Web of things

A BRIEF SURVEY

cristiano.aguzzi@unibo.it

Outline

WhoAmI

Cristiano Aguzzi

Web of Things and Semantic Technologies

- PhD student in *Structural&Enviromental Health Monitoring and Managment* (Computer Science)
- Graduated in Computer Engineering from 2017

cristiano.aguzzi@unibo.it

cris.dev()

@relucri

Background

WEB OF THINGS

Beginnings

A Web of Things Application Architecture - Integrating the Real-World into the Web (2011):

The ultimate goal of these initiatives can be summarized as trying to create a loosely coupled ecosystem of services for smart things. That is, a widely distributed platform in which the services provided by smart things can be easily composed to create new applications and use-cases

Dominique Guinard
ETH Zurich
Founder of Evrything

Beginnings

- Inspired by Web Services
- RESTFul Web architecture
  - Resource Oriented Architecture
  - HTTP as the **only** application protocol
  - Resources descripted with JSON
- The minimal Thing is a client that must implement:
  - IEEE 802 (Ethernet) / IEEE 802.11 (WiFi)
  - Web server supporting HTTP 1.1

Dominique Guinard
ETH Zurich
Founder of Evrything

Reference

site.unibo.it/wot/en/agenda/meeting

vs.inf.ethz.ch/publ/papers/dguinard-awebof-2011.pdf

New frontier

WEB OF THINGS

WoT @ W3C WG

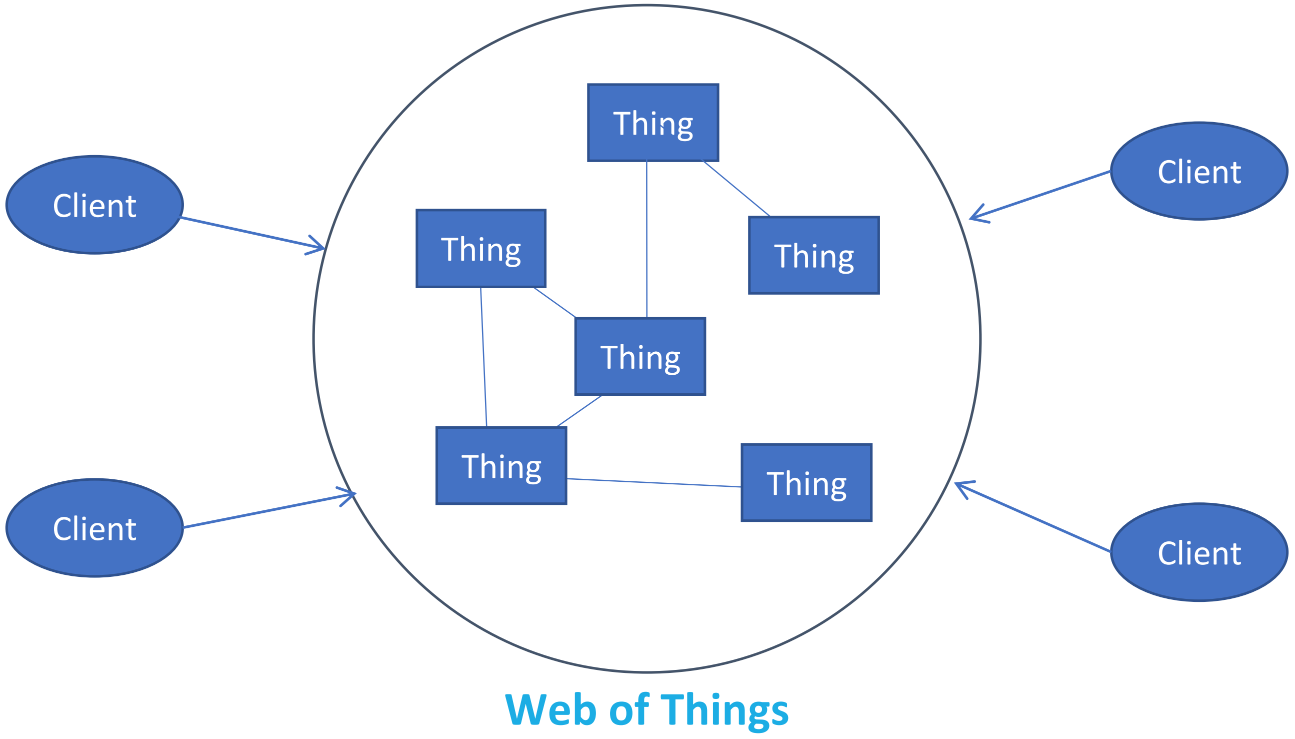
The Web of Things seeks to counter the fragmentation of the IoT through standard complementing building blocks (e.g., metadata and APIs) that **enable easy integration** across IoT platforms and application domains

Enable easy integration

Guarantee interoperability with machine understandable metadata

- Description of the data and interaction models
- Communications requirements
- Security requirements

Everything


Client
Client
Thing
Thing
Thing
Thing
Thing
Thing
Client
Client
Web of Things

Thing definition

*An abstraction of a physical or virtual entity whose metadata and interfaces are described by a **WoT Thing Description**.*

This entity can be:

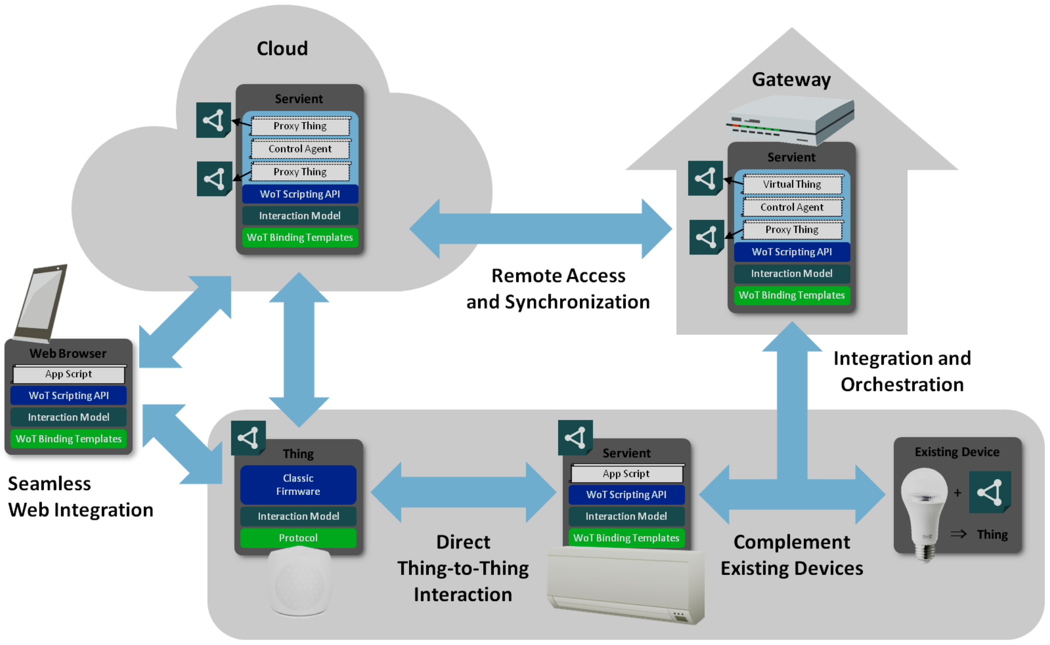
- an existing device
- a logical component of a device
- a local hardware component
- logical entity (e.g., location)

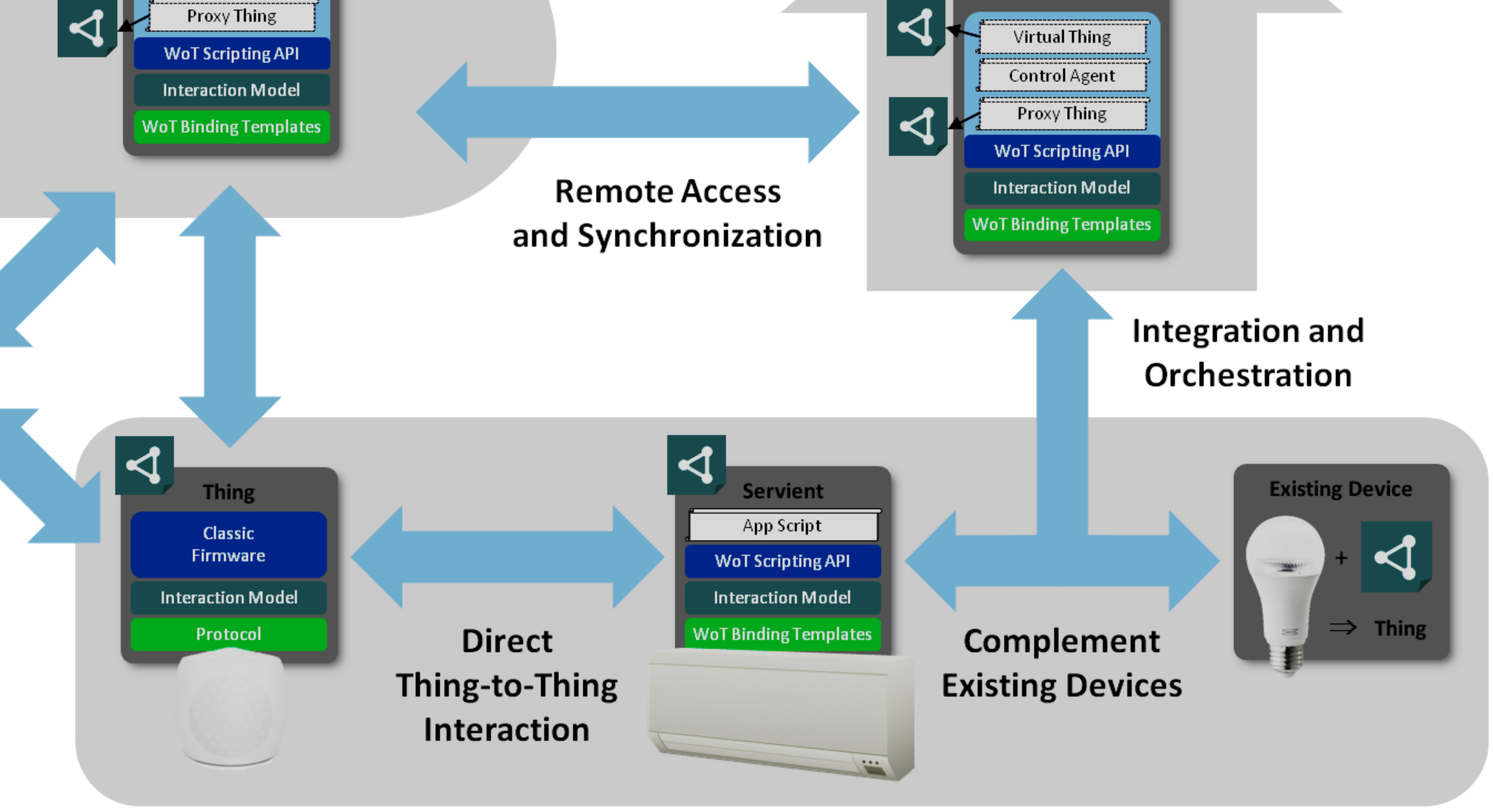
Everything that has a **Thing Description** is a Thing

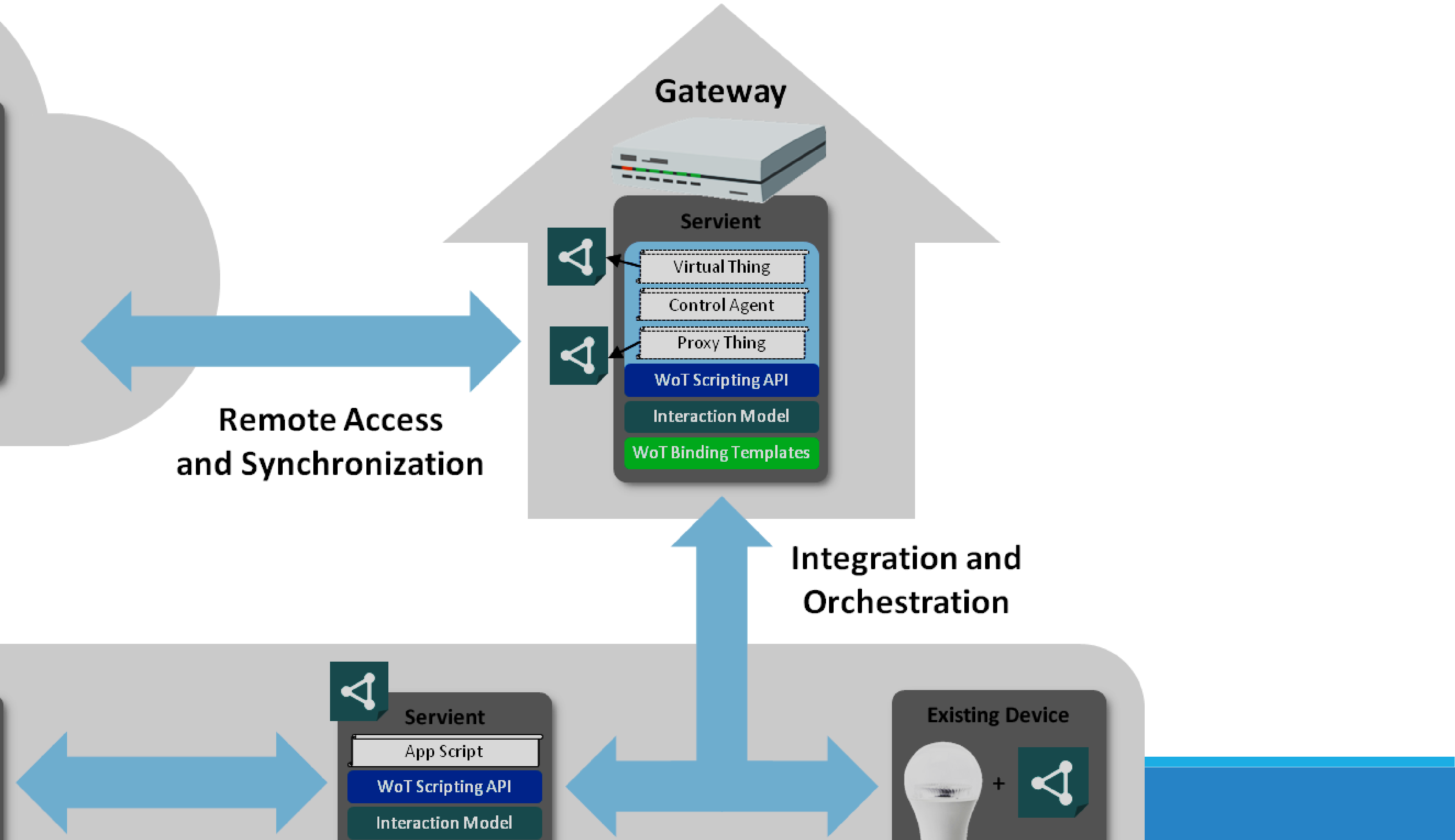
Architecture

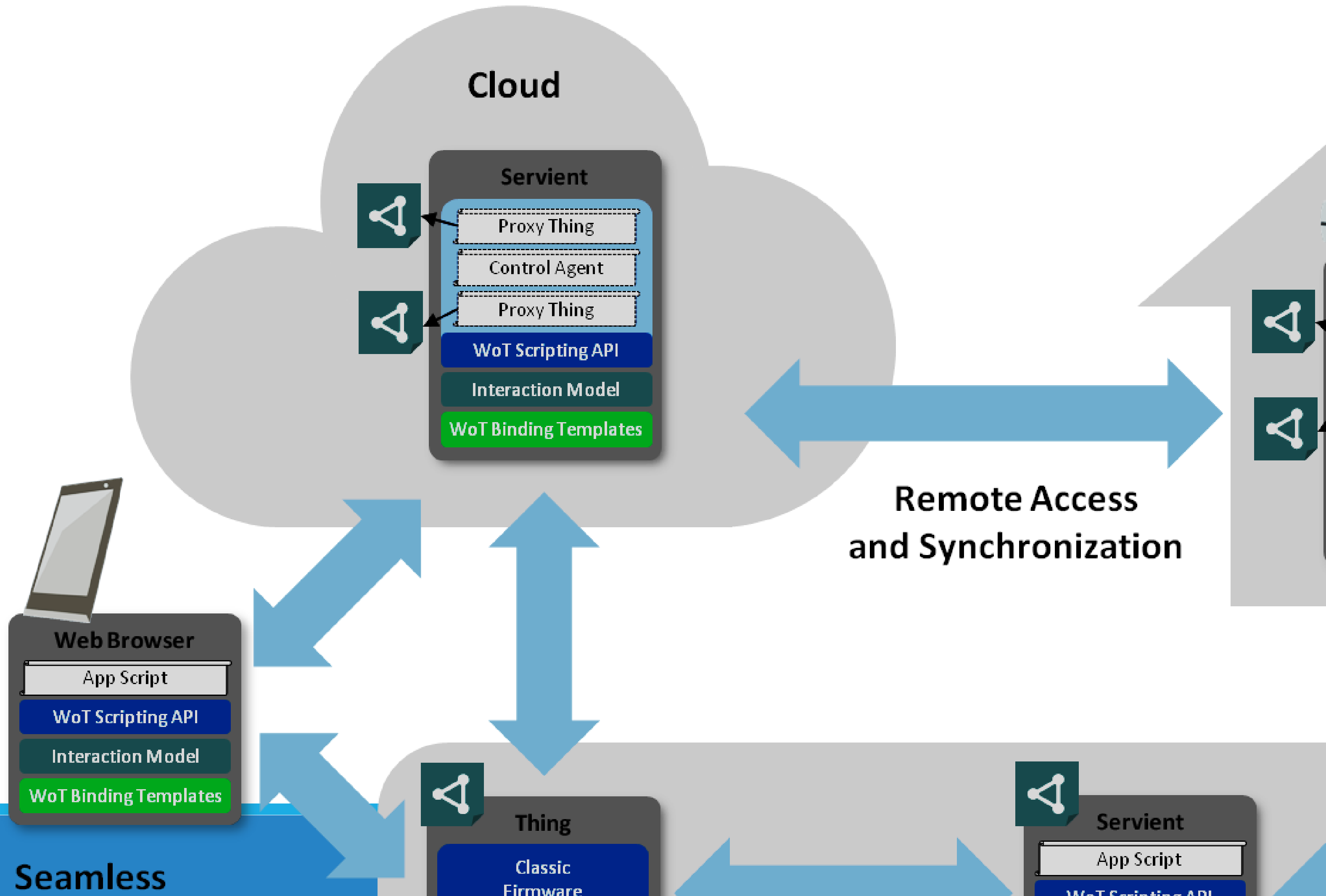
WEB OF THINGS

Overview

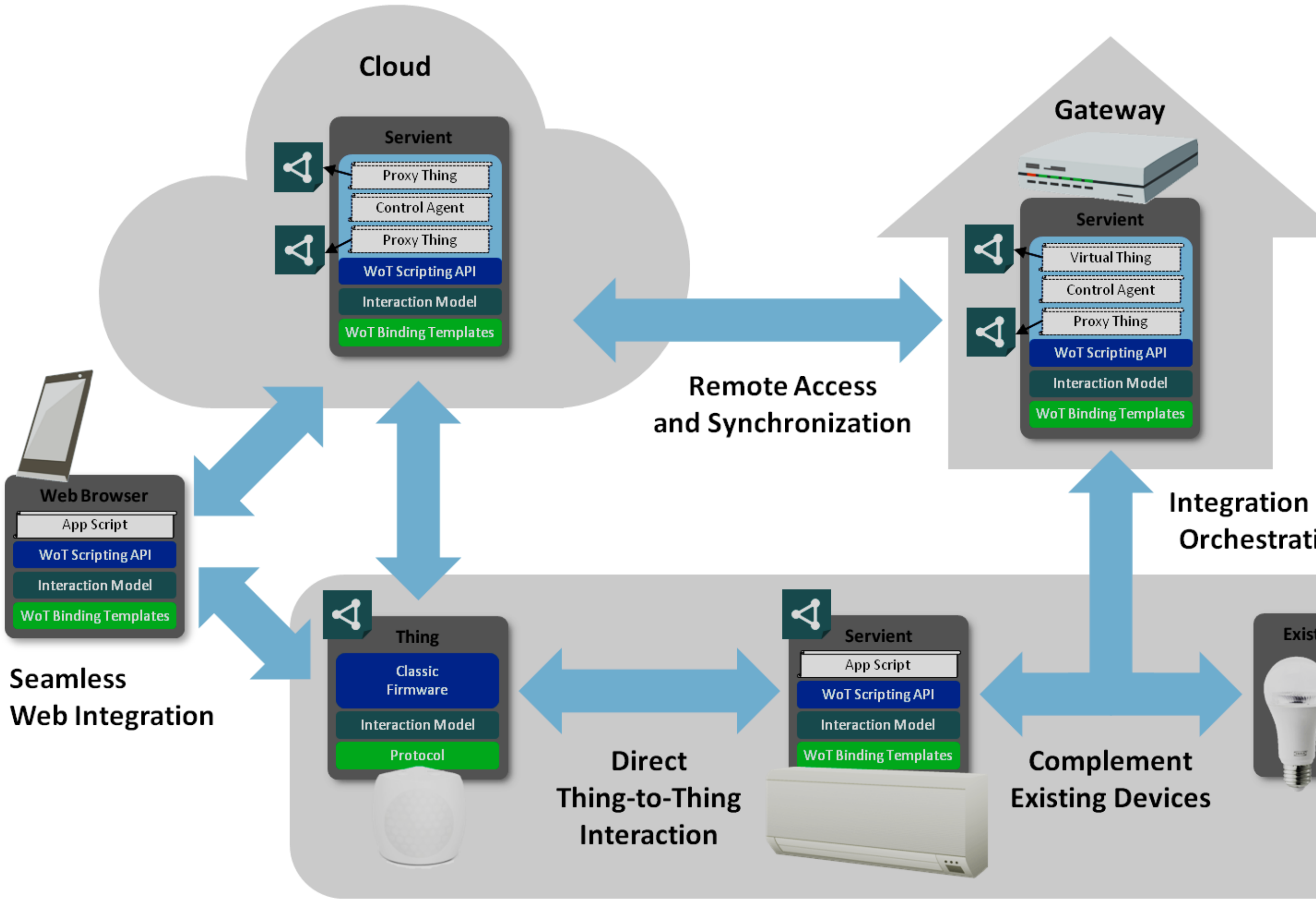

Cloud
Servient
Proxy Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Gateway
Servient
Virtual Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Remote Access
and Synchronization
Integration and
Orchestration
Web Browser
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Seamless
Web Integration
Thing
Classic
Firmware
Interaction Model
Protocol
Direct
Thing-to-Thing
Interaction
Servient
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Complement
Existing Devices
Existing Device
+
⇒ Thing


Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Virtual Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Remote Access and Synchronization
Integration and Orchestration
Thing
Classic Firmware
Interaction Model
Protocol
Direct Thing-to-Thing Interaction
Servient
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Complement Existing Devices
Existing Device
+
⇒ Thing


Gateway
Servient
Virtual Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Remote Access and Synchronization
Integration and Orchestration
Servient
App Script
WoT Scripting API
Interaction Model
Existing Device
+



Cloud
Servient
Proxy Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Remote Access
and Synchronization
Web Browser
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Thing
Classic
Firmware
Servient
App Script
Seamless


Cloud
Servient
Proxy Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Gateway
Servient
Virtual Thing
Control Agent
Proxy Thing
WoT Scripting API
Interaction Model
WoT Binding Templates
Remote Access
and Synchronization
Integration
Orchestrati
Web Browser
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Seamless
Web Integration
Thing
Classic
Firmware
Interaction Model
Protocol
Direct
Thing-to-Thing
Interaction
Servient
App Script
WoT Scripting API
Interaction Model
WoT Binding Templates
Complement
Existing Devices
Exis

Building blocks

Thing descriptor

Binding templates

Scripting API

Json-LD

RDF is very powerful but hard to handle

JSON has been extended to fully support Linked Data: **JSON-LD**

JSON-LD (JSON for Linked Data) is **fully compatible** with JSON

(i.e. every JSON-LD document is a valid JSON one)

Json-LD

Introduces new **reserved keywords** that can be used to “decorate” JSON documents:

- @type
- @id
- @context
- …

JSON-LD: Basic example

```
{
 "@context":"http://schema.org/",
 "@id":"http://ns#FabioViola",
 "name":"Fabio Viola"
}
```

```
<http://ns#FabioViola> <http://schema.org/name> "Fabio Viola" .
```

Further details

site.unibo.it/wot/en/agenda/internal-meeting-1

json-ld.org

Thing Descriptor

The **Thing Descriptor** is the core of WoT architecture. It’s the entry point of a thing and it consist in a collection of semantic metadata that describe its **interaction patterns**.

Furthermore it can have **semantic annotations** to make data models machine understandable and features for **web linking** to express relation among Things

Its default serialization is **JSON-LD**

Interaction patterns

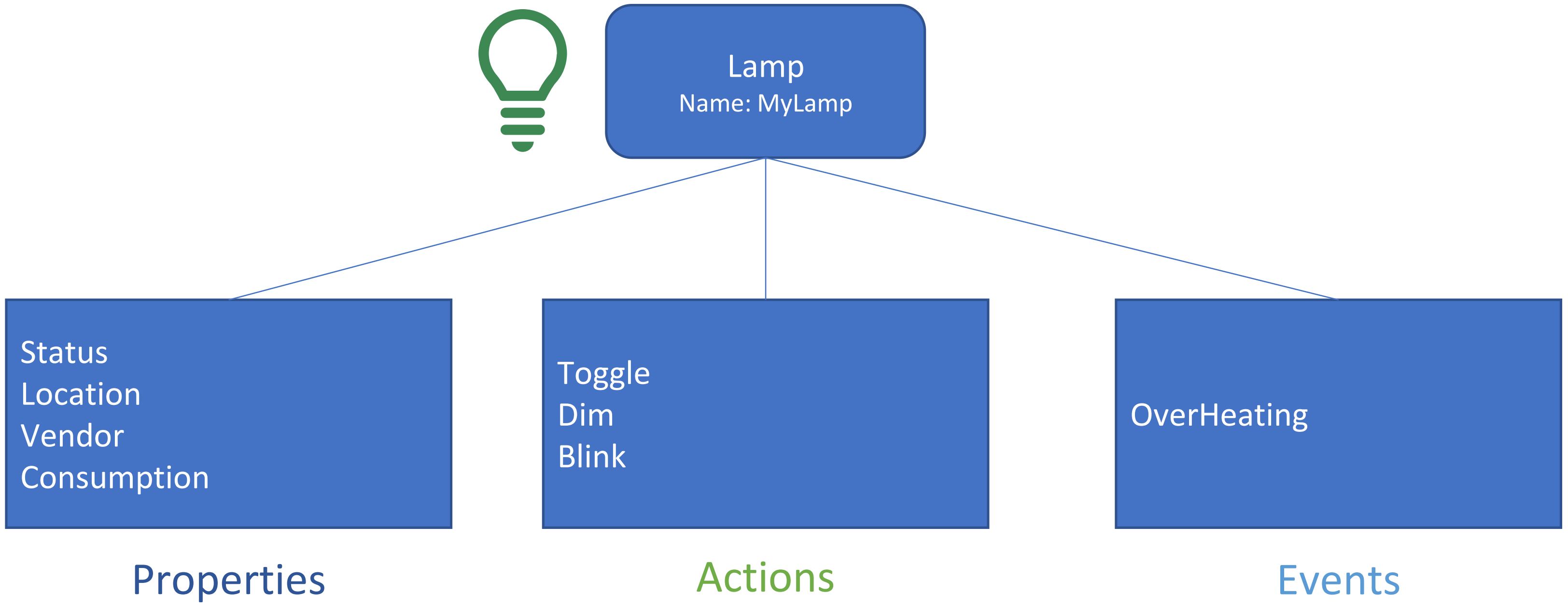
A WoT client can interact with a **Thing** using this three interaction patterns:

- Properties: A thing may have a set of properties (Read, Write)
- Actions: A client can request some processing to a Thing
- Events: A Thing can fire events and clients may subscribe to them

Interoperability

The WoT Thing Description fosters interoperability in two ways: First, and foremost, TDs enable ***machine-to-machine*** *communication in the Web of Things. Second, TDs can serve as a common, uniform format for developers to document and retrieve all details necessary to* ***access*** *IoT devices and make use of their data.*

Example


Lamp
Name: MyLamp
Status
Location
Vendor
Consumption
Toggle
Dim
Blink
OverHeating
Properties
Actions
Events

In details

```
{
  "@context":[
    "https://w3c.github.io/wot/w3c-wot-td-context.jsonld"
  ],
  "@type":[
    "Thing"
  ],
  "name":"MyLampThing",
  "interaction":[
    ...
  ]
}
```

Interaction - property

```
{
  ...
  "interaction":[
    {
      "@type":[
        "Property"
      ],
      "name":“Status",
      "schema":{
        "type":"string"
      },
      "writable":false,
      "observable":true,
      "form":[ … ]
    }

    ...
  ]
}
```

Description of the data inside this property. Schema can be the description of complex objects

Identify how to access this property. (eg. Protocol, port, host …) – **Protocol Binding data**

Interaction - Action

```
{
  …
 "interaction":[
   {
     "@type":[
       "Action"
     ],
     "name":“Dim",
     "inputSchema":{
       …
     },
     "form":[…]
   }
   …
 ]
}
```

```
{
  "type":"object",
  "field":[
    {
      "name":"brightness",
      "schema":{
        "type":"integer",
        "@type":[
          "iot:DimmerData"
        ],
        "minimum":0,
        "maximum":255
      }
  ],
  "required":[
    "brightness"
  ]
}
```

Interaction - Event

```
{
  "@type":[
    "Event",
    "iot:TemperatureExceed"
  ],
  "name":“OverHeating",
  "schema":{
    "type":"string"
  },
  "form":[…]
}
```

Further details

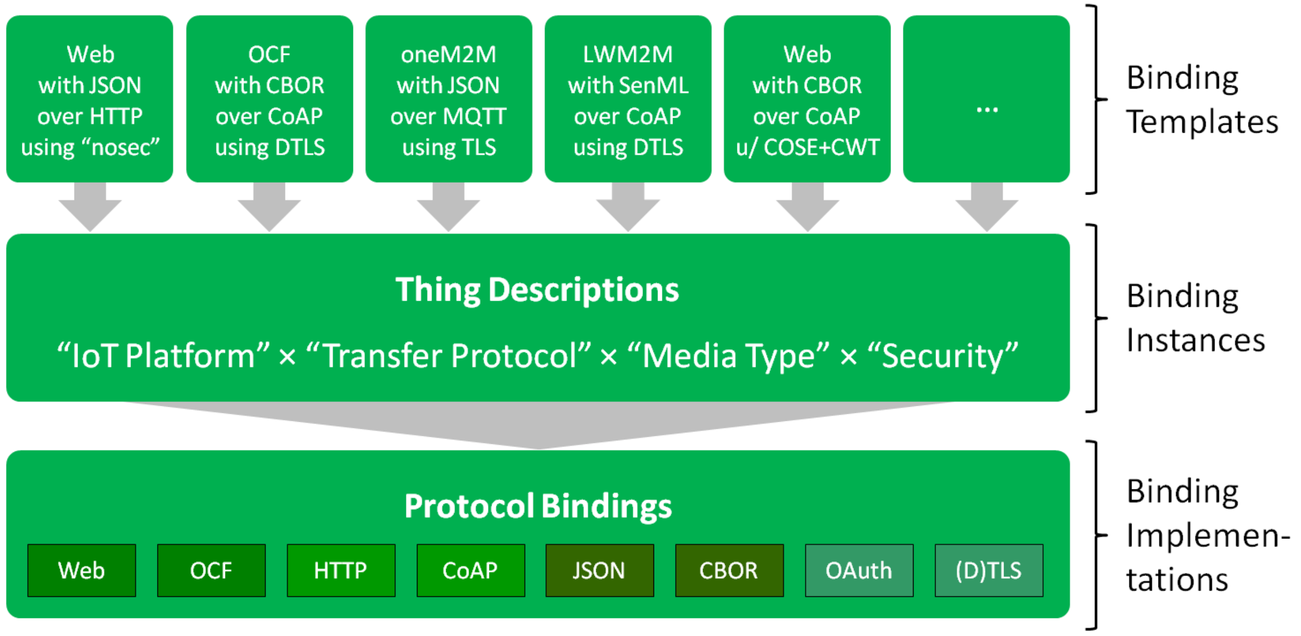
w3c.github.io/wot-thing-description

Binding templates

Problem: enable interactions with a myriad of different IoT Platforms

Solution: define multiple vocabularies (**Binding Template**) to describe communication between Things and provide **extension points** in the Thing Descriptor.

Binding templates


Web with JSON over HTTP using "nosec"
OCF with CBOR over CoAP using DTLS
oneM2M with JSON over MQTT using TLS
LWM2M with SenML over CoAP using DTLS
Web with CBOR over CoAP u/ COSE+CWT
…
Binding Templates
Thing Descriptions
"IoT Platform" × "Transfer Protocol" × "Media Type" × "Security"
Binding Instances
Protocol Bindings
Web
OCF
HTTP
CoAP
JSON
CBOR
OAuth
(D)TLS
Binding Implemen-tations

Protocol binding

A protocol binding enable the communication with a particular IoT platform or protocol or software stack.

It is similar to a driver for a digital device and uses a *binding instance* declared at *interaction pattern* level for configuration.

Wot Interface - verbs

ReadProperty

WriteProperty

ObserveProperty

InvokeAction

SubscribeEvent

UnsubscribeEvent

Td extension: Form element

*The "form" element contains the **URI** pointing to an instance of the interaction and **descriptions** of the protocol settings and **options** expected to be used when between the client and server for the interaction*

In practice

```
{
  "interaction":[
    {
      "name":"Status",
      "@type":[
        "Property"
      ], "schema " : { … }
      "writable":false,
      "observable":false,
      "form":[
        {
          "href":"/example/light/currentswitch",
          "mediatype":"application/json"
        }
      ]
    }
  ]
}
```

The property Switch State can be accessed with HTTP using */example/light/currentswitch* path.

More complex

Interaction resource URI

```
"form":[
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "readProperty"
    ],
    "http:methodName":"http:get"
  },
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "writeProperty"
    ],
    "http:methodName":"http:post"
  },
  {
    "href":"mqtt://example.com/example/light/currentswitch",
    "rel":[
      "observeProperty"
    ],
    "mqtt:methodName":"mqtt:subscribe"
  }
]
```

More complex

Interaction resource URI

WoT Interface Verb

```
"form":[
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "readProperty"
    ],
    "http:methodName":"http:get"
  },
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "writeProperty"
    ],
    "http:methodName":"http:post"
  },
  {
    "href":"mqtt://example.com/example/light/currentswitch",
    "rel":[
      "observeProperty"
    ],
    "mqtt:methodName":"mqtt:subscribe"
  }
]
```

More complex

Interaction resource URI

WoT Interface Verb

Specific vocabulary configuration

https://w3c.github.io/wot-binding-templates/#form-vocabulary

```
"form":[
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "readProperty"
    ],
    "http:methodName":"http:get"
  },
  {
    "href":"/example/light/currentswitch",
    "mediaType":"application/json",
    "rel":[
      "writeProperty"
    ],
    "http:methodName":"http:post"
  },
  {
    "href":"mqtt://example.com/example/light/currentswitch",
    "rel":[
      "observeProperty"
    ],
    "mqtt:methodName":"mqtt:subscribe"
  }
]
```

Reference

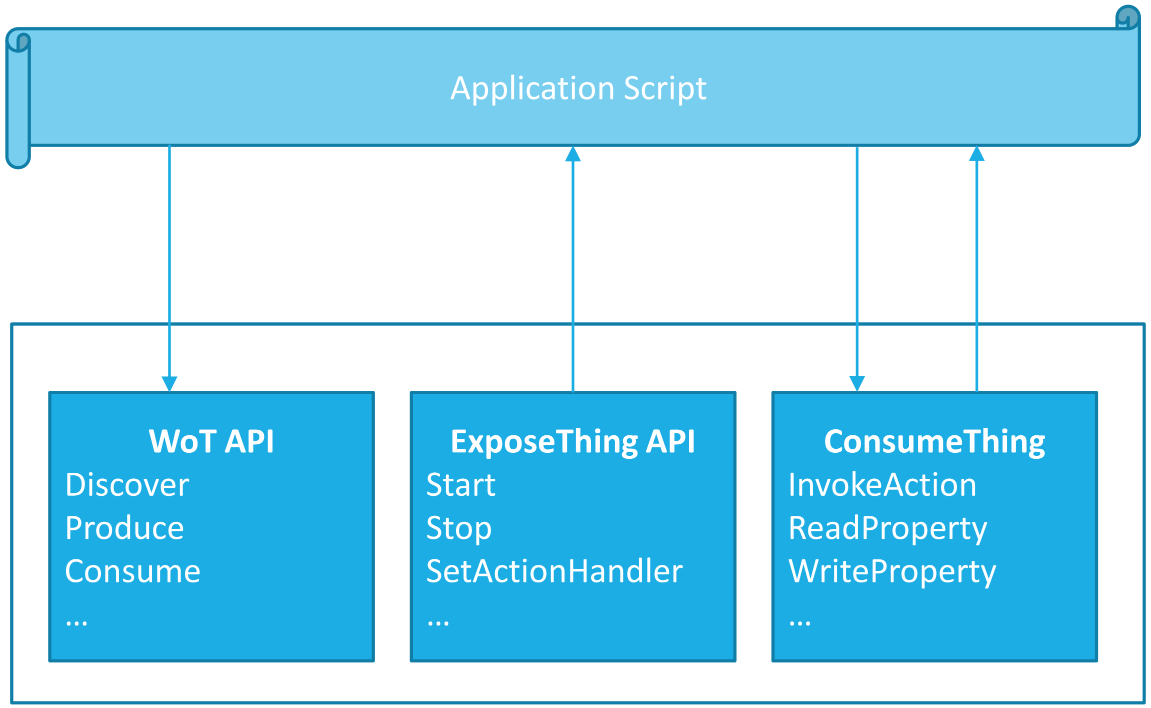
w3c.github.io/wot-binding-templates/

Scripting API

The WoT Scripting API enables having a runtime system for IoT applications.

- Improve **productivity**
- Reduce **integration** costs
- Enable **portability** for application modules

Scripting API



Wot API

```
interface WoT{
        Observable<ConsumedThing> discover(optional ThingFilter filter);
        Promise<ThingDescription> fetch(USVString url);
        ConsumedThing consume(ThingDescription td);
        ExposedThing produce(ThingModel model);
};
typedef USVString ThingDescription;
typedef (ThingTemplate or ThingDescription) ThingModel;
```

ThingFilter

```
dictionary ThingFilter {
        DiscoveryMethod method = "any";
        USVString url;
        USVString query; //SPARQL
        sequence<Dictionary> constraints;
};
```

Example: Local discovery

```
let subscription = wot.discover({
            method: "nearby",
            constraints: [{ protocol: "BLE-4.2" }, { protocol: "NFC"}]
}).subscribe(
             thing => { console.log("Found nearby Thing " + thing.name); },
            error => { console.log("Discovery error: " + error.message); },
            () => { console.log("Discovery finished successfully");
} );
```

ConsumeThing API

```
interface ConsumedThing {
        readonly attribute DOMString name;
        ThingDescription getThingDescription();
        Promise<any> invokeAction(DOMString name, any parameters);
        Promise<void> writeProperty(DOMString name, any value);
        Promise<any> readProperty(DOMString name);
        Observable onEvent(DOMString name);
        Observable onPropertyChange(DOMString name);
        Observable onTDChange();
};
```

ExposeThing API

```
ExposedThing implements ConsumedThing;
interface ExposedThing {
// define how to expose and run the Thing
        Promise<void> start();
        Promise<void> stop();
        Promise<void> register(optional USVString directory);
        Promise<void> unregister(optional USVString directory);
        Promise<void> emitEvent(DOMString eventName, any payload
);
// define Thing Description modifiers
ExposedThing addProperty(ThingProperty property);
ExposedThing removeProperty(DOMString name);
ExposedThing addAction(ThingAction action);
ExposedThing removeAction(DOMString name);
ExposedThing addEvent(ThingEvent event);
ExposedThing removeEvent(DOMString name);
// define request handlers
ExposedThing setActionHandler(ActionHandler action, optional DOMString actionName);
ExposedThing setPropertyReadHandler(PropertyReadHandler readHandler, optional DOMString propertyName);
ExposedThing setPropertyWriteHandler(PropertyWriteHandler writeHandler, optional DOMString propertyName); };
callback ActionHandler = Promise<any> (any parameters);
callback PropertyReadHandler = Promise<any> ();
callback PropertyWriteHandler = Promise<void> (any value);
```

Code!

GITHUB.COM/THINGWEB/NODE-WOT

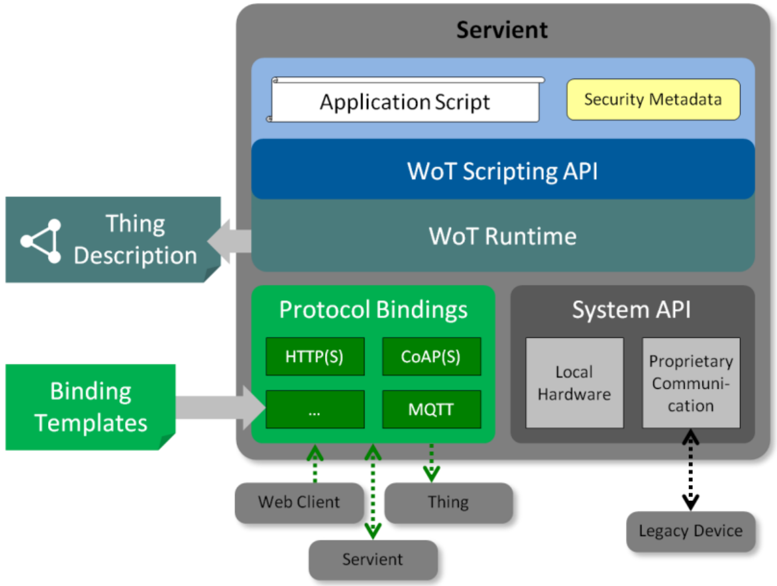
Servient

The core node of the WoT architecture is the **Servient**

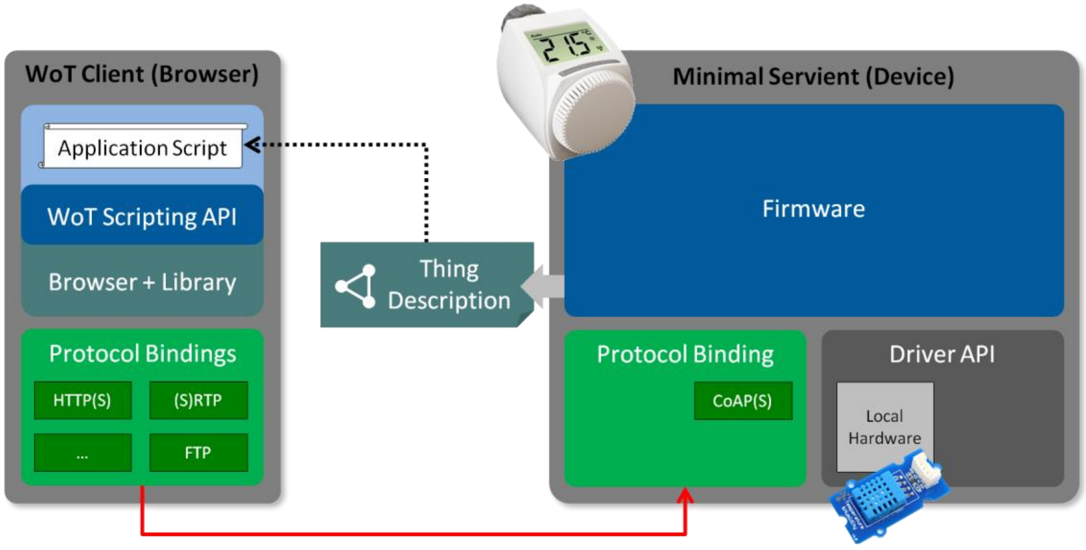
A Servient is a software stack that implements the **WoT building blocks**. Servients can host and **expose** Things and/or **consume** Things. Thus, Servients can perform in both the server and client roles.

Servient

- **Application**: Thing business logic; implement or using a script or in the firmware
- **WoT Scripting API**: contract between applications and the runtime system (Optional Component)
- **WoT Runtime**: contains Thing and interaction model abstractions. (Optional Component)
- **Protocol Bindings**: implementations of Binding templates, the actual network interface between things
- **System API**: things can access local hardware or system services. (out of scope of WoT standardization)



Minimal servient



Discover things

Things capabilities can be discovered throughout their **thing descriptor**.

The discovering process can search different levels:

- **Local**: Thing defined in the same device (no network operation)
- **Nearby**: Spatial locality discovering. A device is “near” if it’s in range of a wireless protocol. (Bluethoot, NFC …)
- **Directory**: use a remote service to discover Things.
- **Broadcast**: open ended discovery based on sending a request to a broadcast address
- **Other**: Proprietary discovering protocol

Thing directory

A thing directory can collect **TD**s and offer services like a SPARQL endpoint to search for a particular Thing

Must be aligned with the CoRE **Resource Directory** specification[1]

May provide Web interface for lookups, usually including a SPARQL endpoint for **semantic queries**

[1] *https://tools.ietf.org/html/draft-ietf-core-resource-directory-11*

CoRE Resource Directory

Designed to use WebLinking discovering process in Constrained RESTful Environments

Stores links in the CoRE link format wich can be inserted in **Groups**

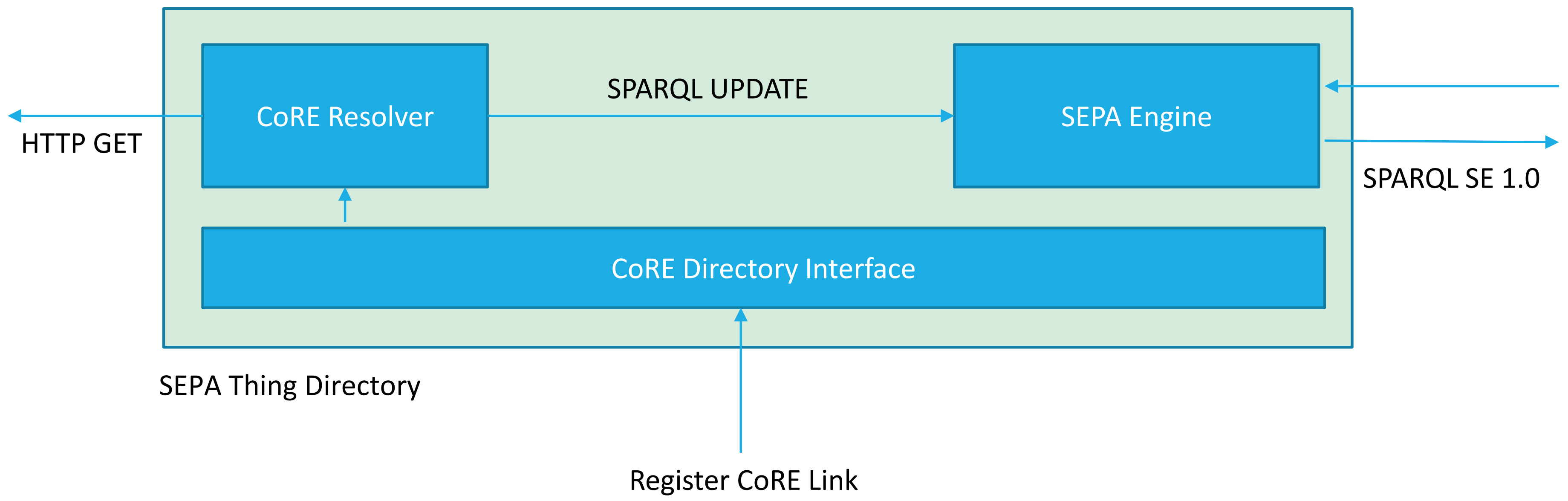
RESTful interface by definition:

- Registration
- Registration updates
- Removal
- Automatic removal after a given lifetime

Lookup based on link format

RFC defines also common scenarios and mechanisms to discover thing directory itself

Discover things - SEPA



Cocktail protocol template

Create a protocol template to enable thing to thing interaction trought **SEPA engine**

```
"form":[
        {
          "href":“sepahost:3456",
          "mediatype":"application/json"
          "sepa:jsap" : "sepahost:3456/thing_interaction.jsap"
          "sepa:update" : "UPDATE_PROPERTY_VALUE"
        }
]
```

Thank you for your attention

Web of things

A BRIEF SURVEY

cristiano.aguzzi@unibo.it