Università degli Studi di Cagliari

**DOTTORATO DI RICERCA**

Ingegneria Elettronica e Informatica

Ciclo XXIX

**TITOLO TESI**

# Architectural and application solutions for the cloud Internet of Things

Settore scientifico disciplinare di afferenza

ING-INF/03

| | |
|---|---|
| Presentata da: | Roberto Girau |
| Coordinatore Dottorato | Prof. Fabio Roli |
| Tutor | Prof. Luigi Atzori |

Esame finale anno accademico 2015 – 2016

Tesi discussa nella sessione d'esame marzo – aprile 2017

PH.D. IN ELECTRONIC AND COMPUTER ENGINEERING
Dept. of Electrical and Electronic Engineering
University of Cagliari

# Architectural and Application Solutions for The Cloud Internet of Things

Roberto Girau

*Advisor:* Prof. Luigi Atzori

*Curriculum:* ING-INF/03 Telecomunicazioni

XXIX Cycle
April 2017

UNIONE EUROPEA
Fondo sociale europeo

REGIONE AUTONOMA DELLA SARDEGNA

POR SARDEGNA

*To my family and Mariella*
*Alla mia famiglia e a Mariella*

# Acknowledgments

Roberto Girau gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.)

# Contents

# Acronyms

| | |
|---|---|
| **BDS** | BeiDou Navigation Satellite System . |
| **CIoT** | Cognitive Internet of Things . |
| **CLOR** | Co-location Object Relationship . |
| **CoAP** | Constrained Application Protocol . |
| **CS-ME** | CrowdSensing Micro Engine . |
| **CSaaS** | CrowdSensing as a Service . |
| **CVO** | Composite Virtual Object . |
| **CWOR** | Co-work Object Relationship . |
| **D2D** | Device to Device . |
| **ECU** | Engine Control Unit . |
| **ESF** | Equally set Sampling Frequency . |
| **GAE** | Google App Engine . |
| **GLONASS** | GLObal Navigation Satellite System . |
| **GNSS** | Global Navigation Satellite System . |
| **GPS** | Global Positioning System . |
| **HAL** | Hardware Abstraction Layer . |
| **HTTP** | Hypertext Transfer Protocol . |
| **IoT** | Internet of Things . |
| **IoV** | Internet of Vehicles . |
| **ITS SA** | Intelligent Transportation Systems Station Architecture . |
| **LCM** | Local Code Manager . |
| **LND** | Local Neighbor Discovery . |
| **MANET** | Mobile Ad-hoc NETworks . |
| **MCS** | Mobile CrowdSensing . |
| **ME** | Micro Engine . |
| **MEC** | Micro Engine Controller . |
| **MOSDEN** | Mobile Sensor Data EngiNe . |
| **MSF** | Minimum Sensing Frequency . |
| **NAT** | Network Address Translation . |
| **ND** | Neighbor Discovery . |
| **NTP** | Network Time Protocol . |
| **OBU** | On-Board Unit . |

| | |
|---|---|
| **OOR** | Ownership Object Relationship . |
| **PaaS** | Platform as a Service . |
| **PD** | Physical Device . |
| **PLC** | Parametric Logic Control . |
| **POR** | Parental Object Relationship . |
| **PSAD** | PubSub-Alerting Discovery . |
| **QoE** | Quality of Experience . |
| **RADA** | Resource Aware Data Accumulation . |
| **RBTP** | Recursive Binary Time Partitioning . |
| **RL** | Reinforcement Learning . |
| **RSU** | Road-Side Unit . |
| **RWA** | Random Asynchronous Wake-up . |
| **RWO** | Real-World Object . |
| **SAD** | Social-Alerting Discovery . |
| **SBD** | Short Burst Data . |
| **SDaaS** | Smart-Device-as-a-Service . |
| **SE** | Social Enabler . |
| **SIoT** | Social Internet of Things . |
| **SIoV** | Social Internet of Vehicles . |
| **SNIP-RH** | Sensor Node Initiated Probing for Rush Hours . |
| **SOR** | Social Object Relationship . |
| **SVO** | Social Virtual Object . |
| **SVOR** | SVO-Root . |
| **SVOS** | SVO Search . |
| **SWIM** | Small World In Motion . |
| **VANET** | Vehicular ad hoc network . |
| **VC** | Vehicle Control . |
| **VO** | Virtual Object . |
| **WSN** | Wireless Sensor Network . |

# Introduction

Society is moving towards an always connected paradigm, where the Internet user is shifting from persons to things, leading to the so called *Internet of Things* (IoT). 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. To address these issues, the idea of using the technologies and results related to the field of social networking in the IoT to enable objects to autonomously set their own social relations has led to the definition of the *Social Internet of Things* (SIoT) paradigm [1]. The main reason is that objects augmented with social capabilities can improve search, selection and composition of services and information, provided by the communities of objects taking part to the IoT. Moreover, Cloud computing can provide the right technologies to implement the infrastructure that meets those 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 [2].

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 implement the digital counterparts of the physical devices, 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 [3] and Paraimpu [4], 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 [5], 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 [6] and Compose [7], 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. The effective design of a Cloud-based platform for the SIoT paradigm has several challenges: (i) *achieving low latency interaction* between physical objects and their digital counterparts, by reducing the round-trip-time in the relevant connections w.r.t. remote cloud hosting; (ii) *scalability* to meet the tremendous increase of traffic generated by the expected huge number of IoT objects; (iii) *developing autonomous social agents* which assist resource-constrained IoT objects to maintain, update, and browse their social relationships in a self-consistent way; (iv) *flexibility* to enable dynamic service provisioning and composition by combining the capabilities of the Social Virtual Object (SVO), also deployed in different Cloud servers; (v) *mobility management* to provide proximity services based on physical device' position by supporting seamless migration of SVOs across geographically distributed Cloud servers.

The idea behind my research activities is the belief that to fully exploit the potentialities of the IoT paradigm, there is a strong need for further advancements in the design of 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; are reliable and efficient to support the interaction of billions of objects. Since the thesis is related to the definition of a cloud architecture for the SIoT and its applications, the chapters are structured as follows.

In Chapter 1, I introduce the SIoT paradigm as background of the whole research activity. Then, each section represent the state of the art for the following chapters.

In Chapter 2, I 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. I 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 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. 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.

In Chapter 3, I address the need of ad-hoc algorithms to detect new relationships in order to grow the social network. The Neighbor Discovery (ND) algorithm is the first step for the establishment of friendship and should allow each object to discover all its neighbors and communicate its presence to every neighbor on the network [8]. To this, three solutions are proposed: the first one relies on the channel scanning; the second one assumes that channel scanning is not possible and makes use of the device localization features; the third one is similar to the second one, but the already existing objects social network is exploited. These algorithm are then integrated in the Lysis platform.

In Chapter 4, I investigate the smart collection of data coming from multiple devices as happens in Mobile CrowdSensing (MCS). I exploit the main features of the Lysis platform for the search of objects to be involved in the sensing, and this also makes easy the integration of the MCS activities into IoT applications introducing the *CrowdSensing as a Service* (CSaaS) model. I propose a new algorithm to address the resource management issue so that MCS tasks are fairly assigned to the objects. Therefore, resources are efficiently managed and no node is overloaded with respect to the others.

In Chapter 5, I introduce the SIoT paradigm to the vehicular environment and its integration to Lysis architecture. Starting from the SIoT concepts, I define some relationships which can be established between the vehicles taking part to the Social Internet of Vehicles (SIoV). Also, I propose an initial definition of the SIoV middleware by extending the functionalities of the Intelligent Transportation Systems Station Architecture (ITS SA) defined by ISO and ETSI standards, to take into account all the needed elements to integrate Vehicular ad hoc networks

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

(VANETs) in the SIoT. Furthermore, I illustrate an effective low-cost and flexible solution for enabling vehicles without any kind of connectivity to participate in the Social Internet of Vehicles. The implemented system is able to collect data from the vehicle's Engine Control Unit (ECU) and to sense the presence of nearby vehicles. This information is then sent to Lysis platform hosted in the Cloud, where friendships are created and managed based on the vehicles' behavior and an application for the remote monitoring of the vehicle is implemented.

In Chapter 6, as extension to the use of conventional cloud computing infrastructures, I investigate on the changes consequent to the use of edge cloud technologies, which require the capability to detect the need for a change in the geographical location of the virtual object and to handle the inter-cloud mobility of related processes and data.

Finally, in the last chapter, conclusions will be drawn regarding the effectiveness of the proposed solutions, and some possible future challenges will be sketched.

# Related papers

- **Roberto Girau**, Salvatore Martis, and Luigi Atzori. "A Cloud-Based Platform of the Social Internet of Things." *Proc. EAI Intl Conf. Cyber Physical Systems, IoT and Sensors Networks.* 2015.

- **Roberto Girau**, Salvatore Martis, and Luigi Atzori. "Lysis: A platform for IoT distributed applications over socially connected objects" *IEEE Internet of Things Journal* (2016).

- **Roberto Girau**, Salvatore Martis, and Luigi Atzori. "Neighbor Discovery Algorithms for Friendship Establishment in the Social Internet of Things." *IEEE 3rd World Forum on Internet of Things (WF-IoT)* 2016

- Luigi Atzori, **Roberto Girau**, Salvatore Martis, Virginia Pilloni, and Marco Uras. "A SIoT-Aware Approach to the Resource Management Issue in Mobile CrowdSensing." *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)* 2017

- Michele Nitti, **Roberto Girau**, Alessandro Floris, and Luigi Atzori. "On adding the social dimension to the internet of vehicles: Friendship and middleware" *Communications and Networking (BlackSeaCom), 2014 IEEE International Black Sea Conference on.* IEEE, 2014.

- Ivan Farris, **Roberto Girau**, Leonardo Militano, Michele Nitti, Luigi Atzori, Antonio Iera, and Giacomo Morabito. "Social Virtual Objects in the Edge Cloud" *IEEE Cloud Computing 2.6* (2015): 20-28.

- Ivan Farris, **Roberto Girau**, Michele Nitti, Luigi Atzori, Roberto Bruschi, Antonio Iera, and Giacomo Morabito. “Taking the SIoT down from the Cloud: Integrating the Social Internet of Things in the INPUT Architecture” *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on.* IEEE, 2015.

# Chapter 1

# Background

The intention of this section is to briefly review the major works in the related areas of interest for each chapter. In Section 1.1 I describe the paradigm of the Social Internet of Things, which is the reference background of my research activities. In Section 1.2 the state of the art of the IoT architectures over cloud infrastructures in order to highlight the strategic requirements that led me to the design of my solution. In Section 1.3 I show the state of the art of the device neighbor discovery in IoT, whereas in Section 1.4 previous works in Mobile Crowdsensing are described. In section 1.5 and section 1.6, I provide a state of the art of the Internet of Vehicles and of the fog computing respectively.

## 1.1 The social Internet of Things

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 [9]. For instance, in [10] the authors introduced the idea of objects able to participate in conversations that were previously only available to humans. Analogously, the research activities reported in [11] consider that, being things involved into the network together with people, social networks can be built based on the Internet of Things and are meaningful to investigate the relations and evolution of objects in IoT. In [12], 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 work, 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:

- *Ownership Object Relationship* (OOR): is created between objects that belong to the same owner;
- *Co-location Object Relationship* (CLOR): is created between stationary devices located in the same place;
- *Parental Object Relationship* (POR): is created between objects of the same model, producer and production batch;
- *Co-work Object Relationship* (CWOR): is created between objects that meet each others at the owners' workplace, as the laptop and printer in the office;
- *Social Object Relationship* (SOR): is 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, 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 OOR and POR relationships), its own movement patterns (this is the case of the CLOR relationship) and the context of the device (CWOR and SOR relationships). 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.

## 1.2 State of the art of IoT architectures and platforms

### 1.2.1 Distributed social objects

[13] is a past implementation of the SIoT paradigm developed starting from the open source project ThingSpeak[1]. 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 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.

### 1.2.2 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 physical devices. Most of these issues can be addressed through the virtualization layer [14], which is where the virtual object implements the digital counterpart of the physical devices, 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 inter-objects 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 [15]. In the Compose platform, virtual representations of physical objects are named Service Objects, which support the handling of communications, the processing of sensor data,

[1]https://thingspeak.com/

and the description of objects' characteristics to support semantic discovery mechanisms [7]. 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 [16] [6] virtualization is a major feature. The proposers define the Virtual Object (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 Virtual Objects (CVOs) implement complex services that are re-used 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 Internet of Things (CIoT) paradigm [17]. Herein, the authors state 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, virtual objects are interconnected and act as agents with minimal human intervention, interact with each other by exploiting the context-awareness, storing and learning the acquired knowledge, and adapt themselves to situations through efficient decision-making mechanisms.

### 1.2.3 PaaS in IoT

Many platforms exploit cloud computing technologies to provide IoT services in different environments, such as smart home [18], smart cities [19], smart management of inventories [20], eHealth [21] [22], environmental monitoring [23], social security and surveillance [24], mine security [25], Internet of Vehicles (IoV) [26]. Although very effective for the purpose they have been proposed, these solutions are most of times vertical implementations, lacking in horizontal enlarge-ability 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 [27], where the authors highlight 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* to make the different PaaS IoT platforms talk each-other. This issue is also the focus of Gubbi et al. 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 [28].

## 1.3 Neighbor discovery in IoT

The Neighbor Discovery is traditionally analyzed in Wireless Sensor Network (WSN) area to solve energy problems [29]. During the deployment phase, it is evident that if the devices were kept always active and listening, it would waste a considerable amount of energy just to accomplish the network topology. So, initially, the research has focused on how to save energy in the process of discovery by finding a compromise with the latency (the time it takes to be aware of the presence of another device). The alternating between sleep and awake states of the radio interfaces has been the focus of several strategies for neighbor discovery.

In the definition of Neighbor Discovery strategies we must address several issues. Firstly, the ability of an algorithm to recognize the presence of other nearby devices. It involves a continuous adapting of the radio resources in order to find the neighbors while staying within a time frame and under contact conditions that vary dynamically. Conversely, figuring out if there are no devices in the vicinity saves energy instead of wasting it in useless discovery operations. Secondly, the use of pattern recognition methods for mobility allows for capturing spatial and temporal characteristics [30] [31]. Other models use metrics of popularity of the visited places, social behavior, tagging places such as roads, schools, airports etc. to better identify mobility patterns [32] [33]. Finally, the problem of the acquisition of information related to future encounters not only in terms of storage but also in terms of understanding them, requires more efforts in research to allow for a greater degree of predictability of the movements and allow for better management of energy resources. Keeping in mind these issues, the devices can be kept operational for a longer time so as to increase the time of better planned communications D2D.

Generally, in ND protocols, all the time-related parameters (arrival/departure time, contact time, inter-contact time) can be managed either synchronously or asynchronously. In ZebraNet [34] the GPS time is used to allow for synchronization of the nodes. In [35] a random sensing technique based on time synchronized using a Markov chain is proposed to optimize scheduling between active and dormant states. In *Recursive Binary Time Partitioning* (RBTP), the authors claim to reduce the latency of discovery between smartphones using the NTP protocol for synchronization. In *WizSync* [36], IoT devices with ZigBee interface can take advantage of the overlap in the radio bands at 2.4 GHz of unlicensed spectrum to achieve synchronization via Wi-Fi beacons. The asynchronous protocols do not require the presence of an accurate time reference to do the discovery between neighboring devices. Usually, they use indirect requests or exploit the overlapping of the time slots to activate the discovery. In *Sparse Topology Energy Management* [37], the authors propose the use of two radio interfaces operating in parallel and at different frequencies, one for the communication channel and one for the wake-up signals. In [38], it is brought an improvement to the concept of “Wake on Wireless” via a platform that uses Wi-Fi, Bluetooth and ZigBee in various combinations on the assumption that low-power radio systems combined with other high-power ones are able to optimize the discovery and communication. In ZiFi [39] signs of interference are used in ZigBee to deduce the presence of Wi-Fi access point. In *Random Asynchronous Wake-up* (RWA) [40], randomization in dense scenarios allows for the maximization of the probability that the nodes can see each other. Choi et al [41] presented a hierarchical and adaptive approach based on difference sets that allow for a choice among different levels of energy efficiency. Bacht et al [42] with *Searchlight*, has proposed a new protocol which can obtain discovery by delay constraints through deterministic search sequentially the time slots. In [43], assuming that the contact time follows a power law distribution, authors show that the probability of contact failure does not depend on the reducing the duty cycle if $T_{ON} \geq T_{OFF}$ and $\tau \geq 2(T - T_{ON})$, where $T$, $T_{oN}$ and $T_{OFF}$ are respectively the period, the sleep and awake time, and $\tau$ is the minimum duration of contact. Other algorithms instead take into account the knowledge of the movements of nodes to decide the schedule and can be based on arrival times and on the encounter rate between nodes. In [44], it is defined an efficient communication protocol that leverages the knowledge of repetitive and predictable movements such as those of the public buses. The authors show that if the spatial distribution of the nodes has a guaranteed minimum distance of separation, then the probability of meeting failure is zero. In [45] instead, the authors introduces a framework for the energy management based on knowledge of contacts between IoT objects, through an approach on “zero knowledge” which adopts a beacon strategy for synchronous systems. In [46], the *Reinforcement Learning* (RL) is exploited to allow for collecting online data on mobility patterns. *Resource Aware Data Accumulation* (RADA) [47] leverages a framework based on the Q-Learning for efficient discovery in terms of energy with fixed devices capable of learning the presence of mobile devices. *Sensor Node Initiated Probing for Rush Hours* (SNIP-RH) [48] proposes an algorithm that exploits the knowledge about the rush hours of the day to concentrate efforts and energy in the planning of discovery when it is more likely the interaction between objects.

## 1.4 Mobile CrowdSensing

Data acquisition from multiple sensors can be achieved either in a participatory way, where users actively contribute to the sensing task (e.g. taking a picture, twitting an earthquake), or in an opportunistic way, where sensing tasks are automatically performed by devices without the explicit involvement of users [49] (e.g. temperature monitoring, geolocation). The acquired data can be shared with other devices or delivered to the cloud, where it can be further aggregated to get what can be defined as crowd wisdom or collective intelligence. This is the principle that lies behind Mobile CrowdSensing (MCS), a pervasive sensing paradigm where users' mobile objects can often replace static sensing infrastructures. MCS is therefore considerably advantageous for applications where user perspective is crucial, such as healthcare analytics [50], route planners [51], social-related applications [52]. In the MCS paradigm, the objective is that of performing an application gathering data from a number of devices that are capable of fulfilling the application requirements. These devices can either contribute to the application execution in a participatory way or the system can collect data from them in an opportunistic way [53].

A huge number of MCS-based applications have been proposed in the literature. Thanks to its characteristics, the MCS has proven to be particularly advantageous for, but not limited to: ambient monitoring [54] [55], pattern recognition, mainly for healthcare analytics [50] or smart city and route planning applications [51], and social-related applications, where users share and compare data about themselves with a community [52] [56].

Compared to traditional sensor networks, MCS entails significant advantages, mainly related to deployment costs and coverage extension [49]. Indeed, in MCS data can be collected by any sensor that meets the application requirements, i.e. that is in the right place at the right time and which information quality requirements are satisfactory. Therefore, spending money and time in deploying new infrastructure to cover new areas is not needed, because the existing one (e.g. WiFi, 3G, Bluetooth) can be used.

Nevertheless, MCS inherited one of the major issues of traditional sensor networks: the limited amount of resources available on devices [53] (e.g. available energy, storage capacity, computing speed). Fair resource allocation mechanisms can be applied to overcome this problem. Some examples are provided by the studies presented in [57] [58] [59]. The Mobile Sensor Data EngiNe (MOSDEN), a framework to discover and efficiently configure and manage sensor nodes, is proposed in [57]. In MOSDEN, schedules, communication frequencies and sampling rates are assigned to sensor nodes by the framework, based on the context. A similar approach is used in [58], where particular attention is put on satisfying global sensing coverage requirements and avoiding redundant sensory activities. In [59], the concept of social vector, where node attributes are defined, is introduced. Social vectors are used to find the most accurate matching between a task with its requirements and a node with its resources. However, none of the analyzed strategies exploits social relations among objects to improve the system performance.

## 1.5 VANET and IoV

Vehicular Ad-hoc NETworks (VANETs) are particular Mobile Ad-hoc NETworkss (MANETs) in which nodes can be mobile or static. Mobile nodes are vehicles while static nodes are Road-Side Unit (RSU). In VANETs, the vehicles are equipped with an On-Board Unit (OBU) through which they can communicate with other vehicles (Vehicle-to-Vehicle or V2V communication) and with RSUs (Vehicle-to-Infrastructure or V2I communication) by employing wireless short-range protocols as IEEE 802.11p. Furthermore, the vehicles and/or the RSUs can be able to connect to the Internet by using mobile cellular systems.

In the last years, VANETs have been rapidly evolving and new scenarios have been coming out. As an example, the integration of the concept of the Internet of Things (IoT) with the VANETs has brought to a novel paradigm, namely the Internet of Vehicles (IoV);

Most state-of-the-art works considered VANETs for different applications which can be mainly classified as active road safety applications, traffic efficiency and management and infotainment applications [60]. Since VANETs are based on V2V and V2I communications, which employ wireless short-range protocols, information dissemination is essential to efficiently transmit the messages among the vehicles. As a consequence, most VANET routing protocols proposed in the literature are focused on disseminating information by exclusively exploiting wireless short-range communications. A classification of the proposed VANET routing protocols is provided in [61, 62].

Since I am interested in the design of a IoV, which needs vehicles to be durably connected to the Internet, I focus on state-of-the-art works which considered mobile cellular systems in VANETs in addition to V2V/V2I wireless short-range protocols. A work which considered a mobile cellular system to deliver packets in VANET is [63], where it is assumed that vehicles are equipped with both 3G radio and short-range radio for V2V and V2I communications, so that each vehicle can directly deliver a packet via 3G or via multi-hop transmissions.

The other state-of-the-art works which made use of mobile cellular systems in VANETs, as [64, 65], perform clustering of vehicles and select a minimum number of suitable vehicles (chosen among "Gateway Candidate nodes", which are the vehicles equipped with a mobile cellular system) that will serve as gateways for the clusters and will create some points of attachment to the 3G network.

Finally, in [66], the authors propose to integrate current 3G/4G systems with VANET networks to enable infotainment services offered in push mode and to compensate for the scarce service coverage initially offered by the vehicular communication systems. Two different scenarios have been proposed: smartphone-centric and OBU-centric. In the first one, the client application is installed in the smartphone, while the OBU is used essentially as a wireless router, bridging the inside car world and the outside world via the IEEE 802.11p, and acting as a WiFi hot spot for the smartphone. In the second one, the OBUs are the natural end-point of services. The device installed on board has a complete stack, up to application layer, and has a monitor and input/output audio and video facilities.

## 1.6 Cloud computing and fog computing for IoT

The recent integration of Cloud computing features in the IoT landscape has represented a major breakthrough toward the deployment of sophisticated applications, requiring heavy processing natively not supported by resource-constrained IoT devices. In such a context, data generated by IoT devices can be sent and processed in the cloud [67], supported by the large amount of computational and storage capabilities available in datacenters.

A further leap forward in this domain is represented by the introduction of solutions which envision the instantiation of digital representatives (“virtual objects”) of the physical objects in the cloud [68], the so-called Cloud of Things. In this way, added value applications can be implemented by exploiting services offered by virtual objects. These applications can be developed independently of the specific hardware features of the IoT devices.

However, for applications with strict requirements in terms of delay, the latency caused by interactions with/between virtual objects in the cloud are unacceptable. An emerging trend leverages on the new concept of edge cloud, allowing to move cloud services much closer to end-users’ devices by introducing intelligence and flexibility into network edge nodes able to host cloud applications [69]. These applications will be capable of cooperating with and offloading corresponding applications residing in the users’ smart objects and in conventional cloud centers, to offer innovative services. This will allow user requests to be manipulated before crossing the network towards data centers in ways that enhance the performance. Manipulations include pre-processing, decomposition, and proxying. This solution perfectly matches the promising Cloudlet and Fog computing paradigms [70] and is testified to also by European projects, such as TROPIC and INPUT. According to the Fog computing paradigm, network edge devices are evolving into micro Cloud servers, able to host not only advanced network functions but also application modules. This allows for instantiating distributed functionalities, accommodating the desired level of QoE (Quality of Experience) and workload/traffic volumes, closer to the end-users. Furthermore, the distributed architecture composed by large-scale geographically deployed nodes natively possesses *scalability* properties.

# Chapter 2

# Lysis: leading requirements, architecture and implementation

This chapter presents Lysis, which is a cloud-based platform for the deployment of Internet of Things applications. The major features that have been followed in its design are the following: each object is an autonomous social agent; the PaaS (Platform as a Service) model is fully exploited; re-usability 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 re-usability 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 objects owners cloud spaces.

## 2.1 Motivation and Strategic Requirements

By analyzing the state of the art of the IoT architecture (see Chapter 1, Section 1.2), some peculiar features arise as strategic for the design of a IoT architecture:

- distributed social virtual objects as main actors of the IoT environment;
- the PaaS model must be fully exploited;
- re-usability at different levels;
- the data is under full control of the users

*Distributed social objects* Whereas the general SIoT features have been implemented in a previous platform, it makes use of a centralized approach, where the objects do not communicate directly with each other, but through the server, also to establish social relationships. Hence, 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 the performance evaluation section.

*Virtualization* The virtualization layer allows for addressing the issues of navigability, management, ubiquity, scalability, reliability, and sustainability of the network and offered services in a heterogeneous environment made of huge numbers of objects as it is the Internet of Things. In the virtualization layer, the virtual object implements the digital counterpart of the physical devices, 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 inter-objects communications possible by translating the used dissimilar languages.

There are two major contributions of my work in this area. Firstly, to my knowledge the state of the art misses a platform that fully implements the complete functionalities of virtual objects for IoT. Secondly, I extend these functionalities by creating the Social Virtual Object, where the social behavior is injected into the VOs.

*PaaS in IoT* According to what shown in Chapter 1 Section 1.2.3 , many of cloud IoT solutions are most of times vertical implementations, lacking in horizontal enlarge-ability 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. Differently from these past works, in Lysis I fully exploit the PaaS model for the implementation of a cross-application IoT platform.

*Data ownership* In the near future, IoT 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, I have designed my platform so that the data generated by the devices is kept in the owner cloud space.

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

*Re-usability* A prerequisite to ensure efficiency in IoT platforms is related to the re-usability 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 virtual objects can be aggregated to provide a service, which can be needed by multiple applications. Also in this case, the composition of virtual object must have the necessary intelligence to adapt to multiple requests of the same pattern. Decisions on which simple virtual object services to aggregate should be taken on the basis of the context. 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, I have designed my platform so that the data generated by the devices is kept in the owner cloud space. In this way the produced information remains in the hands of the user that can dispose of this data at her will.

## 2.2 The overall Lysis architecture

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

In the following subsections I briefly describe the major components, whereas more details with reference to the introduced novelties are provided in the next section.

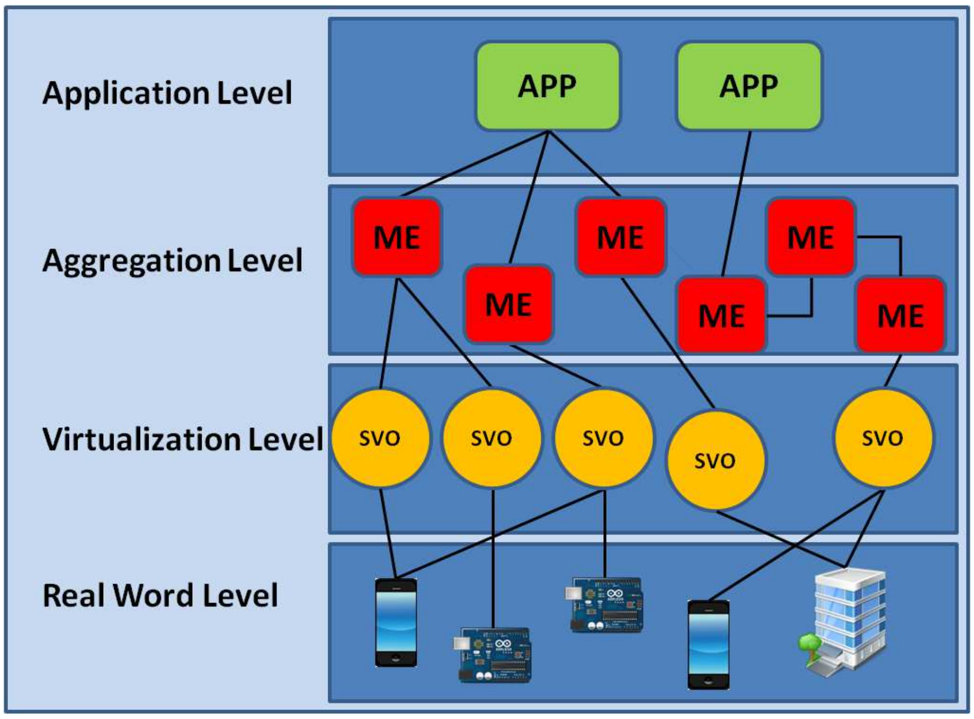


Figure 2.1: The four levels of the Lysis architecture.

### 2.2.1 The Real World level

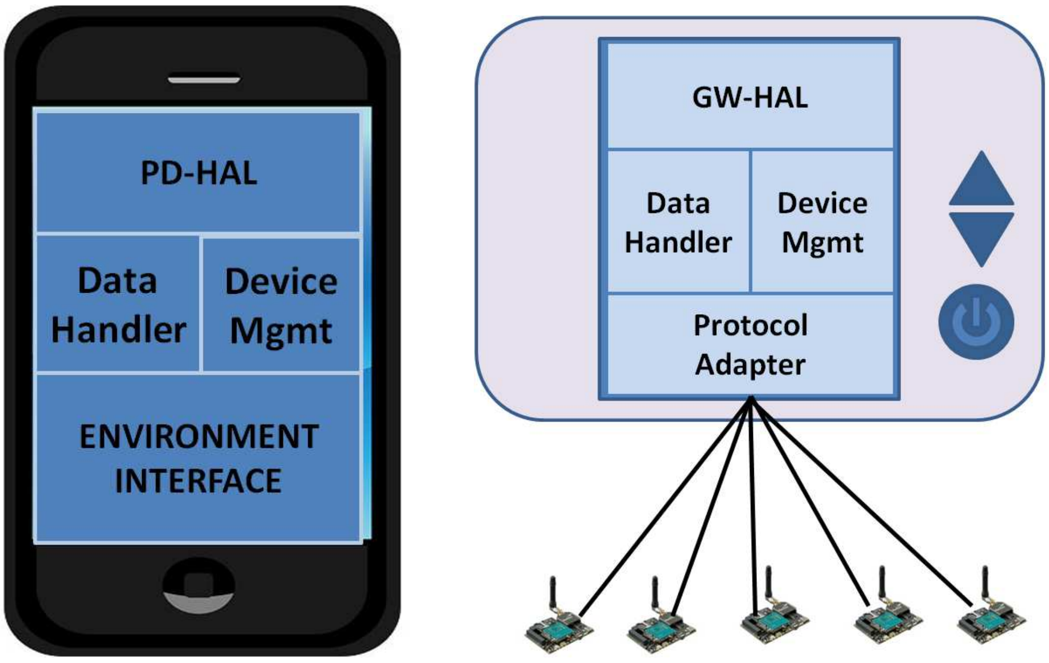
As well-investigated in the iCore project [6], the lowest level is always made up of the RWO. Some of these are Physical Devices (PDs) able to directly communicate through the Internet, such as smartphones, laptops and TV set-top-boxes (see Fig. 2.2a). Some others cannot directly access to the internet and have to use local gateways (GWs) (see Fig. 2.2b).

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

- Hardware Abstraction Layer (HAL): 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.
- Data Handler: it intervenes whenever there is the need to process data from sensors before being sent by the PD-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.
- 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, overseeing the energy consumption. It also runs the code that can be updated in run-time locally in the PD.

- 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.



(a) (b)

Figure 2.2: (a) Physical Devices (PDs) able to communicate with the platform and (b) objects that need a gateway to interact with the platform.

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 HTTP (Hypertext Transfer Protocol) and CoAP (Constrained Application Protocol) are unusable if behind a Network Address Translation (NAT) 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. Firstly, 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 [70], 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.

Secondly, 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 (SBD) 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 pre-processed giving a limited view of the device capabilities.

### 2.2.2 The 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 virtual object 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.

In my solution this layer assumes an even more important role as it is augmented with the social behavior, bringing to the concept of Social Virtual Object (SVO). When designing the SVO I had to identify the functionalities that were in common with all the SVOs independently from the physical device characteristics and define the Social Enabler, as shown in Fig. 2.3. The other functionalities, which are mostly affected by the specific device characteristics, are implemented by the Virtual Object 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.

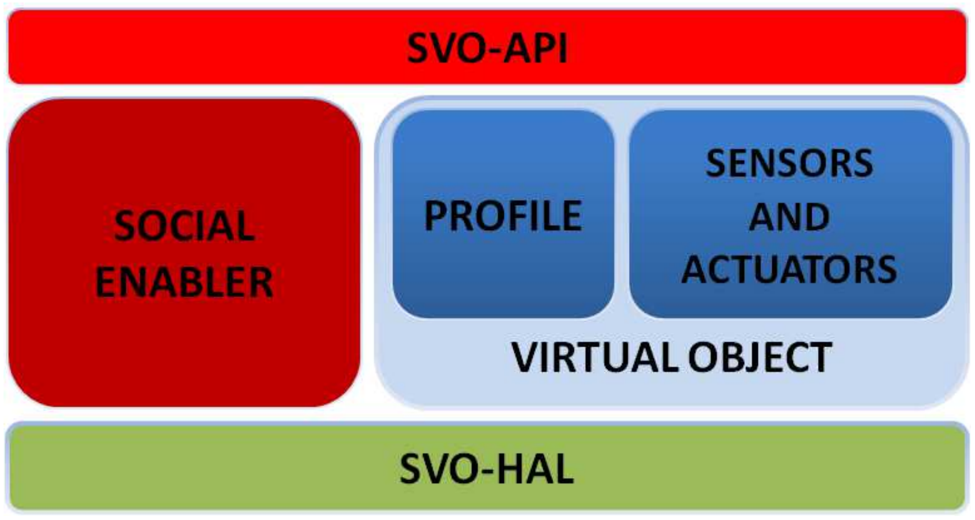


Figure 2.3: The components of the Social Virtual Object

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 given by 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 IFTTT (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.

### 2.2.3 The 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: the *Instance* and 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 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.

### 2.2.4 The Application Level

At this level the applications are deployed and executed exploiting one or more Micro Engines. 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.

## 2.3 Addressing the key requirements

In the following I describe the choices that have been taken to satisfy the requirements of Section 2. I also analyze the trustworthiness and security issues.

### 2.3.1 The 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 [71]. Fig. 2.4 shows the modules implemented in the SE.

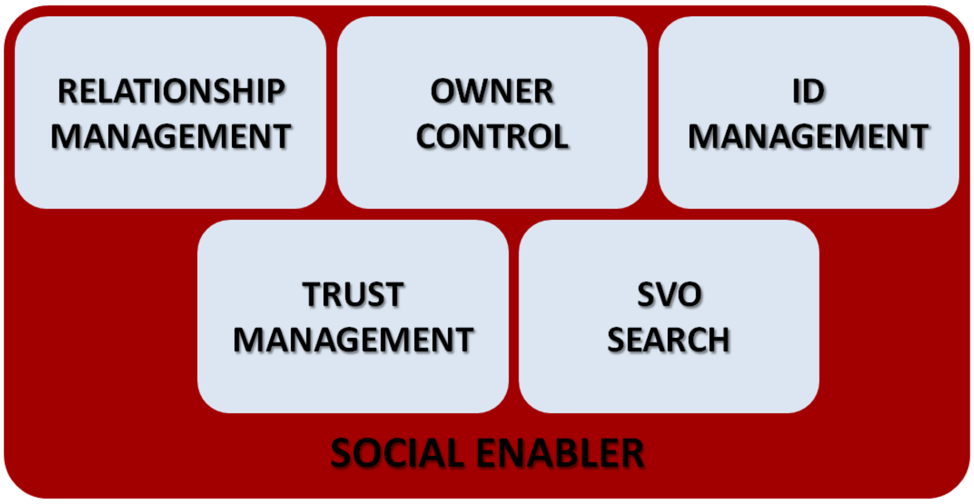


Figure 2.4: The components of the Social Enabler

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 me 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 *Social Virtual Object Search* module implements an algorithm of SVO search 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: public, private and 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. Lastly, 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 the 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: *Hard Binding*, when the association is set up between PD-HAL and SVO-HAL; *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 and with SVO Search (SVOS). The algorithm related to the former has been discussed in details in [71] and incorporated in this Lysis platform. The latter is discussed herein. SVO search is the functionality the application layer is 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. 2.5. The process is repeated until a positive result is found, which is then returned to the SVOS

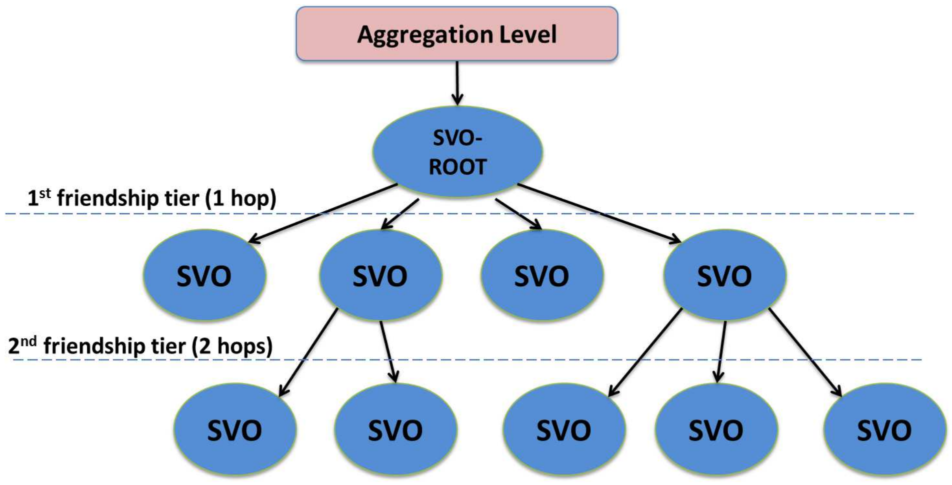


Figure 2.5: Objects involved in a SVO search process

that sends it to the higher levels. Since each SVO is able to respond to SVO Search 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 towards alternative routes of the social graph. In addition, using the SVO with greater centrality decreases the chance of forwarding the request outside the SVO Root.

### 2.3.2 The 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.

*SVO Deployment* To deploy SVOs in the cloud, Lysis provides the infrastructural elements shown in Fig. 2.6. From the Template repository the user chooses the correct Template for the

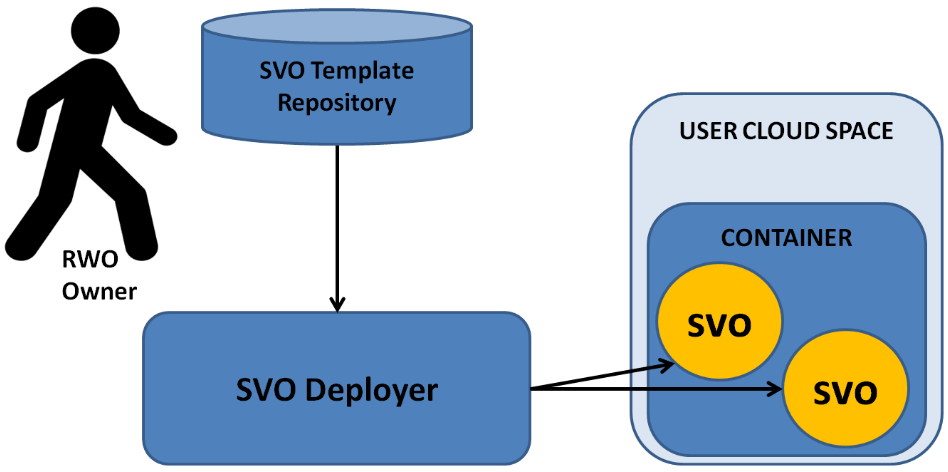


Figure 2.6: The SVO deployment process

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 the 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.

*ME Deployment* Fig. 2.7 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, 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 row 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 Micro Engine Controller (MEC) 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 SVO Root of the user who started the application from which it takes the owner key. Once it has the

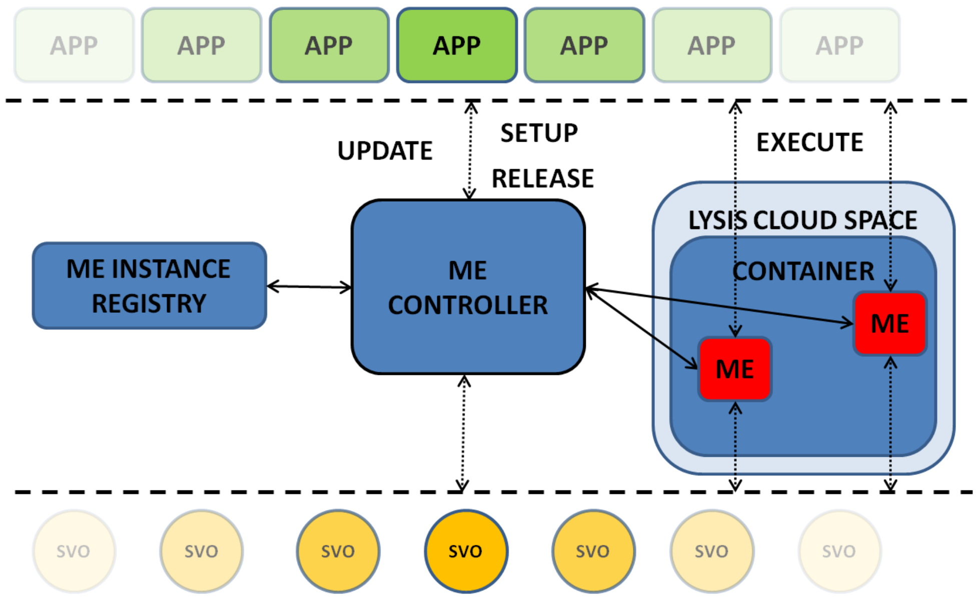


Figure 2.7: The ME deployment process.

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. 2.8 shows all the steps that lead applications to run in

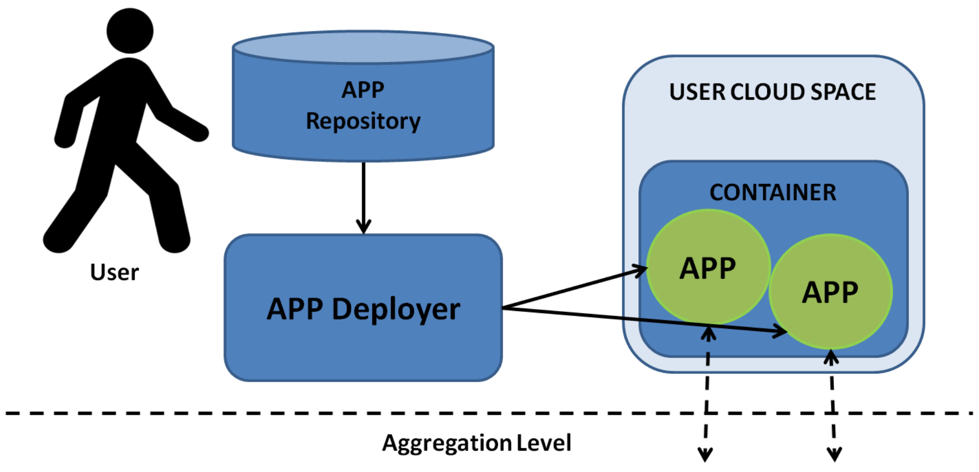


Figure 2.8: The application deployment process

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.

### 2.3.3 Re-usability

The Micro Engine, 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 Micro Engines 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 I foresee two development environments: a scripting environment and 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 SVO, 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.

### 2.3.4 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 Lysis's 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 Deployer 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).

### 2.3.5 Security and trustworthiness

Security issues have to be analyzed at both southbound and northbound SVO interfaces. As to the former, as seen in Section 2.2.1, 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 multi-provider 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 [72]

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 trust management 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. In [71] authors show 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, the use of the SVO distributed approach and of the PaaS technologies partially limits this issue.

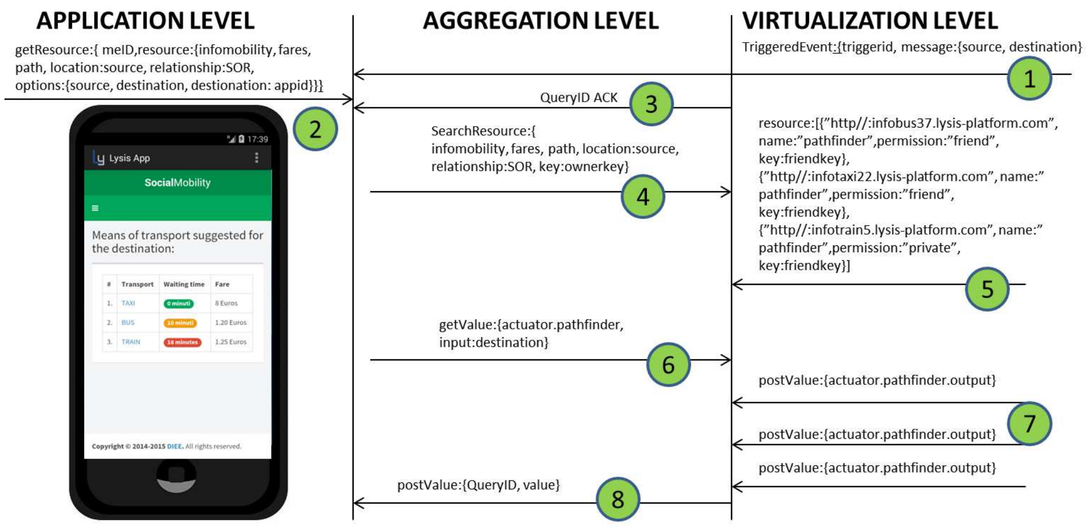
## 2.4 Use-case and performance evaluation

### 2.4.1 Lysis implementation

To implement Lysis, I 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 [73]: 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.

### 2.4.2 Analysis of an use-case

In this section, I 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, 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 a 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


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
QueryID ACK
3
2
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}
7
postValue:{actuator.pathfinder.output}
postValue:{actuator.pathfinder.output}
postValue:{QueryID, value}
8
17:39
Lysis App
SocialMobility
≡

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 |

Copyright © 2014-2015 DIEE. All rights reserved.

Figure 2.9: The SocialMobility application: the execution process

installed, the process follows the steps shown in Fig. 2.9. A trigger set on 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 SVO Root of the user (4). The information is available since the user smartphone has a SOR relationship with the airport infopoint, which has CLOR relationships with the bus infopoint, taxi infopoint and train infopoint. The SVO Root 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 I 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 1 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 2.2, 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 2.3 and by exploiting the Google API, user applications and SVOs are deployed on the user cloud space granting data ownership to the user.

### 2.4.3 Performance analysis

Two important aspects of the proposed platform are the scalability and the reduction of computation overhead at the physical devices thanks to the use of the virtualization layer. I 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 I created a network of social objects with synthetic data as done in past studies [71] [74]. In these experiments I considered two types of social graph: a) 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); b) 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, I 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 a NoSQL DB without social links between the nodes, as in the ThingSpeak solution, which are taken from [75]. Fig. 2.10 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

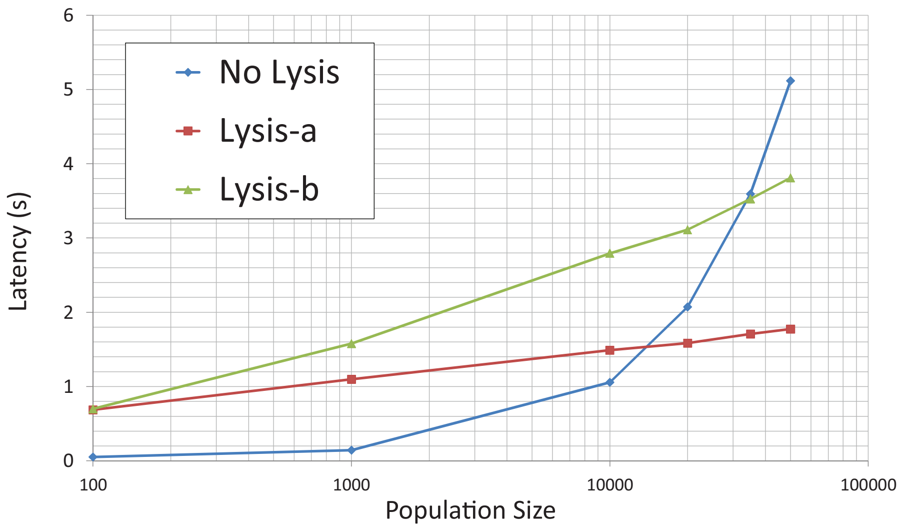


Figure 2.10: Information search latency at increasing size of the IoT population when using Lysis with a constant number of friends (Lysis-a), with a number of friends that increases with the population (Lysis-b), and with a conventional IoT solution (no-Lysis)

for the platform scalability, as studied in [74]. 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.

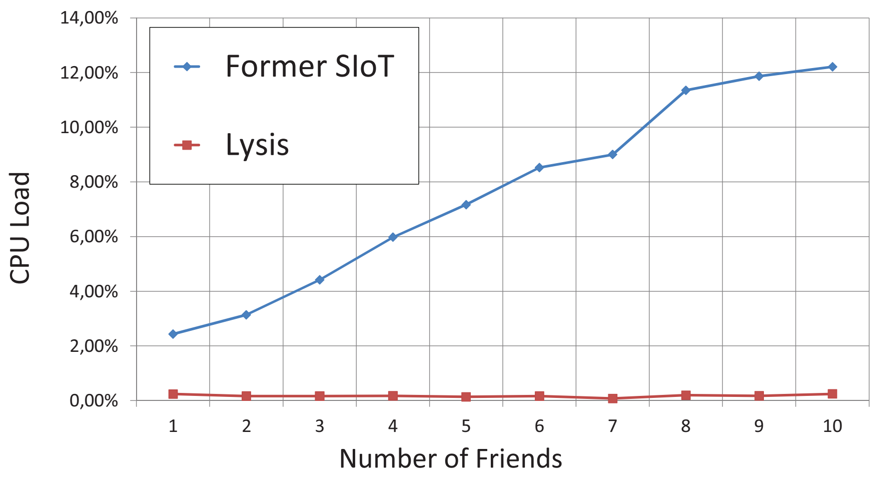


Figure 2.11: Device CPU load when executing a crowdsensing application with the former SIoT and Lysis platforms

The computation overhead at the physical device is show in Fig. 2.11, 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 virtual object 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 physical devices.

## 2.5 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. [76] 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 my proposed solution with respect to the existing platforms in a concise yet effective way, herein I concentrate on the major requirements I have taken as the starting point of my work and I have made a comparison accordingly with five alternative solutions as representative of different categories. The final results are shown in Table 2.1, which are commented on the following.

Table 2.1: Requirement-based comparatives 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 |

iCore [6] is an European FP7 project which gave a first extended definition of Virtual Object, as before the virtualization was only a marginal feature just related to the abstraction of physical device functionalities. Indeed, in iCore there has been a significant work on the definition of the interactions between virtual objects, 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 [3] (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 re-usability of the software, it is only available at the driver level so that the community can share the firmware for common devices. Instead, re-usability 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 [77] in order to develop globally recognized technical specifications for a relevant service common level. In the virtualization layer, resources are uniquely addressable entities in a RESTful architecture. The specifications do not refer to any particular implementation approach and there is no reference to the data ownership.

Paraimpu [4] 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 [13] was created as an add-on to the ThingSpeak solution, introducing the object social behaviour as a functionality of the central server. In this implementation, the virtual objects 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.

# Chapter 3

# SIoT Enabling Technologies: Neighbor Discovery Algorithms for Friendship Establishment in the Social Internet of Things

In the Social Internet of Things (SIoT), objects establish social relationships to create a social network of devices that fosters inter-object communications by improving network navigability and trustworthiness evaluation. An important process of this model is the neighbors discover that should be implemented by each object to create new relationships. Differently from past neighbors discovery works, in the SIoT scenario, the devices need to discover each other when they are assumed to be already able to communicate to the Internet and additionally they do not have to set up direct device-to-device communications after the discovery.

The chapter is organized as follows. In Section 3.1 I describe the related guidelines that drove my design of new algorithms with respect to existing solutions, while in Section 3.2 I show my solution and in Section 3.3 a performance evaluation of the proposed models.

## 3.1 Key Aspects of Neighbor Discovery Algorithms

By analyzing the state of the art on ND algorithms, I am able to identify the major features that characterize the background in which the above mentioned algorithms are thought (Fig. 3.1a):

- Radio Channel Scan: all devices are able to (and have to) scan the radio channel for neighbor discovery.
- ND for device communication: the goal of ND algorithms is always a reliable

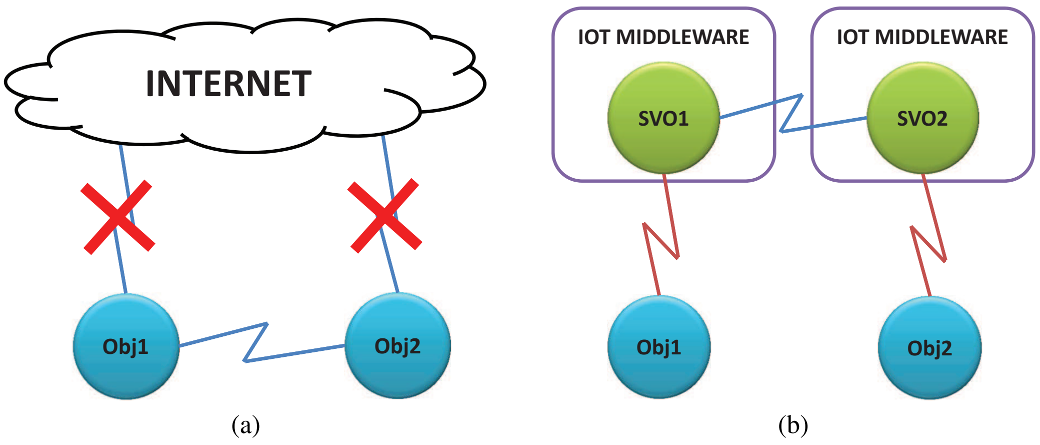


Figure 3.1: Different scenarios of applicability of ND algorithms: standard scenario in ad-hoc, opportunistic and vehicular networks (a); scenario related to friendship establishment in SIoT (b)

and durable communication between the discovered devices through the radio channel.

- No middleware support: applications at higher level are not allowed for accessing the list of retrieved neighbor devices.
- Low level access to radio interfaces: ND algorithms are thought to work in the radio chipset firmwares, and usually at MAC level.

In SIoT applications, I had to face other issues related to a different scenario (Fig. 3.1b):

- Heterogeneous communication interfaces: the IoT includes a wide variety of devices with different hardware components and interfaces often using different radio channels and protocols.
- D2D communication not required: the only requirement is the communication to the Internet in order to access to the IoT virtual counterparts.
- Middleware support: the list of neighbor devices has to be known through middleware APIs to evaluate the friendship accomplishment.
- Standard Control of radio interface: low level access to the radio interfaces is not required.

A consequence of these new requirements, a new approach has to be adopted in order to design new algorithms needed for friendship detection in the SIoT scenario.

## 3.2 Neighbor Discovery Algorithms in SIoT

As already mentioned, Lysis platformis taken as reference solution for the implementation of the SIoT. Accordingly, every device in Lysis has an autonomous virtual counterpart in the cloud named Social Virtual Object (SVO). When two SVOs have established a relationship, they can easily exchange information and resources. As described in Chapter 1, two relationship types (i.e., SOR and C-WOR) depend on the frequency and duration of meeting events between objects. As a consequence, the ND procedure becomes of primary importance. Unlike in opportunistic networks where the ND is necessary as a first step before they can communicate, the ND in SIoT is aimed at the construction of the social graph when the devices are already capable to communicate to the Internet. Among the devices to be considered, we have to make a distinction depending on the level of access to the hardware granted by the respective firmware. To one category belong the devices that may not have either the permissions to perform a scan of nearby devices (i.e., Apple IOS devices) or to know their own MAC address (i.e., devices with Android OS version 6). This category also includes the devices that do not have wireless interfaces, but can provide information about their location using GPS systems. The other category of devices is that of those that have complete control of the wireless interface. To consider the scenarios with these two kinds devices, I propose three ND algorithms aimed at creating social relationships between objects.

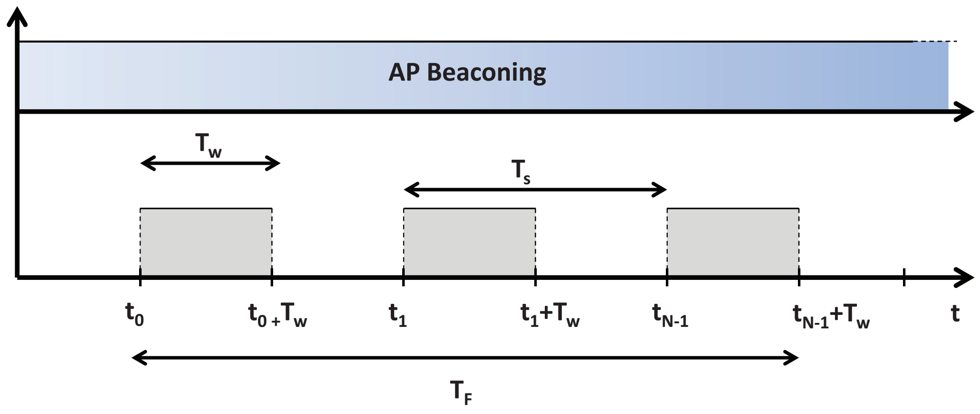


Figure 3.2: Time parameters involved in the neighbor discovery process needed for friendship creation

Despite in our scenario the physical devices do not have to scan interfaces to find neighbors, the SVOs make the actual decisions about scan frequency and duration, and implement the rules for the creation of the relationships. The friendship creation algorithm executed by the SVO requires two devices to be in continuous visibility for a minimum time $T_f$, after which each SVO asks to the other to start a new friendship. To verify this condition, the visibility sampling performed by the radio interfaces must be at the most equal to a value of $T_s$, which is the inter-scan time. Therefore, it seems clear that we must take into account these constraints to evaluate the duty cycle of the radio interfaces. The inter-scan time $T_s$, is determined by the friendship rules, and in the presence of nearby devices must be

$$T_s = \frac{T_F}{N} \tag{3.1}$$

where $N$ is the number of meetings (samples) within the friendship time $T_F$. For example, if the SOR relationship requires 30 minutes of visibility to be established and six encounters, then it has to be $T_s = 5$ minutes. As shown in Fig 3.2, the duration of the discovery is $T_w$, and depends on the radio media (for WiFi it is about 1 second, for Bluetooth it is up to 10 seconds) and usually it is negligible compared to friendship time (30 minutes) but affects the inter-scan time since it has to be $T_s \geq T_w$. In the following three subsections I present the proposed ND algorithms. The local neighbor discovery aims at exploiting the presence of wireless reference points to detect encounters and visibility between objects. The discovery by PubSub aims at allowing objects to share information about the position through systems of publish/subscribe in order to detect ephemeral events of collocation. The same goal is pursued by the discovery through social alerting, which exploits the social graph to spread information about the objects' positions.

### 3.2.1 Local neighbor discovery

The Local Neighbor Discovery (LND) algorithm can be implemented by devices able to scan a radio channel to look for other devices that transmit beacons, such as: WiFi Access Points, Bluetooth beacons for indoor navigation or for advertisement, mobile devices in hotspot mode. The basic approach behind this algorithm is the use of one of these devices as *reference point* (RP). The objective is then to detect the situations where two devices are under the same radio coverage of the same RP. Such reference point devices do not need to be registered to the SIoT platform, since the RP's ID (e.g., MAC Address and SSID) is achievable in the beacon message and it is not required an object-RP interaction. In such a way, by means of this approach it is possible to exploits the high concentration of RPs especially in the urban environment. To better explain the algorithm we consider the scenario depicted in Fig. 3.3 and described in the following. Let us consider the case of two devices, Obj1 and Obj2, both registered on the Lysis platform. These devices have unique IDs generated at the time they have been associated with the respective SVOs (say ID1 and ID2). Let us suppose Obj1 enters into the radio range of an reference point RP which is identified by the MAC address (RP-ID) (1) and where Obj2

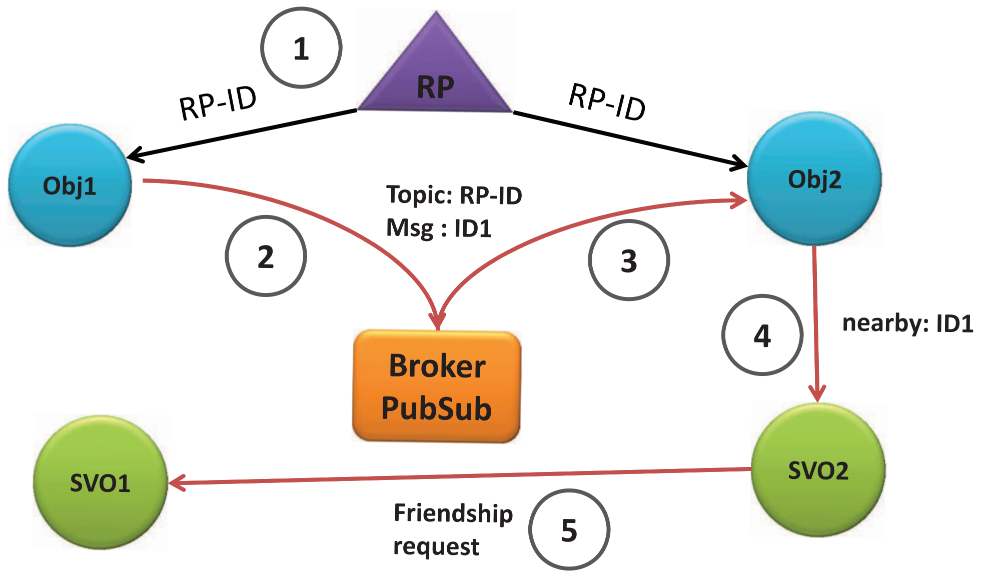


Figure 3.3: Local Neighbor Discovery: the encounter detection is implemented on the physical devices

already is. Obj1 performs a subscribe to topic “RP-ID” and performs a publish in the same topic stating its SVO’s ID (ID1) (2). As long as Obj1 is in the RP coverage area, it performs the same publishing operation each $T_s$ decided by its SVO (SVO1) and represents the minimum time between two samples so that these samples represent the same encounter and not two distinct encounters. Every time Obj1 publishes this message, Obj2 who has subscribed the same topic, receives a message with SVO1’s ID (3) and sends a message to its SVO (SVO2) stating the identity of the neighbor (ID1) (4). After $N$ meetings SVO2 proceeds to send a friendship request to SVO1 (5). The process is dual, and therefore SVO1 after $N$ meetings with SVO2 sends the friendship request to SVO2. Requests are then accepted and the two SVOs create a relationship.

This process requires a passive participation of the devices, which have only to perform the sensing. However, the chances to perform the discovery can be increased by enabling the beaconing on devices (if they are able to), which will become a reference for the neighboring devices. In this algorithm, the crucial point is the discriminating threshold of the signal strength and the inter-scan time of the interfaces. A high threshold will reduce the number of seen objects, while a low threshold may erroneously lead to the conclusion that two objects are close to each other while being at distance of up to a hundred meters, especially in networks with wide coverage.

Beacon duration and frequency are the key elements only in some situations. Indeed, if some reference points are present, then it is not necessary that the devices start to transmit beacons, given that there is already a radio reference point. The beaconing should be activated in the absence of other reference points. The alternating of beaconing/listening states may use algorithms already present in the literature to decide the duration and frequency of beaconing so as to ensure maximum probability of encounters.

### 3.2.2 Discovery by PubSub Alerting

In this scenario, I consider devices that cannot scan radio interfaces for hardware reasons (for example devices with GSM, UMTS, LTE interfaces) or for software permission reasons (for example devices with Apple's IOS operating systems). In this case, the approximate location via GPS or Wifi has to be known. The PubSub-Alerting Discovery (PSAD) algorithm uses an alert system that relies on the Publish/Subscribe protocols (e.g., MQTT) when registering in a place. For this reason, it is needed a common repository of places where SVOs can find the right

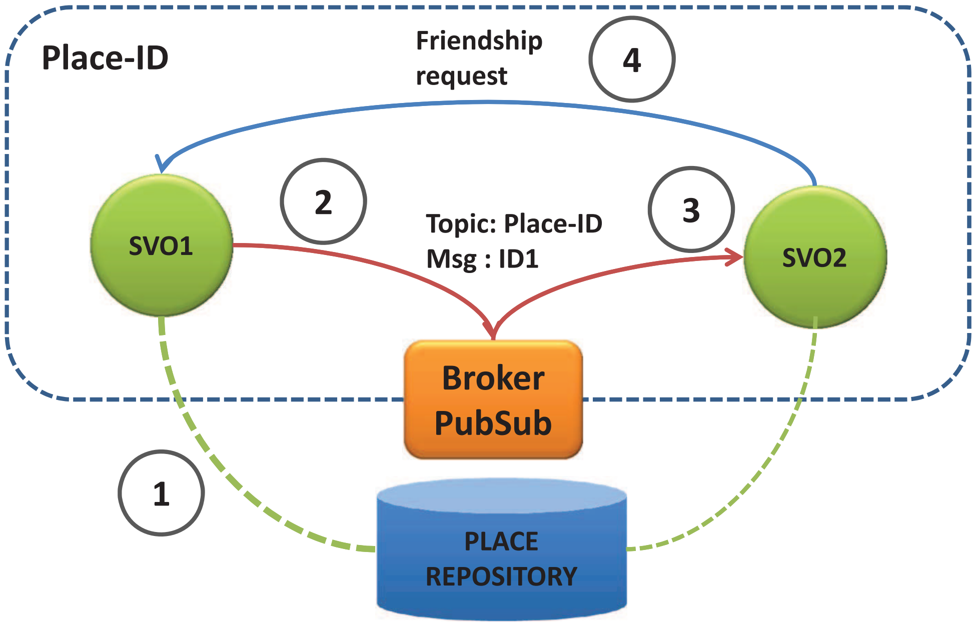


Figure 3.4: Discovery by PubSub Alerting

correspondence between GPS coordinates and the places they have to register in. Every time the physical device detects its position, it sends it to its virtual counterpart. The sampling frequency and the sending of the position are adaptive depending on the speed, in such a way that the positions are evaluated only when the object is stationary or almost-stationary. The discovery algorithm is performed entirely in the cloud virtual counterparts.

In Fig. 3.4, the SVO1 (virtualization of the object Obj1) receives the position from its physical device and retrieves the related Place-id from the place repository (1). Obj2 is already in the same place and the related SVO (SVO2) has already subscribed the topic that refers to this place. At this point, SVO1 performs a subscribe to the topic "Place-ID" and a publish in the same topic stating its own ID (ID1) (2). As long as Obj1 is in this place, SVO1 sends a publish each $T_s$ interval, whose publish messages are received by SVO2 (3). After $N$ received messages with ID1 as ID, SVO2 sends a friendship request to SVO1 (4). The process is dual, and therefore, after $N$ messages with ID2, SVO1 sends the friendship request to SVO2. Requests are accepted and SVO1 and SVO2 create a relationship.

### 3.2.3 Discovery by Social Alerting

Also in this scenario, the devices are not required to scan the radio channels but need to be able to determine their positions. Unlike the previous case that uses a central element (the PubSub broker), the Social-Alerting Discovery (SAD) algorithm exploits the social graph to spread the alerts about registrations in a place in a gossip-like way. By word of mouth, through friends and friends of friends, two nodes that do not know each others can communicate. In Fig. 3.5,

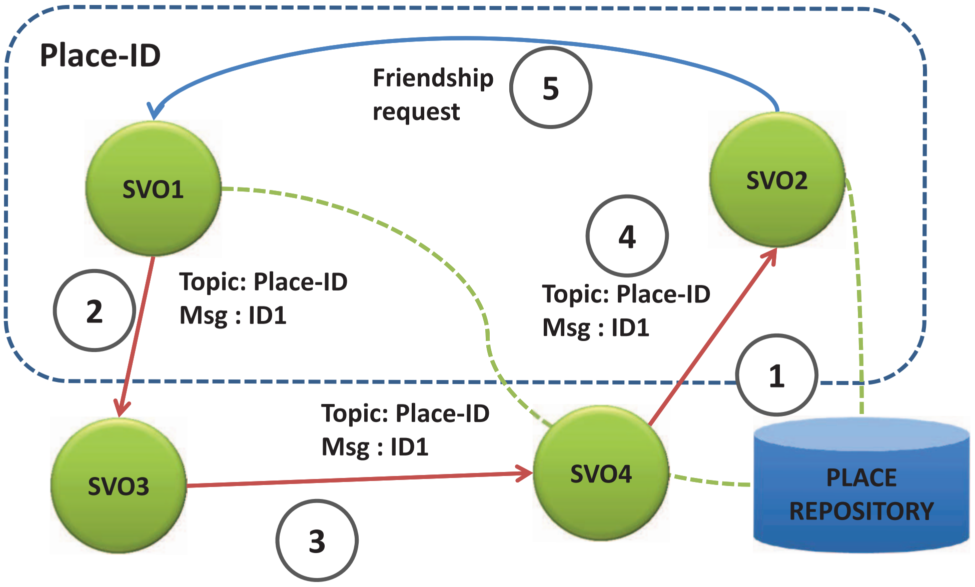


Figure 3.5: Discovery by Social Alerting

SVO1 and SVO2 are virtual objects of devices in the same place, which has ID retrievable from place repository (1). The place where the objects are in, is added in the profile of each SVO of both fixed or mobile devices. In the mobile device case, the descriptor is added or updated only if the object is not moving. SVO1 sends an alert message saying that it is in the place with identifier “Place-ID” to its friends (2) and forwarded by SVO3 (3) and SVO4 (4) to SVO2. In fact, SVO3 is friend with SVO1 and SVO4, whereas SVO2 is friend with SVO4. After $N$ alerts, SVO2 sends a friendship request to SVO1 (5). Also in this algorithm, when the requests are sent through both directions the friendship relationship is created.

## 3.3 Experimental Evaluation

### 3.3.1 Simulation Setup

To perform my experimental analysis, mobility traces of a large number of objects are needed. Since real data of this type are not available at the present, I resorted on a mobility model called Small World In Motion (SWIM) [78] to generate synthetic traces. The outputs of the SWIM model are traces of the position and encounter events of humans. In this case, instead, I am interested in the mobility of things and consequently, I assume that each user brings a device with WiFI interface for testing the first algorithm, or a GPS system for the other two algorithms. To evaluate the performance of the first algorithm, I considered an area of 17000 sq.m. with different numbers of reference points (WiFi access point in the simulation) from 10 to 30 randomly scattered. I assumed a coverage radius of 30 meters for each reference point. For the other two algorithms, I considered the same area without reference points.

### 3.3.2 Simulation Results

To evaluate the LND algorithm, I want to get the number of encounters that the algorithm has been able to detect compared to the total number of real encounters, whose ratio is named Hit Ratio (HR). In each simulation, I varied the scan duration $T_w$ , the total number of access points in the area and I set two different values for the inter-scan time $T_s$ of 60 and 300 seconds resulting into two groups of curves.

In Fig. 3.6 the results of the simulations are shown. As expected the HR grows with the scan duration $T_w$ and the total number of APs. Despite it is true that there are fewer detected meetings with a longer inter-scan time, in this case, devices have to be closer for longer unveiling a higher probability of common interests. Since the APs' density cannot be set arbitrarily in the real world, the only variable parameter which affects the number of discovered neighbors is the inter-scan time and, with less influence, the scan duration.

The other two algorithms are evaluated by using SWIM traces and by simulating GPS coordinates acquisition. From devices' coordinates I inferred the places which the devices visited. In the simulation I considered the same 17000 sq.m area of the LND algorithm positioned in the campus of the Faculty of Engineering in Cagliari, I used Google places service as place

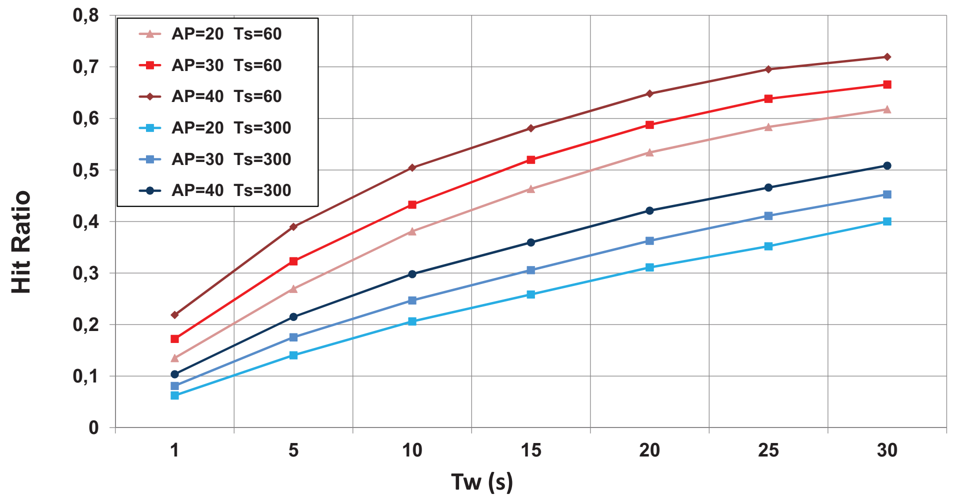


Figure 3.6: Hit Ratio when using LND algorithm

repository, and accordingly I got six places in the considered area.

In the simulation I varied the waiting time $T_{wait}$ needed by the SVO to understand if the related device is moving or steady and for the SAD I considered the cases of 2 and 3 hops as time-to-live of the alert messages (involving friends of friends in the 2 hop case or more in the 3 hop one) over a social graph of 100 nodes and average node distance equal to 4. In Fig. 3.7 we can see that at higher $T_{wait}$ corresponds poorer performance in detecting neighbor devices. The two algorithms perform quite the same so as we can infer that the best choice in this scenario is given by SAD with 2 hops since there isn't a central node like the PubSub broker and the messages' overhead is lower. As general observation, the SAD algorithm is not to be thought

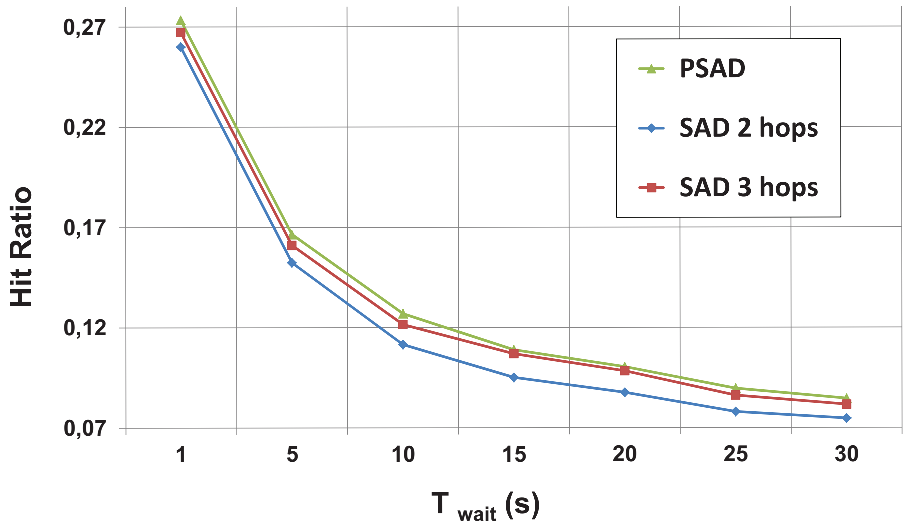


Figure 3.7: Hit Ratio exploiting the devices' positions

as alternative for the LND algorithm because they refer to different scenarios. On the contrary, they should be considered as complementary approaches in order to detect as many devices as possible.

# Chapter 4

# Exploiting the aggregation level in the Lysis architecture: A SIoT-Aware Approach to the Resource Management Issue in Mobile CrowdSensing

Mobile CrowdSensing (MCS) is defined as a pervasive sensing paradigm where mobile devices gather data with the aim of performing a specific application. The major issues in MCS are the following: mobile devices are characterized by limited resources; scalability issues appear when the number of objects that could be potentially involved in the sensing increases together with the application requests; the MCS sensing tasks should be easily integrated in the variety of IoT applications that in a dynamic way requires the crowd wisdom through MCS tasks.

This chapter handles the analyzed issues by providing the following contributions. The Social IoT (SIoT) paradigm is adopted to address the scalability issues when searching for objects that can potentially take part to the MCS scenarios. Indeed in SIoT a social network is created among objects, which exhibits the typical scalability advantages of social networks when looking for peers in big communities. I integrate the MCS logic into the Lysis platform that implements the SIoT paradigm, making easy for applications developers to activate MCS tasks. I propose a new algorithm to address the resource management issue so that MCS tasks are fairly assigned to the objects, with the objectives of maximizing the lifetime of the task groups. Preliminary experimental results prove that the devices' lifetime values tend to a single value and multiple applications can use the same outcome with improvement in terms of latency and computing resources.

The remainder of the chapter is structured as follows. Section 4.1 provides the key elements related to MCS and the motivation the has guided my proposal. Section 4.2 tackles the description of the proposed algorithm, which has been implemented in the Lysis platform. In Section 4.3 the resource allocation model is described. Tests results and performance evaluation are discussed in Section 4.4.

## 4.1 Key elements in MCS and motivation of the proposed solution

In the MCS paradigm, the objective is that of performing an application gathering data from a number of devices that are capable of fulfilling the application requirements. These devices can either contribute to the application execution in a participatory way or the system can collect data from them in an opportunistic way.

Compared to traditional sensor networks, MCS entails significant advantages, mainly related to deployment costs and coverage extension. Indeed, in MCS data can be collected by any sensor that meets the application requirements, i.e. that is in the right place at the right time and which information quality requirements are satisfactory. Therefore, spending money and time in deploying new infrastructure to cover new areas is not needed, because the existing one (e.g. WiFi, 3G, Bluetooth) can be used.

Nevertheless, MCS inherited one of the major issues of traditional sensor networks: the limited amount of resources available on devices (e.g. available energy, storage capacity, computing speed). Fair resource allocation mechanisms can be applied to overcome this problem.

## 4.2 MCS in the Social IoT

This section presents the solution I propose to exploit the SIoT paradigm to implement the MCS.

### 4.2.1 Lysis key elements for the MCS

A common solution to most of the issues raised in the previous Section, recently adopted by the major IoT platforms, is the introduction of a virtualization layer [14], as shown in chapters 1 and 2. Main goal at this layer is to digitally describe, through the creation of Virtual Objects (VOs), the capabilities and features of the corresponding physical objects to augment their potentialities in a way transparent to users.

The proposed model relies on the Lysis architecture, already depicted in chapter 2, that foresees a four level structure as shown in Fig. 2.1. Its lowest layer is populated by the Real World Objects (RWO). At this layer, the Physical Devices (PD), directly access the platform via direct links to the Internet, while other objects (more resource-constrained) must rely on special Gateways (GWs). PDs and GWs are able to perform basic tasks, such as secure communication with the respective virtual counterparts as well as management and presentation of the data coming from sensors. On top of this, the Virtualization layer, directly interfaces with the real world and is populated by the Social Virtual Objects (SVO). The Aggregation layer is responsible for composing several SVOs into entities with extended capabilities, called Micro Engines (MEs); the ME is the entity that implements part of the application logic performed at the upper layer. In each ME, the output for a request coming from an application can be reused to serve requests from different applications that require the same information or service to save bandwidth and CPU. Finally, at the Application layer, user-oriented macro services are provided (APP).

Socialization algorithms implemented in the first two levels allow for the creation of social relations as described in Section 1.1. The resulting social graph is exploited to find the required resources and a special SVO called SVO-Root (SVO-R) is in charge of forwarding the query to its friends and friends of friends as shown in Fig. 2.5. In the context of the CrowdSensing, the MEs have an important role when it is necessary to orchestrate the execution of a set of tasks in a group of SVOs. To find the required resources each application sends a query with the semantic description of the resources needed to the ME Controller (MEC) as shown in Fig. 4.1. The MEC is in charge of forwarding the query to the social graph of SVOs to get a reply with a list of resources and their access keys.

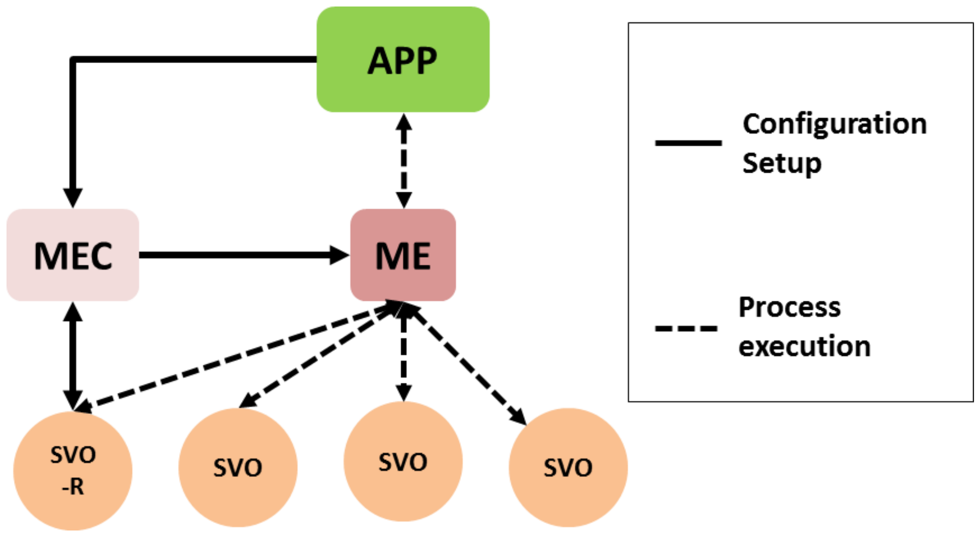


Figure 4.1: Configuration chain

The MEC, after having achieved the required resources, allocates them in the ME, which handles the SVOs for the processing needed by the requiring application. The ME is able to configure each SVO in order to perform some operations in background or to trigger their resources so as to provide an active participation in the required process.

### 4.2.2 The MCS Micro Engine

In this Section the whole procedure to assign the application tasks to the available nodes and appropriately allocate task frequencies in order to optimize energy usage is presented.

Every process starts when an application sends a query with the description of the required resources and the ID of the ME that is able to execute the specific processing of the data. In this section, I show the ME in charge of exploit the crowdsensing algorithm and hereafter, I refer to it as CrowdSensing Micro Engine (CS-ME). Since it is required by Lysis platform, the query is passed to the MEC in JSON format and specifies how the ME handles the input resources and includes the key to verify the access permissions (if different from the public), the maximum number of requested resources (*limit*), the depth of search in the social graph (*hop*), the coordinates of the geofence and a description of the resources in text format in the *description* parameter, and the type of relationship which can be exploited by the SVO search process is specified in the *relationship* field as shown in Fig.4.2.

```
"owner-key":"gsl1hcoldvnk9bar63mif13vhp",
"owner-id":"user1@gmail.com",
"limit":"20",
"hops":"2",
"longitude":"9.109900",
"latitude":"39.229055",
"range":"30",
"description":"sensor brightness",
"relationship":[{"type":"SOR"}]
```

Figure 4.2: Textual description of the resources requested by an application

The MEC checks if it has already received other requests that have the same description of input resources. If so, it says CS-ME to add the new query ID to the previous ones in order to respond to the new request with the data produced for the previous requests. Otherwise, it executes the SVO search process through the SVO-R of the user who started the application. The achieved resources are sent to the CS-ME. Fig.4.3 shows an excerpt of the JSON configuration

```
"input-id":"1",
"resources":[
 {
  "Permission":"public",
  "Key":"",
  "ResourceURL":"svotest-100.appspot.com",
  "FeatureName":"BRIGHTNESS"
 },
 {
  "Permission":"friend",
  "Key":"gsl1hcoldvnk9bar63mif13vhp",
  "ResourceURL":"svotest-101.appspot.com",
  "FeatureName":"BRIGHTNESS"
 }
```

Figure 4.3: The list of SVOs which can participate in the CrowdSensing process is sent by the MEC to the CS-ME

sent to the CS-ME. For each resource, access permissions (*Permission*), the API key (*Key*), the URL of the SVO (*ResourceURL*) and the SVO's sensor name (*FeatureName*) are indicated. In the excerpt in Fig.4.3, the first resource is public and does not require any access key, whereas the second SVO can share its resource only with friends and therefore, it demands to specify the API key in the JSON configuration file.

```
 "input-id":"0",
   "resources":[
            {"MSF":"0.0033"},
             {"lat":"39.229222"},
             {"lon":"9.109670"},
             {"rad":"30"}
```

Figure 4.4: Algorithm parameters: geofence and Minimum Sensing Frequency

Fig.4.4 shows the parameters about the geofence (*lat*, *lon*, *rad*) and the *Minimum Sensing Frequency* (MSF) parameter which is set by the requesting application. The CS-ME checks if the input resources and the geofence are the same of the previous requests. If so, the MSF is set to the highest requested value, while the output of the crowdsensing process is returned also to the applications with less stringent requirements. Furthermore, the CS-ME has the ability of self-healing, so that if a malfunction occurs in one or more of the resources, it can ask to the MEC for a reallocation of resources.

Finally, the CS-ME requests all the power-profiles to all the participating SVOs and configures them in order to receive alerts about the presence within the geofence. Since it is an event-driven process, the CS-ME waits for any alerts from SVOs to assign frequencies to the available SVOs according to the algorithm described in Section 4.3 and shown in Algorithm 1.

**Algorithm 1** MCS task allocation algorithm

```
Require: $\mathcal{X}_k > 0$
 1: while alert do
 2:     if $|\mathcal{X}_k| = 1$ then $f_{1,k} = F_k^{MSF}$
 3:     else
 4:         for all $i \in \mathcal{X}_k$ do
 5:             Compute $f_{i,k}$ according to Eq. (4.6)
 6:             if $f_{i,k} < 0$ then
 7:                 Set $f_{i,k} = 0$ and remove $i$ from set $\mathcal{X}_k$
 8:                 Repeat from step 2
 9:             end if
10:         end for
11:     end if
12:     alert = false
13: end while
```

## 4.3 The Resource Allocation Algorithm

The adopted resource allocation strategy relies on an optimization algorithm where the ME decides the amount of resources to allocate to a task, according to the constraint requests coming from the higher layers, and to the available resources coming from SVOs. The optimization algorithm proposed in the following is based on the one proposed in [79] and focuses on the maximization of the system lifetime, but it can be easily extended to other resources, such as processing speed or storage capacity.

According to [80], a system's lifetime $\tau$ can be defined as the time until the first node belonging to that system depletes its residual energy

$$\tau = \min_i \tau_i \tag{4.1}$$

where $\tau_i$ is node $i$'s lifetime, defined as

$$\tau_i = \frac{E_i^{res}}{P_i^{drain} + \sum_k E_{i,k} f_{i,k}} \tag{4.2}$$

with: $E_i^{res}$ residual energy for node $i$; $P_i^{drain}$ power consumption due to other activities of the nodes that are not assigned by the ME (e.g. other tasks that are started directly by the user); $E_{i,k}$ is the energy consumption needed by node $i$ to perform task $k$; $f_{i,k}$ is the frequency at which node $i$ performs task $k$. Equation (4.2) entails that the ME is able to affect the lifetime of the system made of the nodes of a geofence $\mathcal{G}$, by appropriately adjusting the frequency at which the involved nodes perform the required tasks.

The objective of the proposed framework is that of assigning tasks so that there is not any node that depletes its resources earlier than the others. This implies that, when addressing lifetime, this condition is satisfied when the workload is fairly distributed among all the nodes, i.e. when all the nodes tend to the same lifetime. In other words, taken any two nodes $\{i, j\} \in \mathcal{G}$, their workload is fairly distributed if they have the same lifetime $\tau_i = \tau_j = \tau, \forall \{i, j\} \in \mathcal{G}$. Therefore

$$\sum_k \alpha_{i,k} f_{i,k} + \frac{P_i^{drain}}{E_i^{res}} = \sum_k \alpha_{j,k} f_{j,k} + \frac{P_j^{drain}}{E_j^{res}} \tag{4.3}$$

where $\alpha_{j,k} = E_{i,k}/E_i^{res}$. I define the total amount of power consumption contributions of node $i$ with the exception of task $k$ as $\delta_{i,k} = \sum_{l \neq k} \alpha_{i,l} f_{i,l} + P_i^{drain}/E_i^{res}$. Substituting it in Equation (4.3)

$$f_{j,k} = \frac{\alpha_{i,k}}{\alpha_{j,k}} \cdot f_{i,k} + \frac{\delta_{i,k} - \delta_{j,k}}{\alpha_{j,k}} \tag{4.4}$$

In order to satisfy accuracy requirements coming from the application layer, the following relation about the $F_k^{MSF}$, i.e. the MSF frequency required by the MCS task has to be fulfilled

$$F_k^{MSF} = \sum_j f_{j,k} = \sum_j \frac{\alpha_{i,k}}{\alpha_{j,k}} \cdot f_{i,k} + \sum_j \frac{\delta_{i,k} - \delta_{j,k}}{\alpha_{j,k}} \tag{4.5}$$

Therefore, the ME can assign a frequency to each SVO $i$ for each task $k$, according to the following Equation

$$f_{i,k} = \frac{1}{\alpha_{i,k}} \cdot \left( \sum_j \frac{1}{\alpha_{j,k}} \right)^{-1} \cdot \left( F_k^{MSF} + \sum_j \frac{\delta_{j,k}}{\alpha_{j,k}} \right) - \frac{\delta_{i,k}}{\alpha_{i,k}} \tag{4.6}$$

It may happen that the result of Equation (4.6) is negative: it is the case of a node which lifetime is so low that it cannot take any other load, because its lifetime is still lower even assigning the whole task $k$ to the other nodes. Since a negative frequency does not have physical meaning, in this case frequency $f_{i,k}$ is set to $0$, and Equation (4.6) is evaluated again for every node, excluding node $i$ from the computation.

## 4.4 Performance Analysis

To perform our experimental analysis, I recorded data from 7 Android smartphones of several models. The SVO template and the smartphone driver for this test are available on Lysis platform[1] and require Android 5.0 at least. The geofence was a zone of 30 meter radius in the central park of the Engineering Faculty in Cagliari. I monitored four sensor types: brightness, radio status, accelerometer and gyroscope sensors. This choice was imposed as other types of sensors were not available on all devices. Each test has last to the complete shut down of all devices for battery discharge. I launched an application with MSF= $3.5$ minutes$^{-1}$. Firstly I ran it without CrowdSensing, with Equally set Sampling Frequency (ESF) of $0.5$ minutes$^{-1}$ for each node. Then I compared the outcome with the data recorded from an application using the CS-ME. As expected, Fig. 4.5 shows that the use of the CS-ME allows for longer lifetime for all devices, whereas the ESF causes device shutdowns in just 4 hours. In the CS-ME case, the

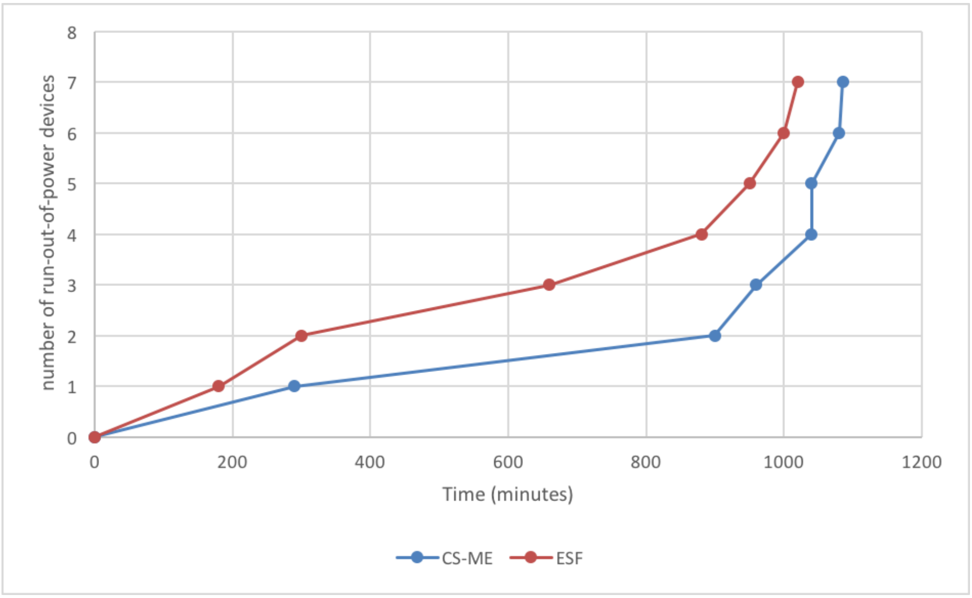


Figure 4.5: Performance analysis: the CS-ME allows for longer lifetime for the involved devices than the ESF case

devices would have shut down almost together but I think that the discordance with the recorded data is due to the fact that some devices were at the limit of the geofence and the GPS system gave wrong position causing an exit message from those actors.

In Fig. 4.6, the different frequency values on each SVOs using the CS-ME are shown. We can see how the sampling of the brightness sensors are distributed with different frequencies from the highest one (Sensor 4) to the lowest ones (Sensors 3 and 7). Also, the frequency values

[1]www.lysis-iot.com

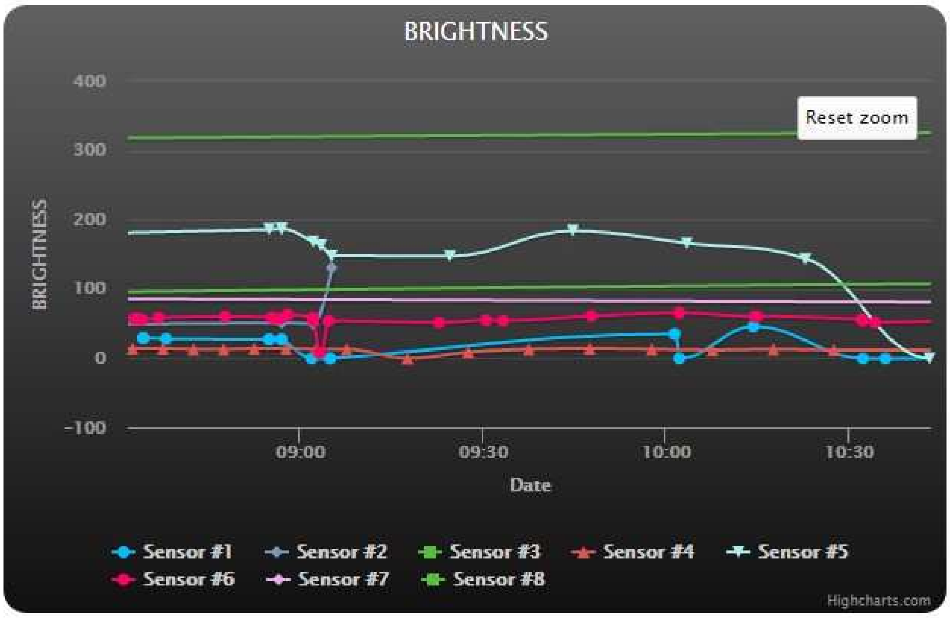


Figure 4.6: The different frequency values set on the devices in the geofence

varies during the process. For instance, we can see how the frequency of sensor 4 is reduced by a half after 9.00.

Finally, I experimented configuration setup latency varying from 10 to 30 seconds since it depends on changing factors like number of devices in the geofence zone, search depth in the social graph and quality of connection of the radio media. Nevertheless, further application queries had lower latency since the geofence was the same but with higher MSF ( about 5 seconds of latency) or with lower MSF (about 2 seconds of latency). Indeed, the MEC executes one SVO search process for multiple applications which ask for the same resources.

# Chapter 5

# On adding the social dimension to the Internet of Vehicles: friendship and integration to Lysis

In this chapter, I firstly analyze the combination of Vehicular Ad-hoc NETworks (VANETs) with the Social Internet of Things (SIoT), i.e., the Social Internet of Vehicles (SIoV) and secondly, I propose an implementation of the SIoV through the integration into Lysis architecture. In the SIoV every vehicle is capable of establishing social relationships with other vehicles in an autonomous way with the intent of creating an overlay social network that can be exploited for information search and dissemination in VANET applications.

The contribution of my research activity is three-fold: firstly, I define some relationships which can be established between the vehicles and between the vehicles and the road side units (RSUs); secondly, I propose a SIoV middleware which extends the functionalities of the Intelligent Transportation Systems Station Architecture (ITS SA), defined by ISO and ETSI standards, to take into account the elements needed to integrate VANETs in the SIoT; finally, present a novel implementation for the Internet of Vehicles, where the vehicles have been virtualized and enabled to create friendships with other vehicles based on their movements and activities.

## 5.1 The Social Internet of Vehicles

The IoV paradigm brings to many unprecedented challenges, such as how to find the right vehicle that can provide the right service and to which extent trust the information provided by other vehicles/RSUs; the proposed SIoV paradigm tries to overcome these problems. For this reason, in this preliminary analysis, I assume that each vehicle is connected to the Internet and then it is equipped with an OBU with both 3G radio for Internet connection and short-range radio for V2V and V2I communications.

### 5.1.1 SIoV relationships

The IoV represents a homogeneous set of objects with high mobility where each vehicle is a source of key information for different types of applications, such as active road safety or traffic management efficiency. In this scenario, a very large amount of data is streamed from vehicles and RSUs so that the discovery of the object that can provide the desired service can result quite difficult.

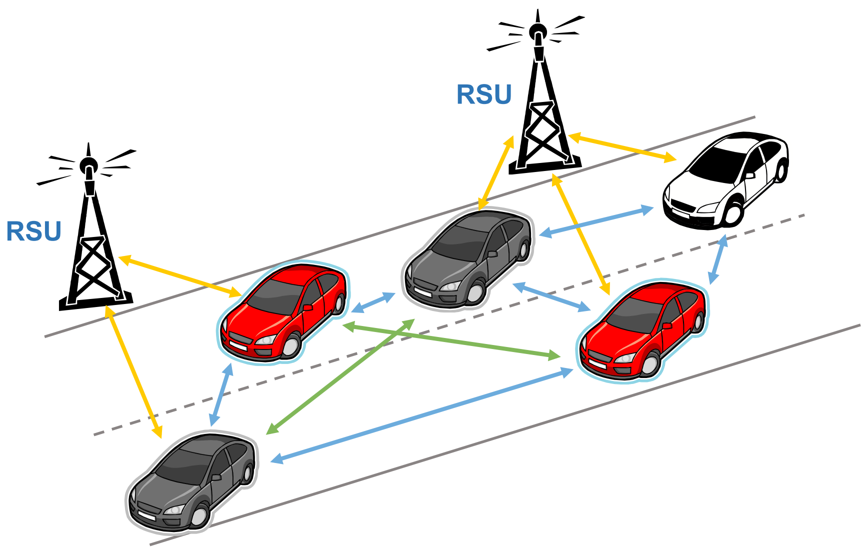


Figure 5.1: Relationships created in the Social Internet of Vehicles scenario. Green lines represent Parental Object Relationships, blue line represent Social Object Relationships and yellow lines represent Co-Work Object Relationships

Starting from the Social Internet of Things vision, I claim that vehicles and RSUs can create their own relationships and use these relationships to efficiently look for services and exchange information in an autonomous way for the benefit of the human user.

As shown in Figure 5.1, I identify different social interactions in the Social Internet of Vehicles scenario:

- vehicles belonging to the same automaker and originated in the same period establish a *Parental Object Relationship* (POR): these relationships can give useful information about the status of a vehicle and can be exploited when providing diagnostic services and remote maintenance to the users;

- vehicles that come into contact through V2V communications create a *Social Object Relationship* (SOR). Vehicles meet randomly, e.g., when their owners go to work and then share the same way: more often two vehicles meet, the stronger the relationship that links each other and the higher the value it provides in service discovery and trustworthiness evaluation [81], [82]. The benefits given from the establishment of SORs are very important in SIoV scenario. In fact, most of the contents provided by the vehicles are related to certain areas and to limited times, as for example the communication of road incidents to vehicles proceeding toward the crashed areas or the sharing of useful information about traffic conditions or petrol stations; SORs take into account common vehicles paths and locations which permit to create social networks among vehicles strictly related to determined areas;
- in a similar way, when vehicles meet continuously with RSUs through V2I communications, they create a *Co-Work Object Relationship* (CWOR) with them. These relationships can be useful to provide traffic information or to guide the drivers in less congestionated routes.

The combination of these relationships brings to the creation of the social network of vehicles where information can be gathered in a distributed manner by crawling the network from friend to friend in the same way humans do in their social networks. Moreover, the SIoV evolves according to the activities of vehicles and RSUs so that it can quickly adapt to the mobility scenario, by creating new links or adapting the strength of the existing ones.

Table 5.1 shows some typical applications in the SIoV scenario that can benefit from the availability of social relationships between vehicles.

Table 5.1: SIoV sample applications

| **Applications** | **Relationships** | **Description** |
|---|---|---|
| Diagnostic service | POR | Vehicles contact friends in order to know if they have had/solved a similar issue |
| Traffic information | SOR | Vehicles obtain from friends, that usually travel in the same routes, updated information about traffic condition |
| Community services | CWOR | RSUs communicate with vehicles to provide information about road conditions or maintenance |

In Section **??**, I analyze the network structure arising from the above types of social relationships and show how the resulting network is navigable.

### 5.1.2 SIoV middleware

VANETs can be viewed as a component of the Intelligent Transportation Systems (ITS), i.e., vehicular network applications that aim to provide innovative services related to different modes of transport and traffic management.

ITS has been the core of several ongoing and terminated European project, such as T-TRANS [83], ICSI [84] and U-CONNECT [85] and one of the most important part of these systems, the ITS Station Architecture (SA), has been defined by ISO [86] and ETSI [87] standards in the possible use-cases. Figure 5.2 illustrates an example of possible elements in the ITS SA: this architecture extends the OSI network layers considering the management and the security entities, which are in charge of managing communications in the ITS station and provide security to the OSI communication protocol stack respectively. Each sub-layer is interconnected with the others via Service Access Points (SAPs) or via APIs.

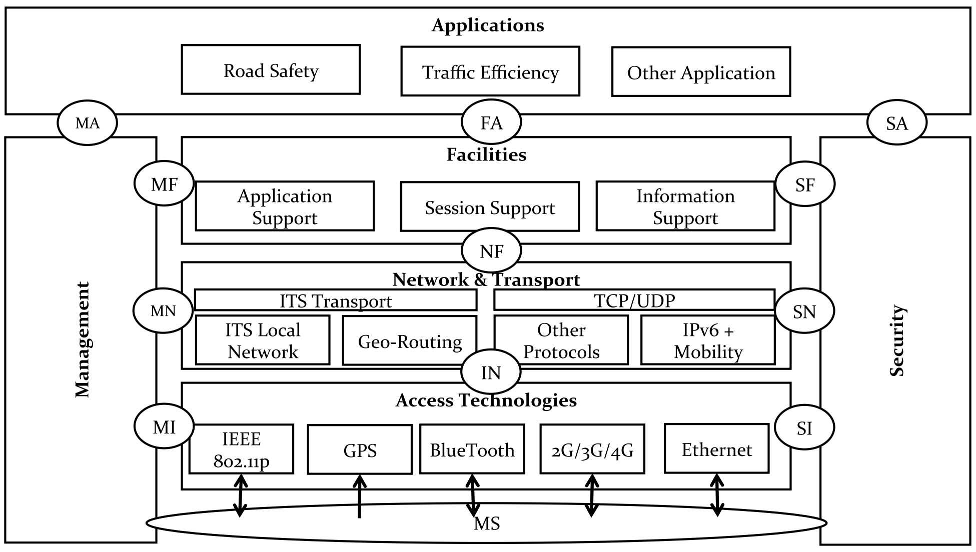


Figure 5.2: ITS Station Architecture defined in ISO 21217 and ETSI EN 302 665

However, a technical discussion of the SA is not the focus of my research activity. Accordingly, in the following, I analyze the modifications of the ITS SA, as depicted in Figure 5.3, to include the middleware functionalities, thus to integrate vehicles in the SIoV. All the new functionalities can communicate with the SIoV server thanks to the SIoV APIs. In particular, this interface formats the data received from the different applications so that they can be sent correctly to the server; moreover, it retrieves the needed data and delivers them to the applications that requested them.

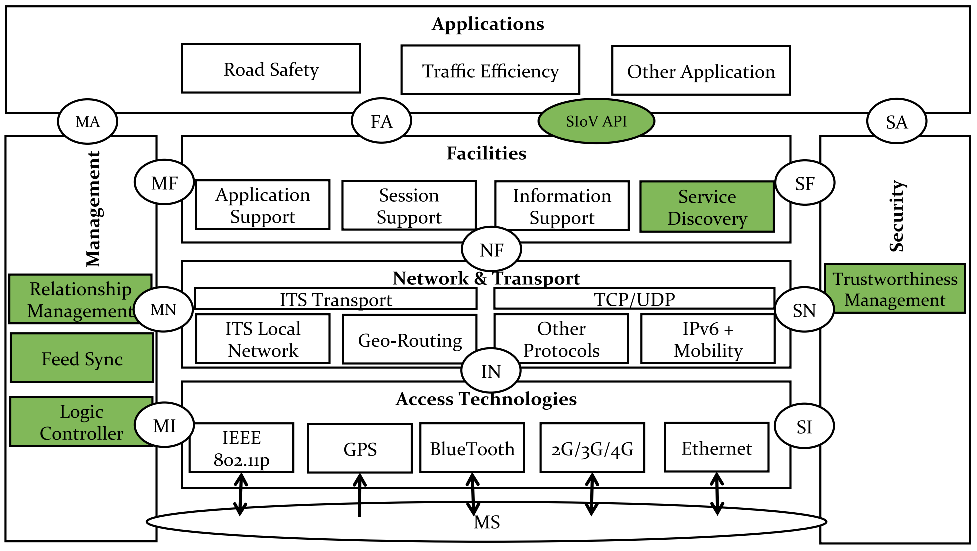


Figure 5.3: Integration of the SIoT middleware in the ITS Station Architecture

## Management

The Management pane contains three major features. The relationship management is a key module in the middleware; it has the task to retrieve from the server information about the relationships created on the basis of the owners control settings, since the vehicles have not the intelligence to select the friendship. The feed sync module handles the transmission and reception of the feed. During transmission, it takes all the data from the different applications, aggregates them whenever possible, and send to the server through the SIoV APIs; during reception, it performs a parsing on the received data and routes them to the module or applications for which they are intended. The logic controller module is the core of the middleware, since it handles all the different events, such as the timing to send the data or the reading by the sensors: these events can be driven by the applications or by the middleware itself.

## Security

In the Security pane the trustworthiness management module is added; this module aims at understanding how the information provided by the other vehicles or RSUs shall be processed. Reliability is built on the basis of the behavior of the object and is strictly related to the relationship management module. Moreover, it provides feedback based on the quality of the service received.

## Facilities

The Facilities layer is extended by the service discovery module that has the role to enable interaction between objects that are friends. It can retrieve an information about the real world posted in the server by a friend or can find a specific service provided by another vehicle/RSU.

## 5.2 Toward the cloud SIoV

As explained in Section 1.5, the IoV scenario is a distributed system characterized by nodes with high mobility, where the vehicles can become unavailable due to the lack of connectivity as a result of the hostile propagation environment. Moreover, to support the deployment of major applications such as transportation safety and traffic management, and to satisfy their quality requirements, the IoV has to collect and analyze large amounts of information about the vehicles' surroundings.

In such a scenario, I envision that, through the use of virtualization technologies, each vehicle/RSU in the real world is associated to its virtual counterpart in the cloud. Associating a digital counterpart to physical objects is a common practice in the latest IoT research activities [14], since through virtualization the physical devices are augmented in their capabilities and are able to: i) fully describe their characteristics with semantic technologies and then be able to interact with other virtual objects; ii) identify, analyze and manage the context related to an object surroundings and take decision accordingly; iii) make easy the search and discovery of services and devices, which continuously join, move and leave the network.

To enable the creation of a Social IoV, I leverage on Lysis platform, which enables to manage the social activity of vehicles as defined by the SIoV paradigm. Moreover, I present a client middleware to enable the vehicles to communicate with their virtual counterpart, and show how it could be integrated with the Intelligent Transportation System station architecture.

### 5.2.1 SIoV cloud-based architecture

The proposed platform relies on the Lysis architecture, already depicted in chapter 2, and the SIoV-enabled four level architecture (see Fig. 5.4). In the SIoV context, the RWO level, which is implemented out of the cloud, includes vehicles and RSUs that are capable of accessing the Internet. A detailed description of the software modules implemented in the vehicles is given in Section 5.2.2. In the SIoV, the deployment of applications involve vehicles moving in large and complex environments, which have to guarantee quality requirements and at the same time not to overload the network resources. To solve these issues, each object in the Real World Level is represented by a virtualization at this level. The Social Virtual Object (SVO) is a key part of the overall solution and depicts the RWO in terms of semantic description and functionalities extended with social capabilities. The aggragation level composes data from multiple SVOs on the basis of patterns to ensure a high re-usability level. At application level, user applications

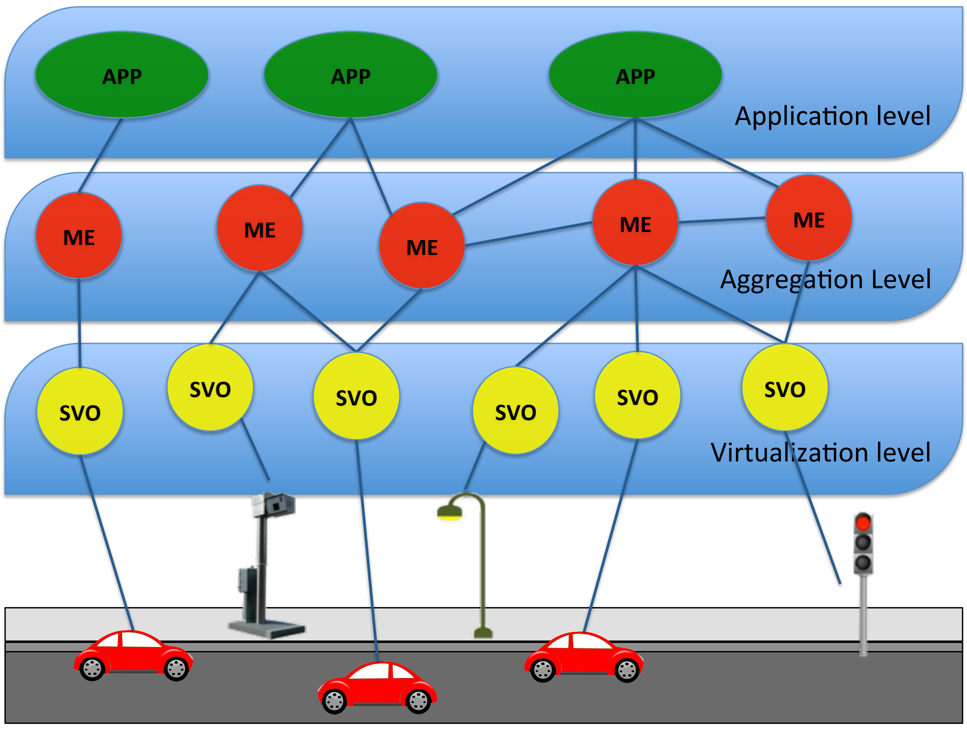


Figure 5.4: Cloud-based SIoT architectural solution.

are responsible for final processing and presentation.

To enable the development of the SIoV I created a new Template to describe the vehicles, which is used every time a user wants to register his/her car to the platform. Moreover, I implemented the logic behind the creation of SOR relationships among vehicles and developed an application for the remote monitoring of the vehicles.

## 5.2.2 Board Software

In order to communicate with the platform through the SVO-HAL, the On-Board Unit needs a software to exploits all the required functionalities: to establish a secure communication channel, to translate protocol related data values in usable value format, to implement real-time logic and to manage the communications with several different media inside the vehicle. For this reason, the vehicles implement the following modules to be part of the platform as depicted in Fig. 5.5:

- *Physical Device-Hardware Abstraction Layer* (PD-HAL): it communicates with the corresponding module in the virtualization level. Its major role is to introduce a standardized communication procedure between the platform. It is also in charge of creating a secure point-to-point communication (encrypted) with the SVOs. The pairing is done at setup phase, when PD-HAL module exchanges IDs, URL and encryption keys with the related module in the SVO.

- *Data Handler*: it intervenes whenever there is the need to process data from sensors before being sent by the PD-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.
- *Device Management*: it implements the actual device logic with reference to the participation of the PD to the Lysis platform, e.g., controlling the sensing frequency, managing local triggers, overseeing the energy consumption. It also runs the code that can be updated in run-time locally in the PD.
- *Protocol Adapter*: it consists in the hardware drivers to handle all local sensors and actuators.

In particular, the protocol adapter has to accomplish many of the hardware related functionalities: sensors/actuators vehicle management and device-to-device communications. The Vehicle Control (VC) module has to handle the access to available sensors and actuators on the vehicle, which in many cases, involves a cable or Bluetooth connection via OBD-II interface. The VC is also responsible for managing the connection with additional sensors on the vehicle, connected via Bluetooth that are not managed by the ECU. The *Device-to-device* (D2D) module handles V2V and V2I communications through several media such as ZigBee, Bluetooth, 802.11p and 802.11a/b/g/n. Although the vehicle SVOs are able to communicate between themselves and the RSU SVOs, some applications need a continuous exchange of information between the devices with such a frequency and latency as to result in a uselessness of the processing on the cloud due to the high data traffic and low performance in terms of latency. The core of the processing on the vehicle is constituted by the Device Management module. Compared to other Lysis-enabled devices (not vehicular), it is responsible for perform-

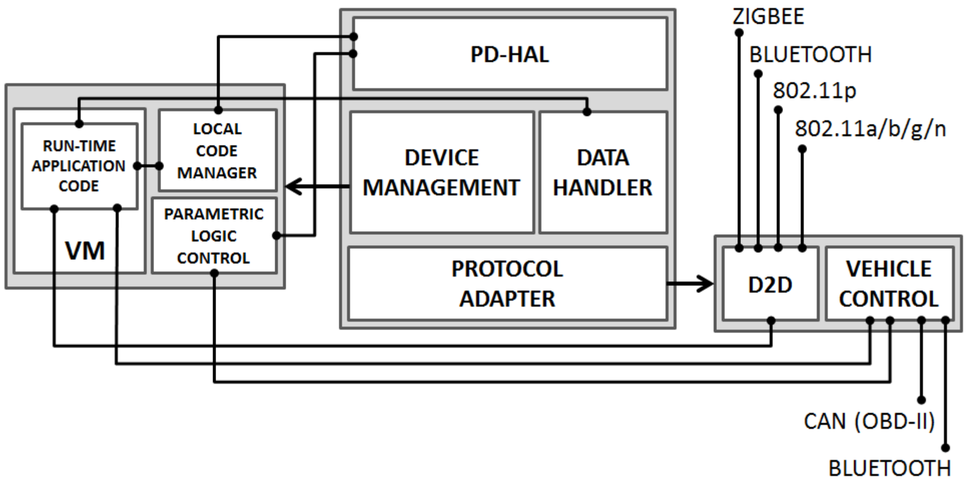


Figure 5.5: Software modules of the On-Board Unit.

ing actions that require low latency or high sampling frequency from sensors or applications involving V2V and V2I communication (via the D2D module in the Protocol Adapter). The development of a huge volume of data can still be supervised by the SVO, by processing statistical data, which, extracted from the application, is translated into usable format by the Data Handler module. There are several works that show how we can update code in run-time by means of Virtual Machines, script interpreters (such as Node.js or Python), Image-based approaches, or via Dynamic loadable native code components [88]. Since I do not aim to manage runtime driver updates, I think that script interpreters have the right mix between flexibility, abstraction and re-usability required by my idea of a SIoV architecture. This allows me to keep the execution of part of the application code of relevance to the device in the device itself when needed (e.g., when the application QoS requirements do not allow for latency level typical of the Cloud Computing paradigm). The source code is deployed and updated via the *Local Code Manager* (LCM) module, which receives the source code from the PD-HAL and gives it to the interpreter. The *Parametric Logic Control* (PLC) module represents the device logic which can controlled through parameters by the SVO: sampling rate of the sensors, the neighbor discovery algorithms, which is used as part of the process for the creation of social relations between the SVO, and any IFTTT (If That Then Else)-based trigger. Both the local application and the logic control have access to the HW through the VC module in the Protocol Adapter.

### 5.2.3 Integration with the ITS station architecture

Similarly to what already shown in Section 5.1.2, in Fig. 5.6 I propose to integrate the ITS SA to include the middleware functionalities for allowing the vehicles to be integrated in the Lysis-based SIoV. Specifically, the *Communication Encryption* module in the Security entity has the role to encrypt the communication between the OBU and the SVO. The *Device Management* and *Vehicle Control* modules in the Management entity have the role to manage the communication between the OBU and the SVO and to handle the access to available sensors and actuators on the vehicle, respectively. The *IEEE 802.15.4* and *IEEE 802.11 a/b/g/n* modules in the Access Technologies layer add respectively the ZigBee and Wi-Fi communication technologies on the OBU. The *Data Handler* module in the Facilities layer processes and converts (if needed) the data acquired by the vehicle before being sent to the SVO. Finally, in the Application layer the *SVO Communication* application handles the communication with the SVO regarding the SIoV applications running in the cloud.

Furthermore, in the ETSI standard [89] is also provided the definition of the typical vehicle ITS station, which contains a vehicle ITS-S gateway, an ITS-S host and an ITS-S router. The ITS-S host contains as a minimum the ITS-S applications and the functionality of the ITS SA

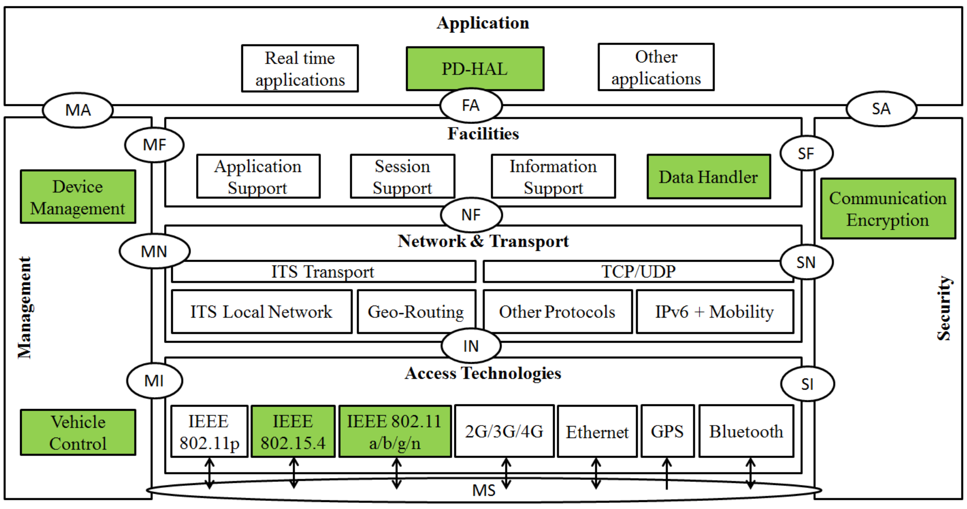


Figure 5.6: ITS Station Architecture integrated with the middleware functionalities for allowing the vehicles to be integrated in the SIoV.

needed for the ITS-S applications. In this case, I refer to the proposed ITS SA shown in Fig. 5.6. The ITS-S gateway interconnects an ITS station-internal network with the proprietary in-vehicle network (e.g. the Controller Area Network of the vehicle). Finally, the ITS-S router interconnects the ITS station-internal network with an ITS ad hoc network or a public network [90]. The former enables ad hoc communication among vehicle, roadside and personal ITS stations (i.e. V2V communication), whereas the latter provides mobile Internet access. In my proposal, the OBU (on which runs the software board) is the hardware device responsible of implementing the vehicle ITS station. It performs the roles of vehicle ITS-S gateway, ITS-S host and ITS-S router simultaneously. In fact, it is able to connect to the CAN through the OBD-II interface (ITS-S gateway role), it implements the functionality of the ITS SA needed for the ITS-S applications (ITS-S host role), and finally it enables V2V, V2I and Internet communication (ITS-S router role). The details about the hardware modules used for the OBU are provided in Section 5.3.

## 5.3 Implementation Details

In this Section, I discuss the hardware modules needed by a vehicle OBU for data acquisition and sharing as well as the specific hardware devices that have been chosen.

### 5.3.1 On-Board Diagnostics

The On-Board Diagnostics (OBD) refers to the possibility of monitoring vehicle vital parameters for implementing self-diagnosis of the vehicle. The monitoring of vehicle parameters is done through an OBD-II connector, whose interface and pinout have been standardized by the Society of Automotive Engineers (SAE) J1962 specification [91]. Although the OBD-II connector is the same for all vehicles, there are 5 different protocols permitted by the OBD-II interface:

- SAE J1850 PWM (Pulse-Width Modulation): $41.6kbps$, standard of the Ford Motor Company;
- SAE J1850 VPW (Variable Pulse Width): $10.4kbps$, standard of General Motors;
- ISO 9141-2: $10.4kbps$, used in Chrysler, European and Asian vehicles;
- ISO 14230 KWP2000 (Keyword Protocol 2000);
- ISO 15765 CAN (Controller Area Network): $250kbps$ or $500kbps$, developed by Bosch.

Each vehicle uses one of these protocols, which depends on the brand and model of the vehicle.

The OBD-II provides access to data from the Engine Control Unit (ECU), from which many vehicle parameters can be read such as vehicle speed, engine revolutions per minute, oil temperature, etc. The OBD-II parameter identifiers (PIDs) permit to identify the vehicle parameters read from the ECU [92].

For my tests I used an ELM327 Bluetooth OBD-II adapter, which can be connected to each device equipped with a Bluetooth interface such as a smartphone or a laptop. It costs about 10.

### 5.3.2 Global Navigation Satellite System (GNSS)

A GNSS is a system of satellites that provide autonomous geo-spatial positioning with global coverage. Currently, there are 4 GNSSs:

- Global Positioning System (GPS), owned by the United States;
- GLObal Navigation Satellite System (GLONASS), owned by the Russian Federation;
- BeiDou Navigation Satellite System (BDS), owned by the China;

- Galileo, owned by the European Union.

While the first 2 are fully operational, the last 2 are only partially operational.

A GNSS is fundamental for acquiring continuously and with high precision the position of the vehicles. For my tests, I used the Telit SL869 module, which supports the following GNSSs: GPS L1, Galileo E1 and GLONASS L1 [93]. It costs about 15.

### 5.3.3 Internet connection

An Internet connection is fundamental to enable a vehicle to communicate with my SIoV platform. Since the vehicle is a mobile object and the vehicle data typically does not require a high data transfer rate [94], 3G wireless mobile Internet services are the best choices. In fact, services advertised as 3G are required to meet IMT-2000 technical standards which include a peak data rate of at least $200kbps$. However, current 3G Internet services provide higher speed than this minimum data rate, making them appropriate for the communication of the vehicle data with the Internet. For my test, I used a 3G USB dongle which enables the connectivity to high speed WCDMA cellular networks and costs about 20.

### 5.3.4 Wireless short-range communication

Wireless short-range communications take place between vehicles or/and between vehicles and RSUs. The 3 main wireless short-range protocols are the IEEE 802.11p, the IEEE 802.15.4 (ZigBee) and the Bluetooth. While the IEEE 802.11p standard has been specifically defined for V2V and V2I communications in the dedicated short range communication (DSRC) spectrum [95], the IEEE 802.15.4 and the Bluetooth technologies are mostly used for Wireless Sensor Networks and Internet of Things applications. However, also these technologies can be used for vehicular purpose but with limited applicability because they need a longer connection time than the 802.11p and are not suitable for dynamic mobility scenarios. For example, ZigBee and Bluetooth can be used to establish V2V and V2I communications in static situations such as between vehicles stopped at the traffic light or between vehicles facing a queue.

For my tests, I used a Bluetooth 4.0 USB dongle which costs about 5.

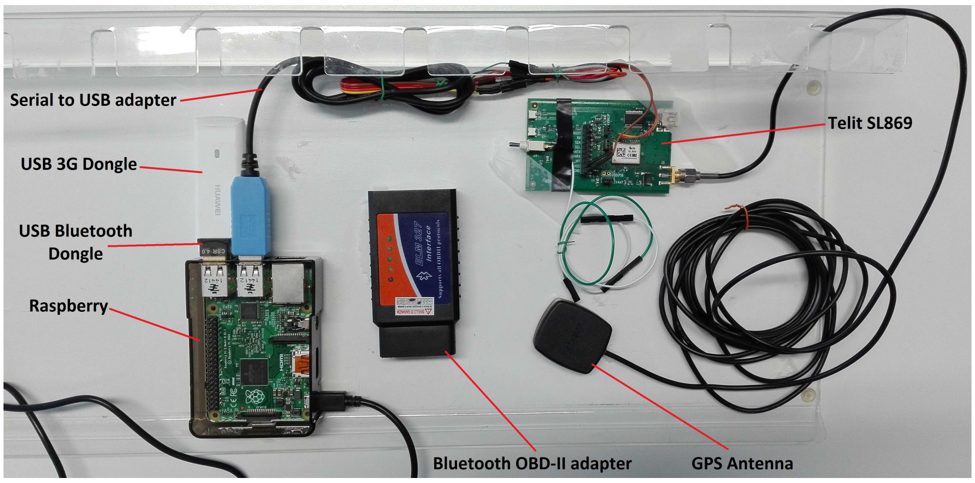


Figure 5.7: OBU implemented for our tests.

### 5.3.5 Control board

A control board is needed for collecting all data acquired from the vehicle and for allowing V2V, V2I and Internet communication. Therefore, the control board must have the needed interfaces for connecting with the OBD-II connector, the GNSS receiver and the wireless short-range module. Furthermore, it must act as a gateway, allowing the vehicle to be connected with the Internet. The control board together with all the other hardware modules form the On-Board Unit (OBU), which implements all the software modules described in Section 5.2.2.

For my tests, I chose the Raspberry Pi 2 model B as the control board [96], as it has all the requested functionalities and a low cost (about 30). The USB Bluetooth dongle and the 3G USB dongle are connected to 2 of the 4 USB interfaces of the Raspberry. The Raspberry communicates with the Bluetooth OBD-II adapter via the Bluetooth interface, which is also used as wireless short-range communication interface for scanning the presence of close vehicles. Finally, the Telit SL869 module communicates with the Raspberry with a serial to USB adapter. The implemented OBU is shown in Fig. 5.7 and has a total cost of about 80.

### 5.3.6 Analysis of an use-case of remote monitoring

In this section, I use a social remote monitoring scenario as an illustrative example to show how the system works. The manufacturers will be able to access the telemetry parameters of the vehicles to locate faults and call the vehicles back for repairing. On each vehicle is present an

OBD-II interface that allows for making requests to the ECU (Engine Control Unit) for current and expected values of the sensors in the engine. The exchange of the data from the OBD-II interface to Lysis platform would enable the various manufacturers / mechanical workshops to check the status the vehicles in order to be able to call back them in case of fault/pre-failure messages. This use case shows how it is possible to allow real-world objects, such as vehicles, to transmit data from the world around them to a virtual counterpart that might expose them on the Internet for the benefit of the user himself. In addition, this use case shows how objects can create social relations between them to be used to navigate more efficiently the objects network. In Lysis platform, once the user launch her application of monitoring, it search for all vehicles belonging to her in a autonomous way. Then, the application requests for the vehicle sensor and actuator list to the related SVO. Fig. 5.8 shows all sensors detected by the associated vehicle. The data from each sensor can be hidden if not considered of interest by the user, or It is possible to expand the sensor details for the values gained during the trip. On the abscissa for each sensor is shown the time at which the measurement occurred. Furthermore, the application allows for setting the alerting thresholds for each sensor. When the vehicle, during its normal movements, exceeds any of the set thresholds, the application sends a warning message to all devices which have a relationship of ownership with the vehicle to warn of threshold crossing.

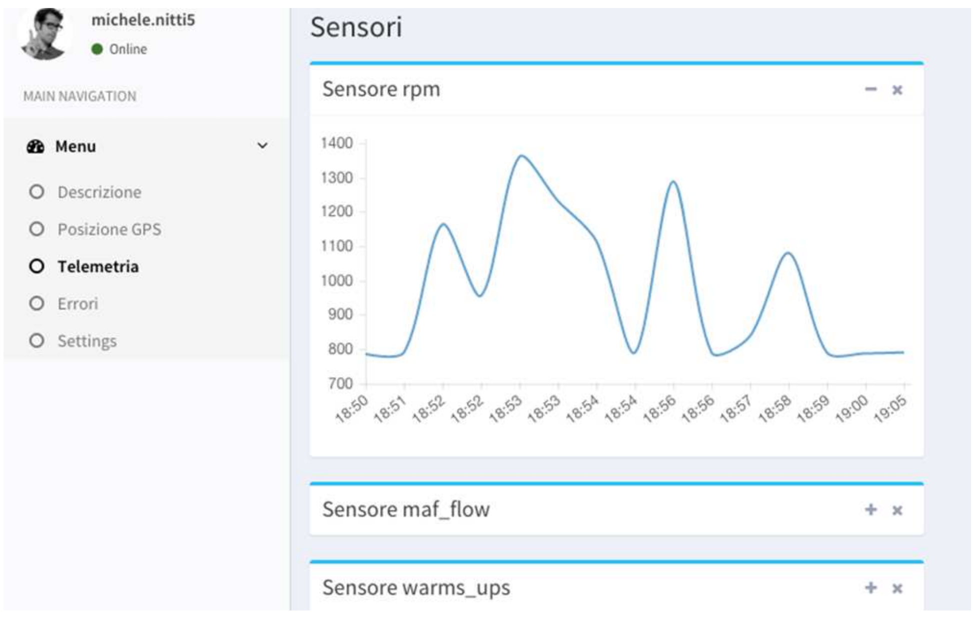


Figure 5.8: The remote monitoring application: sensor value diagrams

## 5.4 Experimental Evaluation

In this section I present the results of simulations that are aimed at analyzing the major characteristics of the social networks resulting from the proposed SIoV.

### 5.4.1 Simulation Setup

To conduct my analysis, I would need mobility traces of a large number of vehicles. Since this data is not easily available, I used a dataset generated by SUMO (Simulation of Urban MObility) [97]. SUMO is an open source traffic simulator designed for road networks of large cities, by mean of which it is possible to generate traces of people mobility and every type of vehicle (cars, buses trucks etc.). Particularly, I used an existing traffic model for the city of Cologne [98], which starting from a set of vehicles allows SUMO to generate synthetic traces taking into account traffic lights, traffic jams, critical hours and so on. The considered dataset was related to around 10,000 vehicles. Assuming vehicles with 802.11p interfaces, I simulated the establishing of SOR-type relationships for different intervals of visibility and for different distances. This type of relationship depends on the extent of time two vehicles stay in contact, and usually when vehicles are still in a jam next to traffic lights. CWOR is another relationship that depends on node movements, but it requires the availability of information about the position of RSUs that are not in the model yet. For this reason, I have decided to proceed with SORs only. Each vehicle recognizes any other in the SIoV by means of 802.11p. However, for those vehicles not equipped with a 802.11p interface, GPS location can be used as well. Indeed, thanks to the SIoV, vehicles equipped with a 3G interface can take part of the social network since their GPS location can be used to detect their position and then estimate the visibility with the vehicles around them.

Several articles in literature use 250 meters as upper bound of distance in urban scenarios [99] [100] for 802.11p communication, so I have chosen the same limits for coherence with other works even if I could have considered bigger distance because I am not interested in throughput evaluation but only in visibility among the vehicles. In this work, I focused on SOR setting-up only considering average visibility distances. A more accurate study could take into account path loss computing algorithms [101] but I chose of leaving theirs integration in SUMO for future works.

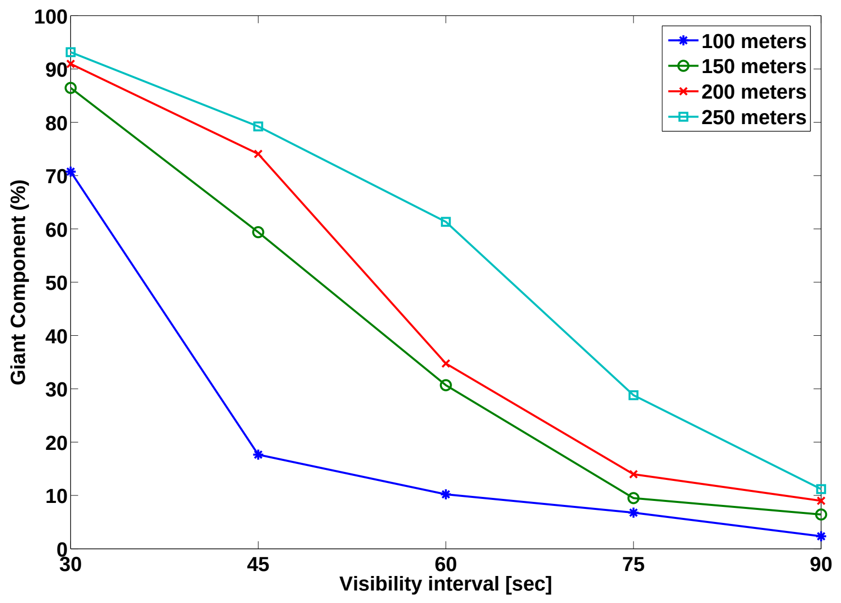


Figure 5.9: Giant Component for different visibility ranges and different visibility intervals

### 5.4.2 Simulation Results

Figure 5.9 shows the giant component for all visibility interval widths and distances. The giant component is represented by the largest set of connected vehicles. As I expected, it is not possible to achieve a giant component including 100% of nodes with only SOR relationships. In case of a visibility range between 250 and 200 meters, the graph is not connected but shows high percentage of nodes in the giant component. Generally, a visibility interval of 30 seconds allows high values but the relationship rate decreases rapidly with increasing intervals. It is understandable since vehicles are very fast moving node in rapidly changing scenarios.

Note that the curves of 200 and 250 meters tend to have almost horizontal tangents for short intervals of visibility, whereas 100 and 150 meter curves tend to have vertical tangents. This is due to how the groups of vehicles are created. Indeed, for long visibility ranges, there are medium size groups interconnected by means of few links forming the giant component. With longer visibility intervals, those links break, resulting in a smaller giant component. Even if the graph is not connected, in the SIoV there are further relationship types that could let the graph to be connected. Indeed, the POR relationship would introduce quite useful connections among vehicles. However, the unavailability of information about the vehicles brand prevents me from introducing this information in the simulation.

# Chapter 6

# Alternative Cloud Infrastructures: Lysis on the Edge Cloud

Like Lysis, implementations of the SIoT model envision cyber counterparts of physical objects, which I call social virtual objects, virtualized in the cloud. Such an approach has several advantages but suffers from a few major problems in some scenarios. In fact, objects might be located far away from the data center hosting the cloud, which results in long delays and inefficiency in the use of communication resources. This chapter is structured as follows: firstly, it investigates how to address the above issues by exploiting the computing resources in the edge of the network to host the virtual objects of the SIoT and provides early experimental result; secondly, I show the challenges to address for the implementation of social virtual objects in a edge-based project, named INPUT.

## 6.1 Preliminaries

As shown in Section 1.6, the integration of Cloud computing features in the IoT landscape allows for the deployment of heavy processing applications, not supported by many resource-constrained IoT devices. to this, data generated by IoT devices can be sent and processed in the cloud, which has almost unlimited amount of computational and storage capabilities.

The introduction of solutions which, as Lysis, envision the instantiation of digital representatives ("virtual objects") of the physical objects in the cloud led to the so-called Cloud of Things. Consequently, added value applications can be implemented by exploiting services offered by virtual objects. These applications can be developed independently of the specific hardware features of the IoT devices.

However, for real-time applications with strict requirements in terms of delay, the latency caused by interactions with/between virtual objects in the cloud are unacceptable. To this, it becomes of paramount impostance to evaluate the use of an infrastructure as it is the edge cloud, which allows cloud services to move much closer to end-users' devices. These applications will be capable of cooperating with and offloading corresponding applications residing in the users' smart objects and in conventional cloud centers, to offer innovative services. This will allow user requests to be manipulated before crossing the network towards data centers in ways that enhance the performance. Manipulations include pre-processing, decomposition, and proxying. This solution perfectly matches the promising Cloudlet and Fog computing paradigms [70] and is testified to also by European projects, such as TROPIC and INPUT. According to the Fog computing paradigm, network edge devices are evolving into micro Cloud servers, able to host not only advanced network functions but also application modules. This allows for instantiating distributed functionalities, accommodating the desired level of QoE (Quality of Experience) and workload/traffic volumes, closer to the end-users. Furthermore, the distributed architecture composed by large-scale geographically deployed nodes natively possesses *scalability* properties. In this context, I show an investigation on requirements to implement Lysis architecture by exploiting edge computing capabilities.

## 6.2 Social Virtual Objects in the edge cloud

### 6.2.1 The SIoT cloud-based architecture

As already shown in chapter 2, Lysis platform has a PaaS-oriented design and foresees a four levels architecture. In Fig. 6.1), I want to highlight the layer of interest for the edge computing compared to the original solution based on conventional cloud

Key part of the overall solution is the SVO, equipped with interfaces to establish a secure connection with the RWO, and allow for a standardized communication procedure between the aggregation level and the extremely variegate set of physical devices. It implements the functions described in [102], among which service discovery is the one of major relevance for this context. This relies on the SVO search process aiming at finding the potential providers of information among the SVOs of the SIoT community. This distributed process is accomplished for each user by the so-called SVO Root (SVOR), which is elected among all the user SVOs so that each user has her own SVOR. The SVOR accepts requests for SVO search from the upper

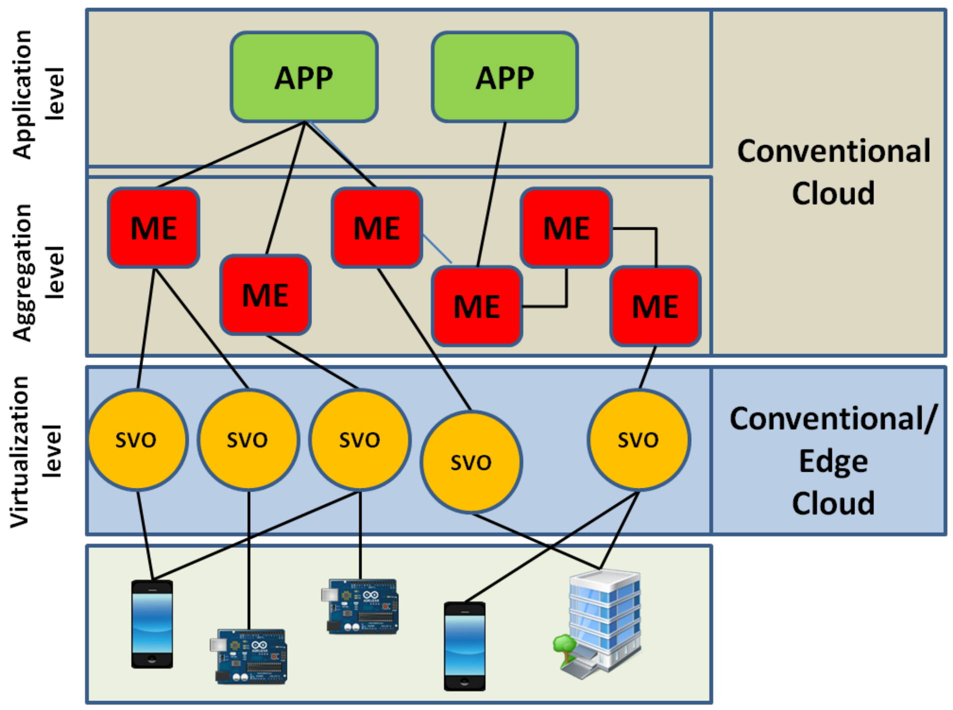


Figure 6.1: Cloud-based SIoT architectural solution

levels and returns the address of the resources by browsing the social graph of objects and the API keys to access them, as described in Chapter 2.

### 6.2.2 Exploiting the edge cloud

The platform envisaged in this chapter aims at exploiting the benefits of a distributed Cloud service infrastructure and Fog computing paradigm as discussed in Section 6.1. Therefore, it is designed to leverage a distributed Cloud solution and to dynamically deploy SVOs closer to their physical counterparts.

The Lysis platform, implemented using the Google App Engine (GAE) PaaS, is the starting point of this investigation activity. GAE offers useful APIs to: (i) implement on each SVO a template repository of friends by means of a document representation enabling full-text search through the social graph, and (ii) create a uniform repository of locations, which are needed to establish C-WOR and C-LOR social relations based on the object positions. The highlighted advantages of a distributed cloud paradigm imply a profound transformation of the Lysis platform. In fact, it relies on a conventional cloud system, which foresees a permanent execution location of Web Services localized in classic partitioning of Availability Zones (AZs) and Regions; thus, no geographical distance between real objects and the virtual counterparts is taken into account.

To take full advantage of the short distance between edge cloud and devices, supplementary architectural elements are necessary.

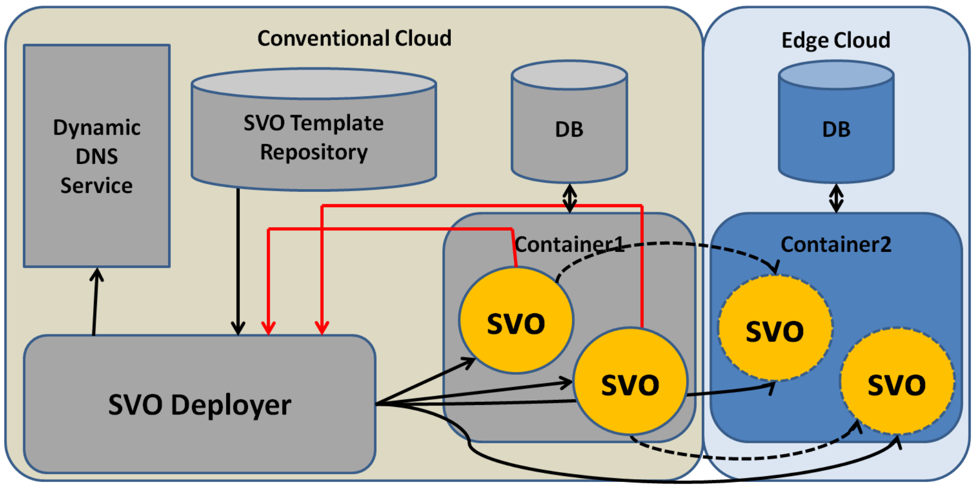


Figure 6.2: The elements in the virtualization level involved in deployment/migration of SVOs

The deployer, see Fig. 6.2, is the component that is in charge of creating the SVOs whenever the user registers a new device. The owner of the devices triggers the creation of a new SVO selecting the right template from the repository, so that a new SVO instance is created in the container located in the owner cloud space. During this phase, the new SVO inherits the semantic description of its template, which is recorded in its profile stored in the owner database space. As soon as the SVO is created, the owner associates the SVO to the physical device, which sends the basic information. The user can add further descriptions with contextual information. The communications between the SVOs is based on RESTful web services. However, to set up the communication the SVOs need to be friends, which is the condition to reciprocally know the respective URI and the API friend key.

Fig. 6.2 highlights the main requirements for SVO deployment and migration. In the edge-based platform, the SVO is a web service that requires two basic elements for its implementation: a container and a database. The container is an execution environment for web services and has a role in both phases of deployment (when users register new objects onto the platform) and migration (when the SVO itself asks the deployer module for a migration).

A first key component to provide, when operating in the edge cloud, is a module able to monitor all parameters, such as QoS, latency, power consumption, which represent the basis for the decision about the opportunity to migrate a SVO from the conventional (remote) cloud to the edge cloud. Whenever a migration is performed the relevant DNS (Domain Name Server) record needs to be updated. When an SVO detects that a migration has to be performed, it sends a relevant request to the deployer together with the history of sensor data and relationships that are stored in the DB in the conventional cloud. The migration request is always accomplished after the completion of the running process. Since the SVO has been designed as an event-driven system, its state can be coherently saved into the DB after every event-triggered process, and it comprises all the relevant SVO information. Thus the SVO migration process requires the deployer to put the SVO template into a container in the edge cloud and to make a DB dump of all data associated to that SVO. The SVO data is copied on the container's DB in the edge cloud. Finally, the new SVO IP is registered in the DNS server and the old instance is disabled.

Several approaches, all aimed at improving the application QoE, can be used by the SVO to decide whether and when to request a migration. The main aspects to account for are:

- **Frequency of communications between the physical object and its SVO**: communications are implemented through inter-domain transmissions and their frequency depends on the applications that are running in the physical device; it can also be predicted based on installed applications and user profile.
- **Frequency of SVO search process instances**: these instances trigger interactions among SVOs till the potential service provider is found. If all the SVOs are in the conventional cloud then the transmission latency is limited. Otherwise, if some SVOs are already migrated to the edge, then it may happen that edge-to-cloud and edge-to-edge communications are needed.

## 6.3 Analysis of performance

A numerical evaluation has been conducted in MATLAB® for a wide set of scenarios, to observe the achievable performance in terms of average number of messages (and relevant cost in terms of latency) exchanged by an object to communicate with its virtual counterpart when moving the SVOs from the cloud to the edge node.

The cited performance analysis would need information about the position and mutual relationships of a large number of objects. This data is not available to date, as real applications have not been deployed yet. For this reason I resorted on the real dataset of the location-based online social network Brightkite obtained from the Stanford Large Network Dataset Collection [103] and extended it to take into account social relationships. I assume that every person carries one smart object, for example a smartphone, so when they get in touch with their friends their objects also come into contact and have then the possibility to create a SOR.

Additionally, on the basis of the mobility of the humans carrying their objects, the C-WOR, C-LOR, and POR relationships are created. The resulting SIoT network has around 5k nodes and 36k edges.

In my setting, I consider that every node updates its status with a frequency randomly chosen from the interval $[6, 60]$ updates/hour. Each interaction among couples of objects begins by randomly choosing an SVO client and, subsequently, selecting the SVO that can provide the requested service according to the geographical distance among them, by following an exponential distribution. This models the location-based interest of the information, which is especially true in the ITS use case we are considering [104]. I adopt a migration strategy based on how often physical objects update their status; in particular, I migrate the SVOs in descending order of their status update frequency.

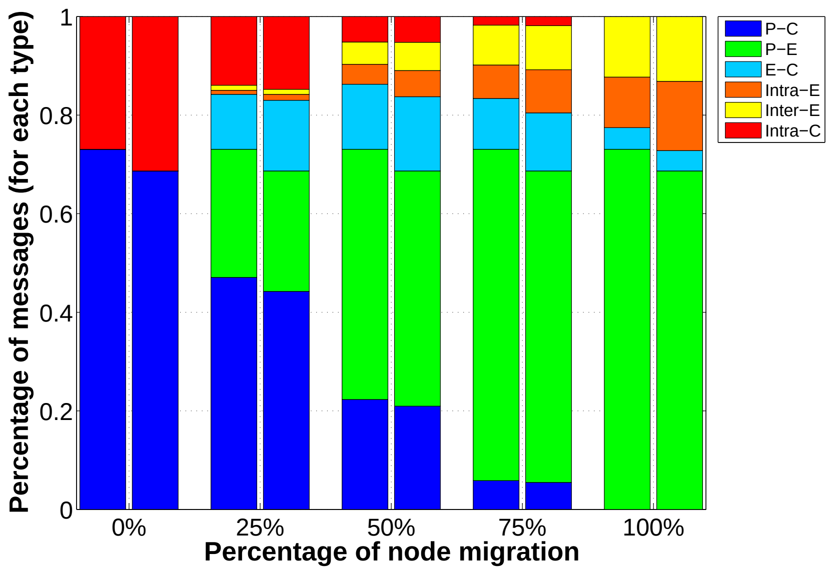


Figure 6.3: Breakdown of the exchanged messages

Fig. 6.3 shows the ratio of each type of message in several scenarios. Results for different values of the percentage of migrating nodes are provided. Furthermore, for each percentage value one observes two bars. The one on the left side is obtained when the interacting objects are 20 km distant from each others, whereas the one on the right side is obtained when the distance is 100 km. A bigger distance brings to a slightly higher number of hops to navigate the network and, therefore, in percentage, to more messages relevant to service discovery w.r.t. status update. This effect can be observed when all the SVOs are in the cloud and there are only P-C and intra-C messages. In fact, the frequency of intra-C messages increases, thus lowering the P-C messages' frequency. Moving the SVOs from the cloud to the edge leads to a reduction of P-C and intra-C messages, which eventually are not involved anymore when all the SVOs are migrated, and to a growth of the messages involving the edge. In particular, physical objects will update their status on the SVO in the edge node and the SVO search process will only involve SVOs in the edge nodes. However, the cloud is still involved in some communications, i.e. E-C messages, when the application located in the Cloud sends a request for an SVO search to the SVOR. It is worth noting that, even with a complete migration, the difference in the number of inter-E messages between the 20km and 100km scenario, is limited. This is achieved following the navigation mechanism of the social networks. In fact, as explained in [105], long distances are usually travelled with a few hops, usually no more than one or two, and a fine-grained search is then carried out locally.

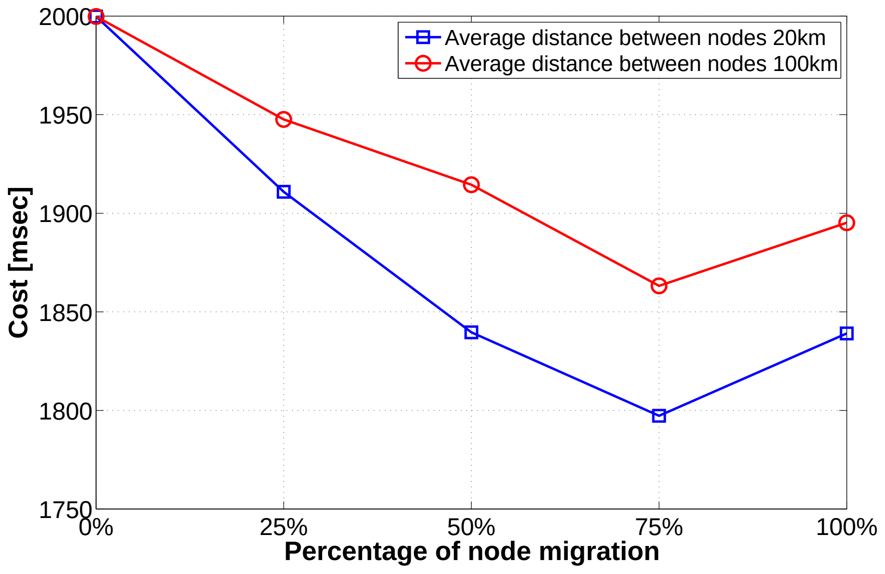


Figure 6.4: Cost of the message exchange versus the percentage of migrating nodes for different distances between interacting objects.

However, messages have not the same cost in terms of latency. To account for their relative importance, I focused on a generic scenario and calculated the latency for each type of message over several runs, when considering different cloud providers and edge nodes. The obtained latency of P-C messages is equal to 60msec; this value corresponds to the sum of the costs of P-E and E-C messages, which are 42 msec and 18 msec respectively. Messages between different edge nodes (Inter-E messages) are the most onerous if one considers the worst case when they have to traverse low speed routers of different service providers w.r.t. routers in the core of the Internet network. These messages have a latency of 72 msec. Nevertheless, the latency of inter-E messages represents an upper bound since messages traveling from one edge node to another of the same Telco operator have a latency similar to the E-C messages. Finally, intra-E and intra-C messages have a negligible cost in terms of latency.

I finally point out that I did not take into account the *una tantum* migration cost from the cloud to the edge node. However, due to the mobility of users with their associated physical objects, it could be required to migrate a SVO from an edge node to another. In this case, additional design choices have to be taken into account.

## 6.4 An exemplary workflow for an ITS use-case

As briefly introduced in Section 6.1, to better describe the workflow of the proposed platform I focus on ITS scenarios, which have greatly attracted the attention of both Cloud and Social Computing research communities [106], [107].

In my ITS use-case, Angela has bought a new smart car and the corresponding SVO has been added to her cloud of things in the platform. The registration phase foresees the creation of the SVO, using the fittest template, and the installation of the appropriate HAL (Hardware Abstraction Layer) module on the physical device, to enable interaction with its digital counterpart. In this way, the SVO can expose and provide access to physical devices' resources, and start the creation of the social relationships with other SVOs. In case of further resource-constrained devices embedded in the vehicle, such as driver monitoring or environmental sensors, the car can act as a gateway for their SVOs in the SIoT framework, implementing the short-range communications with the physical devices.

By browsing the online application store of the platform, Angela selects the novel Social-IoT-based NAVigation (*SIT-NAV*) system, which exploits objects cooperation and resource sharing to assist her in selecting the optimal route towards the destination. To plan the route, Angela can define the preferred rules/conditions for the trip. Accordingly, the platform deploys the *SIT-NAV* service with the relevant modules (MEs). Each ME requires input data, which can be provided by the available SVOs, through an appropriate service discovery.

In Fig. 6.5, a sample scenario is depicted where at time $t = t_0$ Angela is interested in finding the best route to the airport and an estimation of arrival time to be sure she can reach her flight on time. Cars, traffic lights and other objects in the environment collaborate exchanging the relevant information through their friend SVOs at the edge node. The SIoT is supported by the remote and edge clouds. The former represents the conventional cloud servers with significant resources, whereas the latter refers to micro datacenters deployed at the edge of the network. Some SVOs may be located in the remote cloud, whereas most SVOs are located in

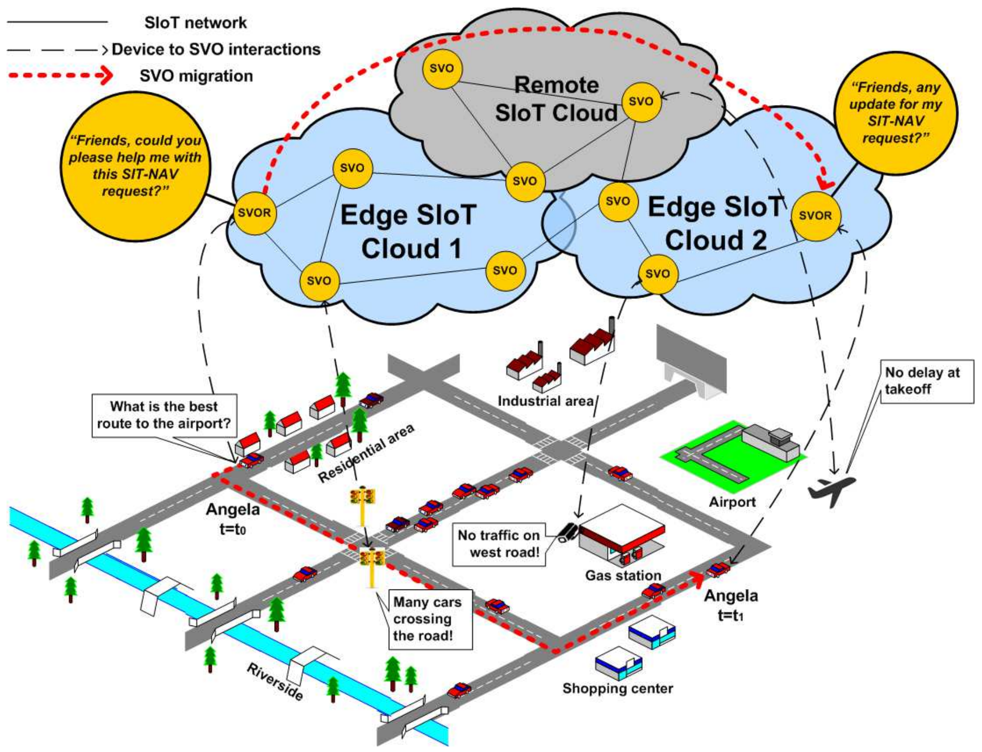


Figure 6.5: Example scenario for the SIT-NAV application.

the edge clouds for more efficient SVO updating and service discovery. Each ME composing the SIT-NAV requires appropriate information to perform the relevant services and forwards the request to the user' SVOR (in our scenario Angela's vehicle). By exploiting the social relationships, the SVOR will obtain the addresses of the other SVOs, which are able to provide the desired resources, and will communicate them to the ME. In this way, the ME could request or subscribe to the desired SVOs' resources. When the user and its associated devices will move along the path, the serving edge node may change and an SVO migration may be triggered according to the SVO migration policies. Referring to Fig. 6.5, at time instant $t = t_1$ the SVO for Angela's vehicle is moved to the second edge node. In doing this, the probability of obtaining the required service/data from co-located SVOs is increased by having virtual images of physically nearby devices at the same edge node in the network.

## 6.5 Integrating the Social Internet of Things in the INPUT Architecture

In the following I present a solution that integrates the SIoT concept in an infrastructure proposed within the INPUT project. More specifically the feature is exploited of the INPUT project which allows for running the virtual representation of a smart/social object in the access router which is nearest to the physical object. In this way it is expected that delay will decrease and efficiency in the usage of network resources will increase.

### 6.5.1 The INPUT platform

The European H2020 INPUT (In-Network Programmability for next-generation personal cloUd service supporT) Project [12] aims at exploiting SDN and NFV (Network Function Virtualization) [13] paradigms to achieve breakthrough towards a fully softwarized network of Telco providers, enabling next generation cloud applications. In particular, the objective is to (i) dynamically deploy application-level services and network-specific functions in the edge network devices, (ii) design a distributed architecture composed by computing and storage appliances and by physical/virtual SDN switches, and (iii) move cloud services closer to end-users.

### 6.5.2 INPUT architecture

Virtualization represents a fundamental aspect of the INPUT architecture, which aims to replace physical Smart Device (SD) with their virtual images, thus introducing the concept of Smart-Device-as-a-Service (SDaaS). This novel paradigm can be exploited:

- to fully dematerialize classic users home appliances (e.g., set-top-boxes, video recorders, network-attached storage server, etc.) and provide all their functionalities by the cloud;
- to add potentially infinite smartness and capacity to resource-constrained IoT devices (such as sensors and actuators), whose physical counterparts must be maintained to perform their location-based monitoring and control functions.

Virtual and physical SDs will be made available to users at any time and at any place by means of virtual cloud-powered Personal Networks (PNs), which will constitute an underlying secure and trusted communication service. These PNs will provide users with the perception of always being within their home Local Area Network composed of their own (virtual and physical) SDs, independently from their location (see Fig. 6.6).

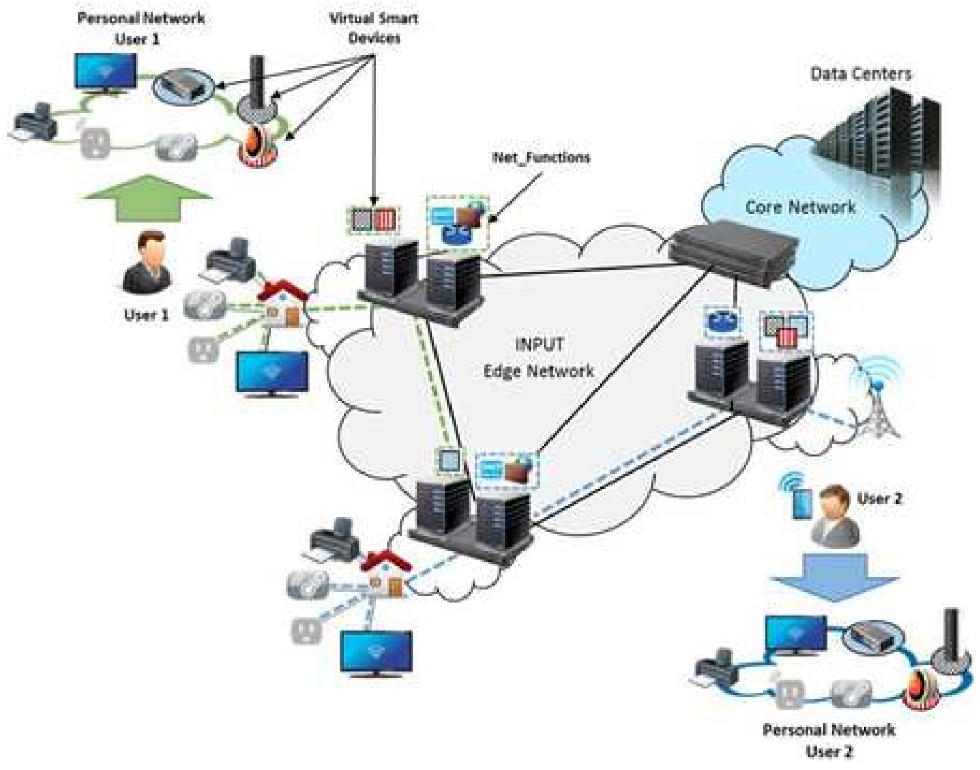


Figure 6.6: . Mapping of the Users Personal Network in the INPUT edge network [12].

To achieve these ultimate objectives, the INPUT Project will enhance edge network devices with computing and storage capabilities, enabling the so-called in-network programmability. This will allow edge network devices to host cloud service applications, which will cooperate with running services in users terminals and remote datacenters.

On the other hand, physical SDs typically connected to the users LAN (e.g., network-attached storage servers, set-top-boxes, video recorders, home automation control units, smart meters, etc.) will be fully or partially virtualized in the INPUT computing facilities by software instances, named Service Applications (Service-App), running at different levels of the edge network infrastructure and providing application layer services.

### 6.5.3 Benefits of INPUT platforms for IoT applications

The INPUT platform provides the ideal substrate to deploy a win-win solution for service cloud providers, Telco provider and users. The current landscape of IoT cloud platforms foresees the virtual representation of IoT nodes. This could imply lack of interoperability among the interfaces defined for the virtual objects and cause significant latency due to the remote location of datacenters. IoT applications present extremely different QoE (Quality of Experience) requirements, ranging from real-time services with stringent deadlines to multimedia applications with strict requirements in terms of bandwidth and jitter, to delay-tolerant scenarios where sensing information should be stored in remote servers, introducing big data issues for their storage and processing. Therefore, the INPUT platforms allows service cloud developers to deploy new cloud-based IoT applications, able to guarantee those service requirements which could not be satisfied by remote datacenters (far from the user). Furthermore, the inherent distributed architecture of INPUT enables scalability features, avoiding server overloading by distributing the load among multiple micro datacenters in the access points of the network.

The Telco providers could implement SDN and NFV paradigms to innovate their network and to open new business areas, offering cloud services from a privileged position. In fact, NFV allows to deploy on-demand services in the telco nodes closer to the users, matching the manifold requirements of applications and allowing for a reduction of Operating and Capital Expenses (OPEX and CAPEX). Thus, Telco providers could increase their revenue opportunities in the promising IoT domain, offering not only network services but also shifting their role to a higher position in the value chain.

These advantages have to be analyzed considering that whereas the provisioning of computing and storage resources are typically location-independent, sensing activities are strictly correlated to their physical positions. According, with a centralized cloud platform the data should be sent from the sensing device position to the central servers and then back to the user. With the distributed cloud solution, sensing information could be processed closer to the devices, thus reducing traffic congestion in Telco network. Furthermore, more processing power close to the user can extend the number of applications and achieve a better QoE experience.

Finally, as remarkable energy consumption reduction is envisioned by the INPUT platform, the advanced virtualization technologies will allow to partially dematerialize end-user IoT devices (zeroing their embodied carbon footprint) and to host their virtual images on a more energy-efficient network infrastructure.

## 6.6 Introducing the SIoT concepts into the INPUT platform

### 6.6.1 SIoT integration and added-value IoT applications

Virtualization of devices is a fundamental aspect in the INPUT platform. The SDaaS paradigm allows for creating virtual representations of users devices. These virtual representations could implement all the functionalities of a device, which could even exist in the virtualization layer only, e.g., the virtual Smart Devices number 4 (vSD4) in Fig. 6.7. For instance vSD4 may be a process running in the cloud which collects physical world information from the web without having a reference physical Smart Device (pSD) counterpart. For devices like IoT nodes the virtual images allow for a standardized interface to access their heterogeneous resources and maintain up-to-date metadata, such as: object type, computational power and mobility capabilities. Finally, the service layer can aggregate and compose the different vSDs functions to create advanced cloud service applications.

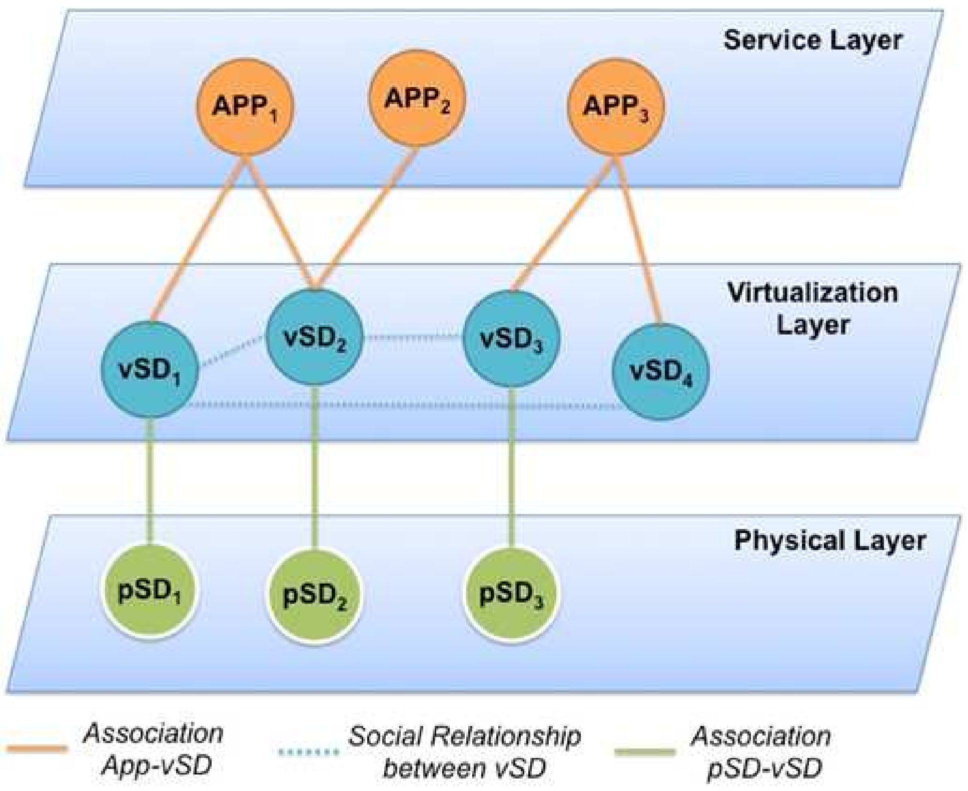


Figure 6.7: Logical associations between physical and virtual Smart Devices

Similarly to the conventional case of Lysis,to integrate the SIoT concepts in the INPUT platform, a middleware is needed to enable the *virtual image* of the physical SD to perform social functionalities and then move the virtual representation of the physical SD from the cloud into the edge nodes of the INPUT platform.

In this platform, every *type* of physical object can be described by a Profile, which consists of a semantic description of the object and its capabilities, resources and permissions. The virtual representation of a SD is actually an instance of the Profile and represents the actual web service running on the Cloud. Social functionalities are assured by the presence of a middleware, common to all the Profiles, which is needed in order to allow the virtual objects to exploit their own relationships.

Depending on its capabilities, the physical device is connected to its virtual counterparts through different communication models and protocols, from HTTP over SSL to MQTT. Additionally, to address the dissimilarities in the hardware characteristics of the devices an abstraction layer of the hardware is used. This is in charge of creating an encrypted secure point-to-point communication channel with the vSD. The association among the physical SD and the vSD is done during registration when the two abstraction layer entities on both cloud and device exchange object IDs, URL and encryption of the key.

vSDs can access resources based on different permissions: public, private and friend. Resources exposed as public do not need an access key and are then available to anyone. A private permission means that only the owner of the object, and its applications, can access the resources, through the use of an Owner Key, while if the permission is set to friend, then the resources are shared with other vSDs when a relationship is established, through the exchange of a Friend API Key.

The social middleware exposes the following functionalities. The relationship management introduces the intelligence that allows objects to start, update, and terminate relationships, on the basis of the objects features (such as: object type, computational power, mobility capabilities, brand) and activity (frequency in meeting the other objects). For example, some relationships are determined by the static characteristics of the object (or slowly varying characteristics): type, brand, ownership. The other kinds of relationship are determined by the movement of the object and by the other objects it comes across. Clearly, the configuration of these functions is controlled by the object owner; accordingly, the resulting links can be asymmetrical.

The second component is the service discovery that has the purpose of finding which objects can provide the required Service App in the same way humans seek for friendships and information, i.e., by navigating the social network of friends. When a user needs a particular cloud service, this module is activated in one of the vSDs of his/her personal network, e.g., in the vSD associated to the smartphone, to find the Service Apps required to perform the specific tasks. The process starts with the smartphone's vSD looking for friends in its local database, which can provide the requested Service Apps; if none of the friends is a positive match, then the request is forwarded to other friends by making use of social measures, such as centrality or local clustering, until a set of objects capable to provide the needed Service Apps is found. If, by any chance, the smartphone vSD is down, the system is able to start the discovery process from any other object inside the user personal network, thus avoiding single point of failure.

To construct the chain of Service Apps, a service composition module is required, which enables the interaction among vSDs. However, in the IoT scenario, many objects, and consequently vSDs, will be able to provide the required Service App(s), therefore to build a reliable system a trustworthiness management module [71] is needed as a mean to understand how the information provided by other friends has to be processed on the basis of their behavior. To this aim, since the forms of socialization among objects have been devised to represent the human relationships, they can be ranked based on the type of relation that links two objects. Between two vSDs that belong to the same owner, and therefore are in the same personal network, we can infer a higher level of strength with respect to the relation between two objects of the same brand but that potentially never met.

The SIoT solution seems to provide several promising added-value features for the INPUT architecture. Indeed, the SIoT structure can be shaped by creating or deleting relationships in order to allow vSDs to easily find the desired Service Apps, for example by creating many connections for popular vSD. Moreover, a mechanism of resource discovery based on social relations is intrinsically distributed, since every vSD can search for information by crawling its own social network and no central database is needed. In the same way, it is possible to use the social network to define model for the management of the level of trustworthiness among the objects not reachable in other ways.

## 6.7 Challenges for implementation

To enable the virtual objects to exploit social capabilities, we need to face the following challenges:

### 6.7.1 Virtualization platform

A Platform as a Service (PaaS) is required to deploy and run any virtualization in the Cloud. First, a container is needed. The choice of the container depends on the virtualization type: it is possible to use an active virtualization, like a web-service that has its own independent code, or a passive virtualization, like a representation on a database. The choice of a database server is another important choice to be done: relational or NoSQL database? The first is reliable in terms of consistence of transactions but suffers in cloud performance and it is stiffer due to the use of fixed structures (tables). The second is more suitable to a changing environment such as the IoT, but the fluidity of a dynamic cloud could affect consistence of data.

### 6.7.2 Naming service

In order to reach the actual vSD running instance a naming service is needed. It must track any change of the cloud network as regards to the locations where vSD have migrated to. Any communication in the social network should be possible without the need of knowing where vSDs are actually running. A CaaS (Container as a Service) cannot provide a way to map static IP addresses to a vSD. In order to optimize the network path between an end user and a vSD (or between two vDSs), end users on different ISPs or geographic locations might use different IP addresses to access the same vSD. DNS might return different IP addresses to access vSD over time or from different network locations. A possible solution could be the use of a hierarchical DNS system combined with Persistent URLs (PURLs) Type 303, which is used for Web Resources. When this solution is selected, the PURL mechanism will perform the dynamic monitoring of vSD addresses updating current temporal addresses and maintaining the persistent address for each vSD [6].

### 6.7.3 vSD migration handling

In order to manage vSD migrations from an edge node to another a controller/orchestrator is needed. It should be able to move a vSD code to a container and the related data to a database on another edge node. The migration must be accomplished in a seamless way to avoid service delivery failures. The migration decisional logic could be on the controller or on vSDs. In the first case, the controller has to observe vSD interactions and it has the algorithm needed to make a choice of migration. In the second case, vSDs decide when migrate and ask for a new deploy to the controller. Lastly, the choice of using a centralized controller (although in cloud) or a hierarchically distributed controller system depending on the cloud partition into zones.

### 6.7.4 Social network management

Social information on vSDs is crucial to fully exploit SIoT advantages and where it should be stored is to be taken carefully into account. In a subjective scenario, vSDs only have their own view of the social network. On the other hand, in an objective scenario, social information is kept on a level on top of the vSD level and is accessed by vSDs and other actors in the same way. Finally, a hybrid solution could give the best compromise when elements out the social network need this information (e.g. deploy controller).

# Conclusions

In this thesis I have presented the IoT platform called Lysis. The major feature of Lysis is that relies on a PaaS (Platform as a Service) model that allows the users to have complete control over their data, which is not always assured by alternative solutions. The platform also fosters re-usability of services and software developed by third parties and users. 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. 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. An use-case is presented to show the way I implemented the Lysis architecture and its potentialities. Performance evaluations are also shown in terms of scalability and reduction of computational load at the physical devices

In Chapter 3, I have presented three algorithms for neighbor discovery needed for detecting new relationships between objects in the Social Internet of Things. This new paradigm, with the intensive use of virtual counterparts in the cloud, demands for new solutions in order to detect as many neighbors as possible without the constraint of direct device-to-device communications. One of the proposed algorithms relies on the scan of the radio channel, whereas the other two assume that this is not possible and make use of the device localization algorithm.

In Chapter 4, by exploiting the aggregation level of Lysis architecture, I addressed the issue of resource management in the Mobile CrowdSensing (MCS), where any sensor-enhanced Internet-enabled object can be used to sense the surrounding environment and acquire knowledge about the context in which it is operating. Exploiting the social attitude of MCS, I built an MCS management framework on top of the SIoT Lysis platform.

In my solution, queries received from the Lysis platform requesting resources needed by

MCS tasks are analyzed and processed. Based on the semantic description of both the query and the Social Virtual Objects (SVOs) associated to Lysis, the query is matched with the appropriate SVOs that are able and available to perform the related sensing task. Accordingly, SVOs' resources, more specifically required power amounts, are fairly allocated so that no node is overloaded with respect to the others. Experimental results achieved testing 7 Android smartphones proved that the devices' lifetime values tend to a single value and multiple applications can use the same outcome of the CrowdSensing Micro Engine (CS-ME) with improvement in terms of latency and computing resources.

In Chapter 5, I have focused on the integration of Vehicular Ad-hoc Networks into the Social Internet of Things, which leads to the Social Internet of Vehicles paradigm. More specifically, I have identified the social relationships that can be established by vehicles and RSUs in the SIoV and I have proposed a middleware, which extends the functionalities of the already standardized Intelligent Transportation System Station Architecture, to take into account all the needed elements to integrate VANETs in the SIoT. I have analyzed the structure of the resulting SIoV network to evaluate its navigability. Results of such analysis show that SOR relationship graph has a large giant component in case the visibility among two vehicles is possible up to 150 meters, at least, and when the link is established even with short intervals of contact. By adding CWOR and POR relationships I expect to achieve connected graphs.

As integration to Lysis, I have illustrated an effective low-cost and flexible solution for enabling vehicles without any kind of connectivity to participate in the Social Internet of Vehicles; moreover, I show how the proposed system can be integrated into the Intelligent Transportation System Station Architecture (ITS SA). The implemented system is able to collect data from the vehicle's OBU and to sense the presence of nearby vehicles. These information are then sent to the Lysis platform hosted in the Cloud, where friendship are created and managed based on the vehicles' behavior and an application for the remote monitoring of the vehicle is implemented. The advantages of the developed system are that it is not bound to a particular technology since all the communications among the vehicles take place at a virtual level, where all the entities *speak* the same language; moreover, the system is able to cope with the vehicles moving in large and complex environments and to guarantee quality requirements of the applications since they are handled by the Cloud.

In Chapter 6, I investigated the use of the edge infrastructures as execution environment for the social virtual objects. The preliminary analysis emphasized the need for strategies to dynamical decide which SVOs to migrate from cloud to edge and vice-versa.

In this context, an important issue to be addressed is the trust of the edge resources where SVOs are hosted. Indeed, trustworthiness of the cloud and of the network edge facilities is considered as granted in my scenario, at least till now. Indeed, the edge cloud should be managed by the network operator that the objects are using for data connection services. As we are experiencing more and more frequently the encryption of the content provided by the Over-the-Top (OTT) service providers (notwithstanding the increase in the traffic overhead), it is plausible that the information associated to the SVOs moved from the conventional cloud to the edge cloud will be encrypted as well. In this way, this information (e.g., collected sensed data, profile of the objects, statistics on the objects usage) will remain private. How the encryption will be performed by the physical devices with limited resources is, however, still an open issue. Additionally, I will also have the issue of trusting the reliability of the computing and storage service at the network edge in terms of availability and retainability. Concerning this aspect I expect that a great role will be taken by the QoE and the relevant impact on the customer churn, forcing the network providers to consider reliability a key aspect not to lose market share.

As analysis of an use-case, I have discussed the advantages that the solution envisioned by the INPUT project can bring in the application of the SIoT concepts and the Lysis architecture. In fact, I have discussed that by bringing the virtual images of the IoT nodes close to their physical counterpart it is possible to reduce delay and increase efficiency of network resources. Furthermore, I have presented how such integration can be achieved as well as the technical challenges which need to be addressed.

# List of Figures

# List of Tables

# Bibliography

[1] L. Atzori, A. Iera, and G. Morabito, “Siot: Giving a social structure to the internet of things,” *Communications Letters, IEEE*, vol. 15, 2011.

[2] L. Da Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] (2016, Sep.) Xively. [Online]. Available: http://xively.com

[4] (2016, Sep.) Paraimpu. [Online]. Available: https://www.paraimpu.com/

[5] (2016, Sep.) Carriots. [Online]. Available: https://www.carriots.com/

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

[7] COMPOSE, “Collaborative open market to place objects at your service,” 2012. [Online]. Available: http://www.compose-project.eu/

[8] R. Pozza, M. Nati, S. Georgoulas, K. Moessner, and A. Gluhak, “Neighbor discovery for opportunistic networking in internet of things scenarios: A survey,” *Access, IEEE*, vol. 3, pp. 1101–1131, 2015.

[9] L. Atzori, D. Carboni, and A. Iera, “Smart things in the social loop: Paradigms, technologies, and potentials,” *Ad Hoc Networks*, vol. 18, pp. 121–132, 2014.

[10] 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,” *Internet of Things Journal, IEEE*, vol. 1, no. 3, pp. 206–215, 2014.

[11] L. Ding, P. Shi, and B. Liu, “The clustering of internet, internet of things and social network,” in *2010 Third International Symposium on Knowledge Acquisition and Modeling*, 2010, pp. 417–420.

[12] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization,” *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, 2012.

[13] R. Girau, M. Nitti, and L. Atzori, “Implementation of an experimental platform for the social internet of things,” in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*. IEEE, 2013, pp. 500–505.

[14] M. Nitti, V. Pilloni, G. Colistra, and L. Atzori, “The virtual object as a major element of the internet of things: a survey,” *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2015.

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

[16] V. Foteinos, D. Kelaidonis, G. Poulios, P. Vlacheas, V. Stavroulaki, and P. Demestichas, "Cognitive management for the internet of things: A framework for enabling autonomous applications," *Vehicular Technology Magazine, IEEE*, vol. 8, no. 4, pp. 90–99, 2013.

[17] Q. Wu, G. Ding, Y. Xu, S. Feng, Z. Du, J. Wang, and K. Long, "Cognitive internet of things: a new paradigm beyond connection," *Internet of Things Journal, IEEE*, vol. 1, no. 2, pp. 129–143, 2014.

[18] 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 *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2. IEEE, 2013, pp. 317–320.

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

[20] L. A. Amaral, F. P. Hessel, E. A. Bezerra, J. C. Corrêa, O. B. Longhi, and T. F. Dias, "ecloudrfid–a mobile software framework architecture for pervasive rfid-based applications," *Journal of Network and Computer Applications*, vol. 34, no. 3, pp. 972–979, 2011.

[21] C. Doukas and I. Maglogiannis, "Bringing iot and cloud computing towards pervasive healthcare," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 922–926.

[22] S. Riazul Islam, D. Kwak, M. Humaun Kabir, M. Hossain, and K.-S. Kwak, "The internet of things for health care: A comprehensive survey," *Access, IEEE*, vol. 3, pp. 678–708, 2015.

[23] B. Rao, P. Saluia, N. Sharma, A. Mittal, and S. Sharma, "Cloud computing for internet of things & sensing based applications," in *Sensing Technology (ICST), 2012 Sixth International Conference on*. IEEE, 2012, pp. 374–380.

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

[25] 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 science*, vol. 50, no. 4, pp. 811–815, 2012.

[26] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Software-defined networking for rsu clouds in support of the internet of vehicles," *Internet of Things Journal, IEEE*, vol. 2, no. 2, pp. 133–144, 2015.

[27] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Efficient and scalable iot service delivery on cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 740–747.

[28] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[29] M. J. McGlynn and S. A. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*. ACM, 2001, pp. 137–145.

[30] T. Karagiannis, J.-Y. Le Boudec, and M. Vojnović, "Power law and exponential decay of intercontact times between mobile devices," *Mobile Computing, IEEE Transactions on*, vol. 9, no. 10, pp. 1377–1390, 2010.

[31] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility," *IEEE/ACM transactions on networking (TON)*, vol. 19, no. 3, pp. 630–643, 2011.

[32] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti, “Wherenext: a location predictor on trajectory pattern mining,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 637–646.

[33] S. Scellato, M. Musolesi, C. Mascolo, V. Latora, and A. T. Campbell, “Nextplace: a spatio-temporal prediction framework for pervasive systems,” in *Pervasive computing*. Springer, 2011, pp. 152–169.

[34] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, “Hardware design experiences in zebranet,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 227–238.

[35] G. Ghidini and S. K. Das, “An energy-efficient markov chain-based randomized duty cycling scheme for wireless sensor networks,” in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 67–76.

[36] T. Hao, R. Zhou, G. Xing, M. W. Mutka, and J. Chen, “Wizsync: Exploiting wi-fi infrastructure for clock synchronization in wireless sensor networks,” *Mobile Computing, IEEE Transactions on*, vol. 13, no. 6, pp. 1379–1392, 2014.

[37] C. Schurgers, V. Tsiatsis, and M. B. Srivastava, “Stem: Topology management for energy efficient sensor networks,” in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 3. IEEE, 2002, pp. 3–1099.

[38] T. Pering, V. Raghunathan, and R. Want, “Exploiting radio hierarchies for power-efficient wireless device discovery and connection setup,” in *VLSI Design, 2005. 18th International Conference on*. IEEE, 2005, pp. 774–779.

[39] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma, “Zifi: wireless lan discovery via zigbee interference signatures,” in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*. ACM, 2010, pp. 49–60.

[40] V. Paruchuri, S. Basavaraju, A. Durresi, R. Kannan, and S. S. Iyengar, “Random asynchronous wakeup protocol for sensor networks,” in *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*. IEEE, 2004, pp. 710–717.

[41] B. J. Choi and X. S. Shen, “Adaptive asynchronous sleep scheduling protocols for delay tolerant networks,” *Mobile Computing, IEEE Transactions on*, vol. 10, no. 9, pp. 1283–1296, 2011.

[42] M. Bakht, M. Trower, and R. H. Kravets, “Searchlight: won’t you be my neighbor?” in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 185–196.

[43] H. Zhou, H. Zhao, and J. Chen, “Energy saving and network connectivity tradeoff in opportunistic mobile networks,” in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 524–529.

[44] A. Chakrabarti, A. Sabharwal, and B. Aazhang, “Using predictable observer mobility for power efficient design of sensor networks,” in *Information Processing in Sensor Networks*. Springer, 2003, pp. 129–145.

[45] H. Jun, M. H. Ammar, and E. W. Zegura, “Power management in delay tolerant networks: a framework and knowledge-based mechanisms.” in *SECON*, vol. 5. Citeseer, 2005, pp. 418–429.

[46] V. Dyo and C. Mascolo, “Efficient node discovery in mobile wireless sensor networks,” in *Distributed Computing in Sensor Systems*. Springer, 2008, pp. 478–485.

[47] K. Shah, M. Di Francesco, G. Anastasi, and M. Kumar, "A framework for resource-aware data accumulation in sparse wireless sensor networks," *Computer Communications*, vol. 34, no. 17, pp. 2094–2103, 2011.

[48] X. Wu, K. N. Brown, and C. J. Sreenan, "Exploiting rush hours for energy-efficient contact probing in opportunistic data collection," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on.* IEEE, 2011, pp. 240–247.

[49] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges." *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[50] N. Lathia, V. Pejovic, K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow, "Smartphones for large-scale behavior change interventions." *IEEE Pervasive Computing*, vol. 12, no. 3, pp. 66–73, 2013.

[51] J. Goldman, K. Shilton, J. Burke, D. Estrin, M. Hansen, N. Ramanathan, S. Reddy, V. Samanta, M. Srivastava, and R. West, "Participatory sensing: A citizen-powered approach to illuminating the patterns that shape our world," *Foresight & Governance Project, White Paper*, pp. 1–15, 2009.

[52] Z. Wang, D. Zhang, X. Zhou, D. Yang, Z. Yu, and Z. Yu, "Discovering and profiling overlapping communities in location-based social networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 4, pp. 499–509, 2014.

[53] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 7, 2015.

[54] L. Ruge, B. Altakrouri, and A. Schrader, "Soundofthecity-continuous noise monitoring for a healthy city," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on.* IEEE, 2013, pp. 670–675.

[55] B. Predić, Z. Yan, J. Eberle, D. Stojanovic, and K. Aberer, "Exposuresense: Integrating daily activities with air quality using mobile participatory sensing," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on.* IEEE, 2013, pp. 303–305.

[56] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "Bikenet: A mobile sensing system for cyclist experience mapping," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 1, p. 6, 2009.

[57] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen, "Sensor discovery and configuration framework for the internet of things paradigm," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on.* IEEE, 2014, pp. 94–99.

[58] L. Skorin-Kapov, K. Pripužić, M. Marjanović, A. Antonić, and I. P. Žarko, "Energy efficient and quality-driven continuous sensor management for mobile iot applications," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on.* IEEE, 2014, pp. 397–406.

[59] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowdsensing-oriented mobile cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, pp. 148–165, 2013.

[60] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 584–616, July 2011.

[61] H. Trivedi, P. Veeraraghavan, S. Loke, A. Desai, and J. Singh, "Routing mechanisms and cross-layer design for vehicular ad hoc networks: A survey," in *Proc. of the IEEE Symposium on Computers & Informatics (ISCI)*, March 2011, pp. 243–248.

[62] P. N. Darisini and N. Kumari, "A survey of routing protocols for vanet in urban scenarios," in *Proc. of the Int. Conf. on Pattern Recognition, Informatics and Mobile Engineering (PRIME)*, Feb. 2013, pp. 464–467.

[63] Q. Zhao, Y. Zhu, C. Chen, H. Zhu, and B. Li, "When 3g meets vanet: 3g-assisted data delivery in vanets," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3575–3584, Oct. 2013.

[64] A. Benslimane, T. Taleb, and R. Sivaraj, "Dynamic clustering-based adaptive mobile gateway management in integrated vanet 3g heterogeneous wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 3, pp. 559–570, March 2011.

[65] R. Manoharan, S. Rajarajan, S. Sashtinathan, and K. Sriram, "A novel multi-hop b3g architecture for adaptive gateway management in heterogeneous wireless networks," in *Proc. of the IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WIMOB)*, Oct. 2009, pp. 447–452.

[66] A. Baiocchi and F. Cuomo, "Infotainment services based on push-mode dissemination in an integrated vanet and 3g architecture," *Journal of Communications and Networks*, vol. 15, no. 2, pp. 179–190, April 2013.

[67] P. Parwekar, "From internet of things towards cloud of things," in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*. IEEE, 2011, pp. 329–333.

[68] S. Distefano, G. Merlino, and A. Puliafito, "Enabling the cloud of things," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 858–863.

[69] M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 57–66.

[70] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[71] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social internet of things," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 5, pp. 1253–1266, May 2014.

[72] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, vol. 44, no. 9, pp. 51–58, 2011.

[73] Google. (2016, Sep.) Google search api. [Online]. Available: https://cloud.google.com/appengine/docs/java/search/

[74] M. Nitti, L. Atzori, and I. P. Cvijikj, "Friendship selection in the social internet of things: challenges and possible strategies," *Internet of Things Journal, IEEE*, vol. 2, no. 3, pp. 240–247, 2015.

[75] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases," in *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. IEEE, 2013, pp. 15–19.

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

[77] (2016, Sep.) onem2m. [Online]. Available: https://www.onem2m.org/

[78] A. Mei and J. Stefa, “Swim: A simple model to generate small mobile worlds,” in *INFOCOM 2009, IEEE*, 2009, pp. 2106 –2113.

[79] A. Carta, V. Pilloni, and L. Atzori, “Resource allocation using virtual objects in the internet of things: a qoi oriented consensus algorithm,” in *19th International ICIN Conference - Innovations in Clouds, Internet and Networks*, 2016, pp. 82–87.

[80] Y. Yun, Y. Xia, B. Behdani, and J. C. Smith, “Distributed algorithm for lifetime maximization in a delay-tolerant wireless sensor network with a mobile sink,” *Mobile Computing, IEEE Transactions on*, vol. 12, no. 10, pp. 1920–1930, 2013.

[81] M. Nitti, R. Girau, L. Atzori, A. Iera, and G. Morabito, “A subjective model for trustworthiness evaluation in the social internet of things,” in *Proc. of the IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, Sept. 2012, pp. 18–23.

[82] M. Nitti, R. Girau, and L. Atzori, “Trustworthiness management in the social internet of things,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1253–1266, May 2014.

[83] “T-TRANS.” [Online]. Available: http://www.ttransnetwork.eu/ttrans/

[84] “ICSI.” [Online]. Available: http://www.ict-icsi.eu/

[85] “U-CONNECT.” [Online]. Available: http://idi.gowex.com/uconnect/

[86] “ISO 21217:2010 Intelligent transport systems  Communications access for land mobiles (CALM) Architecture,” 2010.

[87] “ETSI EN 302 665 V1.1.1 Intelligent Transport Systems (ITS); Communications Architecture,” september 2010.

[88] S. Brown and C. Sreenan, “Updating software in wireless sensor networks: A survey,” *Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep*, 2006.

[89] “ETSI EN 302 665 V1.1.1 Intelligent Transport Systems (ITS); Communications Architecture,” september 2010.

[90] “ETSI EN 302 636-3 V1.1.2 Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture,” March 2014.

[91] “Sae j1962: Diagnostic connector equivalent to iso/dis 15031, society of automotive engineers,” https://law.resource.org/pub/us/cfr/ibr/005/sae.j1962.2002.pdf.

[92] “Parameter IDs for OBD-ii,” https://en.wikipedia.org/wiki/OBD-II_PIDs.

[93] “Telit SL869,” http://www.telit.com/gnss/sl869/.

[94] Y. Bi, X. S. Shen, and H. Zhao, “Explicit rate based transmission control in vehicle to infrastructure communications,” in *2013 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, August 2013, pp. 704–709.

[95] J. B. Kenney, “Dedicated short-range communications (dsrc) standards in the united states,” *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.

[96] “Raspberry Pi 2 Model B,” https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[97] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, “Recent development and applications of SUMO - Simulation of Urban MObility,” *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012. [Online]. Available: http://elib.dlr.de/80483/

[98] “TAPASCologne project.” [Online]. Available: http://sourceforge.net/apps/mediawiki/sumo/index.php?title=TAPASCologne

[99] W. Alasmary and W. Zhuang, “Mobility impact in {IEEE} 802.11p infrastructureless vehicular networks,” *Ad Hoc Networks*, vol. 10, no. 2, pp. 222 – 230, 2012, recent Advances in Analysis and Deployment of {IEEE} 802.11e and {IEEE} 802.11p Protocol Families. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870510000703

[100] S. Eichler, “Performance evaluation of the ieee 802.11p wave communication standard,” in *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, Sept 2007, pp. 2199–2203.

[101] E. Giordano, R. Frank, G. Pau, and M. Gerla, “Corner: A radio propagation model for vanets in urban scenarios,” *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1280–1294, July 2011.

[102] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization,” *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, 2012.

[103] J. Leskovec, “Stanford large network dataset collection.” [Online]. Available: http://snap.stanford.edu/data/

[104] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds,” in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 241–246.

[105] M. Boguna, D. Krioukov, and K. C. Claffy, “Navigability of complex networks,” *Nature Physics*, vol. 5, no. 1, pp. 74–80, 2009.

[106] A. Vegni and V. Loscri, “A survey on vehicular social networks,” *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[107] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, “A survey on vehicular cloud computing,” *Journal of Network and Computer Applications*, vol. 40, pp. 325–344, 2014.