Toward Constrained Semantic WoT

Remy Rojas, Lionel Médini, Amélie Cordier

Toward Constrained Semantic WoT

Remy Rojas
Univ Lyon
LIRIS, CNRS UMR5205
F-69622, Villeurbanne, France
remy.rojas@protonmail.com

Lionel Médini
Univ Lyon, Université Lyon 1
LIRIS, CNRS UMR5205
F-69622, Villeurbanne, France
firstname.lastname@liris.cnrs.fr

Amélie Cordier
Univ Lyon, Université Lyon 1
LIRIS, CNRS UMR5205
F-69622, Villeurbanne, France
firstname.lastname@liris.cnrs.fr

ABSTRACT

More and more things are now connected to the Internet and become part of the Web of Things. The notion of "thing" encompasses various types of devices, from complex robots to simple sensors. In particular, things may rely on limited memory, storage and computing capabilities. We propose an architecture able to embed both semantic and RESTful technologies into constrained things, while being generic and reconfigurable. It combines emerging standards such as CoAP and Hydra to split RDF graphs, process requests, and generate responses on-the-fly. We validate our proposition by implementing such a Server in an Arduino UNO.

Keywords

Semantic Web of Things; constrained, embedded, REST, Hydra, CoAP, arduino

1. INTRODUCTION

One of the WoT challenges is to provide building blocks for applications able to compose services offered by things as standard Web resources [4]. Relying on the Web – and REST principles – as a platform provides WoT applications with efficiency and scalability. One trend of the Web community consists in providing semantic, hypermedia-based documentations for Web APIs (aka RESTful services). This way, intelligent clients can navigate among resources and decide which service to use for a given purpose [14]. Achieving and combining these two objectives is an important direction towards which the W3C WoT Interest Group[1] aims.

A promising solution consists in defining architectures able to provide clients with semantically described RESTful services (e.g. servients[2] or avatars [8]), using for instance the Hydra [6] or the Thing Description[3] vocabularies. From a pragmatic point of view, even if these architectures can theoretically be embedded in things, they can require a large amount of resources that may not be available on all devices. Indeed, there is currently no technology stack that allows at the same time discoverability, scalable access through Web standards and semantic interoperability, aimed to fit in constrained devices[4]. In order for such an architecture to meet the requirements of both the semantic WoT and constrained devices, it must satisfy the following requirements:

- Energy efficiency: Energy consumption of devices, especially battery-powered, should be optimized. In particular, during wireless network communications, the quantity of data transmitted should be minimized.
- Computational resource efficiency: Inside a server, implementing the network protocol stack requires computing resources. On constrained devices, such implementations should choose the adequate protocols to keep their memory footprint minimal.
- Self-description: Constrained devices should memorize their configuration in non-volatile memory, so that they can harmlessly be shut down and restarted, and send their API documentation to clients.
- Availability: Endpoints generated by constrained devices should be available as a Web resource and the required operations to solve requests should be kept minimal to limit processing time.
- Scalability: Web Servers embedded in constrained devices should be optimized to maximize the number of clients they can efficiently handle.

We herein propose a generic architecture designed to fit inside constrained devices, taking into account their limited resources while allowing them to provide clients with discoverable and interoperable access to their sensors and actuators. Our approach relies on two key technologies: Hydra for reconfigurable thing API documentations, and CoAP for block-wise transfers, which allow analysing and generating RDF graphs on the go. We intend to show how today's Web tools can be used to embed semantic, RESTful and auto-descriptive services on such constrained devices. Our approach targets things equipped with a network interface capable of implementing the IP protocol and minimally

[1] https://www.w3.org/WoT/IG/
[2] https://w3c.github.io/wot/architecture/wot-architecture.html\#general-description-of-wot-servient
[3] https://w3c.github.io/wot/current-practices/wot-practices.html#thing-description
[4] *Constrained devices* refer to connected things equipped with bare minimal power, energy, communication capabilities, and computing power (memory, processing, storage).

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

structured with CPU, RAM and persistent memory, plus enough program memory to store the algorithms implementing this approach. To demonstrate our design we implement our solution in a constrained device: the Arduino UNO[5].

Section 2 presents a state of the art regarding both semantics in the WoT and Technical aspects of Constrained Devices. Section 3 describes our proposition of architecture, along with the algorithm that links API semantic descriptions with the internal capabilities of a thing. Section 4 presents our implementation and evaluates our proposition against the above criteria. Section 5 concludes the paper.

2. STATE OF THE ART

2.1 Semantic Web and the IoT

Today, efforts are focused on solving heterogeneity of hardware, software, and syntactic formats in the IoT [1]. Through the IERC AC4 [10], Serrano et al. identified the problems yet to be solved in order to reach interoperability and underline the current lack of a widely used standard. Additionally, a number of initiatives tend to describe similar concepts and end up generating redundant ontologies, thus inducing the need of a commonly shared and accepted ontology [5]. This tendency is reinforced with projects such as Schema.org, a vocabulary created to promote the usage of a Web of Data which employs common semantics.

On the other hand, the Linked Data initiative[6] encourages entities to publish structured data enriched with semantics and relies on the notion of hypermedia link to map them together. Lanthaler et al. point out that unfortunately, Linked data principles have not yet found widespread adoption among Web APIs[6]. Even though semantically annotating data is a big challenge today at a global scope, the services that deliver this data need also to be given meaning. This is especially true in IoT where heterogeneous devices, communication schemes and data are encountered. Guinard et al. proposed a semantic description of services in the WoT by the means of JSON on Wireless Sensor Networks [7]. Their work stresses the use of RESTful architectures as part of the WoT. Lanthaler et al. share the same vision, in which the description of services that link the data also need to be specified with semantics. Their efforts resulted in Hydra [6], a vocabulary designed to describe Web APIs in RDF applying Linked Data principles to RESTful services.

An example of application of Hydra in the WoT can be found in [13]. Ventura et al. demonstrate how a semantic web client is able to discover self-describing APIs and to query and build request sequences with no previous knowledge of these APIs. Their implementation uses Notation3 (N3) pre and postconditions rules to describe services, and JSON-LD[7] as a vehicle to transport self-describing state transfers. Although we use this work as one of the starting points of our contribution, it is unfortunately too heavy to be embedded in constrained devices.

2.2 Constrained Devices and the Web

A Constrained Device or Node, as defined in IETF's RFC 7228 "Terminology for Constrained-Node Networks", is a device where "the characteristics that are often taken for granted for an Internet node are not attainable at the time of writing, due to constraints of cost and physical nature, such as limited battery and computing power, little memory, and insufficient wireless bandwidth and ability to communicate"[2]. Some facets often apply combinations of maximum code complexity (ROM, Flash), size of state complexity (RAM), processing power, available power, and user interface and accessibility in deployment.

Research work in the Sensor Network field faces challenges that can often be applied to constrained nodes. Wireless Sensor Networks have also implemented power saving modes of operation such as Power Aware Sensor Model, Event Generation Model, Energy workload Model among others [12]. Another Sensor Network initiative [9] reuses concepts from smart homes such as the 6LoWPAN TCP/IP stack on top of which sensor data is expressed in JSON rather than XML, since the payload size is decreased. To tackle this issue, the W3C has initiated work on Efficient XML Interchange (EXI)[8], which aims at dramatically reducing message payloads. These works stress the importance of REST architectures on top of HTTP and minimal payload size.

However, even though HTTP and TCP are widely accepted and used in today's WWW, there exist alternatives better adapted to Constrained Devices. **Constrained Application Protocol (CoAP)** [11] was conceived by the IETF CoRE task force[9]. It has been specially designed to fit Constrained Devices, by implementing a minimal header, while still enabling devices to keep track of multiple client requests, URI components, payload types, message codes, amongst other features found in traditional HTTP/TCP. As Castellani et al. mention, CoAP's similarity to the aforementioned protocols makes it easy for HTTP clients or servers to interact, since all that is needed is a translation-proxy between them [3]. Moreover, CoAP servers are able to implement REST principles, even while relying on UDP, a simpler protocol compared to TCP's and its range of technologies.

CBOR[10] stands for Concise Binary Object Representation. A JSON-compatible model expressed in binary. Its code size is greatly reduced in comparison to traditional JSON. It is an advantageous approach when dealing with tight and bad quality networks, since the data to be transmitted does not require a significant number of packets. We find it disadvantageous to use since we already process JSON. Developing a module to encode/decode CBOR is an additional unnecessary load for an already limited memory.

3. ARCHITECTURE

We herein propose the conceptual architecture of a server designed to be implemented in constrained devices, and able to communicate with its clients by exchanging RDF graphs, serialized in JSON-LD. Network communications rely on the CoAP protocol that fits both constrained devices and REST. As the server exposes a semantic and RESTful API, it documents this API according to the Hydra specification. Through this means, it can handle three types of requests:

- The server responds to *documentation requests* by sending an RDF document generated according to the device internal configuration.

[5]https://github.com/ucbl/arduinoRdfServer
[6]http://linkeddata.org/
[7]http://json-ld.org/
[8]https://www.w3.org/XML/EXI/
[9]https://datatracker.ietf.org/wg/core/documents/
[10]urlhttp://cbor.io

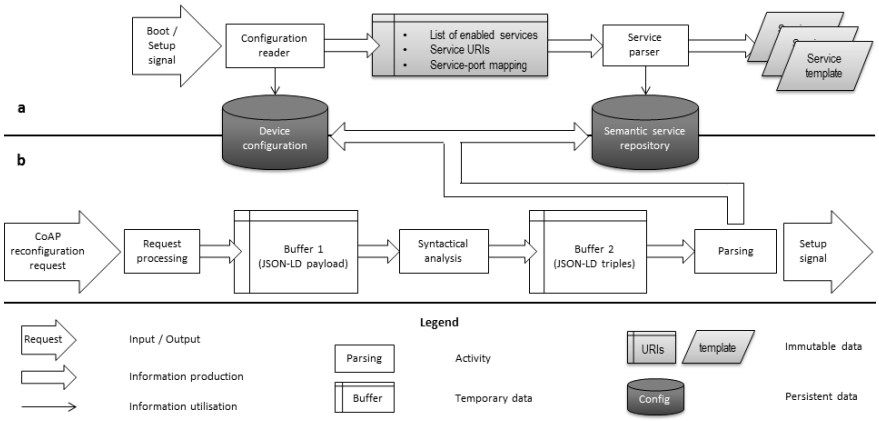

Figure 1: Startup (a) and Configuration Request (b)

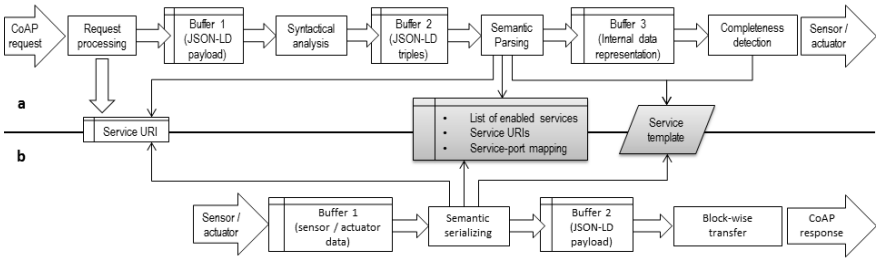

Figure 2: Request Processing Algorithm: Inbound(a), Outbound(b)

- *Configuration requests* allow clients to choose which of the available operations on the object are to be enabled, disabled or configured. This gives the user control over which services are available at any given time. With the help of non-volatile memory, the state of enabled operations can be stored, which makes such configuration resilient to shutdowns, down-time and power shortages.
- *Operational requests* allow clients to query the sensors and actuators connected to the device and enabled in its configuration, in a semantic and RESTful manner.

The different building blocks of our architecture are depicted in Figure 1 and Figure 2. The most important of them are detailed below.

- The RAM contains complete representations of the server current configuration. Clients can only request services provided by operations stored in RAM at the time of the request. RAM variables establish the link between the semantic layer and the internal functions that access the configuration and the devices. These representations are necessary for the main algorithm to execute the correct function and write a complete response, including any result obtained from a correctly requested service.
- The Device Configuration stores information about the state and configuration of the object. It may vary from object to object, but the interest is that probably not all the possible capabilities of an object need nor can be available at any given time. It stores the service configurations that were made available by users as well as the involved physical components. It also acts as a backup to restore the setup previous to a power cut, intentional or not.
- The Semantic Service Repository stores JSON-LD Semantic annotations. The algorithm refers to this memory space whenever it needs to send or interpret JSON-LD documents.

The design of the server relies on the following technical principles. Switching to CoAP from traditional HTTP gives a more in-depth packet control and allows for streaming of large payloads. Indeed, using the CoAP Block-wise Transfer option on top of UDP as a synchronous transport solution removes package size limitations. Taking into consideration that most clients on the Web do not implement CoAP, we developed a simple translation proxy from and to HTTP meant to be ran outside the constrained device.

3.1 API and Semantics

To illustrate the semantic aspects of our approach, we propose the following application scenario: a crop field is populated with numerous constrained things, disseminated over the field. Each of these fixed things can bear different sensors and actuators. Mobile clients such as drones can move over the field, discover the available fixed things and interact with them. As all these things embed their own semantic descriptions, the client can more easily integrate data and service descriptions, reason about them, and provoke an "intelligent" global behavior, such as watering the parts of the field that lack water or guide other objects to cure diseases detected on the plants. Let us consider one of these single fixed things, equiped with a temperature sensor

Figure 3: Minimal documentation exposed by a constrained thing

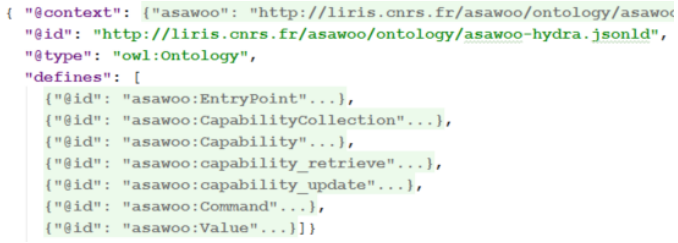

Figure 4: Ontology providing common parts of thing API descriptions

to sense weather conditions and an infrared LED that moving objects can use to proceed to the plant. It exposes a set of physical operations, aka **Capabilities** [8], and is able to serve a Hydra API documentation, as well as to repond to client requests concerning these capabilities.

By relying on Hypermedia, JSON-LD documents can reference context definitions and concepts that are served elsewhere. Using this mechanism, the constrained thing server only exposes the specific parts of its documentation and refers to a common ontology, shared with all other disseminated servers. Referencing this ontology allows to minimize the information embedded in the objects themselves, while maintaining semantic consistency. Figure 3 illustrates the minimal API description stored on the thing, and Figure 4 the global ontology[11].

At the root of the API is a Hydra *EntryPoint* that leads to a collection of the above mentioned Capabilities. A capability instance can expose one or both of the *capability_retrieve* and *capability_update* hydra operations. Our ontology relies on the SSN vocabulary[12] to define the respective outputs and inputs of these operations. In order for the things to be able to expose instances of *ssn:Output*/*ssn:Input* (which are disjoint) and *hydra:Class*, the ontology respectively provides the *asawoo:Value* and *asawoo:Command* classes. Things, however, are responsible to provide the data types they deal with. For instance, in our scenario, the thing uses SAREF[13] for describing a temperature, but another vocabulary could have been used.

The API documentation is queried and processed by clients. They can take advantage of the descriptions in this documentation according to the level of expressivity present in the descriptions and to the clients' "understanding" (i.e. reasoning) capabilities. This allows them to query the services provided in RAM.

[11]These files can be respectively found at https://github.com/ucbl/arduinoRdfServer/blob/master/example/arduino.jsonld and http://liris.cnrs.fr/asawoo/ontology/asawoo-hydra.jsonld

[12]http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[13]http://ontology.tno.nl/saref/

3.2 Startup

Whenever the device is powered up, reset, or flashed, the initial setup is immediately executed once. During this setup two classes are instantiated:

- Coap: our implementation of the CoAP protocol, which is used to handle incoming and outgoing packages. In it we find operations to manipulate payloads, as well as every function that will directly read and write a network packet.
- Resource Manager: a class created to handle representations of the semantic concepts, as well as the link between semantics and low level I/O functions, which most of the time involve Sensor/Actuator calls.

The server populates at startup a set of global variables concerning the enabled operations (*list of services, service URIs, port mappings*) from the Device Configuration, so that they are available in RAM. Although all possible operations are listed in the Semantic Service Repository, only the ones present in the configuration are instantiated as *service templates* that will be used to parse and serialize semantic information. Consequently, all necessary information is accessible to algorithm modules to perform operations. This behaviour is represented in Figure 1.a.

3.3 Thing Configuration

Enabling a service at runtime resembles a request processing, the only difference is instead of acting on a sensor/actuator, its provokes the writing of new information in the Device Configuration.

To enable/disable a service, the client must request the corresponding service URI and provide logical data about its new configuration. The semantics related to the availability of a service are defined by it's boolean *"EnabledStatus"* property. Activation of a service is shown in Figure 1.b.

3.4 Request Processing

When a CoAP packet is received, relevant data relative to the currently requested service is saved, meanwhile the algorithm streams the incoming packages using block-wise,

and recycling the inbound buffers in an effort to maximize available space in memory.

Once all relevant information for a service is retrieved from the request and completeness is detected, the corresponding Sensor/Actuator function for that specific service is triggered. The return values are stored in a buffer. The server then sends the corresponding response using (once again) block-wise transfer. When writing into the outbound buffer, the JSON-LD payload is generated using the Service Template, and is complemented by the results retrieved from the operations on sensors and actuators, thus forming a complete response. The whole process is represented in Figure 2.a and Figure 2.b.

4. EVALUATION

We previously defined a set of criteria to enable constrained devices to integrate the semantic Web of Things: Discoverability, Self-description, Scalability, and Power and Computational Resource efficiency. As a proof of concept, we evaluate these criteria on an Arduino UNO implementation.

4.1 Implementation

The Arduino UNO and similar devices are microcontrollers conceived to host flashed C++ compiled code, also called *sketch*. We make use of libraries that let us interact with the physical/electronic modules as well as UDP packages. The Arduino UNO's memory specification is detailed in table 1. Arduino boards provide access to I/O ports, or *pins*. Either analog or digital, they connect the board to sensors and actuators through libraries provided by the SDK.

Program Space	RAM	Non-volatile (EEPROM)
32KB read-only	2KB read/write	1KB read/write

Table 1: Arduino UNO Memory Specification

Open source CoAP implementations are, to the best of our knowledge, not yet ported to Arduino UNO boards, which is why we developed our own. Due to its size, the Semantic Server Repository SSR needs to be stored in the device's Program Space. The text based descriptions take several KBytes, and storing them as standard strings at runtime would easily overflow the Arduino's RAM. AVR libraries available on Arduino boards provide a set of tools to manipulate Program Space pointers, which help overcome this issue. We use the non-volatile EEPROM memory to store the Device Configuration. By serializing the information about the ID and pins for a service, we effectively describe the physical configuration, as well as providing the necessary information to later fetch a service in the SSR, resulting in a list of enabled services in RAM. We developed a proxy translating headers from HTTP into CoAP and vice versa. It was developed with the TxThings[14] library and the Twisted Framework[15] in Python2. The implementation is represented in Figure 5.

4.2 Experimentation

Arduino devices provide a simple logging tool through a serial connection that allows us to print text during execution. Thanks to it and the observable behaviour of the board

[14]TxThings: https://github.com/mwasilak/txThings
[15]Twisted Framework: http://twistedmatrix.com/trac/

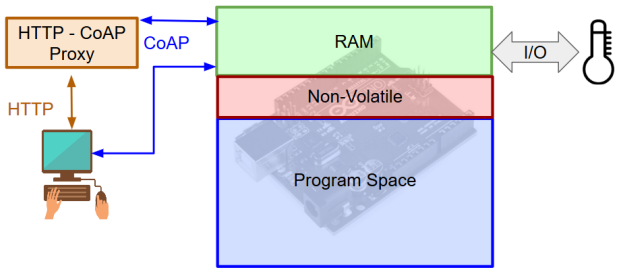

Figure 5: Implementation on an Arduino UNO through a translation Proxy

we are able to have some degree of debugging. We test the 3 primary capabilities of our architecture:

- Service persistence/startup: Verifying the configuration is correctly saved. The same service should respond before and after removing the power source, effectively shutting down the device. To do so, we simply power off the device after using a service. We succeed if the service yields the same previous behaviour.
- Enabling a new service: Making a new service available by modifying its *Enabled* property and specifying its configuration. This is done by performing a POST request on */service_name/enable*, including information about the physical configuration in the payload. If successful, the device responds with a *2.04 Changed* CoAP message. We then request the service itself and verify the output matches our expectations.

4.3 Results

	Footprint	**Cumulative Available RAM (Bytes)**
Algorithm & Class Instantiation	968Bytes	1080 (52.7%)
Capability (x3)	279Bytes	725 (35.4%)
Static JSON Buffer	200Bytes	525 (25.63%)
Package-processing	24Bytes	501 (24.46%)

Table 2: RAM Usage

The CoAP protocol offers the option for any party to set the payload size. We take advantage of this fact to ensure CoAP packets never encapsulate more than 64Bytes, thus transmitting 130Bytes at most between the proxy and the Arduino. Furthermore, taking into account the synchronous nature of the protocol, the bandwidth consumed remains under the maximum mentioned previously. We recall and evaluate the criteria we defined to embed a semantic server into a constrained device:

- Since most clients on the Web rather implement HTTP than CoAP, our HTTP translation Proxy compensates the discoverability drawbacks. Yet, it is not necessary to go through this proxy, which allows other CoAP clients to communicate directly, effectively widening the spectrum of possible requesting parties.

- Services are serialized in JSON-LD and stored in Program Memory. Referencing further definitions on an external server allows us to save up to 31.2KBytes of data, close to 100% of total Program Memory. The remaining local semantics allows the API documentation to be self-descriptive, allowing semantic clients to conceive requests at runtime.

- Power efficiency is achieved thanks to the persistence of configuration and services; we can imagine scenarios where current is cut on an object by an external factor or in periodic cycles to maximise its lifetime.

- Computational Resource Efficiency is achieved through the properties a service needs to be available; non-configured services stay in the SSR, which means they have no representation in RAM nor the configuration, effectively reducing memory load. RAM usage can be visualised in table 2. With a cost of 93Bytes, 10 Capabilities can be instantiated simultaneously.

- Our architecture is able to apply some REST principles that facilitate scalability. Client-state representation is not kept in memory, and there is no out of the band knowledge necessary to retrieve URIs.

5. CONCLUSION

A big number of things are flourishing with the potential to integrate the Web in the coming years. As of today, no clear standard technology stack is ready to integrate them to the Web due to their heterogeneity. This is especially true for constrained devices, that can not rely on extensive resources to support typical Web technologies. Furthermore the price on hardware for Web-capable devices represents a major obstacle for the widespread adoption of these technologies. Embedding them in ever smaller devices minimizes the influence of this factor. The technical feasibility depended on the ability of the chosen tools and standards to fit in the constrained setup of the device. We can imagine better devices to embed the proposed architecture, but we intentionally chose an Arduino UNO to demonstrate that it is capable to cope with our expectations.

We herein proposed a minimal architecture to enhance constrained devices with semantics and improve their interoperability with Web standards. We contribute to a more distributed semantic Web, encouraging the creation of intelligent clients. Our contribution meet a certain number of criteria: discoverability through the use of commonly used technologies, power efficiency by implementing persistent configuration, computational resource efficiency by allowing only configured services to be available, self-description through semantic descriptions of APIs, and scalability by implementing REST principles.

We envisage further work towards implementing a RESTful agent instead of just servers, to learn and generate requests to other connected objects on the Web and then participate in physical mashups, as well as enabling the proxy to serve as a cache and content negotiator.

Acknowledgement

This work is supported by the French Agence Nationale pour la Recherche (ANR) under the grant <ANR-13-INFR-012>.

6. REFERENCES

[1] P. Barnaghi, W. Wang, C. Henson, and K. Taylor. Semantics for the internet of things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 8(1):1–21, 2012.

[2] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. Technical report, 2014.

[3] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi. Web services for the internet of things through coap and exi. In *2011 IEEE International Conference on Communications Workshops (ICC)*, pages 1–6. IEEE, 2011.

[4] D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards physical mashups in the web of things. In *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*, pages 1–4. IEEE, 2009.

[5] A. Gyrard, M. Serrano, and G. A. Atemezing. Semantic web methodologies, best practices and ontology engineering applied to internet of things. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 412–417. IEEE, 2015.

[6] M. Lanthaler and C. Gütl. Hydra: A vocabulary for hypermedia-driven web apis. *LDOW*, 996, 2013.

[7] S. Mayer, D. Guinard, and V. Trifa. Facilitating the integration and interaction of real-world services for the web of things. *Proceedings of Urban Internet of Things–Towards Programmable Real-time Cities (UrbanIOT)*, 2010.

[8] M. Mrissa, L. Médini, J.-P. Jamont, N. Le Sommer, and J. Laplace. An avatar architecture for the web of things. *IEEE Internet Computing*, 19(2):30–38, 2015.

[9] L. Schor, P. Sommer, and R. Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 31–36. ACM, 2009.

[10] M. Serrano, P. Barnaghi, and P. Cousin. Semantic interoperability: Research challenges, best practices, solutions and next steps, ierc ac4 manifesto. 2014.

[11] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). Technical report, 2014.

[12] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design & Test of Computers*, 18(2):62–74, 2001.

[13] D. Ventura, R. Verborgh, V. Catania, and E. Mannens. Autonomous composition and execution of rest apis for smart sensors. In *Joint Proceedings of the 1st Joint International Workshop on Semantic Sensor Networks and Terra Cognita and the 4th International Workshop on Ordering and Reasoning. URL http://ceur-ws. org*, volume 1488.

[14] R. Verborgh, E. Mannnens, and R. Van de Walle. Bottom-up web apis with self-descriptive responses. In *Proceedings of the First Karlsruhe Service Summit Workshop-Advances in Service Research, Karlsruhe, Germany, February 2015*, volume 7692, page 143. KIT Scientific Publishing, 2015.