future internet

Article

Test Bed of Semantic Interaction of Smart Objects in the Web of Things

Santiago Guerrero-Narváez [1,*], Miguel-Ángel Niño-Zambrano [2], Dalila-Jhoana Riobamba-Calvache [1] and Gustavo-Adolfo Ramírez-González [1]

1 Department of Telematics, University of Cauca, Cl. 5 #4-70 Popayán, 190002 Cauca, Colombia; dalilar@unicauca.edu.co (D.-J.R.-C.); gramirez@unicauca.edu.co (G.-A.R.-G.)
2 Department of Systems, University of Cauca, Cl. 5 #4-70 Popayán, 190002 Cauca, Colombia; manzamb@unicauca.edu.co
* Correspondence: santoguerrero@unicauca.edu.co

Received: 15 March 2018; Accepted: 25 April 2018; Published: 9 May 2018

Abstract: Semantic interaction in the Internet of Things (IoT) is an important concept within current IoT development, given that smart things require further autonomy with greater processing, storage, and communication capacities. The problem is now becoming one of how to get these things to interact and collaborate with each other; to form intelligent environments amongst themselves and thus generate better services for users. This article explores a solution approach that consists in providing collaborative behavior to smart things, through the incorporation of an ontology and an architecture. It makes possible things that can communicate and collaborate with each other, allowing the generation of new services of interaction according to user needs. For this task, a real test bed of smart things was created, in which the proposed solution was deployed (Smart Room). Finally, it was concluded that the creation of these types of test bed is feasible, taking into account that response times and the information delivered by the different managed processes are acceptable. New challenges were encountered, however, such as problems of critical region in test beds with conflicting services and management of multiple users.

Keywords: semantic interaction; smart things; semantic ontology; test bed; internet of things and web of things

1. Introduction

People tend to be defined by the way they relate to the physical world. They are described by the things they own or by those things they relate to (e.g., Red Car, the Microsoft owner, Mary's husband). Things of the real world are leaving behind their inert role to become a fundamental part of the people and they organize their quality of life (an example of this is the smartphone). This trend has come to be called the Internet of Things (IoT).

By connecting things (real, physical or virtual) to the Web, digital representations can then be created of them to interact with their owners and monitor the states of real-world entities. This is known as the Web of Things (WoT). A state should be understood as the set of values that one or more properties observes at a given time, related to the entities of interest [1] of the user that are measured through sensors. For example, a webcam located at home can be monitored by the owner through the smartphone from the office and verify its state in relation to the entities of interest: e.g., the children, the family pet, etc.

Things today maintain an inherent characteristic, which is their owner. However, each Thing has an objective of interaction with its environment. Around it there are other things that are part of that environment and that may be related to the context of application of their interactions.

The interaction of things is understood as the ability to collect raw data through sensors and intervene in their environment by actuators. Unfortunately, the development of the IoT, as with many internet technologies, began in a disorganized manner, generating with it problems of heterogeneity and connectivity [2,3] that have resulted in a technical and semantic inability to allow objects to be fully conscious of their environment, deploying features restricted to interaction with users and with other things. It is therefore a good moment to clearly define a means for things to collaborate with one another in a smart way, giving the possibility of generating more complex and useful services to their users.

In the last few years the number of things connected to the Web has exceeded the number of people connected to it, showing an ever-expanding rate of growth of connected things. These things are devices equipped with sensors and actuators that perform properties of interest measurements to the environment for their owners. These measurements generate huge amounts of raw data. Some of the raw data is processed by tailor-made models and programs. With an installed capacity like this—which continues to grow—it is difficult to reuse in other contexts. Context refers to the domain of application and the interests of users in the services that a thing can provide. For example, using a specific temperature sensor, this can be used to measure a person's temperature or to measure the temperature of a room of the house. The entity of interest changes, from person to room, as well as from a health domain to one of climate. Normally, WoT devices are created with specific functionalities and customized configurations. If users wish to use them for other functionalities, they must acquire new things and/or configure new services with an additional deployment of hardware and software, increasing cost, energy, time and overloading networks and communications.

The present project contributes to the search for models and mechanisms to semantically take advantage of the data provided by the smart things and provide a basis to generate a semantic interaction between them and their users in the WoT. This interaction should allow users to define new services transparently and easily, reusing installed capacity. This poses the challenge of enabling smart things to interact autonomously with each other, generating new services that consume information between things and execute specific functions transparently to the user. Achieving this interaction involves first solving the present problems of heterogeneity and connectivity, due to the great diversity and paucity of convergence of IoT devices, protocols, and tools. Middleware [4] and IoT servers partially solve the heterogeneity problem, allowing the things to have a digital representation of their metadata, data and services, unifying the information of the things.

The current work proposes an architectural proposal for semantic interaction in the IoT, which allows the management of a test bed where smart things interact with each other to generate new services. It is intended to get the user to define new behaviors in their environment, assigning the things to new services for which they were not initially designed. This is achieved by creating an interaction service that merges the information and services of the smart things selected freely by the user. It then makes it possible to communicate the states of the participating things and finally executes the new behaviors according to the conditions and information shared between them. To construct the test bed of semantic interaction between these smart things, first the problems of context, connectivity, and heterogeneity are solved. These problems are solved through the implementation of a model of semantic interaction between smart things in the WoT, using an approach of indexing, semantic search and ontologies developed in Niño-Zambrano [5] work.

The article is structured in the following way: in Section 2, related work is presented; Section 3 presents the architecture deployed for the implementation of the test bed; in Section 4, implementation of the tools of the test bed is presented; Section 5 presents the results of the tests and conclusions are drawn in Section 6.

2. Related Work

Various projects have generated specific test beds of interaction of IoT objects in well-defined contexts, such as education [6] and medical assistance [7], mediated by custom-made middleware servers. The servers are in charge of managing the information of the objects, performing procedures that calculate and display the tasks and actions on the data presented to the users. Defining the management is the direct responsibility of the owner and developer of the application. These projects are efficient in their application context, but made to measure and hardly reusable in other contexts, additionally they relegate many configurations and decisions of the system to the user, to other human actors, or third applications. If adding new interactions is desired, the new functions have to be built almost from zero, even the implementation of new hardware. Furthermore, these works focus mainly on the interaction between objects with people, unlike the present project that focuses on the interaction among smart objects themselves.

An important piece of research for the present investigation is that of Perera, et al. [8], in which they present an in-depth analysis on understanding context in the IoT, as well as that presented in [9], which focuses on reducing the tasks that the user must perform in order to connect to the services the objects provide. The present work uses the concepts of context introduced by Perera in order to better manage the interactions between objects.

The project [10] presents a summary of smart object types characterized by three dimensions: awareness, representation, and interactivity. Awareness is a smart object´s ability to understand events and human activities. The smart object in an awareness dimension can be activity-aware, policy-aware, or process-aware. Representation refers to a smart object´s application and programming model. Interaction denotes the object´s ability to converse with the user. The present work uses the policy-aware object's approach, because the interaction with the user is based on rules to initially configure test bed interaction services.

Perera, et al. [11], also studied the possibility of applying semantic techniques to index the objects of the IoT and connect to them. To do this, they proposed the discovery and automatic configuration of the objects, from which it would be possible to later use their services. The model presented (CADDOT) and the tool developed (SmartLink) allow the use of semantic techniques for annotating information in the SSN-XG ontology [12] and its configuration in a proprietary middleware, enabling its subsequent consumption by users and applications. The weak point of the proposed solution is that it indexes the technical characteristics of the device and not its ability to be applied in different application contexts, likewise, the potentiality of each smart object in its processing and storage capacity is not exploited, since this is delegated to the centralized server or middleware, generating a possible bottleneck with respect to the scalability of the solution. Finally, the work does not address the interaction between objects, but it does offer ideas of what a dialogue of identification would look like between an object and an intelligent entity, such as SmartLink.

The Estrada–Martinez and Garcia–Macias project [13] defines how the interaction problems in the IoT can be solved, using Semantic Web technologies. For this, they define the characteristics of a smart object and its deployment in an intelligent space. Ontologies were used to model the reasoning of the information shared between the objects and their users, concluding that semantic technologies can be applied in the IoT domain to generate richer interactions. The article makes it possible to establish first-hand the basic elements that a test bed of smart objects must take into account for its deployment, including the discovery of services, common messaging for all the participants and the notification of events in order to partially solve the problems of centered interaction between the user and the smart object. These elements were taken into account in the solution presented in this proposal. Unfortunately, the article is not very clear in the architecture used and in the complete solution, since it shows examples of query and illustrations of deployment in annexes separate from the ontology. Furthermore, the evaluation of the proposed solution is more descriptive than technical.

The project of Piyare [14] increases the remote monitoring on the states of objects that are provided with sensors. To this end, an architecture was implemented where the decentralization of intelligence is proposed and must be distributed among the physical layer of sensors, the coordination layer, and the supervision layer. Thus, for our project the intelligence of the object is also decentralized, on the one hand, in the object itself and on the other as a coordinating object. This facilitates the development of services and the use of different objects with different purposes, without having to depend totally of the configurations and capacities of other elements of the system. Unlike Piyare's work, our architecture involves the use of diverse platforms and types of smart objects and not necessarily a network of wireless sensors, although its proof of concept reveals the problems that must be overcome in a network of sensors, such as an efficient and common communication protocol, as well as an adequate management of the energy used by the objects.

The project by Rosen [15] aims to provide a standard communication system between all intelligent devices in an environment, be an event handler, and allow the generation of new applications and services to manage the functions of household devices. They are divided into three layers: the device layer defines the characteristics of the devices and their interfaces; the core layer handles information traffic between devices and applications; and the application layer comprises the applications made by users. The project stores the devices in a centralized server, restricting the context to a specific and custom assembly, and the applications have predefined parameters (for example, turning down temperature, switching on lights) that limit the possibilities that can be generated within the interaction of the devices. The interesting thing about the project lies in raising the possibility of creating semantic interaction between objects of the same environment with a purpose defined by the user.

The project by Ryu [16] propose an integrated sematic service platform (ISSP) for solving three main problems. The first problem is the integrated semantic discovery in distributed IoT domains. The second problem is the dynamic semantic representation between a myriad of IoT resources in real time. The third problem is the semantic data repository to archive a large amount of data collected from IoT devices. The ISSP handles various service knowledge domains using ontologies for each domain. The domain ontologies are created by a web-based authoring tool and handled by IoT-based service integration ontology. This approach has a strong link with the present work, but the main difference is in the ontology location within the model and the request handling towards the ontology. The Ryu project centralizes these features in a cloud-based system and the present work distributes the ontology and the request handle in each object. There are also commercial solutions, such as [17–19], where home automation services are provided that allow the user to remotely control home devices using mobile or web applications. They feature such devices as audio equipment, video equipment, lighting, alarms, cameras, curtains, and irrigation systems. These solutions are efficient, but costly, since the development of these proposals are restricted to specific brands or closed protocols, limiting the solution to only one type of device and closed models. The devices that do not have the same protocols will not be taken into account at the moment of generating the interactions.

3. Architecture Deployed

In the following, an architecture is deployed to develop the test bed. The architecture is based on the Niño–Zambrano [5] approach. It presents well defined layers, is service oriented and is addressed by a WoT semantic interaction model. The architecture has these main features:

- Distribute the information logic in the network edge ("fog computing" [20]), reducing the request time and the communication requirements.
- Reuse context and services, these are not linked with any entity, environment or purpose. The relationships depend on the usage of the objects.
- Allow objects linking, in order to cooperate in the new services creation. The objects can take part in different objectives.

Specific devices, protocols, and servers are named. However, depending on the design needs of the developers, this architecture can be instanced in any type of hardware that supports the mentioned characteristics of a smart object.

For a better understanding, the following basic concepts that are proposed in the architecture should be highlighted.

- Physical representation: consisting of the material and tangible part of a device.
- Digital representation: the information stored digitally on a physical representation of the device. In this case it is stored in a middleware called Xively and in each smart object.
- Resource: elements that have the objects to interact with the physical world. These can be either sensors, which examine the physical environment and capture the information that enters the system, or actuators, which perform an action on the physical environment in order to change its status.
- Entity: representation of a person, place or thing, which is related to one or more smart objects and which are subject to measurement or action.
- Entity of interest: an entity that is the object of an information search or an interaction by a user.
- Basic service: the service of a smart object by default, it is associated with the original operation with which it was created.
- Interaction service: generated by the interaction between two objects. Each interaction service is supported by an ECA and is executed by the object that contains the actuator resource.
- ECA (event-condition-action): XML document with the information of the contract between two smart objects. The ECA contains the logic of the interaction service and informs the objects about the variables and the resources that intervene in the execution of an interaction service. The event and action are related to smart objects and the condition is defined by the user.

The architecture presented in Figure 1 organizes the different tools and elements in a service-oriented model in which six layers are defined.

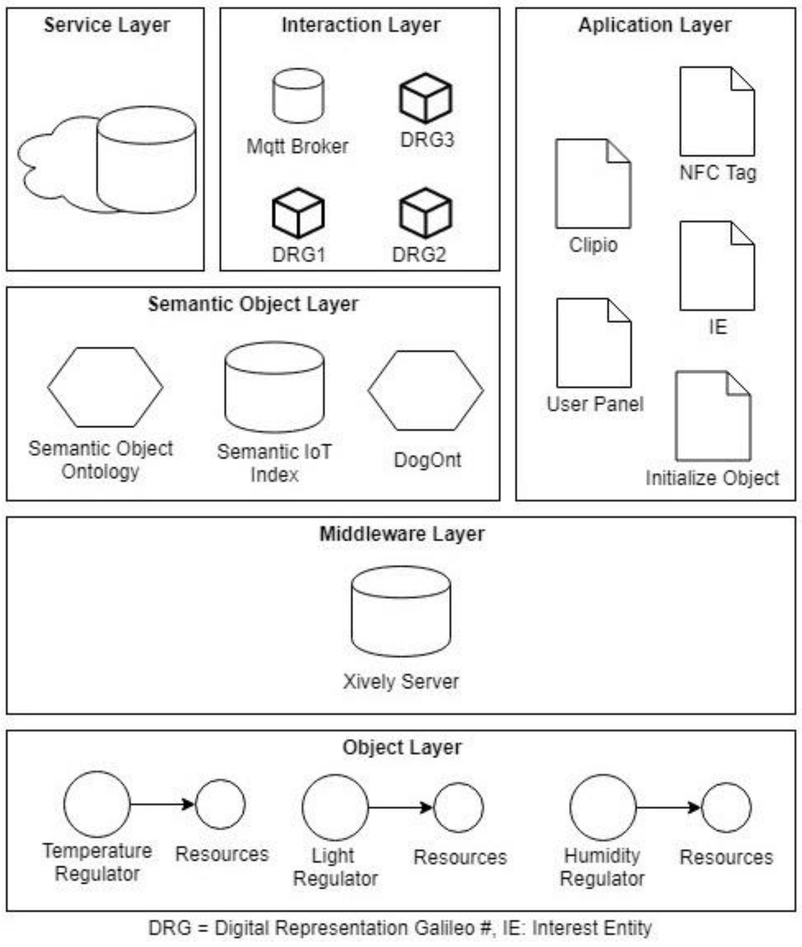

Figure 1. Architecture.

3.1. Object Layer

The physical representation of the objects is found, and the associated resources are defined. Each object is developed on an Intel Galileo board [21] supported on Linux with Yocto distribution containing the class directory named "SmartObject.py". Each object is related to an NFC tag where the unique identification of the object is stored within the system. This is done in order to allow interaction with the user intuitively through a mobile application called "Clipio".

For this test bed, three smart objects were deployed, with the following characteristics:

- Temperature regulator: related to the temperature sensor resource (Arduino Grove-Temperature module), the heating system actuator (physical heater with resistance), and cooling fan actuator (USB fan). These resources define the basic service that enables a constant level of temperature to be maintained in a closed room.
- Light regulator: related to the light sensor resource (Arduino Grove-Light module) and a light bulb actuator (table lamp). These resources define the basic service that facilitates detecting the absence of natural light in a closed room (with window) and turning on a source of artificial light.
- Humidity regulator in houseplant: related to the humidity sensor resource (Arduino Grove-Humidity module) and an irrigation actuator (simulated with a LED). These resources define the basic service that allows a house plant to be watered when the moisture level is low.

3.2. Middleware Layer

The Xively server [22] was used as a tool to store the digital representation of the smart objects. This server presents interfaces to create digital profiles of the objects. Within each profile the channels that determine the resources are created, the unique identification of the object is obtained, and labels and descriptions are created that will be used to characterize the object semantically. This layer allows the access of data and metadata of any object using standardized schemas such as JSON and resolves the heterogeneity and connectivity problem that are presented in the object layer, since it provides an API for the most commonly used platforms in the IoT.

The Xively server was used for practical purposes when implementing the test bed, however, the architectural deployed is not limited to this server—it is possible to add other IoT middlewares that allow hosting objects and their metadata, exposing this information in such standards as XML or JSON. The same applies with the object layer, where it is possible to use other development boards (rather than Galileo's Intel) and different sensors and actuators, giving the proposal a greater richness in object diversity.

3.3. Semantic Object Layer

Responsible for hosting the tools that support the semantic capabilities of the test bed. It has three main elements:

- Semantic Object Ontology (SOO) [5]: Is one of the relevant element of the present work. Uses OWL [23] as its language. It is stored and instantiated in each smart object of the test bed. The SSO have the information necessary to generate collaborative behavior between smart objects and their users. The SOO stores the smart object information of: context, metadata, interaction services (ECA), basic service, entity of interest, and resources in a set of instances. This element is consulted when the object needs to expose information about its metadata or create the contracts of interaction service with other objects. In Figure 2, the upper level of the SOO is presented.

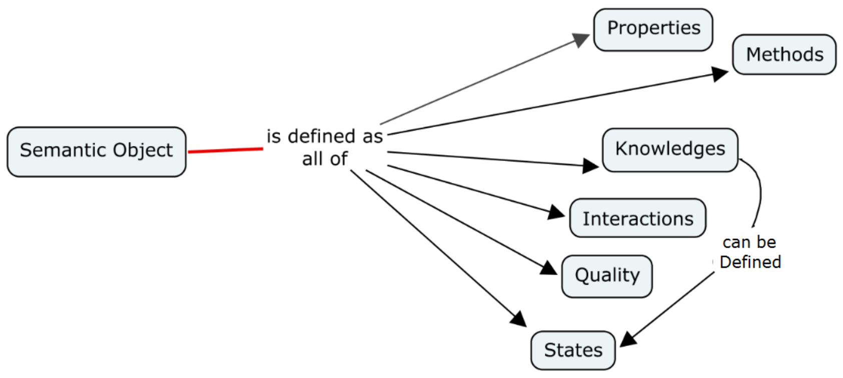

Figure 2. Module of the properties of the Semantic Object (object-oriented pattern).

In Figure 2, the upper level of the SOO is presented. Each of the concepts are defined as follows in Table 1.

- Semantic IoT index: an information structure stored in a server in the cloud, which can be consulted to carry out the discovery and connectivity processes to IoT devices in a specific context and place. When a user made a query, the index uses the concepts contained in a context ontology to organize the information retrieved from the digital representation of each object. Then the index exposes a list of smart objects and their services, according to the concepts of the domain ontology.
- DogOnt [24]: since it was necessary to define a dynamic context, this domain ontology was used because it contains the concepts and relationships to generate the context. This ontology is stored and instantiated in the semantic index element. The semantic index consumes the information from DogOnt and generates the context based on the ontology concepts. The DogOnt can be changed according the test bed needs. In this project we use the context of home automation included in DogOnt.

The semantic index and the DogOnt are not core elements of this project, those are from external developments, but we use those elements as an important part of our project. The union of the semantic index and DogOnt generates the domain, this information can be requested from different elements.

Table 1. SOO upper level concepts.

Name	Synonyms	Description
Properties	Attributes, qualities, traits, peculiarities, characteristics	Attributes that describe the object or device of the IoT. Mainly metadata.
Methods	Functions, Procedures, techniques, rules, plans	Functions to communicate data between objects. The data can be: properties, states or commands.
Knowledges	Judgment, understanding, intelligence, competence	Context information such as user profiles and services.
Interactions	Interrelation, collaboration, cooperation, participation, alliance, association	Relationships established with other objects. Have the contracts that define the dynamic interactions in which actions are modeled on an environment. This concept has the ECA pattern, where an event (can be any sensor or actuator of any object), is compared with a condition (normally defined by the user) if the condition is true, the system takes an action (activate/deactivate of any object actuator).
Quality	Evaluation, effectiveness, efficiency, relevance	Establishes objectives and indicators to monitor the state of the object's processes.
States	Stage, change, phase, course, moment, period, season	Values of the properties measured by the resources of the object in a given time.

3.4. Service Layer

The service layer exposes the results of the semantic index using web services based on SOAP [25], making it possible for applications and objects to consume metadata, data, and other context information, in an interoperable and transparent manner. This server is also responsible for recovering the digital representation of the objects in Xively and exposing it in the semantic index. There is a semantic index for each specific domain ontology that is provided on the cloud server.

3.5. Interaction Layer

The interaction layer supports the tools that allow the objects to communicate with applications and with other objects. It is based on the Eclipse Paho broker server (iot.eclipse.org) that manages the MQTT protocol [26]. The broker server is the intermediary where objects and users can communicate their states, requirements, and queries. All the messaging work of the elements that interact with the broker are supported by XML documents, which were coded to take into account the Open Group ODF standard [27] in order to define a common and interoperable information protocol by any type of smart object. The interactions defined on this layer are:

- Object to object: the event object shares the status of one of its resources to the action object.
- Object to user: the object responds to the user about the result of an internal operation.
- User to object: the user makes a query about the metadata of the object by means of a query to it. The user communicates the structure of a new interaction service to the event and action object by sending them an ECA. The user requests the change of status of a basic service (on/off). The user requests the status change of an interaction service (on/off). The user requests the elimination of an interaction service to the event and action object (eliminates ECA of each object).

Among the documents developed, those that intervene in the creation of an interaction service can be highlighted:

- MetaData.xml: contains the information of the object and its resources, Figure 3 presents a representation of the main elements.
- ECA.xml: supports the creation of an interaction service, Figure 4 presents a representation of the main elements.

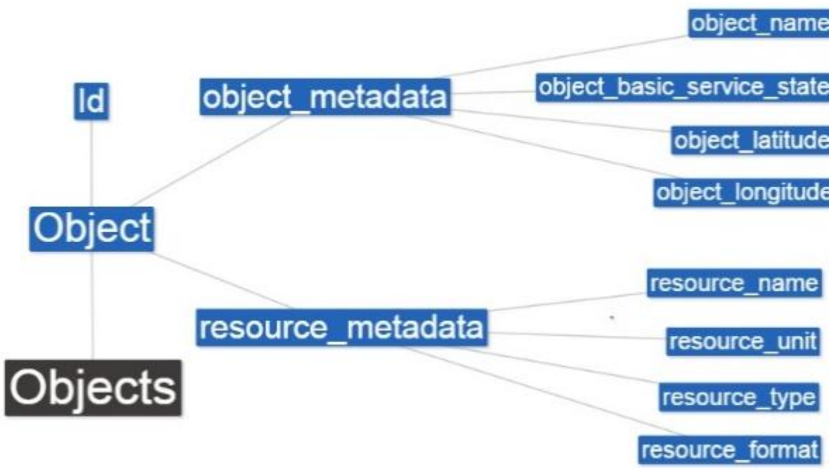

Figure 3. MetaData.xml.

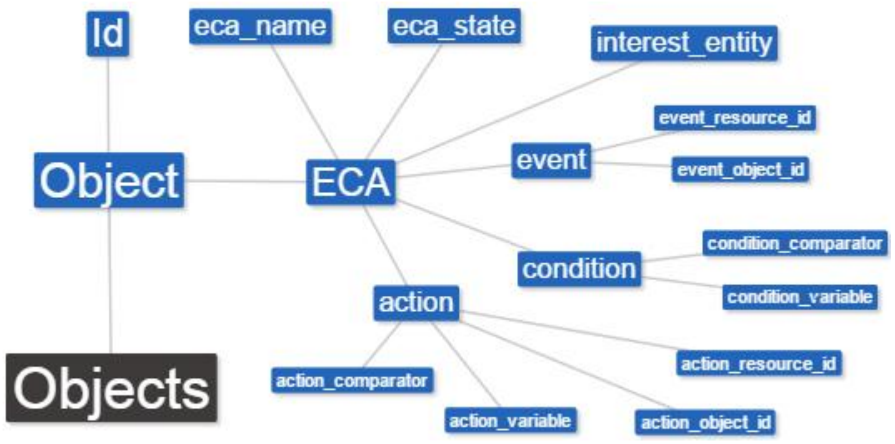

Figure 4. ECA.xml.

3.6. Application Layer

This layer hosts the applications that will allow the user to interact with the information to define the interaction services desired by the user. The applications and interfaces used are listed:

- Clipio: a mobile application that manages the objects of the test bed. It communicates its requests through the broker server, in this way it can create and eliminate interaction services, modify the states (turn on and off) of the basic services and interaction, and consult metadata about the objects. Clipio is developed under Android Lillipop (5.0 or higher); jdom (handling of XML), Paho (management of client MQTT) and kSOAP 2 (handling client SOAP) libraries were used. The application was installed on a LG Nexus 5 mobile device.
- NFC tag: corresponding to an entity or smart object. When aimed at an entity, the tag contains the name of the entity and its global position (longitude and latitude). Mifare Ultralight adhesive tags were used. The tag identified the room of the house and the coordinates in which the other nearby smart objects were displayed.
- User panel: presents information about the test bed such as instantaneous states of each resource of the objects, general information about the operation of each object, real-time information of the interaction activities between the objects. This information is stored and subsequently used in the evaluation of the test bed. It is developed using Html, Javascript, Php, and Ccs. For the database MySQL is used.
- Initialize object: a program in Python that initializes the smart object for the first time, retrieving the metadata from the object of the indexing service and requesting the user for missing data. It synchronizes the physical representation with the digital representation of the object (SOO). This interface interacts with the user when the object is registering in the system. It is supported in the Python "SmartObject.py" directory.

4. Test Bed Implementation

At this point the architecture is materialized in a test bed implementation, in order to analyze strengths and weaknesses in this kind of WoT test beds. The implementation of this is the core of the research because it takes the architecture as a conceptual base and develops a simple test bed, what will be evaluated, and finally the conclusions are extracted from the evaluation results.

For the development of the test bed, the UP Agile [28] software development methodology is used, which is defined in four phases. This section shows the three initial phases and the fourth phase corresponds to the evaluation of the test bed and is dealt with in Section 5.

4.1. Start-Up Phase

The main requirement for the project is: "The test bed must have tools that allow the creation, execution, and removal of services for the interaction and execution of basic services". To meet this requirement, critical use cases are: create, modify and eliminate interaction service, and also modify basic service. Figure 5 shows the diagram of critical use cases.

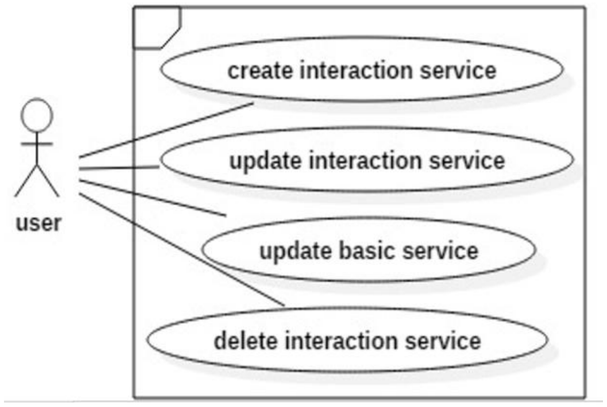
create interaction service
update interaction service
update basic service
delete interaction service
user

Figure 5. Use cases.

4.2. Elaboration Phase

The elements of the test bed are distributed in modules that make it possible to differentiate the processes that will develop the requirements and use cases proposed in the start-up phase. Within these modules, the one that responds to the use cases in Figure 6 is the smart object interaction module—SOIM, represented in Figure 6.

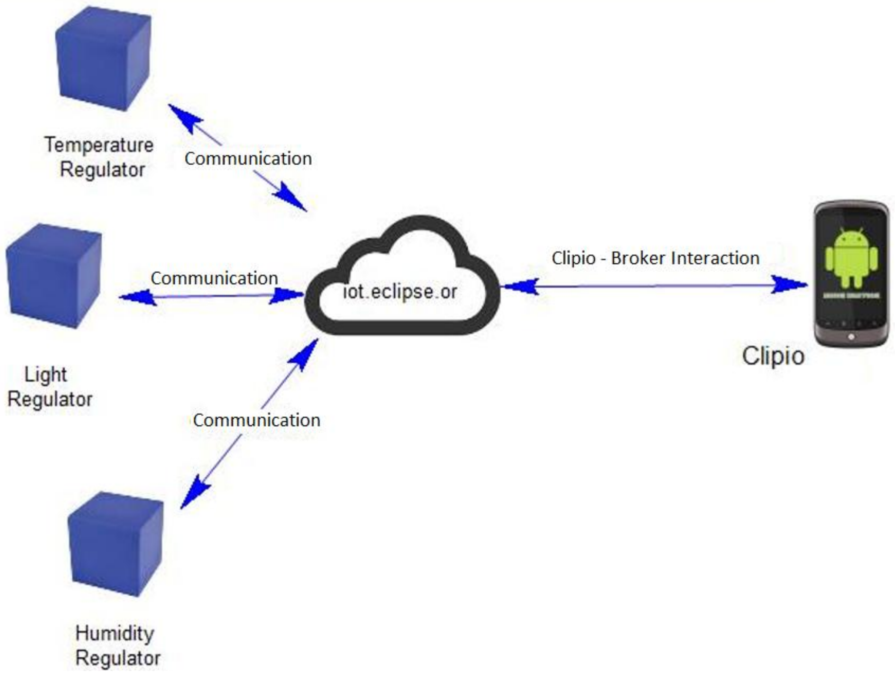

Figure 6. Elements of the SOIM.

SOIM enables Clipio to communicate with objects through the broker server to manage the creation, modification and deletion of service requests, and the objects in turn communicate with each other to execute interaction services. In order for SOIM to work properly, it ought to be taken into account that the test bed must previously comply with the following prerequisites (these are processes that are not mentioned in detail in the document, but are included in other modules):

- The participating objects of an interaction service must have complete metadata information in the SOO. This process is done using the initialization interface of the object (initialize object program).
- The participating objects of an interaction service must have their digital representation in Xively and also the semantic index must have previously recovered this representation and expose the object in the object server.
- Clipio must perform the authentication process within an entity of interest, so that the discovery of the objects includes the context associated with the entity of interest.

After the prerequisites are set, the system can generate a new ECA. To generate a new interaction service, we use the ECA element. The ECA is a xml document (Figure 4) that links two smart objects belonging to the same context. Each smart object stores the ECA relation in its SOO and the ECA document in its root directory. The ECA is configured in Clipio, the user can change the value of any ECA variable, name the ECA, and choose the involved smart objects. Figure 7 (five picture) shows the screenshot of an ECA generation interface. First, the user chooses two resources of any smart object (previously indexed in the semantic index element). Then, the interface enables a combo-box, for choosing the "condition_comparator"; and enable a text-box to enter the "condition_variable" and "action_variable" (this task is the same for both resources). Finally, the user sets the name and a description of the ECA and sends this request to the involved smart objects.

4.3. Construction Phase

The tools that expose the functionality to the users were implemented. In the following section, the main Clipio interfaces are presented with which the user can manage the objects to create, execute, and eliminate interaction services and execute basic services.

Figure 7 shows the different interfaces that Clipio displays to carry out the administration of the test bed services. The user goes through the home and start up interfaces of the application and then enters the interface where the creation of an interaction service begins. The user must enter the searches of the objects using words or phrases that identify him. In addition, he can adjust the search radius to detail the desired object. Clipio then requests the variables of the interaction service and finally the service is created. The remaining processes are similarly supported by Clipio in order to remove and modify services.

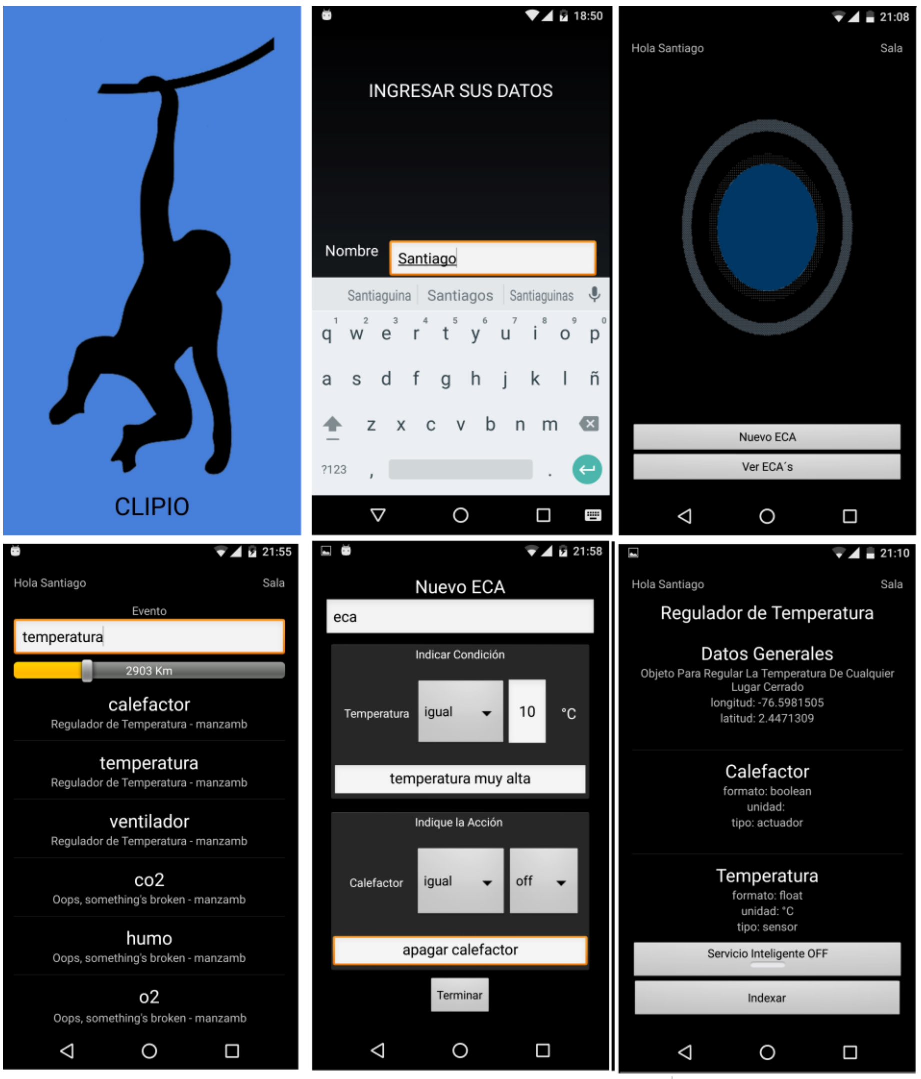

Figure 7. Clipio interfaces.

5. Evaluation

Although some of the measures used most to evaluate test beds and other implementations are the quality of service indicators, it was decided not to manage these since the measurement objective is the interaction between smart objects, and not the quality of the different aspects that the test bed may have. However, the test bed takes the following measures in order to ensure a good quality of service:

- Duplication of tasks [29]: in critical tasks such as sending MQTT messages to the server, the processes dedicated to publishing the messages are replicated at least three times. This is done in order to ensure the arrival of the package to the broker and its replication in the respective subscribers, and in this way increase the fault tolerance of the test bed.
- Focus on programming threads [30]: the elements of the test bed have been programmed to think of thread systems, seeking to organize and synchronize different redundant and heavy processes in separate threads to improve the response times of the test bed.
- Interactivity, delay, and criticality interactivity [31]: corresponds to the queries made by the user to the semantic index to detect objects and their resources. Delay is an indicator that is measured in the experiments under the name of interaction latency, with which a quality of interaction service can be established. Finally, criticality was taken into account when sending the states of each resource to the MQTT server, because these processes are critical in the test bed.

To evaluate the test bed, it was established that the appropriate indicator to perform the measurements should be latency in interaction. This consists of measuring the time elapsed since an element makes a query or sends a message, until the results are returned or executed.

The first tests showed very high interaction latency times, with average values of 13 s. Analyzing the critical points of the system, improvements were made in the processes and bottlenecks, achieving considerably improved response times as the alpha tests of functionality progressed. Therefore, a new group of tests was carried out to detect delays (latencies) with precise times and with suitable analysis. The steps taken for the analysis were:

- Measurements of latency times were applied only to the general processes.
- The times correspond to the exact latency from the start message until the confirmation with the completion message arrives. The times perceived may be a bit longer due to a delay in the recomposition of the interfaces.
- A table was collected that keeps all the messaging work between the objects and Clipio. They were organized according to the use case to which they belong and finally a latency analysis process was carried out.
- Twelve tests were carried out during twelve different days, one per day.
- Not all the tests were performed executing the same interaction services, since each test sought to test a particular functionality. However, the times were averaged, solving the problems of the differences in the number of records collected
- Since at individual times the individual measurements had extreme times (long and short) with respect to the average normal function, the ends were removed to avoid distortions. The eliminated times were stored separately in order to determine what happened at those points.

Taking the interaction latency times into account, an interaction service quality can be measured. The results of the operations on the test bed were identified. To be classified as real time, these should not be longer than 4 ms [15]. This time is taken as a reference for the experiments and Table 2 is defined to interpret the interaction service quality.

Table 2. Interaction service quality by latency.

Interaction Latency	Interaction Service Quality
[0.0–4.99]	Excellent
[5–6.99]	Good
[7–∞]	Poor

For the evaluation, the following considerations are taken into account:

- The measurements were applied to the use cases and not to the internal processes contained in them. Evaluations are presented for the use cases discussed in this document, considered as those that contain the main functionalities of the test bed.
- Using the user panel application, the messaging work was collected between smart objects and Clipio, organized by use cases and the latency analysis was carried out.
- Twelve tests were carried out over twelve separate days, one per day.
- The experiment was conducted using an internet access point that handles speeds of 9.74 Mbps for loading and 17.9 Mbps for download. In addition, the network is shared with approximately 70 other users.
- A stack of tests was designed keeping in mind that situations in which the test bed is sensitive to conflicts must be avoided. These can occur when two or more interaction services are run that execute the same actuator resource with opposite actions (as in the example given earlier, when service interaction 1 has the action of turning on the heating while interaction service 2 has the action of turning off the heating).

- Finally, a total of 57,824 messages are analyzed in each of the use cases.

Next, show the result analysis of all use cases in Figure 5.

5.1. Create Interaction Service

The reported times show an average latency of 7.68 ms, rating the result as a poor quality, according to Table 2. The standard deviation is 3.4, this shows a low critical dispersion in the results. Figure 8 displays low growth trend in time, but the use case has a significant peak in the 7 group. Because of this it is necessary to tune the use case's processes to improve the times. On the other hand, this use case is considered as the heaviest process and it uses a lot of computer resources. Thus the results are close to a good rate and have a low critical dispersion, revealing a relatively acceptable calcification. Table 3 shows the data relation and statistics results.

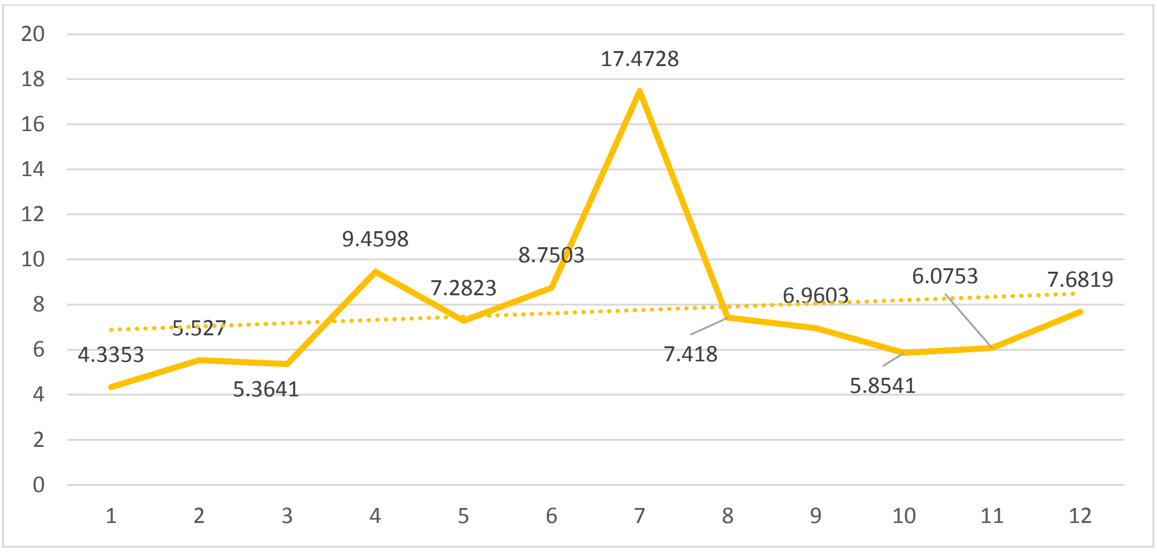

Figure 8. Create interaction service latency (day vs. ms).

Table 3. Create interaction service latency.

Group (day)	Average (ms)
1	43,353
2	5527
3	53,641
4	94,598
5	72,823
6	87,503
7	174,728
8	7418
9	69,603
10	58,541
11	60,753
12	76,819
Total Average	7,681,766,667
Variance	1,164,106,348
Standard deviation	3,411,900,274

5.2. Update Interaction Service

The reported times show an average latency of 5.01 ms, rating the result as a good quality, according to Table 2. The standard deviation is 0.7, this show a low dispersion in the results. Figure 9 displays a very low growth trend in time, and the use case does not have a significant peak. Table 4 shows the data relation and statistics results. This is the ideal use case, because it uses few computer resources.

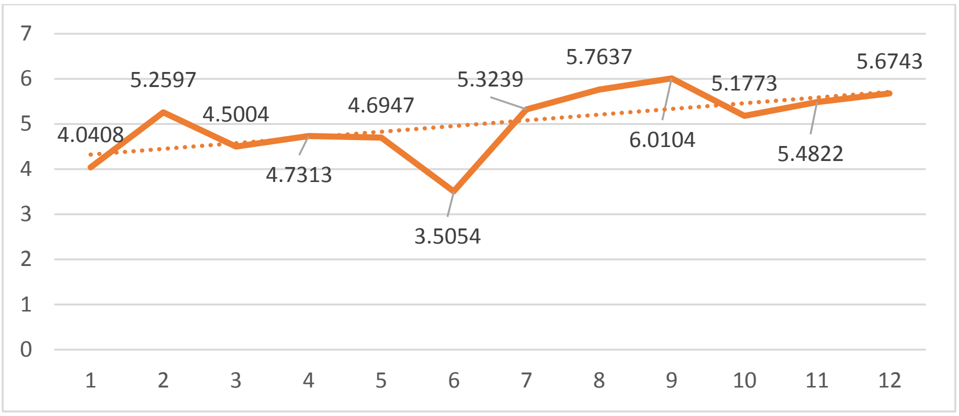

Figure 9. Update interaction service latency (day vs. ms).

Table 4. Update interaction service latency.

Group (day)	Average (ms)
1	40,408
2	52,597
3	45,004
4	47,313
5	46,947
6	35,054
7	53,239
8	57,637
9	60,104
10	51,773
11	54,822
12	56,743
Total Average	5,013,675
Variance	0.551071949
Standard deviation	0.742342205

5.3. Update Basic Service

The reported times show an average latency of 4.92 ms, rating the result as an excellent quality, according to Table 2. The standard deviation is 2.2, this shows a low dispersion in the results. Figure 10 displays a low growth trend in time, but the use case has a significant peak in the 8 group. This peak can be caused by human mistakes, broken elements, or internet problems. Table 5 shows the data relation and statistics results.

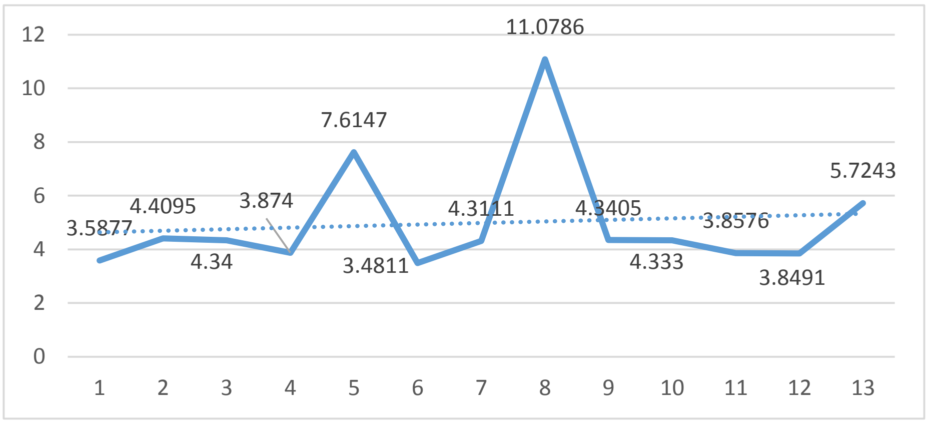

Figure 10. Update basic service (day vs. ms).

Table 5. Update basic service.

Group (dat)	Average (ms)
1	35,877
2	44,095
3	434
4	3874
5	76,147
6	34,811
7	43,111
8	110,786
9	43,405
10	4333
11	38,576
12	38,491
Total Average	4,923,075
Variance	4,913,900,033
Standard deviation	2,216,731,836

5.4. Delete Interaction Service

The reported times show an average latency of 3.82 ms, rating the result as excellent quality, according to Table 2. The standard deviation is 1.3, this shows very low dispersion in the results. Figure 11 displays low growth trend in time, but the use case has a significant peak in the 3 group. Table 6 shows the data relation and statistics results. This use case is considered the lighter process and it uses few computer resources.

Table 7 shows the results of the evaluation for each use case. An analysis of these times then follows below.

The times reported in Table 7 show an average latency of 5.36 ms, qualifying as good quality, according to Table 2. The tendency of most use cases is toward a behavior of quality of service between excellent and good. However, the process of creating an interaction service is the one that presents the worst time. This is because several processes that this use case handles must be replicated in the devices that participate in the creation of the interaction service. Other cases, meanwhile, do not have optimal times. This may be due to human error at the time of implementing the test. It is possible that many interaction services are operated without taking care to turn them off later, generating conflicts and delaying the response of the system. Although in these tests the system has been overloaded, it is not expected that many interaction services will be managed at the same time.

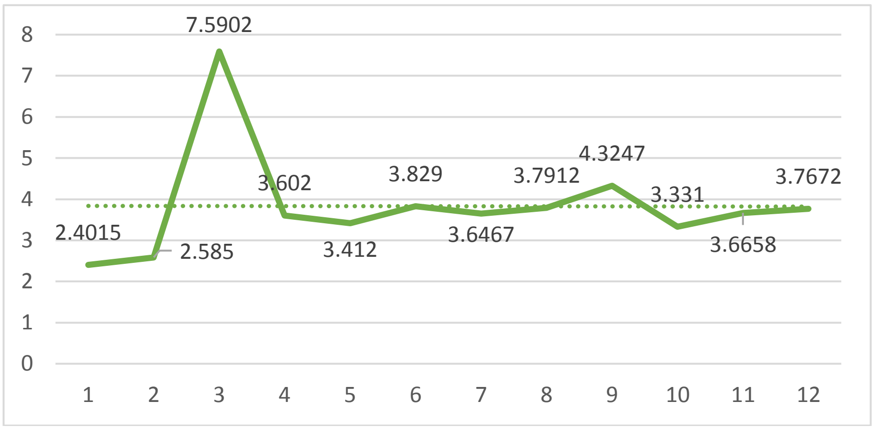

Figure 11. Delete interaction service (day vs. ms).

Table 6. Delete interaction service.

Group (day)	Average (ms)
1	24,015
2	2585
3	75,902
4	3602
5	3412
6	3829
7	36,467
8	37,912
9	43,247
10	3331
11	36,658
12	37,672
Total Average	3,828,858,333
Variance	1,683,288,504
Standard deviation	1,297,416,088

Table 7. Average times for use case.

Use Case	Average (ms)
Create interaction service	7.68
Modify interaction service status	5.01
Modify basic service status	4.92
Remove interaction service	3.83
Total	5.36

6. Conclusions

A test bed of semantic interaction was implemented in the WoT, using real physical devices and all the elements defined in the architecture were incorporated. The preliminary results allow us to confirm that the proposal of an interaction test bed in the WoT based on semantic indexes, web services and semantic objects, is viable for generating interaction test beds in the WoT. However, it is necessary to make more implementations to improve the tools, latency times and applied concepts.

The design of the experiments, their execution, and the analysis carried out suggests that the implemented test bed is viable as a solution to create semantic relationships between objects and generate new services adjusted to the needs of the WoT users. The result of the evaluation concludes that the response times are located in the range of a good rate.

The test bed is susceptible to executing conflicting interaction services. This occurs when two or more interaction services try to execute the same resource in the opposite way (for example, interaction service 1, turn on the heater and interaction service 2, turn off the heater) This problem has not been completely resolved, although the system has the functionality to turn the interaction services on and off, the decision to execute those that present conflicts has been left in the hands of the user's coherence. Likewise, the problem of several users in the same time was not addressed, which can generate conflicting or unwanted interaction services.

The project includes the SOO in each object to give it a semantic profile and generate a semantic representation of the object. However, it is not used in all the potential an ontology can give, such as generating complex reasoning or making autonomous decisions. It is expected that future versions will include other features in the form of use cases, so that user intervention becomes less and less necessary and the relationships between objects themselves more intuitive.

The future challenge is to implement a framework based in the architecture. The framework can generate the elements of a semantic interaction test bed with all the object's features. In order to improve the developer efficiency with the semantic tools and techniques. Other future work aims to achieve better performance and stability in the test bed.

Author Contributions: Conceptualization, Santiago Guerrero-Narváez and Miguel-Ángel Niño-Zambrano. Methodology, Miguel-Ángel Niño-Zambrano and Gustavo-Adolfo Ramírez-González. Software, Santiago Guerrero-Narváez and Dalila-Jhoana Riobamba-Calvache. Validation, Miguel-Ángel Niño-Zambrano and Dalila-Jhoana Riobamba-Calvache. Formal Analysis, Santiago Guerrero-Narváez and Miguel-Ángel Niño-Zambrano. Investigation, Santiago Guerrero-Narváez, Miguel-Ángel Niño-Zambrano and Gustavo-Adolfo Ramírez-González. Resources, Dalila-Jhoana Riobamba-Calvache. Data Curation, Santiago Guerrero-Narváez and Dalila-Jhoana Riobamba-Calvache. Writing-Original Draft Preparation, Santiago Guerrero-Narváez. Writing-Review & Editing, Miguel-Ángel Niño-Zambrano and Gustavo-Adolfo Ramírez-González. Visualization, Santiago Guerrero-Narváez. Supervision, Miguel-Ángel Niño-Zambrano. Project Administration, Santiago Guerrero-Narváez. Funding Acquisition, Gustavo-Adolfo Ramírez-González.

Acknowledgments: We are grateful to the University of Cauca, who provided the spaces and materials necessary to conduct the research. We are also extremely grateful to Colin McLachlan for suggestions related to the English text.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bauer, M.; Boussard, M.; Bui, N.; Carrez, F.; Jardak, C.; Loof, J.D.; Magerkurth, C.; Meissner, S.; Nettsträter, A.; Olivereau, A.; et al. Deliverable D1.5—Final Architectural Reference Model for the IoT v3.0. Available online: http://www.meet-iot.eu/deliverables-IOTA/D1_5.pdf (accessed on 10 March 2018).

2. Teixeira, T.; Hachem, S.; Issarny, V.; Georgantas, N. Service oriented middleware for the internet of things: A perspective. In Proceedings of the Towards a Service-Based Internet: 4th European Conference, ServiceWave 2011, Poznan, Poland, 26–28 October 2011; Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 220–229.
3. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
4. Razzaque, M.A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. Middleware for Internet of Things: A Survey. *IEEE Internet Things J.* **2016**, *3*, 70–95. [CrossRef]
5. Niño-Zambrano, M.A. Interacción Semántica de Objetos en La Web de las Cosas. Ph.D. Thesis, Universidad del Cauca, Popayán, Colombia, 2016.
6. Gómez, J.; Huete, J.F.; Hoyos, O.; Perez, L.; Grigori, D. Interaction System based on Internet of Things as Support for Education. *Procedia Comput. Sci.* **2013**, *21*, 132–139. [CrossRef]
7. Kawsar, F.; Kortuem, G.; Altakrouri, B. Supporting interaction with the Internet of Things across objects, time and space. In Proceedings of the 2010 Internet of Things, Tokyo, Japan, 29 November–1 December 2010.
8. Perera, C.; Jayaraman, P.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context-Aware Dynamic Discovery and Configuration of 'Things' in Smart Environments. In *Big Data and Internet of Things: A Roadmap for Smart Environments*; Studies in Computational Intelligence Book Series; Springer: Berlin/Heidelberg, Germany, 2013.
9. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454. [CrossRef]
10. Kortuem, G.; Kawsar, F.; Fitton, D.; Sundramoorthy, V. Smart objects as building blocks for the Internet of Things. *IEEE Internet Comput.* **2010**, *14*, 44–51. [CrossRef]
11. Perera, C.; Jayaraman, P.P.; Zaslavsky, A.; Georgakopoulos, D.; Christen, P. Sensor discovery and configuration framework for the Internet of Things paradigm. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014.
12. Compton, M.; Barnaghi, P.; Bermudez, L.; García-Castro, R.; Corcho, O.; Cox, S.; Graybeal, J.; Hauswirthf, M.; Henson, C.; Herzog, A.; et al. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semant. Sci. Serv. Agents World Wide Web* **2012**, *17*, 25–32. [CrossRef]
13. Estrada-Martinez, P.E.; Garcia-Macias, J.A. Semantic interactions in the Internet of Things. *Int. J. Ad Hoc Ubiquitous Comput.* **2013**, *13*, 167–175. [CrossRef]
14. Piyare, R.; Lee, S.R. Towards Internet of Things (IOTS): Integration of Wireless Sensor Network to Cloud Services for Data Collection and Sharing. *Int. J. Comput. Netw. Commun.* **2013**, *5*, 59–72.
15. Rosen, N.; Sattar, R.; Lindeman, R.W.; Simha, R.; Narahari, B. HomeOS: Context-Aware Home Connectivity. In Proceedings of the Conference on Pervasive Computing and Communications (PCC'04), Las Vegas, NV, USA, 21–24 June 2004.
16. Ryu, M.; Kim, J.; Yun, J. Integrated Semantics Service Platform for the Internet of Things: A Case Study of a Smart Office. *Sensors* **2015**, *15*, 2137–2160. [CrossRef] [PubMed]
17. Sixte, J.; Peralta, N. SMARTEC: Domótica y Seguridad. Available online: http://www.smartec.com.ar/ (accessed on 25 February 2016).
18. OZOM. Base de Conocimientos | OZOM. Available online: http://help.ozom.me/help_center (accessed on 7 January 2016).
19. Control4. Home Automation and Smart Home Systems | Control4. Available online: http://www.control4.com/ (accessed on 7 January 2016).
20. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
21. IntelSupport. Intel®Galileo Board Specifications. Available online: http://ark.intel.com/products/78919/Intel-Galileo-Board (accessed on 7 January 2016).
22. Xively. Xively by LogMeIn. Available online: http://xively.com/ (accessed on 7 January 2016).
23. Jones, M.B.; Beach, J.H.; Ludaescher, B.; Michener, W.K.; Schildhauer, M.P. *The Science Environment for Ecological Knowledge*; The National Science Foundation: Alexandria, VA, USA, 2008.
24. Bonino, D.; Corno, F. *Dogont-Ontology Modeling for Intelligent Domotic Environments*; Springer: Berlin/Heidelberg, Germany, 2008.

25. Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.-J.; Nielsen, H.F.; Karmarkar, A.; Lafon, Y. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Available online: https://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (accessed on 7 January 2016).
26. Banks, A.; Gupta, R. MQTT Version 3.1.1. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html (accessed on 26 April 2018).
27. Främling, K.; Buda, A.; Kubler, S.; Madhikermi, M.; Maharjan, M.; Lützenberger, J.; Dimitris, K.; Nikolaos, M.; Min-Jung, Y.; Jacopo, C.; et al. Open Data Format (O-DF), an Open Group Internet of Things (IoT) Standard. Available online: www.opengroup.org/iot/odf (accessed on 26 April 2018).
28. Ambler, S.W. The Agile Unified Process (AUP). Available online: http://www.ambysoft.com/unifiedprocess/agileUP.html (accessed on 9 April 2018).
29. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
30. Vithya, G.; Vinayagasundaram, B. QOS by Priority Routing in Internet of Things. *Res. J. Appl. Sci. Eng. Technol.* **2014**, *8*, 2154–2160. [CrossRef]
31. Nef, M.A.; Perlepes, L.; Karagiorgou, S.; Stamoulis, G.I.; Kikiras, P.K. Enabling qos in the internet of things. In Proceedings of the Fifth International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ 2012), Chamonix, France, 29 April–4 May 2012; pp. 33–38.