UNIVERSITY OF CATANIA
Department of Electrical, Electronic and Computer Engineering

Doctoral Dissertation in Computer Engineering and Telecommunications
Cycle XXVIII

A NEW PARADIGM FOR COLLABORATIVE SMART OBJECTS TO ENABLE THE DEVELOPMENT OF USER-CENTRIC SERVICES IN PERVASIVE ENVIRONMENTS

Daniela Ventura
Catania, 2015

Academic Tutor:
Prof. Vincenzo Catania
Company Tutor (*Telecom Italia*):
Dr. Maura Turolla

ABSTRACT

In the near future, people will not be the only consumers of Web content, but also an increasing number of machines will be able to independently search and interpret data received from web servers in order to perform tasks for users. If a machine is an Internet-connected everyday object and its functionalities can be remotely invoked through REST API, then such machine will be part of the *Web of Things*. In addition to core features, objects will be augmented with sensing and adaptive capabilities, reasoning and decision-making abilities, and, as consequence, intelligence will be transferred to the environment.

The new properties of these spaces will change the way in which people interact with objects, as well as services, which users will access to, will become absolutely innovative. From one hand, in fact, you want to reduce or facilitate human-machine interaction. From the other hand, you want to provide context-aware services that are consistent with the context where users are located, personalized services that take into account the preferences and habits of users, and complex services that are based on the aggregation of basic services ("mashups").

There are at least three emerging factors that contribute towards this process of change involving users. First, the spread of wearable devices equipped with a multitude of sensors that provide data about users' activities or their health. This information could be used by an intelligent agent to generate customized services for users. Second, the technological progress led to the dissemination of embedded boards, cheap and easy to use also for people with not high computer expertise, and 3D printers, which generated the phenomenon of "makers". Lastly, there is the trend to provide Web APIs (typically RESTful APIs), that produced the rise of API Economy.

The respective evolutions, that machines and people, inhabitants of Smart Spaces, are going through, are closely connected: if machines become smart, the role and attitude of users change, and vice versa to improve and simplify people's lives, it is necessary to design advanced capabilities for machines. In this thesis we analyze in parallel both the aspects in the context of the Web of Things: we want to make everyday objects intelligent and cooperative in order to introduce innovative forms of interaction between users and machines, satisfy people's expectations, and increase users' eco-awareness to induce them to change their wrong behaviors that generate energy waste.

Underlying the process of collaboration among objects, there is the issue to find a machine-understandable format to describe the effects produced by invoking services exposed by a device, namely REST APIs, and a semantic language that allows to universally interpret exchanged data. Furthermore, to make machines proactive (i.e. a goal-driven attitude), it is necessary to adopt a strategy to determine all the possible "plans", in the form of communication flows involving real objects or Web services (i.e. "physical mashups"), that satisfy a

specific objective. In this thesis we propose to use standard semantic reasoners and Web technologies to overcome these problems.

Considering that pervasive environments are populated by people with different needs and abilities, this thesis presents a platform in which users express "goals" through their voice or via a web app, and Smart Objects cooperate with each other in order to execute tasks for users. The platform monitors three types of contextual data: the user's indoor and outdoor position, the elapsed time, and the state of objects. Moreover, the plan, that is selected to be executed, is personalized on the base of user's preferences and feedback.

Exploiting the method to describe REST APIs in machine-understandable format, this thesis proposes new user-object interactions. Using the *Augmented Reality* and the user-experience of mobile/web applications, we demonstrate how to overcome the heterogeneity in the interfaces to control objects.

To motivate people in to put more attention to energy consumption, in this thesis we describe a method in which everyday objects provide eco-feedback to users giving them advice about the more convenient working-mode (between on/off and standby) to set in order to save energy. These appliances are able to apply predictive algorithms to determine their next-week usage forecast and, thus, the working-mode to use per hours.

Finally, we make some considerations regarding secure communications involving, users and hardware-constrained devices (in terms of computation or available memory). Therefore, we extend the scenario to the *Internet of Things* and propose a lightweight protocol that ensures message encryption, authentication and authorization.

SOMMARIO

In un futuro non troppo lontano, le persone non saranno i soli consumatori di contenuti Web, ma anche un crescente numero di macchine saranno in grado di autonomamente cercare e interpretare i dati ricevuti da server Web, al fine di svolgere delle operazioni per conto delle persone. Se una macchina é un oggetto di vita quotidiana connesso a Internet e le sue funzionalitá possono essere invocare anche da remoto attraverso una API REST, allora tale macchina fará parte del cosiddetto *Web delle Cose*. Oltre alle funzioni di base, gli oggetti saranno aumentati con capacitá adattative e di sensing del contesto, saranno in grado di analizzare i dati e prendere decisioni, e di conseguenza verrá trasferita intelligenza all'ambiente circostante.

Le nuove proprietá di questi spazi cambieranno il modo in cui le persone interagiscono con gli oggetti, cosí come assolutamente innovativi saranno i servizi di cui gli utenti potranno usufruire. Da un lato, infatti, si vuol cercare di ridurre o facilitare l'interazione uomo-macchina. Dall'altro, si vogliono fornire servizi context-aware, cioé servizi che vengono erogati in funzione del contesto fisico o logico in cui gli utenti si trovano, servizi personalizzati che tengono conto

delle preferenze e abitudini degli utenti, e servizi complessi basati sull'aggregazione di servizi piú semplici ("mashups").

A contribuire verso questa fase di cambiamento vi sono almeno tre fattori emergenti. Per prima cosa, la nascita di numerosi dispositivi indossabili ("wearable devices") dotati di una moltitudine di sensori che forniscono dati sul movimento degli utenti o sul loro stato di salute. Queste informazioni potrebbero essere usate da un'entitá intelligente proprio per generare servizi incentrati sul profilo degli utenti. Il secondo aspetto é la diffusione di microprocessori ("embedded boards") a basso costo e di facile utilizzo anche per le persone non dotate di particolari skill tecniche, e delle stampanti 3D, che hanno determinato il fenomeno dei "makers". Infine, vi ẽ la tendenza di fornire servizi Web sotto forma di Web APIs (in genere RESTful APIs), consuetudine che ha portato alla nascita dell'API Economy.

Le rispettive evoluzioni che sia le macchine che le persone, abitanti degli Smart Spaces, stanno attraversando sono fortemente legate: se le macchine diventano smart, cambia il ruolo e l'atteggiamento degli utenti, e viceversa per migliorare e semplificare la vita delle persone, bisogna "ri-disegnare" le capacitá delle macchine. In questa tesi sono stati analizzati, in parallelo, entrambi gli aspetti nel contesto del Web of Things: noi vogliamo rendere gli oggetti di vita quotidiana intelligenti e cooperativi in modo da introdurre innovative forme di interazione tra gli utenti e le macchine, soddisfare le aspettavie delle persone e incrementare la consapevolezza degli utenti riguardo al consumo energetico inducendoli a cambiare certi atteggiamenti non ecosostenibili.

Alla base del processo di cooperazione autonoma tra oggetti c'é la necessitá di trovare un formato machine-understandable per descrivere

gli effetti prodotti dall'invocazione di un servizio esposto da un dispositivo e un linguaggio semantico che consenta d'interpretare universalmente e univocamente i dati trasmessi. Inoltre per rendere proattive le macchine (avere cioé un atteggiamento goal-driven), bisogna adottare una strategia per determinare tutti i possibili piani ("plans"), in forma di flusso comunicativo tra oggetti reali e Web services (generare cioé "physical mashups"), che soddisfano un determinato obiettivo. In questa tesi, al fine di superare le suddette problematiche, proponiamo l'uso di reasoning semantici e tecnologie Web standard.

Tenendo in considerazione che gli ambienti pervasivi sono caratterizzati dalla presenza di persone con eterogenee necessitá e abilitá, il seguente lavoro di tesi descrive inoltre una piattaforma che rivoluziona il ruolo dell'utente nei futuri Smart Spaces. Noi proponiamo che l'utente debba limitarsi a esprimere "goal" (attraverso la propria voce o tramite una web app) lasciando che sia la piattaforma a supervisionare la coordinazione degli oggetti in grado di soddisfare le richieste dell'utente. La piattaforma monitora i dati contestuali sotto tre aspetti: la posizione indoor e outdoor dell'utente, il tempo trascorso e lo stato posseduto dagli oggetti. Il piano da eseguire é selezionato considerando le preferenze e i feedback degli utenti.

Sfruttando il metodo per descrivere le API REST in un formato adatto all'interpretazione delle macchine, in questa tesi viene proposta una nuova forma di interazione uomo-macchina. Utilizzando la realtá aumentata ("augmented reality") e la user-experience delle applicazioni web, dimostriamo come superare le eterogeneitá delle interfacce di controllo degli elettrodomestici.

Per motivare le persone nel porre maggiore attenzione agli sprechi di energia, descriviamo un metodo in base al quale gli oggetti di vita

quotidiana trasmettono eco-feedback agli utenti e forniscono dei consigli sulla modalitá di lavoro (da scegliere tra on/off e standby) che conviene settare in modo da ridurre il consumo energetico. Tali elettrodomestici sono in grado di utilizzare algoritmi predittivi per determinare la migliore working-mode che l'oggetto dovrebbe assumere ad ogni ora del giorno della settimana successiva.

La tesi si conclude con alcune considerazioni riguardanti la sicurezza nelle comunicazioni che coinvolgono da un lato gli utenti e dall'altro i dispositivi con risorse limitate (per computazione o memoria disponibile). Abbiamo, per tanto, allargato lo scenario all'*Internet delle Cose* e proposto un protocollo light che garantisce crittografia dei messaggi scambiati, autenticazione e autorizzazione.

ACKNOWLEDGEMENTS

During these three years of doctoral, I met many people that have contributed to make my PhD exciting, leading me to grow up from both a human and professional point of view and bringing out the best in me.

First and foremost, I would like to express my deepest thanks to my supervisor Professor Vincenzo Catania, who gave me trust, supported my research activities, and financed all my travels abroad.

I was very fortunate to collaborate with Professor Diego López-de-Ipiña, who hosted me at Deusto University, in Bilbao (Spain) and gave me the opportunity to increase my knowledge and methodologies. I am proud to have worked with Diego C.M., Juan L.A. and Aitor G.G. during my stay in Spain. Each of them urged me to do my best and opened my mind towards new research scenarios. Thanks to all the friends known at MORElab research group that welcomed me warmly, in particular to Janire, Iván, Joaquin, Íñigo and Rizwan. Also, a special mention to Seadat, my adventure companion around Bizkaia.

My profound gratitude goes out to Ruben Verborgh who allowed

me to continue my research activity at iMinds in Ghent (Belgium), offered valuable advice, and always supported and encouraged me. Thanks to Erik Mannens and all the friends at Multimedia Lab that made me feel part of the group, helped and stimulated everyday.

Finally, I dedicate this dissertation to my family, who always believed in me, motivated, and showed me the way.

Daniela
December 2015

CONTENTS

LIST OF FIGURES

ONE

INTRODUCTION

You had to live - did live, from habit that became instinct - in the assumption that every sound you made was overheard, and, [...] every movement scrutinized.

– George Orwell, *1984*

1.1 The Vision: Ubiquitous Computing in Everyday Life

There was a time when computers were imposing devices, housed in entire rooms and needed many people to run. Then came a reduction in size; the second wave of computing, which is currently fading out, started with the introduction of personal computers, laptops, and mobile phones. It is characterized by one to one relationships between user and computers [1]. Although, we are surrounded by many computers, one device can be used per time only by a user in order to

resolve tasks required by that user. The next wave involves computers so small they hardly seem like computers at all - liberated from the desktop and pervading every facet of our lives [2]. Sensors, microchips, transmitters, actuators, will be everywhere, both inside and outside buildings. Everyday objects and household appliances' capabilities will be augmented: these devices will be able to "talk" to each other and to make decisions. The computation functionality will be distributed over objects and users will interact with many devices at the same time. Therefore, the third ware of computing will be the realization of paradigm which goes under the name of *Internet of Things* (IoT).

The concept of pervasive technology was introduced in the early 90s by the pioneer Marc Weiser, a chief scientist at Xerox PARC in Palo Alto. He wondered about the role that digital technologies could gain in our daily lives and imagined scenarios in which devices are easy to use or so tiny that become almost invisible. He coined the notion of ubiquitous computing, now evolved in the IoT, describing the phenomenon in these terms:

> "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.[...] Machines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods* [3]".

When Weiser exposed his ideas, the use of a computer still appeared as a luxury reserved for a few people and the Web was making inroads in the world. Thus, the scientific community of that period

considered the Weiser's vision totally inconsistent, like a science fiction. Nevertheless, in recent years, thanks to the advances in the miniaturization of integrated circuits, in the low power wireless communication (as the Bluetooth Low Energy), in the identification methods in short range (as the invention of RFID), pervasive computing is becoming reality.

The modern view of the Internet of Things is based on the concept of interweaving the virtual world and the physical world. Therefore, each real object has to have an URI (Universal Resource Identifier) as way to be uniquely identified, and provide services, remotely usable, in order to set its state or to get its values/data. The IoT is composed by three main research fields which are differentiated by the type of entities involved in the interactions and amount of hardware resources possessed by objects:

- Wireless Sensor Networks (WSNs) leverage low-power radios and multi-hop communication to cover large areas with small and inexpensive sensor nodes. They are characterised by constrained resources (like memory, power) and easy functionalities (usually they are only able to read the values monitored in the physical environment). They have a wide range of potential applications such as industry, transportation, civil infrastructure, and surveillance.

- Machine to Machine (M2M) consists of data exchange between two or more devices in order to execute tasks not explicitly required by a human being. Contrary to the WSNs, the M2M interactions involve more complex objects, equipped with a inner logic and able to perform both low and high level operations.

A typical scenario is the communication between car fleets and a central IT system.

- Human-Computer Interaction (HCI) studies new and more natural forms of interaction between users and their everyday objects endowed with processing and communication capabilities to provide digital services. Many use cases, set in Smart Spaces, concern the topic of HCI.

This dissertation examines some research aspects of all these field.

1.1.1 The Metamorphosis of Spaces

Augmenting everyday objects with sensing, computation, and communication capabilities has as consequence to make smart the environments where users live. The concept of *Smart Space* is used to indicate a physical place, public (like schools, hospitals, airports) or private (like houses, offices), where people and technologies cohabit and continuously exchange information with the purpose to satisfy people's needs and requests in an intelligent and appropriate way.

In [4], D. Cook and S. Das have formally defined the Smart Spaces in the following manner: "*a Smart Space is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment*". The fundamental idea behind this definition is that the environment has to learn about their occupants, have the ability to identify specific characteristics of inhabitants, and help them to carry out their daily activities thanks to the use of the surrounding embedded devices. Therefore, a Smart Space, in addition to have *sensing functionality* to gather information

about the context, has *adaptation functionality* to detect changing situations. Lastly, it uses the *effecting functionality* in order to alter the conditions of the environment according to users' needs [5].

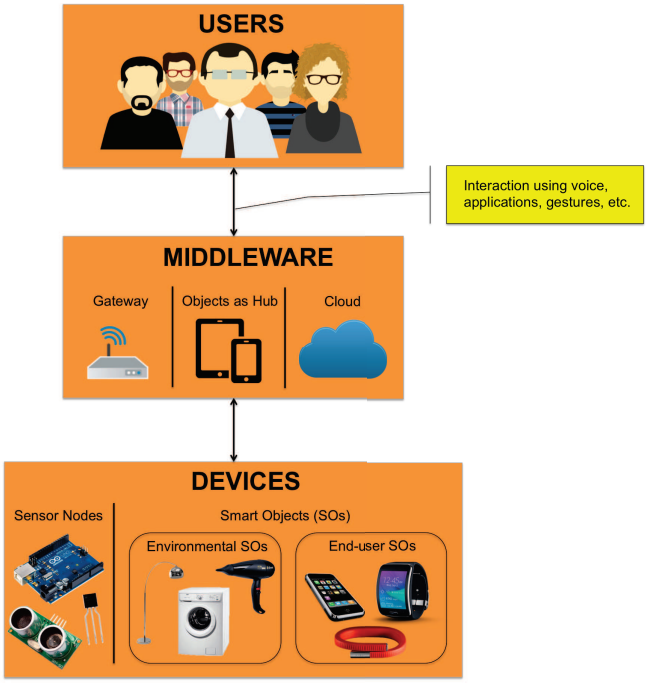


Figure 1.1: *The main entities involved in a Smart Space: users, middleware and devices. The way which users interact with the devices passes through the middleware. Usually the middleware elaborates users' requests, monitors the environment and acts on it.*

The three basic entities of a Smart Space are devices, middleware, and users, as depicted in Figure 1.1. The **devices** are physical components what allow the intelligent agent (i.e. the middleware) to sense and act upon the environment. We distinguish sensor nodes from

Smart Objects for the level of complexity of functionalities provided by devices. After all a sensor temperature, for example, is able only to detect and return the current temperature value without applying any control logic, while an air conditioner not only knows the real-time temperature but also allows users to set its working mode (e.g. fan speed, desired temperature, time interval to reach the desired temperature). Therefore, sensors nodes can be sensors, used to perceive basic data about the environment, and actuators, used to execute simple actions. Usually different sensor nodes are physically connected to one microcontroller, i.e. an embedded board, responsible for acquiring and aggregating sensors' data.

About the Smart Objects, a formal definition of their properties will be described in the next subsection. However, we have separated two categories of Smart Objects: end-user Smart Objects and environmental Smart Objects. Both the sets of Smart Objects can be used by the middleware to perform operations in the Smart Space but the former is also used by the users in order to interact with the environment. End-user Smart Objects are computers, smartphones, smartwatches, wristbands and other wearable devices. These devices allow to identify users and usually know person's physiological state or movements. Environmental Smart Objects are appliances, video projectors, TVs, lamps, heaters, and so on.

The **middleware** is responsible of: 1) merging the flow of data coming from the devices through wired and/or wireless networks; 2) processing and reasoning on the received information in order to extract new knowledge and build a representation of the current state of the environment; 3) selecting the action to execute which causes a change in the state of the environment. Even if the middleware usu-

ally resides entirely in the *Cloud*, a portion of its could be run or in a gateway physically placed in the Smart Space or in an end-user Smart Object (usually the smartphone).

Finally, **users** can specify a desirable state of their Smart Environments using new kinds of interaction like voice, gestures, applications, and others. More details about the evolution in the human-computer interaction and in the user's role will be deepened in Section 1.3. We emphasize, however, that within a Smart Space it is important to find a compromise between acting when the user expects to be helped and refraining from interfering in people's lives when the intervention of the middleware is not desired or would not be appreciated.

1.1.2 Smart Objects: Requirements

Smart Objects (SOs) are important components of the Internet of Things. Over the years, many researchers have discussed on which features a Smart Object should have and which operations it should be able to run.

In [6] a Smart Object "*is an everyday artefact augmented with computing and communication, enabling it to establish and exchange information about itself with other digital artefacts and/or computer applications*". This definition highlights the first requirement of the SOs: the ability to transmit and receive data. This suggests that a Smart Object has to have a communication system and a unique identifier. However, the authors describe the entities involved in the information exchange only as other objects and not also as people. This definition, thus, lacks of the concept of the interaction with humans, absolutely necessary in an Smart Space.

A similar description is presented in [7], where Kortuem et al. state that the SOs are "*objects of our everyday life, augmented with information technologies and equipped with sensing, computation and communication capabilities, that are able to perceive and interact with their environment and other smart objects*". More focus is aimed at the ability of the SOs to capture the context, but the active role of the SOs' and their ability to take autonomous decisions or find out new knowledge through the analysis of the data are not considered at all.

These aspects are explained in [8] with the words: "*smart objects might be able not only to communicate with people and other "smart" objects, but also to discover where they are, which other objects are in their vicinity, and what has happened to them in the past*". Therefore, a form of intelligence (not necessarily sophisticated) is an additional quality that the everyday Smart Objects should have.

From our point of view, a Smart Object has to have the following proprieties:

- Unique identifier: according to the Internet of Things, each object has a virtual counterpart, so it is essential to uniquely identify a Smart Object in the digital world (using the concept of URI).

- Self-awareness: a Smart Object has to be able to describe what its services do, how to invoke them, what states it can assume. These descriptions should be expressed in a machine-understandable format.

- Sensing and/or actuating capabilities: a Smart Object has to interact with the physical world, by sensing what happens in

the environment and by acting accordingly to it.

- Communication systems: They give to the Smart Objects the ability to send and receive queries or commands or the descriptions of services, and interact with humans or other objects.

- Computational and/or reasoning abilities: the SOs have to be able to elaborate the contextual information and discover new knowledge in order to make decisions or autonomously to execute actions.

- Memory: the Smart Objects should have persistent memories in order to store the state of their resources.

Developing SOs raises many issues mostly regarding the heterogeneity in the communication protocols among SOs, the general-purpose design, the distributed computation, security and autonomous interaction with both users and other objects.

1.2 The Evolution of the Web: Towards the Web of Things

While the Internet is becoming the platform for interconnecting everyday physical "*things*" at the Network layer, the choice of a universal platform at the Application level is the Web. A thing becomes Web-enabled when it is augmented with a Web server so that it can expose its functional and non-functional capabilities on the Web through HTTP [9]. Therefore, we are witnessing to the evolution of the Web towards the so-call *Web of Things* (WoT). Since the different phases

of evolution are not silos but tied together, the Web of Things rests on the previous periods. The evolutionary stages of the Web are represented in Figure 1.2.

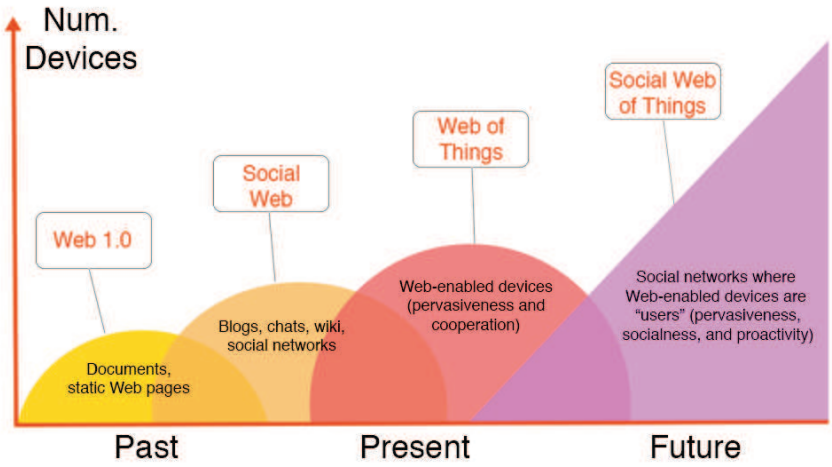


Figure 1.2: *The evolution of the Web.*

In its early years the Web consisted of documents linked together through hyperlinks and hypertext, which allow an immediate connection to other pages or texts. The Web was a sort of worldwide newspaper where people, using search engines, could search and find static web pages and web content.

The rise of CMS, chats, wiki and forums revolutionised the Web and the user's role: from passive consumer of information to creator and aggregator of contents that are made available to other users. Anyway, the leading innovation in the Web 2.0 (also called *Web of People* or *Social Web*) has been the advent of social network sites (such as Facebook, Google Plus, Twitter): evolution of some forms

of social interaction that the Web has always supported (computer conferencing, email, mailing lists, and so on). Although from the conceptual point of view social networks do not constitute a new idea (like blogs, the key issue is sharing content), they introduce some innovative aspects, in particular the concepts of "*profile*" and "*followers*", that will be associated also to the Smart Objects, as described in Section 1.2.1. The profile represents the virtual identity of a person and includes all his preferences, what he likes, where he is located and part of his history (for example places where he has been or lived). Not in all social networks is explicitly expressed the concept of followers, but, however, users can decide who to follow in base of the principles of friendship (Facebook), of job skills (LinkedIn), of interests (Twitter) and so on.

The next stage of the Web will be the Web of Things, characterized by two aspects:

- **Pervasiveness**: the abstraction of physical things as services on the Web. The consequence is that everyday objects become connected, accessible and searchable through the Web.

- **Cooperation**: two or more objects can exchange data in order to progress in the execution of a task. It will be possible to create the so-called "physical mashups", i.e. the composition of Web services and services exposed from physical objects. The realization of physical mashups will be mostly driven by users through the use of specific tools, such as the dashboards.

We think that the fulfilment of the paradigm of the WoT is influenced by the convergence of three elements: the conversion of Web into the Semantic Web, the API revolution, and the spread of the connectivity.

Along with the spread of social networks, the *Semantic Web* made its appearance. At the beginning the Semantic Web technologies were used only to obtain more relevant results provided by search engines. Intelligent softwares interpreted the meaning of web documents and generated metadata described in machine-understandable format (through languages like RDF, OWL). The new challenge is to cover the entire Web with the Semantic Web. The research on Linked Open Data and the definition of ontologies (like schema.org) to generate a common data model represents a fundamental contribution toward this transition. Furthermore, the active development of Semantic Web technologies is transforming the Web into a medium suitable for highly dynamic Machine to Machine interaction and thus contributing to the realization of the Web of Things.

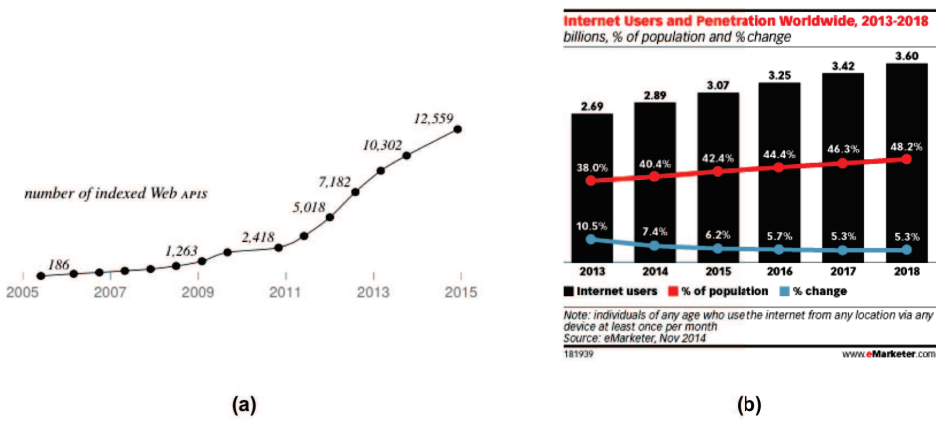


Figure 1.3: *Factors that contribute to the diffusion of the Web: (a) the increasing number of Web APIs (source: programmableweb.com) and (b) the spread of connectivity around the world (source: eMarketer).*

The second key factor that leads towards the WoT is the spread of APIs (Application Program Interfaces), as shown in Figure 1.3 case (a), that has produced the API Economy, examined in Section 1.2.2. Currently APIs are used to allow access to Web services but, at the same time, they are suitable to be used as access point by the physical objects. In other words, APIs represent the interface between the Web and the things. If APIs are documented through a machine-interpretable language, they can be discoverable by autonomous agents and combined to solve workflows.

Lately many financial resources are being spent to bring connectivity everywhere in the world and to upgrade telecommunications networks to 4G LTE and 5G. This fact is also important because the spread of connectivity is an other element that will influence the realization of Web of Things. Currently, even if 3 billion people have already access to the Internet, as illustrate in Figure 1.3 case (b), there are about 4.2 billion users unconnected. To get more people connected, Facebook and Google have both begun efforts to increase access [10]. To deliver Internet access to users below, the former spent $20 million to build drones capable of flying for long period of time, while the latter has been working on a project using giant balloons. More users will be connected to the Web, more and more objects will be connected to the Web, and therefore the WoT will be a disruptive phenomenon.

1.2.1 From the Web of Things to the Social Web of Things

If the Web 2.0 has enabled users to share contents and has generated a huge graph of relationships among people, after that everyday objects will be part of the Web, they will be also integrated in the social graph. As shown in Figure 1.2, the convergence of the Web of People and the Web of Things will produce the *Social Web of Things* (SWoT). The core idea is to extend the concept of social network in a platform that enables users to manage the access to their Web-enabled devices and share them with people they know and trust. Therefore, the first property of the Social Web of Things is the **socialness**.

The two concepts of profile and followers, revolutionary for the Web of People, will be inherited and applied to the objects. The profile of an object represents its counterpart on the Web, as well as the profile of a person is his virtual identity. The profile of an object will contain information such as its current status, the operations it can perform (and thus the services that it exposes) and its current location. In addition, each object could have some followers: users or other objects that can check at any time the status of the object and control it.

From a user experience perspective the use of dashboards, to view the current status of objects, to invoke a service on objects or to create the physical mashups, is troublesome and difficult to manage with a considerable number of objects (more than 10 or 15). *Ericsson User Experience Lab*, in [11], has proposed to introduce the SWoT as an alternative for a simplified interaction model where users interact with things and vice versa using common actions in social networks,

such as posts, comments, and so on.

However, the Social Web of Things does not have to be thought only as a social network of people and things for enabling access control to Web-enabled devices. The second property of the Social Web of Things is the **proactivity**, as suggested by Ciortea et al. in [12]. In accordance with our definition of Smart Objects (given in Section 1.1.2), Web-enabled devices have to become entities able to autonomously undertake actions using their computational and reasoning abilities. The Social Web of Things will allow objects to make decisions and perform tasks not only using data about the context produced by their sensing capabilities, but also through the huge amount of information shared on social networks and Web resources (namely Web services and others Web-enabled devices). In other words, Web-enabled objects have to exhibit goal-driven behavior in order to meet the requirement of proactivity. For instance, in the *Social Internet of Things* [13] the objects have as goal that to autonomously manage their relationships with one another by following some rules (e.g. objects from the same manufacturer or objects belonging to the same user).

1.2.2 The Rise of the API Economy

There are trillions of transactions happening on the Web everyday, and most of them are going through Application Program Interfaces (APIs). At the base of this phenomenon there is the fact that, in recent years, more and more business companies have adopted the strategy to release APIs to enable third-party developers to create applications and services based on well-established platforms. Some examples are Google, Facebook, Twitter, which allow developers to have access to

their data (maps, user postings, tweets) that are used to launch new applications in the maket. From developers' point of view, to rely on already existing and well-established services is a way of abstracting from issues, such as server management, scalability, data backup, etc. At the same time, business leaders are aware of the financial impact that APIs can have, and companies expose APIs in order to generate revenue. For example, *Salesforce.com* gains more than half of its $2.3 billion in revenue from its APIs, not from its user interfaces [14].

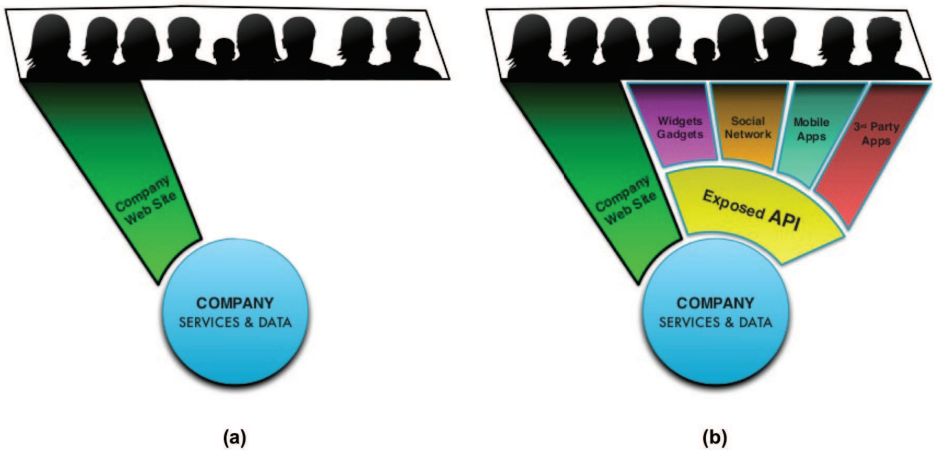


Figure 1.4: *Size of marker reached by a company using only its Web site (a) or also other channels (b) through supply of Web API.*

The API revolution represents a new opportunity for the business-to-business (B2B). In fact, while in the past, a company's market reach was limited to its direct sales organization, distributor channels and its website for online commerce, now virtually every developer is a target for its APIs and every application they create is a channel to reach new customers [15].

As shown in Figure 1.4, for a company that wants to make profit from its data, a Web site would cover only a small portion of the online population. In contrast, leveraging on the APIs, it captures a wide spectrum of customers across channels to be added to the website. These channels consist of third-party applications, social networks, widgets, mobile applications and so on.

1.2.3 Why using the Web to interconnect Things?

The ubiquitous computing environments are characterized by a high degree of heterogeneity in terms of communication protocols (e.g. ZigBee, Bluetooth, BLE), hardware capabilities of the objects and data format. Different IoT architectures have been proposed and implemented in order to address these issues but their main limit is that their usage is restricted only to some application domains. Nevertheless creating Smart Environments populated by interconnecting everyday objects requires to tear down vertical systems and enable seamless interoperability.

The Web is the de facto Application layer of the Internet, as previously said, and ensures interoperability through the use of unique addressing schemes and standard communication protocols. In the Web of Things, a Smart Object is a web server that implements Web services and the REST architectural style is applied to define the resources in the physical world. The main advantage of REST is to focus on creating loosely-coupled services and promoting the easy reuse and composition of Web services. REST uses URLs for identifying resources on the Web and a uniform interface (HTTP's methods) for representing services. As consequence, it is highly versatile, flexible

and can be used in different application domains.

The second key factor in favor of the Web of Things is the popularity of the Web and the Web technologies. Browsing the Web has become part of our everyday lives and concepts as “caching”, “links”, “bookmarking” are known also by not tech-savvy people. At the same time, each person can employ any computing device to manage his Smart Objects (e.g. mobile phones, tablets, PCs), because these devices have access to the Web. The use of the Web to interact with objects not require users neither to learn new concepts nor to buy external devices (like adaptors for communication protocols). In addition, the Web can rely on a large community of developers that can use the Web technologies (e.g., HTML, JavaScript, Ajax, PHP, Ruby) to build applications involving Smart Objects and to bring them quickly to the market.

Date transmitted/received from a web server are structured data and, then, directly machine-readable. Typically they are XML documents or JavaScript Object Notation (JSON) objects. If these formats would be supplemented with semantic information, they could be understood by machines in fully autonomous way. This issue will be addressed in depth in the rest of this thesis.

Finally, the specification of the *Constrained Application Protocol* (CoAP) represents a solution for the seamless integration of highly resource-constrained devices in to the World Wide Web. The low overhead of CoAP also improves performance in unconstrained devices.

1.3 The impact of the Web of Things on Users

The way people interact with devices depends by the technology available. Current computer technology interaction is explicit - the user tells the computer what he expects the computer to do, using mouse, keyboards, command-line, GUI, etc. The vision of the Web of Things, however, are leading to a metamorphosis of environments. Sensors places everywhere and multiple devices with different capabilities will have to coexist with users, their different needs and expectations. Such devices, i.e. the Smart Objects described in Section 1.1.2, will be able to elaborate data about the context and to execute complex tasks.

While environments are evolving, thanks to the technology progress and the "augmented" capacities of everyday things, the communication from users to the environment (i.e. input) tends to shift from explicit towards implicit. Using the definition written in [16], for implicit human-computer interaction we means "*an action performed by the user that is not primarily aimed to interact with a computerised system but which such a system understands as input*". Therefore, the concept of implicit interaction is based on the assumption that devices have perceptional capabilities, to gather information in the space, and interpretation ability, to understand the context and to act according to it. Taking into account that Smart Spaces are populated by different people with different abilities (usually young people, unlike the elderly, have at least the basic computer skills to voluntarily seek direct interaction with the system), deciding the level of implicit interaction that a Smart Space should have is a complex problem.

From one hand, in fact, Ben Shneiderman in [17] points us: "*users want to have the feeling they did the job - not some magical agent*". In other words, the system must allow users to explicitly set their preferences, force it to perform a task, choose the objects to be used to carry out that task, check or stop its execution. On the other hand, people "*want to be couch-potatoes and wait for an agent to suggest a movie to them to look at, rather than using 4,000 sliders, or however many it is, to come up with a movie that they may want to see*", as said by Pattie Maes in [17]. This entails a will to have Smart Spaces that are supposed to know users, infer situations, provide recommendations in fully autonomous and implicit manner. For this reason, it is necessary to find a trade-off between a system that implicitly "feels" the context and understands how to act, and a system that is explicitly controlled and managed by users.

About the communication from the environment to users (i.e. output), it is "distributed" and available in many modalities (e.g., auditory, speech, visual), locations (e.g., monitors, TVs, projected on walls) and forms (e.g., on wearable devices or through the actuation of some ambient lights) [18]. Hence, the environment itself becomes the user interface.

1.3.1 New kinds of interaction in the Internet of Things

In agreement with the vision of Weiser, who believed that technology has to become indistinguishable from everyday life, more natural forms of interaction between users and objects are being introduced. Although the smartphone is currently the best tool communicate with

objects, the following new forms of interaction could be used in the future:

- Gestures recognition: initially it has been applied in the world of gaming (Wii, Kinect), but it has evolved until to include also the world of pervasive computing. The spread of wearable devices has fostered the production of gadgets that allows users to control their objects using gestures. Myo [1] is, for example, a band that has to be wear around the forearm that give the "power" to control various applications (like games and presentation software) with basic arm and hand gestures. It uses electromyography to measure the electricity that runs through the muscles while a users is moving. It then translates the gestures into commands. An alternative to wearable devices is to use cameras in order to recognize gestures. For example, the Leap Motion [2] is a small object, with 2 cameras and 3 infrared LEDs, designed to recognize the fingers' movement. As Micheal Buckwald, CEO of the Leap Motion, told: "*the goal (of the Leap Motion) is to fundamentally transform how people interact with computers and to do so in the same way that the mouse did*". Noteworthy is also the research project WiSee [3] that leverages ongoing wireless transmissions in the environment (e.g., WiFi) to enable whole-home sensing and recognition of human gestures.

- Vocal interaction: it is a topic that involves much current research. Some of the most important implementations are Ap-

[1] `https://www.myo.com/`
[2] `https://www.leapmotion.com/`
[3] `http://wisee.cs.washington.edu/`

ple's Siri and Google Now. Many applications that use speech recognition are implemented to help elderly or disabled people to perform everyday activities. Actuate objects via voice recognition requires the use of a device equipped with a microphone. Consequently the smartphone seems to be the tool that best suits to translate voice in commands for the object.

- Wearable objects: besides the use for gestures recognition, wearable devices enable other forms of interaction with objects. The most famous example of wearable device is the Google Glass. It exploits the concept of Augmented Reality to provide many functionalities: browsing on websites, use of social networks, visualisation of maps, taking pictures and so on. Moreover, the smartwatches revolutionize the way people interact with the environment.

- Control with brain: In the future we will control objects using our brain. A simple prototype [4] that enables the control with brain by parsing data from Neurosky-based EEG headsets was implemented using an Arduino board.

1.4 Research Focus

To realize the paradigm of Smart Spaces, you have to not only examine the objects' perspective, i.e. to study what requirements they should have and how to increase their abilities, but also explore users' role in these contexts. Therefore, we have gazed at the improvement of

[4] `https://github.com/kitschpatrol/Brain`

features of objects in conjunction with users. The hypothesis of this thesis is:

> *Everyday devices have to become smart, i.e. proactive and collaborative, in order to have impact on users under three points of view: 1) changing people's inappropriate behaviors; 2) satisfying people's needs; 3) simplifying user-object interactions.*

As layer to enable communication between objects and people, we have used the Web, that allows not to consider heterogeneity in terms of communication protocols, contains huge quantity of services and data, and also is a well-known "tool" to people. This means that in this dissertation objects are always Internet-connected devices and provide (hypermedia) RESTful APIs. This thesis aims to answer to the research questions below.

Research question 1: *How can an everyday object invoke REST services without pre-knowledge about how using them and what the produced effect on invoking them is?*

An everyday object has to be enabled to use transparently both the traditional Web services and the services exposed by other physical objects. The main difference between the two categories is that the services exposed by objects produce actions that modify the state of the objects-self and/or the context of an environment, such as turning on a lamp means to generate light in the environment, or switching on a TV means to produce sound and light (as well as showing video and audio). Therefore, objects have to acquire understanding

capabilities, in particular: 1) what a web server expects to receive as input and returns as output; 2) what behaviour a web server (object) adopts and what consequence are reflected in the environment when its REST services are invoked.
In order to find a syntactic and semantic technique that allows to describe REST APIs in machine-understandable format, we have scouted existing solutions and chosen the one more suited to solve the problem.

Research question 2: *How can an everyday object combine REST services in autonomous way in order to achieve a goal?*

Making everyday objects collaborative requires not only to enable machines to understand each other, but also they must have reasoning and proactive features. In other words, the machines must be able to generate physical mashups of REST APIs, in the form of communication flows among objects and/or Web services, in order to perform tasks without explicit human intervention. It is interesting to note that this research question is strictly dependent from the previous one.

Research question 3: *Whereas environments where users are present in are populated by objects strongly heterogeneous and often difficult to use by people, how can the capabilities acquired from everyday objects, regarding the provision of machine-understandable descriptions of their services, be used to keep the same user-experience consistent across devices?*

The current most used method to overcome heterogeneity of everyday objects in terms of interfaces and management is to implement dashboards. Their main limit, as previously said, is that the visualization and the control of objects' state become difficult with the progressive increment of devices of the same type (e.g. how to distinguish many lamps placed in the same environment using dashboards?). Therefore, we have investigated new techniques to improve user-experience during the interaction with heterogeneous objects, assuming to have found a response to the research question 1.

Research question 4: *How to meet users' needs and expectations in a non-intrusive and adaptive way exploiting the features of everyday objects to communicate and cooperate with each other?*

Designing pervasive environments requires to interpret users' (explicit or implicit) requests and apply strategies to solve tasks. Interactions between users and objects should be as natural as possible, as well as reduced using an intelligent agent able to orchestrate cooperation among objects (reached in the research question 2) that satisfies users' needs. However, users should not perceive the system as oppressive, an entity whose objective is to reduce their autonomy and control over the environment.

Research question 5: *What persuasive methodologies can everyday objects adopt in order to increase users' awareness about energy consumption and bring them to assume more eco-sustainable behaviours?*

The reduction of energy consumption can be carried out by making both objects and people smart. The former should be able to analyze contextual data and implement appropriate energy-saving policies. The latter could become more wise about energy saving by receiving eco-feedback in the right moment.

Research question 6: *If users/objects interact with resource-constrained devices (not using the Web), how to ensure secure communications in terms of data encryption, authentication and authorization?*

Nowadays tiny and cheap devices can transmit data to users' devices. They have not capability to use complex cryptography algorithms. Therefore, it is necessary to study lightweight solutions to ensure secure communications. This research question is the only that concerns the IoT context, respect to the other ones that are about the WoT.

1.5 Outline

This thesis is structured as follow: **Chapter 2** describes the current state of the art with regard to Web architecture, Web services, and Semantic Web. **Chapter 3** contains our proposed about how to describe functionalities of the Smart Objects in a machine-understandable manner and what format using to exchange data. In order to evaluate this approach we have presented a use case where a

smart client machine generates and executes plans in order to satisfy a goals in a fully autonomous manner. **Chapter 4** faces the problem of user-object interaction and provides a proposal which exploits the webinos platform (implemented during an European Project) and the machine-understandable descriptions described in the previous Chapter. **Chapter 5** showcases a M2M scenario where objects in a smart space work together to achieve a goal which has been expressed by the user. The plan production and execution uses the mechanism introduced in **Chapter 3**. In **Chapter 6** we describe an approach to increase users' awareness about energy consumption making everyday things smart. As proof of concept, a commercial machine has been hacked and a machine learning algorithm has been implemented in order to find the best working mode day by day. Finally in **Chapter 7** we provides some security considerations for resource constrained devices.

TWO

BASIC CONCEPTS AND TECHNOLOGIES

2.1 The Basis of the World Wide Web

The World Wide Web has now become our primary information repository - a vast distributed library of interlinked hypertext documents, software, images, and so on, covering a multitude of subjects and application areas. Each of these data can be called as Web resource. The concept of *resource* is a key concept of the Web (and in particular of the REST architecture style that we will describe in Section 2.1.1) and has evolved during the Web history. However it was explicit defined, for the first time, in the RFC 2396 [19] with these terms:

> "*A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.*

> *The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content - the entities to which it currently corresponds - changes over time, provided that the conceptual mapping is not changed in the process*".

Analyzing this quote, it is clear that a Web resource is always different from the others and, thus, it is a necessary a mechanism of unique identification. A Web resource is something we can use. It contains information that can be obtained or modified over time, i.e. it has a state that can evolve. To meet these requirements at the base of the concept of Web resources, the Web is made up of three core components, each of which is discussed below.

Uniform Resource Locator (URL): A URL is the univocal address that specifies the location of a resource on the Web. It must specify different information, as shown in Figure 2.1: 1) the protocol used to access the resource; 2) the name of the host computer on which it is located (i.e. the domain name); and 3) the path to identify the resource on the server.

Hypertext Transfer Protocol (HTTP): HTTP is the standardized application protocol used by client and server to communicate and exchange data on the Web. It runs on top of the TCP/IP suite of protocols. This protocol uses a simple request/response message paradigm. Therefore, a client sends a message to request the representation of a Web resource and the server returns the message in which the representation is enclosed. The most known client is the

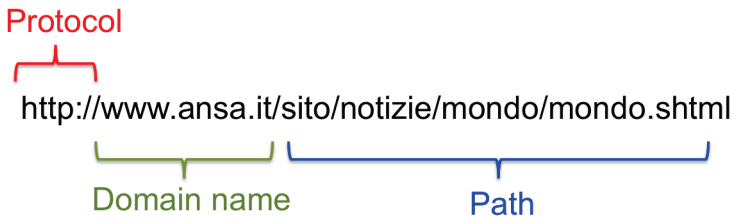


Figure 2.1: *Componenets of a URL: protocol, domain name and path of a Web resource.*

browser. Each HTTP message consists of three parts: a request line (where there is the method name of the requested action and the URL to the resource that is the subject of this action), a header (containing some metadata like Host, User-Agent, etc.) and the body of the message.

Hypertext Markup Language (HTML): HTML is a markup language (one of the most common) to describe Web documents (that are Web resources). Furthermore, HTML documents can contain *links* to other resources, which are identified by their URLs. Links and the resulting networking effects played a fundamental role for the success of the Web.

2.1.1 The Representational State Transfer Architectural Style

When the new century was just around the corner, Roy T. Fielding published his doctoral thesis [20] "*Architectural Styles and the Design of Network-based Software Architectures*" in which analyzed the basic principles of different software architectures (ignoring the implementa-

tions) and described an architectural style that he called *Representational State Transfer* (REST). REST represents a set "guidelines" and constraints that have been used to guide the design and development of the architecture of the modern Web.

The first constraint of REST architecture is a **client-server** communication: the server offers a number of services which a client can invoke by sending requests to the server. The motivation behind the use of a client-server architecture is that allows the components to evolve independently and improves the scalability. In particular, REST adds the constraint that the client-server interaction must be **stateless**, that means each message, exchanged during the communication, must contain all necessary information to elaborate the request/response, and must be understood without any relationship with previously sent messages. The server does not keep the state of client (also said "application state") but, of course, store the state of its resources.

The stateless constraint generates overhead in the interaction phase between client and server. Therefore, in order to reduce the overhead, REST architecture promotes the use of caching. **Cache** constraint requires that the data within responses be implicitly or explicitly labeled as cacheable or non-cacheable. The positive aspects of using the caching mechanism are: 1) to lower the number of exchanged messages, 2) a more efficiency and scalability of RESTful systems, and 3) improving the user-perceived performance by reducing latency. The negative consequence could be the reduction of the reliability of the system, i.e. state data could be used by client.

REST defines also a group of constraints about **uniform interface**. They are:

- **Identification of resources**: as said in Section 2.1, it is necessary that a resource is identifiable so that it can be accessed and manipulated via generic interfaces.

- **Manipulation of resources through representations**: during the communication process, a client receives not directly the resource but a representation of its. A representation is the state of a resource, i.e. a sequence of bytes plus some metadata. It is represented using a fixed media types know by both client and server.

- **Self-descriptive messages**: messages exchanged by client and server should be processed without out-of-band knowledge.

- **Hypermedia as the engine of application state** (HATEOAS): It refers to the use of hyperlinks in resource representations as a way of navigating the state machine of an application. This is one of the constrain more violated.

Finally, the last two REST constraints are **layered system** and **code-on-demand**. The former emphasis the fact that a system has to be composed by hierarchical layers in which components on a specific layer only provide services to components on the layer above and only use services provided by components on the layer below. The latter allows client functionality to be extended by code that is loaded dynamically at runtime. Both of them produce more adaptable and expandable systems. The negative aspects are: 1) adding additional layers generates overhead caused by processing operations; 2) loading code may produce security problems and vulnerabilities.

2.1.2 Services on the Web: RESTful and SOAP Comparison

A Web service is a software system designed to support request/response mechanism that allows a client-application to remotely access/-modify data exposed by a Web server using standard Web technologies (over HTTP). Furthermore, Web services are a key component in "mashups" (whose a state of art will be described in Section 3.2.2). A client could use data from different Web servers in order to generate new content or aggregated Web services.

At present there are two approaches to the creation of Web services: SOAP (*Simple Object Access Protocol*) and services based on REST. Although the goal of both approaches is almost the same, namely the adoption of the Web as a computing platform, their vision is totally different. Even if REST is currently the more popular (and easier) method to create Web services, the use of one rather than the other probably depends on the specific case.

First, REST focuses the attention on Web resources that can be handled using standard CRUD operations (GET, POST, PUT, DELETE) of the HTTP protocol. Contrariwise, SOAP focuses on exposing pieces of application logic and operations as services. Therefore, we can state that REST services are "genuine" Web services because based on the founding technologies of Web (like URLs and HTTP) while SOAP tries to introduce in the Web the concept of remote procedure call.

Second, SOAP-based services allow to describe the interfaces with WSDL [21] and XML Schema ([22]-[23]) documents. The documentation is, as consequence, machine-readable and automatic code gener-

ation on both the client and the server sides are made possible. The other side of the coin is the increment on coupling, lack of flexibility (SOAP only permits XML as data exchange format) and high-difficulty on using WSDL format that discourages developers. In contrast, the REST architectural style does not force the use of a specific data format (it is possible to user JSON, XML, and other languages), has better performance and scalability, and REST messages can be cached. The bad news is the lack of a (universal) formalism to describe REST-based services that allows human documentation to be both automatically transformed into code and interpreted at runtime. Anyway an overview on related work about machine-understandable formats for REST Web services will be presented in Section 3.2.1.

The last aspect to consider is security and reliability on communications. Unlike SOAP, REST supports only SSL mechanism and expects clients to deal with communication failures by retrying. SOAP, in fact, is able to manage interactions with different enterprise security features (WS-Security), has successful/retry logic built-in (WS-ReliableMessaging), and supports ACID Transactions over a service (WS-AtomicTransaction).

2.2 Semantic Web and Linked Data: A Case Study

The BBC is the largest broadcasting corporation in the world. Until 2010, the BBC had different Web sites that dealt with different topics (e.g. news, food, gardening, music, etc.) because different inner teams had the task to update all the news related to the specific topic [24].

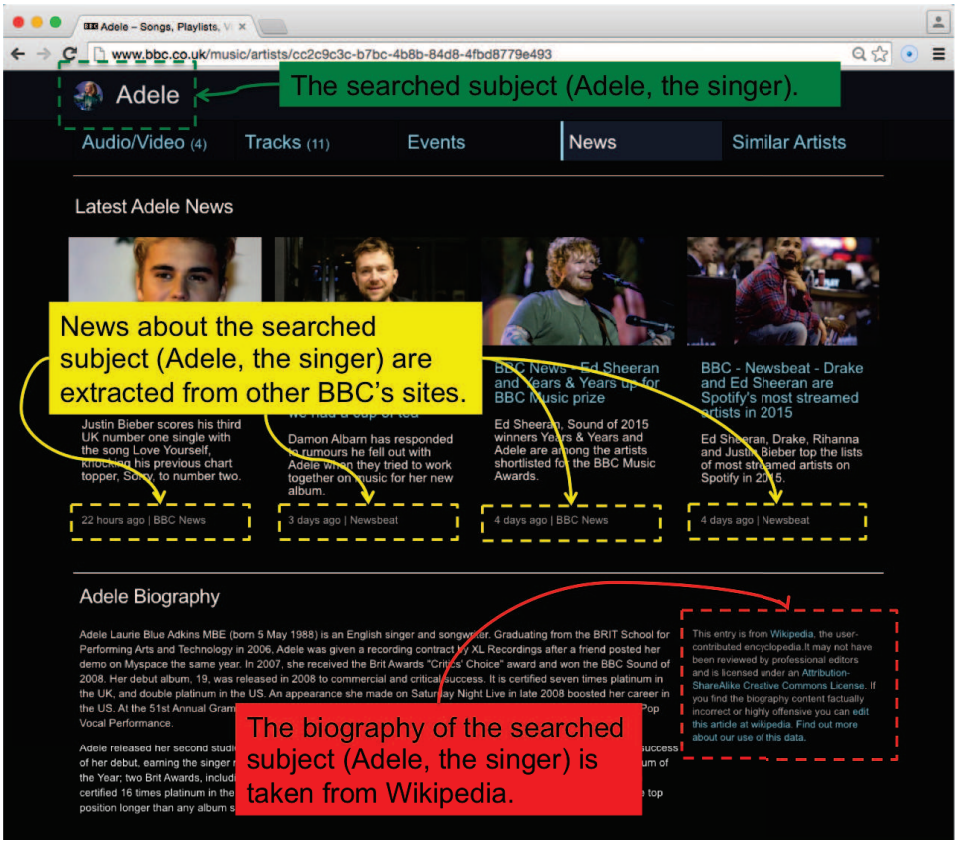


Figure 2.2: *BBC Music site exploits and aggregates data from different sources, both inner and external Web services, showing the potentiality of the Semantic Web and Linked Data. In this example, the Web page about the English singer Adele in the BBC Music site contains all the recent news about Adele, obtained by BBC News and Newsbeat sites, and her biography, extracted by Wikipedia.*

These Web sites tended not to link together. In fact, for example, if a user consulted the information about a program in the "BBC Programmes" site, he could not navigate to the Web page contained in "BBC Music" site about each artist that had played in that program. This lack of cross linking (both inner and external links) limited the type of user interaction the BBC was able to offer.

The new version of BBC's sites is completely different and many new features have been introduced. Look at Figure 2.2. It shows the current "BBC Music" site after having searched info about the English singer "Adele". At least two have been the important updates. First, even though there are already different BBC's sites for different topics, content about the same subject are not anymore isolated but aggregated from different sources. In fact, in the Web page in Figure 2.2 there is a section where news about Adele, extracted from "BBC News" and "Newsbeat" sites, are shown. It is important to notice that the subject of the news is the singer Adele not Adele Cambria (an Italian writer) or Adele Ammendola (an Italian journalist), and so on. Second, BBC's sites use data from external Web services, promoting the reuse of Web content. In particular, BBC Music is underpinned by the Musicbrainz music database and Wikipedia. In Figure 2.2, it is shown Adele's biography taken from Wikipedia. The features presented in BBC Music site are enabled by using of the Semantic Web and Linked Data.

The idea of adding semantics to Web resources was made famous by Tim Berners-Lee, Jim Hendler, and Ora Lassila in the article "The Semantic Web" [25], where they explain their vision in this manner:

"*The Semantic Web is not a separate Web but an exten-*

> *sion of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation*".

Therefore, the Semantic Web, relying on the already existing Web architectures (in particular REST), has to be used to describe the meaning of Web content in a universal way, not only in the form of natural language but also as machine-readable data.

In addition, to allow the reuse of Web content is necessary to connect data with each other and establish direct links when data refer to the same subject. This is the concept of Linked Data. It supports the aggregations of data from different sources to enhance the standard search experience and enrich content. More details about the Semantic Web and Linked Data are explained below.

2.2.1 The Semantic Web Technology Stack

The Semantic Web is based on a stack of components and technologies, as shown in Figure 2.4. The *Resource Description Framework* (RDF) [26], the foundation of this stack, is a data model that represents knowledge as **triples** consisting of a *subject*, a *predicate*, and an *object*. The predicate connects the subject to the object. A set of connected triples generates an **RDF graph**. There are different syntaxes to express triples, but the most used are RDF/XML [27], not particularly popular in developers' community because complex, and Turtle [28], easily readable for humans and machines. An example of different RDF syntaxes is illustrated in Figure 2.3.

Each element of a triple is identified by an IRI (except if the object is a blank node or a literal, i.e. a basic value such as a string or a

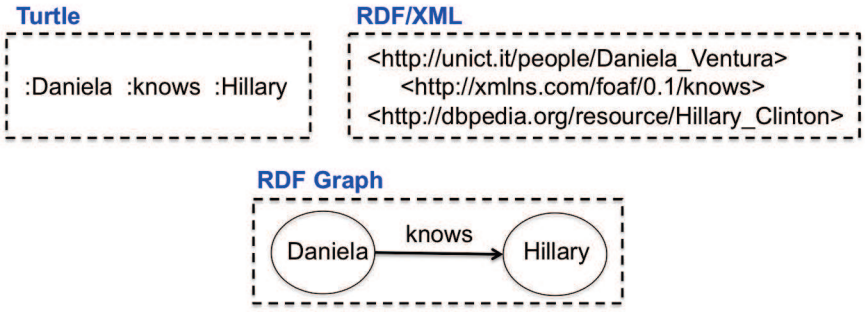


Figure 2.3: *Three different syntaxes to express RDF triples: Turtle, RDF/XML, RDF Graph.*

number). An IRI (*Internationalized Resource Identifier*) is unique and its domain name (or namespace) represents a set of concepts about a topic, i.e. an **ontology** (called also **vocabulary**). Each concept can have more than one instance that represents a specific entity. For example the concept *Person* belonging to the ontology *FOAF* [1] can have different instances, such as *Hillary Clinton*, *Elton John*, etc. Both the concept *Person* and the instances are identified by IRIs, so that machines can process and understand this knowledge.

The W3C standardized two vocabularies, *RDF Schema* (RDFS) [29] and the *Web Ontology Language*-2 (OWL-2) [30], to describe new vocabularies in an interoperable way. RDFS defines concepts to describe classes (and class hierarchies), data types, or properties similar to object-oriented programming languages. OWL-2 allows to express relationships about classes and properties, such as disjoint classes or union of classes. Anyway, both RDFS and OWL-2 are usually used to infer knowledge with automatic mechanisms

[1] `http://www.foaf-project.org/`

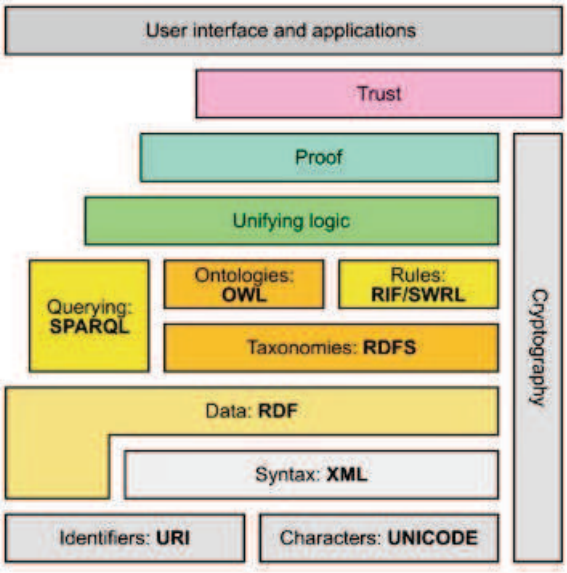


Figure 2.4: *The Semantic Web stack.*

(rules). The process of deduce new concepts is executed by **semantic reasoners**. Some reasoners are designed to use built-in knowledge as well as to allow the creation of new and custom rules (in different languages). Some of the most popular reasoners are Pellet [31], cwm [32] and EYE [33]. Finally, the last block of the Semantic Web stack is **SPARQL** [34], that not only specifies a language to query and manipulate RDF graphs but also is a protocol to invoke such queries over HTTP.

2.2.2 Linked Data

In 2006, Tim Berners-Lee realized that a problem of data expressed in Semantic Web is to browse on them (i.e. to pass from a data to an

other as happens for Web documents using hyperlinks). Therefore, he formulated the **Linked Data principles**:

1. Use URIs as names for things.

2. Use HTTP URIs so that people can look up those names.

3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).

4. Include links to other URIs so that they can discover more things.

The first principle recommends to use URIs to identify any abstract or physical concept, while the second specifically asks for HTTP URIs, i.e. URLs. In fact, URIs are a larger set (a superset) containing URLs. A common example of URIs that are not URLs are ISBN URIs. For instance, urn:isbn:7380221422521 identifies a book, but does not locate it. Third, these URIs should contains information in a machine-processable way. And forth, adding links to other URIs allows machines to discover the meaning of the concept. In other words, the knowledge resides in the links.

In recent years, many initiatives to create *Linked Open Data*, i.e. data which anyone can have access to, have been started. Large semantic data source are *DBpedia* [2], the representation in machine-interpretable format of Wikipedia, and *Freebase* [3], where RDF data are composed mainly by its community members. In this context it

[2] `http://wiki.dbpedia.org/`
[3] `http://www.freebase.com/`

is interesting to note that governments around the world are also providing public data (such as information about tourist spots in a city) in form of Linked Open Data.

THREE

ENABLING AUTONOMOUS COMPOSITION AND EXECUTION OF REST APIS FOR SMART OBJECTS

3.1 Problem Definition

Machine-to-Machine (M2M) interactions, the ability of devices to exchange data in order to execute a task not explicitly required by a human being, has become a promising domain for the next-generation communications, and is undergoing a rapid spread and development. M2M communications are expected to grow exponentially in the near future, aided by the large deployment of sensors, actuators, RFID/NFC tags. Ericsson's Media Vision 2020 research has predicted that by 2020 there will be 50 billion connected devices [35], and of these, 12 billion will be used only for machine-to-machine communications [36]. Therefore, M2M represents a huge potential business opportunity for companies and is expected to advance our lifes in a

significant way, covering a broad range of vertical markets (e.g. smart homes, cars, smart factories).

The first issue is to describe what *functionalities* a device provides using a machine-interpretable format. In other words, each machine has to be able to offer semantic descriptions of its services. As smart objects are commonly equipped with actuators that determine their states, semantic descriptions should also contain information about the behavior an object applies when its services are invoked. The service *composition* problem takes as input a set of service descriptions and a goal, and consists in generating a plan/workflow represented by a ordered sequence of services that have to be invoked in order to satisfy the goal. Finally, in order for two (or more) systems to communicate successfully, there has to be a well-defined and universal *semantics on the exchanged data*, and a way for knowing at runtime the used interfaces.

In this Chapter we present a method to enable machines to automatically compose and use services through standard semantic technologies. Despite the proliferation of protocols with which two or more machines could communicate (e.g. Bluetooth Low Energy, Zigbee, Xbee), we are witnessing the emergence of microcontrollers directly connected to the Internet and which can host small web servers and then provide REST APIs. According to the Web of Things (WoT), the physical world becomes thus "integrable" with traditionally Web services. Therefore, in our work, a *machine* is, first of all, a Web server that exposes a hypermedia API, which demands that a server supplied the possible next steps alongside each resource. That way, an agent does not need to know in advance how to use an API; instead, it can just follow the links at runtime through these supplied hypermedia

controls [37].

To allow cooperation between machines, we propose to describe the functionalities of APIs through RESTdesc [38] and to use JSON-LD [39] as data exchange format. Entities with reasoning abilities will be able to produce plans that involve not only services exposed by physical objects but also Web services, generating what is commonly known as "physical mashups". Instead of specific tools or languages for services composition, this approach only requires generic Semantic Web reasoners. At the end of this Chapter, we also evaluate a complete cycle from the production to the execution of plan(s) in order to determine how it can impact in term of time spent and number of requests done from resource-constrained devices.

This Chapter is based on our previous work [40].

3.2 Related Work

In this Section, we discuss some related works about syntactic and semantic descriptions of REST APIs and present some existing approaches in services composition.

3.2.1 Methods to describe the syntax and semantic of REST APIs

REST-based services are still almost exclusively described by human-readable documentation, due to a lack of universal formalism. Over the years, multiple interface description languages for RESTful services have been proposed but they have never really been adopted.

The *Web Service Description Language* (WSDL) [21] created to

describe Web Service in an XML-based language, was adapted to the REST services in WSDL 2.0 [41]. Even though the result of the second version has been to have a machine-processable form for describing Web services and RESTful services alike, the difficulty of using WSDL 2.0. for RESTful services is that it was not especially designed for resource-oriented services and as a result it is not perceived as suitable by developers.

The *Web Application Description Language* (WADL) [42] was from the outset designed for describing RESTful services in a machine-processable way. It is XML-based and it is focused on the resources and not on the operations, as WSDL. The main critique of WADL is that it is quite complex, in contradiction with the straightness and simplicity of RESTful services.

SA-REST [43], hRESTS [44] are other two ways to describe REST APIs. The REST APIs' interfaces and their corresponding meaning, are typically documented in HTML pages. SA-REST leverages this common practice by annotating those pages with RDFa in order to make the information machine-interpretable. hRESTS is very similar to SA-REST. The main difference is that hRESTS uses microformats instead of RDFa. However, for a more complete overview about these and other methods, we refer to [45].

Since JSON is one of the data exchange formats most commonly used on the Web, many approaches to describe RESTful APIs are based on it. Swagger [46], probably the most common, supports the documentation of REST services interfaces, but their semantics are still described in natural language. Hence, automated composition and discovery are not possible. It is prevalently used to improve human documentation with interactive controls so that the various operations

can be tested directly in the browser.

While interface description languages offer only the syntax of Web services, in order to enable machines to know the functionality of Web APIs, we also need of semantic descriptions. Common approaches to semantically describe Web APIs are based on the definition of specific ontologies. This is the case with OWL-S [47], WSMO [48] and WSMO-Lite [49], that are based on the ontologies built ad-hoc to describe the semantic of input/output/fault massages, functional and non-functional properties, etc. None of them, however, has had success in the developers community.

The Hydra Core Vocabulary [50] is adapted for hypermedia APIs with JSON resources. It describes hypermedia controls (similar to HTML's `<a>`, `<link>` and `<form>` tags) in a machine-processable way. Hydra makes it possible to identify a number of concepts commonly used in Web APIs, such as links or collections of resources. By the definition of a common ontology to describe these concepts, a client can construct HTTP requests at runtime in order to manipulate the server's state. The main limits of Hydra are: 1) it does not work with APIs that do not follow the hypermedia constraint; 2) it is very hard compose Hydra-enabled APIs from different sources.

3.2.2 Web Services Composition and Physical Mashups

The composition of Web APIs interests both academic and industrial world for over 10 years. One of the first tools that allows users to link Web services and Web pages, was Yahoo! Pipes, by now closed. It has been defined as a service that "*lets you remix feeds and create new data*

mashups in a visual programming environment" [51]. It also gave the possibility to a user to publish his created services (so-called pipes) in order that other users could use them. In other words, Yahoo! Pipes tried to anticipate the concept of *User Generated Service* (UGS).

Anyway, the composition of Web APIs is still a task that is mainly done manually by users. Many applications allow to connect logical blocks which represent Web services or physical devices. Two of the best-known of such types of tools are IFTTT [52] and Atooma [53].

The former is based on the paradigm "if-this-then-that". It makes available many channels (i.e. services like Facebook, Evernote, Twitter, and so on), each of them has its own set of triggers (the "if this" part) or actions (the "then that" side). A IFTTT-user can combine these channels in order to create personalized services, called "recipes". An example of recipe could be "if I am tagged on a photo on Facebook, then send a tweet". The latter is an application that provides user with a tool to create rules and compose services using the smartphone's capabilities (such as GPS, accelerometer or WiFi) and the applications already installed on the phone. An example of rule could be "if I am near home (a place identified by geographical coordinates), then switch on the wifi".

In the state of art we have found many architectures supporting creation, management and execution of service mashups directly on the Cloud. SenseStream [54] and SODIUM [55] are two examples. SenseStream is a platform consisting of two components: Data Layer and Mining Algorithms. The first converts data from different sources in a common knowledge. The second elaborates those data and realizes mashups. The SODIUM middleware consists of a set of languages and tools for creation and execution of workflows. Unfortunately, the

solutions proposed by these kinds of platforms are usually domain-dependent.

Different solutions are presented in [56], that proposes decentralized and stateless control-flow patterns in order to support REST APIs' composition, and in [57], in which an extension of BPEL for REST services is described.

In this context, activities of standardization led by ETSI TC M2M [58] and by the Alliances (like Open Mobile Alliance [59]) are noteworthy. Although they aim is prevalently to define architectures and specifications for machine-to-machine interactions regardless of protocols, an important part under study is data exchange based on Semantic Web in order to enable machines to autonomously interpret responses received by peers and provide new APIs created combining the basic services. The biggest challenge is to reach agreements between all the member manufacturers.

3.3 Basic Approach and Adopted Technologies

In order to create an ecosystem of interoperable and proactive devices able to communicate each other and satisfy goals involved in an algorithm in a fully automated way, the first step is to find the technologies able to answer the following questions: a) how can a server describe the semantic meaning of its APIs and its states in machine-understandable format?; b) how can a client invoke an function of an API without prior knowledge about the service name and what the server expects and returns?. The answers to these questions will be

described in the next subsections.

3.3.1 API Semantic Description

Candidate technologies, in order to describe the functionalities of the Web APIs, must take into account that real-world objects have properties that represent their physical characteristics and capabilities. The value for each property at a given time determines the object's state. For example, the defining *property* for a multicolor lamp is its color. Its *capability* is to change color when the value of this property is modified. Therefore, descriptions need to express the concepts of properties and interstate transitions.

We have selected RESTdesc [1] [38, 60] as method to describe REST Web APIs. It expresses the semantics of Web services by pre- and postconditions in Notation3 (N3) [61] rules and, integrates existing standards and conventions such as Link headers and URI templates. An example of a RESTdesc description is presented in Listing 1. It describes the process to switch on or off an irrigation pump. The precondition is included in the lines 6-12, while le lines 14-23 are the postcondition. This RESTdesc description indicates that if a resource exists whose type is *IrrigationPump* and a new *SwitchingState* would be set, invoking a HTTP PUT request to the URI which identifies the resource plus the value of new state, is possible the transition from the precondition to the related postcondition. In particular, the postcondition expresses the meaning to replace the old status with the new.

As RESTdesc is a technique based on N3 rules, every N3 reasoner

[1] `http://restdesc.org/`

Listing 1 This RESTdesc description represents a service to transit from the current on/off state to another for an irrigation pump

```
@prefix vocab: <http://example.org/vocab#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix bonsai: <http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#>.
{
  ?actuator a vocab:IrrigationPump.
  ?state a st:State;
      log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }.
  ?newValue a bonsai:SwitchAction;
      vocab:hasValue ?val.
}
=>
{
  _:request http:methodName "PUT";
            http:requestURI (?actuator ?val);
            http:resp [http:body ?actuator].
  [ a st:StateTransition;
    st:typeOperation "replacement";
    st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
    st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
    st:originalState ?state ].
}.
```

can process RESTdesc descriptions and apply suitable inference rules. We have used inference rules exactly in the postcondition of the example in Listing 1. Finally, the fact that RESTdesc descriptions focus on resources and links between them, makes them an excellent way to describe hypermedia APIs and to accomplish services mashups. The use of N3, a superset of RDF 1.0, supports variables and quantification, necessary to express such rules. N3 is compatible with RDF, and thus JSON-LD, which we discuss next.

3.3.2 Data Interchange Format and Services Interface

One of the main obstacles for more flexible clients are heterogeneous data format and services interface issues. Semantic technologies can mitigate the problem of interoperability and expressiveness in the exchanged data, but formats like RDF/XML, Turtle or N3 are widely disliked to APIs' developers and have not been optimized for Web APIs.

Listing 2 JSON-LD response sent by a irrigation pump when it is switched on

```
{ "@context" : {
  "vocab" : "http://example.org/vocab#",
  "schema" : "https://schema.org/",
  "bonsai" : "http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#",
  "actuatorState" : "vocab:hasSwitchingState",
  "hasValue" : { "@id" : "vocab:hasValue", "@type" : "schema:Boolean" }
  },
  "@id" : "http://localhost:3300/actuators/1",
  "@type" : "vocab:IrrigationPump",
  "actuatorState" : { "@type" : "bonsai:SwitchAction", "hasValue" : 1 } }
```

The power of JSON-LD [2] has been to combine technologies from both the world of Web APIs and the Semantic Web in order to produce a data interchange format human/machine-understandable. Each JSON-LD message is a self-descriptive message, both data and their semantic meaning are transmitted at the same time in the same message. An important its feature is that each JSON-LD message can be converted in RDF format and viceversa. Furthermore, since

[2] `http://json-ld.org`

JSON-LD is 100% compatible with traditional JSON, it is not *needed* to understand RDF to work with it.

JSON-LD requests and responses optimally fit with data represented in RESTdesc descriptions allowing a simple and straightforward execution of chains of RESTful APIs. Listing 2 is the response received from a client when it does the request to switch on the irrigation pump identified by *http://localhost:3300/actuators/1* URL. This JSON-LD message shows that the resource is a *vocab:IrrigationPump* and has a *SwitchingState* whose value is the boolean data "true" (the meaning is that the actuator is switched on).

Listing 3 JSON-LD of Listing 2 converted in to RDF (without prefixes for brevity)

```
1  <http://localhost:3300/actuators/1> a vocab:IrrigationPump.
2  <http://localhost:3300/actuators/1> vocab:hasSwitchingState _:b0.
3  _:b0 a bonsai:SwitchAction; vocab:hasValue "1"^^<https://schema.org/Boolean>.
```

The conversion of this JSON-LD in to RDF will generate Listing 3. Comparing the resulted RDF with the RESTdesc description of Listing 1, it is possible to see the high degree of correlation and correspondence in them. In fact, RESTdesc description not only suggests that the response to the PUT HTTP request (line 17) will be an actuator resource but also contains already the information decoded in RDF (the type of resource and properties). In other words, if we interpret a JSON-LD as triples, we can see it actually realizes the N3 description we have.

3.4 Autonomous Composition and Invocation of Hypermedia APIs

In order to illustrate the theoretical services composition and invocation process, we have implemented a use case on the field of smart gardens.

3.4.1 Use case

The objective of our use case is to create a smart garden system that autonomously monitors environmental conditions and decides how and when to act in order to ensure plants thrive. Real-time data about the current environment conditions can be got through sensors (light, temperature and soil moisture) directly located in the garden and associated to each plant or using the information produced by weather stations. In order to reach the ideal environment conditions, each plant is associated to one or more actuators (irrigation pumps, lamps, heaters and automated windows for greenhouses) whose status changes during the execution of the decision-algorithm.

The entities involved in our use case are:

- one Internet-connected microcontroller, i.e. an **embedded board**, that runs as server and implements the Garden API[3]; sensors and actuators are directly connected to the board (that could be an Arduino Yun, STM32 Nucleo, Intel Edison, and so on). Moreover, the board stores plants' information like type, species and ideal light, temperature and soil moisture values

[3] `https://github.com/dventura3/irrigation-api`

during the different parts of day (morning, afternoon, night). Finally, we assume that the embedded board has a GPS module in order to know its geographical coordinates.

- **Weather web server** which implements the Weather Forecast API[4]; it returns the current and forecast weather conditions for the required location (expressed in geographical coordinates or city name).

- **Smart client** which is implemented as a software module[5]; it has the task to execute a decision-algorithm in order to ensure the ideal environment conditions for each plant. No preknowledge about what the APIs do and how invoking their services is needed, because the smart client is able to understand the APIs' functionalities and, generate and execute plans that meet the goals whose the algorithm is composed.

The client can execute four types of algorithms progressively more complete:

1. In the first algorithm, the smart client uses only the **real-time information** given by sensors to determine what status the actuators should have.

2. In case that a sensor is temporarily out of service (e.g. it is not connected to the Network) or a plant has not been associated to a certain type of sensor, the use of the **current weather conditions**, given by a weather API, can overcome the lack of

[4] `https://github.com/dventura3/weather-forecast-api`
[5] `https://github.com/dventura3/plants-planning-agent`

sensors' data. Therefore, the second algorithm combines the current weather conditions and sensors' values to determine the actuators' states.

3. Using the forecast weather condition is a good practice to decide when to **switch on/off the irrigation system** in a garden or how to **increase or decrease temperature/light** in a greenhouse. For example, if a plant needs to be watered but it is expected that soon it will rain, it is possible to avoid turning on the irrigation pump and simply wait. Therefore, in this third algorithm, the client generates plans to know the weather forecast for the next three hours before to set the actuators' states.

4. It is the same of the third algorithm, except for the fact that the smart client uses the **forecast weather conditions for a longer time,** the next three days.

3.4.2 Smart Client's Architecture

Since the generation and the execution of APIs compositions is done by the smart client, we describe its architecture, depicted in Figure 3.1, highlighting its main features in the next subsections.

The client has a list containing IP and Host for each server (board or Web service) and, during the initialization phase, uses it to get all the RESTdesc descriptions invoking HTTP OPTIONS requests. After that, the Algorithms Manager can execute one of four algorithms every 5 minutes. Whatever algorithm is chosen, it consists in a sequence of instructions that involve remote services. Therefore, the Proxy is the block that has to select/generate the goal associated to

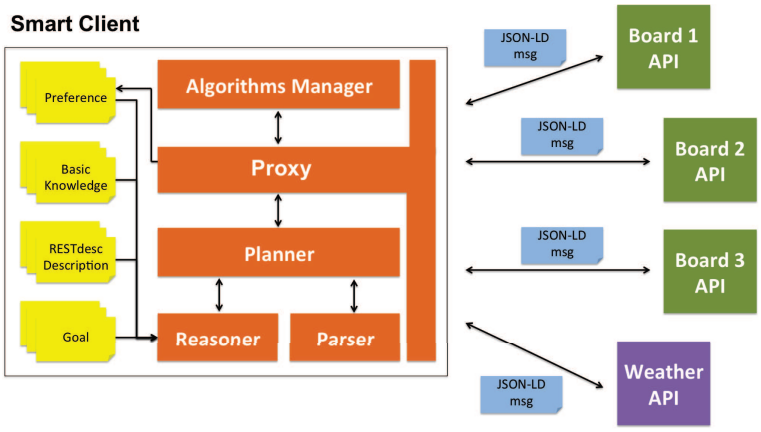


Figure 3.1: *The smart client's architecture. Plans are generated by Reasoner and parsed by Parser. The Planner selects only one of them and manages its execution. The services invocation is done by Proxy under the supervision of Planner*

each specific service required by Algorithms Manager and initialize the Planner. The latter executes the Reasoner which produces all the possible plans that achieve the goal. These plans are transformed in a machine-readable format by the Parser so that the Planner can determine that one to execute. The plan execution consists in to invoke the services in order and use the JSON-LD data of "X response" as input of "X+1 request" until the end of the plan. When the whole plan is terminated, the final result is processed by the Proxy and communicated to the Algorithms Manager, so that the execution of the algorithm can progress.

3.4.3 Reasoning and Parsing

All the four algoritmhs include the following goal: "*Get sensors values associated with each plant monitored by each embedded board known by the client*". In order to solve this statement, the first operation to do is to find all the possible plans that fulfil the goal. This is the Reasoner's task. As defined in [60], a plan is a chain of implications that starts from an Initial state and leads to entail the Goal state:

$$I \Rightarrow S_1 \Rightarrow S_2 \Rightarrow ... \Rightarrow S_x \Rightarrow G$$

Of course, the way to meet the goal may not be unique, so many chains of dependency (i.e. plans) can be found. As the RESTdesc descriptions are N3 descriptions, any N3 reasoner can generate these plans. We have used EYE [62]. The input information, used by Reasoner, is: 1) the RESTdesc descriptions associated to one embedded board at time and all remote web servers; 2) basic knowledge, that are RDF metadata about the resources exposed by each server or embedded board and, is used to infer knowledge; 3) preference, usually is something that we want to set. For example, if we want to switch on/off an actuator, this information has to be converted in N3 format as input for the reasoning process (in the above example we don't need any preference); 4) the goal expressed as N3 rule. After a pre-parsing operation, part of output generated by Reasoner, in order to solve the goal of the above example, is showed in Listing 4. The lines 1-5 contain the dependency among the different lemmas. The details for each lemma are described in the remaining lines. The *lemma27* is the initial lemma and we already know its URI (it is a basic knowledge). The URIs for the other lemmas (lines 11 and 23) will be found only at

plan's execution time. Therefore, at the moment, we know only the order in which the lemmas have to be run, namely the following plan:

$$I \Rightarrow lemma27 \Rightarrow lemma28 \Rightarrow lemma29 \Rightarrow G$$

In order to select one plan and execute the composition, the plans must be understood by client. The N3 format presented in Listing 4 is not easy to use for a machine/software. Therefore, the Parser converts the output generated by Reasoner in JSON object properly formatted.

3.4.4 Plan Selection and Execution

The Planner has the task to select and manage the plan execution. Using the example in the previous subsection, the Planner should decide one of the three plans shown in Figure 3.2. As previously said, only the first column of URIs, used to pass from Initial state to the next state, are known to the Parser, while we reveal the URIs of inner states just for descriptive purpose. Lacking a mechanism of cache server (we have not implemented yet) and taking into account the third plan does not have any reference to the associated plant(s), we have decided to always select the longest plan, i.e. the first. Although more complex, it is more reliable.

In order to execute the plan, the Planner needs to know all details of what each HTTP request to those APIs should like. Fortunately, this information is contained in RESTdesc descriptions and is kept by Parser. The execution starts from the first lemma and the result is used to find the URI(s) in order to execute the next lemma. For this scope, a bottom-up algorithm was implemented. To understand how it works, look at Listing 4. Suppose we have to find the *requestURI* of

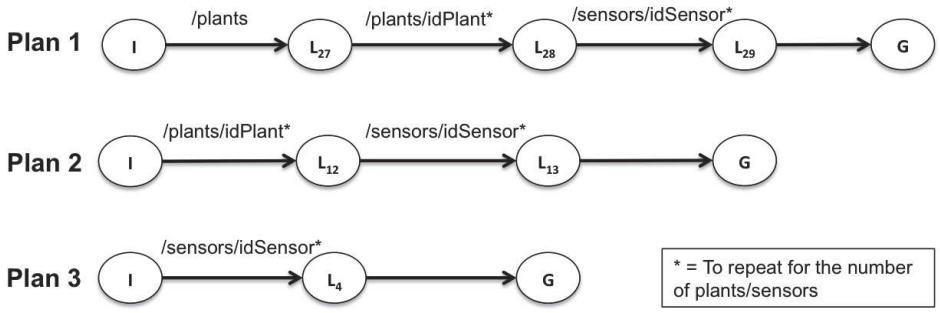


Figure 3.2: *All the three plans that could satisfy the goal about getting sensors values for each plant of each embedded board. The selected plan is the longest, i.e. the first*

lemma28. The blank node associated to the *requestURI* is *_sk5_2*. The first part, i.e. "sk5", identifies the blank node in the *lemma27* from which extracting the URI that we are searching. This blank node is *_sk5_1*, a Plant. We can't stop here, but we have to go up until the URI of *lemma27* because we have to know if *_sk5_1* is only one or are many plants. In other words, we have to understand if we are searching just one or many URIs, because one (or many) of visited nodes could be a collection. This is exactly what happens in our example. Infact *_sk5_1* is member (the semantic property is *hydra:member*) of the collection *http://127.0.0.1:3300/plants*. As demostrate in Section 3.3, REST-desc descriptions are strictly correlated with the JSON-LD responses returned by server. In fact, the *http://127.0.0.1:3300/plants* call returns a JSON-LD with a *vocab:PlantsCollection* whose members (the property is *hydra:member*) are the URIs (of type *vocab:Plant*) we are searching. We use them to advance in the execution of the plan. Similar considerations can be done to find the *requestURI* of *lemma29* and

complete the plan.

3.5 Evaluation

In order to evaluate our approach, we have measured the average time spent by the Smart Client's blocks to generate and execute each of the four algorithms. The simulations have been conducted using a Raspberry Pi Model B+ with processor Broadcom SoC at 700 MHz and 512 MB of RAM. The results are shown in Table 3.1. The measurements have been split in two operations: a) reasoning and parsing (to know the time used to produce all the possible plans that achieve the correspondent goal); b) selection and execution (involving the only plan that is really executed). The simulations have been conducted with two different configurations: a) one plant, one sensor and one actuator; b) three plants, each of them has associated the same three sensors and actuators.

The number of plants, sensors or actuators for each board does not influence the Reasoner and Parser's tasks. This happens because their work is strictly correlated with the number of APIs known to system and not with the data received when the APIs are invoked. In other words if the number of APIs involved in to achieve a goal increases, also the time spent to generate all the possible plans will increase linearly, as demonstrated in [60]. In contrast, the number of results returned by each HTTP request, done at each step of selected plan, influences greatly the time spent to complete the plan execution. While the operations to read the sensors values on the board are simulated (that means the time measured is not affected by the

time required to physically reading each sensor), our implementation of Weather API represents a JSON-LD wrapper for the forecast.io [63] web service and therefore is affected by network delays. This shows how, in a real-world setting, the time spent to generate the plan will be less than the time used to execute the plan. As expected the first algorithm is the fastest (less than one second) because it is the easiest but less accurate. On the contrary, the time spent to complete the execution of more sophisticated algorithms (as 3-4) is approximately five seconds.

An other important aspect is about the number of requests necessary to accomplish a plan. Note that selecting the longer plan has often as consequence to replicate HTTP requests already executed. For example, all the three plans used to get information about the ideal conditions of the plants or sensor values or actuators states associated with each plant, need to invoke: 1) */plants* and then 2) */plants/idPlant*. These two requests are done three times for the three different plans. A more accurate implementation could take account of historical requests. The alternative, as already suggested, is to use a cache server mechanism and choose not to perform the longer plan. Moreover, in other use cases, information as energy consumption could be used to choose the best plan.

Listing 4 Part of output generated by Reasoner. For the sake of brevity, we report only the plan that will be really executed in order to get the sensors values for one board

```
p:lemma27 a c:ServiceCall.
p:lemma28 a c:ServiceCall.
p:lemma29 a c:ServiceCall.
p:lemma28 c:hasDependency p:lemma27.
p:lemma29 c:hasDependency p:lemma28.
p:lemma27 c:details {_:sk3_1 http:methodName "GET".
  _:sk3_1 http:requestURI <http://127.0.0.1:3300/plants>; http:resp _:sk4_1.
  _:sk4_1 http:body <http://127.0.0.1:3300/plants>.
  <http://127.0.0.1:3300/plants> hydra:member _:sk5_1. _:sk5_1 a vocab:Plant}.
p:lemma28 c:details {_:sk36_1 http:methodName "GET".
  _:sk36_1 http:requestURI _:sk5_2; http:resp _:sk37_1.
  _:sk37_1 http:body _:sk5_2.
  _:sk5_2 vocab:hasAssociatedSensors _:sk40_1; vocab:hasAssociatedActuators _:sk41_1.
  _:sk5_2 vocab:hasIdealTemperature _:sk42_1; vocab:hasIdealMoisture _:sk43_1.
  _:sk5_2 vocab:hasIdealLight _:sk44_1.
  _:sk40_1 a vocab:SensorsPlantCollection; hydra:member _:sk45_3. _:sk45_3 a vocab:Sensor.
  _:sk41_1 a vocab:ActuatorsPlantCollection; hydra:member _:sk46_1.
  _:sk46_1 a vocab:Actuator; vocab:hasSwitchingState _:sk47_1.
  _:sk42_1 a <http://dbpedia.org/resource/Temperature>.
  _:sk43_1 a <http://dbpedia.org/resource/Moisture>.
  _:sk44_1 a <http://dbpedia.org/resource/Light>}.
p:lemma29 c:details {_:sk104_1 http:methodName "GET".
  _:sk104_1 http:requestURI _:sk45_4; http:resp _:sk105_1.
  _:sk105_1 http:body _:sk45_4. _:sk45_4 vocab:madeObservation _:sk108_1.
  _:sk108_1 vocab:hasTimestamp _:sk109_1; vocab:outputObservation _:sk110_1}.
```

Table 3.1: *Time (expressed in milliseconds) spent by EYE reasoner to generate and parse all the plans and to select and execute the only final plan that accomplish each goal for each algorithm considering two configurations: A (smaller dataset) and B (bigger dataset).*

Tasks		**Algorithm 1**		**Algorithm 2**		**Algorithm 3**		**Algorithm 4**	
Goal	**Operation**	**A (ms)**	**B (ms)**	**A (ms)**	**B (ms)**	**A (ms)**	**B (ms)**	**A (ms)**	**B (ms)**
Get Plants	Reas. + Pars.	361	366	426	417	403	408	394	399
	Selec. + Exec.	33	79	46	96	50	85	47	83
Get Sensors	Reas. + Pars.	380	387	465	459	401	420	413	420
	Selec. + Exec.	35	80	49	104	42	93	50	94
Get Actuators	Reas. + Pars.	–	–	572	560	489	504	482	498
	Selec. + Exec.	–	–	35	98	38	87	39	98
Get Current	Reas. + Pars.	–	–	464	470	552	514	467	501
Weather	Selec. + Exec.	–	–	657	702	776	785	837	799
Get Forecast	Reas. + Pars.	–	–	–	–	451	456	500	459
Weather	Selec. + Exec.	–	–	–	–	835	850	805	865

FOUR

CONTROLLING HETEROGENEOUS EVERYDAY OBJECTS THROUGH AN HOMOGENEOUS MOBILE APPLICATION

4.1 Introduction

According to the Web of Things vision, real devices will be connected as Web pages and will be accessible via URIs. On the other hand, it should be said that the real objects can have completely different requirements compared to Web pages: a) unlike a website, only a few people should access objects inside a Smart Space; b) we do not care where the server which hosts a web page is physically located, but we need to know where the objects that we want to control are placed; c) the web pages are made with standard technologies, the real objects instead are produced by different manufacturers and with different specifications. The main WoT scenario in which these considerations are fundamental is Smart Home/Building Automation where house-

hold appliances, such as ovens, air conditioners, TVs, media players, will be controlled by users via REST Web services.

In order to enable users to control their everyday objects, a fundamental aspect is the identification of objects. Several technologies have been proposed to identify Smart Objects: bar and QR codes, RFID and recently NFC. Each of these technologies has a different peculiarity that makes it suitable in certain contexts.

- QR code is a cheap approach which requires only a simple tag stuck to the object. Using QR codes we can achieve a direct identification of the object (the one we point). On the other hand this approach require a good camera for the recognition and might not be so fast.

- RFID (radio frequency identification) is a technology that allows the identification of objects by the application of passive tags that respond to queries made by a transmitter in a range of a few meters. Unlike the QR code, the RFID approach is not directional and can be used to find multiple items at the same time (for example, all the objects in a room).

- NFC (Near Field Communication) NFC is a standard for the transmission of data between two devices placed in contact or distances up to 4 cm. NFC is mainly used for contactless transactions such as micro-payments with smartphones, which are going to support this communication. Even NFC technology can be used for a short range identification of objects.

These approaches for objects' identification could be adopted today without any problems. The next step, however, should be to lay

the foundations for facilitating the interaction with the desired objects. These objects are often heterogeneous and each of them has a control interface which differs from the others. Setting up a timer for an air conditioner (e.g. 25 °C from 4pm to 8pm) should not be so different that setting up a cooking program for an oven. But, as things stand, these two operations may require two completely different procedures on the two devices, which more often require the user to refer to several user manuals. In recent years, more and more users have acquired the ability to deal with a mobile application right from the first use. In contrast to what happens for objects like household appliances, there is not a user manual for mobile applications. This has been made possible by a well-designed application design based on user-experience. Furthermore, Smart Objects can provide several features in different ways, and users may wish to access this features with any device connected to the network, e.g. smartphone, laptop, board computer car, tablet. We think that the approach adopted for web applications is the one which makes sense to use: we need to interact with Smart Objects using the virtual abstraction which mobile applications can provide.

The other important aspect to consider in Smart Home domain is related to privacy and data security while accessing objects. The amount of sensitive and context data is very large and they give some information about habits and characteristics of the user. Connecting to the Web objects of everyday life can give rise to serious security problems. These objects in fact, may be found via search engines and used by unauthorized persons. The search engine Shodan sorts background data on every computer attached to the Internet-including industrial control systems and computers embedded in household ob-

jects: such as televisions and garage doors. The issue of safety related to the search engines in the domain of WoT is treated in a comprehensive manner in [64]. If a user has a smart object, it should be himself to decide who can access to it. So, in a WoT environment it is necessary to handle the access to resources in a dynamic and safe way and provide mechanisms to prohibit or restrict the use of objects in agreement with user needs.

In this Chapter, based on [65], we introduce briefly the *webinos* platform realized within the EU FP7. Webinos defines a set of software components to enable the sharing of services from different devices owned by users in secure way. Each webinos enabled device can expose its capabilities as services. What we propose is an extension of the webinos platform by introducing a new API for controlling smart objects through web applications. This API uses RESTdesc descriptions proposed in the previous Chapter in order to describe the functionalities of the Smart Objects. We finally present a testbed application which, using the augmented reality and the proposed API, allows user to identify and control real objects in home automation scenario exploiting the user-experience of mobile applications.

4.2 The webinos platform

Webinos is an EC FP7 project which aims at defining and delivering an open-source platform and software components for the Future Internet in the form of web runtime extensions, to enable web applications and services to be used and shared consistently and securely over a broad spectrum of converged and connected devices, including

mobile, PC, home media (TV) and in-car units. The webinos basic concept is "write once, run anywhere". It is an approach to application development that is independent from the operating system and concerns web applications executed in the browser. In fact a developer has only to have knowledge about CSS, HTML and JavaScript to implement a webinos application.

Webinos introduces the concept of "personal zone" as the set of all devices owned by a user, taking into account several domains such as: mobile (smartphones, tablets), home-media (PC, TV, set-top boxes), automotive (in-car computers) and IoT (sensors, actuators). Webinos further provides a set of JavaScript APIs to allow developers to create web applications which exploit the features (software/hardware) provided by these devices. Each webinos device implements all, or a subset of, these APIs: a webinos application can then behave, without being changed, in the same way on different devices.

The innovation proposed by webinos, which makes it different from other platforms, is considering each device as a "service provider". The various features offered by a device (filesystem access, location, contact management, etc..) are exposed as remote services that can be invoked from other devices. This approach opens the door to new scenarios in which applications become cross-device since applications can use features which do not reside on the same device in which the applications are running, but in any other device that: a) belongs to the same personal zone, b) belongs to another personal zone (hence to another user) in case the owners have agreed to share their services. We can think at a personal zone as a set of services from several devices. Each device must be registered to the personal zone through an enrollment phase during which it is identified and information about

its services is saved and synchronized to the already registered devices. To make this possible, webinos defines two main components: the Personal Zone Hub and the Personal Zone Proxy.

The Personal Zone Hub (PZH) is the connection point for devices in the personal zone. It is responsible for the registration of the devices and the synchronization of services. The PZH can be installed on a user's machine or it can reside on the cloud. The PZP is a software component that must be installed on each webinos device. The PZP provides for a device the point of access to the personal zone: it is the component that communicates with the PZH for the registration and synchronization of services. In a basic webinos scenario, a personal zone contains two devices D_1 and D_2 which have been registered on the same PZH. If an application which runs on D_1 wants to use a service from D_2, it has to ask the PZH for this service, using the discovery API specified by webinos.

As already said, webinos defines a set of APIs for developing web applications that can be executed on multiple classes of devices. Each API defines a set of JavaScript methods to access certain device's capabilities. The most important APIs defined by webinos are: file, geolocation, TV, devicestatus and sensors and actuators APIs for IoT devices. Each capability can be considered as a service offered by a device and it is identified by: an ID, an URI and a service address.

To better understand the components on which the webinos platform is based, we report the service address used to uniquely identify a PZP inside the PZHs.

The service address, as we can see in Figure 4.1 , contains the following information:

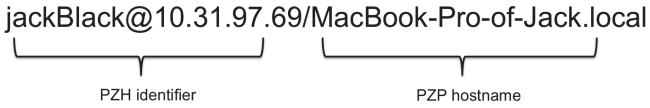


Figure 4.1: *Webinos service address composition*

- PZH identifier, composed by the user's nickname and the IP address of the device which provides the service;
- name of PZP where the service resides.

The webinos Discovery API can be used within a web application to search for a service based on its URI. It allows also to specify the device (or even the personal zone) where the service resides.

Each personal zone has a one-to-one correspondence with a user. Webinos provides the ability to connect together two or more personal zones. The services provided by devices of each personal zone are shared with the other zones and can be invoked by any device inside one of the involved zones. Obviously, for security reasons, webinos allows user to specify for every service which other user can access it.

Figure 4.2 shows a scenario where two users (Alice and Bob) connect together their zones which are identified by their PZH_A and PZH_B. In case 1, PZP_1 is using a service from PZP_2 which resides in the same personal zone. This is a case of *intra-zone* communication: only PZH_A is involved. In case 2, PZP_1 requires a service from PZP_4 which belongs to another personal zone. This could happen only after the two users have agreed to connect their zones and Bob has further decided to share services from PZP_4 to Alice. This is a case of *inter-zone* communication where both PZH_A and PZH_B are involved.

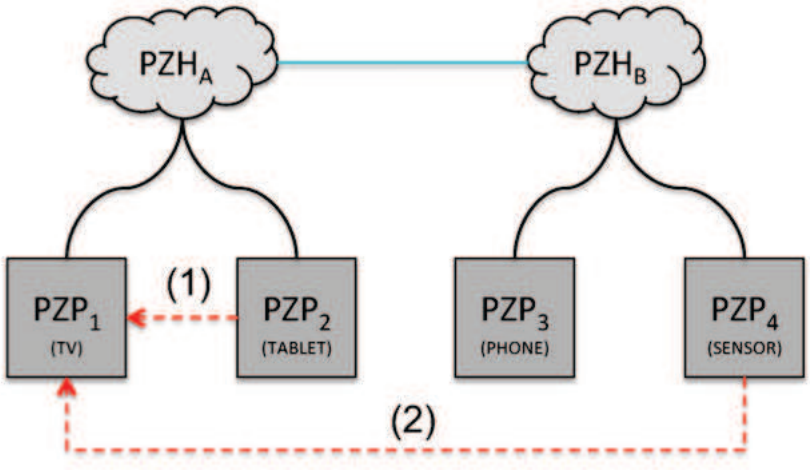


Figure 4.2: *Intra and Inter Zone communication*

The webinos platform is well suit the IoT/WoT world. It provides Sensors and Actuators APIs that allow to virtualize each sensor or actuator as a service in order to obtain an abstraction of the real object. However, considering generic sensors or actuators as Smart Objects is too restrictive. An actuator may be a simple light bulb but also something more complex that we call Smart Object. Sensors and Actuators APIs could present limits in some contexts since they were designed to interact with basic devices like current (or voltage) sensors or switches. If we consider a more complex object like an oven, it may perform many high-level operations, for example setting the cook program "180°C - Fan" from 6pm to 8pm.

Webinos is a modular platform: users can decide which API install on their PZPs in accordance with the capabilities of the devices. For

example if a user doesn't care about NFC functionality he can avoid to install the NFC API. This is a very important feature to avoid overloading devices with not useful APIs. This approach allows to users to exploit the resources that are actually needed and to developers to add, modify and remove APIs in simple way.

In this section we have shown how webinos allows to consider each device's capability as a service. This characteristic can be exploited to model the behavior of real objects. Webinos can be extended by the introduction of new APIs and also provides the possibility to decide in a flexible manner which services a device must be able to exhibit. These interesting features are the prerequisites for using webinos as platform for smart objects. In the next section we will describe how we extended the webinos platform by introducing a new API for describing and controlling Smart Objects from web applications.

4.3 Why webinos is a good platform for Smart Objects?

Webinos is not intended only for general purpose devices such as smartphones or tablet that implement the entire set of APIs. Each device can selectively choose which API to implement according to their characteristics. This modular approach allows the installation of webinos on devices with limited memory and computational resources. For these types of devices, webinos introduces the concept of "microPZP". MicroPZP is an implementation of the PZP when the device is too low spec to deploy a full PZP. A device supporting a microPZP typically has a target memory of 2 MB. A full-featured PZP

implements a rich set of functionality, including the ability to run interactive webinos apps, to expose locally-implemented APIs to those apps, and also expose locally implemented APIs to remote connected clients. This functionality naturally entails that the device hosting the PZP has certain capabilities - either explicitly (for example means for display and user interaction) and also implicitly (for example having sufficient memory and connectivity to be able to support a PZP). However, there is a wide class of potential webinos applications, ranging from small personal devices to mass-deployed IoT nodes, that are not required to support the full range of PZP functionality and have only limited hardware capability; factors such as device cost, battery life or connectivity would prevent such devices from being able to host a fully-featured PZP. These might be intended to expose locally-implemented APIs to remote clients, but are not required to support other PZP functionality such as being able to run local applications. The microPZP is therefore a perfect abstraction of a generic smart object capable of supporting the webinos platform. For the purposes of this paper we have a little forced the specification, whereas a *Raspberry PI* as a microPZP. Thanks to the webinos approach, the user's personal zone will not only contain TVs, tablets, etc., but also include Smart Objects such as fridge or oven.

Another important feature taken into account by webinos regards both security and privacy of users. If the security problem is important in computer science, such as for documents' protection, it will certainly even more delicate with regard to access to objects, as described in the Introduction of this Chapter. Within webinos, each user (the owner of a Personal Zone) can specify a set of security policies to decide which services have to be shared and which users will have the right to

invoke them. Since webinos applications are cross devices, they can use services that are not on the same device on which they are installed. A fine-grained access control to services is therefore essential to ensure the security of users and thus of their devices. This user-oriented security management applies well to Smart Objects, especially in the home scenario where most items can be used by several categories of people: parents and children, but also by people who are not strictly part of the family. For example, we can think at a smart door which can be unlocked through a web service invocation: parents, which have a young child, could authorise their baby sitter to open and close the door only in the morning, revoking the right in the rest of the day. Webinos therefore presents good conditions for being able to manage the user's smart objects.

4.4 A Smart Object API for webinos platform

In Section 4.2, we have already said that webinos defines generic APIs for sensors and actuators. Now we would extend webinos through introducing a support for more complex objects like household appliances. One possible approach to handle Smart Objects in webinos platform could be to consider them as microPZPs and create a new API for each of the appliances. For example, the oven could be a microPZP which only exposes the "Oven API" allowing users to set a cook timer, regulate the temperature or the cook program, etc. The same could be for the fridge which can implement an API to provide its state, the best before of the food it contains and so on. Unfortunately

this would lead to a huge number of APIs.

The API we are proposing provides three public methods:

- *getMethods*: returns a description of all the functionalities provided by the object in a well defined format. The description comprises information about the behaviour that the object applies when its services are invoked, and about the interfaces (input/output parameters, function names, etc.).
- *callMethod*: is used to invoke a specific service of the object.
- *callAsyncMethod*: is used to invoke a specific service of the object. The result of the operation will be sent back as a callback function's parameter.

As previous said, the work described in this Chapter is based on the paper [65]. In the original version of the webinos API for Smart Objects, we have defined our format, based on JSON, to describe objects' functionalities. Nevertheless, in light of the considerations that we have shown in Chapter 3, we decided that "getMethods" function will return a RESTdesc description of all the services exposed by an object.

4.5 An Augmented Reality Application to Control Smart Objects

Every day people use several applications for mobile devices, particularly web applications. Numerous studies have been carried out on how to design graphical user interfaces that allow users to learn and

understand how to use an application from the first time. On the other hand, the interaction with objects such as ovens, washing machines, still requires the average user to consult the instruction manuals supplied by the manufacturers and to deal with different interfaces (embedded displays, remote controllers, knobs, etc.).

In this Section we propose an application which, relying on the webinos platform and in particular on the API for Smart Objects described before, allows users to interact with real objects in a domestic environment. Thanks to the proposed application, a user can head his device towards a Smart Object to get a description about all the services and functionalities that object is able to provide. This description is provided by the Smart Objects API in the form of RESTdesc description. Using a proper transformation algorithm, the application uses the received information to create an on-the-fly graphical interface that allows the user to invoke the desired method by providing the required parameters. The user interface is created by the application in an intelligent way: it's adapted to the focused object and tries to render the best components depending on the object which the user is framing. For example, if a user frames an oven, the augmented interface will show knobs and wheels, but if he frames a thermometer, the application will display on the screen a gauge in the form of a thermometer. The purpose of this adaptive and consistent interface is to allow a more homogeneous and intuitive use of all functionalities of the Smart Objects.

Although a Smart Object has different functionalities, at the end, only the basic ones are used because they are easy to set up and keep in mind. On the other hand, using the proposed application, the average user will be able to realize which features the Smart Object

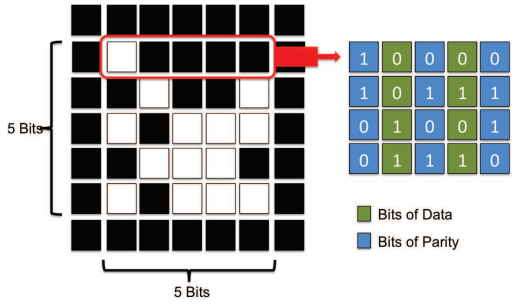


Figure 4.3: *ArUco Marker on the left side and the only 4 possible types of code for each row on the right side.*

is capable of providing, and use them, without the burden of reading complex instruction manuals. Using *Augmented Reality* (AR) we want to simplify the interaction between user and real objects and provide the same user-experience of web applications. Azuma *et. al* [66] claim that *An AR system supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world*. This suggests that users, framing an object with their devices' webcam, can view supplementary information on the screen such as sounds, videos, graphics or GPS data. Augmented Reality requires the object identification in order to provide in real-time a graphical layer on the screen of the device with object-specific information.

Our application uses an ArUco [67] marker to identify objects. Each marker is a 5x5 matrix where each row is composed by 2 bits of data (in green on Figure 4.3) and 3 bits of parity (in blue) thus 4 possible codes (the matrix on the right side in the image). Each row is decoded using bits on column 2 and 4. The 5 rows are then gathered to generate a 10 bits code that represents a number. We

decided to use markers such as ArUco because they are simple to apply on any smart object and don't require special additional costs. A user can realize and print his marker and stick it on a smart object in his home. We preferred to use ArUco rather than QR codes since the web library for decoding ArUco markers is faster and lighter. We tested the same scenario using QR codes but we noticed some issues using the decoding library on mobile devices (e.g. smartphones and tablets). As described in the introduction, other technologies usually adopted to identify objects are RFID and NFC. We did not take into account RFID since we don't have it on our smartphone or tablet yet. Moreover, although NFC is becoming available on mobile devices it requires short distances so, it could be suitable in the case of mobile payment or to open a gate but not for controlling an air conditioner.

Referring as example to a smart calculator, we show in Figure 4.4 the graphical interface created on-the-fly using the information in the description given by the "getMethods" of the framed Smart Object.

On the left side, there is the list of all the services that the calculator implements. In our case, there are only addition and modulo operations. When the user selects one of the services, on the right side the application generates a graphic interface which suits the input fields of the description.

Let's assume that the user wants to carry out the modulo operation. According to the description format discussed in the previous section, the calculator's service provides a method called "moduloMath" which requires two numbers (dividend and divisor) as inputs. The application, which can understand this description, generates on the fly two text fields for the required inputs. When the user provides values for dividend and divisor and presses the button, the

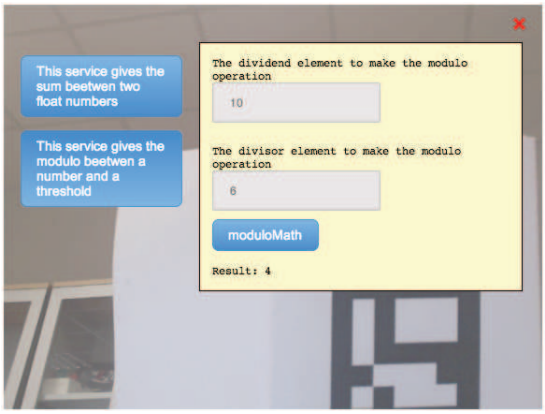


Figure 4.4: *Graphical user interface for modulo operation implemented by a smart calculator*

"callMethod" function is called on the calculator's service. Once the required operation has been carried out, the returned information is interpreted by the object.

To evaluate the application, we have considered a scenario where three Smart Objects are located in two different rooms (a kitchen and a living room). Each Smart Object is considered as a microPZP hosted on a Raspberry PI. Such objects are: a DVD recorder and an air conditioner placed in the living room and an oven in the kitchen. On each Raspberry PI we have sticked an ArUco marker to identify the related object. Each object is installed with a webinos PZP and has been registered in the user personal zone. Using the proposed application the user can use his tablet to point the object and interact with it using a UI built at run time, according to the methods' description provided by the smart object API which each object implements. This means that the application will provide a different UI depending on the ob-

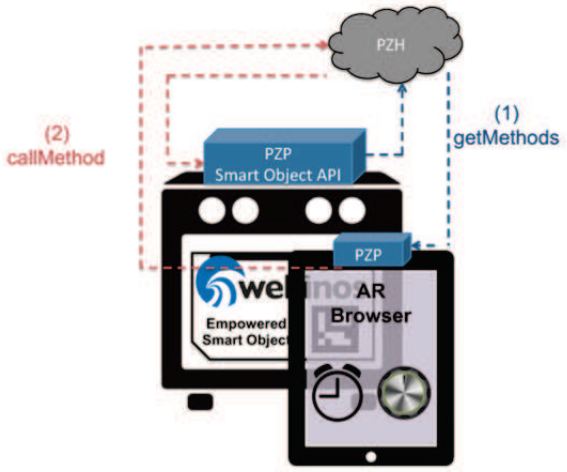


Figure 4.5: *Application scenario to simplify user-object interaction*

ject which is currently framed. Figure 4.5 shows the instant when the user points his tablet towards the oven. The communication between the tablet and the oven is mediated by the PZH: i) the application on the tablet reads the oven's id from the ArUco tag and asks for the PZH the service provided by the oven (since it implements the smart object API), ii) the application invokes the *getMethods* on the service to build the UI at run time, iii) the application invokes the *callMethod* on the service to interact with the oven.

4.6 Discussion about Future Improvements

The Smart Object API described in this Chapter is built ad-hoc for the webinos platform. In a more general context the "getMeth-

ods" function is not necessary, because in order to know the REST-desc descriptions of an Smart Object we can use HTTP-OPTIONS calls. Similar considerations can be done about the "callMethod" and "callAsyncMethod". Anyway, the basic concept of web application that we propose, i.e. the idea to interact with objects using the user-experience of mobile application, is the key concept that we would transmit to the readers.

Furthermore, currently add a new object means to include it in the personal zone of a user, print QR code with the id assigned to the object and place the QR code on the object itself. This approach is not very scalable and requires a commitment on user part. A future improvement of the web application can be obtained using an augmented reality framework called Vuforia [1] by Qualcomm. It uses Computer Vision technology to recognize images and simple 3D objects in real-time. Vuforia provides API in different languages (C++, Java, .Net), SDK for iOS and Android and Unity 3D. As consequence, developers can write an app that can reach the most users across the widest range of smartphones and tablets.

[1] `https://www.qualcomm.com/products/vuforia`

FIVE

SMART EDIFICE: GOAL-ORIENTED COOPERATIVE PLATFORM TO PERFORM TASKS FOR USERS

5.1 Introduction

Designing pervasive environments, where technology should be able to respond to people's needs is not simple. In fact, as anticipated in Section 1.3, there are many kinds of people with different and unpredictable needs and also, people may not expect the same results even when they act in the same way. This may be due to different, sometimes hidden, conditions such as their mood state or changes in the environmental context.

Ensuring an effective interaction between users and objects is the crucial point of Internet of Things/Web of Things paradigm, in which the objects of everyday life are connected to the network and can be controlled remotely by users. Today, in the Smart Spaces users have

a key role since they have to specify which objects to use and how. In the future, the way in which users will interact with objects within intelligent environments can radically evolve towards fully autonomous objects. In their most pure definition, Smart Spaces along with Home Automation have the aim to simplify people's lives, intercepting their needs and automating those operations that usually represent a burden (and a source of stress) or that cannot be carried out when a person is away from his/her home.

In this Chapter, we particularly refer to smart home scenarios characterized by the presence of both people, who live in the environment, and smart objects such as appliances and sensors which can autonomously cooperate to achieve some tasks requested by the users.

We propose a novel architecture, *Smart EDIFICE*, for a control system that enables autonomous and cooperative interaction between multiple objects. Through the proposed system, a user can express a goal (composed by actions and triggers) that is transformed by some functional blocks into a list of tasks, each of which is assigned to those objects that can better perform it in order to reach the goal. The problems encountered during the design and development of the proposed solution were related to three different perspectives, (1) regarding the user, (2) the proposed platform and (3) the smart objects, respectively. From the user's point of view, the main issues are connected to both the translation of a goal from the natural language to a machine understandable format and the feedback-based assessment to evaluate the way each object act to complete its task in relation to the goal.

The control system stands between user and objects: it must be able to understand the semantic meaning of the goal and decide, on the basis of appropriate criteria, which objects can be taken into account

to perform those tasks that will lead to the fulfillment of the given goal. It is fundamental for the control system to be able to select, among all the available objects, those most suitable to achieve the given goal and use them through the APIs they offer. This means that the platform has to be able to discover all the possible "plans", in the form of communication flows among objects, that satisfy a specific goal.

Furthermore, from the point of view of smart objects, the biggest challenge is describing *what* an object can do: in terms of both operations and resulting effects from the operations themselves. For example, if the goal is "reach a certain room temperature", the control system must be able to understand that the Smart Object "air conditioner" has the ability to act on the temperature parameter and can do this change through the exposed method *PUT:temperature/*{*int*} where the integer value is a temperature. To overcome this challenge, we have used the strategy proposed in Chapter 3.

Finally, the discovery phase, to find out all the plans, is followed by a filtering phase, to select the one to execute. We would highlight that the platform is able to do the selection based on user preferences and feedback. This ensures that the selected plane is always the most appropriate for the user, and then same goals could be executed using different plans because personalized for the user.

This Chapter is based on our previous work [68] and a paper currently under revision.

5.2 Open Issues

The evolution of the objects towards autonomous systems will involve several scientific and technological fields. For example, in order to cooperate with other peers to carry out a task, each object needs to know which operations the other objects are able to perform. For example, if a smoke detector has to notify that the smoke level is above a certain threshold, it should contact an object which can alert the user (for example a buzzer or whatever is able to emit a sound). Therefore, first of all, each object must provide a description of all its capabilities, in a machine-readable format based on semantic technologies in order to be understood and processed by the other objects. Second, it is also necessary the machine-client knows what the machine-server expects to receive as input and what will return as output. This problem leads to the use of semantic techniques also in the exchanged data.

The interpretation of users' requests expressed in natural language is also an issue of great interest in recent time. In fact, the interactions of users with the smartphone based on voice are increasing significantly. This fact is demonstrated by development of services like Google Now by Google and Siri by Apple.Therefore a current challenge is to associate to users requests expressed with both voice and text a syntactic and semantic analysis in order to understand and to act accordingly to them.

Another important aspect to take into account concerns the position of the user (which is, for our purposes, identified by his smartphone) inside the environment. Being able to figure out when the user is inside a particular area of the environment (e.g. a particular room) is very important for the system to be able to meet a goal which is

related to the user position. In the following we give a brief summary of several indoor user localization techniques which we have examined:

- *Audio Systems:* exploit some characteristics of sound signals to locate a smartphone inside a building. They are low-cost systems that work by using a microphone to record room ambience. These acoustic room fingerprints are used to recognize previously-tagged rooms when they are revisited[69].

- *VLC (Visible Light Communication) Systems:* exploit some LEDs properties to enable visible light communication which in turn is used to perform very accurate indoor positioning. Simple and power-efficient switch-mode amplifiers can be used on the transmission side. On the receiver side, the camera sensor is able to decode the location information transmitted by the lights, and also to compute accurately the position relative to this lights and even the device orientation in space.

- *Radio Signal Systems:* use radio signals to determine the position of a device. They are the most used and studied by the scientific community for several reasons: i) they do not require a line of sight, ii) they use technologies which are already embedded in modern smartphones, iii) and they can easily work in background mode. The classical approach for these systems is to use the RSSI (Received Signal Strength Indication) to localize the smartphone inside an indoor environment. This can be achieved (a) measuring RSSI values and comparing them with previously measured values (saved in a database) to estimate the position of the user (this approach, often used with Wi-Fi

signals which are available for free in the buildings, provides a good level of accuracy but needs experienced personnel to create the fingerprint for each new area), or (b) by using triangulation algorithms (this approach is less accurate in real environments due to the multipath problem).

One of the most recent promising technologies to perform a low-cost indoor localization is the *Bluetooth Low Energy* (BLE), which is a radio technology which let two devices to communicate each other with a reduced power consumption compared to the classic Bluetooth, but with a low data rate transmission. In a BLE configuration we can identify *Slaves* devices which broadcast an "I'm here" signal which is called "advertisement" and *Master* devices which listening for advertisements and extract information from them (e.g. the IDs of the slaves) or connect to the slaves to exchange data. Almost every moderns mobile operating systems support natively BLE. Apple has recently proposed "*iBeacon*" standard which relies on Bluetooth Low Energy technology. It allows mobile applications to listen for BLE signals from a new generation of low-cost wireless transmitter called iBeacons placed inside environments in a known point, in order to understand the smartphone's micro-location in an accurate way: when a smartphone is inside the range, it is able to sense iBeacons, localize itself in term of proximity to the beacons and enable some functionalities programmed by the app developer.

Moving the focus on objects deployed in security/privacy-critical environments such as industrial automation or smart homes, it is important to pay particular attention in ensuring the access to the smart objects functionalities only to those users who have permission. In

facts, making the objects publicly available on the Internet through a public IP address lead to potential unconditional accesses to them by anyone. Privacy and security issues, which were already extensively discussed in the field of web services, if considered in the context of Smart Objects may imply critical problems in relation to the safety and security of users. While a privacy fault in a social network can lead to the inadvertent spread of contents, on the other hand, wrong security configurations of the objects inside a smart home can allow strangers to attempt the safety of users. It becomes clear that is absolutely necessary for these objects to implement some sort of security mechanisms (e.g. firewall white-list or by exchange of documents) to avoid unwanted intrusions, but in many cases objects have limited resources in terms of memory and processing capability, therefore it is hard for them handling complex algorithms such as cryptography ones (we will talk about a lightweight secure algorithm for resource-constrained devices in Chapter 7). For these reasons, often the security problems for limited-resource objects are resolved at the architectural level, by avoiding to publicly expose them on the Internet: individual objects connect with the outside world through a gateway, which will be a more powerful element able to guarantee the adequate security and to control the access to the objects from the web in a safe way.

5.3 State of Art on IoT Platforms

In the last few years a lot of architectures, models and frameworks have been introduced to enable the simple management of smart-home appliances and services. All these works are characterized by the as-

sumption that users interacts (almost) directly even with a Smart Home management system to fulfil their needs.

Recently some commercial solutions have been presented by companies like LG, Revolv, Samsung, SmartThings and Staples that understood the importance of this new market. Some of these solutions are already available (see Revolv [70], SmartThings [71] and Staples [72]) and basically consists of one or more physical hubs and an application for handset to put together and manage hardware like lights, locks, speakers and sensors produced by different manufacturers. Other commercial solutions like those presented by Samsung [73] or LG [74] promise to give a seamless experience while managing the smart-home but probably will work only with their respective appliances. For what concerns academic activity, many solution have been proposed and implemented.

In [75] authors considers those problems related to the configuration and the updating of applications in the Smart Home context. They present a distributed system that allows the remote managing and deployment in the context of a distributed and pervasive environment for cognitively impaired people.

In [76] authors present a platform and a framework for design, development and deployment of smart-home services. Their work embeds the use of OSGi technology develop and deploy home services using common automation technologies. The authors propose also the ROCob API Specification to enable developers building different kind of applications, such as presentation layer applications (e.g. a web based UI), monitoring applications to collect data and send them to a backbone server and other home control and pervasive applications.

In these works, the main objectives were focused on how to manage

in a simple way, remotely or not, Smart Home services and objects. Users still needed to interact with the Smart home giving precise and step-by-step commands to achieve their objectives. This paper, instead, presents a system that, through the introduction of some more intelligence in the Smart Home, enables users setting their goals but not the required steps.
Other works that take into account the existence of an intelligent home and user defined rules to cover different aspects are [77, 78, 76, 79, 80].

Most of the works cited can be revised due to the advancements in IT and electronic technologies. For example in recent years, new general purpose platforms based on cloud computing are spreading. They could be used not only in smart-home context but also in other different domains, such as health, smart-cities, logistics/retail. They, also, provide an infrastructure to enable device-to-client and device-to-device communication between heterogeneous devices and to develop innovative applications. In recent times, some platforms like Xively, Evrythng and Alljoyn, designed to manage IoT objects and communication, gained a good success. Xively [81] is a web platform based on PaaS infrastructure. The aim of this platform is supporting the use, composition and sharing of "things". To achieve such a goal, it provides a range of services to read/write from/to user devices, store data and selectively share them. Along the same line, Evrythng [82] supports the creation of an online profile, called Active Digital Identity (ADI), for products or other physical objects. An ADI is simply a web resource, identified by an URI, with information about a "thing" in the form of dynamic attributes (e.g. where it is now) called "property", or static attributes (e.g. when and where it has been made) called "custom field". One of the most important international play-

ers which aim to propose a cross-platform technology to provide this common language is the Allseen alliance, a non profit consortium who is developing and proposing Alljoyn [83]: "an open source project that provides a software framework and set of services that enable interoperability among connected products and software applications, across manufacturers, proximal to create dynamic networks". It give to developers and companies the possibility to easily create applications for *Internet of Everythings* (IoE) and aim to become the basis for a standard de facto IoT intercommunication technology.

Our work differs from those presented because it is not only about "things composition" and "data sharing" but covers all the aspects needed to understand the goal a user wants to achieve, to break it up in tasks and to manage them to get the desired results. To conclude this overview on the state of the art, it is important to mention also a couple of ongoing projects for home automation operating systems which are HomeOS [84] from Microsoft Research Lab and Linux MCE [85] these projects represent the growing interest in this area.

5.4 Architecture Description

The architecture proposed in this article is depicted in Figure 5.1. We can distinguish the smart home side, the user side, and three macro blocks which are named "Understanding block", "Discovery block" and "Task Coordinator block".

The smart home side contains the Smart Objects. We can identify, according to their roles, two kinds of Smart Objects:

- Smart Home Objects: are the Smart Objects which are inside

each room, as the lights, lamps, alarm clocks, smart home appliances, etc. They are responsible for performing tasks.

- Smart Home Gateway: is placed one for home/Smart Space. It is a central unit that acts as intermediary between Smart Home Objects and the architecture. Its tasks include: monitoring of user's location, collecting objects' data (like energy consumption or time to complete a task) and supporting the execution of operations for each task.

Each Smart Home Object is a RESTful server that exposes RESTful APIs semantically described through RESTdesc and communicates using JSON-LD format. In addition, it has an associated ontology, represented in N3 format, which contains information useful to classify each object and its "Features of Interest" (FoI: they are physical quantities manipulated or observed by an object when it is in the ON state). For example the temperature is a FoI for heaters. All this information is stored in a database and used by the architecture during the services' discovery process, necessary to identify which object or group of objects can accomplish a given task.

On the user side we have the smartphone (which in our hypothesis identify the user) and an application which: (a) enables the user - through the UI - to set the goals and the preferences; (b) is responsible for performing indoor/outdoor localization of the user. The goals and the preferences expressed by the user are passed to the Understanding Block. The Understanding Block is responsible for translating the goal in a common format which the Task Coordinator block is able to read. To perform the translating of the goal expressed by the user in natural language (by talking to the phone, or by writing a text message) the

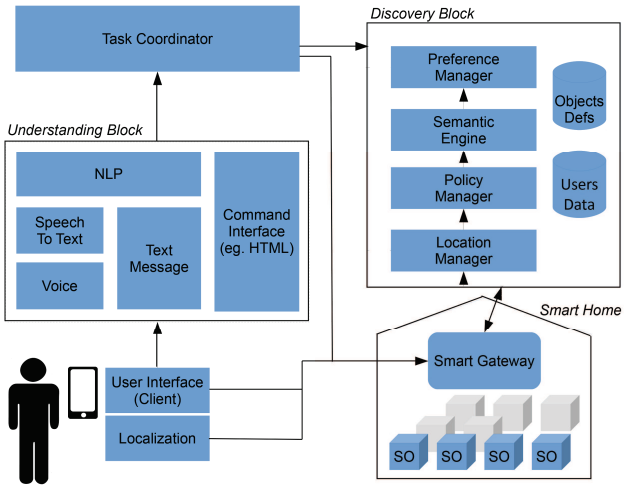


Figure 5.1: *Smart EDIFICE: architecture overview*

Understanding Block contains a Natural Language Processing block (NLP) and some ontologies. Otherwise data are translate through a command interface.

The outdoor/indoor localization information collected by the smartphone are sent to the Smart Home Gateway and then forwarded to the Location Manager which is the block able to locate the user inside the house. The Semantic Engine is the block able to generate all the plans that achieve a goal according to the availability of the objects. The filtering operation is done by Preference Manager that determines the plan to execute and transmit it to the Task Coordinator, responsible to coordinate the order of requested services by different users.

In summary, when the Task Coordinator requires the objects that

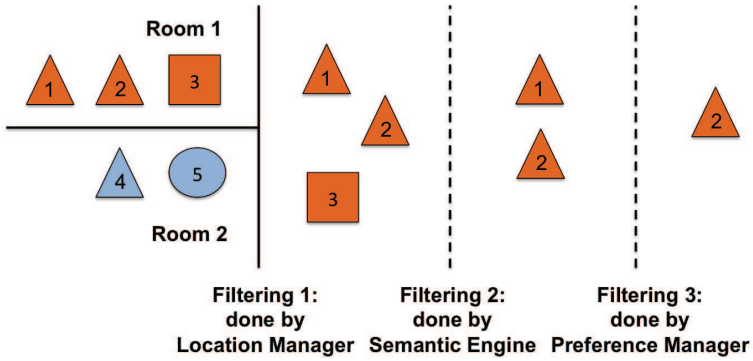


Figure 5.2: *Discovery and filtering process segmented in three phases. Objects with similar features have same shape (triangle, square and circle). Objects located in the same room have same color (orange and blue). Filtering 1: based on availability and location of objects. Filtering 2: based on user goal and objects features matching. Filtering 3: based on non-functional elements (historical data, user feedback).*

can perform a certain task, a series of filterings are done on objects available in the Smart Space, as outlined in Figure 5.2. The first filter is applied by the Location Manager, that selects only the active objects present in the environment in which the action will take place. The next filter is applied by the Semantic Engine that selects only those objects with useful features to perform the action. The last filter is applied by the Preference Manager, based on user's preferences for a given action, which choose the object to utilize among multiple objects with similar capabilities.

The three main cloud blocks, mentioned before, are detailed in the following three sections.

5.5 Understanding Block

The part of the system which mediates between the user and the rest of the platform is the "Understanding" macro block. In our vision, users express the objectives (goals) and expect that objects coordinate themselves to meet these goals. The Understanding Block (UB) stands in the middle between the user and the task coordinator. Its main role is to translate a goal, which could be provided by the user in several ways, to a machine readable format which the Task Coordinator will able to understand. Each goal is formed by two parts: the action (the final result to be achieved) and the trigger (a condition that, once verified, triggers the action). Assuming that the entry point for the user to the system is the smartphone, he could specify and assign goals to the objects inside the house by using his voice, a text message or through an assisted user interface. The user interface can be built with Web technologies in order to be cross platform for different devices. The interface guides the user in specifying the goal through the typical HTML controls (radio buttons, selection boxes, etc..). The conversion of the goal, in the format understandable by the Task Coordinator in this case is straightforward. Goals expressed through voice commands or text message are considerably more complicated to be handled. A voice command needs to be converted into a text passing through a "speech to text" service, afterward it can be considered in the same way as a text message. A Natural Language Processing (NLP) block receives as input a string of text in natural language and extrapolates the SVO (Subject-Verbs-Object) elements of the sentence. The most difficult task which the NLP block should carry out is to recognize the semantic meaning the user wanted to attribute to the sentence. NLP

comprises several outgoing research tasks.

Using tools such as Link Grammar[1] or the Standford Parser[2], the command "*Start the washing machine with the program J at 8:00 PM if the washing machine is full load*" is splitted into its main components, as shown in Listing 5.

Listing 5 Syntactic processing of a user request that is resulted using Link Grammar

```
(ROOT
  (SINV
    (VP (VBD Start)
      (NP (DT the) (JJ washing) (NN machine))
      (PP (IN with)
        (NP (DT the) (NN program) (NN J)))
      (PP (IN at)
        (NP (CD 8:00))))
    (NP (NNP PM))
    (SBAR (IN if)
      (S
        (NP (DT the) (JJ washing) (NN machine))
        (VP (VBZ is)
          (NP (JJ full) (NN load)))))))
```

The parts of speech are then associated with the semantic meaning using appropriate rules and ontologies. Therefore, the output provided by the Understanding Block to the Task Coordinator is a JSON-LD object, like that showed in Listing 6, which contains information about the goal, split into *actions* (A) and *triggers* (T) and their relationships (AND or OR).

[1] `http://www.abisource.com/projects/link-grammar/`

[2] `http://nlp.stanford.edu:8080/parser/index.jsp`

Listing 6 List of Actions and Triggers expressed by a user converted in JSON-LD format by Understanding Block

```
{ "@context" : "http://ub/contexts/goal1.jsonld",
"triggers": [{
    "what": "db:Washing_Machine"
    "func": "ex:checkStatus",
    "value": "ex:Loaded"
}],
"triggers_relationships" : [],
"actions": [{
    "func" : "ex:programming",
    "what": "db:Washing_Machine",
    "value" : "J",
    "startingTime": "20.00"
}],
"actions_relationships" : [] }
```

5.6 Task Coordinator

The Task Coordinator (TC) is the heart of the proposed architecture. His job is to take care of the goals expressed by users and generate tasks to be distributed to various objects in the house to get to the satisfaction of the goal. If goals expressed by different users involve the status of the same Smart Objects, the Task Coordinator applies management policies based on users' priority.

Since a goal is composed by actions and triggers, the first thing that the Task Coordinator does is to check all the triggers. There are three main types of triggers:

- Time-dependent: the action will be carried out at a certain time (e.g July 4th, 2014 at 8:30 AM) or in accordance with a certain periodicity (eg. every Friday at 7:00 am, every day of May at 5:00).

- User Position-dependent: the action will be performed when the user is in a certain position within the home. With appropriate indoor localization techniques, the system can track users (their smartphone location) and ensure that the action "turn on the TV" could be launched only when a user enters the living room.

- Object-dependent: the execution of an action will be subjected to the occurrence of appropriate situations that relate to the status of one or more objects. For example, a rule for a washing machine could state "start every day at 5:00 AM using the program 6" only if the machine is full loaded.

In the case of an object-dependent task, the Task Coordinator:

1. Obtains, from the Discovery Block, which objects will be responsible for generating all the information needed for the trigger (i.e. a plan). For example, if the trigger is: "if the temperature in the kitchen is less than 30 degrees", the Discovery Block will return the URI of the object that can measure the temperature in the kitchen, the name of the service to invoke (e.g. temperature), the type of method (e.g. GET), and expected inputs/outputs extracted by the RESTdesc description of the correspondent object.

2. Uses the information retrieved to get the status of the subject (e.g. temperature). The Task Coordinator initializes a thread on the Smart Home Gateway that periodically listens to changes in the state until it satisfies the trigger condition (e.g. temperature is less than 30 degrees).

3. Requires the Discovery Block to find a new plan involving the objects that can carry out the action and the information to invoke them.

4. Performs the action by calling the appropriate methods on the objects following the sequence of operations expressed in the plan.

The execution of the selected plan for both object-dependent triggers and actions follows the implementation of "planner" and "proxy" blocks described in Chapter 3.

In the case of a time-dependent task, the task coordinator will perform 3^{rd} and 4^{th} steps at a certain time of the day.

In the case of a user-position dependent task, the Task Coordinator waits until it is notified that the user has entered a specific area or environment. In facts, the user through the smartphone, which is able to sense advertising signals sent by BLE beacon placed in every room, can notify his presence in a specific place.

5.6.1 Management of inconsistent goals

Each scenario set in a smart space (home, office, shop, etc) has a multitude of users that act in the environment. It is not uncommon that users express different and inconsistent goals. A simple example of inconsistent goals may be when user-1 says: "*if I am at home, turn on the lights in the garden*" but another user-2 says, "*from 22:00 to 07:00 turn off all the lights at home*". We can notice that the actions of both goals have an effect on the same Features of Interest, i.e. the temperature. Therefore the two triggers may be inconsistent when

they overlap, i.e. when the user-1 is at home between 22.00 and 07.00.

The first problem is to figure out if two or more goals are inconsistent. First we decided to understand if the triggers of two goals occur at the same instant. Then we compare the respective actions considering objects used and their features of interest dealt with to fulfil the two goals. Therefore, the Task Coordinator, after that the Discovery Block returns the plan that satisfies the action of a goal, checks if the triggers associated with the same goal overlap with the triggers of other goals currently in running state. In case of simultaneous triggers, the Task Coordinator compares the actions using the characterization of semantic properties and classes. In particular, if a property or a class "*disjointWith*" another, the Task Coordinator can deduce an inconsistency between goals.

After having understood that two (or more) goals are contradictory, the Task Coordinator have to choose which one runs. We decide to use an approach based on user groups and privileges. Therefore, users are grouped according to their the role and a hierarchy of groups is fixed (using the mobile app). For example, in a house father and mother have highest priority on their son who has more privileges than his friends and relatives. In order that this mechanism can work, every time a new goal is expressed, it is necessary that the system keeps track of the user who has created this goal (the device ID is used to identify the user). When the Task Coordinator has to decide which goal runs among inconsistent goals, it verifies the group membership of the users "owners" of the goals. Of course, it chooses to execute the goal with higher privileges. If the goals are expressed by users which belong to the same group, the goal that is executed is the first that is expressed (and is already in running state).

5.6.2 Secure communication

As already mentioned the Task Coordinator takes care of the goals defined by a user. To reach the proposed objectives it has to communicate with all the objects which work is needed to achieve each goal. Since, Task Coordinator and objects are connected to the Internet, the communication between them is exposed to security risks that can lead to safety risks when an uncontrolled (or unwanted) object's behaviour can harm people around it. Managing security might not be easy in the current scenario in which the different kind of elements (devices) that need to be secured have different hardware resources and in particular might not have big computation resources.

One solution to cope with these issues is the use of VPNs (Virtual Private Networks). A VPN allow to extend a private network, such as a LAN (Local Area Network) across a public network, such as the Internet. In a secured VPN all the nodes (clients) can communicate as they are in the same network and in a secure way. In our scenario, all the objects without enough resources to manage cryptographic functions (needed to act as a VPN client) can connect to a VPN router that will create the needed LAN among objects and Task Coordinator. Another approach, consists in reducing the resources needed to create a secure channel between each object and the Task Coordinator. This second method is more close to the IoT/WoT philosophy that promotes unique URIs (IP addresses in VPN are private addresses: they are not reachable out of the network and are not unique). The basic idea is that every Home has its own components to guarantee a reasonable security level for the communication among objects inside the Home itself and among components that reside in the Cloud or

are external to the defined environment. To secure communication we propose the structure depicted in Figure 5.3.

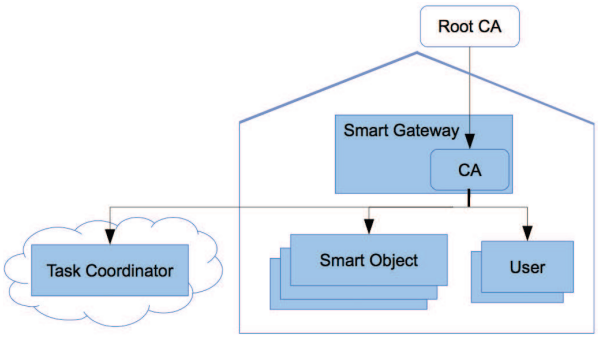


Figure 5.3: *PKI for the proposed system*

A Certification Authority associated to each home and manages certificates for all the entities, i.e. Task Coordinator, smart objects and users, that interact within the proposed system. Each entity the first time (e.g. user registration, object setup, Task Coordinator/remote components binding) requests a certificate from the HomeCA. This certificate is exchanged among entities the first time they come in touch. Then, it will be possible to instantiate SSL/TLS channels to communicate. Some works like [86] and [87] provide solutions to use public key cryptography and SSL also on cheap 8 bit platforms. These solution make possible to ensure the secure communication among the various entities.

5.7 Discovery Block

The Discovery Block is composed by three simpler blocks: Location Manager, Semantic Engine and Preference Manager. Each of them executes one filtering operation, as described in Section 5.4; the first on objects' availability; the second on objects' features; the third on users' preferences/feedback. In the following subsection, they are explained more deep.

5.7.1 Localization

The Location Manager (LM) is the block which is responsible for locating the user, who, as previously said, is identified by his own smartphone, inside the smart home. The whole process of indoor user localization can be summarized in the following steps:

1. The user has installed on the smartphone the app which constantly monitors his geographical position. The app works both in foreground and background modes and uses the classical localization system embedded on smartphone (GPS, Wi-Fi, Cell towers) to locate the user in an safe-battery way. The localization accuracy needed in this step is quite large (an approximation of about $200mt$ is permitted, but it should be set appropriately based on the user requirements).

2. The user moves towards the smart home (e.g. he left the office and went to home for dinner): when the position detected by the smartphone is in the previously set proximity range of the smart home (e.g. $200mt$), the smartphone switches in the

discovery mode for indoor localization (as we will explain more deeply later, in our proposed system this means to activate the Bluetooth Low Energy and deactivate the GPS). In order to distinguish situations where an user is always near the smart home but he is not moving towards it (e.g. he works in an office near the smart home) the set proximity range must meet the user requirements and some algorithms to discriminate this kind of ambiguity should be implemented.

3. As the user moves inside the house, its location is notified to the smart gateway that forwards this information to the Location Manager. The Location Manager knows if there are user position-dependent triggers and the concerned rooms. Therefore, when the user enters in one of these rooms, the Location Manager informs the Task Coordinator which is responsible for performing the previously set goal.

Therefore, in our platform, in order to detect when a user enter inside a room, we have to understand when the the smartphone of the user is inside a room. We can exclude all approaches which need a line of sight for locating the smartphone (such as visible light systems), because the localization process must be transparent for the user and must be work even if the smartphone, for example, is in the pocket. We also can exclude approaches which uses audible sounds because they are too invasive and annoying for the user. Audio fingerprint approaches are not accurate enough to use it standalone. From our point of view the best solution is to use iBeacons and BLE technology because it is simple to deploy, low-cost and low-energy: each room is associated with an iBeacon that advertises its ID. The smartphone can

put itself in discovery mode, sense the iBeacon signal in the proximity range and send its position to the Smart Home Gateway. If all the iBeacons are well-positioned (usually near the rooms' entrance) and by properly setting the RSSI thresholds, it is possible to achieve a good level of accuracy.

Another responsibility of the Location Manager is to determine if a smart object is available at the moment of the discovery phase, or if it is not reachable because it is turned off or is no longer inside the smart home. To achieve this the Location Manager try to reach the Smart Objects involved in the task and analyze the response. If the response is "not available", the Location Manager sends a notification to the Semantic Engine that will remove the plan involving the object currently out of service.

5.7.2 Semantic

In order to carry out the discovery operation and dynamic selection of smart objects that provide services of interest to meet user's needs available in a place, the block Semantic Engine (SE) has been added to the proposed architecture. In particular, its tasks are to collect all the descriptions of the RESTful APIs exposed by all the objects located in the smart environment and create all the possible plans to achieve a trigger or a action expressed by the user. Each plan has to contain the order of HTTP calls involving the objects able to satisfy the task.

In our architecture, such as problem has been resolved using REST-desc descriptions and a semantic reasoner, in our case EYE (Euler YAP Engine), able to interpret Notation3 (N3) rules. We suppose

that the RESTdesc descriptions and other basic semantic information about the Features of Interest should be written directly from the manufacturer of object and could be downloaded by the object using the Net.

The Semantic Engine is singled out in two stages:

- during the object's bootstrap: the Semantic Engine is responsible to keep updated the list of descriptions. In fact whenever a new object performs the bootstrap phase, the Semantic Engine is notified with information about the descriptions of object's services;

- during a user's request: the Task Coordinator needs to know what objects can perform the action or unblock the trigger. Therefore it interrogates the Semantic Engine for giving start to the discovery phase.

To better understand the operations executed by the Semantic Engine in these two phases, we have divided the rest of the paragraph in two subsections.

Operations of the Semantic Engine during the Bootstrap Phase: Keeping the List of Descriptions Up-to-date

The bootstrap phase occurs only one time for each Smart Object and consists in setting (using technology like WiFi-Direct) the parameters that the Smart Object should use to connect itself to the home WiFi (SSID, PSW and Gateway's IP). The enrollment process needs that object's owner approves the request via web app and associates a room to the object. If the Smart Object enrollment request is accepted,

its descriptions are forwarded from the Smart Home Gateway to the Location Manager. The LM converts the raw data about object's location in N3 format and stores them into the database together with RESTdesc descriptions and basic knowledge (like the FoI). At the same time the Understanding Block is notified in order to update the content of the web application with new state, type of objects and FoI.

Listing 7 RESTdesc description of a RESTful service used to switch on/off an alarm clock.

```
@prefix ex: <http://www.smartobject.org/example#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix bonsai: <http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
{
?obj a ex:AlarmClock.
?state a st:State;
  log:includes {?obj ex:hasSwitchAction ?oldValue.}.
?newValue a bonsai:SwitchAction;
  ex:hasValue ?val.
}
=>
{
_:request http:methodName "PUT";
  http:requestURI (?obj ?val);
  http:resp [http:body ?obj].
[ a st:StateTransition;
st:typeOperation "replacement";
st:oldComponent {?obj ex:hasSwitchAction ?oldValue.};
st:newComponent {?obj ex:hasSwitchAction ?newValue.};
st:originalState ?state ].
}.
```

For example, we suppose a user has three smart objects, a radio, an alarm clock and a lamp. Each of them exposes a service that lets us

switch on and off the object. The description of this service provided by the alarm clock is presented in Listing 7.

The meaning of this description is: if there is a resource whose type is an alarm clock and a new switching action is requested (by a user) to be set, then, invoking a PUT request to the URI which identifies the alarm clock plus the new setting value, the alarm clock will change its state. In addition, the information about the location of the Objects has to be described using N3 in order to be used by reasoner and will be updated during the bootstrap phase for each object. Therefore, if we suppose to have the lamp in the kitchen and, the alarm clock and the radio in the bedroom, the correspondent file will contain the information of Listing 8.

Listing 8 Semantic description to express what smart objects are contained in each room.

```
@prefix ex: <http://www.smartobject.org/example#>.
@prefix r: <http://www.rooms.org/example#>.

r:bedroom_1 ex:hasSmartObject ex:MyLamp.
r:kitchen_1 ex:hasSmartObject ex:MyAlarmClock;
            ex:hasSmartObject ex:MyRadio.
```

Operations of the Semantic Engine during the Discovery Phase: Plans Production

In order to understand how the Semantic Engine works during the discovery phase, we suppose the user request is: "*if the gas in the kitchen is greater or equal to 200 units, then switch on the alarm*". A trigger and an action compose this goal. We can focus the attention only on the action, although similar operations are done also for the

trigger. The Semantic Engine will receive from the Task Coordinator the JSON-LD object shown in Listing 9 representing the action requested by the user.

Listing 9 JSON-LD object send from the Task Coordinator to the Discovery Block.

```
{ "@context" : "http://tc/contexts/action1.jsonld",
"what" : "foi:Alarm",
"value" : "ON",
"func" : "ex:hasSwitchAction" }
```

The Semantic Engine processes this data and dynamically, at runtime, creates the query in form of N3 rule to select all the objects able to produce an alarm. The action requested by user and converted in N3 rule is presented in Listing 10.

Listing 10 Query in N3 rule format generated by Semantic Engine in order to find all the objects that have as feature of interest the concept of Alarm and their states can be switched on.

```
@prefix ex: <http://www.smartobject.org/example#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix foi: <http://www.featuresOfInterest.org/example#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
{
?obj foi:hasFeatureOfInterest ?alarm.
?alarm a foi:Alarm.
?state a st:State;
  log:includes {?obj ex:hasSwitchAction ex:SwitchON.}.
}
=> { ?obj a rdfs:Resource. }.
```

To infer new knowledge by the reasoner, we have create an ontology of features of interest (FoI) that uses the predicate “canBe”. When two

features of interest are interconnected each other through the predicate "canBE", the Features of Interests of an object are augmented.

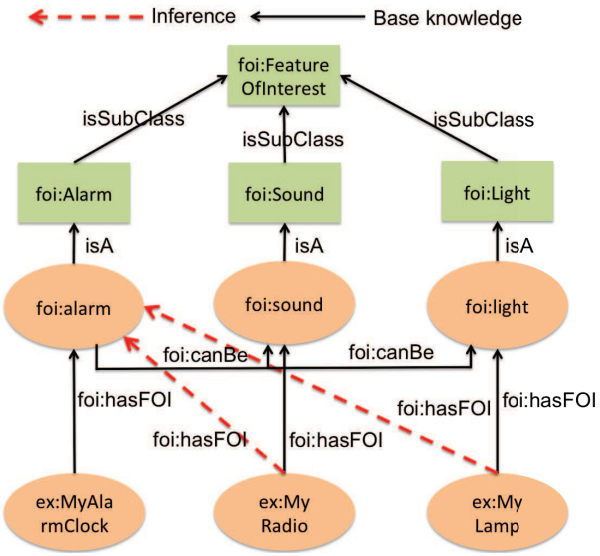


Figure 5.4: *Ontology to indicate the Features of Interest of three types of Smart Objects: radio, alarm clock and lamp*

In our case, an alarm can be every type of sound or light, as the ontology shows in Figure 5.4. Therefore, the reasoner will find not only the alarm clock as object able to emit an alarm, but also the radio and the lamp because their Features of Interest will be not only respectively the sound or light but (thanks to the reasoner's deduction) also the alarm.

The reasoner uses all the RESTdesc descriptions, the basic knowledge about the FoI, the objects' locations, the inference rules and the action/trigger converted in N3 rule format in order to produce all the plans that satisfy the goal, as shown in Figure 5.5. After having

executed the reasoner, the Semantic Engine checks what objects are active by asking the Indoor Localization the list of detected objects. Finally, the Semantic Engine communicates the plans to the Preference Manager block. Therefore, the output of Semantic Engine will be a list of active objects and the HTTP sequences that should be executed in order to resolve the action or trigger of a user's request.

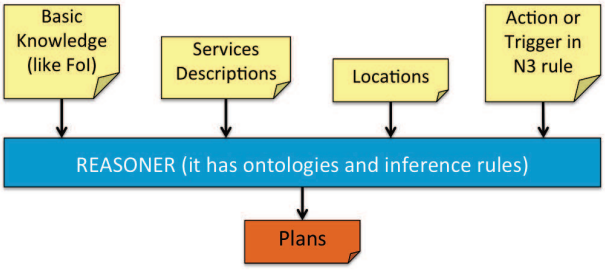


Figure 5.5: *Inputs used and Outputs generated by N3 reasoner executed by Semantic Engine*

5.7.3 Preferences

In the proposed system, the Preference Manager (PM) is the component which takes into account objects' features, user's preferences and feedback to better address the filtering process. The main aspects to select the more convenient plan are:

- *Features of Interest's type*: using a SPARQL query on basic knowledge files it is possible to know that some types of FoI are in contradiction each other. For example there are objects that generate cold and others that generate heat. In these cases, the PM has to select two different more convenient plans. Only at

execution time, in base of user request, one plan will be really used by Task Coordinator.

- *User' preferences using historical data*: each time a service is invoked on a object, the Smart Home Gateway measures the amount of energy used to complete that operation and stores this data and the correspondent timestamps in the database. Therefore the PM could use this information to evaluate what plan requires less energy or less time to achieve the goal according to the user's preference. If there is not information about the consumed energy, nominal values, semantically represented in the knowledge files, are used.

- *User feedback*: each time a goal is executed the user can give a vote through the web app to express his approval about how the goal was reached. The user feedback is a numerical value from 1 (goal reached using a very inadequate plan) to 5 (goal reached using a good plan).

A user can order the priority of four types of preferences through the web app. These four possible preferences are:

- *maximum energy saving*, that selects the plan that requires less power regardless of the produced effect;

- *maximum quickness*, that selects the plan that reaches the goal in less time regardless of the energy consumption;

- *priority to energy saving*, that selects the plan with higher energy saving and quicker effect in the environment;

- *priority to quickness*, that selects the plan that spends less time to reach the goal and uses less energy;

We think that it is necessary that users explicitly set their preferences and give feedback to the platform during the learning process. With the progressively knowing of its users, the system will decide the plan to be executed in a fully autonomous manner.

5.8 A Use Case: Same Goals but Different Plans Adapted to Users' Needs

In order to evaluate the platform and describe the logical flow of the interactions among users, the various blocks of the platform and Smart Objects, we have implemented a use case. In our scenario, we suppose that two people, an elderly lady and a young student, live in their houses where Smart Objects have the same arrangement: a heater, a fan and a temperature sensor are located in their bedroom; an air-conditioner, a mixer and an oven are placed in their kitchen; a water heater is located in their bathroom, as illustrated in Figure 5.6. All of these appliances are equipped with a Wi-Fi module, are able to measure their energy consumption, are equipped with RESTdesc descriptions to describe their RESTful services and use JSON-LD to exchange data.

The interface of the web application drives the user in the process of specifying the goal, through HTML components (like pickers) and by using If-Then-That paradigm as shown in Figure 5.7. Therefore, we suppose that both the users express the same goal through the web app (the speech recognition and interpretation is not under study in

Figure 5.6: *Smart home scenario to evaluate Smart EDIFICE platform*

this thesis):

"If starting time is 21.00 and ending time is 22.30, then temperature has to be equal or greater than 22° Celsius".

The Understanding Block converts the goal in to JSON-LD object using inner knowledge. In fact, each time a new object executes the bootstrap phase, its capabilities (in semantic format) and properties are conveyed to the UB in order to update the graphical interface of the web application. The produced JSON-LD is with the same form of the example described in Listing 6 and is communicated to the Task Coordinator.

The TC is able to understand that the trigger is time-dependent and therefore it waits until 20.50 (ten minutes before the task has to be executed) in order to manage the action. We have currently decided to handle the action shortly before the condition is verified and not when the Task Coordinator receives the goal from Understanding Block, because we must be sure that objects selected for satisfying the action are available (which means that their states must be on

Figure 5.7: *Screenshot of the web app that shows the process to generate a action for a goal*

and they must be connected to the Internet).

The Action Management starts when the Task Coordinator sends to the Semantic Engine the action in order to be processed. The Semantic Engine is composed by three modules: query generator, reasoner and parser of plans. The first one elaborates the JSON-LD representing the action, converts it in N3 rule(s) and applies the appropriate management logic (which depends on the type of operation requested in the action). In our use case, the action is a setting operation that causes two discovery procedures. Hence, first of all, the reasoner module selects all the objects which are able to modify the temperature in the environment and reach the desired temperature in an autonomous manner. Afterwards, it find those objects that observe the physical quantity "temperature", and those objects that have an effect on temperature if their status is "switched on".

The number of plans that the Semantic Engine finds are four. They are: 1) to set the desired temperature and the air conditioner will

try to reach it ("Plan-1"); 2) to use the temperature sensor and the fan in the bedroom ("Plan-2"); 3) to use the temperature sensor and the heater in the bedroom ("Plan-3"); 4) to use the air conditioner (through the service to obtain the current temperature) and the oven in the kitchen ("Plan-4"). These four plans are translated in an easier form by the parser module. Both the water heater and the mixer are not objects able to modify the temperature (water temperature and air temperature are different semantic concepts), and as consequence they are ruled out from the plans.

Before to forward the plans to the Preference Manager, the Semantic Engine requires to the Location Manager if all the objects involved in the plans are available. We can suppose that all the object are active, therefore all the plans are communicated to the PM, which is responsible for filtering them and selecting the more convenient.

Listing 11 SPARQL query invoked to know if the objects, that have temperature as Features of Interest, can warm or cool the environment

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foi: <http://www.featureOfInterest.org/example#>
PREFIX ex: <http://www.smartobject.org/example#>
SELECT ?obj ?prop
WHERE {
  ?obj a ex:SmartObject.
  ?obj foi:hasFeatureOfInterest ?f.
  ?f a dbpedia:Temperature.
  ?f foi:typeOfFoIGenerated ?prop
}
```

As previously described, the Preference Manager decides what plan is more convenient depending on the combination of three aspects. Using the SPARQL query of Listing 11 on basic knowledge

files (they are ontologies and triples of data), it understands that two different plans have to be selected: one between the air conditioner and the fan to cool; the other between the air conditioner, the heater and the oven to warn. The other aspects taken into account are the users' preferences and feedback. As a simplification, we assume that the feedback are the same for all objects for both the two users and therefore do not affect the choice of the more convenient plan. Contrariwise both the elderly lady and the young student have set different preferences: the former would prefer quickly reaching a comfortable house's temperature without to be interested in energy consumptions (i.e. her preference is maximum quickness); while the latter is interested to save money in his rented house (i.e. his preference is maximum energy saving).

	Working Mode Min		**Working Mode Max**	
Plan	**Power (W)**	**Effect (°C)**	**Power (W)**	**Effect (°C)**
Plan-1	1000	+2.0	1400	+4.0
Plan-3	810	+1.0	1610	+2.0
Plan-4	2000	+2.0	2800	+4.0

Table 5.1: *List of energy consumptions and number of Celsius degrees that increase in 1 hour considering each plan generated by Semantic Engine involving objects able to warm the environment.*

Looking at Tables 5.1 and 5.2, using semantic information about nominal power combined with historical data (that contain both timestamps and absorbed energy from each appliance), it is possible to calculate the average power used for each plan to warm or cool the

environment. We are not considering the fact that the objects involved in the plans are located in different rooms. In this use case we want to select only one plan to execute (after all in the goal it is not expressed the location of the objects such as kitchen, bathroom or the whole house). Therefore, according to users' preferences, the Preference Manager selects the Plan-1 (using only the air conditioner) both to warm and to cool the house of the elderly lady; while it chooses the Plan-4 (using the heater and the temperature sensor) to warm and the Plan-2 (using the fan and the temperature sensor) to cool the house of the young student. This example shows as Smart EDIFICE is flexible to users' needs. Even if the goals expressed by people are the same, the choice of how achieve each goal is adapted to the specific person.

	Working Mode Min		**Working Mode Max**	
Plan	**Power (W)**	**Effect (°C)**	**Power (W)**	**Effect (°C)**
Plan-1	1000	-2.7	1400	-3.9
Plan-2	310	-0.8	560	-1.2

Table 5.2: *List of energy consumptions and number of Celsius degrees that decrease in 1 hour considering each plan generated by Semantic Engine involving objects able to cool the environment.*

Finally, in order to execute the action of goal the Preference Manager sends to the Task Coordinator the two selected plans (to warm and to cool), the URI of the objects, and the types of HTTP calls. It is job of the TC to understand if it is necessary heating or cooling down the environment. Therefore the first HTTP call is to GET the current temperature and, then, the TC can decide which plan has to be used. The cooperation of objects is always supervised by Task

Coordinator which is responsible for giving commands to the Smart Home Gateway in the correct logical order.

SIX

EMBEDDING INTELLIGENT ECO-AWARE SYSTEMS WITHIN EVERYDAY THINGS TO INCREASE PEOPLE'S ENERGY AWARENESS

6.1 Introduction

The potential of the IoT/WoT to drive a sustainable everyday life is more than probable. This fact is easily evidenced through its current application domains such as agriculture, energy saving at home or in industrial settings and the pollution and traffic control within the cities. One example of such potential is the Google's Nest Thermostat, perhaps the most famous IoT gadget during 2014. Their designers disclosed that it can become carbon neutral in a period of just eight weeks after its first usage. Carbon neutrality refers to the greenhouse gases that were created by manufacturing and distributing the device are offset by the energy savings one obtains from using it [88].

From our point of view, the two main causes of energy waste are: 1)

inappropriate use of everyday devices from users due to the intangible nature of energy, i.e. people are unaware of energy waste; 2) use of not eco-friendly devices. The former especially occurs in working places where workers do not pay a monthly invoice to electricity providers. About the second cause, it is still controversial how other myriads of IoT/WoT devices (everyday consumer appliances, fitness trackers or kitchen appliances) can be also labeled as green devices along their life-cycle: from manufacturing to disposal [89]. These new devices are designed to replace old-fashioned ones. Therefore, their inclusion will rise to an augment of electronic waste that probably will end in the landfill.

This Chapter, based on our two article [90] and [91], describes an approach that addresses the challenge of energy saving. Our proposal lies in two pillars. First, it is focused on embedding intelligence through open hardware electronics within everyday appliances of shared use (e.g. beamers, coffee-makers, printers, screens, portable fans, kettles, etc.). Our aim is transforming these electronic devices into Internet-connected eco-aware everyday things rather than replacing them by new ones. As a proof of concept, in the presented work we have focused on electronic coffee machines located in four different work-laboratories. Each coffee-maker performs two tasks: *1)* report its daily usage pattern to a Cloud-server; and *2)* get its usage prediction back in order to know the more convenient working-mode to set in the following week. In other words, the Cloud-service infers when it would be advisable (in energy terms) that the coffee-makers remain turned on or off as a function of the number of people that previously used them, i.e. on the consumption patterns from past weeks.

The second pillar, it is to increase eco-awareness of people through

the provision of persuasive feedback to users about the necessity of keeping the appliances on or, in contrast, switch them off over certain periods of time throughout the work-day. We believe that timely and frequently persuasive interactions could help users to operate the appliances as efficiently, energy-wise, as possible.

6.2 Background

As it will be described in the next Section, in order to save energy for our coffee-machines, we need to forecast the number of coffees intakes for hour in the next working week. Therefore, first of all we have to select an algorithm to obtain predictions. We have compared two methods, Artificial Neural Networks and ARIMA models, in order to determine the technique that better fit our data.

Artificial Neural Networks as a soft computing technique are widely used as forecasting models in many areas. They are data-driven, self-adaptive methods with few prior assumptions. They are also good predictors with the ability to make generalised observations from the results learnt from original data, thereby permitting correct inference of the latent part of the population. The wide use of Artificial Neural Networks is due to their very efficient performance in solving nonlinear problems including those in real world. This is in contrast to ARIMA models, which assume that the series are generated from linear processes and as a result might be inappropriate for most real-world problems that are nonlinear [92].

Here we present a summary about the main concepts of both the techniques.

6.2.1 ARIMA models in a Nutshell

The *AutoRegressive Integrated Moving Average* (ARIMA) model, also known as the Box-Jenkins model, is widely regarded as the most efficient forecasting technique in Social Sciences and is used extensively for time series. It assumes that past patterns will similarly occur in the future, and therefore are predictable [93].

The procedure to generate ARIMA predictive models is usually divided in three parts, according to Box-Jenkins methodology: 1) data preparation; 2) model identification and estimation; 3) forecasting and model validation.

The first operation, i.e. **data preparation**, consists to plot data and examine for stationarity. Sometimes just looking at the representation of data in a graph is possible to identify a time series as non-stationary due to the fact that typical behaviors of non-stationary data are trends, cycles, "random walks" or combinations of the three [94]. In statistics, a non-stationary series has mean, variance and covariance that change over time in contrast with a stationary time series in which the data in the series do not depend on time or seasonality (and it is composed of "random variables"). A more formal definition is presented in [95], as:

Definition 6.1 *If y_t is a stationary time series, then for all s, the distribution of $(y_t, ..., y_{t+s})$ does not depend on t.*

In general, a stationary time series will have no predictable patterns in the long-term. In order to determine if a series is stationary or not-stationary, it is used the *Autocorrelation Function* (ACF). For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly and often the

value of the first lag is large and positive [95]. In addition to the ACF, it is commonly used the *Kwiatkowski-Phillips-Schmidt-Shin* (KPSS) test, whose the null hypothesis is that the data is stationary.

In order to proceed with the production of ARIMA model, the non-stationary time series needs to be transformed into stationary. The method to be applied depends from the type of non-stationary data. The operations often used are differencing and/or detrending. The first helps to stabilize the mean and can be defined as [95]:

Definition 6.2 *The differenced series is the change between each observation in the original series:* $y_t^{'} = y_t - y_{t-1}$

The differencing operation loses one observation each time the difference is taken since it is not possible to calculate $y_1^{'}$. The second,i.e. detrending a series, means to remove the trend components. There are many techniques to execute a detrending procedure with different effects [96]. The simplest way to detrend a time series is to fit a straight line through the data, using a least square procedure for instance. Examining the ACF we can know if the transformations have effects and the series is became stationary. If it is not, more than one differencing and/or detrending operations have to be done.

However, when the series is finally stationary, the second step, i.e. **model identification and estimation**, can be executed. In order to understand the values of the coefficients "p,d,q" of *ARIMA(p,d,q)* model, we have to have a look at the Autocorrelation Function and *Partial Autocorrelation Function* (PACF) of the stationary series. The "*d*" represents the number of times the data have been differenced to become to stationary. The "*p*" measures the order of the autoregressive component. Its value is obtained considering the PACF. The

number of lags that can be counted in the PACF before that the function quickly drops to near zero, is the number of significant partial correlation coefficients and also is the value of coefficient "p". The "q" measures the order of the moving average component and its value is given by the ACF in a similar manner of the coefficient "p". In fact, The number of lags that can be counted in the ACF before that the function quickly drops to near zero, is the number of significant autocorrelation coefficient and also is the value of coefficient "q". After having found the values of the three coefficients, we can estimate all the possible models combining the different information (for example by maintaining to zero the "q" and varying the other two coefficients, or, on the contrary, maintaining to zero the "p" and varying the other two coefficients, and so on). A very popular method to select the better model is to calculate the *Akaike Information Critera* (AIC). When comparing two or more models, the one with the lower AIC is generally the better.

The last operation is the **forecast** of a fixed number of values and the **validation** of the selected model (compared with the others). A conventional (graphical) technique for model validation is the *residual analysis*. Therefore, the autocorrelation plot of the forecast errors as well as the Box-Ljung test[1] are used to know the number of residuals that appear to be random (i.e. the number of residuals that are inside the confidence interval) and if the model provides an adequate fit to the data. Using the time plot and histogram of the forecast errors we can also determine the mean (that has to be zero) and the variance (that has to be constant).

[1] `http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4481.htm`

For a short tutorial on ARIMA models using "R" as programming language we refer to [97].

6.2.2 Artificial Neural Networks in a Nutshell

An *Artificial Neural Network* (ANN) is an interconnected group of nodes able to approximate a functional relationship between input and output variables in a certain domain of interest. Using the analogy with human neural networks, the nodes, that compose the ANN, are said neurons and the directed edges which communicate them are called synapses. The neurons are organised in layers which are usually fully connected by synapses. Each of the synapses is attached with a weight indicating the effect of the corresponding neuron in the whole model. The data pass through the neural network as signals. They are first processed by the so-called integration function combining all incoming signals (usually a summation); and second, they are processed by the so-called activation function obtaining the output of the neuron. A neural network with zero hidden layers is a linear expansion. The general model of a neural network usually has three layers (one is the hidden layer) with a single output. It is represented by Equation (6.1).

$$o(x) = f(w_0 + \sum_{j=1}^{q} w_j \cdot f(w_{0j} + \sum_{i=1}^{n} w_{ij} x_i)) \tag{6.1}$$

where w_{ij} (i=1,2,..,n; j=1,2,..q) and w_j are the weights of synapses; n is the number of input nodes and q is the number hidden nodes; f is the activation function. The definition of an ANN predictive model consists in determining the weights that provide the best fitting of the

real data through usage of learning algorithms [98].

6.3 Design Rationale for Smart Coffee Machines

The coffee machines that we have used can operate in two working-modes: *1)* `On-Off mode`, consisting in repetition of actions "switch on", "waiting for the coffee machine to heat", "prepare the coffee" and immediately "switch off"; *2)* `Standby mode`, in which the appliance is permanently ready to be used and no long warming time is needed when one wants to prepare a coffee. These working-modes generate different energy consumptions over the time due to the fact that the former needs big quantity of energy during the `START_TIME` (when the coffee machine is switched on); while the latter is characterized by periodical `PEAKs` (to maintain warm the machine's engine).

In [99], the authors have theoretically demonstrated that depending of the number of people that use the appliance in each hour (note that it is not the same the usage of a couple in their private setting as the usage of many workers in a workplace), it is convenient to introduce a new appliance's operating mode in order to save energy. Thus, one that adjusts the coffee-machine's operation depending of the usage it is subjected to. In rush hours (3 coffees or more per hour) the coffee-maker should remain on (`Standby mode`), while periods of lower use it has to be switched off (`On-Off mode`).

To enable Internet access to the old-fashioned coffee machines and to overcome their energy efficiency issue, we have attached to them a microcontroller, that we call *eco-adaptor*, which features Ethernet

interface and that is compatible with Arduino MEGA (`iBoard Pro`), and a non-invasive current sensor that generates data about the real energy consumption. These data are used in order to predict the appliances utilization. The selection of the model that better fit our data has been done comparing the forecast obtained with ARIMA models and ANNs. Both these methodologies assume that past patterns (number of coffee intakes per hour) will similarly occur in the future, and therefore are predictable.

6.3.1 Determining of the ARIMA Model for a Coffee Machine

The process to determine the ARIMA model for one coffee machine is described in [99]. For the sake of completeness, in this thesis we summarize the whole process. According to the steps that we have already described in Section 6.2.1, the first operation to find an ARIMA model is the data preparation. To check if stationarity is present on the coffee's time series, the KPSS is used. The result of the test is a p-value of 0.022 which is lower than 0.05, enough level of significance to reject the null hypothesis, thus the series is not stationary and needs at least a difference. The differencing operation is executed one time, therefore the coefficient "d" is 1. Using Akaike's Information Criterion, different ARIMA models have been compared and the final selected model has been ARIMA(3,1,1)(2,0,2). The latter set of this model refers to the ARIMA's seasonal part and this type of model is called SARIMA. It is classified as ARIMA(p,d,q)(P_1,D_1,Q_1) where: P, the number of seasonal autoregressive (SAR) terms, Q is the number of seasonal moving average (SMA) terms and D is the number of

seasonal differences.

To validate the ARIMA(3,1,1)(2,0,2), the authors of [99] have compared the forecast over the last 11 days of the coffee-machine's dataset with the real values observed during the forecast period. Through a graphical comparison, it is possible to see that the prediction is not far away to the real case and thus the model fits well to data. As we will describe more deep in Section 6.4, we would use ARIMA model to forecast coffee machine's next-week usage using data about 23 days earlier.

6.3.2 Determining of the ANN for a Coffee Machine

In order to determine the number of layers and nodes of as ANN that provide the best fitting of the real energy consumption of a coffee machine, we have to execute three phases: *1)* the training-set with the number of coffees prepared throughout each of the one-hour slots (starting at 7.00 a.m. and ending at 7.00 p.m) in 18 working days (weekends excluded). *2)* The test-set which is the same type of data corresponding to the 5 consecutive working days; *3)* 5 more working days of real data to test the models' performance.

Taking into account that one hidden layer is sufficient to model any piece-wise continuous function as stated by [100], we have chosen to use only one hidden layer in our model. Regarding the input layer, we have modelled the neural network with five input variables (i.e. five input neurons) that represent the number of coffees counted in the same hour slot along five consecutive days (from Monday to Friday). In the output layer we have decided to use only one neuron. It rep-

resents the number of predicted coffees that are prone to be prepared in the same hour-slot of those of the input nodes.

The idea of the proposed model is to feed it with a window of 5 working days of data (i.e. 12 vectors of 5 values each) to obtain the next-day forecast (i.e. a 12 values vector), slide the window forthward one day, including the previous predicted day, and perform again the prediction (see Figure 6.1). To clarify, if we feed the model with 5 values corresponding with the coffees prepared in one hour slot from Monday to Friday, we will expect to obtain the forecast number of coffees from the next Monday at the corresponding slot. If we do the same form Tuesday of the previous week until Monday of the current week, we will expect to obtain Tuesday's forecast number of coffees. If we slide again we will obtain the Wednesday forecast. So we repeat this process until the whole week is predicted.

ANN's Training phase: To select the most accurate model in order to compare it with ARIMA, we tested three different training configurations with respect to the number of hidden neurons: `5:2:1` (i.e. two hidden neurons); `5:5:1` (i.e. five hidden neurons); and `5:10:1` (i.e. ten hidden neurons). The training of the network is performed applying resilient backpropagation as a learning algorithm. The training-set is composed by the coffees counted in each hour slot during eighteen working days. Therefore, we train the ANN with 36 input vectors of five values each (15 days) and another 36 output vector of one value each (3 days). In this phase, the ANN calculates in an iterative manner the output for each given inputs, it measures the difference between the predicted and given output (i.e. the error) and it uses this error to modify the weights. All the three model configurations were respectively trained with 10 epochs by using the same

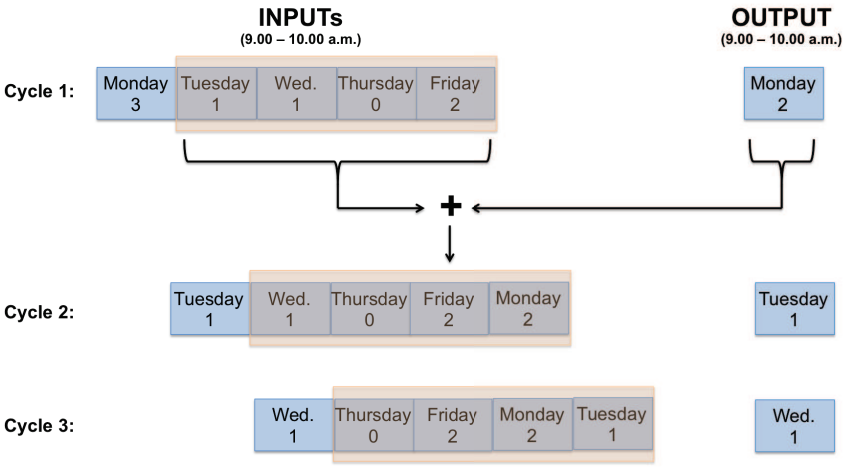


Figure 6.1: *The sliding window of 5 working days of data used to predict the next-day coffee intakes forecast. The process is repeated until the whole week is completed.*

training-set and learning algorithm.

Testing Phase and Model Selection: The test-set was composed by the coffees counted in one week. As the neural network model provides only one output, the execution of one test session is repeated by applying the sliding window approach of Figure 6.1.

To determine the best performing structure, we have calculated the different prediction errors for each of the models: root mean squared error (RMSE), mean absolute percentage error (MAPE), mean absolute error (MAE) and mean absolute scaled error (MASE). They are shown in Table 6.1. For each neural network configuration these metrics were computed using the predicted values and the remaining five days of real data of our dataset.

	ARIMA	**ANN_2**	**ANN_5**	**ANN_10**
RMSE	0.9708	1.6278	2.3614	2.0953
MAPE	0.8418	0.9413	1.0143	1.0447
MAE	1.0130	1.3142	1.8307	1.5076
MASE	0.3709	0.4812	0.6703	0.5520

Table 6.1: *Comparison of forecasting models. As it can be seen ARIMA outperforms the three configurations of ANN in all the error metrics.*

In all evaluated indexes, ANN_2 (two hidden nodes) has smaller values than the other ANNs. However, assuming a Gaussian error distribution for our predictions, we have used the root mean squared error (RMSE) to determine the best performing structure as was suggested by [101]. Table 6.1 shows that the RMSE value of testing error is 1.6278 for the configuration `5:2:1`, while it is `2.3614` and `2.0953` respectively for the neural networks' structures with five and ten hidden nodes. The smallest RMSE bares the best neural network configuration. Therefore, we can then conclude that for the time series data we had, the most accurate predictive model, when forecasting the weekly usage of a coffee machine, was a network with two nodes in the hidden layer.

6.3.3 Comparison of forecasting models: ARIMA vs Artificial Neural Network

To compare the models' autocorrelated structure performance, we have confronted the ARIMA error measurements with regard to the three ANN configurations previously discussed. According to Table 6.1, ARIMA has the lowest percentage error in each of the metrics calculated. In view of the results, we consider that ARIMA is the more suitable forecasting technique for the coffee-maker appliance's usage.

6.4 Implementation of ARIIMA Architecture

We have managed the forecasting of coffee machines' usage on a RESTful server. ARIIMA architecture delegates the computing intensive coffee consumption forecasting process to a Cloud-based service, thus reducing the microcontrollers' processing as much as possible. The RESTful server provides REST APIs to receive energy data from appliances and to generate the weekly forecast associated to each sustainable coffee machine. According to RESTful principles, it exposes stateless services that can be observed in Table 6.2.

For a better understanding of the interaction among the ARIIMA's components, we have divided the logic in two subsections: data storing and forecasting.

Table 6.2: *The three different Cloud-based services offered by the ecoserver.*

URI	HTTP Method	Description
/ecoserver/	POST	request to save new energy consumption events
/ecoserver/predictions/	GET	get the prediction for all coffee-makers
/ecoserver/predictions/[*deviceID*]	GET	get the prediction of the coffee-maker with parameter [*deviceID*]

6.4.1 Data Storing of Energy Consumption Events

It is necessary to keep track of the timestamps of every coffee intake to predict the working mode that the coffee machine should hold in each hour-slot along the working day. Each `iBoard Pro` features a RTC clock which is synced within a one second precision once a day by means of a pool of NTP servers depending of the country where the appliance is located. The retrieved time is stored within the RTC clock that has its own battery. If the NTP servers are not reacheable or the mains goes down, the Arduino board can always remains in synchronization by using its local time.
These data have to be stored to be used later for time series analysis. Taking into account that an Arduino board is a resource constrained device with reduced memory storage resources (Arduino MEGA have 4 Kb of EEPROM memory), it is not suitable to store large amounts of

data. Therefore, we have designed the architecture shown in Figure 6.2 to manage the data storing of consumed coffees.

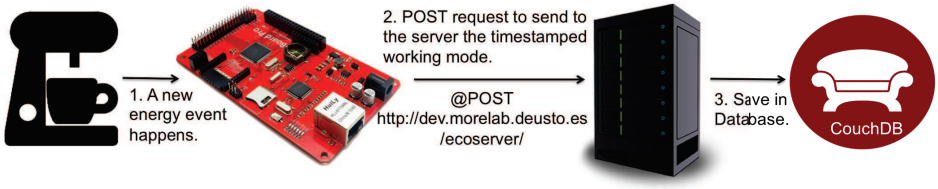


Figure 6.2: *ARIIMA architecture used for data storing of energy consumption events.*

Whenever a new energy event is detected, the Arduino board captures some information like its timestamp, the energy value consumed in Wh, the state in which the machine is set, and so forth (an example of the complete JSON object in which these data are structured can be observed in Listing 12). Then, the microcontroler sends the JSON to the ecoserver via a POST request. Finally, the server carries out the storing of the JSON object as a document inside a CouchDB NoSQL database. CouchDB is itself an HTTP server acepting CRUD operations over JSON documents.

6.4.2 Forecast of Coffee Machine's Next-Week Usage

The data collection process described in the previous subsection is used to predict the appliance's operating mode for the working days of the next-week (from Monday to Friday). The computational phase

Listing 12 Sample of a JSON energy event sent by Lab1's coffee-maker

```
{
    "deviceID": "Lab1",
    "device_type": "COFFEE-MAKER",
    "datetime": "2014-03-05T10:23:41Z",
    "time_secs": 43215,
    "consumption_type": "MAKING_COFFEE",
    "consumption_time_in_secs": 34,
    "energy_consumption_kWh": 16.8
}
```

is done by a Cloud-based web service and it is performed weekly (on Sundays). The sorted data-flow related with the time-series prediction can be observed in Figure 6.3.

The first task is to search for the number of coffees consumed in each of the hour-slots along the previous 23 working days (about 1 month of data). These data are used to infer the operating mode that the appliance should perform in each hour-slot for each working day on the new week. The information returned by the CouchDB database has to be transformed by the server in a dataset processable by an R script. To perform this conversion, every working-day is divided into slots of 12 hours (from 7am to 7pm) and the total amount of coffees made in each hour is calculated. Using this vector as input parameter, the ARIMA forecasting is executed (second step in Figure 6.3). The outcome prediction gives the number of coffees that are expected to be consumed in 5 days ahead for each hour slot. The prediction is furthermore valuated in different confidence bounds (80% and 95%). We have tested the different forecasting intervals and we selected 80% con-

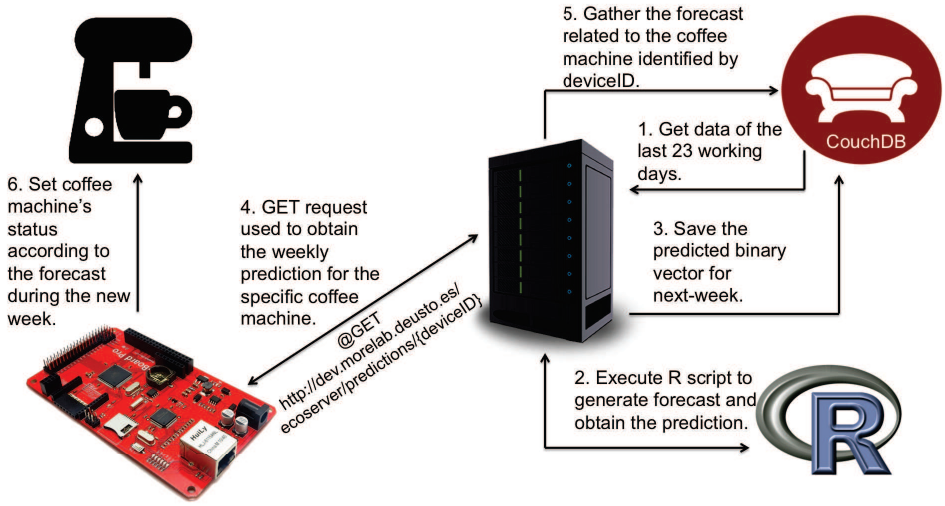


Figure 6.3: *ARIIMA architecture used to forecasting the coffee machine's next-week usage*

fident value as the more accurate. Its selection is discussed in the next Section. Since the Arduino board needs to know the operation mode to assume for each hour-slot, the forecast is translated to a binary vector. For this aim, we logically evaluated whether each predicted number of coffee intakes exceeds the threshold of three coffees. In that case we set the correspondent time slot to 1 (work in `Standby mode`), or contrary set it to 0 (work in `On-Off mode`). This binary vector is saved as JSON object inside the database with the format showed in Listing 13.

In the 4th step of Figure 6.3, each smart coffee-maker gathers weekly its prediction through a HTTP GET request sending its *deviceID*. When the server receives the request, it queries the database

Listing 13 Predicted 60 bits of binary data. Each bunch of 12 bit corresponds to each working-day (7am-7pm) of the new week.

```
{
    "deviceID": "Lab1",
    "prediction": "001110000110011100000 11..."
}
```

filtering by the *deviceID* received getting a JSON object (see Listing 13). The server sends the prediction vector back to the microcontroler and the Arduino board saves it in its EEPROM memory. In this way, everyday of the new week, it reads the sequence of 12 bits corresponding to that working day. In each hour-slot it applies the forecast automatically by using a relay leveraged within each device's On-Off button.

6.5 Evaluation and Results

To select the prediction which is closer to the real data, we compared the 5 days-ahead forecasted values with the real data observed during the forecasted period. The "training set" used to compute the prediction refers to 23 days from 21st May 2014 to 20th June 2014. Therefore, the empirical data are the next five working days (from 23rd June 2014 to 27th June 2014).
The ARIMA forecast issued by the R script give us different confident intervals (80% and 95%) for the exact number of coffees that are predicted in each slot, a.k.a. point forecast. Hence, we measured the binary closeness between the real data, and four different predicted data: *1)* point forecasted values; *2)* the values corresponding to the

upper bound with 80% rate of confident; *3)* those corresponding to the upper bound with 95% rate of confident; and *4)* the mean between the point forecasted values and the values corresponding to their upper bound with 95% rate of confident.

Table 6.3: *Distances calculated for 4 different coffee machines placed in four laboratories. Hamming distance gives an scalar measure (**h**, refers to heavy errors and **l** to light ones), while Jaccard and SM give their closeness value normalized between 0 and 1.*

Coffee Maker	**Distances**	**P.Forecast**	**80%**	**90%**	**Mean**
Lab1	Hamming	10 (**h**:10, **l**:0)	21 (**h**:3, **l**:18)	37 (**h**:1, **l**:36)	16 (**h**:7, **l**:9)
	S.Matching	0,16	0,35	0,61	0,26
	Jaccard	1,0	0,75	0,80	0,84
Lab2	Hamming	6 (**h**:6, **l**:0)	5 (**h**:5, **l**:0)	27 (**h**:0, **l**:27)	6 (h:6, **l**:0)
	S.Matching	0,1	0,08	0,45	0,1
	Jaccard	1,0	0,83	0,88	1,0
Lab3	Hamming	6 (**h**:5, **l**:1)	11 (**h**:0, **l**:11)	34 (**h**:0, **l**:34)	8 (**h**:1, **l**:7)
	S.Matching	0,1	0,18	0,56	0,13
	Jaccard	0,6	0,55	0,79	0,5
Lab4	Hamming	7 (**h**:6, **l**:1)	7 (**h**:5, **l**:2)	7 (**h**:5, **l**:2)	7 (**h**:5, **l**:2)
	S.Matching	0,11	0,11	0,11	0,11
	Jaccard	1,0	0,87	0,89	0,87

The prediction's vectors that we compared are those already transformed to binary data as shown in Listing 13. Therefore, we reviewed

a survey of binary distances and similarities [102] to select, among 76 methods proposed there, Hamming, Jaccard and Sokal-Michener (also called Simple Matching) as candidate distances. The Hamming distance gives us the exact number of binary mismatching, while Jaccard and Simple Matching give equal weight for matches and non-matches as expressed in[2]. To ease the selection of the most accurate measure, we distinguished two types of prediction's errors: *1)* heavy (false negative); and *2)* light (false positive). The former refers to the mismatching that occurs when the real coffee machine's operating mode was `Standby` but the value predicted was to set `On-Off mode`; The latter alludes to the error occurred when the real coffee machine's operating mode was `On-Off` but the value predicted was to set `Standby mode`. The energy wasted related to heavy errors is greater than light errors.

The closeness of the four different predictions intervals using each of the proposed distance-measures for four coffee machines are shown in Table 6.3. The analysis of the results shows that for Lab2 and Lab4's coffee machines the prediction closest to the real data is provided by the upper bound with 80% rate of confidence for all the evaluated distances. Lab1's coffee-maker shows that the 80% confidence bound is the most accurate when applying the Jaccard coefficient while using Simple Matching and Hamming distances is more accurate the point forecast. The point forecast presents a higher number of heavy errors than 80% (10 vs. 3). Thus, the amount of waste energy using the point forecast is greater. For Lab3, the average seems to be the most accurate prediction, however, the upper bound with 80% rate

[2] `http://tinyurl.com/murubf3`

of confidence presents closer results in every distance evaluated. In base of these considerations, we have decided to use always the upper bound with 80% of confidence level whenever we want to calculate the forecast of coffee machine's next-week usage for any coffee-maker. The Jaccard coefficient is selected as the more suitable for our scatter dataset since this distance does not take into account the number of zero-matching.

6.6 Adopted Strategies to Increase Eco-awareness of People

Energy conservation concerns not only to apply appropriate strategies directly on the devices, but also empowering people to operate them in an intelligent manner in order to minimise energy wastage. For example, one typical inefficient end-user behaviour is to leave these devices on when nobody is using them. Absent mindedness and comfort are the typical causes of that issue as was suggested in [103].

Chapman et al. in [104] explain the theory of "objects as mediators" according to which everyday technological products now have the potential to change our opinions and behaviours. The eco-aware coffee-maker has been created with this principle in mind since it has information that people do not have access to, i.e. the most-efficient operation mode in each moment. Therefore, it not only decides the working-mode to be used, but also collaborates with its users by providing appropriate and persuasive information. As consequence, users can even learn to reduce energy waste on other non eco-instrumented devices.

Our coffee-maker is able increase the awareness of people about energy consumption in different manners. We summarize them below.

Ambient Eco-feedback: The eco-aware coffee-maker provides three types of ambient feedback to its users. *1)* Informative visual feedback through an ambient light arch; *2)* subtle suggestions about how to operate the device appropriately through built-in markers in the machine itself; and *3)* a side-by-side bar chart to compare the energy wasted in two consecutive days. These types of ambient eco-feedback can be appreciated on Figure 6.4. *1)* The ambient light arch, which is placed close to the coffee-maker - left-side of Figure 6.4, provides information about the energy that is being wasted each day. Usually, the depicted waste corresponds with the time the device is on without being used. The arch starts the day being completely green, but it progressively turns into red as the wasted energy increases. *2)* The subtle suggestions are triggered when the eco-aware coffee-maker's schedule informs that throughout a whole hour-slot it is more appropriate to remain in standby mode rather to switch itself off (middle of Figure 6.4). The coffee-maker can detect when somebody is about to switch the device off after a coffee preparation - a proximity sensor pointing to the start-button has been attached to detect such action. When such detection occurs, the eco-aware appliance suggests the corresponding user to avoid it depicting the message: *'Please, leave me on.'* This message was before disguised in the outside of the water container and can only be seen under ultraviolet light. *2)* A physical display has been installed close to the coffee-maker that mimics a side-by-side bar chart (right-side of Figure 6.4). This informative eco-feedback showed to users their previous day's wasted energy (left side), whilst the energy being wasted on a given day was shown on

the right side. This design was inspired by the Energy Aware Clock of [105].

Figure 6.4: *Three different ambient eco-feedback. a) The light arch shows the increase of the wasted energy; b) the coffee-maker gives advices to people about how to behave efficiently; c) the physical display allows to compare the current wasted energy with the previous day.*

Teammates: According to [106], one strategy to transform the offices into more sustainable places, is to create green-teams among workers to better attain the sustainable goals. Therefore, the eco-aware coffee-maker is designed to collaborate with people to save energy. The goal is that workers create a sense of mutual-interdependence with such device which should be transformed into a promoter of these green-teams.

Social Networks: Internet-connected objects may contribute to reduce energy consumption by giving them an "active-voice" towards energy efficiency. Social Networks are expecting to become a way used by people to communicate with intelligent objects, as described in Section 1.2.1. They could also use for sustainability purposes. After

all, if we now want to reach thousands of users to raise their energy awareness, a simple "tweet" could achieve the task. On the basis of these considerations, our eco-aware coffee-maker is provided of a public Twitter profile where the breakdown of their daily energy consumption is reported. The expected goal it is that workers become followers of the appliance's Twitter profile in order to keep updated of its performance.

SEVEN

ENABLING USER ACCESS CONTROL TO HARDWARE-CONSTRAINED DEVICES IN THE INTERNET OF THINGS

7.1 Introduction

Security is an open issue of fundamental importance in the Web of Things that involves both Machine to Machine communications and user-object interactions. In previously Chapters 4 and 5, we have already discussed about the fact that having everyday objects connected to the network increases the risk of hacker attacks. Famous are the cases of a smart fridge of Samsung that has been violated during the DEFCON hackathon in Las Vegas, and of a baby monitor used as object to spy inside house. Since the Web of Things is a paradigm for interconnecting physical objects at application layer, security is prevalently seen from the point of view to restrict the access of objects only to authorized users/objects. Expanding the vision to the Internet of

Things, the problems concerning security are extended to both data encryption, and authentication and authorization.

Due to the large amount of objects that will populate the different scenarios of IoT (such as Smart City, Smart Environments, Smart Factories), these devices are usually designed to be small and inexpensive, resulting in limited processing capability. Additionally, these devices are often running 24 hours a day so low power consumption is required to enable sustainable computing. Hence, very lightweight security routines are needed.

In this Chapter we presented an access control solution for wireless environments in which users access services offered by hardware-constrained devices (e.g. wireless sensors). This solution provides efficient encryption, authentication and authorization on a per-user basis, i.e. a given user can access the services offered by a given sensor based on her identity. Furthermore, it needs no additional messages in the user-sensor communication. We have also considered differentiate groups of credentials in the authorization process, i.e., users can access the services offered by a group of sensors when they have the corresponding group credentials. The groups can be either hierarchical or non-hierarchical. In the latter, members in different privilege groups enjoy different non-hierarchical sets of services. In the former, members in higher privilege groups enjoy more services than lower level users. A security analysis is performed and experiments are conducted in the Arduino platform.

This Chapter is based on the works in [107], [108] and [109].

7.2 Background

In the following section we want to give some information about basic concepts in security field: Message Authentication Code and Key Derivation Function.

7.2.1 Message Authentication Code

A *Message Authentication Code* (MAC) is a small block of data (a bit string) used for the authentication of a digital message and to verify its integrity, generated according to a symmetric encryption mechanism. A typical process to generate and use a MAC during the communication between two parties is visible in Figure 7.1. Suppose Luigi wants to send a message (M) to Mario and both of them know the same private key (K). Therefore, Luigi uses an MAC algorithm that takes as inputs M and K and returns as output the corresponding MAC. He sends both the message and the MAC to Mario. The receiver, in order to check the authentication and integrity of the message, executes a fresh operation to create the MAC (using the received M and the shared secret K). if the comparison between the two MACs is successful, the message is authenticated and integrity is verified.

The property of a MAC algorithm is the irreversibility, i.e. it is not possible to determine the original message from the MAC. Of course, it is necessary that the two parties share the private key using a secure channel. In order to ensure also confidentiality, the MAC is usually used in conjunction with a symmetric encryption algorithm that encrypts the whole message. There are many implementations of MAC algorithm. One of the most popular is HMAC [110].

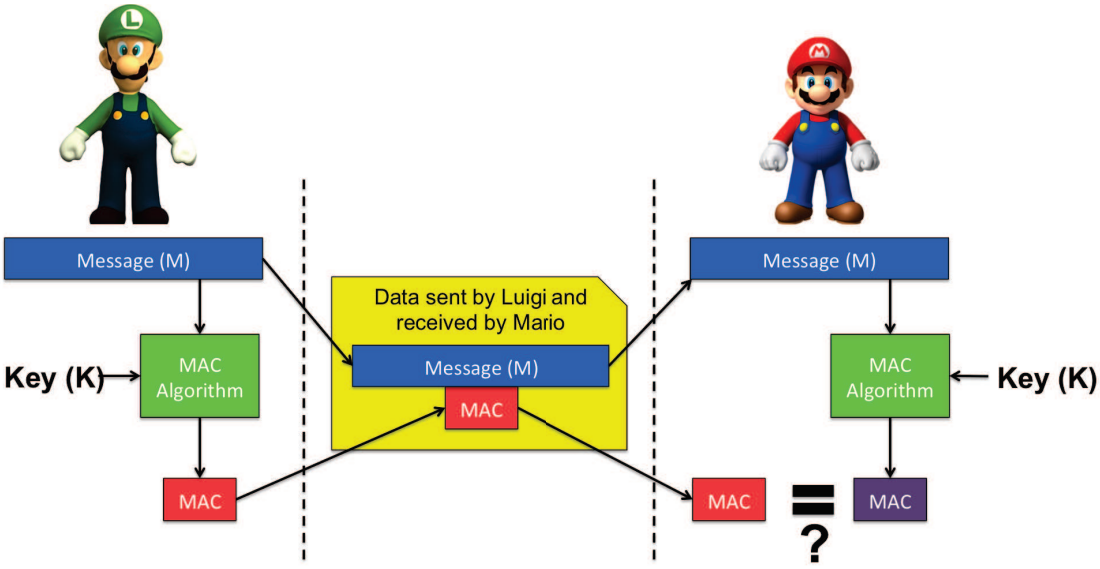


Figure 7.1: *Generation of a Message Authentication Code and communication flow. Luigi is the sender and Mario is the receiver. If the comparison between the received MAC and the generated MAC is successful, the message is authenticated and integrity is verified.*

7.2.2 Key Derivation Function

Key Derivation Function (KDF) is a family of algorithms that allow to create new symmetric cryptographic keys from a shared secret key and (optionally) a random public piece of information called *salt*. There are several reasons that can lead to use KDF: 1) the shared symmetric key does not meet specific security properties (e.g. to avoid weak keys); 2) more than one key is needed (e.g. when one key is used with a symmetric encryption algorithm to ensure confidentiality and an other key is used to produce a MAC in order to have authentication

and integrity of a massage). A generic Key Derivation Function can be expressed as:

$$DK = KDF(Key, Salt, Iterations)$$

where:

- DK is the derived key;
- Key is the original shared secret key;
- Salt represents data that can be send as plaintext
- Iterations is the number of iterations that KDF is used in order to derive one key.

Salt and Iterations are used to improve security. In particular, adding a random salt prevents a dictionary attack and repeating the process of KDF several times hinders brute force attacks. However, there are different recently standards that "suggest" how to implement KDF algorithms. In the next sections we will use the NIST SP 800-108 [111].

7.3 Scenario

The scenario we address in this work involves three kinds of players: sensors, Base Stations and user devices (e.g. smartphones), interacting together in a Smart Space.

Sensors are extremely constrained wireless devices, frequently battery-powered and with reduced computational capabilities, which provide users with services of any kind.

Base Stations are better equipped devices that handle groups of sensors for message routing purposes, data collection and also for key management in our case. They are assumed to have a more powerful hardware and a permanent (or at least much larger) power supply and large storage space.

Finally, users communicate with Base Stations and sensors through their powerful smart devices, such as mobile phones or tablets.

The key point here is that sensors need to perform access control on users but face several limitations: 1) they are not able to handle complex public-key authentication nor encryption routines and 2) they do not have enough memory space so as to keep large sets of user keys. In consideration of those constraints the proposed basic protocol provides an access control mechanism with symmetric encryption and authentication routines which minimizes storage requirements. On top of that, a groups extension is introduced in order to allow to manage users on a per-group basis: each user group has a different set of privileges, meaning that they can access different sets of the services provided by the sensors. Table 7.1 shows the notation used throughout the Chapter.

7.4 The basic protocol

The basic protocol (describes in deep in [107]) provides encryption and user access control to user $\leftrightarrow$ sensor one-to-one communications. The Base Station, a more powerful device, performs high-level authentication on the user (with authorization certificates based in public key cryptography, for example) and provides her with two symmetric keys

MS_S	Master secret for sensor S
$Kenc_{S,A}$, $Kauth_{S,A}$	Encryption and authentication keys for communication between sensor S and user A
$Kenc_{S,A}\{x,\ ctr\}$	x is encrypted in counter mode using key $Kenc_{S,A}$ and counter ctr
$MAC_{Kauth_{S,A}}(x)$	A MAC is done on x using $Kauth_{S,A}$
$KDF(x,\ \{a,\ b\})$	A Key Derivation Function is applied to master secret x using a as public salt and b as user-related information
$H(x)$	A hash function is applied to x
$x\|\|y$	Concatenation of x and y
ID_A	Identifier of user A
a	Random integer salt
$init\_time$, exp_time	Absolute initial and expiration time of a given key
MS_p	Master secret for privilege group p
$Kenc_{p,A}$, $Kauth_{p,A}$	Encryption and authentication keys between sensors offering services for group p and user A
ID_p	Identifier of privilege group p
$A \rightarrow *$	User A sends a message to any listening sensor
$S_p \rightarrow A$	One sensor giving services from privilege group p sends a message to A

Table 7.1: *Notation*

(for encryption and authentication, respectively) and parameters for their generation at the sensor. If those parameters are attached to the

first message of a conversation then the sensor can input them to a Key Derivation Function in order to obtain an identical pair of symmetric keys that make communication possible. Figure 7.2 depicts the message exchange in the protocol. Let us explain it with more details.

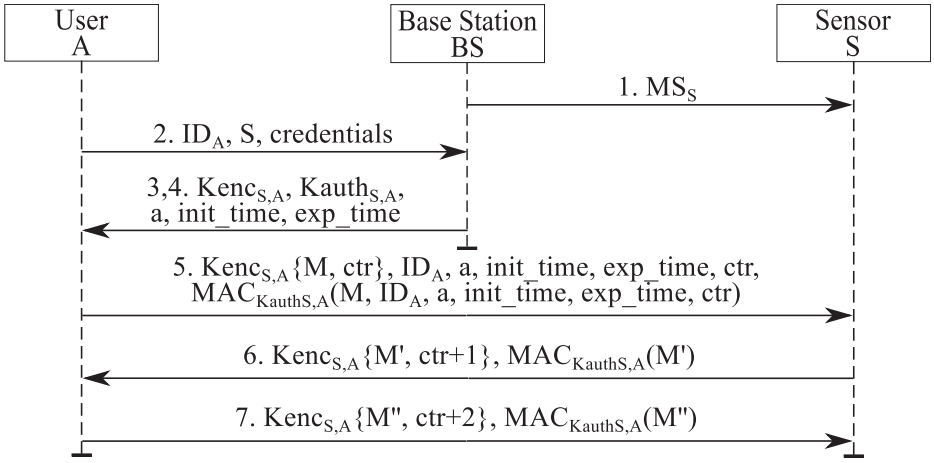


Figure 7.2: *Messages involved in the original protocol*

1. At the time of sensor deployment, the latter receives a master secret MS_S, which is *secretly shared* by the Base Station BS and the sensor S (see the end of this section for secret channels).

2. Upon arrival, user A sends her credentials (e.g. an authorization certificate) to BS so high-level access control can be performed, and the list of sensors she wants to communicate with (in Fig. 7.2 we only consider S). This step is run only at user arrival.

3. BS computes:

   (a) a, random integer salt

(b) ($init\_time$, exp_time), keying material validity interval

(c) $Kenc_{S,A}$, $Kauth_{S,A} = KDF(MS_S, \{a,\ ID_A||init\_time||exp\_time\})$

4. BS sends the information generated in the previous step to A under a secure channel (see the end of this section).

5. A encrypts her first message to S with $Kenc_{S,A}$ in counter mode (thus using a fresh counter ctr), attaches parameters ID_A, a, $init\_time$, exp_time, ctr in plain text and a MAC obtained with $Kauth_{S,A}$.

6. Upon reception of the message, S obtains the key pair $Kenc_{S,A}$, $Kauth_{S,A}$ by feeding the Key Derivation Function with the attached parameters; S can now decrypt the message. The reply is encrypted in counter mode with $Kenc_{S,A}$ and $ctr + 1$ and authenticated with a MAC using $Kauth_{S,A}$.

7. Any subsequent message is encrypted and authenticated with the same key pair after increasing the counter by one.

When the message exchange finishes the sensor may delete all information related to the user since it can be recomputed at the beginning of the next exchange, thus saving space at the sensor. Caching techniques can be applied though, as we will see in Section 7.6.

The sensor is sure of the authenticity of the user since the only way of knowing ($Kenc_{S,A}$, $Kauth_{S,A}$) is either knowing MS_S (which is kept secret) or obtaining it from the Base Station (which is actually the case). What is more, the MAC at the end of the message provides integrity assurance in addition to authentication. We refer the

reader to [107] for more considerations on security, efficiency, message overhead and storage of the basic protocol.

Regarding the trasmission of MS_S from BS to S in Step 1, a secure channel between them can be easily established by pre-installing a shared symmetric key in S before deployment. Having a secure channel allows also to renew MS_S to enhance security (note that changing MS_S will affect the keys generated in Steps 3c and 6 so active users should receive a new pair). For the secure channel between A and BS mentioned in Step 4 we assume both the user device and the base station can use public-key cryptography (e.g. SSL/TLS).

7.5 A groups extension

In this section we address a scenario with different groups of users, each group giving its members access privilege to a given set of services provided by sensors. Services provided by a sensor may (but not necessarily) belong to more than one group. The associated access control routines should not be intensive in terms of computations or message exchanges.

Let us assume that there are $l > 0$ groups. The main idea is that there exists a different master secret MS_p for every privilege group $p \in [1,\ l]$, hence sensors should only reply to service requests encrypted and/or authenticated with a key pair derived from the corresponding master secret. We propose two different approaches based on how services are arranged into groups. In Approach 1 privilege groups are not hierarchical, like in the case of employees that are allowed to enter different areas of a facility based on their activity (though some

services might be in more than one group). In Approach 2 privilege groups are hierarchical, hence a user with privilege level p should enjoy all privileges from groups $[1,\ p]$. An example of this scenario is a smart house with different privilege groups based on age: children would have access to certain services of the house, while parents should have full control of the house.

Due to lack of space we refer the reader to [108] for theoretical considerations about security and overhead in message length and storage.

7.5.1 Approach 1: non-hierarchical privilege groups

In this case, the Base Station generates l independent random master secrets $MS_1, \ldots, MS_l$ assuming there exist l different privilege groups. Sensors offering services from any privilege group p receive MS_p from the Base Station under a secure channel. In this scenario, users will typically belong to one group only, and sensors will provide services to one group as well. Figure 7.3(a) shows an example with three users and three sensors. However, if a sensor offers services to different privilege groups (or if a given service is included in more than one group), then the sensor should store each group's master secret. In a similar way, users assigned to more than one group (if that occurred) should receive a different pair of keys per group, and use the appropriate one to the requested service.

When user A arrives at the system the Base Station authenticates her and generates a different pair of symmetric keys $(Kenc_{p,A},\ Kauth_{p,A})$ for the privilege group A belongs to (group p in this case). These keys are generated by the BS and sensors assigned

to group p in the same way as in the basic protocol: the user identifier, a random salt a and a key validity interval ($init\_time$, exp_time) are fed to a Key Derivation Function along with the corresponding master secret as shown in Equation (7.2).

$$Kenc_{p,A},\ Kauth_{p,A} = KDF(MS_p, \{a,\ ID_A||init\_time||exp\_time\}) \tag{7.2}$$

These keys are sent to A by the BS under a secure channel (see Section 7.4). When user A wants to request a service from privilege group p she needs to encrypt and authenticate her message with that pair of keys like in the basic protocol (note that ID_p has been added).

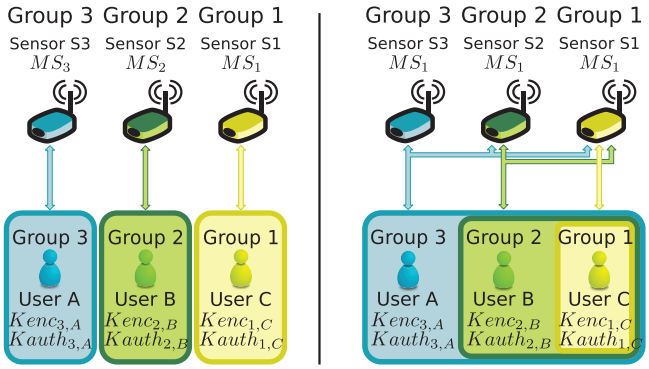
$$A \rightarrow * : [Kenc_{p,A}\{M,\ ctr\},\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr,\ MAC_{Kauth_{p,A}}(M,\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr)]$$

Any nearby sensor providing services from group p (let us name it S_p) can now reply to A after deriving the appropriate pair of keys from the received information and MS_p. The counter is explicitly stated on plain text so synchronization is not lost due to an arbitrary sequence of messages if more than one sensor is involved in the conversation.

$$S_p \rightarrow A : [Kenc_{p,A}\{M',\ ctr+1\},\ ctr+1,\ MAC_{Kauth_{p,A}}(M',\ ctr+1)]$$

7.5.2 Approach 2: hierarchical privilege groups

In this case, services are arranged in hierarchical groups: users assigned to privilege group p should be granted access to all services



(a) Approach 1 (b) Approach 2

Figure 7.3: *Examples of the two approaches with three groups.*

in groups $[1,\ p]$. Here every sensor in the system receives the lowest group's level master secret MS_1 from the BS. The rest are obtained by hashing the immediately lower master secret, i.e. $MS_p = H(MS_{p-1})$. This requires lower permanent storage requirements at the cost of a slightly higher computational demand and more security risks as every sensor can obtain the master secret for any privilege level. Figure 7.3(b) shows an example with three users and three sensors.

Thanks to this modification, user devices need to store only one pair of keys, that of the highest privilege level they are granted. For example, a user A in group 3 will only receive $(Kenc_{3,A},\ Kauth_{3,A})$ from the Base Station. However the use of this key pair is enough for being granted access to any service in groups 1 to 3.

The verification of user credentials at the sensor side goes as follows. After receiving a message encrypted and authenticated with $(Kenc_{p,A},\ Kauth_{p,A})$ (see Equation (7.3)) the sensor derives $MS_p =$

$H(...H(MS_1))$. From MS_p and user-bound parameters the sensor obtains $(Kenc_{p,A},\ Kauth_{p,A})$ as in Equation (7.2). Communications can now be established as in Equation (7.4).

7.5.3 Combining hierarchical authentication with individual privacy

The basic protocol provides one-to-one authentication and encryption between a user and a sensor. On the other hand, approaches 1 and 2 allow to perform one-to-many authentication and encryption: all sensors holding the affected master secret will be able to authenticate the user and decrypt the conversation. Next, we consider the possibility of having services that demand one-to-one private communications and group-based authorization at the same time. For achieving this we base on Approach 1, however the extension to Approach 2 is straightforward.

In this case sensor S is assigned by the Base Station an individual master secret MS_S (as in the basic protocol) and one master secret MS_p for each privilege group p the sensor provides services from (in Approach 2 the sensor would be assigned MS_1 and would derive the rest by hashing).

User A is assigned a pair of keys for individual communication with S, i.e. $(Kenc_{S,A},\ Kauth_{S,A})$, and a pair of keys $(Kenc_{p,A},\ Kauth_{p,A})$ for the privilege group she is entitled to, say p. Like before, these keys are generated for A by the Base Station by feeding MS_p and user-related parameters $ID_A,\ a,\ init\_time,\ exp\_time$ to a Key Derivation Function.

Now, when A wants to communicate only with S while proving

her authorization level, she encrypts her messages with $Kenc_{S,A}$ and computes the corresponding MAC with $Kauth_{p,A}$ as in Equation (7.5). S replies using the same pair of keys and incrementing the counter, which needs not to be included on plain text given that the message exchange takes place between two players only:

$$\begin{aligned} A \rightarrow S \ &: \ [Kenc_{S,A}\{M,\ ctr\},\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr, \\ &\quad MAC_{Kauth_{p,A}}(M,\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr)] \\ S \rightarrow A \ &: \ [Kenc_{S,A}\{M,\ ctr+1\}, MAC_{Kauth_{p,A}}(M,\ ctr+1)] \end{aligned}$$

7.6 Performance Evaluation

To evaluate our solution we measure two different aspects: how it behaves in a embedded platform and its security strengths and weaknesses.

In order to measure its performance we have implemented and tested step 3c of the protocol in Section 7.4, which is the core of our proposal and its most resource-demanding stage. The tests were run in the Arduino platform [112] and the code is publicly available [1].

We run the experiments in two Arduino boards (Uno and Mega) to analyse whether they behave differently. Table 7.2 shows their technical specifications.

Furthermore, for the derivation function we used keys of different lengths: 128 bits, 256 bits and 512 bits. To derive the keys, we

[1] `http://github.com/dventura3/Nist`
[2] `http://arduino.cc/en/Main/arduinoBoardUno`
[3] `http://arduino.cc/en/Main/arduinoBoardMega`

Table 7.2: *Technical characteristics of the assessed Arduino boards*

Arduino Board	Microcontroller	Flash Memory	SRAM	EEPROM
Uno [2]	16 MHz ATmega328	32 kB	2 kB	1 kB
Mega2560 [3]	16 MHz ATmega2560	128kB	8 kB	4kB

used the NIST Key Derivation Function in Counter Mode [111] together with the HMAC-SHA-1 and HMAC-SHA-256 authentication functions [110]. SHA-1 is used as a baseline, although its use is currently not advised anymore. SHA-512 implementation is not currently implemented in the Cryptosuite library used [4]. Therefore, to obtain the key 512 bits key we call to SHA-256 twice with two different subkeys.

To know the impact of our solution in Arduino, we derived keys of 128, 256 and 512 bits 100 times in each of the Arduino models tested. The average time needed to derive each key is 28.67, 99.84 and 288.15 ms respectively with standard deviations of less than 1 ms. The measures showed no difference between models.

The sensor, i.e., the Arduino board, will need to generate two keys: one for authentication and another for encryption. Considering, the 288 ms needed to generate a 512 bits key, it will require 576 ms in total. If we also take into account the additional tasks to be performed afterwards (decryption, encryption and MAC), the sensor might take too much time to answer a request. To mitigate this effect, the sensor can cache the pair of keys for each user/group at the cost of using

[4] `https://github.com/dventura3/Cryptosuite`

more memory.

To analyse how our solution would affect the memory consumed by an Arduino board, we first checked how deriving keys affects their free memory. Each Arduino board allocated 38, 54 and 106 bytes during the generation of each key of 128, 256 and 512 bits respectively.

Since the most power-demanding operation in a sensor is airing messages through its antenna [113, 114], we only send the data needed to create a key once (see Figure 7.2). This reduces the message length of the subsequent requests, but forces the sensor to store ID_p, a, $init\_time$, exp_time and the updated ctr to regenerate each key. Arduino needs 16 bytes to store each of these set of fields. Figure 7.4 represents this case with the blue bars. Considering the most limited board (i.e., Uno), it is able to manage the data of 26, 24, 20 users or groups for each key length in SRAM with the current program. As the program is loaded also in SRAM, a more complex program will reduce this number. However, if we ignore all the MS_S stored and other additional data stored by the program, the EEPROM could store enough information for other 64 additional users or groups.

The available EEPROM will be additionally reduced in approaches 1 and 2 because they require the sensor to store permanently a pair of keys for the group. Approach 1 requires the sensor to store a master secret for every privilege group it might be assigned to. In Approach 2, the sensor can decide (a) to permanently store a single master secret or (b) to store all master secrets once derived. *Case a* saves computations at the cost of the space, while *Case b* the saves memory increasing the computation.

If the sensor caches the keys as suggested before, it will keep in

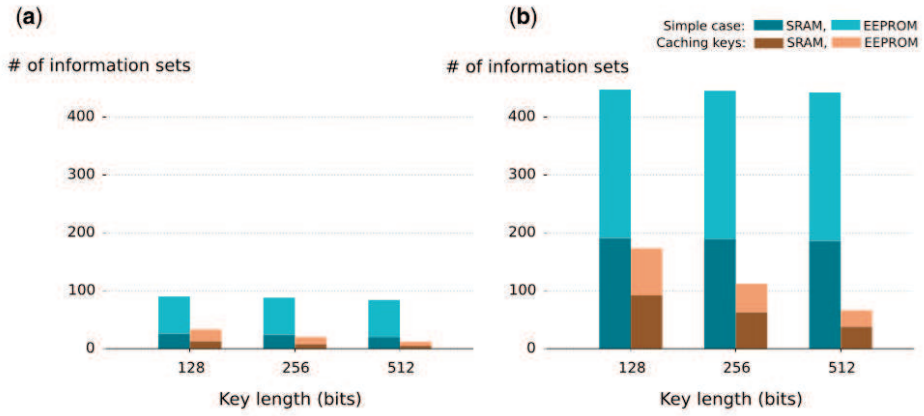


Figure 7.4: *A sensor needs to keep a set of fields about each user or group it communicates with. The chart shows how many sets each board can manage. In the most simple case it derives the keys needed after each request. In the other, it caches the keys.*

memory $Kenc_{S,A}$, $Kauth_{S,A}$, ID_a exp_time, and the updated ctr. This will require it to store 50, 82 and 146 bytes for 128, 256 and 512 bits. Figure 7.4 represents this case with the brown bars. Considering the memory already consumed by the program, Arduino Uno will be able to keep the data for 13, 8 and 5 users in SRAM for each key length. Using the EEPROM, it could manage the information of 20, 12 and 7 additional users.

To summarize, normal operation of the device generating two keys at each request is slow for the worst case (512 bits), but acceptable for the 128 and 256 bits cases. The storage of the keys in cache reduce the response time but it also reduces the maximum number of users that can use the sensor due to its memory limitations. The minimum of the maximum number of concurrent users is five, which is determined

taking into account the most limited board and the most demanding memory instances of our experiments, i.e. memory needed by the program, the 512 key lengths and the use of cache. The settings of the systems will depend on the characteristics of the final devices and the number of concurrent users the application demands.

EIGHT

CONCLUSIONS

The goal of this thesis has been to investigate how the capabilities of everyday objects should be augmented in order to enable the development of user-centric services in order to make user-object interactions more natural, meet users' needs and influence people's behavior in the context of Web of Things. We enabled machines to provide descriptions of their REST APIs in machine-understandable format and to communicate using semantic data. After an overview about the state of art on syntactic and semantic methods to describe REST services, we decided to use a solution based on standard Web technologies. Therefore, we proposed *RESTdesc*, as method to describe hypermedia REST APIs based on Notation3 rules, and *JSON-LD*, as data exchange format.

Adding a semantic reasoner on everyday objects and using REST-desc and JSON-LD, we demonstrated, in Chapter 3, that is possible to generate and execute *plans* (composition of web APIs exposed both objects and Web services) that satisfy goals. We have evaluated this

strategy through the implementation of a use case about a smart and autonomous client able to monitor the environmental conditions of plants in a greenhouse using boards with sensing and actuating capabilities and a web server to know weather forecast. This client has not any pre-knowledge about the APIs provided by the boards and the web server, but knows only the tasks that has to perform.

In Chapter 4, we have addressed the problem of interaction between users and heterogeneous objects, e.g. that have different features and various control procedures difficult to use. The idea was to provide to users a mobile application that, through the recognition of a QR-Code associated with each object, was able to generate on the fly a graphical interface that fits the type of object framed by the camera and allows him to control the object. For this purpose we extended webinos platform (an European project), described the capabilities offered by the object using RESTdesc, and used these descriptions to dynamically generate the graphical interface. This study has the intention to leverage on the user-experience of mobile applications. In fact, users are able to use web applications immediately after the installation, i.e. there is no a user manual that explains how an app works but user deduces how using it helped by a standard GUI (containing menu, buttons, scrolls, etc.).

Exploiting the ability of reasoning and plans' production of everyday objects presented in Chapter 3, we proposed Smart EDIFICE, a platform able to achieve users' needs and requests, described in Chapter 5. The basic idea is that user does not explicitly define the objects to be used to perform tasks, but is limited to express the objectives to be achieved. Hence, the platform is able to understand the user's request (expressed by voice, text or web app), determine how and when

accomplish the required goal and coordinate the objects around him. If more plans can achieve a goal, the choice of which plan using is based on user's preferences and feedback. Therefore a same goal expressed by different people could be executed by different personalized planes.

Making objects smart includes that the objects must be able to elaborate the context and make decisions. The analyzed information can also be reported to users in order to influence their behavior or choices. We studied, in Chapter 6, a way to increase people's motivation about energy saving through the reception of eco-feedback sent directly by everyday appliances. We augmented fashioned not eco-friendly devices with an eco-adaptor, composed by an embedded Internet-connected board and a current sensor. In particular we applied this system to coffee machines placed in shared spaces (offices, schools, etc.). Each coffee-maker works in two modes, on/off and standby, that generate different energy consumptions. We determined that if the number of coffees intakes per hour is more than three, it is better to use the standby mode than on/off mode in order to save energy. The coffee-makers report their usage patterns to a Cloud-server where the data are processed by time-series algorithms in order to obtain the appliances' next-week usage forecast. To overcome the lack of energy awareness of people, each coffee machine triggers timely persuasive interactions about predicted consumption so that users can operate the appliance as efficiently, energy-wise, as possible.

Finally, in Chapter 7, we considered a IoT scenario where hardware-constrained devices have to communicate with users/objects in secure way. We proposed a lightweight protocol that, based on using Key Derivation Function, ensures encryption, authentication and

authorization. We also considered differentiate groups (hierarchical or non-hierarchical) of credentials in the authorization process. We evaluated the proposed solution implementing the protocol for Arduino boards and measuring memory and time spent for the execution.

Writing this thesis, we have not got the claim to propose definitive solutions to topics that today are under study and standardization. The aim of this thesis is to address some open issues about machine to machine interactions and exploit the smartness of everyday objects for improving users' life in their environments (especially their houses). We hope that reading this thesis may give rise to constructive criticism and valuable insights to undertake new and promising research activities.

8.1 Future work

Before finishing, we propose future work that is based on the research topics discussed in this thesis. Smart EDIFICE architecture is based on using a cloud-block, the Task Coordinator, as orchestrator for the execution of plans. We would lighten its tasks by enabling the Smart Objects to self-organized during the execution phase. In other words, we would convert the platform from centralised to distributed system allowing that directly the objects could manage the sequence of operations representing a plan. The consequence would be to increase the reliability of the system. A framework implemented to allow objects in proximity to communicate each other is *AllJoyn* [83]. Therefore a solution under study is to include AllJoyn in Smart EDIFICE and implement policies to let objects to self-coordinate.

Regarding Smart EDIFICE, an other improvement could be to analyze users' goals and tried to know habits and behaviours of inhabitants of the Smart Space so that to anticipate or recommend some tasks. For example, examining plans expressed by different users that live in the same environment within a year, the platform could understand that the water heater has to be switched on only from the 8.00 to 9.00 a.m. at summer.

Having objects able to semantically describe their REST APIs, to communicate and understand each other, and to aggregate services in order to satisfy goals and create physical mashups, can lead towards many new scenarios. First, we are studying a *Location Based Services* (LBS) system dependent by user's profile and interests. The core idea is to extract user's interests using social networks and convert them in a semantic format (e.g. *Open Graph Protocol*[1] used by Facebook is based on RDFa). An intelligent agent could match user's information with semantic descriptions of Web APIs exposed by objects or web services placed near user, and recommend a subset of these services. For example, a user likes watching film to the cinema. When he is near a cinema, could discover a service to book tickets online for that cinema.

Second, we are planning to extend the vision from Smart Spaces to Smart City. In this wider context, a new topic is the so-call crowd-sensing tasks. Crowd-sensing is a capability by which users can create tasks and recruit devices of other users of the same Smart City to provide sensor data to be used towards a specific goal. For instance, a user would like to know the meteo in a specific area of his city. This

[1] `http://opengraphprotocol.org`

task could be executed using the humidity sensors of users' smartphones that are in that specific location. Therefore, a platform has to be able to match the user's request with the capabilities of the devices considering a specific context.

We have already discussed in Chapter 4 about future work concerning the application that simplifies user-object interaction, stating to want to use the framework *Vuforia* for Augmented Reality. Finally, we would extend the use case to reduce energy consumption regarding the coffee machines to other type of appliances. This means that we need to generate different time series models to predict new usage patterns.

8.2 Relevant Publications

This thesis is based on several articles published in international workshops, conferences, and journals. These articles are listed below:

1. Vincenzo Catania, **Daniela Ventura**, "An approach for Monitoring and Smart Planning of Urban Solid Waste Management Using Smart-M3 Platform", *In Proceedings of the 15th Conference of Open Innovations Association*, Saint Petersburg, Russia, 2014.

2. Vincenzo Catania, Giuseppe La Torre, **Daniela Ventura**, "Controlling Smart Objects from Web Application using the Webinos Platform", *In Proceedings of the European Conference on Smart Objects, Systems and Technologies*, Dortmund, Germany, 2014.

3. **Daniela Ventura**, Diego Casado-Mansilla, Juan López-de-Armentia, Pablo Garaizar, Diego López-de-Ipiña, Vincenzo Catania, "ARIIMA: A Real IoT Implementation of a Machine-learning Architecture for reducing energy consumption" *In Proceedings of the 8th International Conference on Ubiquitous Computing and Ambient Intelligence*, Belfast, UK, 2014.

4. Diego Casado-Mansilla, Juan López-de-Armentia, **Daniela Ventura**, Pablo Garaizar, Diego López-de-Ipiña, "Embedding Intelligent Eco-aware Systems within Everyday Things to Increase People's Energy Awareness", *Soft Computing Journal*, 2015.

5. **Daniela Ventura**, Aitor Gómez-Goiri, Vincenzo Catania, Diego López-de-Ipiña, Juan Alvaro Muñoz Naranjo, Leocadio González Casado, "Security analysis and resource requirements of group-oriented user access control for hardware-constrained wireless network services", *Logic Journal of the IGPL*, 2015.

6. **Daniela Ventura**, Salvatore Monteleone, Giuseppe La Torre, Gaetano Carmelo La Delfa, Vincenzo Catania, "Smart EDIFICE - Smart EveryDay Interoperating Future devICEs", *In Proceedings of the International Conference on Collaboration Technologies and Systems*, Atlanta, GA, USA, 2015.

7. **Daniela Ventura**, Ruben Verborgh, Vincenzo Catania, Erik Mannens, "Autonomous Composition and Execution of REST APIs for Smart Sensors", *In Proceedings of the 14th International Semantic Web Conference (ISWC2015)*, Bethlehem, PA, USA, 2015.

BIBLIOGRAPHY

[1] S. Gunter, "The third wave of computing." http://www.olafurandri.com/nyti/papers2008/Ubiquitous_computing.pdf, 2008. Accessed: 17-11-2015.

[2] National-Post, "In future, everything will be a computer." http://www.canada.com/nationalpost/story.html?id=42620fdf-6339-40c6-9775-dcad5d623f51&k=8721, 2007. Accessed: 17-11-2015.

[3] M. Weiser, "Human-computer interaction," ch. The Computer for the 21st Century, pp. 933–940, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995.

[4] D. J. Cook and S. K. Das, "How smart are our environments? an updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007.

[5] G. David, "Smart spaces and augmented reality." https://www.encorewiki.org/display/KMDI2003/Smart+Spaces+and+Augmented+Reality, 2013. Accessed: 19-11-2015.

[6] M. Beigl, H.-W. Gellersen, and A. Schmidt, "Mediacups: Experience with design and use of computer-augmented everyday artefacts," *Comput. Netw.*, vol. 35, no. 4, pp. 401–409, 2001.

[7] G. Kortuem, D. Alford, L. Ball, J. Busby, N. Davies, C. Efstratiou, J. Finney, M. White, and K. Kinder, "Sensor networks or smart artifacts? an exploration of organizational issues of an industrial health and safety monitoring system," in *UbiComp 2007: Ubiquitous Computing* (J. Krumm, G. Abowd, A. Seneviratne, and T. Strang, eds.), vol. 4717 of *Lecture Notes in Computer Science*, pp. 465–482, Springer Berlin Heidelberg, 2007.

[8] F. Mattern, "From smart devices to smart everyday objects (extended abstract)," in *Proceedings of sOc'2003 (Smart Objects Conference)*, (Grenoble, France), pp. 15–16, 2003.

[9] S. Mathew, Y. Atif, Q. Sheng, and Z. Maamar, "Web of things: Description, discovery and integration," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pp. 9–15, 2011.

[10] Los-Angeles-Times, "60% of world's population still won't have internet by the end of 2014." `http://www.latimes.com/business/technology/la-fi-tn-60-world-population-3-billion-internet-2014-20140507-story.html`, 2014. Accessed: 20-11-2015.

[11] Ericsson-User-Experience-Lab, "A social web of things." http://www.ericsson.com/uxblog/2012/04/a-social-web-of-things/, 2012. Accessed: 21-11-2015.

[12] A. Ciortea, O. Boissier, A. Zimmermann, and A. M. Florea, "Reconsidering the social web of things: Position paper," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, (New York, NY, USA), pp. 1535–1544, ACM, 2013.

[13] 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.

[14] E. Savitz, "Welcome to the api economy." http://www.forbes.com/sites/ciocentral/2012/08/29/welcome-to-the-api-economy/, 2012. Accessed: 20-11-2015.

[15] M. Stuart, "Rise of the api economy." http://amplifiedcontentmarketing.com/rise-of-the-api-economy/, 2014. Accessed: 20-11-2015.

[16] A. Schmidt, "Implicit human computer interaction through context," *Personal Technologies*, vol. 4, no. 2-3, pp. 191–199, 2000.

[17] B. Shneiderman and P. Maes, "Direct manipulation vs. interface agents," *interactions*, vol. 4, no. 6, pp. 42–61, 1997.

[18] L. De Russis, *Interacting with Smart Environments: Users, Interfaces, and Devices*. PhD thesis, Politecnico di Torino, Italy, 2014.

[19] R. F. I. L. M. X. C. T. Berners-Lee, MIT/LCS, "Uniform resource identifiers (uri): Generic syntax." `https://tools.ietf.org/html/rfc2396`, 1998. Accessed: 04-12-2015.

[20] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[21] W3C, "Web services description language (wsdl) 1.1." `http://www.w3.org/TR/wsdl`, 2001. Accessed: 05-12-2015.

[22] W3C, "W3c xml schema definition language (xsd) 1.1 part 1: Structures." `http://www.w3.org/TR/xmlschema11-1/`, 2012. Accessed: 05-12-2015.

[23] W3C, "W3c xml schema definition language (xsd) 1.1 part 2: Datatypes." `http://www.w3.org/TR/xmlschema11-2/`, 2012. Accessed: 05-12-2015.

[24] P. S. L. M. S. B. Yves Raimond, Tom Scott and F. McNamara, "Case study: Use of semantic web technologies on the bbc web sites." `http://www.w3.org/2001/sw/sweo/public/UseCases/BBC/`, 2010. Accessed: 05-12-2015.

[25] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, pp. 34–43, May 2001.

[26] W3C, "Rdf 1.1 concepts and abstract syntax." http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/, 2014. Accessed: 06-12-2015.

[27] W3C, "Rdf 1.1 xml syntax." http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/, 2014. Accessed: 06-12-2015.

[28] W3C, "Rdf 1.1 turtle." http://www.w3.org/TR/2014/REC-turtle-20140225/, 2014. Accessed: 06-12-2015.

[29] W3C, "Rdf schema 1.1." http://www.w3.org/TR/rdf-schema/, 2014. Accessed: 06-12-2015.

[30] W3C, "Owl 2 web ontology language document overview (second edition)." http://www.w3.org/TR/owl2-overview/, 2014. Accessed: 06-12-2015.

[31] Complexible-Inc, "Pellet: An open source owl dl reasoner for java." https://github.com/complexible/pellet, 2015. Accessed: 06-12-2015.

[32] Cwm, "Cwm." http://www.w3.org/2000/10/swap/doc/cwm.html, note = Accessed: 06-12-2015, 2004.

[33] J. D. Roo, "Eye reasoning." http://eulersharp.sourceforge.net/. Accessed: 06-12-2015.

[34] W3C, "Sparql 1.1 overview." http://www.w3.org/TR/sparql11-overview/, 2013. Accessed: 06-12-2015.

[35] w. p. Ericsson, "Media Vision 2020: A Vision of the Television Future." `http://www.csimagazine.com/csi/whitepapers/MediaVision-Brochure-RevA.pdf`. (2014).

[36] MachinaResearch, "M2M connections to hit 18 billion in 2022, generating USD1.3 trillion revenue." `https://machinaresearch.com/static/media/uploads/machina_research_press_release_-_m2m_global_forecast_&_analysis_2012-22_dec13.pdf`. (2013).

[37] R. Verborgh, T. Steiner, D. Van Deursen, S. Coppens, J. Gabarró Vallés, and R. Van de Walle, "Functional descriptions as the bridge between hypermedia APIs and the Semantic Web," in *Proceedings of the Third International Workshop on RESTful Design*, pp. 33–40, Apr. 2012.

[38] R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. Gabarró Vallés, "Capturing the functionality of Web services with functional descriptions," *Multimedia Tools and Applications*, vol. 64, pp. 365–387, May 2013.

[39] M. Lanthaler and C. Gütl, "On Using JSON-LD to Create Evolvable RESTful Services," in *Proceedings of the Third International Workshop on RESTful Design*, 2012.

[40] D. Ventura, R. Verborgh, V. Catania, and E. Mannens, "Autonomous composition and execution of REST APIs for smart sensors," in *Joint Proceedings of the 1st Joint International Workshop on Semantic Sensor Networks and Terra Cognita and*

the 4th International Workshop on Ordering and Reasoning, Oct. 2015.

[41] W3C, "Web services description language (wsdl) version 2.0." `http://www.w3.org/TR/wsdl20-primer/`, 2007. Accessed: 29-11-2015.

[42] W3C, "Web application description language." `http://www.w3.org/Submission/wadl/`, 2009. Accessed: 29-11-2015.

[43] J. Lathem, K. Gomadam, and A. Sheth, "SA-REST and (S)mashups : Adding Semantics to RESTful Services," in *International Conference on Semantic Computing*, 2007.

[44] J. Kopecky, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services," in *Web Intelligence and Intelligent Agent Technology*, 2008.

[45] R. Verborgh, A. Harth, M. Maleshkova, S. Stadtmüller, T. Steiner, M. Taheriyan, and R. Van de Walle, "Survey of Semantic Description of REST APIs," in *REST: Advanced Research Topics and Practical Applications*, pp. 69–89, Springer New York, 2014.

[46] Swagger, "Swagger — the world's most popular framework for apis." `http://swagger.io`, 2015. Accessed: 29-11-2015.

[47] W3C, "Owl-s: Semantic markup for web services." `http://www.w3.org/Submission/OWL-S/`, 2004. Accessed: 29-11-2015.

[48] W3C, "Web service modeling ontology (wsmo)." `http://www.w3.org/Submission/WSMO/`, 2005. Accessed: 29-11-2015.

[49] W3C, "Wsmo-lite: Lightweight semantic descriptions for services on the web." http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/, 2010. Accessed: 29-11-2015.

[50] M. Lanthaler and C. Guetl, "Hydra: A Vocabulary for Hypermedia-Driven Web APIs," in *Linked Data on the Web Workshop*, 2013.

[51] A. Sen, "From search to application building-yahoo pipes for novice programmers," vol. 13, pp. 199–208, 2012.

[52] "IFTTT: Connect the apps you love." https://ifttt.com/.

[53] "Atooma: A touch of magic." http://www.atooma.com/.

[54] K. Kenda, C. Fortuna, A. Moraru, D. Mladenif́a, B. Fortuna, and M. Grobelnik, "Mashups for the web of things," in *Semantic Mashups*, pp. 145–169, 2013.

[55] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, C. Pautasso, T. Heinis, R. Grønmo, H. Hoff, A.-J. Berre, M. Glittum, and S. Topouzidou, "Developing scientific workflows from heterogeneous services," *SIGMOD Rec.*, vol. 35, 2006.

[56] J. Bellido, R. Alarcón, and C. Pautasso, "Control-flow patterns for decentralized restful service composition," *ACM Transactions on the Web (TWEB)*, 2013.

[57] C. Pautasso, "{RESTful} web service composition with {BPEL} for {REST}," *Data and Knowledge Engineering*, 2009.

[58] "ETCI-m2m." http://www.etsi.org/technologies-clusters/technologies/m2m.

[59] "Open mobile alliance." http://openmobilealliance.org.

[60] R. Verborgh, V. Haerinck, T. Steiner, D. Van Deursen, S. Van Hoecke, J. De Roo, R. Van de Walle, and J. Gabarró Vallés, "Functional composition of sensor Web APIs," in *Proceedings of the 5th International Workshop on Semantic Sensor Networks*, Nov. 2012.

[61] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax." http://www.w3.org/TeamSubmission/n3/.

[62] R. Verborgh and J. De Roo, "Drawing conclusions from linked data on the web," *IEEE Software*, vol. 32, pp. 23–27, May 2015.

[63] "Forececast.io." http://forecast.io.

[64] X. Qian and X. Che, "Security-enhanced search engine design in internet of things.," *J. UCS*, vol. 18, no. 9, pp. 1218–1235, 2012.

[65] V. Catania, G. L. Torre, and D. Ventura, "Controlling smart objects from web applications using the webinos platform," in *Smart Objects, Systems and Technologies (Smart SysTech), 2014 European Conference on*, pp. 1–7, 2014.

[66] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 6, 2001.

[67] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, no. 0, 2014.

[68] D. Ventura, S. Monteleone, G. La Torre, G. La Delfa, and V. Catania, "Smart edifice: Smart everyday interoperating future devices," in *Collaboration Technologies and Systems (CTS), 2015 International Conference on*, pp. 19–26, June 2015.

[69] A. Mandal, C. V. Lopes, T. Givargis, A. Haghighat, R. Jurdak, and P. Baldi, "Beep: 3d indoor positioning using audible sound," in *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pp. 348–353, IEEE, 2005.

[70] Revolv Inc, "Revolv: The universal smart home automation hub and app," June 2014.

[71] Physical Graph Corporation, "Easy & affordable smart home automation," June 2014.

[72] Staples Inc, "Home automation hub & kits," June 2014.

[73] Samsung Electronics CO Ltd, "Samsung unveils new era of smart home at ces 2014," June 2014.

[74] LG Electronics, "Lg homechat^{TM}makes it easy to communicate* with smart appliances. - lg us blog," June 2014.

[75] C. Gouin-Vallerand and S. Giroux, "Managing and deployment of applications with osgi in the context of smart home," in *Wireless and Mobile Computing, Networking and Communications, 2007. WiMOB 2007. Third IEEE International Conference on*, IEEE, 2007.

[76] N. Papadopoulos, A. Meliones, D. Economou, I. Karras, and I. Liverezas, "A connected home platform and development framework for smart home control applications," in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, IEEE, 2009.

[77] A. M. Bernardos, L. Bergesio, J. Iglesias, and J. R. Casar, "Meccano: a mobile-enabled configuration framework to coordinate and augment networks of smart objects," *Journal of Universal Computer Science*, vol. 19, no. 17, pp. 2503–2525, 2013.

[78] C. Y. Leong, A. R. Ramli, and T. Perumal, "A rule-based framework for heterogeneous subsystems management in smart home environment," *Consumer Electronics, IEEE Transactions on*, vol. 55, no. 3, 2009.

[79] V. Ricquebourg, D. Menga, D. Durand, B. Marhic, L. Delahoche, and C. Loge, "The smart home concept: our immediate future," in *E-Learning in Industrial Electronics, 2006 1ST IEEE International Conference on*, pp. 23–28, IEEE, 2006.

[80] D. Zhang, T. Gu, and X. Wang, "Enabling context-aware smart home with semantic web technologies," *International Journal of*

Human-friendly Welfare Robotic Systems, vol. 6, no. 4, pp. 12–20, 2005.

[81] LogMeIn-Inc, "Xively by logmein business solutions for the internet of things," June 2014.

[82] Evrythng, Ltd, "Evrythng: Make products smart," June 2014.

[83] All Seen Alliance, "Alljoyn framework - a common language for the internet of everything," June 2014.

[84] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proc. NSDI*, 2012.

[85] Linux MCE community, "Home: Linuxmce home automation," June 2014.

[86] M. Sethi, J. Arkko, and A. Keranen, "End-to-end Security for Sleepy Smart Object Networks," pp. 964–972, 2012.

[87] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Chang Shantz, "Sizzle: A standards-based end-to-end security architecture for the embedded internet," *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 425–445, 2005.

[88] Y. Matsuoka, "Today's earth day. tomorrow should be too." `https://nest.com/blog/2014/04/22/todays-earth-day-tomorrow-should-be-too/`, 2014. Accessed: 03-12-2015.

[89] K. Finley, "The internet of things could drown our environment in gadgets." http://www.wired.com/2014/06/green-iot/. Accessed: 03-12-2015.

[90] D. Ventura, D. Casado-Mansilla, J. López-de Armentia, P. Garaizar, D. López-de Ipiña, and V. Catania, "Ariima: A real iot implementation of a machine-learning architecture for reducing energy consumption," in *Ubiquitous Computing and Ambient Intelligence. Personalisation and User Adapted Services* (R. Hervás, S. Lee, C. Nugent, and J. Bravo, eds.), vol. 8867 of *Lecture Notes in Computer Science*, pp. 444–451, Springer International Publishing, 2014.

[91] D. Casado-Mansilla, J. López-de Armentia, D. Ventura, P. Garaizar, and D. López-de Ipiña, "Embedding intelligent eco-aware systems within everyday things to increase people's energy awareness," *Soft Computing*, pp. 1–17, 2015.

[92] A. Ariyo Adebiyi, A. Oluyinka Adewumi, and C. Korede Ayo, "Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction," *Journal of Applied Mathematics*, 2014.

[93] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*, vol. Holden-Day, Incorporated. 1990.

[94] T. Iordanova, "Introduction to stationary and non-stationary processes." http://www.investopedia.com/articles/trading/07/stationary.asp, 2007. Accessed: 03-12-2015.

[95] G. A. Rob J. Hyndman, *Forecasting: principles and practice*. OTexts, 2013.

[96] F. Pouzols and A. Lendasse, "Effect of different detrending approaches on computational intelligence models of time series," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–8, July 2010.

[97] A. Negron, "Arima models in a nutshell." `http://altons.github.io/r/2013/05/12/arima-models-in-a-nutshell/`, 2013. Accessed: 03-12-2015.

[98] M. Khashei and M. Bijari, "An artificial neural network (p,d,q) model for timeseries forecasting," *Expert Syst. Appl.*, vol. 37, pp. 479–489, Jan. 2010.

[99] J. López-de Armentia, D. Casado-Mansilla, S. López-Pérez, and D. López-De-Ipiña, "Reducing energy waste through eco-aware everyday things," *Mobile Information Systems. IOS Press*, vol. 10, no. 1, pp. 79–103, 2014.

[100] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Journal of Neural Networks*, vol. 2, pp. 359–366, Jan. 1989.

[101] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae). arguments against avoiding rmse in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[102] S. seok Choi and S. hyuk Cha, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, pp. 43–48, 2010.

[103] R. Yang and M. W. Newman, "Learning from a learning thermostat: Lessons for intelligent systems for the home," in *Proc. of UbiComp'13*, vol. ACM, pp. 93–102, 2013.

[104] J. Chapman, *Emotionally Durable Design: Objects, Experiences and Empathy*, vol. Earthscan LLC. 2005.

[105] L. Broms, C. Katzeff, and et al., "Coffee maker patterns and the design of energy feedback artefacts," in *Proc. of DIS'10*, vol. ACM, pp. 93–102, 2010.

[106] T. Crosbie and M. Houghton, "Sustainability in the Workplace," tech. rep., Sustainability at Work, 2012.

[107] J. Álvaro Muñoz Naranjo, P. O. na, A. Gómez-Goiri, D. L. de Ipiña, and L. G. Casado, "Enabling user access control in energy-constrained wireless smart environments," vol. 19, no. 17, pp. 2490–2505, 2013.

[108] J. Á. M. Naranjo, A. Gómez-Goiri, P. Orduña, D. López-de-Ipiña, and L. G. Casado, "Extending a user access control proposal for wireless network services with hierarchical user credentials," in *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13 - Salamanca, Spain, September 11th-13th, 2013 Proceedings*, pp. 601–610, 2013.

[109] D. Ventura, A. Gómez-Goiri, V. Catania, D. López-de Ipiña, J. Naranjo, and L. Casado, "Security analysis and resource requirements of group-oriented user access control for hardware-constrained wireless network services," *Logic Journal of IGPL*, 2015.

[110] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, (London, UK, UK), pp. 1–15, Springer-Verlag, 1996.

[111] L. Chen, "Sp 800-108. recommendation for key derivation using pseudorandom functions (revised)," tech. rep., Gaithersburg, MD, United States, 2009.

[112] M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, ill ed., 2008.

[113] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "Spins: Security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[114] S. Zhu, S. Setia, and S. Jajodia, "Leap+: Efficient security mechanisms for large-scale distributed sensor networks," *ACM Trans. Sen. Netw.*, vol. 2, pp. 500–528, Nov. 2006.