WebThings

An introduction...

F. Viola[1]

[1]ARCES
University of Bologna

Feb. 2017

Outline

The Web of Things

The Internet of Things (IoT) suffers from a lack of interoperability across platforms. The aim of **Web of Things** initiative is to bring harmony in the fragmented market of the Internet of Things.

Before we begin...

In this presentation the **Web Thing Model** will be our Bible.

Produced mainly by EVRYTHING, this is a cookbook for integrating Things to the Web.

Outline

The Web Thing Model and the W3C

How the Web Thing Model relates to the W3C?

The Web Thing Model was only the beginning of the Web of Things work at W3C. The relationship with the current WoT IG work is easy: the **Integration Patterns** of the Web Thing Model are similar to what the WoT IG Architecture Document references.

Similarly, the Thing Description of the WoT IG takes a very similar approach and terminology to the Web Thing Model.

The Web Thing Model and the W3C

How the Web Thing Model relates to the W3C?

However, there are also differences: the Web Thing Model focuses solely on readily Internet (TCP/IP, UDP) and browsers compatible protocols (WebSocket, HTTP, HTTP/2) and relies on translators for other protocols.

The WoT IG Architecture Document extends this and proposes the notion of Protocols Bindings. A Protocol Binding is a way to map an existing IoT protocol to the Thing Description and its interaction patterns.

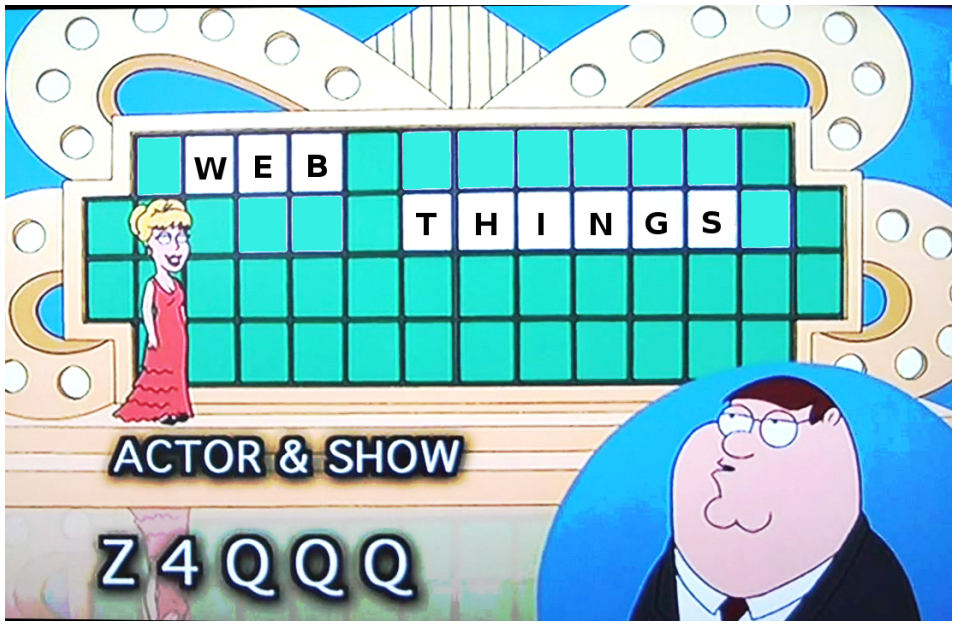
The Web Thing Model and the W3C

So, do we need the Web Thing Model?

The Web Thing Model is a first step in the world of Web of Things. Our second step will be the analysis of the WoT IG's Thing Description.

What is a WebThing?

A **Web Thing** is a digital representation of a physical object accessible via a **RESTful web API**.



. . . so let's review the main concepts of REST.

REST in Peace

Do you remember the previous lesson about REST?

With an **element** (e.g. *http://example.com/car14*), HTTP verbs allows to:

- retrieve the representation (**GET**)
- update the element (**PUT**)
- create the element (**POST**)
- delete the element (**DELETE**)

REST in Peace

Do you remember the previous lesson about REST?

With an **element** (e.g. *http://example.com/car14*), HTTP verbs allows to:

- retrieve the representation (**GET**)
- update the element (**PUT**)
- create the element (**POST**)
- delete the element (**DELETE**)

With a **collection** (e.g. *http://example.com/cars*), HTTP verbs allows to:

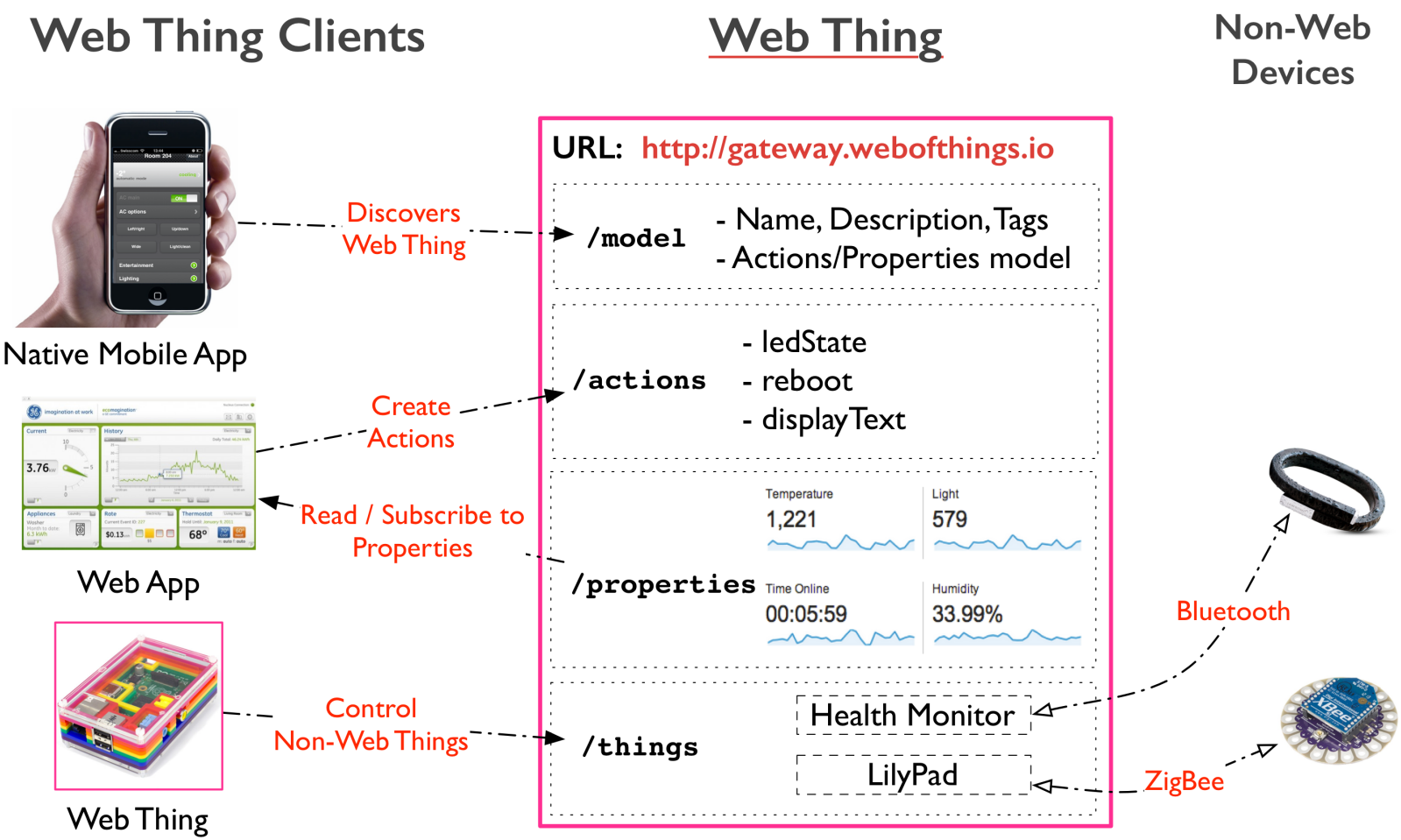
- retrieve the list of elements (**GET**)
- update the entire collection (**PUT**)
- create a new element in the collection (**POST**)
- delete the entire collection (**DELETE**)

The Web Thing Model

The data model of a Web Thing is composed by the following **resources**:

- **Things**: this resource contains all the web Things that are proxied by this web Thing;
- **Model**: a set of metadata that defines various aspects about it such as its name, description, or configurations;
- **Properties**: properties represent the internal state of a web Thing. Clients can subscribe to properties to receive a notification message when specific conditions are met; for example, the value of one or more properties changed;
- **Actions**: an action is a function offered by a web Thing.

The Web Thing Model



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server
- have a root resource accessible via an HTTP URL

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server
- have a root resource accessible via an HTTP URL
- support GET, POST, PUT and DELETE HTTP verbs

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server
- have a root resource accessible via an HTTP URL
- support GET, POST, PUT and DELETE HTTP verbs
- implement HTTP status codes 200, 400, 500

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server
- have a root resource accessible via an HTTP URL
- support GET, POST, PUT and DELETE HTTP verbs
- implement HTTP status codes 200, 400, 500
- support JSON as default representation

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MUST**:

- at least be an HTTP/1.1 server
- have a root resource accessible via an HTTP URL
- support GET, POST, PUT and DELETE HTTP verbs
- implement HTTP status codes 200, 400, 500
- support JSON as default representation
- support GET on its root URL

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

- use secure HTTP connections (HTTPS)

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

- use secure HTTP connections (HTTPS)
- implement the WebSocket Protocol

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

- use secure HTTP connections (HTTPS)
- implement the WebSocket Protocol
- support the Web Things model (!!!)

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

- use secure HTTP connections (HTTPS)
- implement the WebSocket Protocol
- support the Web Things model (!!!)
- return a 204 for all write operations (NO CONTENT)

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **SHOULD**:

- use secure HTTP connections (HTTPS)
- implement the WebSocket Protocol
- support the Web Things model (!!!)
- return a 204 for all write operations (NO CONTENT)
- provide a default human-readable documentation

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MAY**:

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MAY**:

- support the HTTP OPTIONS verb for each of its resources

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MAY**:

- support the HTTP OPTIONS verb for each of its resources
- provide additional representation mechanisms (RDF, XML, JSON-LD)

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MAY**:

- support the HTTP OPTIONS verb for each of its resources
- provide additional representation mechanisms (RDF, XML, JSON-LD)
- offer a HTML-based user interface

The Web Thing Model

The requirements to build a Web Thing are classified in three levels: **MUST**, **SHOULD** and **MAY**.

A Web Thing **MAY**:

- support the HTTP OPTIONS verb for each of its resources
- provide additional representation mechanisms (RDF, XML, JSON-LD)
- offer a HTML-based user interface
- provide precise information about the intended meaning of individual parts of the model

Outline

OUR Web Thing

In the rest of the presentation we will build our Web Thing following the Web Thing Model.

Our Web Thing is a simple temperature sensor that can be rebooted with a specific command.

So we can identify a **property** (i.e. the temperature) and an **action** (i.e. reboot). So let's see how I implemented the Web Thing!

The Web Thing Model

A Web Thing **MUST** at least be an HTTP/1.1 server:

I used Flask, one of the most powerful Python modules for building a Web server.

The Web Thing Model

A Web Thing **MUST** have a root resource accessible via an HTTP URL:

Since the Web Thing runs on `http://mml.arces.unibo.it:10996`, this is also the URI of the root.

The Web Thing Model

A Web Thing **MUST** support GET, POST, PUT and DELETE HTTP verbs

- {wt}/model GET provides the model
- {wt}/actions GET provides a list of the actions
- {wt}/actions/*action* POST requests the execution of the *action*
- {wt}/actions/*action*/*id* GET returns the execution status of the *action* with code *id*
- {wt}/properties GET provides a list of the properties
- {wt}/properties/*prop* GET provides the value of the *property*
- {wt}/properties/*prop* PUT sets the value of the *property*
- ...

The Web Thing Model

A Web Thing **MUST** *implement HTTP status codes 200, 400, 500.*

This task is achieved. For every successfull request 20* is returned. A client-side error results in a 40* error, while a server-side error gives a 50* status code.

The Web Thing Model

A Web Thing **MUST** *support JSON as default representation.*

A simple test allows to verify that we support JSON as the default representation. This is possible thanks to the `jsonify` function provided by Flask.

The Web Thing Model

A Web Thing **MUST** *support GET on its root URL.*

As previously said, `http://mml.arces.unibo.it:10996` represents the root resource. My Python Flask application supports accessing the resource with the GET verb.

The Model

As we said before, the model of our Web Thing is reachable at:

`http://mml.arces.unibo.it:10996/model` with **GET**

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/model
```

Properties

A list of the properties can be obtained with:

`http://mml.arces.unibo.it:10996/properties` with **GET**

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/properties
```

Properties

Values of a property can be obtained with:

`http://mml.arces.unibo.it:10996/properties/temperature` (**GET**)

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/properties/temperature
```

Properties

A value can be set with:

`http://mml.arces.unibo.it:10996/properties/temperature` (**PUT**)

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/properties/temperature \
-X PUT -H "Content-Type: application/json" \
--data '{"temperature":24}'
```

Actions

A list of the actions can be obtained with:

`http://mml.arces.unibo.it:10996/actions` with **GET**

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/actions
```

Actions

An action can be invoked with:

`http://mml.arces.unibo.it:10996/actions/reboot` (**POST**)

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/actions/reboot -X POST
```

Actions

The result of an action can be verified with:

`http://mml.arces.unibo.it:10996/actions/reboot/<ID>` (**GET**)

With cURL:

```
$ curl http://mml.arces.unibo.it:10996/actions/reboot/<ID>
```

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The class that compose my Web Thing are:

- **WebThing** – the main class that compose the Web Thing

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The class that compose my Web Thing are:

- **WebThing** – the main class that compose the Web Thing
- **WebThingProperty** – the class to implement a property

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The class that compose my Web Thing are:

- **WebThing** – the main class that compose the Web Thing
- **WebThingProperty** – the class to implement a property
- **WebThingAction** – the class that implement an action

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The class that compose my Web Thing are:

- **WebThing** – the main class that compose the Web Thing
- **WebThingProperty** – the class to implement a property
- **WebThingAction** – the class that implement an action
- **WebThingPropertyList** – the class to handle a list of properties

The code is hosted on a git repository on BitBucket (SEPA/WebThing).

The program to execute with `python3` is `wt.py`.

The class that compose my Web Thing are:

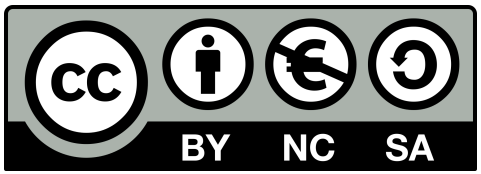
- **WebThing** – the main class that compose the Web Thing
- **WebThingProperty** – the class to implement a property
- **WebThingAction** – the class that implement an action
- **WebThingPropertyList** – the class to handle a list of properties
- **WebThingActionList** – the class to handle a list of actions

Outline

Conclusion

Thank you for the attention!

This presentation is released with license:
Creative Commons 3.0 - BY,NC,SA



The source code of my Web Thing is released with license:
GNU GPL v3