Functionality and object capability ontology

Deliverable 5.1 (version 1)

Université Lyon 1 - LIRIS
Université de Grenoble Alpes - LCIS

06 March 2015

Project ASAWoO
Adaptive Supervision of Avatar / Object Links for the Web of Objects
Grant Agreement: ANR-13-INFR-0012-04

Abstract

As the number of connected objects increases, there is a need to offer rich user experience and facilitate communication between physical objects with Web-based solutions. Our work relies on the notion of avatar to extend an object on the Web. We herein propose a model for the avatar to expose functionalities based on the capabilities objects offer. We motivate our work with a temperature regulation scenario and we evaluate the applicability of our proposal with an implementation.

Contents

1 Introduction

The Web of Things is envisioned as an evolution of the Internet of Things, where connected objects leverage Web-based languages and protocols to reach interoperability and offer users advanced interaction possibilities. As the number of connected objects is rapidly increasing, there is an urgent need to propose solutions that exploit the possibilities the Web offers to facilitate communication between physical objects and make rich user interaction a reality. Moreover, in a typical situation, objects are currently not able to communicate with one another without a complex preliminary setup process.
We herein propose to ease user-object, as well as object-object interaction by exposing semantic, high-level behaviors (aka functionalities) on the Web, as RESTful services. Our work relies on the notion of avatar, designed to extend an object on the Web. An avatar not only provides an entry point to the object it extends, but it provides additional intelligence to the object via internal functioning, as well as its interaction with other avatars and external resources.

1.1 Motivating Scenario

In the following we motivate our work with a simple temperature regulation scenario, freely adapted from [8]. The scenario helps define the requirements that guide the development of our proposal. Carrying his mobile phone in his pocket, a user called Bob enters a room that contains several connected objects: a temperature sensor, a temperature increaser (e.g. heater), a temperature decreaser (air conditioner), and a programmatically controllable motor that can open and close the room window. Bob's preferred temperature is 21°C and the room's current temperature is 24°C. The external temperature is unknown. The goal of the scenario is to reach Bob's desired temperature without human intervention. In the following, we show how our WoT infrastructure based on avatars and their reasoning features contribute to reach this goal.
First, whenever an object connects to the local network, the WoT infrastructure of the room creates its avatar and connects it to the object. Before Bob's arrival, the temperature sensor, the temperature increaser, the temperature decreaser and the controllable window are each connected to their own avatars. These avatars form a community in which they expose the functionalities their objects offer: the sensor exposes that it can measure the room temperature, the increaser (resp. decreaser) that it can heat (resp. cool) the room and the motor that it can modify the room temperature to reach the external temperature.
Second, avatars proactively search for exposable collaborative functionalities: each avatar embeds an inference engine that processes a functionality ontology. This engine discovers that a functionality called "regulate the room temperature" can be achieved as a combination of other functionalities exposed by the avatars of objects in the room: measure the temperature, heat and cool the room. It also discovers that cooling the room can be achieved either by the air conditioner or by the window motor, if the external temperature is lower than the room temperature. As the priority of decreasing the temperature using the window motor is higher than that of doing it with the air conditioner (obviously for energy saving reasons), it chooses the second solution. As several avatars may have found that their objects are involved in the temperature regulation functionality, they start a negotiation process to find out which one will expose this collaborative functionality. Their contextual adaptation modules find out that the functionality requires numerous calls to the temperature sensor (see below), so they decide to host this functionality on the avatar of the sensor.
When Bob enters the room, his cell phone also gets an avatar. But as the phone has already been connected to the WoT infrastructure before, this infrastructure recognizes the object: the cell phone avatar is not a new instance, but a deserialization of its last avatar instance. This avatar then reminds that Bob once informed his cell phone of his preferred temperature. Once in reach of the room WoT infrastructure, Bob's cell phone avatar can discover and exploit available functionalities to reach Bob's goal as follows:

1. The avatar of Bob's cell phone broadcasts the network upon connection to discover available functionalities. It receives a response from all other object avatars in the room, containing a list of their exposed functionalities.

2. It invokes the service exposed by the avatar of the temperature sensor (24°C).

3. It detects that Bob's need is not fulfilled and initiates a new goal, which is to decrease the temperature to 21°C. It then computes, with the help of its internal reasoner, that this goal can be achieved by a functionality called "regulate the room temperature".
4. As this functionality is exposed by the avatar of the sensor, it invokes it with a targeted temperature of 21°C.
5. The avatar of the sensor invokes the service exposed by the avatar of the temperature increaser to turn off the heater and invokes an external weather forecast Web service, that returns that the external temperature is 19°C and that it is a sunny day.
6. The avatar of the sensor then invokes the avatar of the window motor to get the window position (open) and leaves it open to let the temperature decrease.
7. It regularly queries the temperature sensor, and when the temperature reaches 21°C, it closes the window with an invocation to the service the avatar of the controllable window exposes.
8. Bob found the room too warm and left during the regulation process. But this did not interrupt the process, since it is managed between the avatars of objects that are still in the room and can communicate together. His cell phone regularly queries the avatar of the sensor to get to the current temperature and inform Bob on its screen when the room has reached the desired temperature.

This simple scenario shows how an avatar-based architecture enables the proactive emergence of collaborative behavior to fulfill users' needs. However, the realization of such an architecture raises numerous scientific locks, such as how to build the avatar community, how to handle inter-avatar communication and deal with communication faults, how to manage avatars during their whole life cycle and how to cope with resource-constrained objects and network problems (bandwidth limitations and frequent disconnections). For the sake of brevity, in this paper we summarize our avatar-based architecture and do not give details about the different aspects involved. The architecture is being developed in the context of the ASAWoO project, we refer the reader to the project homepage[1] for more details.

1.2 Contribution and Paper Organization

This paper specifically focuses on the following scientific lock: how to discover, compose and transform low-level capabilities as well as high-level functionalities in order to answer users' needs? There is a need to develop the required ontologies to enable reasoners to infer the available functionalities from a set of low-level capabilities provided by the physical objects. Reasoners must also be able to identify complex functionalities that are partially achievable locally (on one avatar-object couple) to send a request to the community of avatars and ask for help on missing functionalities, enabling collaborative behavior.
The paper is organized as follows: Section 2 summarizes the avatar architecture. It provides the required information to understand how avatars function and communicate with each other. Sections 3 presents our model to reason about capabilities and infer exposable high level functionalities from a set of available capabilities. Section 4 gives details about how avatars query the model and shows its advantages. Section 5 demonstrates the applicability of our proposal with the implementation of our scenario. It shows how the reasoning process integrates into our architecture and demonstrates how it is adaptable to other scenarios. Section 6 reviews related work and highlights the originality of our contribution. Section 7 discusses the possible strategies for deploying WoT applications on top of our model and Section 8 gives guidelines for future work.

2 Avatar Architecture

The avatar architecture is called the "WoT Runtime Environment", as depicted in Fig. 1. It is composed of a framework in which are plugged a set of components called managers, which are grouped in modules. Each manager takes charge of a specific concern and interacts with other components using a specific API.

[1] http://liris.cnrs.fr/asawoo/

The core module of the avatar architecture deploys the components into the framework via the “Component Deployment Manager" and furnishes low-level components that other managers require to perform caching and reasoning tasks. Each logical component of the avatar architecture can be executed either on the object or in the cloud. Avatars belong to a WoT infrastructure that is restricted to the local network for privacy reasons. However, the infrastructure enables inter-avatar communication as well as limited interactions with the external Web[2].

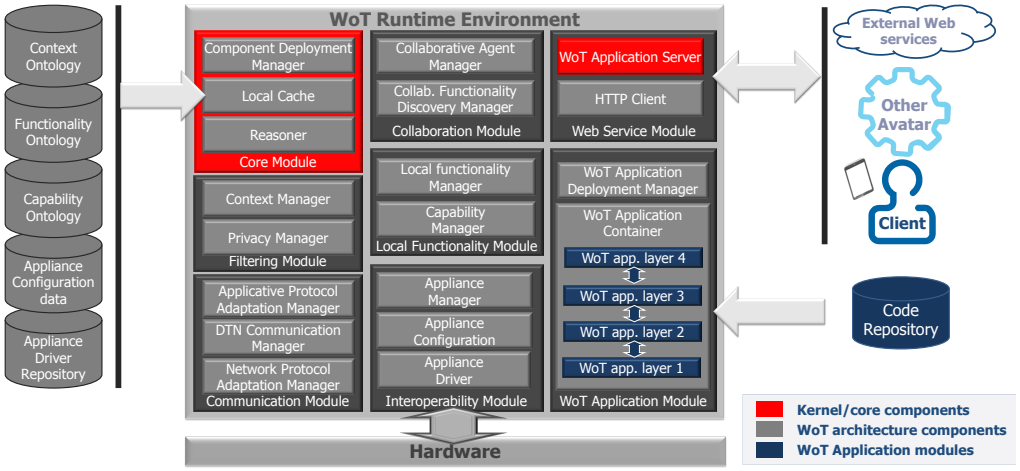

Figure 1: Avatar Architecture

At initialization time, the “Capability Manager" of the “Local Functionality Module" queries the “Appliance Manager" to retrieve the basic capabilities that the object can perform. Then, the “Local Functionality Manager" semantically discovers the functionalities that the object can achieve using these capabilities. To do so, it queries the internal reasoner of the avatar that reasons about the functionality ontology. Some functionalities are complex and involve several functionalities to be fully realized. In such case, there is a need for a specific language to describe the orchestration of functionalities. In this paper, we use the SCXML language, as proposed in [12], however, it is possible to rely on a different language to describe complex functionalities depending on their needs. The discovered functionalities are filtered using the “Context Manager" and “Privacy Manager", in order not to expose unsafe or unrealizable functionalities.
The filtered functionalities are exposed as services to users and other avatars by the WoT application server. When a service is invoked, the local orchestration engine embedded in the avatar triggers the invocation of services based on the description of the functionality orchestration. For each “atomic" functionality (that is realized by a capability), the actual implementation of the capability is specific to each object and stored in an application code repository. The application code that implements the functionalities is executed in a second internal framework, called the “WoT Application Container”. The code is dynamically deployed from the code repository into the framework at execution time.
In the same fashion, the “Collaborative Functionality Discovery Manager" semantically discovers the functionalities in which the object can be involved but require collaboration with other objects. It is queried by the “Collaborative Agent Manager", that interacts with other avatars through the Web service module. The “WoT Application Server" component exposes available functionalities as REST resources. Each avatar thus uses the HTTP client module to lookup or invoke avatar resources in the WoT infrastructure, or query external Web services. The avatar also uses the Interoperability and Communication modules to respectively connect to the object and communicate with it using the most optimal network protocols. In the following, we describe our model and the reasoning mechanisms that the avatar utilize to expose local and collaborative functionalities.

[2]For the sake of brevity, the operation of the WoT infrastructure as well as the code deployment strategy are not detailed here.

3 Functionality and Capability Model

Our model relies on a few main classes that facilitate the understanding of the different concepts put into play in the Web of Things. It enables describing appliances and the capabilities they have, and makes it possible to establish a link between a set of capabilities that objects provide and a set of functionalities that WoT applications need and that avatars can in turn expose as Web resources. The classes in our ontology are defined according to this model, as follows:

- The **Appliance** class denotes physical connected objects. It is related to the Capability class through the *hasCapability* object property that denotes the possibility for an object to realize an action. An object may be related to several capabilities to describe its capability to realize several actions.
- The **Capability** class denotes the physical actions that objects can realize. Capability instances provide an implementation for a functionality. We defined three subclasses for the Capability class : ActuatorCapability, ProcessingCapability and SensorCapability, that respectively denote physical actions, data processing, and detection.
- The **Functionality** class denotes high level functionalities that avatars can expose on the Web and can be viewed as the interfaces that capabilities implement. Functionalities are utilized in WoT applications. There are three ways for implementing a functionality: directly with a capability exposed by an object or a service endpoint, or indirectly through a combination of several functionalities that are themselves implemented. The functionality class has a *isComposedOf* property that allows for describing complex functionalities that are composed of other functionalities. It also has an *isImplementedBy* property that describes how a functionality is realized with a capability.

More formally, we define the model that describes the different concepts of our ontology as follows:

- Let a functionality f be an individual in the set of functionalities F described in the ontology, noted $f \in F$.
- Let a capability c be an individual in a set of capabilities C described in the ontology, noted $c \in C$.
- The relation $isComposedOf(f, f')$ denotes the fact that a functionality f requires another functionality f' to take part in a composition to be accomplished.
- The relation $isImplementedBy(f, c)$ denotes the fact that a functionality is implemented by a capability.

The set $F'(f)$ of functionalities that compose f is defined from the $isComposedOf$ property as follows:

Definition 1 (Composition)**.** $\forall f \in F, F'(f) = \{f' \in F \mid isComposedOf(f, f')\}$.

A functionality $f \in F$, with $F'(f)$ the set of its composed functionalities, is *exposable* if it is directly implemented by a capability or if all the members of $F'(f)$ are (directly or indirectly) exposable. Then, we define that $f \in EF$, where EF is the set of exposable functionalities, as follows:

Definition 2 (Exposable functionality)**.** $f \in EF \iff \exists c \in C \mid isImplementedBy(f, c) \vee \forall f' \in F'(f), f' \in EF$

According to these definitions, the model offers the means to explore the properties in the ontology to know how the functionalities are composed, what functionality is exposable, and how it is implemented. It offers the underlying support that contributes to build the different queries to be presented in the following.

4 Querying the Ontology

Each functionality is described in terms of composing functionalities and implementing capabilities in a common shared ontology. We rely on a reasoner that processes this ontology and responds to SPARQL queries as shown in Fig. 2. We developed a set of Java functions to build the appropriate SPARQL queries and get the required information about functionalities and capabilities:

- The *getFunctionalities(List<Capability> l)* function builds a SPARQL query that selects all the functionalities in which at least one of the capabilities passed as parameter is present.
- The *getIncompleteFunctionalities(List<Capability> l)* function builds a SPARQL query that selects all the functionalities in which at least one of the capabilities passed as parameter is present, and in which a capability is required that is not one of the passed parameters.
- The *getInvolvedCapabilities(List<Functionality> f)* function returns the list of capabilities involved in a set of functionalities.
- The *getComposingFunctionalities(Functionality f)* function returns a list of functionalities that compose a given complex functionality.

During the avatar initialization, the capability manager queries the appliance manager that retrieves the list of capabilities that the object can fulfill. The queries are then utilized as follows:

1. The avatar retrieves the functionalities that can be fulfilled using these capabilities, using the *getFunctionalities* function. These functionalities are filtered to determine those that are actually realizable and the realizable ones are exposed by the application server, as explained in Section 2.
2. The avatar retrieves the functionalities in which these capabilities are involved, but for which other functionalities/capabilities are required, using the *getIncompleteFunctionalities* function. The list of incomplete functionalities is then used for building collaborative functionalities that imply other objects. For this, the avatar uses the *getInvolvedCapabilities* and *getComposingFunctionalities* functions to determine the missing functionalities to realize each collaborative functionality. Then it starts to negotiate with other avatars to find out if this functionality is collaboratively realizable and which avatar should expose it.

Each avatar generates and deploys WoT applications depending on reachable objects over the network. Different deployment strategies are possible, such as event-based updates upon the arrival and removal of objects in the network, as well as runtime handling of user requests. Such strategies are discussed in Section 7.

```
SELECT ?capability WHERE { PropertyValue(<http://localhost/Thermometer>, <http://localhost/isImplementedBy>, ?capability) }
SELECT ?functionality WHERE { PropertyValue(<http://localhost/Thermometer>, <http://localhost/isComposedOf>, ?functionality) }
SELECT ?capability WHERE { PropertyValue(<http://localhost/TemperatureRegulation>, <http://localhost/isImplementedBy>, ?capability) }
SELECT ?functionality WHERE { PropertyValue(<http://localhost/TemperatureRegulation>, <http://localhost/isComposedOf>, ?functionality) }
```

Figure 2: A few SPARQL query examples

5 Implementation and Evaluation

We relied on the Derivo SPARQL-DL API[3], together with the OWL API[4] and the HermiT reasoner[5] to describe, access and reason about functionalities and capabilities. The HermiT reasoner classifies data at design time based on the hypertableau algorithm. Then, once all the facts have been generated, they are stored in a database that is made available through a SPARQL endpoint via the QueryEngine object of the Derivo SPARQL-DL API. We implemented our solution in a Java™ 1.7 environment with the Eclipse IDE. We evaluated our proposal in terms of response time.
Our ontology[6] describes both the classes and properties of our model together with the different instances (individuals) that take part in our temperature regulation scenario. The ontology counts 52 axioms and our algorithm finds all the available composed functionalities with an average response time over of 45 milliseconds with a very low standard deviation (< 1 millisecond), which indicates that our model and algorithms offer acceptable response time.

[3] http://www.derivo.de/en/resources/sparql-dl-api.html
[4] http://owlapi.sourceforge.net/
[5] http://www.hermit-reasoner.com/
[6] Available at http://soc.univ-lyon1.fr

While the discovery of object functionalities and capabilities is performed with the help of the ontology, the actual implementation of a capability is specific to each object and stored in an application code repository as shown in Fig. 1. In the same manner, the orchestration of a functionality into one or several capabilities is described by an orchestration language. In our work, we relied on finite state machine, using the SCXML language, as proposed in [12], but other languages can be adapted in our architecture thanks to its design into modules. It is simple to substitute a module for another as long as the offered API is respected.
Our solution applies to a temperature regulation scenario, however, it is important to highlight that it remains completely independent from any scenario and application domain. What is required to adapt our solution to a particular application domain is to design the functionality and capability descriptions according to the targeted domain and to provide the required properties ("isComposedOf" and "isImplementedBy"). Once this work has been performed, our solution is able to extract the required information from the ontology and compute the set of exposable functionalities.

6 Related Work

Taking advantage of the most recent advances in the semantic Web for the benefit of the Web of Things is a hot research topic, as prove the increasing number of publications in the field [6, 11, 4].
The problem of semantic interoperability is tackled in [6], where the authors introduce the concept of semantic smart gateway. A semantic smart gateway acts as a hub that provides the tools to perform ontology alignment, registration of smart objects on the network, on-the-fly ontology learning to support semi-automated description of smart objects. The work relies on existing approaches in the domain of ontology alignment to facilitate interoperability between smart objects. While this work is addressing an important problem, it does not provide specific support for exposing object functionalities.
In [11], the authors present a framework for the semantic Web of Things. They give general information about the operation of the framework, in particular a focus is provided on the unique identification of devices. The paper summarizes different results on the annotation of existing standard protocols with semantic information to enable semantic discovery of device resources.
In [4], the author proposes a distributed platform to exploit heterogeneous sensor measurements. The focus of this work is on the integration of different ontologies and protocols based on semantic reasoning to align heterogeneous ontologies. The work addresses the data and protocol heterogeneity problem mainly focuses on information coming from sensors, it does not take into account the composition and orchestration of different objects that performs actions.
In [8, 7], the authors provide Web-based solutions to facilitate semantic discovery as well as in smart thing infrastructures. They rely on a reasoning mechanism that accepts different types of queries. The reasoner produces a proof that enables the coordination of devices through a set of partially ordered HTTP requests. The work requires objects to be exposed as RESTful resources and annotated with semantic descriptions.
In [2], the authors present a work-in-progress concerning an ontological framework to retrieve connected things in the WoT. To achieve their goal, the authors create and use semantic profiles of connected objects. Similarities between the semantic profiles are gathered into clusters. Then a score is associated to a 'context of search' from an incoming request which enables the selection of the most appropriate search algorithms.
In [9], the authors propose a mechanism of human-assisted simplification of semantic matching to enable interoperability between entities in the Internet of Things. It concretely consists in facilitating ontology alignment through visually-enriched ontology and thing descriptions.
In addition to integrating things in the Web, [10] provide abstractions for things, basic services for search and annotation, so that the things participate in the Linked Open Data[7] (LOD) cloud.
As well, a lightweight matchmaking engine based on a reasoner that supports Semantic Web technologies through the OWL API and implements both standard reasoning tasks for knowledge base management (subsumption, classification, satisfiability) and non-standard inference services for semantic-based resource discovery and ranking (abduction, contraction and covering) is presented in [3]. The main target is the Android platform (first experiments were made on laptop and smartphones).

[7] Linked Data is a method for publishing structured data on the Web [1]

Some projects like HYDRA[8], SemSorGrid4Env[9], SENSEI[10], SOfiA[11], and SM4All[12] propose middleware to support IoT application. In [13] a service-oriented middleware is proposed. Its architecture is based upon a layer that describes thing-based services. The core of this service description layer enables to describe things according to [5] with:

- A device ontology to identify the things which can be utilized to satisfy an application requirements,
- A physics domain ontology to model real world entities as physical concepts and to model associated mathematical functions,
- An estimation ontology to describe models which can be used when a service/data is unavailable.

This last ontology is an originality in comparison with previously mentioned projects.
As presented above, existing work mainly focus on specific parts of our work: integration of heterogeneous data from sensors, exposition of devices as resources, reasoning about semantic description for the IoT, design of frameworks for device interoperation, but to the best of our knowledge none of them provides an architecture that puts together all these advances to interconnect and control different devices, not only sensors but also actuators, and connects to reasoners to provide users with complex functionalities via WoT applications. In our work, we rely on the notion of avatar to build our framework, and in this paper, we present how avatars exploit our model and reason about functionalities and capabilities to propose WoT applications based on available devices.

7 Discussion

Our work allows to create and expose WoT applications from a set of objects in a common environment. Our model is aimed at being as flexible as possible with respect to the choice of an implementation for a functionality. To reflect that flexibility, we designed functionalities as interfaces. These interfaces can be implemented either using generic workflows described as state charts (in SCXML), or using capabilities (i.e. system code furnished by the object vendors). Then, WoT applications remain implementation-independent and can be deployed on different objects, provided that they can locally implement the requested functionalities. We envision different strategies to manage and deploy WoT applications based on the exposition of the functionalities that take part in the application.
One possibility is to allow late binding a functionality to different implementations: at runtime the task of the reasoner consists in finding the workflows and/or capabilities that can be combined to implement the required functionality. Our reasoner can perform this task with the *getComposingFunctionalities* and *getInvolvedCapabilities* functions. Such a setup allows to substitute an object capability with a functionality exposed by another object at runtime. However, it has the drawback to offer the possibility to fail if one of the functionalities is disabled.
Another possibility is to pre-calculate a functionality implementation and to update it using event-triggered actions, depending on context changes. Such a possibility is less flexible in terms of how to implement the workflow because the implementations are statically bound into the application, but it offers the advantage that exposed functionalities are always consistent with exposed ones.

8 Conclusion

In this paper, we presented a model that supports semantic discovery and invocation of functionalities for the Web of Things. Based on our avatar architecture, we show how our model describes the capabilities objects offer and enables reasoning about these capabilities to expose high level functionalities into WoT applications. We present a formal definition of our model, together with an implementation that show its

[8] http://www.hydramiddleware.eu
[9] http://www.semsorgrid4env.eu
[10] http://www.sensei-project.eu
[11] http://www.sofia-project.eu
[12] http://www.sm4all-project.eu

applicability in the context of our motivating scenario. We discuss the different possibilities our model offers for the management of WoT applications.
Future work includes exploring how our model can be extended to describe different alternatives for a composed functionality. There is also a need to extend our algorithms to select the best available alternative based on additional quality of service information.

Acknowledgement

We would like to thank Guillaume Charmettant and Agathe Grünberg for their participation in the preliminary work and implementation of the prototype. This work is supported by the French ANR (Agence Nationale de la Recherche) under the grant number <ANR-13-INFR-012>.

References

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[2] Benoit Christophe, Vincent Verdot, and Vincent Toubiana. Toward an adaptive semantic search mechanism for the 'web of things'. *Int. J. Semantic Computing*, 5(4):337–361, 2011.

[3] Filippo Gramegna, Saverio Ieva, Giuseppe Loseto, Michele Ruta, Floriano Scioscia, and Eugenio Di Sciascio. A lightweight matchmaking engine for the semantic web of things. In *21th Italian Symposium on Advanced Database Systems (SEBD 2013)*, pages 103–114, jun 2013.

[4] Amelie Gyrard. An architecture to aggregate heterogeneous and semantic sensed data. In Philipp Cimiano, Óscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *ESWC*, volume 7882 of *Lecture Notes in Computer Science*, pages 697–701. Springer, 2013.

[5] Sara Hachem, Thiago Teixeira, and Valérie Issarny. Ontologies for the internet of things. In *Proceedings of the 8th Middleware Doctoral Symposium*, MDS '11, pages 3:1–3:6, New York, NY, USA, 2011. ACM.

[6] Konstantinos Kotis and Artem Katasonov. Semantic interoperability on the web of things: The semantic smart gateway framework. In Leonard Barolli, Fatos Xhafa, Salvatore Vitabile, and Minoru Uehara, editors, *CISIS*, pages 630–635. IEEE, 2012.

[7] Simon Mayer and Gianin Basler. Semantic metadata to support device interaction in smart environments. In Friedemann Mattern, Silvia Santini, John F. Canny, Marc Langheinrich, and Jun Rekimoto, editors, *UbiComp (Adjunct Publication)*, pages 1505–1514. ACM, 2013.

[8] Simon Mayer, Dominique Guinard, and Vlad Trifa. Searching in a Web-based infrastructure for smart things. In *IOT*, pages 119–126. IEEE, 2012.

[9] Olena Kaikova Oleksiy Khriyenko, Vagan Terziyan. User-assisted semantic interoperability in internet of things. In *The Sixth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'12)*, pages 104–110, 2012.

[10] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, Marcel Karnstedt, Myriam Leggieri, Alexandre Passant, and Ray Richardson. Spitfire: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, 2011.

[11] Michele Ruta, Floriano Scioscia, and Eugenio Di Sciascio. Enabling the semantic web of things: Framework and architecture. In *ICSC*, pages 345–347. IEEE Computer Society, 2012.

[12] Álvaro Sigüenza, José Luis Blanco Murillo, Jesús Bernat, and Luis A. Hernández Gómez. Using SCXML for Semantic Sensor Networks. In Kerry Taylor, Arun Ayyagari, and David De Roure, editors, *SSN*, volume 668 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[13] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: A perspective - (invited paper). In *Towards a Service-Based Internet - 4th European Conference, ServiceWave 2011*, volume 6994 of *Lecture Notes in Computer Science*, pages 220–229. Springer, 2011.