# Lysis: A Platform for IoT Distributed Applications Over Socially Connected Objects

Roberto Girau, Salvatore Martis, and Luigi Atzori, *Senior Member, IEEE*

***Abstract*—This paper presents Lysis, which is a cloud-based platform for the deployment of Internet of Things (IoT) applications. The major features that have been followed in its design are the following: each object is an autonomous social agent; the platform as a service (PaaS) model is fully exploited; reusability at different layers is considered; the data is under control of the users. The first feature has been introduced by adopting the social IoT concept, according to which objects are capable of establishing social relationships in an autonomous way with respect to their owners with the benefits of improving the network scalability and information discovery efficiency. The major components of PaaS services are used for an easy management and development of applications by both users and programmers. The reusability allows the programmers to generate templates of objects and services available to the whole Lysis community. The data generated by the devices is stored at the object owners cloud spaces. This paper also presents a use-case that illustrates the implementation choices and the use of the Lysis features.**

***Index Terms*—Cloud computing, Internet of Things (IoT), social Internet of Things (SIoT), social networks.**

## I. Introduction

SOCIETY is moving toward an "always connected" paradigm, where the Internet user is shifting from persons to things, leading to the so called Internet of Things (IoT). In this respect, successful solutions are expected to embody a huge number of smart objects identified by unique addressing schemes providing services to end-users through standard communication protocols. Accordingly, the huge numbers of objects connected to the Internet and that permeate the environment we live in, are expected to grow considerably, causing the production of an enormous amount of data that must be stored, processed and made available in a continuous, efficient, and easily interpretable manner. Cloud computing can provide the right technologies to implement the infrastructure that meets these requirements and can integrate sensors, data storage devices, analytic tools, artificial intelligence, and management platforms. Additionally, the pricing model on consumption of cloud computing, enables access to end-to-end services in an on-demand fashion and in any place. At the same time, service-oriented technologies, Web services, ontologies, and semantic Web allow for constructing virtual environments for application development and deployment [1].

In the last five years many IoT architectural proposals and implementations appeared in the literature and in the market. A great effort has been devoted to defining architectures and relevant functionalities which often rely on the concept of virtualizing the physical objects. Indeed, virtual objects (VOs) implement the digital counterparts of the physical devices (PDs), spoke for them and introduce some functionalities that could not be taken by the real world objects, such as: supporting the discovery and mash up of services, fostering the creation of complex applications, improving the objects energy management efficiency, as well as making inter-objects communications possible by translating the used dissimilar languages. Additionally, virtualization technologies can hide the physical characteristics of industrial equipment implementing an effective connection, communication, and control between the real world and the virtual counterpart. Some of the existing implementations have also been designed to exploit the cloud computing features. In this context, an important category is that of distributed cloud-based applications, where different components are executed in separate platforms, devices included. Indeed, the application level functions are assigned to different virtual and real components to reduce latency and bottlenecks. This is the case for instance of Xively [2] and Paraimpu [3], where the data sensed by the devices is processed locally and the results are sent to the cloud to trigger the execution of some centralized tasks. Something similar happens also in Carriots [4], where the user can also write some code with provided domain-specific languages that can be executed in the cloud when something is detected in the devices. Other platforms, such as iCore [5] and Compose [6], bring forward the concept of distributed applications: the codes to be executed in the cloud is distributed to different virtual entities at different levels. Some simple triggers are executed by simple virtual entities, the complex operations are assigned to aggregations of simple virtual entities, and operations of management functions are assigned to Web services. Usually, such virtual entities are managed by central elements running in the cloud.

As it is discussed in the following section, we still believe that to fully exploit the potentialities of the IoT paradigm, there is a strong need for further advancements in the design of

Manuscript received May 13, 2016; revised September 6, 2016; accepted September 30, 2016. Date of publication October 10, 2016; date of current version February 8, 2017. This work was supported in part by the Project SocialMobility, P.O.R. FESR 2007–2013 Regione Sardegna—Asse VI Competitivit 6.2.1 a, CUP F25C10001420008, and in part by the Telit Communications SpA under project "Platform for the deployment of distributed applications in Vanets and WSNs."

The authors are with the Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy (e-mail: roberto.girau@diee.unica.it; salvatore.martis@diee.unica.it; l.atzori@diee.unica.it).

Digital Object Identifier 10.1109/JIOT.2016.2616022

platforms that make easier the communications among objects, help the developers in creating new applications on top of the available objects' services, allow the users to have complete control of their own data and objects, and are reliable and efficient to support the interaction of billions of objects.

To further advance in this respect, this paper provides the following major contributions.

1) We analyze the major requirements that should be addressed by an IoT platform to make easy the deployment of services, sharing of code, and services among different users, guarantee that every user can correctly handle the data generated by her own objects.
2) We present the IoT solution named Lysis,[1] which addresses the previously presented requirements. The major feature of Lysis is that relies on a platform as a service (PaaS) model that allows the users to have complete control over their data, which is not always assured by alternative solutions. The platform also fosters reusability of services and software developed by third parties and users.
3) The concept of social IoT is followed to develop the virtualization layer. As a result, each object is an autonomous social agent, according to which objects are capable of establishing social relationships in an autonomous way with respect to their owners with the benefits of improving the network scalability and information discovery efficiency.
4) The notion of social objects is used to develop an architecture that allows for deploying fully distributed applications. Accordingly, the application is deployed as a collaboration among social objects that are running in different cloud spaces and that are owned by different users, whereas in past works all the involved system components are managed by the same entity. In Lysis, these social objects are implemented in a horizontal distribution by using independent Web services that run in the cloud spaces managed by the users.
5) A use-case is presented to show the way we implemented the Lysis architecture and its potentialities. Performance evaluations are also shown in terms of scalability and reduction of computational load at the PDs.

This paper is organized as follows. Section II presents the considered requirements, past works and Lysis novelties. Section III describes the major layers of the Lysis platform. Section IV illustrates the way we have addressed the requirements. Section V analyzes a use-case and provides performance evaluations. Section VI provides a comparison of Lysis with alternative solutions and Section VII draws final conclusions.

## II. Requirements, Past Works, and Lysis Innovation

In this section, we present the key requirements that have been considered as the major input in this paper, with relevant past works. We also highlight the major novelties of the proposed Lysis platform.

### A. Distributed Social Objects

There are recent studies demonstrating that the issues related to the management and effective exploitation of the expected huge numbers of heterogeneous devices could find a solution in the use of social networking concepts and technologies [7]. For instance, Ortiz *et al.* [8] introduced the idea of objects able to participate in conversations that were previously only available to humans. Analogously, the research activities reported in [9] consider that, being things involved into the network together with people, social networks can be built based on the IoT and are meaningful to investigate the relations and evolution of objects in IoT. In [10], 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 according to rules set by the owner. In this paper, authors demonstrate that an approach derived from human social networking can provide a high level of scalability due to a high correlation between required information data and social relationships. According to this model the registered objects are augmented with the attitude to create the following relationships.

1) *Ownership Object Relationship:* Created between objects that belong to the same owner.
2) *Co-Location Object Relationship:* Created between stationary devices located in the same place.
3) *Parental Object Relationship:* Created between objects of the same model, producer and production batch.
4) *Co-Work Object Relationship:* Created between objects that meet each others at the owners' workplace, as the laptop and printer in the office.
5) *Social Object Relationship:* Created as a consequence of frequent meetings between objects, as it can happen between smartphones of people who use the same bus every day to go to school/work and people hanging out at the same bar/restaurant/gym.

Each type of relationship is created whenever certain conditions are satisfied. Each social object has to verify the occurrence of these conditions by analyzing its own profile [this is the case for the ownership object relationship (OOR) and parental object relationship], its own movement patterns [this is the case of the co-location object relationship (CLOR)] and the context of the device [co-work object relationship (CWOR) and social object relationship (SOR)]. With reference to the last case, the object has to understand in which places it is located during the day and to detect when it is located in the working places. Whenever it encounters for a given period of time other objects in this place, then it starts the process about the creation of the CWOR.

Girau *et al.* [11] is a past implementation of the SIoT paradigm developed starting from the open source project ThingSpeak.[2] Whereas the general SIoT features have been implemented in this platform, it makes use of a centralized approach, where the objects do not communicate directly with

[1]Lysis is the only dialogue of Plato in which the philosopher Socrates discusses the nature of friendship with his disciples.

[2][Online]. Available: https://thingspeak.com/

each other, but through the server, also to establish social relationships. This is inherited by the ThingSpeak project which, as most of the existing solutions, focuses on the server-object communication rather than object-to-object interactions. Whereas this prototype allowed for verifying the efficacy of the social networking concepts in IoT, it does not exploit the benefits of a distributed approach that can be achieved by allowing object-to-object direct and autonomous communications. Indeed, with the distributed approach we can improve the scalability, which is also demonstrated in our performance evaluation section.

## B. Virtualization

A network with the expected huge number of IoT connected devices introduces big issues in terms of navigability, management, ubiquity, scalability, reliability, and sustainability of the network and offered services. Not only it is a matter of numbers but also of heterogeneity and the limited resources that frequently characterize the PDs. Most of these issues can be addressed through the virtualization layer [12], which is where the VO implements the digital counterpart of the PDs, spokes for it and introduces some functionalities that could not be taken by the real world objects, such as: supporting the discovery and mash up of services, fostering the creation of complex applications, improving the objects energy management efficiency, as well as making the interobjects communications possible by translating the used dissimilar languages.

The virtualization feature has been introduced by major research projects. In the IoT-A project, physical entities are represented in the digital word by means of virtual entities, which are the access point to the real world by IoT applications through well-defined and standardized interfaces [13]. In the Compose platform, virtual representations of physical objects are named service objects, which support the handling of communications, the processing of sensor data, and the description of objects' characteristics to support semantic discovery mechanisms [6]. This platform has the limit in terms of the functionalities the object owners are provided with for the management of their own objects. Also in the European FP7 iCore project [5], [14], virtualization is a major feature. The proposers define the VO as the virtual alter ego of any real-world object (RWO), which are dynamically created and destroyed. Herein, cognitive technologies guarantee a constant link between RWO and VO and ensure self-management and self-configuration. iCore also proposes an aggregation layer where composite VOs implement complex services that are reused by different applications. The iCore team has developed a preliminary prototype, which however has not been devised for being deployed in the cloud. The autonomicity of VO is also addressed in the Cognitive IoT (CIoT) paradigm [15]. Herein, Wu *et al.* stated that connecting the objects is not enough but they should be able to learn, think, and understand from both the real and the social worlds. In CIoT, VOs are interconnected and act as agents with minimal human intervention, interact with each other by exploiting the context-awarness, storing and learning the acquired knowledge, and adapt themselves to situations through efficient decision-making mechanisms.

There are two major contributions of this paper in this area. First, to the authors' knowledge the state of the art misses a platform that fully implements the complete functionalities of VOs for IoT. Second, we extend these functionalities by creating the social VO (SVO), where the social behavior is injected into the VOs.

## C. PaaS in IoT

Many platforms exploit cloud computing technologies to provide IoT services in different environments, such as smart home [16], smart cities [17], smart management of inventories [18], eHealth [19], [20], environmental monitoring [21], social security and surveillance [22], mine security [23], and Internet of Vehicles [24]. Although very effective for the purpose they have been proposed, these solutions are most of times vertical implementations, lacking in horizontal enlargeability to become cross-application platforms, de facto limiting their adoption in other IoT domains. Indeed, in these realizations, domain-specific or project-specific requirements drove the design of the major system components and determine most technological elements ranging from sensors and smart devices to middleware components and application logic. This is discussed in [25], where Li *et al.* highlighted that isolated IoT platforms are implemented like silos and have been also named *virtual verticals*. Accordingly, any client of IoT solutions is isolated from the others and just share the storage and computing resources. They then propose an additional component, named "domain mediator" İto make the different PaaS IoT platforms talk each-other. This issue is also the focus of Gubbi *et al.* [26] that present a user-centric cloud-based model to design new IoT applications through the interaction of private and public cloud showing an attempt of usage of cloud computing to provide horizontal solutions.

Differently from these past works, in Lysis we fully exploit the PaaS model for the implementation of a cross-application IoT platform.

## D. Data Ownership

Existing IoT services provided in software as a service modality are rapidly conquering the market as it does not require the user to manage the infrastructure, to configure the used platform and services [2], [3], [27], [28]. The users are then granted with the required cloud space for storing sensor data and to run simple applications such as trigger actuations, send alerts, and visualize log graphs. In the near future, these services will permeate our everyday activities with all our devices connected in the cloud where any information about our activities will be stored and analyzed, without however, any user control about where this information is stored and who can access to. This process is felt as a strong threat to the user privacy and people feel increasingly observed, causing a strong feeling of distrust of the whole series of applications that offer services using data from personal devices. On the basis of these considerations, we have designed our platform so that the data generated by the devices is kept in the owner

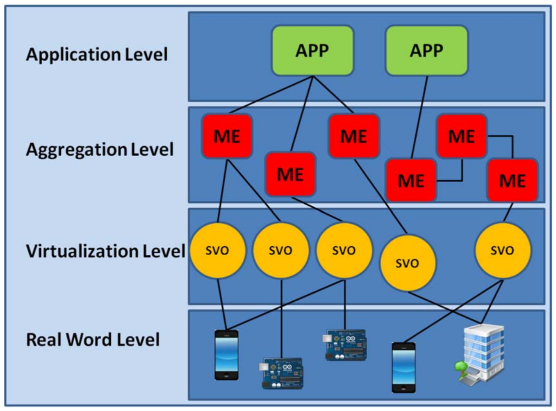


Fig. 1. Four levels of the Lysis architecture.

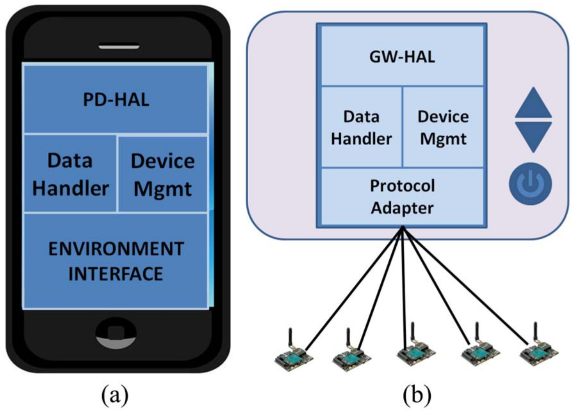


Fig. 2. (a) PDs able to communicate with the platform. (b) Objects that need a GW to interact with the platform.

cloud space. In this way, the produced information remains in the hands of the user that can dispose of this data at her will.

## *E. Reusability*

A prerequisite to ensure efficiency in IoT platforms is related to the reusability of data. Requests for the same data from the same sensors from different IoT applications cause extreme inefficiency in accessing hardware and result in a waste in terms of energy and bandwidth, if not handled properly. This issue should be handled by the virtualization level, as the data requests for a chosen physical sensor should be aggregated by its virtual counterpart allowing for saving vital energy. A similar way of thinking can be done at the aggregation level. A set of VOs can be aggregated to provide a service, which can be needed by multiple applications. Also in this case, the composition of VO must have the necessary intelligence to adapt to multiple requests of the same pattern. Decisions on which simple VO services to aggregate should be taken on the basis of the context.

Reusability also refers to the code and services as done by the FI-WARE project [29], where the objective is to build the basic platform of the Internet of the future made of basic elements (generic enablers) that allow for sharing functions on a multiplicity of areas in the Internet. In the specific domain of IoT, instantiating a new process for the communication with the PD and processing of the data should not require rewriting code already developed but should rely on sharing of codes among the communities of users and developers. The sharing should be done at all the architectural layers, from the PDs drivers till the upper most application layer. This represent another important requirement for the studied solution.

## III. Overall Lysis Architecture

Fig. 1 shows the overall architecture of the Lysis platform through four functional levels: 1) the lower level is made up of the “things” in the real world; 2) the one above is the virtualization level, which interfaces directly with the real world and is made up of SVOs; 3) the level of aggregation is responsible for composing different SVOs to set up entities with augmented functionalities called micro engines (MEs); and 4) the last level is the application level in which user-oriented macro services are deployed.

In the following sections we briefly describe the major components, whereas more details with reference to the introduced novelties are provided in Section IV.

## *A. Real World Level*

As well-investigated in the iCore project [5], the lowest level is always made up of the RWO. Some of these are PDs able to directly communicate through the Internet, such as smartphones, laptops and TV set-top-boxes [see Fig. 2(a)]. Some others cannot directly access to the Internet and have to use local gateways (GWs) [see Fig. 2(b)].

The PDs and the GWs implement the following modules to be part of the platform.

1) *Hardware Abstraction Layer:* It communicates with the corresponding module in the virtualization level. Its major role is to introduce a standardized communication procedure between the platform and the extremely variegate set of PDs, simplifying the platform southbound APIs. It is also in charge of creating a secure point-to-point communication (encrypted) with the SVOs.
2) *Data Handler:* It intervenes whenever there is the need to process data from sensors before being sent by the PD-hardware abstraction layer (HAL) to the virtualization level. For example, data coming from sensors could be strings of hexadecimals, which have to be processed to extract actual numerical values to encapsulate them in JSON format ready for dispatching.
3) *Device Management:* It implements the real device logic with reference to the participation of the PD to the Lysis platform, e.g., controlling the sensing frequency, managing local triggers, and overseeing the energy consumption. It also runs the code that can be updated in run-time locally in the PD.
4) *Environment Interface/Protocol Adapter:* In the case of the PD, it consists in the hardware drivers for all local sensors and actuators. In the case of the GW, it implements the communication with the ICT objects through the available protocol.

The HAL component is necessary to overcome the significant heterogeneity of the real word devices and to make the service always reachable. This is not always true for SVO to RWO communications. Most of times hypertext transfer protocol (HTTP) and constrained application protocol are unusable if behind a network address translation service, so it is better to use either protocols like MQTT/AMQP that use persistent sockets or vendor push protocols like Google Cloud Messaging or Apple Push Notifications. In case of multiple communication channels, the HAL has to manage the switching among these. This happens for instance when the device is equipped with two or more communication interfaces, such as cellular and satellite, which have to be used according to the user preferences (e.g., to save money). These issues are addressed by the HAL component in the RWO and in the visualization layer with a smart management of the possible communication protocols.

Virtually, all the physical devices can be abstracted in the cloud through the HAL. However, there are two limitations in this respect. First, for some real-time applications, sensed data in the cloud is useless and this is particularly true when the device should quickly react to the surrounding context. Whereas edge and fog computing technologies tend to reduce the latency [30], this may not be enough. Additionally, it also happens that applications require device-to-device communications as some decisions have to be taken in a cooperative way, and again this prevents from completely exploiting the cloud virtual counterparts. In this case, the device management modules has the role to accept code injected from the cloud to perform local processing. Indeed, it is able to receive, understand and execute code sent from the upper levels. To allow the highest reachable abstraction, a virtual machine is needed on the devices able to interpret programming languages like Python and Node.js.

Second, for some devices the vendors do not disclose all the details to have complete access to some capabilities. For this reason the abstraction allows only for accessing to a controlled view of the device sensors. It also happens that the device is only reachable through vendor Web services available through the cloud. For example, satellite communication systems based on short burst data do not support device direct communications letting only a connection with a control platform through REST APIs. Again, the SVO-HAL can connect to this platform and make this connection transparent to application developers, but still the data is preprocessed giving a limited view of the device capabilities.

## B. Virtualization Level

The hardest challenge of the IoT is to be able to address the deployment of applications involving heterogeneous objects, often moving in large and complex environments, in a way that satisfies the quality requirements of the application itself, while not overloading the network resources. For this reason, the VO has become a key component of many IoT platforms, representing the digital counterpart of any real (human or lifeless, static or mobile, solid or intangible) entity in the IoT.

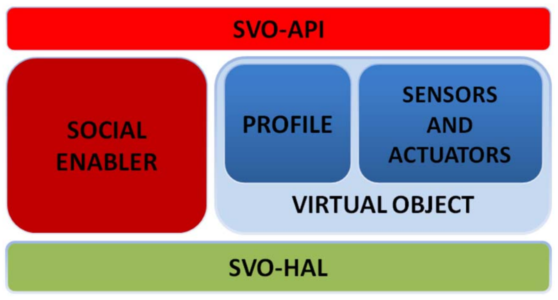


Fig. 3. Components of the SVO.

In our solution this layer assumes an even more important role as it is augmented with the social behavior, bringing to the concept of SVO. When designing the SVO we had to identify the functionalities that were in common with all the SVOs independently from the PD characteristics and define the social enabler, as shown in Fig. 3. The other functionalities, which are mostly affected by the specific device characteristics, are implemented by the VO module. The "type" of RWO is represented by a *template* of VO. For example, every smartphone model has the same VO template; however, there is a different VO instance per smartphone PD, which is the actual Web service running in the cloud.

The template consists of the VO schema, a semantic description of the related RWO. Capabilities and resources of the real object are depicted inside the VO schema. The second component of the template is the *software agent* source code, which is the computational engine of the VO to be run in the cloud.

The VO schema can be seen as the semantic description of the class of RWOs of the same type, while the VO Profile is a precise description of the object itself. It is important for the installer to complete the semantic description of the instance of VO to allow for a correct search of the resources needed for the creation of services. It can also be modified by the RWO itself through the SVO-HAL or by the social enabler which can extract a social context description. A major component in the VO is data points representing sensors and actuators available through REST APIs, which are available for the levels above. In addition to allowing access to the HW, the SVO-API allows for setting a minimum of logic if this than that on SVOs, whose actions can involve the object itself or even its friends. The SVO-API also provides access to social resources to dynamically change the behavior of each SVO within the social network, to search the resources, and to receive feedback needed to evaluate the trustworthiness of friends.

## C. Aggregation Level

The ME is an entity that is created to implement part of the applications running in the upper layer. It is a mash-up of one or more SVOs and other MEs. With reference to this entity there are two important components: 1) the *instance* and 2) the *schema*. The instance is a piece of programming code running in the cloud; it must be able to reuse the output of an instance to respond to requests that present the same inputs in order to save redundant data requests that consume bandwidth and CPU

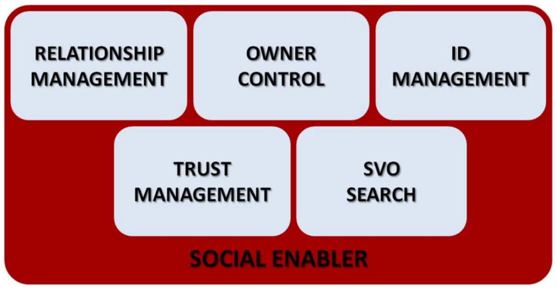


Fig. 4. Components of the social enabler.

unnecessarily. It must also be able to understand whether there is a malfunctioning in one of the input or output. In this case, it requires the reassignment of resources to the control unit. Each ME is described by a schema that contains a semantic description of the input, the output (if any) and the processing activities. It also contains a summary of the help that is useful to support the developers in using MEs.

### D. Application Level

At this level the applications are deployed and executed exploiting one or more MEs. The interface with the user also assumes a key role; in fact, although we are in the field of the IoT solutions, which are centered around device-device communications, the center of gravity at the end is still the user. An application at this level shows a font-end interface to the user, and a back-end interface to the underlying layers.

## IV. Addressing the Key Requirements

In the following we describe the choices that have been taken to satisfy the requirements of Section II. We also analyze the trustworthiness and security issues.

### A. Social VO

The social enabler (SE) extends the functionalities of the VO and, consequently, the relevant real world object by adding social capabilities. The SE is in charge of the socialization of the SVO by allowing the establishment, management, and termination of social relations. A social graph connecting each SVO with the others according to their friendships is used to find the services required at the application level. The type and strength of relationships created among objects also deliver key information for predicting the trustworthiness of an object to provide a desired service [31]. Fig. 4 shows the modules implemented in the SE.

It is important to highlight that the frequent interactions among the social objects for the implementation of the SIoT paradigm was the major element that brought us to implement SVOs as autonomous processes that could implement all the requested functionalities in a distributed way. Accordingly, once the SVO is instantiated it interacts with the other members of the community without the need of a centralized component, which could represent a limit to the system scalability.

The *owner control* module interfaces directly with the GUI platform. It allows the user to manage general permissions of the SVO resources and allows for the management of permissions and rules about the establishment of social relations. It interfaces directly with the *relationship management* module that contains the logic for the creation of the social relations foreseen in the SIoT paradigm. It uses an alerting system to send and update the information related to the life in the social network. Also, it is in charge of handling the association to groups divided by topic as it happens in human social networks. The *SVO search* (SVOS) module implements an algorithm of SVOS required in the upper levels. The search algorithm will be explained in the following, and uses the trust values computed for the friends and stored in the local database to build a ranking list of SVOs. These values of trust are created and updated by the *trustworthiness management* module and stored in the local database. The *identity management* module manages all the operations of authentication and authorization by means of cryptography tokens and different levels of API keys to access to resources (at hardware level or at SVO level) depending on the required permissions. At SVO level, three classes of permissions are foreseen: 1) public; 2) private; and 3) friend. In the first case, accessing to the resources is allowed to anyone without the need of any API key. If the permissions are set to “private” the owner key is required. Of course, in this case only, applications instantiated by the owner are allowed to access to resources. Last, if the permissions are set to “friend,” the access is allowed only by SVO friends which have a friend API key. The owner key is generated when the user creates a new account on the platform for the first time and it is stored in the user profile. It can be obtained by the Lysis deployer only and can be reset by the user through the user interface on the Lysis platform. The friend API key is generated by the SVO and redistributed to its friends (and friends of friends) to access the shared resources (i.e., the friend resources) as a result of the search process. The *identity management* module is also in charge of managing the binding to the hardware. We talk about *static binding* when the association is made during SVO deployment. The association is persistent and grants full access to hardware resources. We talk about *dynamic binding* when the association is opportunistic and for a limited period of time. There are two ways to achieve a dynamic binding: 1) *hard binding*, when the association is set up between PD-HAL and SVO-HAL and 2) *soft binding*, when the association is set up between SVO and SVO. In the first case, the association is allowed only to SVOs that are OOR friends (belonging to the same owner): the requester asks for a PD hard binding to the controlling SVO. This one verifies the relationship type and, if authorized, sends a cryptography token and URLs to both PD-HAL and requesting SVO. In the soft binding, the association is set up at the virtualization level, so that only resources with public or friend permissions are allowed to be accessed to. In this case, the requesting SVO has fictitious resources pointing to real resources of the granting SVO.

The most powerful modules in the SVO are those that deal with trust management (TM) and with SVOS. The algorithm related to the former has been discussed in details in [31] and incorporated in this Lysis platform. The latter is discussed herein. SVOS is the functionality the application layer is

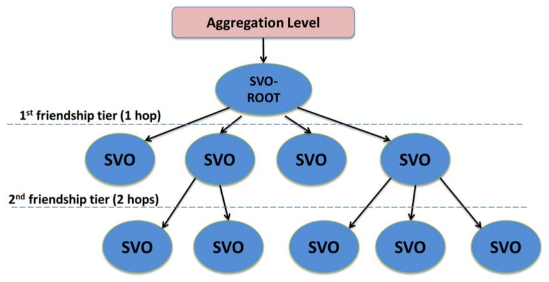


Fig. 5. Objects involved in an SVO search process.

provided with when there is the need of a service and/or information that can be provided by other objects. Accordingly, this function is triggered every time there is an SVO (say SVO-a) that needs to interact with an other SVO (say SVO-b), which have to be found on-the-fly to reach the application goals. To this a profile of SVO-b is used, which is indeed a description of the desired service that should be capable to provide. A key role is taken by a node called SVO root (SVOR), which is elected among all the SVOs owned by the same owner of SVO-a. It is elected as the most powerful node that is well-connected with other nodes and represents the first node to interact with when someone has to be found by any other friend in the co-ownership community (recall that this is made by all the SVOs belonging to the same person). The SVOS module accepts requests from the upper levels. Once the SVOS is activated, the first action is to check if the required profile matches its own profile. In this case, the SVOS responds with its own resources. If it is not the case, it checks its local database if there are matches among its friends. In case of positive result, the SVOS returns the address of the found resource(s) (more than one node may match the profile) and the friend API key(s) to access to it. In the case of mismatch, the query is forwarded to its friends with high potentials to know the target node or with links with strong network hubs, as shown in Fig. 5. The process is repeated until a positive result is found, which is then returned to the SVOS that sends it to the higher levels. Since each SVO is able to respond to SVOS queries, other SVOs in OOR provide the necessary redundancy to the SVOR in case of congestion or malfunctioning. The strength of this system is that there are no single points of failure, and in the case of failing nodes, the network adapts itself by forwarding the requests toward alternative routes of the social graph. In addition, using the SVO with greater centrality decreases the chance of forwarding the request outside the SVOR.

## *B. PAAS Oriented Solution*

A PaaS service provides the tools needed to develop, run, and manage Web applications, such as the execution containers of Web services and related databases for data storage. Since Lysis is a virtualization-based IoT PaaS, it must also provide all the tools to deploy SVOs, MEs, and Apps, as well as tools for the search of these ones and a development environment for developers.

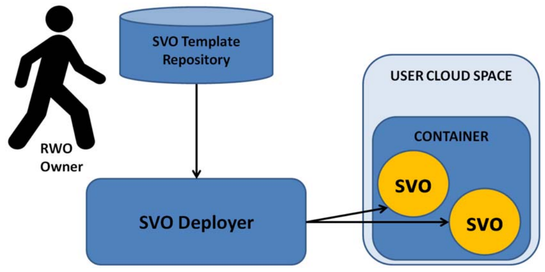


Fig. 6. SVO deployment process.

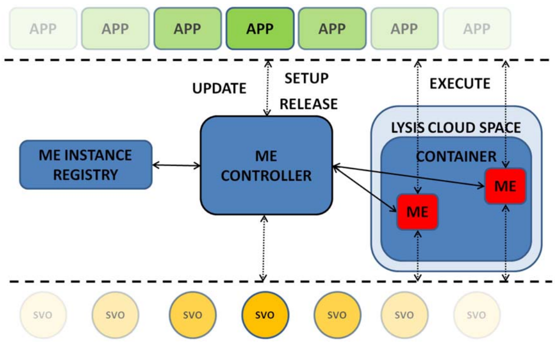


Fig. 7. ME deployment process.

*1) SVO Deployment:* To deploy SVOs in the cloud, Lysis provides the infrastructural elements shown in Fig. 6. From the template repository the user chooses the correct template for the installation of each SVO. The template is then taken as input by the SVO deployer, which is in charge of instantiating the agent and giving an initial configuration to SVO. The deployer works only during the set up phase because once instantiated the SVO is an autonomous Web service able to introduce self-updates and manage the communication with its friends as in a human social network. Once instantiated, the SVO runs in the user cloud space.

*2) ME Deployment:* Fig. 7 shows the elements of the aggregation level. Herein, SVO resources are combined in different MEs, which are entities that inherit some or all of the functionalities of the SVOs and are augmented with more advanced features such as: statistical analysis, data forecasting, and artificial intelligent cooperation. Associations between MEs and SVOs are managed by the ME controller. During this phase, the controller triggers the execution of the search operation to find the right SVO and to retrieve the relevant permissions. This SVO search functionality is the one implemented by the root SVO of the user where the App is running.

To be found by the ME controller, each ME has to be documented in the registry. This element of the aggregation level contains a database of instances of active MEs. Each line of the DB is related to a single ME and contains: the ME ID, the ME URLs, the access permissions, and the time-stamp of the last check.

The ME controller is the coordination element of the entire level. When an application at the upper level sends a query for the first time, the MEC checks all the involved MEs asking for the related URLs to the ME registry. It asks for SVO search to the SVOR of the user who started the application

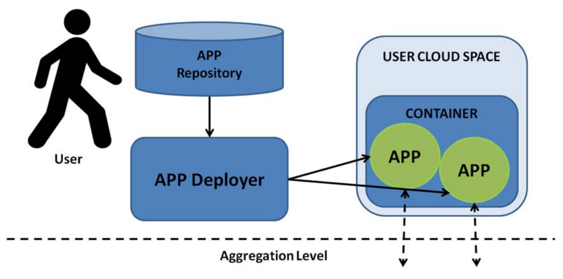


Fig. 8. Application deployment process.

from which takes the owner key. Once it has the resources by the SVOs, it associates them to the MEs which register the query ID and the required resources (input and output) in the local database. Finally, the MEC notifies the latest ME in the processing chain to the application. This ME will also be the one elected as responder to the queries of the application. Fig. 8 shows all the steps that lead applications to run in Cloud. Users can choose among a list of applications in the repository, and the deployer puts the source code in the user cloud space, as shown for the SVOs.

## C. Reusability

The ME, which offers reusable and commonly shared functions serving a multiplicity of application areas, is usually developed by the Lysis platform maintainer. By observing queries on the ME controller, it is possible to identify recurring patterns of queries. Then, new MEs serving these patterns are developed and deployed. Nevertheless, third party ME development is allowed: there could be a pay per use service at aggregation level too.

To promote horizontal spread of IoT cloud applications we foresee two development environments: 1) a scripting environment and 2) a GUI environment. In the first case the developer is required to know the programming language of the container of the PaaS platform hosting the implementation of the Lysis architecture. The actual query that should be sent to the ME controller is written in a domain specific language (JSON or XML). In that query, semantic search and processing chains are defined. Differently, the GUI development platform provides a Web platform where it is possible to connect the blocks in the processing chain. These blocks are nothing but the MEs and the SVOs. In this case, the developer is required to set only few parameters of the blocks without knowing any programming language. The result in both cases, is the source code of a Web application. The code must then be verified and certified by the Lysis platform maintainers before being placed in the repository. The application repository can then be visited by the user who can choose which application could be installed in the user cloud space. In fact, just as with the case of the VO, the burden of running the application is taken by the user, who can maintain full ownership of data. Accordingly, a classic IoT data logging platform that in old IoT cloud architectures is performed by a single Web application with one db managed by the service provider and serving multiple users, evolves into a system of multiple Web applications, each under responsibility of the user who uses the service.

## D. Data Ownership

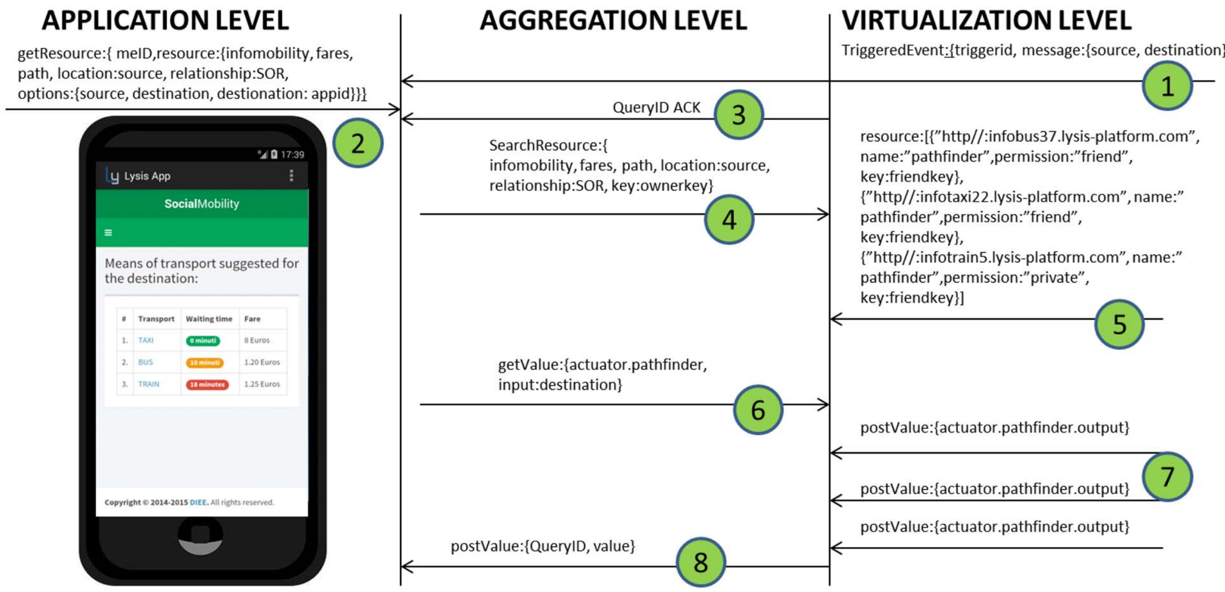
One of the main features of the proposed architecture with respect to other IoT architectures is the fact that the user keeps the ownership of their own SVOs and consequently of the relevant data. Indeed, the deployer deploys the SVOs in the container of the user cloud space so that she is directly responsible for her own data and for the running and storage costs in the cloud. Indeed, whereas the template repository and the deployer are hosted in the Lyser service provider infrastructure, once the SVO is created it is under the complete control of the owner who can decide to change its configuration and kill the process whenever she wants.

The same happens at the application level when a similar process is implemented. The applications are selected from the repository hosted in the Lysis platform but the Lysis instantiates it in the user cloud space. Clearly, it is possible because these operations are done by the user herself (and has the permissions to do it).

## E. Security and Trustworthiness

Security issues have to be analyzed at both southbound and northbound SVO interfaces. As to the former, as seen in Section III-A, at the time of binding an SVO with a PD, they exchange identifiers and encryption keys, representing the weakest phase during which successful sniffing attacks could compromise any future SVO/PD activities. From this moment on, the communication is encrypted and secure. However, still, the keys could be stolen from weak and simple devices. As to the northbound interfaces, data access is regulated by the ME controller that provides a layer of anonymity between users and providers of resources. In the case of mono-cloud provider, i.e., the MEs and the SVOs are running in the same provider space, the encryption of the communication is not needed because they take place within the infrastructure of the cloud provider and any security problem is overcome by the ability to ensure the security of this overall infrastructure. In the case of a multiprovider infrastructure or hybrid cloud/local server scenario, each interface represents a vulnerable point in the architecture for which security technologies related to Web applications such as asymmetric key encryption SSL/TLS over HTTP/MQTT are needed [32].

Security has to be addressed in conjunction with the evaluation of the objects' trust in an IoT environment, as there are misbehaving nodes that can perform discriminatory attacks on the basis of their social relationships for their own gain penalizing others nodes. In addition, misbehaving nodes with close social ties can coalesce and monopolize a service class. Since the trust evaluation of the nodes is highly integrated with the search of the IoT services, the concept of TM it is of paramount importance. There are research works which can demonstrate how the social approach of the SIoT paradigm can be useful for trustworthiness evaluation. Nitti *et al.* [31] showed a subjective algorithm which can be executed by SVOs to retrieve a trust ordered list of resources needed by the upper layers. One of the drawbacks of these algorithms is the high traffic between nodes needed to keep updated friend trust values and the relevant required computational power. In Lysis,


APPLICATION LEVEL
AGGREGATION LEVEL
VIRTUALIZATION LEVEL
getResource:{ meID,resource:{infomobility, fares, path, location:source, relationship:SOR, options:{source, destination, destionation: appid}}}
TriggeredEvent:{triggerid, message:{source, destination}
1
2
QueryID ACK
3
SearchResource:{ infomobility, fares, path, location:source, relationship:SOR, key:ownerkey}
4
resource:[{"http//:infobus37.lysis-platform.com", name:"pathfinder",permission:"friend", key:friendkey}, {"http//:infotaxi22.lysis-platform.com", name:" pathfinder",permission:"friend", key:friendkey}, {"http//:infotrain5.lysis-platform.com", name:" pathfinder",permission:"private", key:friendkey}]
5
getValue:{actuator.pathfinder, input:destination}
6
postValue:{actuator.pathfinder.output}
postValue:{actuator.pathfinder.output}
7
postValue:{actuator.pathfinder.output}
postValue:{QueryID, value}
8
17:39
Lysis App
SocialMobility
Copyright © 2014-2015 DIEE. All rights reserved.

Means of transport suggested for the destination:

| # | Transport | Waiting time | Fare |
|---|---|---|---|
| 1. | TAXI | 0 minuti | 8 Euros |
| 2. | BUS | 10 minuti | 1.20 Euros |
| 3. | TRAIN | 18 minutes | 1.25 Euros |

Fig. 9. SocialMobility application: execution process.

the use of the SVO distributed approach and of the PaaS technologies partially limits this issue.

## V. Use-Case and Performance Evaluation

### A. Lysis Implementation

To implement Lysis, we chose the Google App Engine (GAE) PaaS as container at the different architectural levels. The platform is available to any user with a Google account at: `http://www.lysis-iot.com`. The choice was guided by the fact that any user is provided with an user-friendly environment where to instantiate 25 free Web services. This fact is very important to get an initial population of SVOs allowing people to try this new IoT environment. Furthermore, GAE comes with key useful APIs [33]: Search API and Maps API. The former allowed us to implement on each SVO a template repository of friends by means of document representation enabling full-text search through the social graph; the latter allowed us to use an uniform repository of locations which are needed for the social relations CWOR and CLOR, which rely on information about objects positions. Specifically, the search API provides a model for indexing documents that contain structured data and supports text search on string fields. The documents and indexes are stored in separate datastores optimized for search operations. It does not fit applications with large result sets; however, it is used in our social environment, where there is a separate database instance for each SVO, with a limited size given by the number of friends. The search API are principally used during SVO discovery, according to which an SVO looks for friends that may provide a target service in its local database.

### B. Analysis of Use-Case

In this section, we use a social infomobility scenario as an illustrative example to show how the platform works. In this scenario, a user arrives at a new airport where her smartphone, already registered to the Lysis platform and with the relevant middleware installed, becomes friend with the airport multimedia infopoint while waiting for the baggage. The new friend immediately sends a message suggesting the installation of the SocialMobility App, which is confirmed by the user. Other infopoints in the surroundings, one for each means of transport, can give information about waiting time, ticket price, provided transport services, and nearby ticket selling points. Information about taxi fares are offered by an ME that carries out statistical analyses on the services provided by the taxis in the city. Once the SocialMobility App on the user cloud space is installed, the process follows the steps shown in Fig. 9. A trigger in the user smartphone sends an alert to the App when a hotel reservation occurs with the related hotel position (1). Then, the App sends a query for infomobility resources to the aggregation level by means of the ME controller (2), (3). The latter forwards the query to the SVOR of the user (4). The information is available since the user smartphone has an SOR relationship with the airport infopoint, which has CLOR relationships with the bus infopoint, taxi infopoint, and train infopoint. The SVOR responds with the resources and the friend API key needed to access those resources (5). The ME controller aggregates the received output into an instance of the ME that takes care of dealing with all the value requests to the needed SVOs queried by the App (6), (7). Indeed, note that at this point there is no need to reach the application level but this part of App is implemented by this ME. There is also another ME in the process since the information about taxis is provided by a private taxi statistic ME which aggregate positions, paths, and fares of hundreds of taxis in a city. It is important to highlight that this last ME may not be instantiated by the considered application; indeed, it is likely it was already running as needed by other applications which all for different purposes need the same service. The final aggregated information is delivered to the requesting application, which renders it in a responsive Web view (8).

In the social mobility scenario, we have shown a typical automatic social interaction among devices. The user SVOR is responsible to start a discovery among SVO friends to find the requested resource. The requested information is at a two hop distance from the SVOR but the query is forwarded for one hop, from SVOR to airport infopoint, since this one has a description of its friends (bus, train, and taxi infopoints) and knows whether they can satisfy the request. Respect to other IoT platforms described in Section II, there is not need for dedicated elements such as SVO template repository, SVO registry, SVO management unit, and any other centralized system, since this information or actions are distributed among the social network of objects. The information about hundreds of objects such as that provided by taxis are aggregated by a Web service, the taxi ME, which guarantees the vehicle privacy, giving at the same time high level information through the OOR relationship between the taxi infopoint and all the taxis belonging to the same company. There is only one application running in the cloud usable by any Lysis-enabled device so that the SocialMobility developer had only to make one application without the need of creating one for each HW platform. As described in Section IV and by exploiting the Google API, user applications, and SVOs are deployed on the user cloud space granting data ownership to the user.

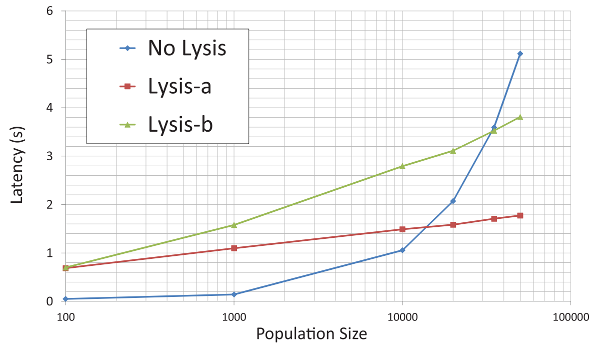


Fig. 10. Information search latency at increasing size of the IoT population when using Lysis with a constant number of friends (Lysis-a), with a number of friends that increases with the population (Lysis-b), and with a conventional IoT solution (no-Lysis).

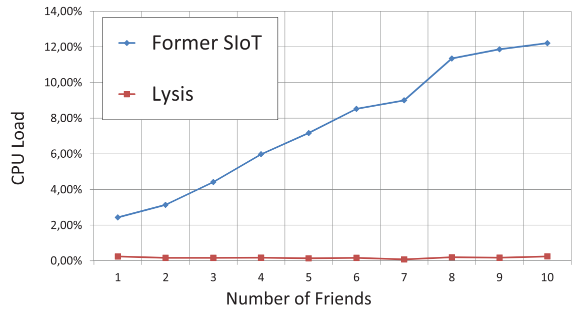


Fig. 11. Device CPU load when executing a crowdsensing application with the former SIoT and Lysis platforms.

## *C. Performance Analysis*

Two important aspects of the proposed platform are the scalability and the reduction of computation overhead at the PDs thanks to the use of the virtualization layer. We show some performance results with respect to these in the following.

A first experiment is related to the analysis of the time needed to search a service provider in the SIoT community. To this, we created a network of social objects with synthetic data as done in past studies [31], [34]. In these experiments we considered two types of social graph: a first one with a constant average friend number for each node, which brings to a graph diameter that increases with the population size (Lysis-a) and a second one with an average number of friends per node that increases with the population size so that the network diameter is constant (Lysis-b). For each type, we considered graphs with a population size in the range 100–50 000. The results are compared with those of a commonly adopted approach (no-Lysis) for an IoT solution with an NoSQL DB without social links between the nodes, as in the ThingSpeak solution, which are taken from [35]. Fig. 10 shows that when the population size is small, Lysis performs bad because the requested interactions among SIoT friend nodes impact on the latency more than the time for the database look-up. However, when the population size increases, the overhead for these communications is balanced with a low number of queries to look for the service provider in the SIoT, as it is quickly found with only few hops from the service client. Differently, in the no-Lysis solution the search in the whole database clearly increases significantly. Accordingly, the Lysis-a and Lysis-b curves cross the no-Lysis curve when the population size is approximately 16.000 and 35.000 nodes, respectively. This graph also shows that the performance of Lysis depends on the number of friends (see differences between Lysis-a and Lysis-b in the figure) per node and their selection is then quite important for the platform scalability, as studied in [34]. Indeed, each node has an average number of friends which is higher in Lysis-b than in Lysis-a, resulting in a bigger local database to store information about the friends in the former case with respect to the later case. Accordingly, in Lysis-b, the database latency increases (with the average number of friends), resulting in worse performance than the Lysis-a case.

The computation overhead at the PD is show in Fig. 11, which graphs the CPU load in a smartphone device for a crowdsensing application. This application consists in the retrieval of temperature values from friends (five Android devices and six Raspberry pi 2). In Lysis all this process is done through the VO in the cloud. Differently, in the former SIoT platform, the computation is made entirely on the client (smartphone): retrieval of measured values from friends and computation of average values. In the former SIoT, for growing numbers of friends the CPU load increases in a linear manner, whereas, in the case of Lysis, it remains almost constant. This proves that the execution of applications in Lysis has a far less effect in terms of CPU load and energy consumption compared to previous SIoT implementation and, in general, compared to platforms that are based on running the applications on the PDs.

TABLE I
REQUIREMENT-BASED COMPARISONS OF IOT PLATFORMS

| Platform | Social objects | Distributed Approach | Virtualization | PaaS | Data Ownership | Re-usability |
|---|---|---|---|---|---|---|
| iCore | No | Partially | Yes | N/A | N/A | Yes |
| Xively | No | No | Limited | Partially | No | Partially |
| oneM2M | No | No | Limited | N/A | N/A | N/A |
| Paraimpu | Limited | No | Limited | Limited | No | No |
| SIoT | Yes | No | No | No | No | No |
| Lysis | Yes | Yes | Yes | Yes | Yes | Yes |

## VI. COMPARISON WITH ALTERNATIVE SOLUTIONS

The number of IoT platforms that are being proposed is rapidly increasing as a consequence of the huge economic market value that the IoT-related applications are expected to take soon. Razzaque *et al.* [36] is a proof of this fervent activity where tenths of solutions in this field are surveyed with a focus on the middleware functionalities, which go from resource discovery to event management, from context awareness to privacy management. To highlight the novelties of our proposed solution with respect to the existing platforms in a concise yet effective way, herein, we concentrate on the major requirements we have taken as the starting point of this paper and we have made a comparison accordingly with five alternative solutions as representative of different categories. The final results are shown in Table I, which are commented on the following.

iCore [5] is an European FP7 project which gave a first extended definition of VO, as before the virtualization was only a marginal feature just related to the abstraction of PD functionalities. Indeed, in iCore there has been a significant work on the definition of the interactions between VOs, giving them some major cognitive capabilities. However, there is not any analyses about the setting where to execute these VO processes and there is not any available implementation of the whole platform. Additionally, the communications among the VOs are only partially distributed as a central management unit is necessary to implement important functionalities, such as VO search. The issue of data ownership is not addressed in this solution. Reusability is also a key aspect that is addressed in this project.

Xively platform [2] (formerly Cosm and Pachube) is a commercial IoT solution to transmit, store, and access to the data generated by the objects. The major focus is in the creation of big communities of objects (and owners) and to provide tools for the management of data objects and to simplify the development of applications. It exploits the PaaS features only partially as it does not provide an execution environment for IoT applications. Data is stored in the provider database and no separate datastore is available for each user. As to the reusability of the software, it is only available at the driver level so that the community can share the firmware for common devices. Instead, reusability is not a feature provided at the application level neither for the sharing of data. Virtualization consists only in a datapoint in the database for the access to the data related to each object.

The efforts to synchronize tasks for M2M standards led to the oneM2M Global Initiative [37] in order to develop globally recognized technical specifications for a relevant service common level. In the virtualization layer, resources are uniquely addressable entities in an RESTful architecture. The specifications do not refer to any particular implementation approach and there is no reference to the data ownership.

Paraimpu [3] is a social Web of Things platform to connect physical and virtual things to the Web. In this case, virtual things are resources available in other IoT platforms and are not intended as autonomous software agents. Moreover, the term social is intended from a human point of view as Paraimpu gives the possibility of sharing things among users through the human social networks. The PaaS approach is again implemented only to provide the user with the possibility to instantiate interfaces to access the devices but there are not functionalities for the deployment of applications. Also in this case the data is owned by the platform provider.

The first SIoT platform [11] was created as an add-on to the ThingSpeak solution, introducing the object social behavior as an functionality of the central server. In this implementation, the VOs are just entries in the central remote database for data logging and the relevant actions about data management, included social relationship management, are performed in a centralized way. The PaaS technologies are not considered.

As it has been described in the previous sections, the Lysis platform implements all these features as these were the major requirements. It can be observed that virtualization and the use of the PaaS approach are characteristics that can be found in alternative solutions, but the combination of these features with the distributed social approach in the implementation of the VO makes Lysis a distinctive solution.

## VII. CONCLUSION

In this paper, we have presented the IoT platform called Lysis, which presents four major features: 1) it has been designed to exploit the PaaS service model; 2) the SVO is a key element; 3) user data and applications are stored and executed in the user cloud space; and 4) reusability of templates and applications is put forward. The implementation of the platform on the GAE PaaS showed that this solution was greatly facilitated by the available API for semantic search and localization. Other important aspects remain to be explored: the issue of task allocation among the real objects and the virtual counterparts through runtime code injection into the real devices; deployment of the SVO in distributed cloud (edge/fog clouds) to follow the PD to reduce latency; large use-cases deployments.

## REFERENCES

[1] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[2] *Xively*. Accessed on Jun. 2016. [Online]. Available: http://xively.com

[3] *Paraimpu*. Accessed on Jun. 2016. [Online]. Available: https://www.paraimpu.com/

[4] *Carriots*. Accessed on Jun. 2016. [Online]. Available: https://www.carriots.com/

[5] (2012). *ICORE-Project Deliverable 2.1*. [Online]. Available: http://www.iot-icore.eu

[6] COMPOSE. (2012). *Collaborative Open Market to Place Objects at Your Service*. [Online]. Available: http://www.compose-project.eu/

[7] L. Atzori, D. Carboni, and A. Iera, "Smart things in the social loop: Paradigms, technologies, and potentials," *Ad Hoc Netw.*, vol. 18, pp. 121–132, Jul. 2014.

[8] A. M. Ortiz, D. Hussein, S. Park, S. N. Han, and N. Crespi, "The cluster between Internet of Things and social networks: Review and research challenges," *IEEE Internet Things J.*, vol. 1, no. 3, pp. 206–215, Jun. 2014.

[9] L. Ding, P. Shi, and B. Liu, "The clustering of Internet, Internet of Things and social network," in *Proc. 3rd Int. Symp. Knowl. Acquisition Model.*, Wuhan, China, 2010, pp. 417–420.

[10] 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," *Comput. Netw.*, vol. 56, no. 16, pp. 3594–3608, 2012.

[11] R. Girau, M. Nitti, and L. Atzori, "Implementation of an experimental platform for the social Internet of Things," in *Proc. IEEE 7th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Taichung, Taiwan, 2013, pp. 500–505.

[12] M. Nitti, V. Pilloni, G. Colistra, and L. Atzori, "The virtual object as a major element of the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1228–1240, 2nd Quart., 2016.

[13] (2012). *IoT-A-Project Deliverable 1.4*. [Online]. Available: http://www.iot-a.eu

[14] V. Foteinos *et al.*, "Cognitive management for the Internet of Things: A framework for enabling autonomous applications," *IEEE Veh. Technol. Mag.*, vol. 8, no. 4, pp. 90–99, Dec. 2013.

[15] Q. Wu *et al.*, "Cognitive Internet of Things: A new paradigm beyond connection," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 129–143, Apr. 2014.

[16] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, and C.-H. Lung, "Smart home: Integrating Internet of Things with Web services and cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, vol. 2. Bristol, U.K., 2013, pp. 317–320.

[17] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 112–121, Apr. 2014.

[18] L. A. Amaral *et al.*, "eCloudRFID—A mobile software framework architecture for pervasive RFID-based applications," *J. Netw. Comput. Appl.*, vol. 34, no. 3, pp. 972–979, 2011.

[19] C. Doukas and I. Maglogiannis, "Bringing IoT and cloud computing towards pervasive healthcare," in *Proc. IEEE 6th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Palermo, Italy, 2012, pp. 922–926.

[20] S. M. R. Islam, D. Kwak, M. D. H. Kabir, M. Hossain, and K.-S. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.

[21] 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. IEEE 6th Int. Conf. Sens. Technol. (ICST)*, Kolkata, India, 2012, pp. 374–380.

[22] B. Kantarci and H. T. Mouftah, "Trustworthy sensing for public safety in cloud-centric Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 360–368, Aug. 2014.

[23] E. Sun, X. Zhang, and Z. Li, "The Internet of Things (IOT) and cloud computing (CC) based tailings dam monitoring and pre-alarm system in mines," *Safety Sci.*, vol. 50, no. 4, pp. 811–815, 2012.

[24] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Software-defined networking for RSU clouds in support of the Internet of Vehicles," *IEEE Internet Things J.*, vol. 2, no. 2, pp. 133–144, Apr. 2015.

[25] F. Li, M. Voegler, M. Claessens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput. (CLOUD)*, Santa Clara, CA, USA, 2013, pp. 740–747.

[26] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[27] *ThingWorx*. Accessed on Jun. 2016. [Online]. Available: http://www.thingworx.com/

[28] *Sen.se*. Accessed on Jun. 2016. [Online]. Available: https://sen.se/

[29] (2011). *FIWARE-Project*. Accessed on Jun. 2016. [Online]. Available: http://www.fiware.org/

[30] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, 2012, pp. 13–16.

[31] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social Internet of Things," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1253–1266, May 2014.

[32] R. Roman, P. Najera, and J. Lopez, "Securing the Internet of Things," *Computer*, vol. 44, no. 9, pp. 51–58, Sep. 2011.

[33] Google. *Google Search API*. Accessed on Jun. 2016. [Online]. Available: https://cloud.google.com/appengine/docs/java/search/

[34] M. Nitti, L. Atzori, and I. P. Cvijikj, "Friendship selection in the social Internet of Things: Challenges and possible strategies," *IEEE Internet Things J.*, vol. 2, no. 3, pp. 240–247, Jun. 2015.

[35] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proc. IEEE Pac. Rim Conf. Commun. Comput. Signal Process. (PACRIM)*, Victoria, BC, Canada, 2013, pp. 15–19.

[36] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[37] *oneM2M*. Accessed on Jun. 2016. [Online]. Available: https://www.onem2m.org/

**Roberto Girau** received the M.S. degree in telecommunication engineering from the University of Cagliari, Cagliari, Italy, in 2012 for his thesis entitled "Trustworthiness Management in the Social Internet of Things." He is currently pursuing the Ph.D. degree at the University of Cagliari. His doctoral dissertation is entitled "Integration of Cloud Computing and Social Internet of Things: Study, Design and Development of a Cloud Platform for Social Internet of Things."

**Salvatore Martis** received the bachelor's degree in electronic engineering, and the M.Sc. degree in telecommunications engineering from the University of Cagliari, Cagliari, Italy, in 2011 and 2014, respectively.

He is currently a Research Fellow with the DIEE, University of Cagliari.

**Luigi Atzori** (SM'09) is an Associate Professor with the Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy, where he leads the Laboratory of Multimedia and Communications. His current research interests include multimedia communications and computer networking (wireless and wireline), with emphasis on multimedia QoE, multimedia streaming, NGN service management, service management in wireless sensor networks, and architecture and services in the Internet of Things.

Dr. Atzori is a Chair of the Steering Committee of the IEEE Multimedia Communications Committee. He is the Coordinator of the Marie Curie Initial Training Network on QoE for multimedia services (qoenet-itn.eu), which involves ten European Institutions in Europe and one in South Korea. He is an Editorial Board member of the IEEE INTERNET OF THINGS JOURNAL, *Ad Hoc Networks* (Elsevier), and *Digital Communications and Networks* (Elsevier). He served as a Technical Program Chair for various international conferences and workshops. He served as a Reviewer and a Panelist for many funding agencies, including FP7, cost action, Italian MIUR, and regional funding agencies.