Implementation of an experimental platform for the Social Internet of Things

Roberto Girau, Michele Nitti, Luigi Atzori
Department of Electrical and Electronic Engineering - University of Cagliari, 09123 Cagliari, Italy
{roberto.girau, michele.nitti, l.atzori}@diee.unica.it

***Abstract*—The convergence of the Internet of Things (IoT) technologies with the social networking concepts has led to a new paradigm called the Social Internet of Things (SIoT), where the objects mimic the human behavior and create their own relationships based on the rules set by their owner. This is aimed at simplifying the complexity in handling the communications between billions of objects to the benefits of the humans. Whereas several IoT platforms are already available, the SIoT paradigm has represented only a field for pure research and simulations, until now.**

The aim of this paper is to present our implementation of a SIoT platform. We begin by analyzing the major IoT implementations, pointing out their common characteristics that could be re-used for our goal. We then discuss the major extensions we had to introduce on the existing platforms to introduce the functionalities of the SIoT. We also present the major functionalities of the proposed system: how to register a new social object to the platform, how the system manages the creation of new relationships, and how the devices create groups of members with similar characteristics. We conclude with the description of possible simple application scenarios.

***Index Terms*—Internet of Things; social network; IoT platforms**

I. Introduction

Society is moving towards an “always connected” paradigm, where the Internet user is shifting from persons to things, leading to the so called Internet of Things (IoT) scenario. The IoT is expected to embody a large number of smart objects identified by unique addressing schemes providing services to end-users through standard communication protocols and be composed of trillions of elements interacting in an extremely heterogeneous way in terms of requirements, behavior and capabilities. Communications will not only involve persons but things thus bringing about the IoT environment in which objects will have virtual counterparts on the Internet. Such virtual entities will produce and consume services, collaborate toward common goals and should be integrated with all the other services.

To give the possibility to all these entities to be able to communicate efficiently, new paradigms are needed . Indeed, there are scientific evidences that a large number of individuals tied in a social network can provide far more accurate answers to complex problems than a single individual. For instance, in [1] the authors introduce the idea of objects able to participate in conversations that were previously only available to humans. Analogously, the research activities reported in [2] consider that, being things involved into the network together with people, social networks can be built based on the Internet of Things and are meaningful to investigate the relations and evolution of objects in IoT. This has also brought to the convergence of IoT and social network paradigms, as analyzed in [3], which depicts the scenarios where an individual can share the services offered by her/his smart objects with her/his friends or their things through widespread social networks. In [4] and [5], explicitly, the Social IoT (SIoT) concept is formalized, which is intended as a social network where every node is an object capable of establishing social relationships with other things in an autonomous way with respect to the rules set by the owner; indeed, such a network has the potential to solve problems of network navigability and information/service discovery thanks to the possibility to navigate a social network of “friend” objects instead of relying on typical Internet discovery tools that cannot scale to the trillions of future devices.

While the IoT scenario presents several implementations, such as [6] and [7], until now the SIoT has represented only a field for theoretical analysis. The purpose of this work is to propose a possible implementation for the SIoT, where objects can create their own relationships, create groups and produce and consume services. Moreover, some possible applications are discussed in order to show the benefits of our platform.

The paper is organized as follows. In Section II we describes the guidelines that drove our implementation with respect to the already existing platforms, while in Section III we show the major functionalities implemented by our platform. Section IV illustrates some possible applications that can use our platform, where objects can create their own relationships and provide simple services to the users. Finally in Section V, we draw final remarks.

II. Background

In this section, we show the main features of the SIoT model. Moreover, we highlight some requirements which have been considered for the system specification, including a reasoned analysis of the state of the art of IoT implementations and the description of the solutions and choices which have driven the design and implementation of our system architecture.

A. Main SIoT features

Without losing of generality, we refer to the SIoT model proposed in [8]. According to this model, the objects mimic the human behavior and create their own relationships based on the rules set by their owner. In the same way humans have relationships with their family, objects create the *parental object relationship* with similar objects, which are built in the same period by the same manufacturer (the role of family is played by the production batch). Furthermore, humans establish relationships when they share the same location, this is personal, e.g. cohabitation, or public, e.g. work, and likewise the objects create *co-location object relationship* and *co-work object relationship*. A further type of relationship is established when objects come into contact, sporadically or continuously, for reasons purely related to relations among their owners (e.g., devices/sensors belonging to friends) and it is named *social object relationship*. Finally, objects owned by the same user, such as smartphone, tablets and game consoles can create *ownership object relationship*.

B. IoT platforms

From the analysis of the IoT scenario [9], we identified around 10 different platforms. All these systems have common characteristics, especially the followings:

- the objects use a HTTP protocol to send and receive data. This choice allows a high interoperability among the different platforms;
- an intermediary server is used. The objects do not communicate directly with each other;
- every object has a "data point" associated with it on the server-side to keep track of the data sent;
- the methods POST and GET are used to send and request data;
- a tag is assigned to every data point;
- data point discovery is performed using tags through an internal search engine;
- the system identifies every object with its API key.

The need to store and process data from a multitude of sensors has led the implementation of the IoT to the creation of platforms with the main purpose of data logging. Among the European platforms, Cosm (former Pachube) [6] is one of the biggest ones: it has been presented as a platform to store and redistribute real-time data, freely usable, which manages millions of devices per day. One of the most dramatic demonstrations of Cosm's potential was the visualizations of data that showed the radiation levels around the Japan and especially near the nuclear reactor in 2011.

Nimbits [10] is an open source java web application, built on Google App Engine, which can provide complex functionalities such as email alert, math calculations and complex queries on API Wolfram Alpha, in addition to simply storing and processing data. Every user can define data points and use them to share several kinds of data. The integration with Twitter, Facebook and Google+ allows to manage your data points, to share sensor diagrams, activate alarms, etc.

Paraimpu [11] is a Web of Things platform, which allows people to connect together sensors, actuators and other web applications, taking care to forward data among the objects [12]. Moreover, through the integration with Twitter, it is possible for a user to obtain and use data from a friend.

Finally, ThingSpeak [13], was founded as an open source branch of IoBridge [7] and shares with it the main features. ThingSpeak is an IoT application that allows users to store and retrieve data from objects through HTTP communications. Moreover, it enables the creation of several kinds of application, involving different pairs of API keys, such as GPS tracking and data logging. ThingSpeak API enables to perform averaging, summing, rounding and time-scaling. In addition, this platform allows for integrating several data representation, such as JSON, XML and CSV.

However, none of the platforms foresees some kinds of social relationships between objects. Indeed, even if integration with main human social networks is allowed, the objects can not communicate independently.

C. RESTful vs WS-*

Web Services (WSs) are widely used in several IT systems; they can be defined as development techniques for interoperable and distributed applications that make use of standard protocol such as HTTP [14]. WSs can be classified in two major categories: WS-* and Representational State Transfer (REST).

The former declares its functionalities and interfaces using a Web Services Description Language (WDSL) file. Communications are encapsulated using Simple Object Access Protocol (SOAP) usually using the HTTP protocol. WS-* is mainly used in enterprise applications where interoperability with other applications is not an important issue; moreover, for applications with advanced security requirements [15] or for WSNs solutions [16] it can provide good results.

The RESTful architecture is based on the concept of resources as representation of the objects, that are uniquely identified through Uniform Resource Identifiers (URIs). Through the HTTP protocol is then possible to obtain, delete, post or update object information using a given method (GET, DELETE, POST, PUT). The payload of the message can be incapsulated in a negotiated format such as XML or JSON. A RESTful architecture is then lightweight and scalable and then fits perfectly with the principles and the current protocols of the Internet.

To maintain interoperability with existent IoT platforms and future implementations, we decide to adopt a RESTful approach, and then every entity in the SIoT is represented using JSON, XML or CSV format.

III. Platform Implementation

In this section we describe our implementation of the SIoT platform, pointing out its major functionalities required to run simple applications.

A. Server Architecture

As presented in [8], Figure 1 shows the main components of the platform. The network layer is needed in order to transfer data across different networks, while the core of the proposed platform is represented by the application layer, where IoT applications and middleware functionalities are

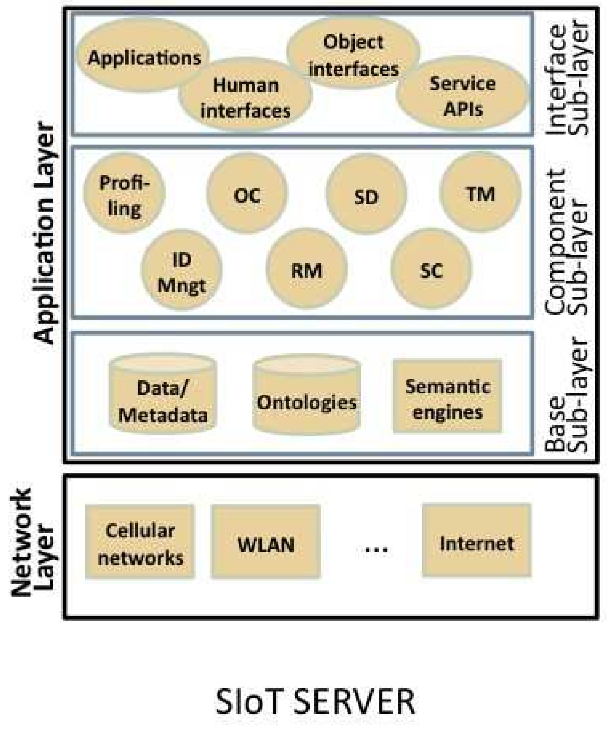


Fig. 1: Proposed platform implementation (taken from [5])

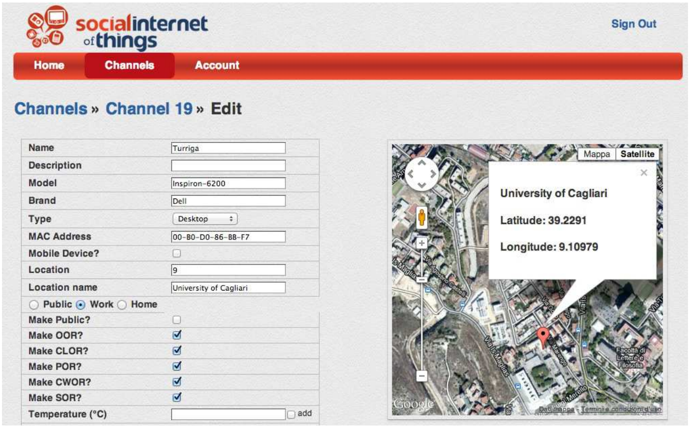
developed and which consists of three sublayers. Not all the functionalities have been implemented, since the platform is still in its infancy and the developed vertical applications are very specific.

The *Base Sublayer* includes the database for the storage and the management of different data types, such as temperature, latitude, longitude and humidity. Objects can memorize up to 16 different data fields; the first 12 fields have a fixed type, whereas the others 4 are reserved for future uses. For example, field 1 is always used to track temperature data, field 3 is used to track voltage data and so on. The last four fields are left for uses decided by client application devolopers.

The *Component Sublayer* implements the functionalities of: objects profiling, which is needed in order to configure information about the objects; ID management, which assigns a unique ID to every object in order to identify them; owner control (OC), which enables the users to specify the objects' behavior; and the relationship management (RM), which has to create and manage the relationships of every object.

The other functionalities, i.e. the service discovery and composition (SD and SC) and the trustworthiness management (TM) [17], are not implemented and, when needed, are provided by the specific vertical application.

The *Interface Sublayer* is where the interfaces and the service APIs, such as read/write API keys, are located.



Channels » Channel 19 » Edit

Name	Turriga
Description	
Model	Inspiron-6200
Brand	Dell
Type	Desktop
MAC Address	00-B0-D0-86-BB-F7
Mobile Device?	☐
Location	9
Location name	University of Cagliari

Public [✓ Work] Home

Make Public?	☐
Make OOR?	✓
Make CLOR?	✓
Make POR?	✓
Make CWOR?	✓
Make SOR?	✓
Temperature (°C)	☐ add

University of Cagliari

Latitude: 39.2291

Longitude: 9.10979

Fig. 2: Registration of a new channel

B. *Server Functionalities*

As a RESTful architecture, a URI is associated to every resource. These resources are modeled as follow:

- every object in the server is identified as a *channel*. A channel represents a real entity, such as a smartphone, a laptop or a sensor;
- every device can have one or more fields associated with it, based on the number of its sensors; each field is identified with a data point.

When a user wants to register a new channel, the profiling module is activated. As shown in Figure 2, the owner can then indicate the characteristics of the objects such as the name, a description and its mobility. For those devices with enough computation capabilities, such as a laptop or a smartphone, some information about the object itself are provided automatically by the application, e.g. the brand and the MAC address: in this way it is possible to greatly shorten the registration process for the benefit of the owner. Eventually, if the owner is registering a fixed device, such as a desktop or a printer, it is possible to insert the location of the object, which enables the creation of location-based relationships, such as the co-location one. During the registration phase, the owner can choose which relationships the objects can create with other peers and which sensors, and consequently which fields, should be activated. When the registration is completed, the ID management module assigns a unique ID number to the object.

Objects can then start to create their own social relationships that are managed by the RM in two different ways:

- **Profiling relationship**. These relationships are generated based only on the profile information of the objects, and do not depend on the owner behavior. To this category belong *Ownership Object Relationship* (OOR), *Co-Location Object Relationship* (CLOR) and *Parental Object Relaziontship* (POR). Indeed, OOR are created among objects registered in the SIoT by the same user. When objects have the same value of the attribute *model*, a POR is created. To activate a CLOR two objects need to be fixed in the same *location* (numeric ID).

- **Dynamic relationship**. These relationships are created when users, and consequently objects, interact with each other and satisfy the rules defined in [5]. To this category belong *Co-Work Object Relationship* (CWOR) and *Social Object Relationship* (SOR). In particular, it is important that the server recognizes two objects in the same location, even if the objects are not in visibility.

The RM module is activated every time a new object is registered in the SIoT or every time an object sends information about its own location or about the IDs (i.e. mac address, RFID id) of the objects it has encountered. For Dynamic relationships, the RM module is activated by events about devices visibility when a device posts a sensed mac address or RFID. For CWOR events, it is necessary either a work-type location post in the same time of the ID post or that at least one of the two devices is fixed in a work-type location.Two devices must be in visibility in two separate thirty-minute intervals spaced at least 8 hours for a friendship request storing. Every pass in the friendship request process is managed by the server and devices need only to send the sensor data. For example, as shown in Figure 3, device 1 senses the presence of another device, number 2, and then sends this information to the server, updating the relative data point, in particular the mac address data field. The server recognizes this field as a potential event and checks if this mac address belongs to a registered object. If this is the case, the RM module is activated to verify if, with the last data received, there are the conditions to create a new dynamic relationship. Eventually, the friendship request from device 1 is stored and if the device 2 performs a friendship request toward the device 1 as well, a new relationship is created.

When an object needs to send or retrieve its own data to the server, it uses its write API key, known only by the object itself. Instead, when an object needs to retrieve friends data from the server, it uses the read API key of the object to which it wants to retrieve data. The read API key is only known by the object itself and by its friends but it is allowed to share read API keys in those cases in which data from friend of a friend are required. Data from the server can be obtained using one of the two following methods:

- Pull. Every object requires data at regular intervals or when needed.
- Push. Data are sent from server to objects when available. Indeed, a HTTP daemon always listening is needed on every capable device. On smart devices, such as smartphone, tablet or laptop, it is also possible to use the push system of the operating system of the device itself.

If, during registration, the owner has set the object as public, any object that wants to retrieve information about the status of the public object, using the pull method, just needs to know its ID number; however, if an object wants to require data from a private object, it will also need the read API key of that object. Indeed, a device can retrieve the list of the IDs and the read API keys of its friends. This list can be obtained, in JSON, XML or CSV format as shown in the Listing 1, through a POST to the REST resource *friendships*, using the write API key.

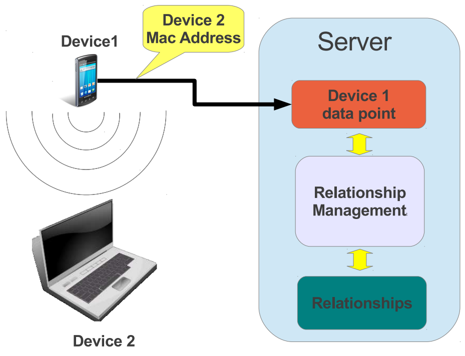


Fig. 3: device identification during a meeting

Instead, with the push method, the server sends directly to all the objects in the friendship list the available data. In the same way, when a new friendship is created the server sends the updated list to the new object.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<relationships>
   <relation-type>OOR</relation-type>
   <channel-id type="integer">12</channel-id>
   <read-api-key>UXFN5F6SWXWIM3UM</read-api-key>
   <relation-type>OOR</relation-type>
   <channel-id type="integer">167</channel-id>
   <read-api-key>Q4CL7WQM5SUNSMS5</read-api-key>
   <relation-type>POR</relation-type>
   <channel-id type="integer">32</channel-id>
   <read-api-key>GIG8Z8WQBAPMO17G</read-api-key>
   <relation-type>POR</relation-type>
   <channel-id type="integer">126</channel-id>
   <read-api-key>WV3EOODHHISN8JZK</read-api-key>
   <relation-type>SOR</relation-type>
   <channel-id type="integer">4</channel-id>
   <read-api-key>C71DKJVOEDLU2K06</read-api-key>
</relationships>
```

Listing 1: Response to a *friendship* list request

C. *Groups management*

The relationships we identified so far, can include a large number of objects, and this leads to problems such as a long time for service discovery, since not all the objects can be helpful for a particular request. If we consider, for example, a large company, all the devices of every employee and the devices of the company itself, such as printers, scanners and desktops, would be tied by a co-work relationship, whereas it would be useful to have a group of objects belonging to different departments.

It is then important to have a tool to divide social relationships into groups. In the same way, humans create groups of particular interest (football teams, politic groups, online shopping) in SNs, the objects can create their own groups based on the applications they are using.

Three different solutions can be implemented, as described in the following:

Client-side groups management. The application must be able to create automatically the various groups with minimal user intervention and to do so, it has to verify some conditions that are specific to the use-case. These conditions are not limited by the fields of the device, i.e. its sensors, but may include other information. For example, an application can distinguish among the devices of employees that belong to different departments: it first verifies if they share a CWOR and then it could check in a file for the employee ID to perfectly associate the devices of the other employees to the group. The benefit of this solution is that the groups created are exactly those required and specific to the application. The disadvantage is that, since the application manages the groups, different devices may create different groups that instead should be coincident, if, for example, a device has an out-of-date file, and moreover this solution increases the workload of the devices.

Server-side groups management. In this case, a user must create manually the group and set the rules, based on the fields associated to each device. Then the server takes care of binding to the group all the devices that comply with the above mentioned rules, in the same way the RM module creates the dynamic relationships. This solution has the advantage that the devices have a lighter workload but there may be excessive fragmentation of the groups, due to users that should belong to the same group creating new groups with different rules and the need to identify superusers for the creation of groups.

Hybrid solution for groups management. Groups are managed on the server-side, but the rules are set by the client using the fields provided by each device; information are tagged in order to help the server to identify the characteristic of each group. Only one group is created since the server associated all similar groups with the same tags, and then the workload on the devices remains light. The list of members of a group, as a REST resource, can be required in the same way of the friendship list.

D. SIoT Prototype Development

We realized our implementation based on ThingSpeak project [13], and we concentrated our efforts on social networking aspects. We implemented the relationship management module and the possibility to create and manage groups. Furthermore, we developed a location indexing system based on Google Maps to localize coherently fixed devices. We modified the channel structure to handle more than eight sensors and to manage text messages with tags among devices; we improved the owner control management of the objects and allowed devices to update remotely their own profile.

The service is available for tests here[1]. It is an alpha version still under development, but it is already capable to create relationships among registered devices. As ThingSpeak, SIoT is an open source project, and source code is released by the beta version.

[1]http://siot.diee.unica.it:8088/

IV. SCENARIOS

In this section we describe some scenarios under development at the Faculty of Engineering of Cagliari that use the defined platform, where objects create their own relationships and groups, in order to provide several functionalities to the final users. Considering, smartphones, laptops and sensors in the area of the campus, we show some possible applications that exploit their social relationships. The focus of these applications is to provide usuful information to the object owners (the students, tipically) with minimal human intervention, except in the initial configuration phase. Every object can send messages, i.e. the data obtained by its sensors, to its own dashboard, and these messages can be seen by its friends as updates.

A. Lectures Information

The first application addresses the problem of communicating in an efficient way information such as the time of classes/tests and the availability of new teaching materials. The main problem to address concerns the identification of the actual recipients of these information. In the area of the campus, all the students' devices would be tied by a CWOR; this relation could be useful to share generic information, such as holidays, employers strikes, student elections. However, it should be impossible to provide more granular services based on the field of study, the academic year or the single classes. Here comes into play the groups of interests concept.

With the use of the hybrid method, described in Section III-C, the application decides the parameters to create the groups and the server manages and merges them, if necessary. This solution allows the same precision as the client-side method with a light workload on the devices.

The application gives information (alerts, web links, timetables and others) about the lectures to all the devices participating to the same group. As initial configuration of the application, it is possible to download the class timetable and the building map of the campus, for example using QR-codes. The application is able to assess when the professor and at least 50% of the students of the group are in the same location by sensing vicinity with the Bluetooth interface, and then it can notify to all the missing students of that group that a class is started and provide information to guide them to that class location.

B. Student car pooling

Rebecca needs to go to the university cafeteria after her class to have lunch but unfortunately she does not own a car. Her smartphone can create SOR with other students' smartphones at the cafeteria or CWOR with her colleagues' smartphone, so when Rebecca needs a ride, she can simply use an application to discover if any other student is going to the cafeteria. The application automatically sends messages asking for a ride to all the devices that meet the parameters set by the user, for example:

- the number of common friends
- the membership to the same groups

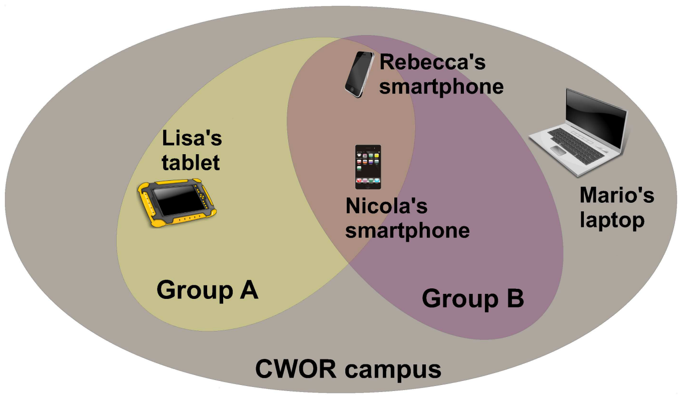


Fig. 4: Trustworthiness by social parameters

- the frequency meeting

In the same way, the application can be set to receive requests only from devices with a certain relationship or that belong to certain groups and sort them accordingly. It is thus guaranteed a certain level of trustworthiness using only social parameters. As shown in Figure 4, consider for example the set of all the devices tied by a CWOR in the campus in the grey area. Since Rebecca attended courses in Mathematics and Computer Science, she also belongs to groups A and B, respectively. Nicola's smartphone belongs to groups A and B as well, since Nicola is her study partner. Mario's laptop shares only a CWOR with Rebecca's smartphone since he studies in another department, whereas Lisa's tablet is part of the group A. In this scenario, Nicola's device represents the most trustworthy device and then it represents the best choice to share a ride.

C. *Other university social life events*

In the same way of the use-cases illustrated above, it can be possible to manage general information on the use of common areas such as university cafeteria, library and computer workstations. The library room is usually very crowded depending on the time of day, especially the PC workstations run out quickly. In this case, the application provides an assessment of the crowding of these environments by monitoring the number of devices and allows to give an approximate indication of the availability of seats. By the statistics of attendance, the application provides information on timing, categorized by crowding. Likewise, you can assess the crowding of canteen or cafeteria, in order to decide whether you prefer a less crowded place for time limits, or a popular spot for increasing socialization.

V. Conclusions

It is common opinion that the IoT will increase its size by orders of magnitude in the next years and then it is necessary to find new strategies to make the objects communicate efficiently. Several independent papers investigated the benefits of the integration between IoT with social networking concepts, which leads to the new paradigm named the Social Internet of Things. In this paper, we briefly present some of the main platforms among the IoT implementations, and identify their common characteristics. Then, we propose the first, to the best of our knowledge, implementation of an experimental platform for the Social Internet of Things. The main innovations, with respect to the others IoT platforms, are the possibility for the objects to create their own relationships, based on the rules set by their owners, and to create groups of interest as it happens in human social networks. We also introduced some prototype applications, currently under development at the University of Cagliari. We are now planning to add more functionalities to our implementation, e.g. the possibility to assign different reading permission based on the relationship, the discovery of complex services and a system to evaluate the trustworthiness of the service received. In parallel, we are carrying out the development of the applications that implement the presented use-cases.

VI. Acknowledgements

This work has been partially funded by the project Artemis JU Demanes, Design, Monitoring and Operation of Adaptive Networked Embedded Systems, grant agreement no. 295372.

References

[1] P. Mendes, "Social-driven internet of connected objects," in *Proc. of the Interc. Smart Objects with the Internet Workshop*, 25th March 2011.
[2] L. Ding, P. Shi, and B. Liu, "The clustering of internet, internet of things and social network," in *Proc. of the 3rd Inter. Symp. on Knowl. Acquis. and Modeling*, 2010.
[3] D. Guinard, M. Fischer, and V. Trifa, in *PERCOM Workshops*, 29 2010-april 2 2010, pp. 702 –707.
[4] E. Kosmatos, N. D. Tselikas, and A. C. Boucouvalas, "Integrating rfids and smart objects into a unified internet of things architecture," *Advances in Internet of Things*, vol. 1, no. 1, pp. 5–12, 2011.
[5] L. Atzori, A. Iera, and G. Morabito, "Siot: Giving a social structure to the internet of things," *Communications Letters, IEEE*, vol. 15, no. 11, pp. 1193 –1195, november 2011.
[6] (2013) Cosm. [Online]. Available: http://cosm.com
[7] (2013) Iobridge. [Online]. Available: http://www.iobridge.com
[8] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization," *Computer Networks*, 2012.
[9] (2013) Postscapes. [Online]. Available: http://www.postscapes.com
[10] (2013) Nimbits. [Online]. Available: http://www.nimbits.com
[11] (2013) Paraimpu. [Online]. Available: http://www.paraimpu.com
[12] A. Piras, D. Carboni, and A. Pintus, "A platform to collect, manage and share heterogeneous sensor data," in *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, june 2012, pp. 1 –2.
[13] (2013) Thingspeak. [Online]. Available: http://www.thingspeak.com
[14] A. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi, "Architecture and protocols for the internet of things: A case study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 29 2010-april 2 2010, pp. 678 –683.
[15] D. Guinard, I. Ion, and S. Mayer, "In search of an internet of things service architecture: Rest or ws-*? a developers perspective," *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 326–337, 2012.
[16] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 253–266. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460438
[17] M. Nitti, R. Girau, L. Atzori, A. Iera, and G. Morabito, "A subjective model for trustworthiness evaluation in the social internet of things," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, sept. 2012, pp. 18 –23.