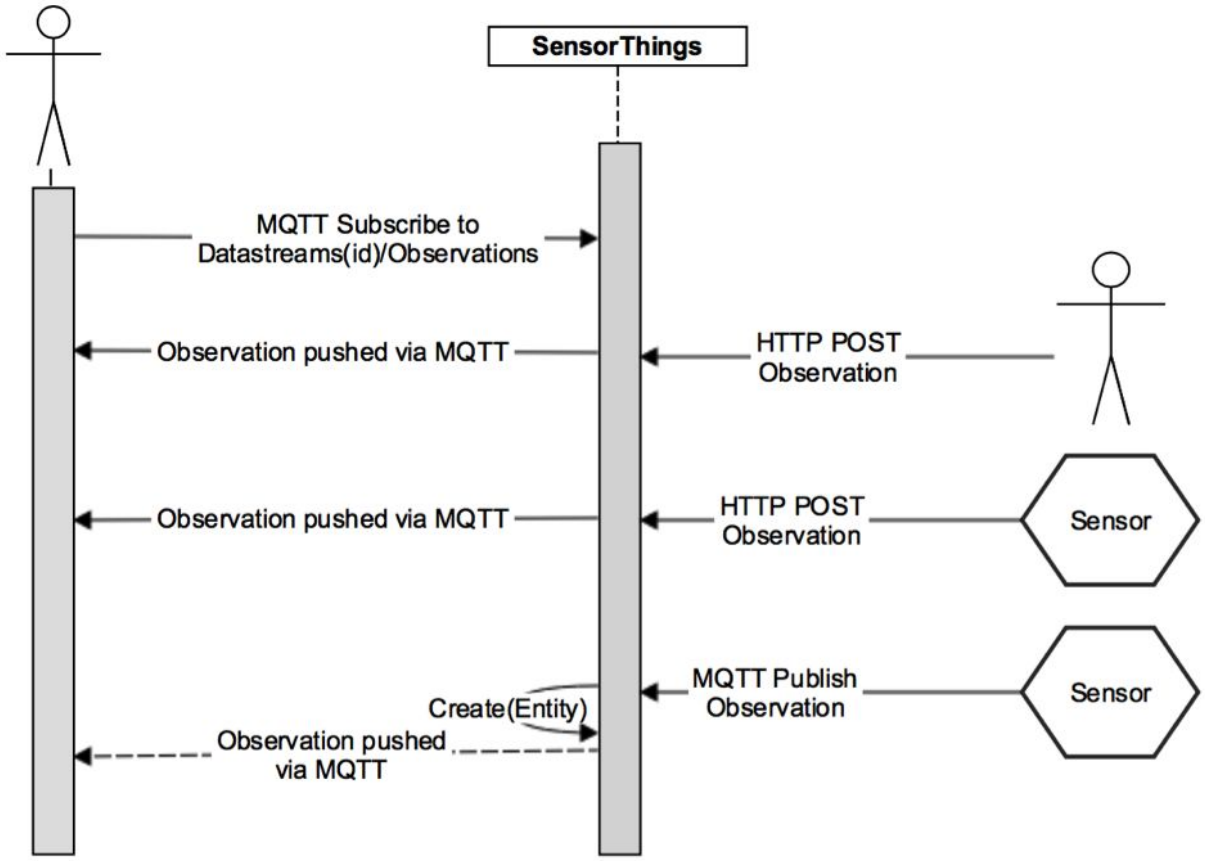
SensorThings API MQTT - Read

- Topic: entity collection name
  - Example: v1.0/Things, v1.0/Datastreams(id)/Observations
- Payload: SensorThings JSON entity



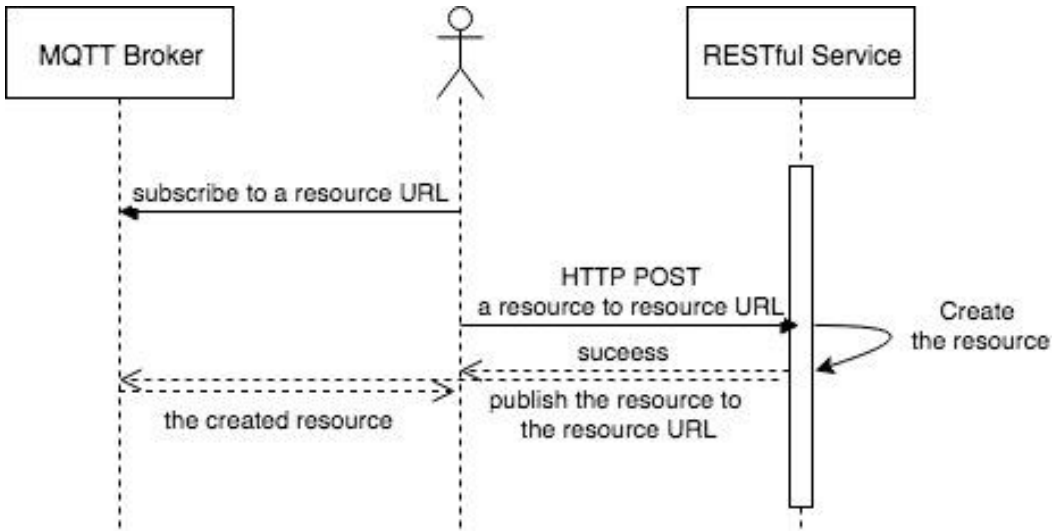
SensorThings API MQTT- Create Observations/Tasks

- Topic: Resource Path to Observations
  - Example: v1.0/Observations, v1.0/Datastreams(id)/Observations
- Payload: Valid Observations JSON entity



Lesson Learned – Read

- Each RESTful API has a potential for MQTT binding to receive updates for a resource collection
- The topic would be the resource GET URL
- The payload would be the same as content of HTTP GET
- Whenever there is a new resource, it will be published to the resource GET URL topic



Lesson Learned - Create

- For any RESTful API, to create a resource, MQTT can be an option just like HTTP POST
- The topic will be same as the POST topic
- The payload will be the same as POST payload
- The service would subscribe to the topics
- When it receives the payload, it uses the same process as POST for creating the resource