Discovery and Mash-up of Physical Resources through a Web of Things Architecture

Luca Mainetti, Vincenzo Mighali, Luigi Patrono, and Piercosimo Rametta

***Abstract*— The Internet of Things has focused on new systems, the so-called *smart things*, to integrate the physical world with the virtual world by exploiting the network architecture of the Internet. However, defining applications on top of *smart things* is mainly reserved to system experts, since it requires a thorough knowledge of hardware platforms and some specific programming languages. Furthermore, a common infrastructure to publish and share resource information is also needed. In this paper, we propose a software architecture that simplifies the visual development and execution of mash-up applications based on *smart things*, exploiting Internet Web protocols and their ubiquitous availability even on constrained devices. We have developed a distributed architecture that allows to create and control mash-up applications in an easy and scalable way, without specific knowledge on both hardware and programming languages. In addition, we have also defined a centralized public database deployed on the Internet, to manage and share physical resource information. The effectiveness of the proposed framework has been tested through a real use case and experimental results have demonstrated the validity of the whole system.**

***Index Terms*—CoAP, Mash-up, REST, WoT, WSN, Resource discovery.**

I. Introduction

THE recent technological innovations in the field of microelectronics and microcontrollers have led to the wide dissemination of the so-called *smart things*, that is, physical devices with sensing and actuating capabilities, but low computational and energy power. Generally, a quite large number of these devices are deployed in the real world and interconnected to create a wide range of applications for environmental monitoring, smart cities, home and building automation [1, 2], etc. In this perspective, some emerging technologies such as Radio Frequency Identification (RFID) [3] and constrained networks, first of all Wireless Sensor Networks (WSNs), are rapidly asserting as the most important type of distributed pervasive systems [4]. In order to improve the communication among these devices, several low-power protocols have been developed, such as Bluetooth, ZigBee, IEEE 802.15.4 [5], and more recently, 6LoWPAN [6] and Constrained Application Protocol (CoAP) [7]. However, despite all efforts made to integrate smart things at the network layer, embedded devices still form small and separated islands for the application layer.

Manuscript received March 28, 2014; revised May 26, 2014.

L. Mainetti, V. Mighali, L. Patrono and P. Rametta are with the Department of Innovation Engineering, University of Salento, Lecce 73100, Italy (e-mail: {luca.mainetti, vincenzo.mighali, luigi.patrono, piercosimo.rametta}@unisalento.it).

In this context, the Internet architecture appears to be the best model to adopt in order to integrate *smart things* into the digital world, leading to the well-known Internet of Things (IoT) concept. Furthermore, at a higher level, the Web is an example of how a set of open and relatively simple protocols and languages (e.g., HTTP, HTML, XML, etc.) can be used to implement flexible systems while ensuring efficiency, scalability, and ubiquitous availability on a wide variety of devices, included constrained devices. These considerations have led the IoT concept towards the *Web of Things* (WoT) concept [8]. According to it, all the traditional paradigms of the Web are adapted to integrate *smart things* not only at the network layer, but also into the Web (i.e., at the application layer). More in detail, adopting the WoT concept, the data produced by smart devices should be directly accessible as normal Web resources, so as providing the so-called *physical mash-up* [9].

Despite these interesting perspectives, the use of *smart things* in physical mash-up is hindered by some drawbacks, mainly related to the power consumption of nodes and to the difficulty in programming the network. While for the first aspect there are many solutions in the literature focused on minimizing the power consumption at various levels of the protocol stack [10, 11, 12, 13], for the second problem, a dominating solution has not yet emerged.

Firstly, defining capabilities and resources on each physical device is not a trivial activity, since it implies a deep knowledge of hardware platforms and embedded operating systems. Secondly, developing end-user applications requires the knowledge of some programming languages. Thirdly, it is needed to associate some contextual information to physical resources, so that any kind of user can easily manage, publish and discover them regardless devices heterogeneity. Finally, resource availability must be consistent with the execution state of the devices on which they are defined. Several solutions exist in literature that address separately some of the above outlined aspects, but none of them proposes a single software architecture for applications development/control and for resources management, fully integrated in the Web of Things.

In this work, a distributed Web of Things architecture has been defined and validated (i) to support the development of applications interacting with constrained devices and (ii) to manage private and public resources information.

The overall architecture consists of two main components: the *Mash-up Definition Execution and Control (MaDEC) Platform* and the *ResourceBase*. The first component is a local

1845-6421/06/8340

framework for the definition, execution and control of applications dealing with physical resources. Through its layered architecture, the *MaDEC Platform* allows resources definition on constrained devices and execution of mash-up applications interacting with them. Moreover, it provides a visual Integrated Development Environment (IDE) to graphically define applications business logic and to control their execution through visual dashboards. The IDE also includes a Web interface to manage local resources, associating some functional and/or spatial information to them, and to discover public resources according to a set of searching criteria.

The second component of the architecture, the *ResourceBase*, is a centralized database deployed on the Internet, where users can store all information regarding their private resources. By setting a particular flag in the resource description, owner can tag the resource as public, thus making it usable by another user on the Internet.

The rest of the paper is organized as follows. Section II summarizes the state-of-the-art related to solutions for implementation of mash-up applications and for resource discovery. Section III defines the main requirements that an optimal WoT architecture should guarantee. An overview on the proposed solution is presented in Section IV, whereas Section V provides an overview about the starting technologies. The detailed description of the proposed architecture is given in Section VI, and in Section VII a case study for evaluating the effectiveness of the proposed solution is described. Conclusions and open issues are drawn in Section VIII.

II. Related Works

Wireless Sensor and Actuator Networks are the core of many recent IoT applications because of their ubiquity and growing diffusion. However, the complexity in developing applications for these platforms has been a key issue over the last decade. In the recent years, several research works have exploited the advent of IP technologies for WSNs [14], and they have taken advantage of the development of very small-footprint Web servers implementing full HTTP stack [15]. So, these approaches are based on the Representational State Transfer (REST) architectural paradigm [16], which allows the manipulation of network resources by means of the basic HTTP methods. An important concept in the REST architecture is the possibility to access the information sources via a Uniform Resource Identifier (URI). Sensor.Network [17] is an example of the use of the REST paradigm for storing, sharing, searching, and viewing data coming from heterogeneous devices. A more intuitive tool is known as WoTkit [18], a toolkit for the Web of Things. It is a Java Web application that abstracts both sensors and actuators through a single physical model consisting of a virtual sensor to which the data coming from physical devices are associated.

In [19], the authors discuss, by illustrating two concrete implementations, how the REST principles can be applied to embedded constrained devices. Then, they show how RESTful interactions can be leveraged to quickly create new mash-up applications that integrate physical and virtual world. In [20], authors describe their idea of Web of Things architecture and the best-practices based on the RESTful principles.

Although the architectures and the platforms described so far are very promising, they implement their RESTful interfaces using the HTTP protocol at the application level. It exploits Transmission Control Protocol (TCP) as transport protocol, which can be quite inadequate for constrained devices, due to computational and energy requirements and the lack of a native server-push mechanism.

An interesting approach to address the HTTP issue is represented by the use of CoAP. Indeed, in addition to the HTTP features, it offers a built-in mechanism for the resources discovery, supports the IP multicast, and natively provides a server-push model and an asynchronous exchange of messages. It also has a small-size header and it bases the communication on the User Datagram Protocol (UDP) as transport protocol. A recent solution regarding the development of IoT applications based on CoAP is reported in [21, 22]. A key aspect of the proposed framework is the Thin Server [23], a light CoAP server installed on the physical devices that exposes hardware capabilities through a RESTful interface. Another fundamental building block is the application server Actinium, which allows the execution of WSN-based applications written in JavaScript.

Resource discovery, i.e. finding suitable local/remote resources, is another important topic in this context. The greatest part of the published works addressing this issue basically belong to two categories: those that implement this feature at device level, and those that provide a middleware dealing with resources/data description and discovery.

According to the taxonomy depicted in [24], IP-based solutions belong to the first category. They try to adapt well-known Internet discovery mechanisms, like Domain Name System (DNS), to constrained networks, or they exploit protocols specifically designed for such networks, like CoAP. Both solutions can operate in a centralized fashion (CoAP-RD [25] and DNS-SD [26]), or in a distributed way (CoAP Discovery and Multicast-DNS [27]) with no special nodes. Jara et al., in [28], propose a lightweight multicast DNS Service Directory, i.e. a set of implementation guidelines and design recommendations in order to make suitable the use of mDNS and DNS-SD in *smart things*. TRENDY [29], instead, is a CoAP registry-based service discovery protocol with context awareness, provided with an adaptive timer and a grouping mechanism to minimize control overhead and energy consumption. The self-configuration mechanism, proposed in [30], combines CoAP with DNS in order to address constrained nodes with user-friendly fully qualified domain names. This makes resources globally discoverable and accessible from any Internet-connected client, by means of IPv6 addresses or DNS names. Gramegna et al. take a step further and in [31] propose some CoAP extensions to allow semantic annotation of resources with respect to a reference ontology, in order to provide resources discovery and ranking based on non-standard inferences. All of these works, despite their efficacy, require that all nodes in the networks are aware

of the discovery mechanisms, so this implies computational overhead on devices and additional communications among them.

An alternative approach is to build a middleware-based discovery architecture, where resource owners can actively decide which resources can be shared with others. Sensor-Cloud Infrastructure [32] is one of the first examples of Cloud platform where owners can register their physical sensors and manage them through a virtual sensor model. Other users can access real data provided by virtual sensors for monitoring applications. IoTMaaS [33] is a Cloud-based middleware that proposes a new mash-up service model, which composes thing models, software models, and computation resource models. During mash-up process, these three components can be customized depending on end-users preferences. In [34], a layered middleware architecture, along with a method for efficient device interoperation through the generation of a generic device attributes structure, is presented. It proposes an algorithm to create a hierarchical device attributes structure and to form device clusters. This way, faster device interoperation, efficient sensor discovery, management and posting of sensed data can be achieved.

Other kinds of middleware architectures, in addition to discovery capabilities, also offer the possibility to store data generated by sensors in distributed or centralized data-stores, in order to make it available for any application. Rao et al. [35] propose a middleware in which owners expose their devices in a catalogue and through which the captured data is periodically stored in the cloud. Remote users can access this data simply by selecting the appropriate components in the catalogue. The drawback of such solution is the big amount of data generated by physical devices, and thus the needs of vast storage. Moreover, not all the data stored may actually be required by applications, with the effect of unnecessary communications between devices and wasted memory on data-stores.

III. Key Features of a WoT Architecture

The development of IoT applications on top of constrained devices and embedded operating systems requires specific skills, thus, it is only accessible to experts in embedded systems. Moreover, the management of local resources and the discovery of remote resources, used in these applications, are not still integrated in a complete WoT architecture, as seen in the Section II. For this reason, the overall goal of a WoT architecture should be to facilitate the development of IoT applications, regardless of both the target physical technology and the location of the physical resources. In this perspective, the main guidelines in the design of a WoT architecture should be:

- Low entry barrier for developers. This would allow a wider range of developers, tech-savvy users and ordinary end-users to foster rapid prototyping using *smart things.*
- High usability and ubiquity. Users should be able to access and use *smart things* data and services from everywhere, by using different kind of platforms and systems.
- Capability to manage and share resources. The architecture should allow users to associate additional contextual information to their local resources and to share them with other consumers on the Internet. It should also allow to discover public resources shared by remote users.
- Lightweight access to smart things data. This allows creating applications in which real-world data are directly consumed by resource-constrained devices, such as mobile phones, without requiring high computational and storing resources.
- Remote control of the embedded devices. The architecture should provide a bidirectional communication channel through which resources can be queried to retrieve information, but also actuated to change their state.
- Low computational load to meet poor computational and memory capabilities of the embedded devices. The fulfillment of these principles has led to several design choices. The most important ones are the following:
- The embedded devices must be kept free of any business logic. They only have to expose their resources through a common interface that abstracts hardware heterogeneities. This way, embedded devices act as small servers, exposing their physical resources as normal web resources.
- The architecture core must be in charge of resources discovery, indexing and sharing. In particular, the management mechanism has to allow resource owners to enrich their local resources with useful contextual information, in order to ease the indexing and discovering procedures.
- The architecture core must be also responsible of the business logic execution. Mash-up applications should run into dedicated application servers, which should take care of data processing, communication with physical resources and delivery of results to user interfaces.
- The mash-up functionalities of the architecture should be supported by simple APIs. This would allow developers to quickly deploy their applications, since they could be able to interact with physical devices through high-level methods. Also graphical editors could be exploited to visually implement and interact with mash-up applications.

IV. The Proposed WoT Architecture

The proposed WoT architecture is composed by two macro-components: one deals with applications development and execution, and the other deals with resources information storage.

The *Mash-up Definition Execution and Control (MaDEC) Platform* consists of a four-layered architecture, as shown in Fig. 1.

At the highest level, the closest to the user, there is the composition layer, which allows application development by means of visual programming. This layer provides the user with some graphical blocks that model the embedded devices

of a smart network and it allows, at run time, to interact with the running applications through some dashboards.

It also provides the user with a Web interface to manage local resources, associating some functional and/or spatial information to them, and to discover public resources according to some searching criteria. Then, at the next layer, a 2-ways Proxy Server has been defined in order to enable the communication between the development environment (in particular, the applications dashboards), the running applications and the *ResourceBase*. Further down, the architecture deals with monitoring the connectivity of embedded devices and with execution of mash-up applications: an application server is devoted to these tasks. At the lowest level, a CoAP-based RESTful Thin Server allows physical devices to expose their resources for the WoT applications. This framework is deployed on a local gateway and connected, on the one hand, directly with the WSN and, on the other hand, with the Internet through a Local Area Network (LAN).

In order to interact with a CoAP resource, whether local or remote, it is sufficient to know only the URI of the resource itself. However, a URI is often poorly significant from a user point of view, since s/he may want to evaluate some other information related to resources, like spatial or contextual information, in order to choose which resource best fits her/his application requirements. For this reason, the second component of the proposed architecture has been introduced.

The *ResourceBase* is a centralized database deployed on the Internet, which is used to store all contextual information associated to private resources, such as type and/or location, in addition to the URI. Owner can tag a given resource as public, by setting a particular flag in the resource description; this makes it usable by another consumer on the Internet, who handles in his/her application the data coming directly from the resource. It is worth noting that this architecture does not deal with storing data generated by physical devices, but it only stores metadata in order to ease remote resource discovery.

The Fig. 2, illustrates how several instances of the *MaDEC Platform* are interconnected with the *ResourceBase*, so realizing a distributed architecture for discovering and mashing-up of physical resources.

V. Underlying Technologies

The starting point of the proposed WoT architecture is represented by three main components that are not currently compatible with each other, but whose features, properly combined, are suitable to meet the requirements described in the Section III.

A. Thin Server architecture

Thin Servers provide a low-level API that wraps the elementary functionalities of devices, so allowing accessing and configuring the device parameters to interact with the physical world. This approach enables multiple concurrent applications to interact with physical devices without the need to reprogram IoT nodes for each application. One of the most promising Thin Server implementations is the Erbium REST Engine [36], which includes a comprehensive embedded CoAP implementation for the Contiki operating system [37], providing application level interoperability through a RESTful interface. Erbium defines three kinds of RESTful resources (namely *RESOURCE*, *EVENT_RESOURCE* and *PERIODIC_RESOURCE*) and natively implements *publish/subscribe* mechanism that, for the PERIODIC_RESOURCE, automatically stores subscriptions and then notifies the subscribers if a status change occurs. Finally, in order to support the resource discovery, Erbium creates an interface that returns a list of links, compliant with the *CoRE Link Format* [38] (the *./well-known/core* resource), about resources hosted by that server.

B. Actinium Server

Actinium [21] is a novel RESTful runtime container for scripted IoT applications and it is based on Californium, a modular Java-based CoAP implementation. Each running application, called Actinium App, is modeled as a resource, which can be installed, updated, and removed exploiting a RESTful API. The running scripts are written in JavaScript language, which has been enhanced by adding an interface to directly communicate with CoAP resources.

Despite Actinium seems to be extremely suitable in the IoT context, it has some shortcomings that make it currently immature to be used in a WoT architecture. The first issue to address is the absence of a discovery mechanism, which is needed by the application server in order to detect the local

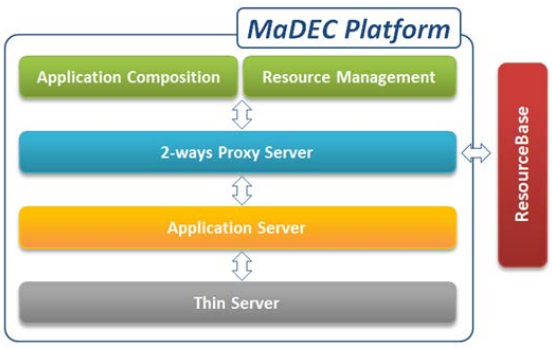


Fig. 1. Layered structure of the *MaDEC Platform*

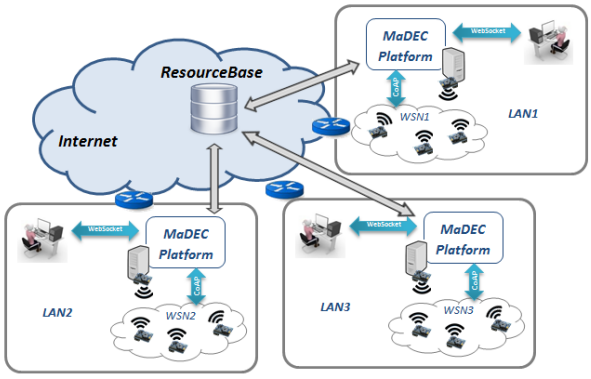


Fig. 2 Several *MaDEC Platform* instances connected to the ResourceBase

WSN devices and their resources. Moreover, Actinium currently lacks a specific mechanism for the management of the server-push mode, i.e., the ability of a physical resource to send notifications upon the occurrence of certain events.

C. The ClickScript Editor

The ClickScript project [39] defines an interpreted visual programming language, based on JavaScript and provided by means of a Web application, which allows non-expert users to graphically implement applications that run in a browser. The user can easily extend the language by defining new components and s/he can control the execution of each script through graphical dashboards provided by the integrated development environment. ClickScript is a data-flow-based interpreted programming language, in which each instruction is evaluated at runtime: once all components of an application are placed in the programming area and are interconnected according to the business logic, each of them is executed only when all input data are available.

ClickScript has no knowledge of CoAP and it is currently not able to communicate with Actinium, since its native purpose is far from IoT context. In addition, as said before, it is designed so that its applications run only locally, within the browser, and one at a time.

VI. Implementation Details Of The Proposed Architecture

The technologies described in the previous section are the basis on which the proposed architecture was designed. This section explains how the above components have been adapted and enhanced to realize the *MaDEC Platform* and the *ResourceBase*.

A. RESTful interface for physical devices

The Erbium implementation was adapted to the available hardware. According to the Thin Server paradigm, in order to read the value of the sensors, each of them was registered in Erbium as a resource and a proper handler was defined for each sensor. Upon receipt of a GET request coming from client applications, the handler polls the sensor and builds the response message using the sensor state as payload. The same approach was used for the actuators, but, in this case, the defined handlers are able to respond also to POST requests sent by client applications to change actuator state. Finally, in order to define an observable sensor on Erbium, an *EVENT_RESOURCE* was implemented. At each occurrence of the desired event, it exploits a proper handler to retrieve the actual resource state, create the update message, and notify all the observers.

B. MaDEC Platform core

The execution and control of mash-up applications are the core features of the *MaDEC Platform* and they are mainly accomplished by the Actinium application server. For these purposes, Actinium was properly enhanced in order to create an environment where application can interact with CoAP resources placed within the local WSN or distributed over the Internet. The *MaDEC Platform*, also plays an important role in local resource discovery and in the process of resource availability updating. The architecture is depicted in Fig. 3.

The local discovery procedure is performed by a JavaScript application, called *discover-motes*, which runs at the Actinium startup and stores the available resources in a resource directory. In particular, it retrieves, from the border-router, the IP addresses of the connected WSN devices and, for each of them, it installs on Actinium a clone-application that acts as a "proxy" for the corresponding embedded device. More in detail, each proxy application has the same resources of the corresponding node (sensors, actuators, etc.), so as to realize a mapping of physical resources on the Actinium resources. Each client application sends its requests to the proxy applications and not directly to the embedded devices. Then, the proxy applications have the burden to forward the requests to physical devices, in order to get the responses and to deliver them to the client applications. The advantage of using the proxy applications is clear in a subscription/notification scenario, where multiple client applications can subscribe to the same observable resource, without affecting the limited capabilities of the physical devices. The *discover-motes* application is also responsible for motes availability check. It periodically checks each IP address associated to the border-router by sending a GET request to the *.well-known/core* resource. If no answer is received within a predefined timeout, the discovery application supposes that the mote is no more available, maybe due to battery depletion or to shut down by owner. In this case, the related proxy application is shut down and deleted from the application server.

Finally, the structure of a generic Actinium IoT application, called Actinium App, was reorganized in a more appropriate way, in order to drive the mapping algorithm explained in section VI.E. More in detail, the listing of the application consists of the following sections:

- *CODE*: it contains the functional code of the objects and the input/output sub-resources used in the application;
- *INSTANCES*: it contains the definition of the object

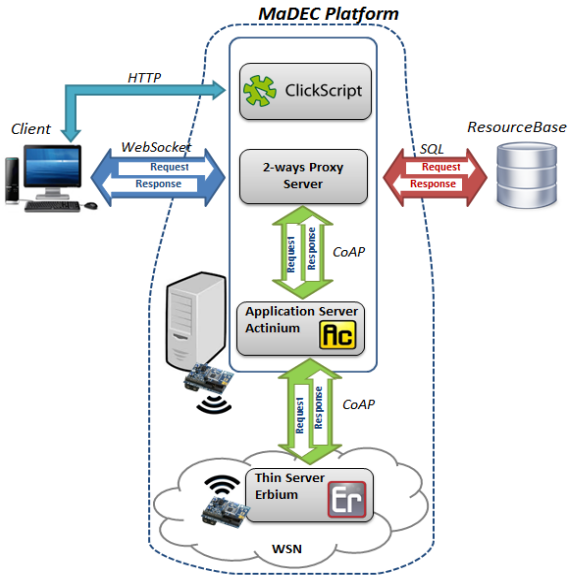


Fig. 3. *MaDEC Platform* implementation

instances used in the application;

- *STATE*: it defines the *Status* object, which aggregates the status of each input/output component and sends it to the application GUI;
- *BODY*: it contains the application logic instructions and the execution cycle;
- *APP-ROOT*: it includes the handlers of the global application resources.

C. 2-ways Proxy Server

The 2-ways Proxy Server is the middle component of the whole architecture, since it enables communications between the *MaDEC Platform* components and the *ResourceBase*. It has the burden of translating requests coming from visual dashboards into CoAP messages for the Actinium applications and vice versa. It also translates the operations regarding the resource management and discovery, made through the graphical user interface, into SQL queries understandable by the ResourceBase. The Fig. 4 summarizes the proxy features.

The proxy server is fully based on the JavaScript language, exploiting the capabilities of the Node.js framework [40]. To implement the CoAP side, an open-source CoAP plug-in for Firefox, called Copper [41], was analyzed and suitably adapted for our purposes. Particular attention was paid to the implementation of both the *blockwise* messages and the simultaneous observation of multiple observable resources.

Regarding the WebSocket side, it was realized starting from an open-source implementation of the WebSocket protocol [42] for the Node.js framework. The translation between the two sides of the proxy is carried out by exploiting specific numerical codes included in the WebSocket request, each of which corresponds to a specific command of the CoAP interface of the proxy. Once the WebSocket request is received, the proper CoAP method is invoked and the CoAP message is created basing on the other parameters of the WebSocket request. Then, the message is sent to the CoAP server Actinium through an UDP socket. In the opposite direction, once the CoAP side of the proxy server receives the response coming from Actinium, it uses this message to create the payload of the response for the WebSocket client. In this way, the CoAP communication is completely transparent to the ClickScript client.

The same pattern was followed for the SQL side of the proxy. It is based on an open-source implementation of the MySQL client [43] for Node.js and, once the WebSocket request is received, the payload is translated into the proper SQL query, and sent to the *ResourceBase* through a previously instantiated TCP connection. This allows any CRUD operation on the *ResourceBase*. Query results follow the opposite direction, flowing from the *ResourceBase* back to the client through the WebSocket channel. The "*Availability Observer*" module, shown in Fig. 4, is the central component of the resource availability updating mechanism and its functionalities are explained in detail in the next section.

Finally, the proxy also provides storing capabilities, allowing to store ClickScript files and graphical user interfaces on the gateway file system.

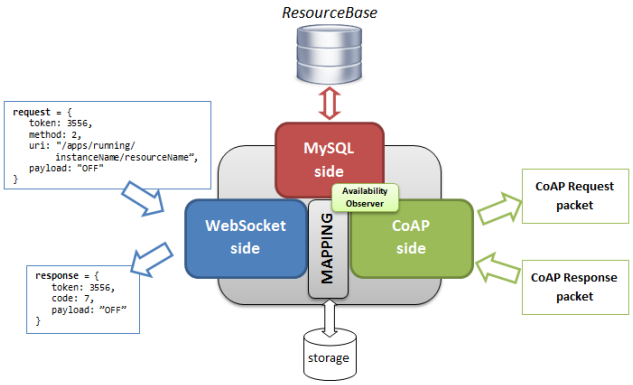


Fig. 4. 2-ways Proxy server

D. The ResourceBase

The *ResourceBase* is a centralized MySQL database, with a well-known public address, that stores all information related to sensors and actuators owned by each WSN owner. It contains two tables (one for Sensors and one for Actuators), each of which consists of a number of fields regarding the characteristics of each resource (for example name, type, owner, if it is observable and/or public, etc.), its location (absolute and logical), its URI and its running state. The *public* field is crucial in this context; if it is set to *yes*, it states that the given private resource becomes publically available and any remote user can use it in his/her applications. The *running* field also has an important role. If it is set to *true*, it means that the resource is up and running, whereas, if it is set to *false*, it means that the resource is not available at the moment, maybe because the physical device is off, or the corresponding board proxy application is not running.

Since the *ResourceBase* does not have the burden to store the data produced or consumed by physical devices, a centralized solution is suitable enough to handle all resource metadata and to guarantee scalability and robustness. However, if the number of stored resource were to grow disproportionately, the theory of distribute databases would help us by managing the tables on physically distributed database in a transparent way respect to user.

The interaction with the *ResourceBase* can be mainly divided into three phases, as explained below.

D.1 Resource registration and management

This phase takes care of creating a record in the *ResourceBase* for each resource defined in the local WSN. After the local discovery procedure accomplished by the *discover-motes* application, the first time a proxy application associated to the resource is executed, a new line is automatically inserted into the proper table of the *ResourceBase*. At this point, only some fields in the table are set: the *URI* field is set with the full path of the resource (that acts as an index, since it is globally unique) and the *public* field is set to *false* (default for privacy purposes). At a later moment, the resource owner can use the ClickScript "Management" interface to add or update all the remaining

information in the resource record, including the value of the *public* field. S/he can also manually insert a new record independently from the automatic mechanism. It is worth noting that resource owners can only manage the resources they own, whether local or public.

D.2 Resource availability updating

An automatic mechanism was implemented to promptly update the availability state of a public resource. This functionality is carried out cooperatively by several components of the whole architecture, as illustrated in Fig. 5.

The first component is the "*notify*" function in each board proxy application. This function, when invoked, sends a notification to the "*Availability Relay*" resource (discussed later), containing the list of resources defined on the board. When the "*notify*" function is invoked at proxy application start up (case a)), it sends also a string "*true*" in the notification message, to notify that all the resources are available and running. Instead, when the application is unloaded (case b)), the "*notify*" function sends a string "*false*", meaning that above resources are no longer available.

The second block is the "*Availability Relay*" resource. This is a double-sided RESTful CoAP observable interface (running on the Actinium server), which the 2-ways Proxy Server subscribes to at start up. The "*Availability Relay*" acts as a relay that gathers availability notifications from board proxy applications and passes them to the proxy server, in a CoAP request, as soon as they arrive.

The third component of the mechanism is the "*Availability Observer*" module of the 2-ways Proxy Server. It subscribes the proxy to the "*Availability Relay*" resource of the Actinium server, and includes a callback function that is executed when availability notifications arrive. For each resource contained in the notification, the suitable SQL query is composed. Then it is sent to the *ResourceBase*, by exploiting the MySQL side of the 2-ways Proxy Server. The resource URI is used as index in the database tables.

It is worth noting that the resource registration procedure explained in Section D.1, partially overlaps with the resource availability updating phase. In fact, the "*Availability Observer*" module can discriminate if it has to send an INSERT query to the *ResourceBase* (registration phase) or an UPDATE query (availability updating phase). For the sake of space, Fig. 5 shows the case of availability updating upon resource registration. With this mechanism, every time a resource is started or stopped, either by user action or due to board (dis)connections, its availability state is immediately reported to the *ResourceBase*. This way, when remote users browse the public resources, they can be sure that the value of the running field reflects the real execution state of the resource. If a resource is tagged as not running at searching time, it should not be instantiated in any application. If a resource used in some application becomes unavailable at run-time, instead, it should be care of the application itself to handle this situation and properly alert the user.

D.3 Public resource discovery

Once a private resource is declared as public, it is publically available on the Internet and any remote user can exploit it in any application. This is done by simply searching it through the "Discovery" Tab in the local ClickScript interface. This tool allows the user to perform a string-matching query on the *ResourceBase* by setting the proper fields value in the form. Query results are visualized in a table format, in which only publically available resources are shown. The interaction between user and ResourceBase is handled by the Clickscript GUI and the 2-ways Proxy Server.

E. Graphical interface for discovery and mash-up

The original architecture of the ClickScript application has been widely extended in order to adapt it to the WoT context. First of all, the execution of the applications has been transferred on the Actinium server and it is controlled by means of graphical dashboards. All interactions between ClickScript and proxy server are done through a WebSocket

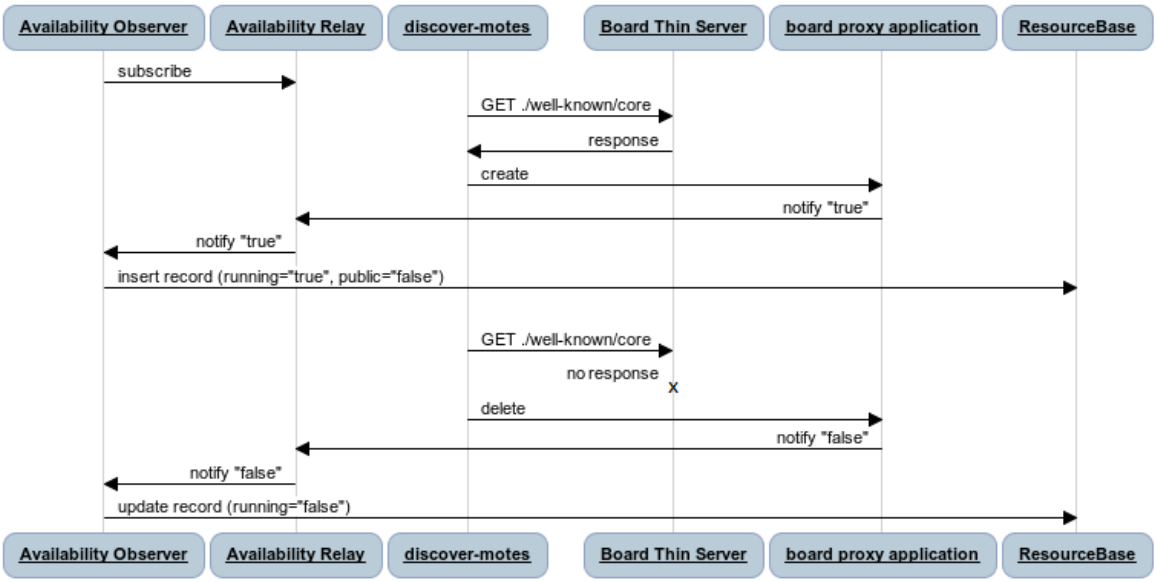


Fig. 5. Sequence diagram of the resource availability updating process

channel.

The new architecture is shown in Fig. 6. The first three layers, on the left side, are used in the Programming phase. The Library Layer defines the graphical and functional structure of each ClickScript visual component, in terms of number of inputs, outputs and configuration fields. The Data Model Layer handles the creation of the application control flow, concerning the data and control dependencies among involved components. Then, the Programming Layer deals with user interaction to visually define the structure of the application and its dashboard. The Mapping Layer is in charge of the implementation of the mapping algorithm, which translates the visual script into a single JavaScript file, having the structure introduced in Section VI.B. The GUI Layer handles user interaction with the graphical interface during the Execution phase. It translates user actions on application dashboard into messages for the proxy server, and, in the opposite direction, it displays messages coming from the server in the proper dashboards. Finally, the Communication Layer implements a WebSocket client to interact with the proxy server.

The new architecture has implied changes to the ClickScript user interface. The Programming View now includes a dedicated area for the dashboard associated to the application, and a sidebar useful to install the script on Actinium and to select the available local resources. Two new tabs have been added (i) to run an application already installed on Actinium (Instantiating View), and (ii) to graphically control one or more applications through their dashboards (Execution View). Moreover, one new tab has been added to manage local resources (Management View): through several sub-tabs and forms, a WSN owner can insert/search/update/delete information related to his/her resources stored in the *ResourceBase*. Finally, another tab (Discovery View) allows string-matching queries in the *ResourceBase* to discover any remote resource that best fits the application requirements.

New ClickScript components have been defined to allow IoT applications implementation. Sensor, Observable Sensor and Actuator components are used to model the physical resources and to define their graphical behavior at runtime.

The mapping algorithm (Fig. 7) is a key component of this

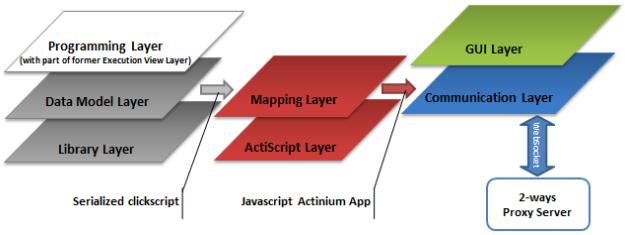


Fig. 6. The new ClickScript architecture

new version of ClickScript. It translates the data-flow of the visual application, defined within the Programming View, into a sequential JavaScript file structured as an Actinium App. The algorithm consists of three phases: parsing, component mapping, and merging. In the parsing phase, it takes an XML-like serialized file containing the structure of the visual script

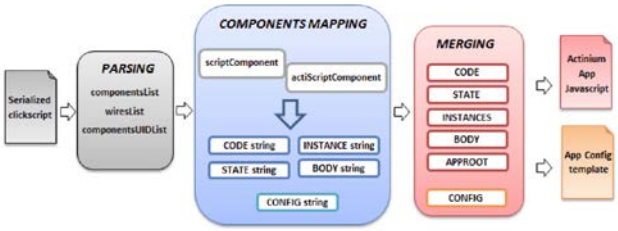


Fig. 7. The Mapping algorithm

and parses it in order to obtain a number of JavaScript objects (*scriptComponent*) that wrap all the structural dependencies of application components. During the component mapping phase, for each *scriptComponent*, the JavaScript lines of code associated to the component are generated. Finally, all code snippets generated in the previous step are merged together to obtain the Actinium App final listing (merging phase). The mapping algorithm also applies to the creation of the Configuration File template associated to the application.

VII. A Proof-of-Concept

To validate the whole architecture, a simple temperature controller application has been implemented. To better explain

the resource discovery mechanism, we suppose that the scenario, modeled in Fig. 8, consists of a campus with several buildings. Each building has its own independent Wireless Sensor Network, reachable from the Internet through a local gateway. In particular, the buildingB WSN contains an external temperature sensor and the buildingA WSN contains the other needed devices, i.e. an internal temperature sensor and an ON/OFF actuator node that directly drives an air conditioner. Each device runs a Thin Server and communicates with the related network gateway. The overall goal of this application is to keep the difference between the external and internal temperature in a room above a user set threshold: if this threshold is exceeded, the air conditioner is activated until the set-point is reached.

The app runs in the Actinium server installed on the gateway and consists of a polling cycle that, at each iteration, reads the sensors values, computes the difference between the two readings and compares it with the user threshold. Based on the comparison result, the air conditioner is turned on or off by means of the actuator node.

To save the physical actuator from unnecessary

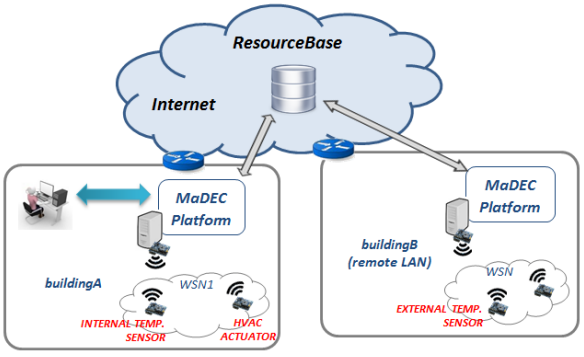


Fig. 8. Use case scenario

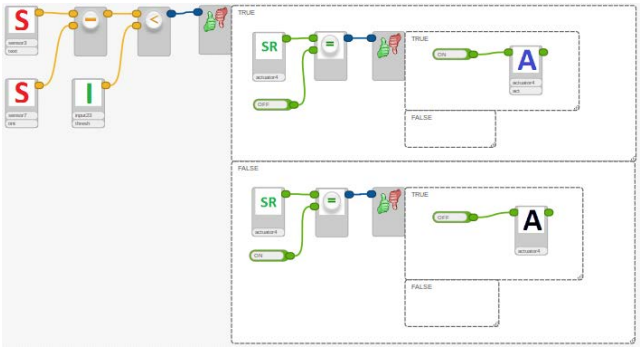


Fig. 9. Data-flow logic of the use case application

transmissions, its local application state is checked before acting on it. The pseudo-code of the application is the following.

```
period = 30 (seconds)
while (true) {
  read external temperature value Text
  read internal temperature value Tint
  read user threshold th
  compute Δ=Text-Tint
  if (Δ<th) {
    if (actuator = OFF) actuator = ON
  } else {
    if (actuator = ON) actuator = OFF
  }
  sleep(period)
}
```

As a preliminary step, we suppose that each WSN owner has already registered his/her resources, both private and public, to the *ResourceBase*, as explained in Section VI.D.

A user can visually create and control the above-mentioned application by using the ClickScript interface on his/her browser. S/he can create the control flow of the algorithm by selecting the proper components from the toolbar in the Programming View and connecting them according to their data dependencies (Fig. 9). While the script components are added to the programming area, the dashboard for controlling the application is created in the lower section of the Programming View. Once the application has been completely defined, it can be installed on the application server Actinium. More in detail, by clicking on the *Install on server* button, the mapping algorithm is started, and then the application JavaScript listing is sent to the server. At the same time, a

Fig. 11. Graphic control dashboard of a running application

template of the application Configuration File is also created and stored on the file-system of the proxy server, as well as the dashboard template. After these steps, only the application logic and the graphical dashboard are defined and stored, but no application is actually started.

In order to execute the application, an instance must be configured and started. These operations can be easily done in the Instantiating View by selecting the application name from the Installed Application drop-down menu and by filling in the configuration form. The local resource URIs can be selected from the Running Resources listed in the right menu, or by querying the *ResourceBase* through the Management View tab in ClickScript. For the remote resource URI, the Discovery View tab in ClickScript must be used. In this case, for example, user can search all running temperature sensors, situated outside the buildingB, by setting the discovery form as illustrated in Fig. 10. From the results table, user can copy the URI of the chosen resource and paste it into the configuration form.

To control the execution of an application instance, the user can simply select its name in the Available Applications list and click the *Load UI* button. This way, the application dashboard is loaded in the Execution View, so that the user can control the remote devices and automatically display all updating messages (Fig. 11).

With this new version of ClickScript, two or more applications can be controlled at the same time in the Execution View, managing them independently of one another. Through the *Start* and *Stop* buttons the user can control the execution of the application, whereas with the *Manual/Automatic Mode* buttons s/he can decide whether the status of the physical devices has to be controlled manually by the user or automatically by the business logic. Finally, through the *Close UI* button the user can close the dashboard in the Execution View without affecting the execution on server. Indeed, the application can run autonomously even if the user client is turned off. With the *Configure* button, the application parameters, such as the polling period or the URI

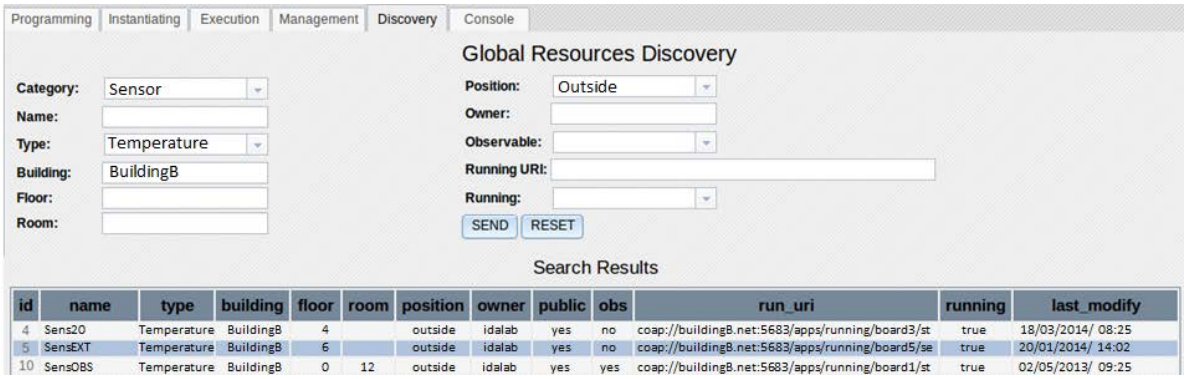


Fig. 10. External temperature sensor discovery query and results

of physical resources, can be modified at runtime.

VIII. CONCLUSIONS

In this work, a distributed software architecture to discover and mash-up physical resources in the Web of Things has been defined and validated. The reasons behind this choice are twofold: (i) to facilitate the management of the physical resources of an embedded network, both in a local and in a global scope, and (ii) to ease the creation of mash-up applications interacting with CoAP-based resources. With the proposed architecture, *smart things* owners can manage information related to their resources, eventually sharing them through a public centralized database. Another contribution of this work is a distributed platform for the definition, execution and control of mash-up applications based on CoAP resources. This platform allows, through visual programming, the implementation of mash-up applications for the WoT, without having knowledge of both the embedded hardware and specific programming languages. The proposed solution was validated, from a functional point of view, through a simple use case that represents only one area of applicability in which the architecture may be employed.

The peculiar characteristic of the proposed architecture compared to the other solutions already in the literature is mainly represented by the composition layer of the MaDEC Platform. Indeed, other architectures usually provide the user with the ability to exploit a set of applications already implemented or, however, they require high programming skills to implement new ones. Instead, with our approach, the graphical editor of the MaDEC Platform allows users to autonomously implement new IoT mash-up applications.

To extend the present work, it is in plan to enhance the resource description with a semantic annotation with respect to a reference ontology, in order to help users during the public resources discovery phase. This way, through a semantic reasoner, users can perform semantic queries to the ResourceBase, which can produce more accurate results than a string-matching query.

REFERENCES

[1] Mainetti, L., Patrono, L., Vergallo, R.: IDA-Pay: A secure and efficient micro-payment system based on Peer-to-Peer NFC technology for Android mobile devices, *Journal of Communications Software and Systems*, Volume 8, Issue 4, December 2012, Pages 117-125.

[2] De Luca, G., Lillo, P., Mainetti, L., Mighali, V., Patrono, L., Sergi, I.: The use of NFC and Android technologies to enable a KNX-based smart home, 2013 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2013, 2013, Article number 6671904, pp.1-7, 2013.

[3] Catarinucci, L., Colella, R., Mainetti, L., Patrono, L., Pieretti, S., Sergi, I., Tarricone, L.: Smart RFID antenna system for indoor tracking and behavior analysis of small animals in colony cages, *IEEE Sensors Journal*, Volume 14, Issue 4, April 2014, Article number 6678175, Pages 1198-1206

[4] L. Mainetti, L. Patrono, and A. Vilei: Evolution of wireless sensor networks towards the Internet of Things: a survey, 2011 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2011, pp. 16-21, 2011.

[5] IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC Sublayer, 802.15.4e, 2012.

[6] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC4944, 2007.

[7] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," draft-ietf-core-coap-18, 2013.

[8] D. Guinard, V. Trifa, and E. Wilde, "A Resource Oriented Architecture for the Web of Things," in *Proc. IoT*, Tokyo, Japan, 2010, pp. 1–8.

[9] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proc. DAC*, Anaheim, California, 2010, pp. 731–736.

[10] L. Anchora, A. Capone, V. Mighali, L. Patrono, and F. Simone, "A novel MAC scheduler to minimize the energy consumption in a Wireless Sensor Network," *Ad Hoc Networks*, vol. 16, pp. 88–104, 2014.

[11] L. Catarinucci, S. Guglielmi, L. Mainetti, V. Mighali, L. Patrono, M.L. Stefanizzi, and L. Tarricone, "An Energy-Efficient MAC Scheduler based on a Switched-Beam Antenna for Wireless Sensor Networks," *Journal of Communication Software and Systems*, vol. 9, no. 2, pp. 117–127, June, 2013.

[12] L. Catarinucci, R. Colella, G. Del Fiore, L. Mainetti, V. Mighali, L. Patrono, M.L. Stefanizzi, "A cross-layer approach to minimize the energy consumption in wireless sensor networks," *International Journal of Distributed Sensor Networks*, Volume 2014, Article number 268284, DOI: 10.1155/2014/268284.

[13] D. Alessandrelli, L. Mainetti, L. Patrono, G. Pellerano, M. Petracca, and M. L. Stefanizzi: Implementation and validation of an energy-efficient MAC scheduler for WSNs by a test bed approach, 2012 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2012, Article number 6347615, 2012.

[14] J.W. Hui and D.E. Culler, "IP is dead, long live IP for wireless sensor networks," in *Proc. SenSys 2008*, Raleigh, NC, USA, 2008, pages 15–28.

[15] S. Duquennoy, G. Grimaud, and J.J. Vandewalle, "The Web of Things: interconnecting devices with high usability and performance," in *Proc. ICESS 2009*, HangZhou, Zhejiang, China, 2009, pp. 323–330.

[16] L. Richardson and S. Ruby, "RESTful web services," O'Reilly Media, May 2007.

[17] V. Gupta, A. Poursohi, and P. Udupi, "Sensor.Network: An open data exchange for the web of things," in *PERCOM Workshops*, Mannheim, 2010, pp. 753–755.

[18] M. Blackstock and R.Lea, "WoTKit: a lightweight toolkit for the web of things," in *Proc. WoT 2012*, New York, NY, USA, 2012.

[19] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices," in *Proc. WWW 2009*, Madrid, Spain, 2009.

[20] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the Web of Things," in *Proc. IOT 2010*, Tokyo, 2010, pp. 1–8.

[21] M. Kovatsch, M. Lanter, and S. Duquennoy, "Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications," in *Proc. IoT 2012*, Wuxi, China, 2012, pp. 135–142.

[22] L. Mainetti, V. Mighali, L. Patrono, P. Rametta, S.L. Oliva: A novel architecture enabling the visual implementation of web of Things applications, 2013 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2013, pp.1-7, 2013.

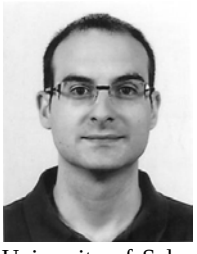
[23] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving Application Logic from the Firmware to the Cloud: Towards the

Thin Server Architecture for the Internet of Things," in *Proc. IMIS 2012*, Palermo, Italy, 2012, pp. 751–756.
[24] B.C. Villaverde, R. De Paz Alberola, A.J. Jara, S. Fedor, S.K. Das, and D. Pesch: "Service Discovery Protocols for Constrained Machine-to-Machine Communications," Communications Surveys & Tutorials, IEEE, vol.16, no.1, pp.41-60.
[25] Z. Shelby, C. Bormann, and S. Krco: "CORE Resource Directory" draft-ietf-core-resource-directory-01, 2014.
[26] S. Cheshire, and M. Krochmal: "DNS-Based Service Discovery", RFC 6763, 2013.
[27] S. Cheshire, and M. Krochmal: "Multicast DNS", RFC 6762, 2013.
[28] A.J. Jara, P. Martinez-Julia, and A. Skarmeta: "Light-Weight Multicast DNS and DNS-SD (lmDNS-SD): IPv6-Based Resource and Service Discovery for the Web of Things," in *Proc. IMIS 2012*, Palermo, Italy, 2012, pp.731-738.
[29] T.A. Butt, I. Phillips, L. Guan, and G. Oikonomou: "TRENDY: An Adaptive and Context-Aware Service Discovery Protocol for 6LoWPANs", in *Proc. WOT 2012*, Newcastle, UK, June 2012.
[30] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman, and P. Demeester: "Facilitating Sensor Deployment, Discovery and Resource Access Using Embedded Web Services," in *Proc. IMIS 2012*, Palermo, Italy, 2012, pp.717-724.
[31] F. Gramegna, S. Ieva, G. Loseto, and A. Pinto: "Semantic-enhanced resource discovery for CoAP-based sensor networks," in *Proc. IWASI 2013*, Bari, Italy, 2013, pp.233-238.
[32] M. Yuriyama, and T. Kushida: "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing," in *Proc. NBiS '10*, Takayama, Japan, 2010, pp.1-8.
[33] I. Janggwan, K. Seonghoon, and K. Daeyoung: "IoT Mashup as a Service: Cloud-Based Mashup Service for the Internet of Things," in *Proc SCC 2013*, Santa Clara Marriott, CA, USA, 2013, pp.462-469.
[34] S. Bandyopadhyay, and A. Bhattacharyya: "Architecture supporting discovery and management of heterogeneous sensor for smart system using generic middleware" International Journal of Computer Networks & Communications (IJCNC) Vol.4, No.5.
[35] B.B.P. Rao, P. Saluia, N. Sharma, A. Mittal, and S.V. Sharma: "Cloud computing for Internet of Things & sensing based applications," in *Proc. ICST 2012*, Kolkata, India, 2012, pp.374-380.
[36] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A Low-Power CoAP for Contiki," in *Proc. MASS 2011*, Valencia, Spain, 2011, pp. 855–860.
[37] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki – a lightweight and flexible operating system for tiny networked sensors," in *IEEE LCN 2004*, Florida, USA, 2004, pp. 455–462.
[38] Z. Shelby and Sensinode, "Constrained RESTful Environments (CoRE) Link Format," RFC 6690, August 2012.
[39] "ClickScript". Internet: http://clickscript.ch/site/home.php [Mar. 10, 2014].
[40] "Node.js". Internet: http://nodejs.org/ [Dec. 28, 2013].
[41] M. Kovatsch: "Demo abstract: Human-CoAP interaction with Copper," in *Proc. DCOSS 2011*, Barcelona, Spain, 2011 pp.1-2.
[42] "ws: a Node.js WebSocket library". Internet: https://github.com/einaros/ws.
[43] "node-mysql". Internet: https://github.com/felixge/node-mysql.

Luca Mainetti is an associate professor of software engineering and computer graphics at the University of Salento. His research interests include web design methodologies, notations and tools, services oriented architectures and IoT applications, and collaborative computer graphics. He is a scientific coordinator of the GSA Lab - Graphics and Software Architectures Lab and IDA Lab - IDentification Automation Lab at the Department of Innovation Engineering, University of Salento.

Vincenzo Mighali received the "Laurea" Degree in Computer Engineering with honors at the University of Salento, Lecce, Italy, in 2012. Since January 2009 he collaborates with IDA Lab — IDentification Automation Laboratory at the Department of Innovation Engineering, University of Salento. His activity is focused on the definition and implementation of new tracking system based on RFID technology and on the design and validation of innovative communication protocol aimed to reduce power consumption in Wireless Sensor Networks. He is also involved in the study of new solutions for building automation. He authored several papers on international journals and conferences.

Luigi Patrono received his MS in Computer Engineering from University of Lecce, Lecce, Italy, in 1999 and PhD in Innovative Materials and Technologies for Satellite Networks from ISUFI-University of Lecce, Lecce, Italy, in 2003. He is an Assistant Professor of Network Design at the University of Salento, Lecce, Italy. His research interests include RFID, EPCglobal, Internet of Things, Wireless Sensor Networks, and design and performance evaluation of protocols. He is Organizer Chair of the international Symposium on RFID Technologies and Internet of Things within the IEEE SoftCOM conference. He is author of about 80 scientific papers published on international journals and conferences and four chapters of books with international diffusion.



Piercosimo Rametta received the "Laurea" Degree in Computer Engineering with honors at the University of Salento, Lecce, Italy, in 2013. His thesis concerned the definition and implementation of a novel mash-up tool for Wireless Sensor Networks' configuration. Since November 2013 he collaborates with IDA Lab — IDentification Automation Laboratory at the Department of Innovation Engineering, University of Salento. His activity is focused on the definition and implementation of new mash-up tools for managing smart environments based on Wireless Sensor Networks and Internet of Things.