UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

IoT Systems

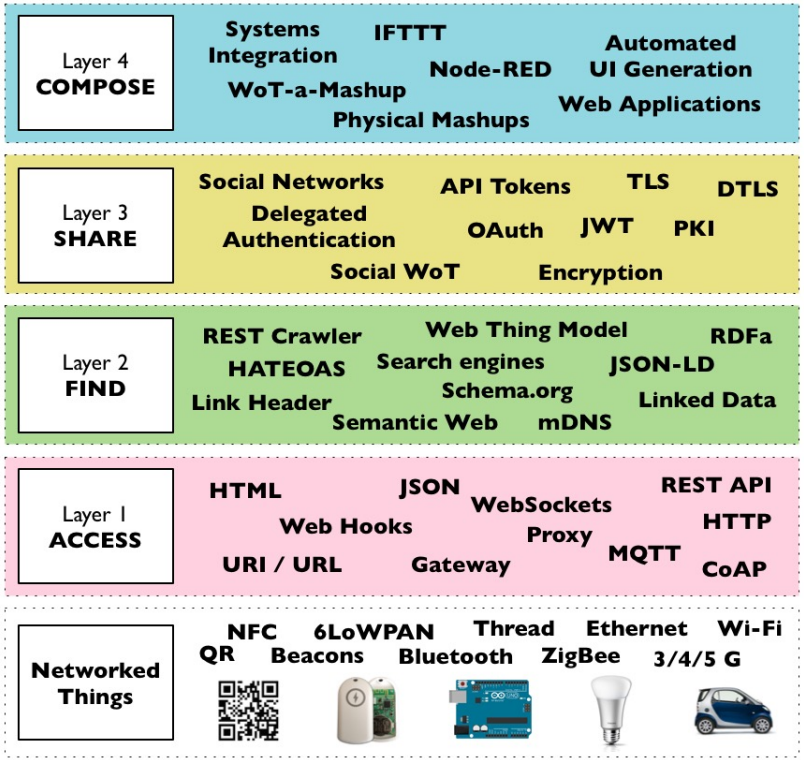
Web of Things

Prof. Luca Bedogni

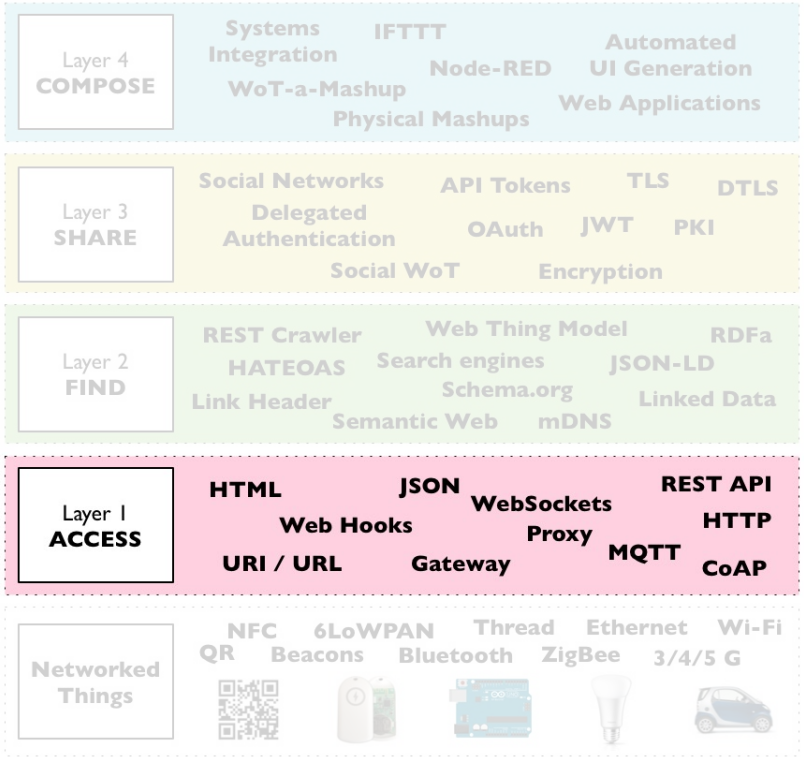
What is WoT?

- WoT stands for Web of Things
- It is a platform of platforms
- Enables interoperability in the IoT
- The Internet is not much different from the Internet of Things
  - However, there are a lot more devices in it
  - There are also novel protocols, such as MQTT, CoAP, AMQP, Zigbee, BLE…
- So the WoT provides interoperability, by defining a common representation of a “Thing”

The WoT Architecture



The WoT Architecture



Things

- Things are described by a hierarchical structure
- Types abstraction
  - Sensors
  - Actuators
  - Modalities and Actors
  - Devices and Capabilities
- Their description can be done with various languages
  - Such as JSON, RDF, Turtle
  - Usually, JSON-LD
- There are Web Things and Extended Web Things
  - More on this on https://model.webofthings.io

Things

- They can be queried through a URL
  - http://myhome.local/studio/lamp
  - http://192.168.1.254:5555/smartLight1
  - All Things in WoT must implement a Web Server, and ~~may~~ should use HTTPS
  - All things in WoT should be organized hierarchically
- They may offer different representations of it
  - Clients can query for a specific representation

HTTP Request
… Accept: application/xml …

HTTP Response
… Content-type: application/xml …

Things

- WoT Things must implement a series of HTTP operations
  - GET, POST, PUT, DELETE

Things

- WoT Things must implement a series of HTTP operations
  - **GET**, POST, PUT, DELETE

Read a Value from a WoT resource

GET /myhome.local/garden/smartlight/status

200 OK
{‘light’: 0}

Things

- WoT Things must implement a series of HTTP operations
  - GET, **POST**, PUT, DELETE

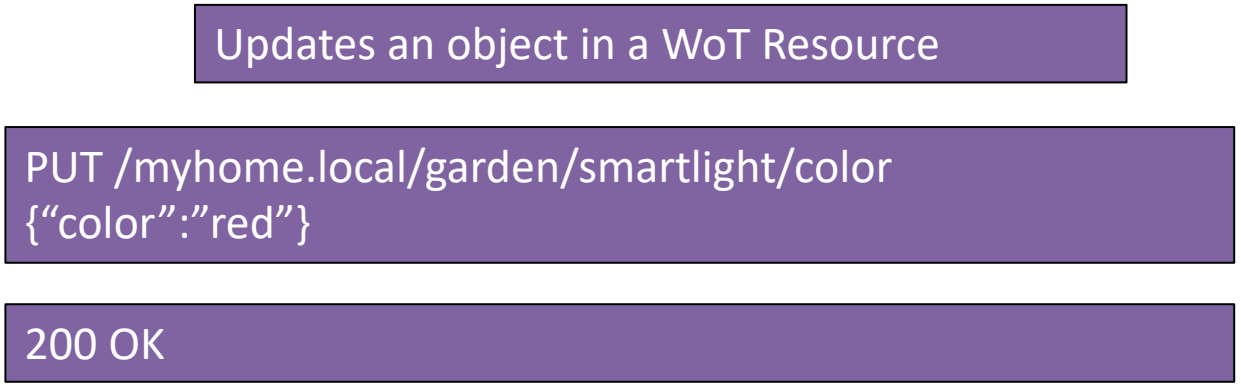
Creates a new object in a WoT resource

POST /myhome.local/garden/smartlight/program
{“time”:”20:00”, “action”: 1}

201 Created
Location /myhome.local/garden/smartlight/program/xa5

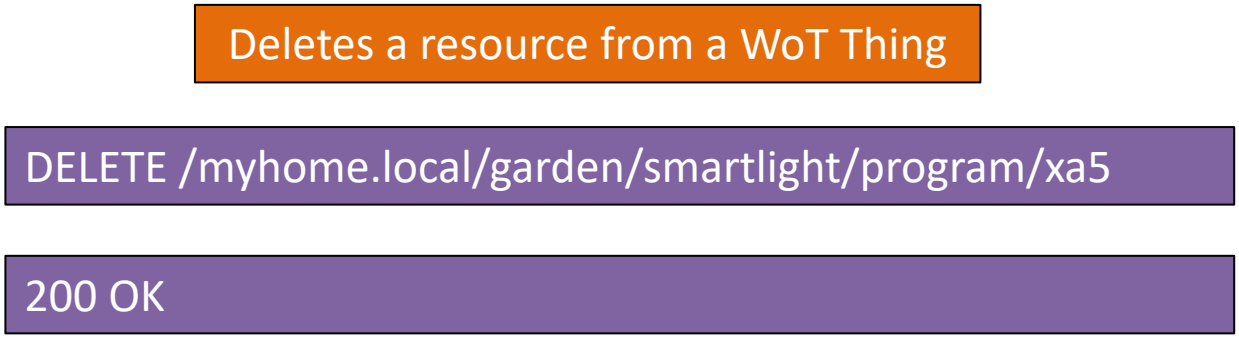
Things

- WoT Things must implement a series of HTTP operations
  - GET, POST, **PUT**, DELETE



Things

- WoT Things must implement a series of HTTP operations
  - GET, POST, PUT, **DELETE**

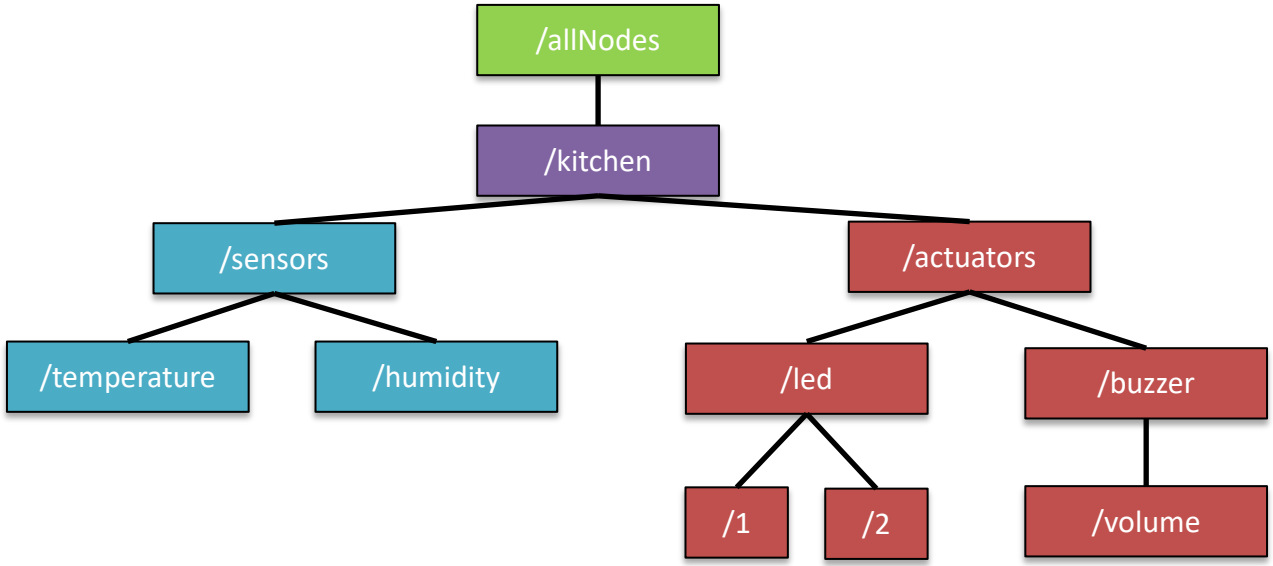


Requirements

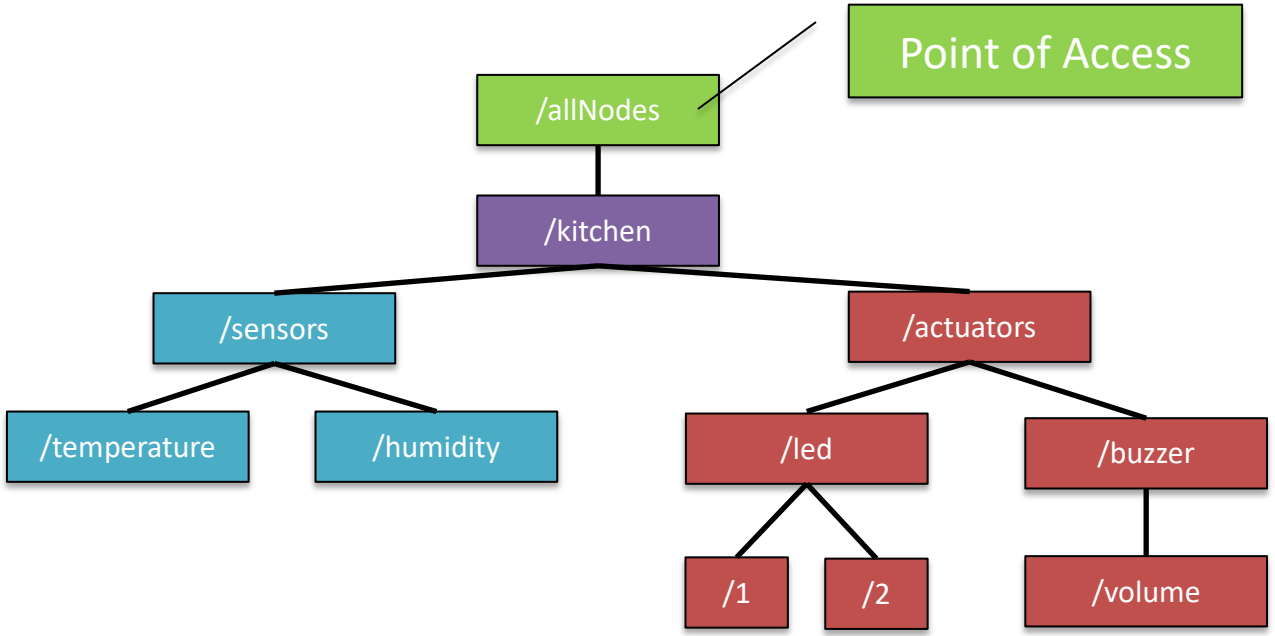
- Each Web Thing must
  - Have at least an HTTP 1.1 server
  - Have an accessible root URL on which HTTP GET can be performed
  - Must support HTTP GET, POST, PUT, DELETE
  - Must implement HTTP status code 200, 400, 500
  - Must be able to understand JSON
- Each Web Thing should
  - Use HTTPS
  - Implement WebSocket
  - Return 204 for any write operation
  - Provide a human readable documentation
- Each Web Thing may
  - Support HTTP options
  - Provide additional representations such as RDF, JSON-LD etc
  - Offer an HTML interface

Source: https://model.webofthings.io/

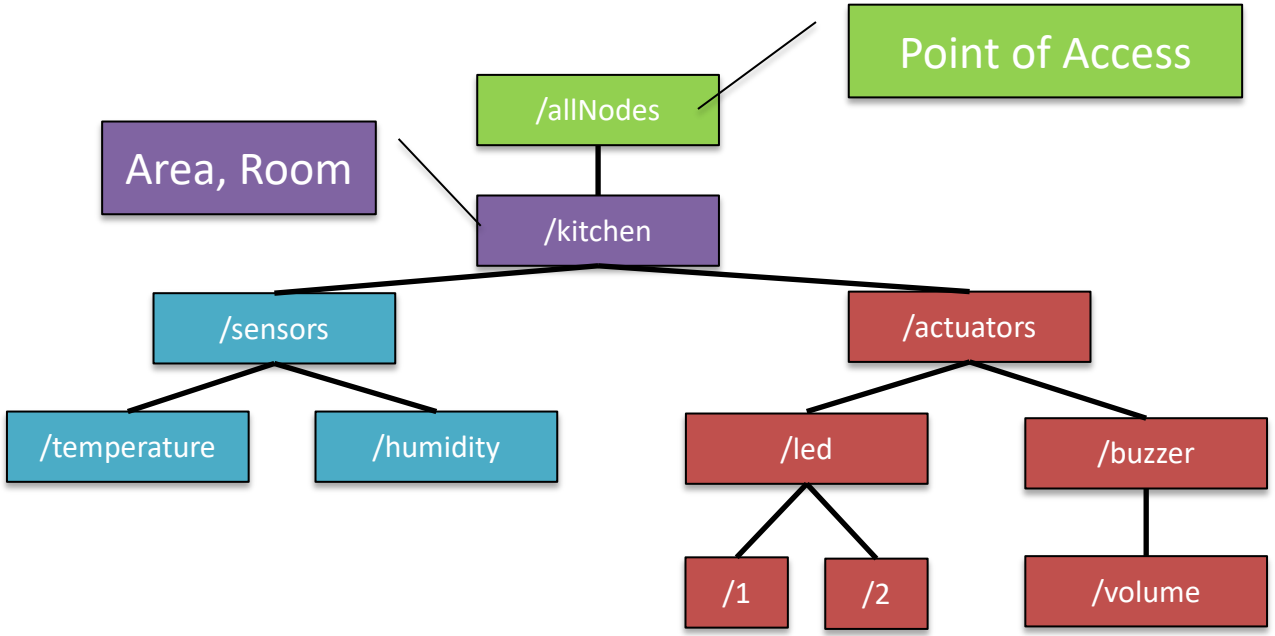
Layer 1 - Access



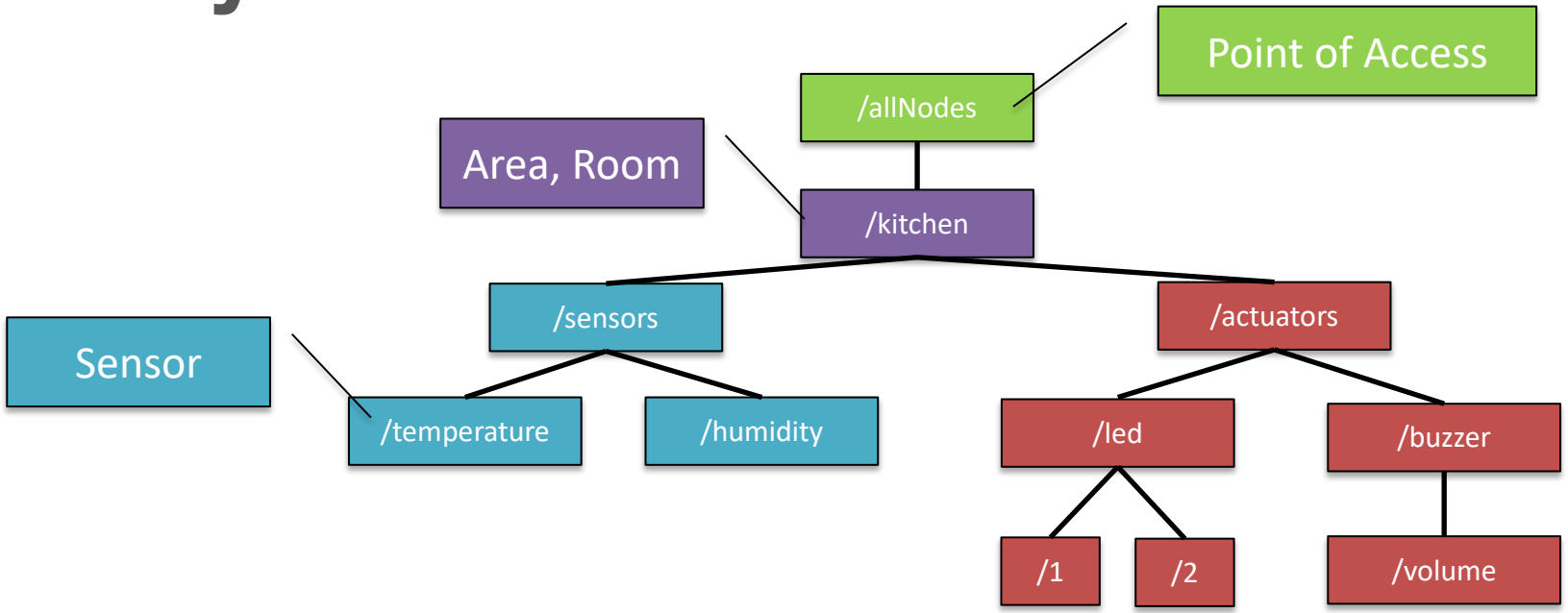
Layer 1 - Access



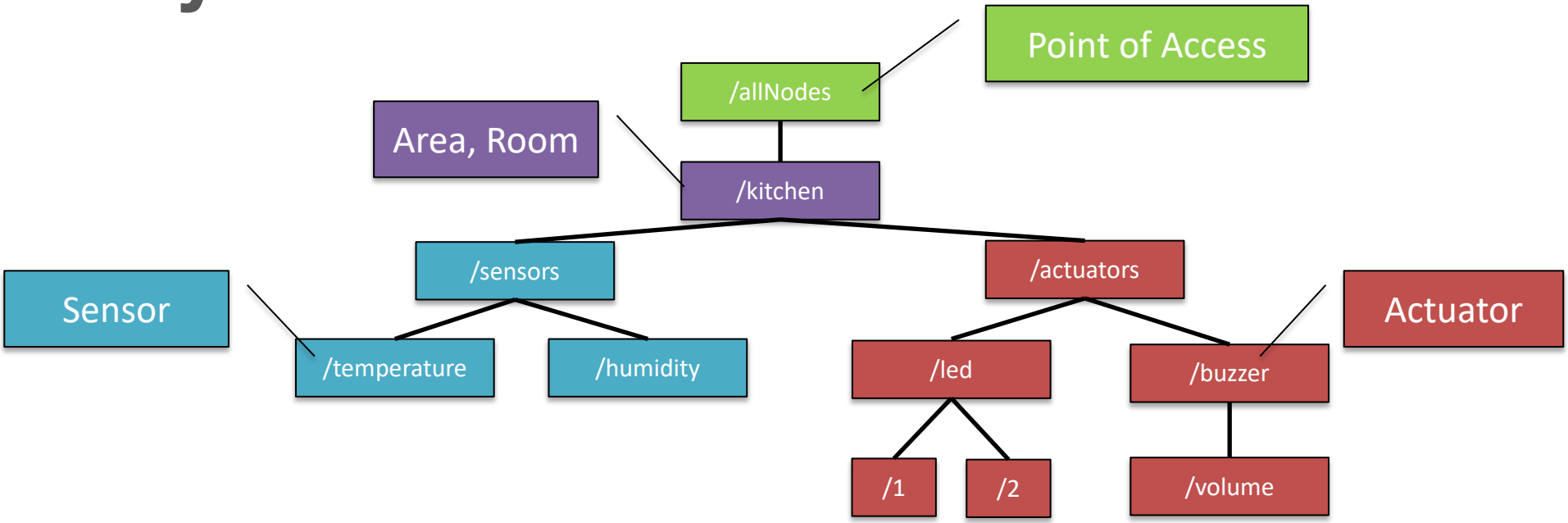
Layer 1 - Access



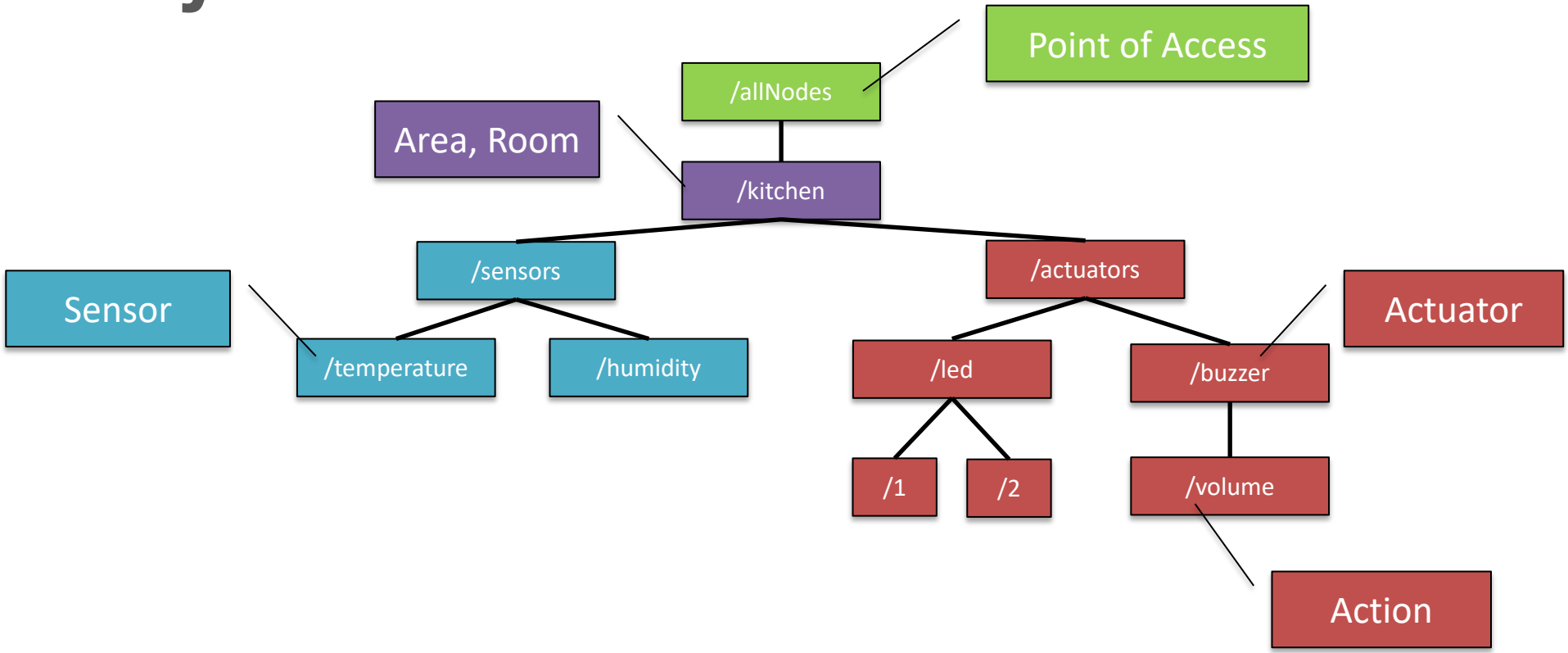
Layer 1 - Access



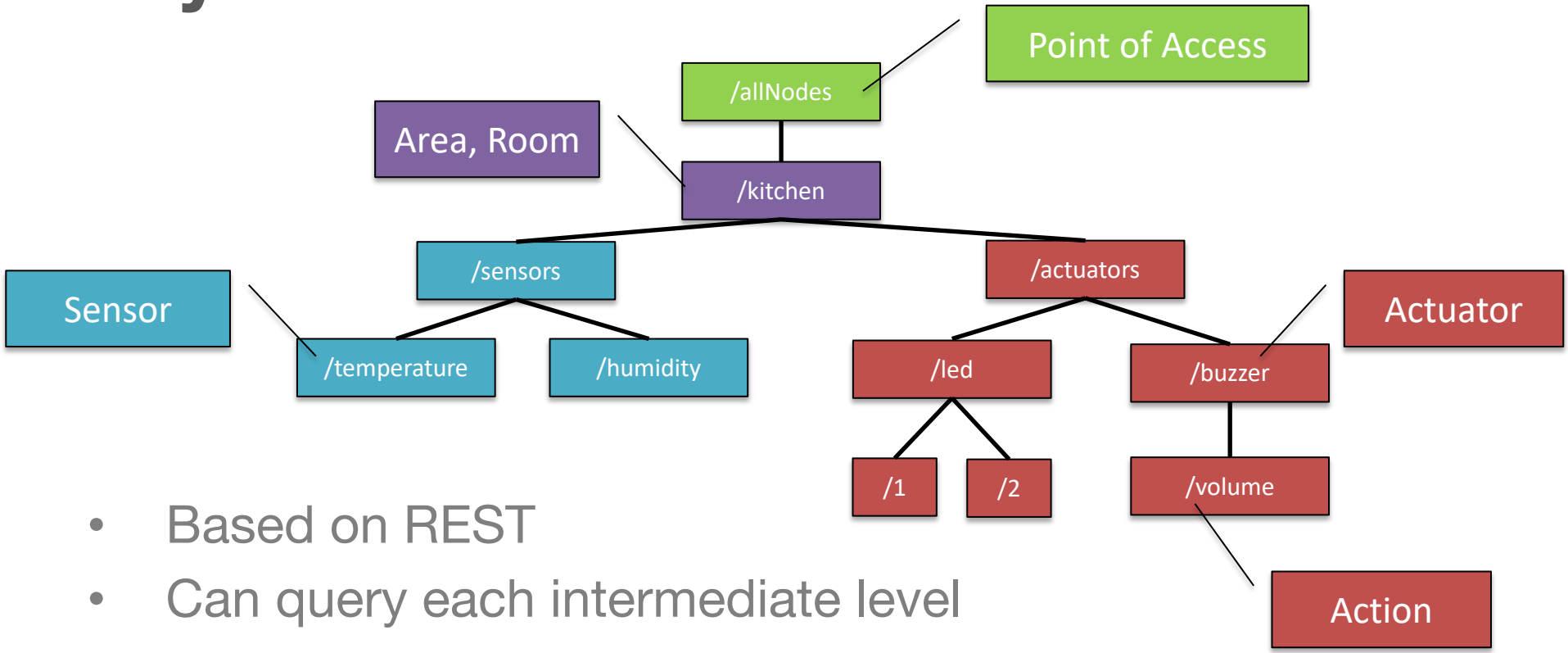
Layer 1 - Access



Layer 1 - Access

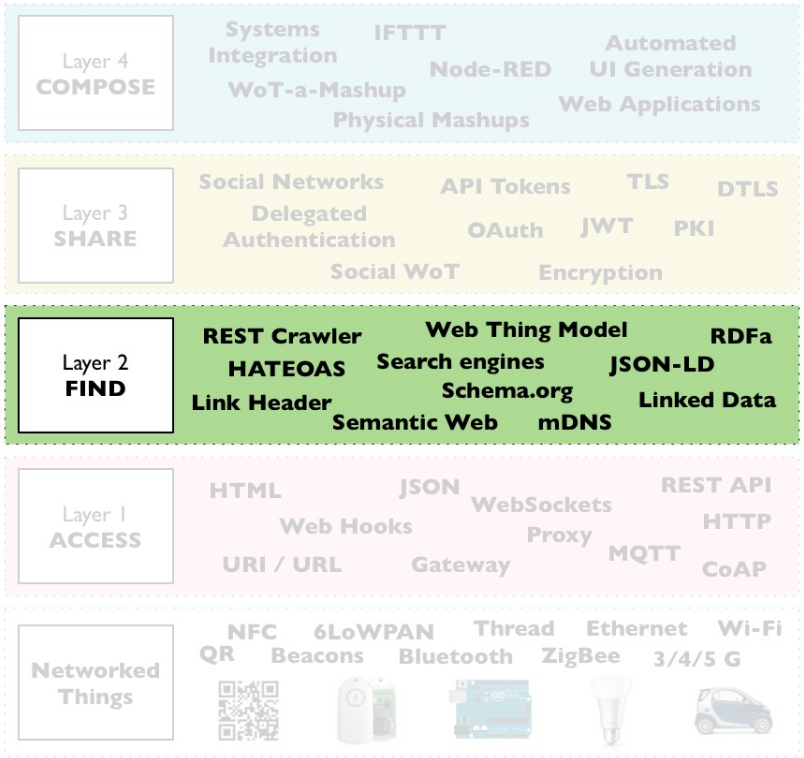


Layer 1 - Access



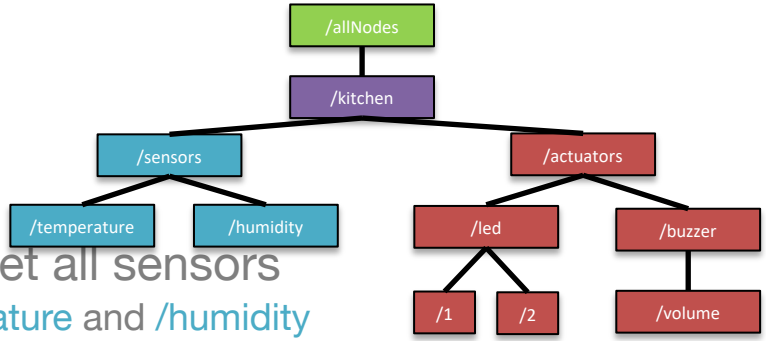
- Based on REST
- Can query each intermediate level

The WoT Architecture

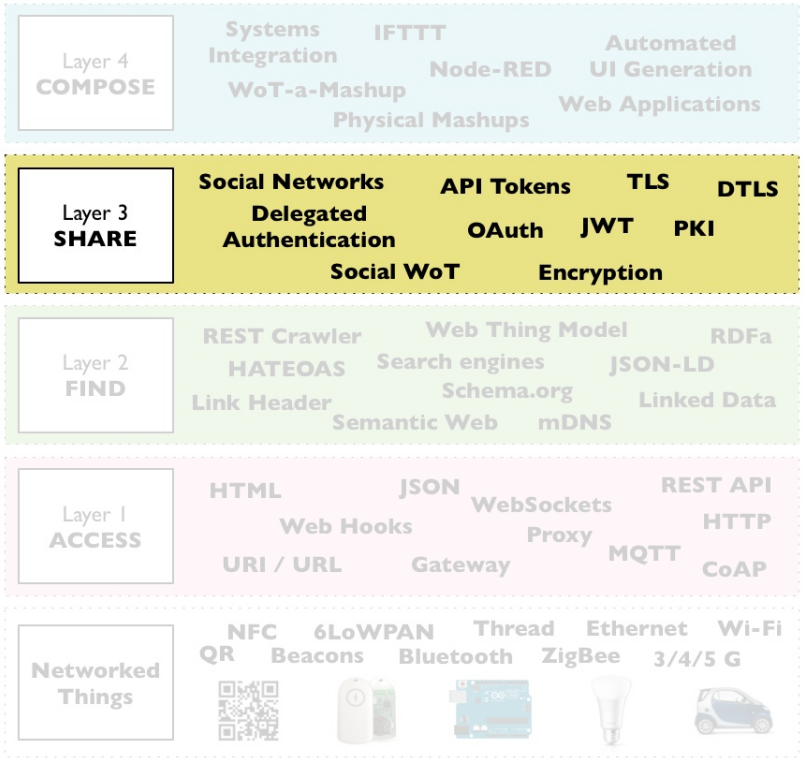


Level 2 - Find

- The hierarchical structure enables also to find things and things capabilities
- For instance
  - Give me all rooms
    - Query /allNodes, get /kitchen
  - Now you know there's a kitchen. Get all sensors
    - Query /kitchen/sensors and get /temperature and /humidity
  - Now you can get the values of the sensors
  - A Thing may support the OPTIONS request
    - Get the capabilities of the device



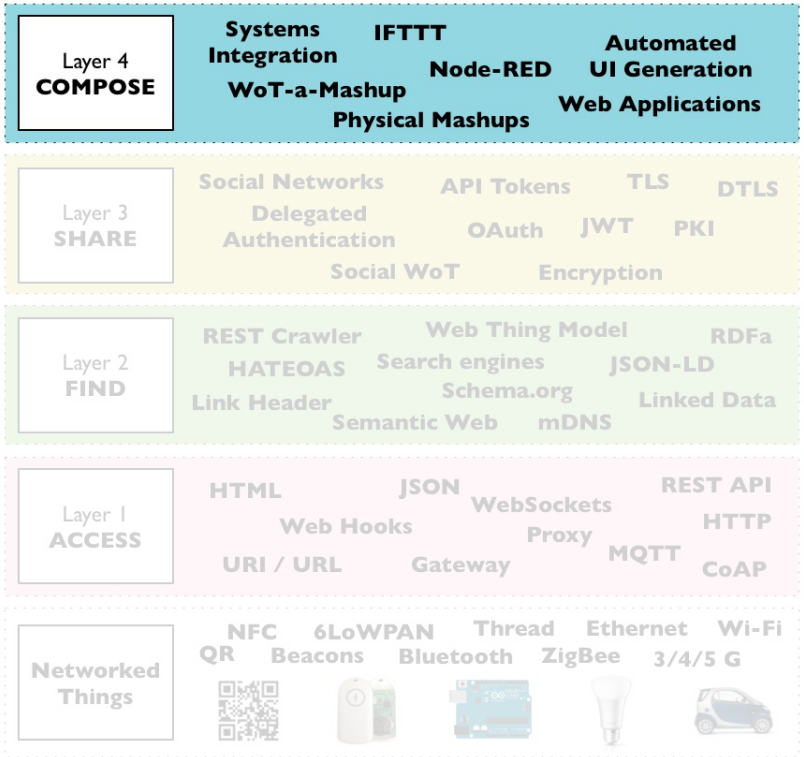
The WoT Architecture



Level 3 - Share

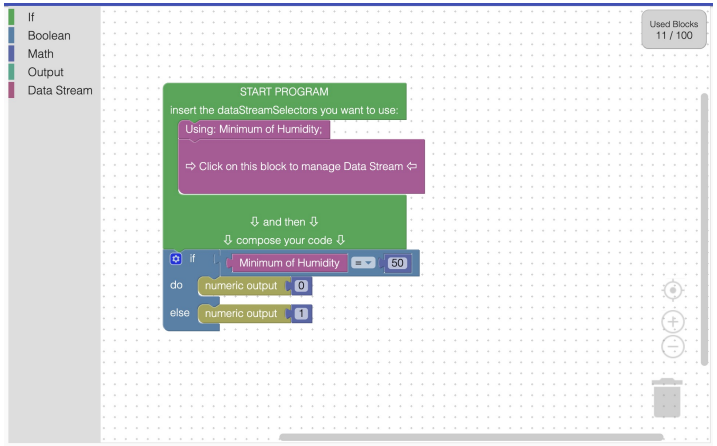
- Things can be reffered by URIs
- So sharing can take place by simple querys
- Clearly Access control can be implemented
  - In order not to give access to unintended data
  - Or to perform actions

The WoT Architecture



Level 4 - Compose

- Here ”simple” things can be composed to perform more general services
- Composition can be pre-made
  - For instance by manufacturers, by defining a set of applications which can be triggered depending on the type of sensors installed
- It can be automated
- It can be performed by users
  - Through visual languages



Source: F. Montori, L. Bedogni and L. Bononi, “A Collaborative Internet of Things Architecture for Smart Cities and Environmental Monitoring”

UNIMORE

Pub/Sub on Things

- The client and the server can also act with a pub/sub mechanism
  - They both are HTTP clients AND HTTP servers
  - They use WebHooks
    - In the Client's request, also provide a call-back URL

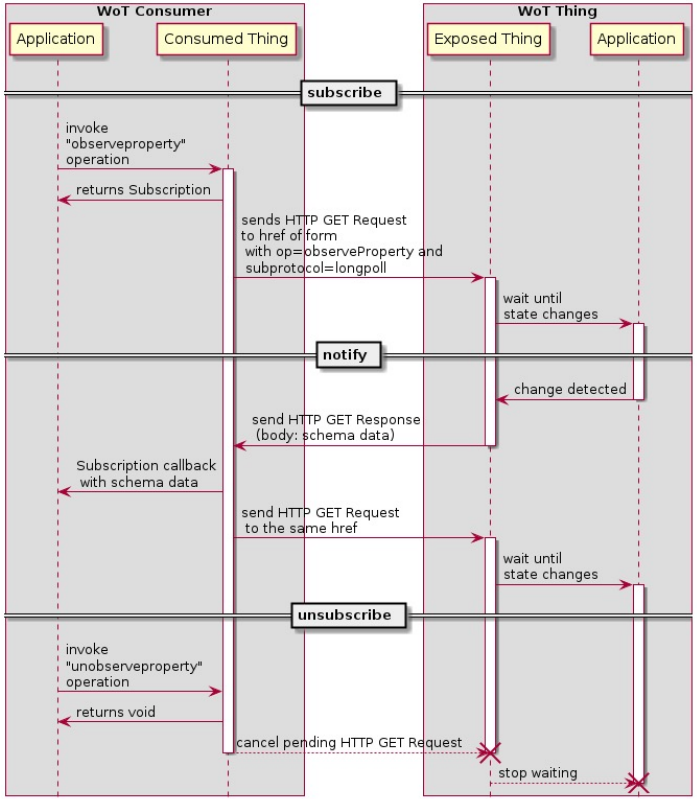
```
POST /myhome.local/garden/smartlight/subs
{"callback": "http://192.168.1.1/pubs"}
```

    - When the WoT Thing is ready to reply, it will have the URL on which it should POST the requested value

Long Polling on WoT Things

- Clients send requests to WoT Things
- WoT Things keep the request
  - Whenever a fresh value is available, the WoT Thing will reply
- To do this, add the ***observeproperty*** to the HTTP request

Long Polling



Source: https://www.w3.org/TR/wot-binding-templates

Issues to discuss

- 1: Somehow, you need to know which URI to query
- 2: You also need to know which operations can be done
- 3: You also need to know how to give meanings to values and operations

Issue 1: Discovery services

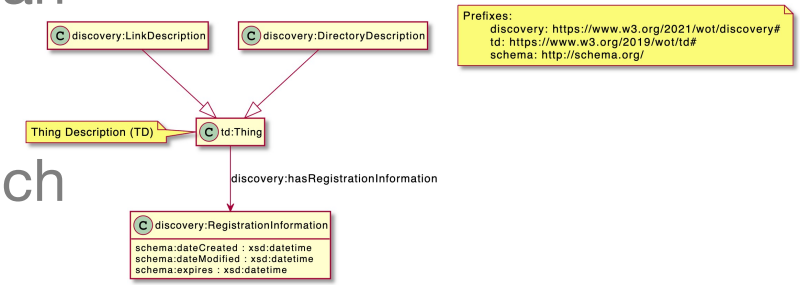
- To discover things there are many different protocols
  - DLNA, UPnP, mDNS and many others
- Basic principle
  - Send a message on the network
  - WoT Things reply
  - DNS is updated, hence they can be discovered
- W3C defines two methodologies:
  - Self-description
  - Directory

Self Description

- Each Things hosts its own Thing Description
  - Exposed through a URL
    - This can be exposed in several different ways as we already discussed
- It should use HTTPS but it may use HTTP
  - The Thing should reply to a GET method

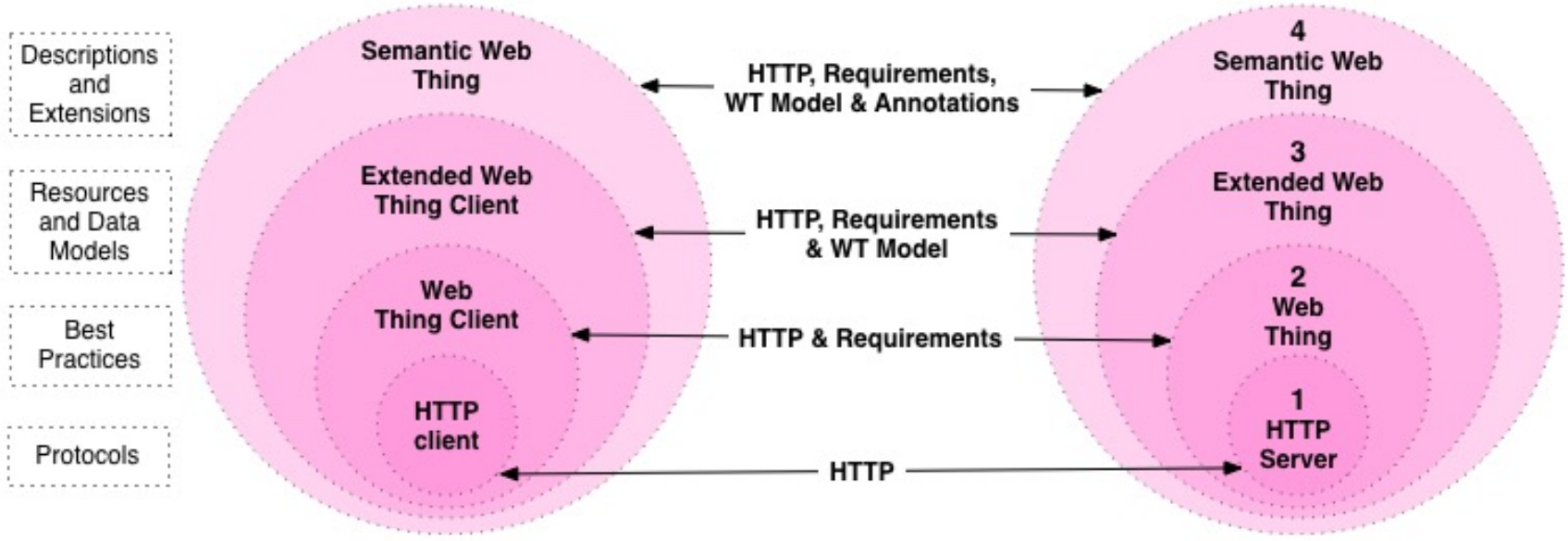
Thing Description Directory

- Each Thing Directory maintains an ontology
- It must use HTTPS
- There are different operations which can be performed by a Thing
  - Registration
    - Creation, Retrieval, Update, Deletion, Listing, Validation
  - Management
  - Notification
  - Search
  - Security and Privacy



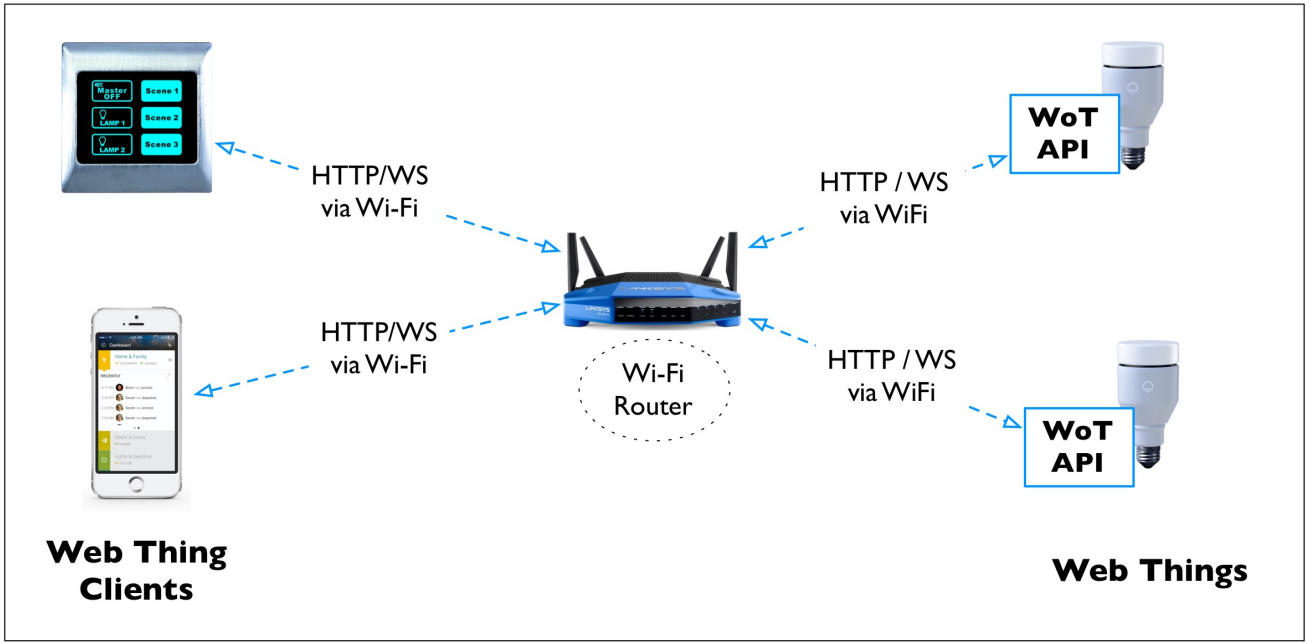
Think of it as a place where Thing description, how to access them and capabilities are stored.

Issue 2: WoT Model

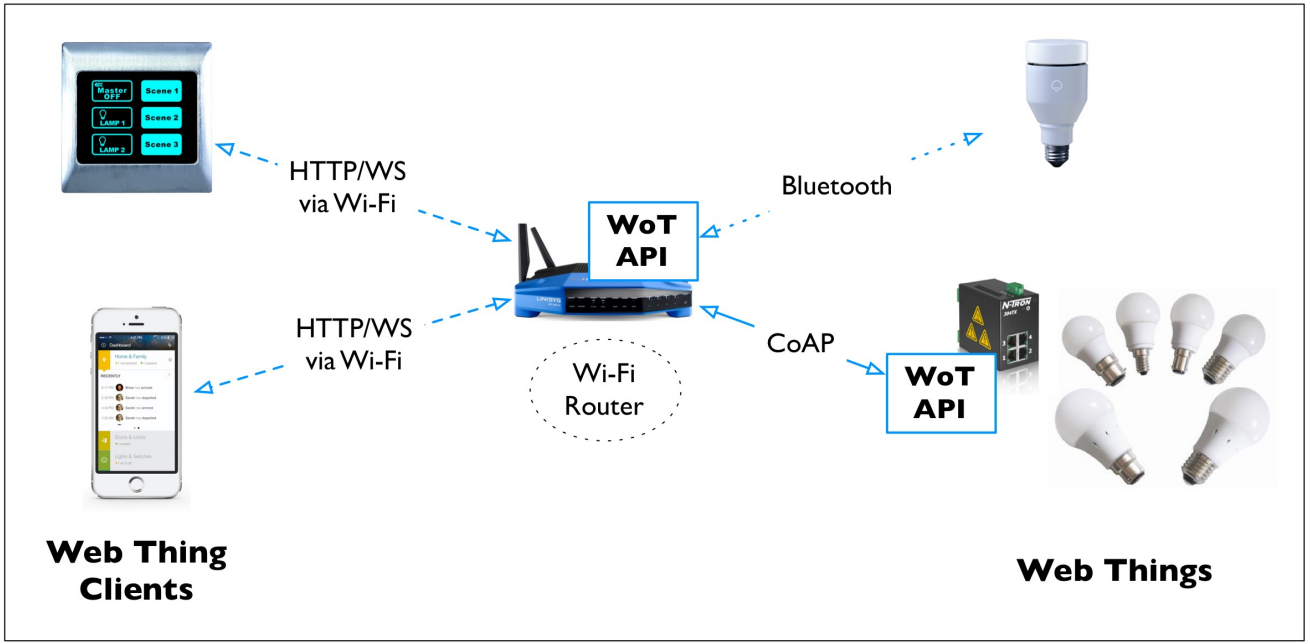


Source: Guinard, Trifa, “Building the Web of Things”

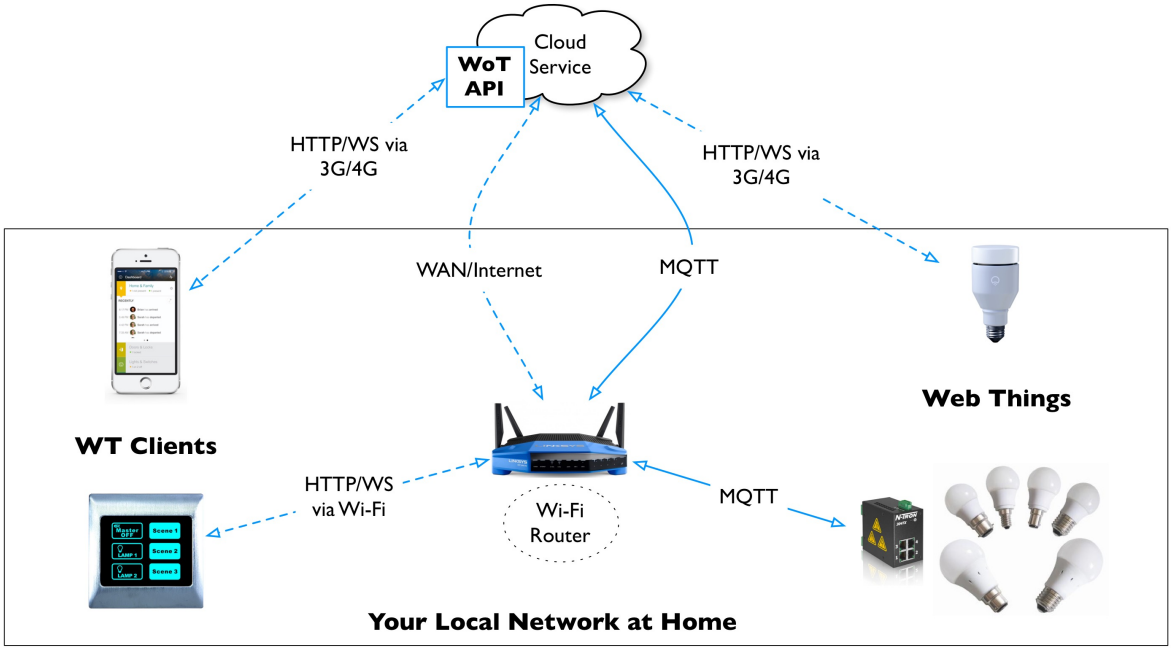
A model for the WoT
Connectivity: Direct



A model for the WoT
Connectivity: Gateway-based



A model for the WoT
Connectivity: Cloud-based



A model for the WoT

- Ex-Things MUST create a URL endpoint for all the resources that it exposes, including itself.
- Ex-Things should create a logical tree structure for resources
- Ex-Things should use relative URLs to keep JSON payloads short

REST URL	POST	GET	PUT	DELETE	OPTIONS	Details
{wt}	G,C	D,G,C	D,G,C	G,C	D,G,C	Thing Resource
{wt}/model	-	D,G,C	G,C	-	D,G,C	Model Resource
{wt}/properties	G,C	D,G,C	-	-	D,G,C	Properties Resource
{wt}/properties/{id}	-	D,G,C	G,C	-	D,G,C	Properties Resource
{wt}/actions	G,C	D,G,C	-	-	D,G,C	Actions Resource
{wt}/actions/{id}	D,G,C	D,G,C	-	-	D,G,C	Actions Resource
{wt}/actions/{id}/{actionId}	-	D,G,C	-	D,G,C	D,G,C	Actions Resource
{wt}/things	G,C	G,C	-	-	G,C	Things Resource
{wt}/things/{id}	-	G,C	G,C	G,C	G,C	Things Resource
{wt}/subscriptions	-	D,G,C	-	D,G,C	D,G,C	Subscriptions Resource

D: Direct
G: Gateway-based
C: Cloud-based

A model for the WoT

- These fields must be returned by an Ex-Thing

Field name	Type	Description
`id`	String	The URL of this resource following the URI format [RFC3986]. Relative URLs are resolved against the URL of the enclosing resource. **This field is required**.
`createdAt`	String	Timestamp when this resource was created, following the ISO8601 notation [ISO8601].
`updatedAt`	String	Timestamp when this resource was last updated, following the ISO8601 notation [ISO8601].
`name`	String	A short human-readable name for the resource.
`description`	String	A human-readable description of the resource.
`tags`	[String]	An array of tags.
`customFields`	Object	A JSON object with key-value pairs to store custom information about this resource (e.g. {"key":"value",...}).
`links`	Object	A JSON object that lists the sub-resources that this resource links to. See § 6.2 Links for details.

A model for the WoT: Example

EXAMPLE 2: Basic JSON payload

```
{
  "id": "myCar",
  "name": "My great car",
  "description": "This is such a great car.",
  "createdAt": "2012-08-24T17:29:11.683Z",
  "updatedAt": "2012-08-24T17:29:11.683Z",
  "tags": [
    "cart",
    "device",
    "test"
  ],
  "customFields": {
    "size": "20",
    "color": "blue"
  },
  "links": {
    "model": {
      "link": "model/",
      "title": "Model this Web Thing."
    },
    "properties": {
      "link": "properties/",
      "title": "Properties of this Web Thing."
    },
    "actions": {
      "link": "actions/",
      "title": "Actions of this Web Thing."
    },
    ...
  }
}
```

A model for the WoT

- Each resource may link to a number of sub-resources
- Each link is defined by a relation-type, the URL of the sub-resource and a human readable title
- Links should be

exposed by the model resource using the links field of the JSON payload

  - Or in the HTTP Link field in the HTTP Header

Relation name	Description	Section
`model`	A link to an Extended Web Thing compliant description of a resource.	Model Resource
`properties`	The properties of this resource.	Properties Resource
`actions`	The actions that this resource can perform.	Actions Resource
`things`	The Web things proxied by this resource (if applicable).	Things Resource
`subscriptions`	The endpoint to manage subscriptions to this resource.	Subscriptions Resource
`type`	Can be used to indicate that the context resource is an instance of the resource identified by the target external URL.	
`product`	A link to authoritative product information for this Web Thing.	Product
`help`	A link to the online manual page for this Web Thing.	4.8.4.4 Link type "help"
`ui`	A link to the HTML-based user interface for this Web Thing.	External Resource
`<_customRelType>`	A link to a custom resource.	Custom Resource

Source: https://model.webofthings.io/

A model for the WoT: Example

EXAMPLE 5: Links Object JSON Example

```
{
    "model": {
      "link": "model/",
      "title": "Model of this Web Thing."
    },
    "properties": {
      "link": "properties/",
      "title": "Properties of this Web Thing."
    },
    "actions": {
      "link": "actions/",
      "title": "Actions of this Web Thing."
    },
    "product": {
      "link": "https://www.raspberrypi.org/products/raspberry-pi-2-model-b/",
      "title": "Product this Web Thing is an instance of"
    },
    "type": {
      "link": "http://webofthings.org/schemas/web-thing/",
      "title": "External information about the instance type of this Thing"
    },
    "help": {
      "link": "http://webofthings.org/docs/pi/",
      "title": "Documentation"
    },
    "ui": {
      "link": "ui/",
      "title": "User Interface"
    },
    "_myCustomLinkRelType": {
      "link": "custom/",
      "schema": "http://webofthings.org/schemas/custom.html",
      "title": "My custom resource"
    },
    "_..." : {"..." : {"..."}}
}
```

Issue 3: Semantic

- There is a common model for Values as well
- However they are optional
- Still, not easy to automatically understand what a value can do
- Some proposals:
  - Ontologies
  - Semantic web

Field name	Type	Description
`<valueName>`	String	The identifier of this value **This field is required**.
`name`	String	A human-readable caption for this value.
`description`	String	A human-readable description of this value.
`type`	String	The type of this value. The supported types are `integer`, `float`, `boolean`, `string`.
`unit`	String	The unit of this value using one the (case-insensitive) full names defined in the International System of Units [SI]. Example: `"meter per second squared"`
`required`	Boolean	Specifies whether this value is required or optional. If omitted, the default is `true`.
`minValue`	Number	The minimal concrete value that this value can take.
`maxValue`	Number	The maximal concrete value of this value
`enum`	Object	Specifies a set of concrete values that this value can take, e.g. `{"LOCK":"Locks the door.", "UNLOCK":"unlocks the door."}`