UNIVERSITÀ DEGLI STUDI DI CATANIA
Dipartimento di Ingegneria Elettrica, Elettronica ed Informatica

Dottorato di Ricerca in Ingegneria Informatica e delle Telecomunicazioni
XXVII Ciclo

SECURE ACCESS TO CONTEXT-AWARE SERVICES IN A SMART CITY

Ing. Giuseppe La Torre

Coordinatore	Tutor
Chiar.ma Prof.ssa V. Carchiolo	Chiar.mo Prof. V. Catania

To my family,
who showed me the way.

We're still
in the first minutes of the first day
of the Internet revolution.
SCOTT COOK

SOMMARIO

Accesso Sicuro a Servizi Context-Aware nella Smart City

La tesi affronta alcune delle problematiche inerenti l'interazione da parte degli utenti con i servizi che saranno presenti nelle future smart cities. Tali servizi saranno progettati per migliorare la qualità della vita di chi la città la vive quotidianamente, ossia i cittadini, e avranno l'obiettivo di migliorare aspetti oggi critici: la mobilità sostenibile, il risparmio energetico, l'inclusione sociale, la sicurezza e salute del cittadino. Il cittadino, è considerato il fulcro delle città del futuro e attorno a lui graviteranno dei servizi di nuova generazione di tipo context-aware, che cioè vengono erogati in funzione del contesto fisico o logico in cui gli utenti si trovano nei momenti della loro attività quotidiana.

La presente tesi mette in risalto come nel corso degli ultimi anni, grazie a fattori come la trasformazione del Web (con la nascita dell'API Economy), la diffusione dei social network, degli smartphone ed an-

che dei *wearable device*, le abilità e dunque le potenzialità degli utenti si siano evolute a tal punto da parlare oggi di veri e propri "utenti smart", in grado non solo di consumare ma anche di generare nuovi contenuti e servizi e renderli disponibili ad altri utenti. Sulla scia dei cambiamenti che il Web sta avendo ed avrà nei prossimi anni, la tesi affronta inoltre le problematiche alla base dell'interazione tra utenti e oggetti in scenari tipici del Web of Things prima e del Machine to Machine in seguito, mettendo in risalto la mutazione che sta avendo il ruolo che ha l'utente nell'interagire con i servizi. Un ulteriore aspetto considerato nella tesi è quello della sicurezza per gli utenti nel momento in cui accedono ai servizi offerti dalla smart city. Questo tema è di particolare interesse dal momento che in scenari di Web of Things gli utenti non interagiscono solo con contenuti virtuali presenti sul Web (foto, video, etc) ma anche con oggetti reali che se usati senza controllo possono creare dei danni tangibili. Il concetto di sicurezza va quindi declinato non solo nella classica accezione di sicurezza informatica, intesa come controllo di accesso ai servizi, ma anche nella forma di *safety*, intesa come salvaguardia dell'incolumità del cittadino.

La tesi affronta con particolare attenzione gli scenari che riguardono gli User-Provided Mobile Services (servizi forniti dagli utenti in mobilità) e la User-Objects Interaction (interazione tra utenti e oggetti reali) proponendo delle soluzioni che poggiano sulla piattaforma *webinos*, realizzata nel corso dell'omonimo progetto europeo conclusosi alla fine del 2013. La tesi inoltre delinea alcuni scenari, nell'ambito del Machine to Machine, nei quali gli smart objects possono cooperare tra di loro senza (o con un minimo) intervento dell'utente, e propone una possibile architettura a blocchi per logicamente abilitare tale cooperazione.

SUMMARY

Secure Access to Context-Aware Services in a Smart City

The thesis addresses some of the issues related to interaction by users with the services that will be available in future smart cities. These services will be designed to improve the quality of life of people who live the city daily, ie citizens, and will aim to improve critical aspects which today affect our cities: sustainable mobility, energy saving, social inclusion, health and safety for the citizen. The citizen is considered the heart of the future city and a lot of new generation services will surround him. This services will be context-aware, that is, they will be provided according to the physical or logical context where users are located in the moments of their daily activities.

This thesis highlights how over the past few years, thanks to factors such as the transformation of the Web (with the birth of the API Economy), the spread of social networks, smartphones and even the *wearable* device, the skills and thus the potential users have evolved

to the point of talking now of real "smart" users, able not only to consume but also to generate new contents and services and make them available to other users. In the wake of the changes that the Web is having and will have in the coming years, the thesis also deals with the issues underlying the interaction between users and objects in typical scenarios the Web of Things first and Machine to Machine later, highlighting the mutation that is taking the role that has the user to interact with the services. Another aspect considered in the thesis is that of security for users when accessing services offered by the smart city. This topic is of particular interest since in scenarios such as the Web of Things users not only interact with virtual content on the Web (photos, videos, etc) but also with real objects that when used without control can create tangible damage. The concept of security should therefore declined not only in the classic sense of IT security, understood as control of access to services, but also in the form of *safety*, understood as safeguarding the citizen.

The thesis particularly focuses scenarios that concern the User-Provided Mobile Services (services provided by mobile users) and User-Objects Interaction (the interaction between users and real objects) proposing solutions that rest on the platform *Webinos*, made during the homonymous European project, which ended at the end of 2013. The thesis also outlines some scenarios in the context of the Machine to Machine, in which the smart objects can cooperate with each other without (or with minimal) user intervention, and proposes a possible block architecture to logically enable such cooperation.

Pubblications Related To This Research

1. V. Catania, **G. La Torre** and D. Ventura. *"Controlling Smart Objects from Web Applications using the Webinos Platform"*, ITG-Fachbericht-Smart SysTech 2014.

2. V. Arena, V. Catania, **G. La Torre**, S. Monteleone and F. Ricciato. *"SecureDroid: An Android security framework extension for context-aware policy enforcement"*, IEEE International Conference on Privacy and Security in Mobile Systems (PRISMS), 2013, pp.1,8, 24-27 June 2013.

3. V. Catania, **G. La Torre**, S. Monteleone and D. Panno. *"A Cloud Platform to support User-Provided Mobile Services"*, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization (IARIA CLOUD COMPUTING), 2013, pp. 191-194, May 27 - June 1, 2013.

4. V. Catania, **G. La Torre**, S. Monteleone, D. Patti, S. Vercelli and F. Ricciato. *"A Novel Approach to Web of Things: M2M and Enhanced Javascript Technologies"*, IEEE International Conference on Green Computing and Communications (GreenCom), 2012, pp.726,730, 20-23 November 2012.

CONTENTS

LIST OF FIGURES

CHAPTER

ONE

INTRODUCTION

1.1 How has the role of the user changed and where it is heading to

It has been 25 years since that March 12, 1989 when Tim Berners-Lee at CERN suggested that it was a new model for the organization and retrieval of information. According to this model, each piece of information was defined by links within a hypertext. That document was the foundation of what would become the World Wide Web, the construction of which began only a few months later by activating the first server and bringing online the first web page ever. After 25 years it is increasingly clear the scope of what is in fact an incredible revolution that changed and still is changing the way of life of the users. According to Wikipedia, a user is a "person who uses a computer or network service". Users generally use a system without the technical

expertise required to understand it fully, while power users use advanced features of programs. In its early years the Web consisted of texts, initially in their own right, which over time have been linked together. The main innovation of the Web was in fact the hyperlink, which allows an immediate connection to other pages or resources. From the user point of view this was revolutionary because it made it possible to speed up the way he retrieved, almost instantly, contents that were physically on the other side of the world. The Web and the e-mail made it possible to cancel geographical distances by becoming users "citizens of the world".

1.1.1 The rise of social media

Since from the beginning of the new millennium, the appearance of Web sites has begun to change, and subsequently, the way users interact with them. Figure 1.1 shows the rise of social media web sites since 2001. Users have gone from being simple passive users of information to those who personally create and add content that is made available to other users. The case of Wikipedia and its plans to build the largest free encyclopedia is based on the concept of sharing user's "knowledge" in favor of others.

The blogging phenomenon has begun to catch on in America in 1997; July 18, 1997, was chosen as the symbolic date of birth of the blog, referring to the development, by the US Dave Winer, the software that allows the publication. At this stage it was used the term *weblog* or *blog* with which we referred only to lists of links (a type of very useful information to users before the widespread use of search engines). The technological enabler that allowed the spread of blogs

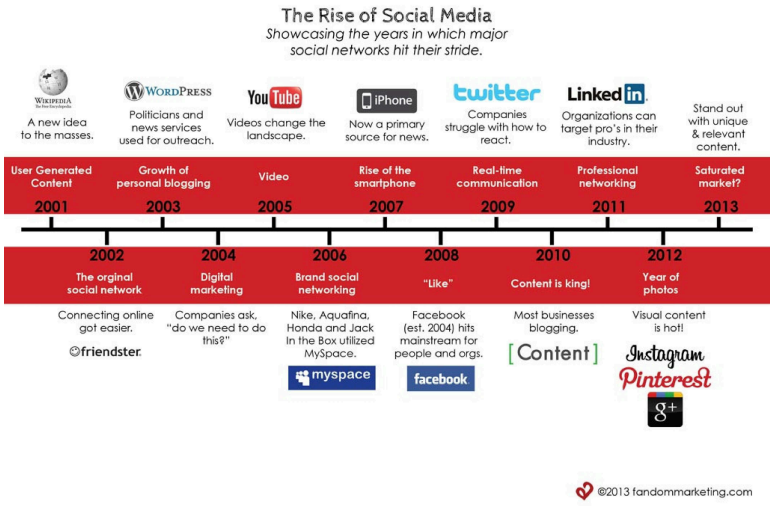

Figure 1.1: *The rise of social media*

was the Content Management System (CMS): a framework for creating high level web pages according to the paradigm of WYSIWYG (What You See Is What You Get), that is, without the stringent need to know programming languages like HTML Web. Thanks to the CMS non-technical users were able, through their personal blogs, to create contents that contributed to the spread of the Web. Between the years 2009 and 2010, however, the crisis of blogs began, mainly due to the rapid rise of social networks.However, the reason why blogs are so popular is to be found in several factors: by the exhibition's public private life to the creation of complex texts and specific; at the base of the diffusion is in any case the feature of sharing.

The phenomenon that has largely characterized Web 2.0 has been the rise of social networks: evolution of some forms of social interaction

that the web has always supported (computer conferencing, email, mailing lists, etc.). The definition of social networks is as follows:

> *A network of social interactions and personal relationships*

In more technical terms it means:

> *A dedicated website or other application which enables users to communicate with each other by posting information, comments, messages, images, etc.*

Although the conceptual point of view social networks do not constitute a new idea (like blogs, the key issue is that of sharing content) they introduce some innovative aspects, including the "Profile". Each user of a social network has its own account which allows you to manage the settings in your profile that identifies the user in all respects within the social network. In any social networks (facebook, twitter, youtube) there is no concept of followers: Users decide among themselves who to follow based on the principles of friendship (Facebook), of job skills (linkedin), of interest in what they have to say (twitter). The main innovation introduced by Facebook was the "like". While it may seem a minor issue, when a user expresses his preference for a content it adds to its popularity. This affects a lot especially for the commercial products as to accumulate the number of likes contributes to product advertising. Another key aspect of like it the way it is used to "profile" the user according to his preferences. This information is then conveyed by Facebook to provide users of commercial proposals as they match their profile, making ad hoc campaigns to market and

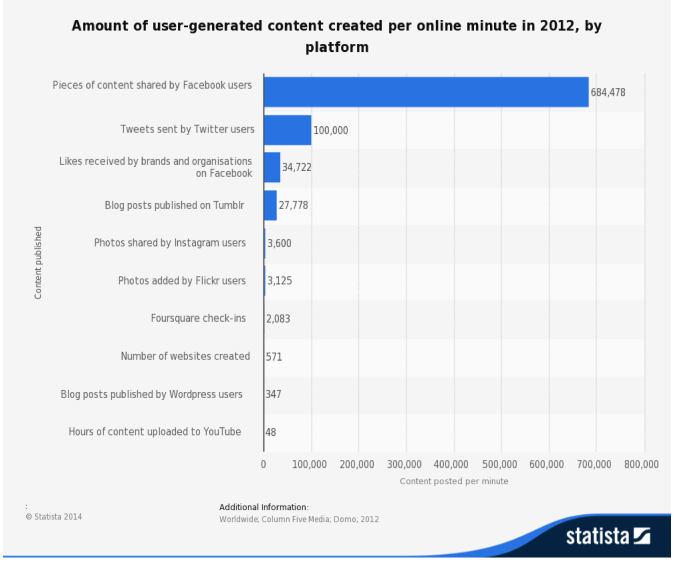

Figure 1.2: *The amount of User Generated Contents in 2012*

therefore much more effective than traditional methods of broadcast (TV, newspapers).

Another interesting phenomenon that has been able to analyze thanks to social networks is trending. Thanks to Twitter, for example, you can know in real time the topic in a given time which collects more interest in the entire globe. These information are part of the so-called big data, thanks to techniques of data analysis (data mining) are used to implement a number of sociological studies on how the company is evolving.

Social networks have introduced the concept of *social influencer*. Social influence occurs when one's emotions, opinions, or behaviors are affected by others. For Social Influencer mean a subject very specialized and active in producing information about a particular indus-

try/topic and with a large following on the Web. A social influencer is not just a popular person on the Net, but specifically it is a subject capable of: Providing 'detailed information continuously, influencing the opinions of others and creating around him a community of people working on a theme, which follows him every day.

The advantages of Web 2.0 are essentially related to the growth of a sense of "social", but it is necessary to highlight some inherent risks. Just the fact that they contribute so substantially to the content of a site, with a clear commitment and expenditure of energy, it makes the user "addicted" from that site, linking it to the final data format adopted, so any change of environment will inevitably be costly. Similarly, if the user decides to participate in multiple social networks, in the absence of common standards could be forced to repeat the operation several times. Some researchers point out that the distribution of user-generated content would be detrimental to the traditional sources of knowledge, and the fact that this content is created by users using different systems (podcasts, blogs, wikis, chat systems, and other software for social networking) makes it difficult to keep track of where we find the information, and problematic access to it, both for regular and casual users.

The semantic technologies are in a phase of diffusion also in industrial reality. In a study (May 2007) Gartner foresees a wide spread over the next ten years. Web 2.0 and the Semantic Web (or Web 3.0) are considered two complementary approaches, rather than alternative. The Web 2.0 has a low input level (it is very easy to use), but also quite limited horizons (in particular, the approach of folksonomy has inherent limitations). On the other hand, the Web 3.0 requires initial investments most relevant, and therefore presents a higher input

level, but has a much higher potential.

1.1.2 The iPhone and the app disruption

Steve Job's creation was not just a cell phone; rather, it was the world's first, handheld computer. Its data processing capabilities - not voice - are what disrupted the cell phone market. Although other smartphone manufacturers offered web browsers, they were clumsy and difficult to use. In contrast, Apple's web browser made surfing the Internet easy. Compared to its rivals, the iPhone's user interface was simple, intuitive and uncomplicated. At the swipe of a finger on touch sensitive glass, one could get access to e-mail, text messaging, video, photography, maps, books, music, games and mobile shopping. The iPhone was a game-changer, the industry's Swiss Army knife.

The market launch of the first smartphone from Apple was an event that changed the concept of mobile phone for users. Other products (Blackberry, Windows Phone, etc.) were already present in a market with Personal Digital Assistants (PDA), which did not succeed. By 2003 - 2004, there were numerous smartphones on the market competing against personal digital assistants. Although they were bulky, at the time PDAs had numerous advantages over smartphones, e.g., Windows operating system, compatibility with different file types, support for both Bluetooth and Wi-Fi, higher-performance processors, higher quality screens and audio output. However, by the year 2006 smartphones evolved tremendously: they got support for Wi-Fi and also featured 3G baseband, in addition, their multimedia capabilities were a far cry from what they were just several years before that. As a result, in 2005 - 2006 time frame the popularity of PDAs among

business users started to decrease and at present almost nobody use them for business purposes.

In 2007, it was the revolution: the original iPhone blew away the competition and was preparing to begin a new era of telephony, the one we are experiencing today. Gradually, it has gone from revolution to innovation, from the innovation to the improvement of the "phone that has changed phones forever". Maybe because this product has become part of our daily lives, that will be the approach to the device is pretty simple for everyone, it will be for features and ease of use of iOS, it will be just for the brand that is created around this "different" phone, in each case, the phenomenon iPhone was unique and probably unrepeatable.

There were two factors that contributed to the spread of the iPhone and subsequent radical transformation of the smartphone market. The first was the introduction of the *touchscreen*, which is not so much a technological innovation but constituted a revolution in the way the phone could be used. With gestures and a uniform user experience across different apps, users could use with simplicity never seen on a smartphone. An example is the zoom feature by "pinch": let us to enlarge or restrict what we are seeing and debugger regardless of whether we are the browser, image gallery or a PDF reader. In a similar way the "swipe" allows us to scroll through the content in each application you use. The presence of menus, gestures (eg zooming), a mode of navigation between screens have well-defined and well-established fact that the app is being used by many age groups. Many people who are not able to use a PC instead can use mobile applications. The second factor that contributed to the success of the iPhone was the opening towards the developers by Apple with the launch of the App store

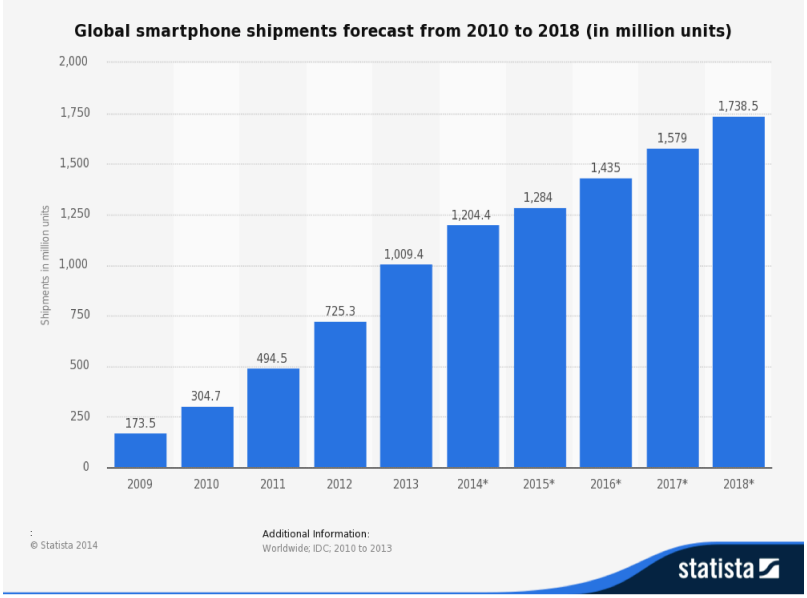

Figure 1.3: *Market forecast for smartphones until 2018*

and release of high-level tools for application development. The presence of a market with many applications has encouraged the spread of platforms such as iOS and Android at the expense of others, such as Windows Mobile, Blackberry and Symbian until the appearance of the iPhone held the largest share in the PDA market.

The market launch of the iPhone has therefore initiated the era of smartphones, today considered one of the most disruptive of the last 30 years. As shown in Figure 1.3, since 2009 (the year of release of the iPhone 3GS) to date the number of smartphones sold is rising.

Figure 1.4 highlights two important aspects: firstly, the use of the web from mobile devices(smartphone, tablet) has now surpassed that of desktop computers, and secondly, this phenomenon occurs for all

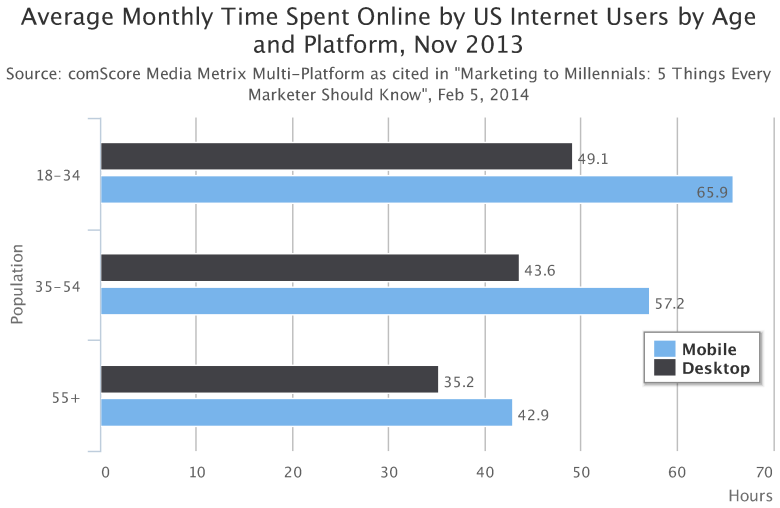

Figure 1.4: *Time spent online by age*

age groups. This means that the smartphone has considerably cut down the barrier of entry to the Web for users younger than with desktop computers struggled to make full use of the opportunities provided by the web. The smartphone is thus intended to be the tool that will allow the user to interact with the services of digital cities. It will be the point of contact between the user and the service and will be the virtual identity of users in smart cities.

1.1.3 New kinds of interaction in the Internet Of Things

Today users are improving themselves in the way they use mobile applications. In particular, the social applications are those with more following and allow users to create and share virtual content: tweets,

images, video, audio. However, the technological landscape is evolving into what it is uniformly recognized as the Internet of Things: a scenario where not only computers and smartphones, but all the objects that we use every day will be connected to the Internet. The IoT will be a real technological revolution that will make possible the application scenarios today that will bring a new day to the transformation of the city in which we live in the real smart cities, in which the user will be surrounded by objects (typically sensors and actuators) with which he can communicate and exchange information. A typical example of IoT device which is spreading in this period is the Fitbit[1]: a bracelet with inertial sensors that helps users monitor their daily physical activity and the quality of their sleep. The data collected by the Fitbit are carried on a cloud platform by connecting the user's smartphone and turn into statistics. The user then generates and uses contents that have not been created by other users as was the case up to now, but coming from the smart objects that are around him. Although the smartphone is currently the best tool to enable the interaction between users and objects, including other forms of interaction can be used in the future, including

- Vocal interaction: a topic of much current research, some of the most important implementations are Apple's Siri and Google Now.

- Gestures Recognition: when used in the world of gaming (Kinect, Wii) is well suited for interacting with objects. Another alternative is Leap Motion to recognize the fingers' movement.

[1] www.fitbit.com

- Complementary interaction. Google Glass and and Apple Watch are different, but underlie a common concept: exploiting the connection and the data from the mobile in a new way to interface with technology, such as using augmented reality.

This dissertation will take into account the concept of the Internet of Things and it will be explained in what circumstances such a concept has turned into the Web of Things. The transition from Web Documents to the Web of Things was made possible by an intermediate step, characterized by the emergence of Web Services. The next two sections describe the details of this transformation.

1.2 From the Web of Documents to Web of Services

In recent years more and more big companies have adopted the strategy to release public Application Program Interfaces (APIs) to enable third-party developers to create applications and services based on well-established platforms. Some examples are Google, Facebook, Twitter, which made it available to developers access to their data (maps, user postings, tweets), which in turn used this information to create applications for end users by generating profits. For developers, rely on already existing and well-established services, is a way of abstracting from issues such as server management, scalability of their product increases users who use, data backup, etc. In this section we will analyze what were the reasons that led to the birth of the API Economy and the immediate consequences will be described from the point of view of the developers from the end-user.

1.2.1 The rise of the API economy

According to Ross Mason's vision [1] (the founder of the US company MuleSoft), the API is considered a revolution as a few decades ago it was the industrial revolution. According to Mason, four are the conditions to start a revolution: Demand, Resources, Innovation and Adaptation. In the case of the industrial revolution these conditions have been characterized by:

- Demand: Population growth specialist and free trade
- Resources: The abundance of raw materials such as coal, iron, steel
- Innovation: Some inventions of machines that sped up the work of man
- Adaptation: Man's ability to generate profit using manpower, resources and innovations

The first real digital revolution has been the Web, but today we are witnessing what is considered the second digital revolution that is the rise of the API.

- Demand: The population of the Web will increase from 2.8 billion to 5 billion in the near future
- Resources: The enormous amount of data available (social networks, open data, IoT)
- Innovation: The Web is now a platform on which to build any kind of application

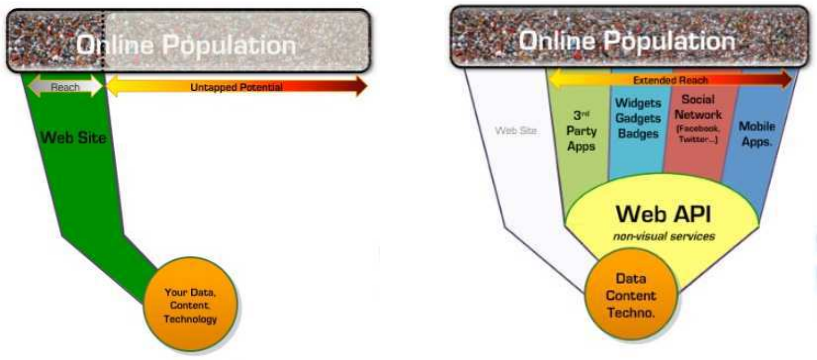

Figure 1.5: *Advantaged for companies in the API Economy*

- Adaptation: The ability of companies to make a profit by creating products that use the API of the Web

As shown in Figure 1.5, for a company that wants to make profit from the data that is available, a Web site that would cover only a small portion of the online population (0.001%). In contrast, leveraging on the APIs that allow users to access data allows to capture 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 everything you can take advantage of the Web API.

In conclusion, we can say that the Web API represent a new opportunity for the business-to-business (B2B) and represent a new channel for communicating partners (third parties) with customers, indirectly generating profit.

1.2.2 The impact on users

The Web services have introduced a number of advantages for developers and, indirectly, also for end users who have seen an increase in the number of applications available to them. An important concept originated by the presence of numerous services available on the Web is the *Mashup*. Mashup means the composition of existing services with the aim to create new applications for end-users or new services that may in turn be used other by developers. Chapter 2 will highlight the main benefits of mashups and the platforms which exist today and make it possible.

Although the mashup is definitely a tool for developers, recently we have seen the emergence of some high-level tools which allow users (not necessarily experts) to compose, using the graphical tools, existing services and creating new ones. A popular platform today for the composition of services is "If This Than That" (IFTTT), which will be described in the next chapter. Such tools thus giving users the ability to create, as well as content, new services that can then be used by other users. This could lead in the future to have, as is the case today for applications, markets for services. Chapter 3 describes the work carried out in the context of User Generated Services in which it was considered the case in which, in addition to being generated, the services were also provided by users through their mobile devices. Many scenarios can be enabled in this area, one of them is known as *crowdsensing*[2].

1.3 From the Web of Services to Web of Things

In the previous section it was shown that users are able to exploit the power of the Web in the form of public services it offers. This fostered the transition to the Internet of Things in which objects of daily life are connected to the Internet. These objects are heterogeneous from several points of view: hardware, protocols, interfaces. In order to control these objects we need to "virtualize" them and consider each object as a service provider. In [3] Guinard et al. propose a process and a suitable system architecture that enables developers and business process designers to dynamically query, select, and use running instances of real-world services running on physical devices. It is therefore a direct consequence that the Web of Services constitutes the basis for the Web of Things: each object has its virtual counterpart that implements the services that coincide with the operations that the object is able to perform.

1.3.1 The phenomenon of makers

The more frequent appearance of new smart objects that are part of the Web of Things was certainly encouraged by the birth of the *makers* movement. The makers concept is a contemporary evolution of the technological DIY (do it yourself) determined by a number of changes taking place in technology and society. The makers are the natural consequence of the Internet, social platforms and the dissemination of techniques for rapid production. They operate within a digital community of thousands of fans and founded on the philosophy of

knowledge sharing and open source. The birth of the subculture of the makers is closely associated with the birth of hackerspace, spaces for collaborative innovation. In 2009, there were over a hundred in the United States. The secret behind the revolution lies in the intertwining between digital and analog, and the most important technology is the 3D printer: a machine that produce a solid, three-dimensional model from a digital computer. Many open source projects are based on Arduino, which enabled several of possibilities related to the design of robots, wearable, and IoT applications.

1.3.2 Social networks of things

In recent years, social networks are evolving hand in hand with the consolidation of what is called the Internet of Things (IoT). The IoT can be understood as the evolution of the Internet of computers to an Internet where everyday objects are connected to each other and exchange information with each other and with users. The concept of IoT will be further elaborated in the course of this dissertation. In this direction we have already seen the born of social networks where objects are the main players and not the users. Within these social networks objects (in most cases of simple sensors) have their own "life" and publish on a dashboard information which they are capable of measuring. Each object can have followers: users that can see at any time the status of the object and control it. To simplify the discovery, objects can be geo-located and linked to a tag helping users to find them. Mechanisms of access control are also present, to determine which other users, as well as the owner can view the status of an object. In addition, some platforms provide the API to control

the state of objects also outside of the platform itself, for example from a mobile app. The Web then becomes something tangible to users, which for the first time are able to control the objects that surround them in a completely seamless, without noticing that the resource with which they are interacting has its real counterpart. Chapter 4 will address in detail various aspects of the interaction between the user and objects.

1.3.3 Machine to Machine

The Machine to Machine (M2M) is considered as a special case of the Web of Things in which objects (called Machine) have their intelligence and carry out specific tasks without human intervention. Traditionally the machines involved in these scenarios are simple sensors and actuators (a typical example of the M2M application is that of monitoring the boiler with relative closing of a solenoid valve in case of emergency). In this thesis, we want to consider machines in a broadest sense, ie quite complex objects able to perform both low and high level operations. For example, in the case of the smart home, the machine are constituted by appliances such as oven, TV, air conditioner, refrigerator. In a generic smart space, M2M scenarios are those in which the objects will cooperate with each other to satisfy goals that are expressed by users through high-level interfaces. M2M then changes once again the role of the user: he does not anymore directly control objects to achieve a result (e.g. to manually control the air conditioner), but he merely "asks" the desired result (goal), leaving to the objects the burden of self organize them-selves and carry out the assigned task.

Various aspects of research are involved in M2M, a discussion of all open issues will be provided in Chapter 5 together with a proposal for a possible architecture for the deploy of M2M strategies for smart spaces.

1.4 Services within a Smart City

The term *smart city* has become particularly popular in recent period. This expression identifies an urban area that, thanks to the use and pervasive of advanced technologies (not only ICT), is able to address in an innovative way a series of problems and needs. There are many forms according to which a city can become smart. Among the most mentioned, it is possible to certainly remember the following:

1. A city that helps people to move. The city (and territories around them develop) are becoming increasingly congested and therefore require new models of management and governance of mobility that enhance public transport, introducing types and transport models (eg, patterns of sharing of the medium), providing innovative services for monitoring, analysis, planning and management of the flows of people and resources.

2. A city that helps people to not move: In apparent contrast to the previous point, the city is also smart to the extent that it helps people to stay put. In particular, a widespread and pervasive use of ICT products and services allows us to perform remotely, without moving, a lot of activities from shopping, meetings, activities, group work and projects.

3. The city that helps people to know. A smart city is able to collect and disseminate information in an extensive and continuous, as regards both the normal social and economic life, both as regards the management of emergency situations.

4. The virtuous city. A smart city is able to exploit all the modern technologies for energy saving and, in general, to reduce the impact on the environment and on the planet that comes from the presence and activities of thousands of people and products in various forms consume energy and produce waste.

5. A city which is alive and dynamic. A city is smart even when it is able to generate and promote cultural and recreational activities that qualify the territory, attracting talent, enrich the fabric of the city and will stimulate creativity and social growth.

6. A city which is participated. The growth of cities and their gradual transformation into large agglomerations where you lose the size of the "medieval square", makes it more real danger of the loss of social cohesion and the impoverishment of opportunities to meet and socialize. A smart city is capable of inventing new forms of participation that combining the use of new technologies and new forms of social encounter, they are able to renew and rebuild the fabric of human relationships and opportunities for discussion and dialogue.

7. A city which is safe. The security of people and property in many cities has become a major concern. A smart city raises the level of reliability through the use of innovative solutions for ground surveillance and assistance to citizens.

8. A city which is well-governed. Finally, but not least, a smart city offers new forms of governance that can both monitor and manage the land and the dynamics that develop in it, is to enhance the ongoing relationship and two-way with citizens, businesses, entities live that operate on it and develop.

A smart city is a place where all life processes and nerve centers of social life are read, thanks to the use of technology in order to radically improve quality of life, opportunities, welfare, social and economic development.

1.4.1 Context-aware Services

The context-aware applications allow to provide content, information and services tailored to the context in which the user is located. With the term context indicates precisely a set of data relating to the state of the user that the environment in which this is located. The context-aware services are intended to provide information consistent with the situation that surrounds the user, adapting to possible changes in circumstances.

The knowledge of the context in which the user is located allows him to offer a wide range of services to help the customer in his daily life, working-operative or private, to better manage time, revealing what is around him and where are the people he want to share emotions and experiences, introducing new forms of entertainment. The CA (Context Awareness) is a set of engineering features that can add value to services in different application segments. Context Awareness applications and services may exploit these features for various purposes:

- present information: services involving context information to the customer or using context to suggest a selection of proper care actions;

- execute commands: services that run commands or reconfigure systems for the customer in terms of changes in the context;

- tagging of information: services to associate information or objects to to a service (documents, meeting rooms, meetings, printer/fax/pc, ...) with context information (time, location, identity , activities).

A user can be in a *physical* context that is characterized by physical parameters, such as his location, his status (he is moving, sitting, doing sports), environmental data which can be snatched around him (temperature and atmospheric pressure, brightness, humidity), by physiological parameters (blood pressure, body temperature), etc. other than *physical*, the context can be *logical* and consists of both user's real and virtual identity. The logical context can enable services to users who are really allowed, for example in a factory only users with a certain role may be able to use certain types of services.

1.4.2 Location Based Services

New technologies have become a mass phenomenon that involves most of the population. Mobile, Smartphone and Social are all words that begin to know: According to Cisco half the world's population will be connected to the Internet in 2017, and of these about 93 % is also present on social networks. And, in 2017 more than a third of the

world population will have a smartphone. The proportion of smartphone owners using geo location services (LBS - Location Based Services) is definitely growing. Geo location, ie the identification of the geographical position of a given object in the real world, is present in everyday life and the concept of Check In, the action that allows a user to share a moment, it is more and more integrated services used from Mobile. If you want to analyze the reasons for the growth of geo location systems. Mainly, the three factors which have fostered LBS are: technology, data and app market. In particular:

- the increase of the precision of the devices is turned in a few years from 100 meters to 5 meters and the time alignment of the detection is increased from 10 seconds to 1 second, generating a reduction of costs and the possibility of information almost in real time;
- The emergence of indoor localization techniques which are under study and implementation.
- the adoption and development of technology pre-installed in smartphones has made geo location capabilities available to all; the availability of the mobile connectivity has allowed more people to interact with more platforms and social applications;
- the ecosystem of app stores and the API generated and accelerated the development of applications for mobile devices, the ability to integrate data and generate business opportunities.

The answer to "Why are the location and the LBS becoming so important today?" it becomes almost trivial: geo location, viewed

from the business answers questions important means of monitoring and analyzing its users, not only from the point of analytically, but mostly behavioral. Anyway, the most important aspect concerns the "social" effects resulting from the sharing of a position, even more analytical and strategic importance of the study of that data. Mobile proximity marketing and social games are only a few examples of LBS implications.

1.5 Structure of this Dissertation

The thesis is structured as follows: Chapter 2 describes the current state of the art with regard to Web services, considering their description, use, generation and all the problems related to security and privacy of users who use such services. In Chapter 3 a proposal for User Generated and Provided Services is described. Chapter 4 faces the problem of user-object interaction and provides a proposal which exploits the webinos platform. 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. Finally in Chapter 6 the author provides some security and privacy considerations that embrace all the cases which have been taken into account in this dissertation.

1.6 Acknowledgments

Part of the results described in this dissertation comes from the research funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2).

The code produced while working on this project is freely available at [4] and has been forked from / contributed to *webinos* project repositories [5, 6]. Requirements, specifications and all the other deliverables are available in the project's site.[2]

[2]http://www.webinos.org

CHAPTER

TWO

STATE OF THE ART

2.1 What is a Web Service?

Web services have made their appearance around the year 2000 and since then have revolutionized the way we think the Web. A Web service is a software system designed to support interaction between applications, using technologies and Web standards. The mechanism of Web Services enables users to interact in a transparent applications developed with different programming languages, running on heterogeneous operating systems .

This mechanism allows users to create pieces of functionality independently and potentially incompatible platforms interacting via the various pieces and Web technologies to create modular architecture easily. At present there are two approaches to the creation of Web Service: One approach is based on the standard protocol SOAP (Simple Object Access Protocol) to exchange messages for the invocation

of remote services, intends to play in the Web an approach to remote calls, remote procedure call, which is typical of protocols for interoperability such as CORBA, DCOM, and RMI.

A second approach is inspired by the traditional architectural principles of the Web and focuses on the description of resources on how to find them on the Web and how to transfer them from one machine to another. This is the approach that has been the reference for this thesis and it is named REST (Representational State Transfer).

2.1.1 REST Architecture

REST defines a set of architectural principles for the design of a system. It is an architectural style, which does not refer to a specific, well-defined, nor is a standard established by a standards body. Its definition appeared for the first time in 2000 in the doctoral thesis of R. Fielding[7], "Architectural Styles and the Design of Network-based Software Architectures", which was discussed at the University of California, Irvine. This thesis analyzed the basic principles of different software architectures, including precisely the principles of software architecture that allow to see the Web as a platform for distributed processing.

In recent years, REST approach came to the fore as a method for creating Web services highly efficient and scalable and has to his credit a significant number of applications. REST architecture nor is not a standard, but a set of guidelines for the realization of a system architecture. In particular:

- Identify resources

- Usage of explicit HTTP methods
- Self-descriptive Resources
- Links between resources
- Stateless Communication

Resources are the key elements on which RESTful Web services are based. Conversely, SOAP Web Service-oriented are based on the concept of remote call. A resource is any item that is being processed. To give some concrete example, a resource can be a client, a book, an article, an object on which operations can be performed. As in the Web, the most natural mechanism for identifying a resource is given by the concept of URI.

REST allows developers to perform operations on the resources that match the verbs defined by the HTTP standard that is: GET, POST, PUT and DELETE. The principle of stateless communication is well known to those working on the Web. This is in fact one of the main features of the HTTP protocol, that is, each request has no relation to the previous requests and later. The same principle applies to a RESTful web service, that is, the interaction between the client and server must be stateless. The main reason for this is scalability: keeping the status of a session has a cost in terms of resources on the server and as the number of clients that cost can become unbearable. In addition, a communication without a state can create clusters of servers that can respond to clients without constraints on the current session, thus optimizing the overall performance of the application.

2.1.2 REST and SOAP comparison

Although the goal of both approaches is almost the same, namely the adoption of the Web as a computing platform, their vision and the suggested solution are totally different. The first noticeable difference between the two types of Web Service is the vision of the Web as processing platform.REST offers a vision of the Web which focuses on the concept of "resource", conversely SOAP approach emphasizes the concept of "action". A RESTful Web Service is the custodian of a set of resources on which a client can request the canonical operations of the HTTP protocol A SOAP-based Web Service exposes a set of methods that can be called remotely from a client. The approach of SOAP Web services has borrowed the architecture from SOA, Service Oriented Architecture, which has recently opposed the architecture ROA, Resource Oriented Architecture, inspired by the principles of REST.

SOAP (Simple Object Access Protocol) defines a data structure for the exchange of messages between applications, presenting in a sense of what was already the HTTP protocol. SOAP uses HTTP as the transport protocol, but is not limited nor bound to it, since it may very well use other transport protocols. Unlike HTTP, however, the specification of SOAP do not address issues such as security or addressing, for which standards have been defined in part, in the specific WS-Security and WS-Addressing. So SOAP takes full advantage of the HTTP protocol, using it as a simple transport protocol. REST uses HTTP instead for what it is, an application layer protocol, and uses the full potential.

2.1.3 Web Service Description

SOAP-based Web Services provide the standard Web Service Description Language (WSDL[1]) to define the interface of a service. This is further evidence of the attempt to adapt to the Wb the approach based on remote calls. In fact, the WSDL is nothing more than a IDL (Interface Description Language) for a software component. On the one hand the existence of WSDL favors the use of tools to automatically create client in a particular programming language, but at the same time causes it to create a strong dependency between client and server. REST does not explicitly provide any standard way to describe how to interact with a resource. The operations are implicit in the HTTP protocol. Something similar to WSDL is WADL[2], (Web Application Definition Language), an XML application to define resources, operations and exceptions provided by a Web Service REST. WADL was submitted to the W3C for standardization in 2009, but at present there are no plans for its discussion and possible approval. In fact it has not had a very favorable reception from the community REST, as it offers a static view of a Web Service, contradicting the principle HATEOAS (Hypermedia as the Engine of Application State) that arises in the presence of connections within the representation of a resource the definition of a contract with the client, with a vision so much more dynamic and a weak coupling between client and server. Another specification, which has been used in this thesis to describe a REST web service is *SWAGGER* which is discussed in next section.

[1]http://www.w3.org/TR/wsdl20/
[2]http://www.w3.org/Submission/wadl/

Swagger

The goal of Swagger [3] is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined via Swagger, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interfaces have done for lower-level programming, Swagger removes the guesswork in calling the service. Use cases for machine-readable API interfaces include interactive documentation, code generation for documentation, client, and server, as well as automated test cases. Swagger-enabled APIs expose JSON files that correctly adhere to the Swagger Specification, documented in this repository. These files can either be produced and served statically, or be generated dynamically from your application. Without going into a long history of interfaces to Web Services, this is not the first attempt to do so. We can learn from CORBA, WSDL and WADL. These specifications had good intentions but were limited by proprietary vendor-specific implementations, being bound to a specific programming language, and goals which were too open-ended. In the end, they failed to gain traction. Swagger does not require you to rewrite your existing API. It does not require binding any software to a service–the service being described may not even be yours. It does, however, require the capabilities of the service be described in the structure of the Swagger Specification. Not all services can be described by Swagger–this specification is not intended to cover every

[3] http://swagger.io/

possible use-case of a REST-ful API. Swagger does not define a specific development process such as design-first or code-first. It does facilitate either technique by establishing clear interactions with a REST API.

2.2 Web Services Mashup

The current trend in the development of modern Web applications, and in particular Web 2.0 applications, clearly points to involve more and more the user. The so-called social applications are proof of the value initially unexpected, the integration of end-users in the process of content creation. Another practice has emerged recently is the development of web mashups, web applications resulting from the combination of content and services available on the Web in the form of APIs (Application Programming Interface), open programming interfaces or, more generally, reusable services. The first and fundamental step in the development of mashup is the production of public services, published on the Web and therefore easily accessible and reusable.

These services are heterogeneous and can be: i) remote API services based on the exchange of messages (eg, Web services), ii) API based on the integration of programmatic code (as with the Google Maps API and Twitter), iii) feed RSS / Atom feeds (for instance, information on government grants), or iv) contents from many different websites (for example, the prices of certain products). The components used in the development of mashup are therefore of three types:

- data services such as RSS (Really Simple Syndication) or Atom, content formatted in JSON (JavaScript Object Notation) or

XML or plain text files. For example, almost all the newspapers now publish the titles of their news via RSS feeds that can be read by a so-called RSS reader and allow the user to easily skip detail of various news.

- or Web Services API (Application Programmable Interfaces) accessible through Web services as SOAP (Simple Object Access Protocol) or REST (Representational State Transfer). These services typically do not provide simple data, but allow the reuse of application logic as, for example, the calculation of the name of a city from its GPS coordinates.

- UI components (ie have a user interface) as pieces of HTML code or programmable interfaces in JavaScript (for example, the so-called widget[4]. The typical example of a UI component is Google Maps, which provides not only data in the form of maps but also a user interface can be easily integrated into a Web page that allows the user to navigate the maps. However, also the extraction of content from traditional Web pages is still a very common practice, especially in the absence of equivalent services already available and ready for use.

2.2.1 Low Level Mashup

ProgrammableWeb is a portal where Web API providers and developers may end respectively publish and use of the API. The portal provides a search engine for APIs within it. Each API, and its associated service, may be associated with the tags and categories to allow

[4]http://www.w3.org/TR/widgets-apis/

for a more efficient search. Among the most important resources made available to ProgrammableWeb we find:

- The API directory where developers can search for APIs to include in their next software development project.
- The Mashup Directory to see a showcase of Web applications that put Web APIs to work
- A list of How-To's and Source Code; a resource that we think developers will find useful for enhancing their skills
- The ProgrammableWeb Research Center where audience members can view or download the latest statistics on the API economy.

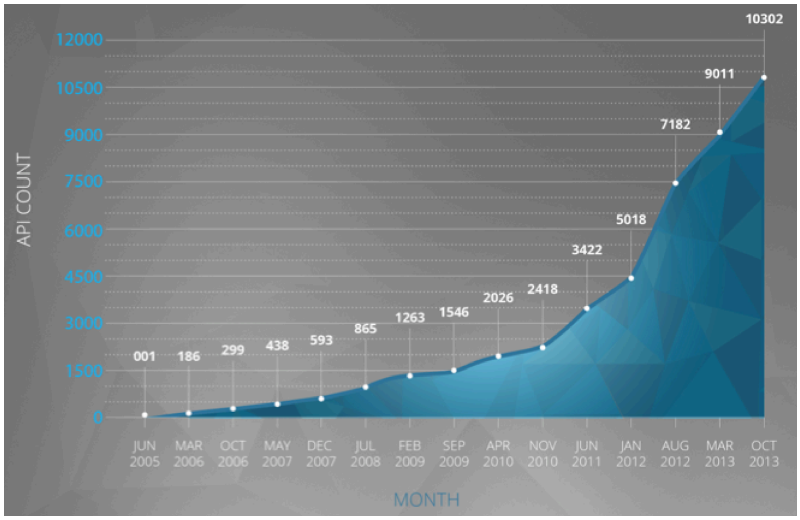

Figure 2.1: *Public APIs growth since 2005*

ProgrammableWeb does not provide a hosting for Web services to which the API reference: it only allows developers to enter a pre existing API documentation and create a pointer to the endpoint where the service is actually hosted. For these reasons ProgrammableWeb can not be used by a user that is completely foreign to the world of server-side Web programming. However, ProgrammableWeb is a very valuable resource for developers of applications and new services through mashups.

2.2.2 High Level Mashup

High Level Mashup means the ability to create new services by simply reusing APIs made available by service providers. One of the platforms of high level mashup that is currently gaining the momentum is IFTTT [5] (If This Than That).

Figure 2.2: *An example of recipe in IFTTT*

IFTTT is aimed at users who do not have expertise in Web development and allows to create through a wizard a sort of rules called "recipes". As shown in Figure 2.2 each recipe consists of a trigger and an action: the trigger is a condition that once occurred triggers

[5] https://ifttt.com

the action. IFTTT offers to its users over 140 channels (services like Facebook, Evernote, Twitter, ...) that can be used both as a trigger and as action. The trigger is the "this" part of the rule. An example of a trigger is "Check in on Foursquare" or "I'm tagged in a photo on Facebook ". The action is instead the "that" part of the rule. Some example are: "create a status message on Facebook" or "send me a text message". Each user can create personal recipes, enable or disable them at their convenience. IFTTT runs a polling in accordance with a predetermined time to control the triggers and possibly trigger the actions. IFTTT then provides two services for end users. The first is to help them through a wizard when creating the recipe: users then do not need any knowledge of software development. The second service provided by IFTTT is the hosting on its servers for mashups created by users.

2.3 Semantic Web Services

A semantic description of a Web Service is required in order to obtain its discovery, its composition with other Web services and its implementation on the part of users and heterogeneous platforms. Existing technologies for Web services descriptions provide only the syntactic level, making it difficult for applicants (requester) and providers (ISP) to interpret or represent the meaning of the inputs and outputs or application constraints. This restriction can be relaxed by providing a rich set of semantic annotations that enrich the description of the service. A Semantic Web Service is defined through an ontology of service (service ontology) that enables the interpretation

by machines of its capabilities as well as integration in a knowledge domain. The infrastructure[8] for Semantic Web Services, as already said, can be characterized along three orthogonal directions: usage activities, architecture and service ontology. The usage activities define the functional requirements that a framework for Semantic Web Services must support. The architecture of a SWS describes the components necessary to achieve an activity defined by the SWS, while the service ontology aggregate all the concepts related to the description of a Semantic Web Service.

The publication (publishing) or insertion (advertisement) of SWS enables software agents or other applications to discover services based on their skills and their objectives (goals), a semantic register is used to record the instances of the ontology of the individual service. The ontology of the service must distinguish between the information that is used for matching during the discovery of the service, from that used for the invocation of the service itself. In addition, the domain knowledge (ontology) should be posted or linked to the ontology service. The discovery of a service consists of a semantic matching between the description of a service request and the published service description. Any queries that involve the service name, inputs, outputs, the precondition, and other attributes can be constructed and used to search the semantic registry. The matching can also be done at the level of tasks or objectives to be achieved, followed by a selection of the services that fulfill the given task. The degree of matching can be based on different criteria, such as the inheritance relationships between types: Inputs of type *Professor* for a service provider can "match as" a kind of Academic input of a service request. The selection of a service is required when there is more than one service that

corresponds to a given request. At that point may be used in non-functional attributes, such as the cost or quality, for the choice of the appropriate service. The composition or choreography enables SWS to be defined in terms of other services. A workflow that expresses the composition of atomic services can be defined in the ontology of service using the appropriate control constructs. This description may be based on a syntactic description such as BPEL4WS [9]. The invocation of a SWS involves a series of steps, once the required inputs were provided by a service request. First, the service and the associated domain ontology must be instantiated. Second, the input must be validated with respect to the types of ontology. Finally, the service can be invoked or a workflow can be run through the base provided. It is also important to monitor the status of the decomposition process and notify the applicant in case of exceptions or problems. Deploying a Web Service from a provider is independent from the publication of his descriptive semantics, since the same Web Service can perform different functions, but the architecture of SWS can provide help for the deployment of code for a given description semantics.

In recent years, many tools and frameworks have been developed that support the publication, discovery and composition of Semantic Web Services. These initiatives include OWL-S [10], WSMO [11], SWSF [12] and WSDL-S [13], but despite these, no tool or framework provides everything required for modeling platform for general Web Services ready for the Semantic Web. All these standards are still incomplete and may not meet the future demands of the industry such as increased complexity, scalability, reliability, to name a few. Moreover, the semantic information of a Web Service must be general enough to allow the support of automated interactions between Web

services and software agent. Ideally, the language of the Semantic Web Services should allow dynamism in all types of use of a Web Service, such as the selection, discovery, composition, invocation, negotiation and recovery after a failure. Furthermore it has to be extensible and tightly integrated with the knowledge resources of the Semantic Web. The next few sections of this article will address a list and comparison of existing languages and modeling framework of Semantic Web Services.

2.3.1 RDF

The Resource Description Framework (RDF) is the basic tool proposed by W3C for encoding, exchange and reuse of structured metadata and enables interoperability between applications that share information on the Web. The term "resource" is used from the beginning of the web to indicate anything available on the Internet through the use of its protocols and the generality of this term has encouraged a process of generalization methods of access to the resources themselves: from an initial idea (URL) of simply locating a resource, it has gone to the idea of being able to identify regardless of location (URI) and finally (RDF) to want to be defined by semantic connections. The basic RDF syntax provides the conceptual links among resources by defining predicates (or properties) that connect a subject and an object providing a means to build relational tuples. Each of the participants in these conceptual links is actually a URI and each URI can participate in other relationships, even with different roles. However, while RDF provides more syntactic details which are more subtle and powerful of the sole definition of semantic tuples, it is not enough to implement

features in the semantic web applications. To obtain a complete semantic enable we need to add special links to RDF: this is the purpose of OWL.

2.3.2 OWL

The goal of OWL (Ontology Web Language) is naturally not only to allow the attribution of meaning to resources (enough an efficient computerized vocabulary), but also to make these meanings computable, ie to allow automatic mechanisms (especially to computers or computer networks) to evaluate inferences about these meanings. A set of definitions that respects the syntax OWL is called ontology, for the love of a kind of metonymic brevity common jargon used for computer and mathematical formalism in general. RDF specifications, finally, allow to conceive open architecture and is easily understandable as the extensive use of URI allows any ontology using other ontologies already defined elsewhere. An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, ie facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using predefined OWL mechanisms.

2.3.3 OWL-S

OWL-S (Ontology Web Language for Services) is an ontology of services defined in Owl, developed by DARPA, to help to users and soft-

ware agents for the discovery, invocation, composition and monitoring of Web Services . This ontology has been submitted to the W3C in November 2004. The structure of Owl-s can be divided into three main parts:

- Service Profile for publication and discovery services; Process Model for the description of the operation of a service; Grounding to define the interoperability of a given service.

- The Service Profile, describes the three basic types of information: the organization that provides the service, what it does or what provides the service and other features of the service. The Service Profile is mainly used for the discovery of a service; the description of the service (and the query) is built from functional properties (such as inputs, outputs, precondition and effect - IOPES) and from non-functional (property interpretable by human users as the service name and parameters for define metadata about the service itself, such as the quality of service).

- The Process Model describes the composition or the orchestration of one or more services in terms of their constituent processes. This is used both to perform a reasoning on the possible compositions of services (for example to determine if a model is executable given a specific context) is to control the invocation of a service.

As described previously, the OWL language has three dialects according to a progressively higher level of expressiveness: OWL Lite, OWL DL and OWL Full. OWL DL is designed for maximum expressiveness

without losing computational completeness (it is guaranteed that all the implications will be computed) and decidability (all computations will be completed in a finite time) and is therefore the main choice when you are interested in having an efficient support systems of reasoning ("systems thinking"). Ontologies OWL-S are written in OWL DL, to support applications where the computational completeness is guaranteed.

2.4 Platforms for the Web of Things

The acceleration that we have seen in recent years toward the Web of Things was mainly due to the appearance on the market of board prototyping and development tool suited to the average user.

Arduino[6] is an open-source electronics platform based on easy-to-use hardware and software. Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

Tessel[7] is a microcontroller that runs JavaScript. Tessel runs JavaScript server side scripts. Just like web or mobile development, use your own IDE and libraries to program physical applications. Tessel supports modules that add new capabilities to the board and interact with the physical world from sensing to actuation to connecting with other devices, combining multiple modules for unique experiences.

In addition to hardware support consists of the prototyping board,

[6]http://www.arduino.cc
[7]https://tessel.io/

some platforms to support the Web of Things have recently appeared.

WEIO[8] is a Web of Things platform. It lets users connect and control their objects from any device using only the web browser. Connect easily objects between them or with Internet services like social networks, It's Node-compatible and ships with Wifi built in.

Xively[9] is IoT public cloud, web-based tools and developer resources empower organization, allowing customers to focus critical resources on connected product innovation rather than on enabling infrastructure. Xively's Platform as a Service (PaaS) provides the tools and services needed to create compelling products and solutions on the Internet of Things. Xively provides free, open and supported libraries along with tutorials and documentation to allow users to connect to Xively using the hardware they want and the languages they know. To make it even easier, the company certifies Xively Enabled hardware platforms every day from a variety of vendors. The libraries leverage standards-based API over HTTP, Sockets and MQTT to make connecting to the Internet of Things simple, intuitive and fast.

SmartThings[10] is a commercial solution for the Web of Things, which leverages on its ease of use. Users can purchase different types of smart objects (sensors, bells, smart lock). These objects communicate through a hub that has Internet connectivity and can be controlled by the user with a dashboard through a website or a mobile app. SmartThings makes it easy to connect the things in the physical world to the Internet: it allows to monitor, control, automate them from anywhere - at home, office, or on the go.

[8] http://www.we-io.net/
[9] https://xively.com/
[10] http://www.smartthings.com/

2.4.1 The COMPOSE Project

The COMPOSE [11] project aims at enabling new services that can seamlessly integrate real and virtual worlds through the convergence of the Internet of Services with the Internet of Things. COMPOSE will achieve this through the provisioning of an open and scalable marketplace infrastructure, in which smart objects are associated to services that can be combined, managed, and integrated in a standardised way to easily and quickly build innovative applications. The COMPOSE project builds upon existing European research projects and ongoing standardisation activities to provide a comprehensive marketplace framework that will be able to cover the whole service lifecycle by integrating a number of innovative technological enablers in a coherent way. The project will develop novel approaches for virtualising smart objects into services and for managing their interactions. This includes solutions for managing knowledge derivation, for secure and privacy-preserving data aggregation and distribution, and for dynamic service composition advertising and discovering objects' capabilities and service provisioning and monitoring.

The plan is to apply Web Technologies and to build a working implementation as a testbed for the ideas, and to use that to bootstrap a community of users and developers of innovative services. This can build upon a wide variety of existing standards, and this report provides a survey of uses cases, requirements, architectural concepts and technologies as a basis for identifying relevant standards and standards development organizations. The vision [14][15] of the COMPOSE Project is to advance the state of the art by integrating the

[11] http://www.compose-project.eu/

IoT and the IoC with the IoS through an open marketplace, in which data from Internet-connected objects can be easily published, shared, and integrated into services and applications. The marketplace will provide all the necessary technological enablers, organized into a coherent and robust framework covering both delivery and management aspects of objects, services, and their integration.

- Object virtualization: enabling the creation of standardized service objects Interaction virtualization: abstract heterogeneity while offering several interaction paradigms
- Knowledge aggregation: creating information from data
- Discovery and advertisement: of semantically-enriched objects and services
- Data Management: handle massive amounts and diversity of data/metadata
- Ad hoc creation, composition, and maintenance: of service objects and services Security, heterogeneity, scalability, and resiliency: incorporated throughout the layers

The COMPOSE project is expected to give birth to a new business ecosystem, building on the convergence of the Internet of Services with the Internet of Things and the Internet of Content. The COMPOSE marketplace will allow SMEs and innovators to introduce new Internet of Things-enabled services and applications to the market in a short time and with limited upfront investment. At the same time, COMPOSE will allow major European players in the information and communication industry, particularly cloud service providers

and telecommunications companies, to reposition themselves within new Internet of Things-enabled value chains.

2.4.2 The webinos Project

Webinos[16] is an Open Source Cross-Device Platform for widgets and mobile/web applications that allows developers to write applications able to run on multiple devices belonging to different domains (mobile devices, TV and automotive). In fact, the main goals of the project are applications' interoperability across devices and usability in order to create a multi-device user experience based on data synchronization and context-awareness taking into account the related security aspects.

Webinos provides a web runtime extension for browsers, which supports widget and web applications written with standard web technologies such as HTML, CSS and Javascript. *webinos* further provides a set of device-specific Javascript APIs to

- Provide access to hardware and software capabilities offered by a device such as address book, telephony manager, messaging manager, information about device status and so on.
- Access to capabilities on remote devices inter or intra Personal Zones.

The first characteristic allows developers to interact with the device, for example sending an SMS or getting geo location and contacts information using the set of Javascript APIs. The second characteristic represents the most innovative contribution of *webinos* and allows applications running on a device to use APIs provided as services by other devices. This mechanism will be further described in the rest

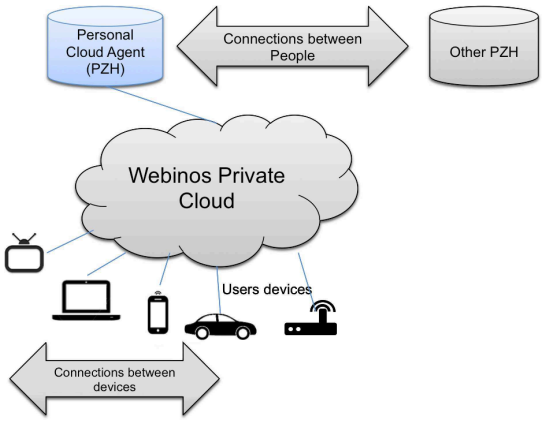

Figure 2.3: *An overview of the* webinos *architecture*

of this section along with a comprehensive description of the *webinos* architecture. *Webinos* introduces the concept of *Personal Zone* (PZ), defined as the set of all devices owned by a user. Each PZ has a main component called *Personal Zone Hub* (PZH), which is the point where the devices are registered and also provides data synchronization, communication among other PZs and secure access to the PZ from Internet. Multiple PZHs, one for each user, may also be linked together creating relationships among users as it happens in social networks. Figure 6.2 describes the overall *webinos* architecture.

Each *webinos*-enabled device placed inside a PZ has two main components called *Personal Zone Proxy* (PZP) and *webinos runtime* (WRT). The WRT represents the environment where the apps are executed. *webinos* provides two kinds of WRTs: the first is a browser extension for the execution of web applications, the second is a widget runtime for the execution of locally stored applications (widgets). *Webinos* provides a WRT version for each the considered domains

(mobile, PC, in-car units and home media), this means that the same application may run over all these domains without the need of a code refactoring.

The PZP connects the device to the PZH and enables the communication among devices inside the same PZ and exposes the *webinos* APIs. WRT and PZP act respectively as browser and local server, allowing each device to communicate with each other passing through the PZH (canonical way) or through a direct communication PZP-to-PZP in those situations where an Internet connectivity is not available. Also devices belonging to different PZs can communicate if their PZHs are connected. The PZH is responsible to issue identities (through PKI mechanism) and acts as messaging hub for devices and as a synchronization agent for data. User's data and services can be shared securely with other people connecting together multiple PZHs using a permission-based infrastructure. Both PZP and PZH represent the main components of *webinos* cloud architecture. Each user's content, such as an address book's contact, a calendar's event and so on, could be synchronized in every devices belonging to the user. Contents thus, are not related to a single devices but they are stored in the cloud. Although this concept is not too distant from Apple's iCloud, the most significant innovation provided by *webinos* is the possibility to share not only contents among devices but also services. In such way, devices belonging to different domains, with different OSs and produced by different manufacturers could seamlessly interoperate with each others.

Using *webinos*, users get all the benefits of a cloud platform with also the possibility to ensure privacy for their contents: *Webinos* also provides to each user the possibility to get all the benefits of a cloud

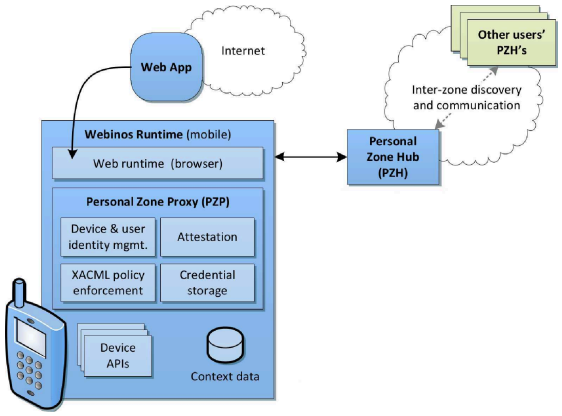

Figure 2.4: *Personal Zone Proxy and* webinos *runtime*

platform *Webinos* provides users with all the benefits of a cloud platform offering also the possibility to ensure privacy for their contents by setting up a PZH in a private device. Figure 2.4 shows a detailed representation of PZP and WRT modules placed inside each *webinos* enabled device.

Other components inside PZH and PZP, called managers, are responsible for authentication, policy management, context handling, messaging, etc.

The main characteristic, which differentiates *webinos* from other apparently similar platforms such as Phonegap or Titanium or even respect mobile operating systems like Android or iOS, is the possibility to consider each API as a service provided by the device. As a consequence of this approach it is possible to create applications by invoking API on devices different from the one where the application is executed.

One of the demos presented in the *webinos* context, which mainly

stands out the potentiality offered by the platform, is the *webinos Travel* application [17]. It enables user to manage his point-of-interests while a user is traveling. POIs are automatically synced between the user's devices. There is no 3rd party server integrated, where the information is stored. Syncing mechanism of the app is based on the *webinos* personal zone middleware. All data is owned by the user and resides inside zone. The application enables the interaction with the in-car navigation system. POIs can be pushed for guidance to the in-car navigation software. When the vehicle is parked, the smartphone can pick up the guidance.

CHAPTER

THREE

FROM USER GENERATED CONTENTS TO USER GENERATED SERVICES

3.1 Current Research Issues in User-Generated Services

The growing popularity of Internet-enabled devices and the consolidation of social networks have increased the amount of multimedia contents generated by users. Everyday people live a second life on social networks generating original contents such as pictures, videos, comments and so on [18]. Table 3.1 contains some statistics about user content generation.

This phenomenon has been encouraged by the spread of many kinds of Internet-enabled devices such as smartphones, tablets and entertainment devices.

Shipments of Internet-enabled devices are projected to hit 503.6

Table 3.1: *Statistics related to user-generated contents*

Average amount of tweets per day	190 million
Average pictures uploaded to Flickr per minute	3000
Total amount of articles hosted by Wikipedia	17 million
Total pieces of content shared on Facebook each month	70 billion

million units in 2013, up from 161 million in 2010. By 2015, however, shipments of Internet-enabled consumer devices are projected to break three-quarters of a billion units - at 780.8 million units - exceeding PC shipments of 479.1 million units [19]. Mobile devices give a new experience to users, offering them the possibility to obtain information about the surrounding environment through several built-in sensors (GPS, accelerometer, gyroscope). All these information let users create context-related contents, like geolocalized photos or tweets, which embed current user's position. A key role in this scenario is played by end-users, which are becoming the main contributors of the contents available on the web. The most likely next step in this direction will be the generation of services by non-expert users. Generating new service implies the creation of a set of API to interact with the service itself. According to the Service Oriented Architecture (SOA)

paradigm, a new service could also be generated by composing one or more existing services. The result of this operation is commonly referred as "mashup". In this paper, we want to emphasize that in a not too distant future, services will be not only generated but also provided by users, primarily through mobile terminals. In particular, we refer to common users who do not have an advanced computer knowledge. A series of both software and hardware resources are necessary in order to support the user in generating and providing a service, especially if this is provided by means of a mobile device. Devices such as smartphones or tablets have peculiar characteristics due to their portability and small size. Battery life, reception problems, reduced computational and storage resources are just an example of the limitations which characterize this kind of devices. In addition, issues related to the publication of a new service, its discovery, privacy and access control raise the need of a platform to support the user in the generation and supply of services through mobile devices. In this paper, we describe *webinos*, a cloud platform for running applications and services over heterogeneous devices belonging to different domains. In the following, we will show how *webinos* can be adopted to solve typical problems of generation and supply of mobile services.

3.2 User-Provided Mobile Services

The aim of this section is to explain what is meant by mobile services and then outline the main issues that there are when this kind of services is provided by users through their devices. We have already said that users are increasingly involved in the generation of

multimedia web content. The role of users gains even more and more importance also in the field of service generation. The emergence of Services Oriented Computing (SOC) allows end-users to develop applications by composing existing services. In this context, tools such as Yahoo Pipes [20] provide users the possibility to create own mashups composing web services. As a result, the Web is rapidly progressing towards a highly programmable platform and end-user programming has become a very popular and common trend nowadays. This enables end-users to take advantage of different Application Programming Interfaces (APIs) to create and publish their own contents and services. Major companies like Facebook, Google and eBay have already provided interfaces to their services extending their market possibilities. In this article, we focus on those services generated by users based on other applications or services provided by other mobile devices.

Mobile services are those services designed to be accessed through mobile devices. Their main aspect is the mobility for what concerns both their invocation and their supply. The difference with traditional services is remarkable: a service that allows a user to view bus timetables can be provided through a web site and can be accessed in the same way on a personal computer or on a smartphone. The same service designed to be used on the move will take into account the user's context. For example the mobile service could give information for only those buses which route is close to the user's position that can be obtained through smartphone's GPS.

The potentialities of mobile services are huge. To date, there are already many context-aware applications for smartphones allowing users to benefit from mobile services. Considering the evolution of user's role from consumer to producer of content and services, is presumably that

in the next few years, the average user will be able to create applications for his smartphone making a mashup of services also offered by other devices. As an example, suppose that the mobile phone owned by an elderly person provides the ability to be managed remotely. In this way, using this "device ability" a more experienced user could help the elderly to perform operations such as the remote phonebook's management.

There are several issues to consider in the creation and sharing of services across multiple devices. In particular, there would be the need of:

- A protocol to describe services and their exposed features.
- An access control mechanism to specify, through policies, the access / composition constraints of each service.
- Hosting environments (service providers) where to run services.
- Repositories where services have to be registered.
- A discovery mechanism to retrieve services (eg. by exposed features).
- A toolkit to help users to create, deploy and manage services.

In the next sections, we will give an overview of the state of the art in the field of user-generated mobile services. We will also present the *webinos* platform and how it can help to satisfy the aforementioned requirements.

3.3 Related Work

The scientific interest about User Generated Service (UGS) and User Generated Content (UGC) fields is growing in these last years. Zhao et al. present in [21] a comprehensive survey of current state of art in UGSs. They give the specific description of UGS by comparison with the concept of UGC, and then go through different technologies to analyze the challenges of UGS describing advantages and limitations of each approach. Jensen et al. describe in [22] some guidelines to support users creation and management of services. Tacken et al. investigate in [23] the state of the art and the requirements to let the vision of the super prosumer concept become true. They review the current technologies for an easy creation and discovery of mobile services and list the identified requirements for user generated mobile services. In [24], authors discuss the concept of mobile-services generated by the user itself. They investigate some conceptual requirements and concluded with an architecture proposal for IT service providers. Authors also provide a proof-of-concept system development performed within the European-funded project *m:Ciudad*. The European FP7 research project *m:Ciudad - a metropolis of ubiquitous services* - aims at the empowerment of users to create services on mobile terminals. The project demonstrates various scenarios in which users either act as creator of services or interact with the system to search for services or service construction components. *m:Ciudad* envisions a system for service providers, which enables a mobile user to create and consume mobile services on the fly on his mobile device. *m:Ciudad* architecture is exhaustively described in [25]. In the next section, we are going to introduce another European funded project called *webinos*.

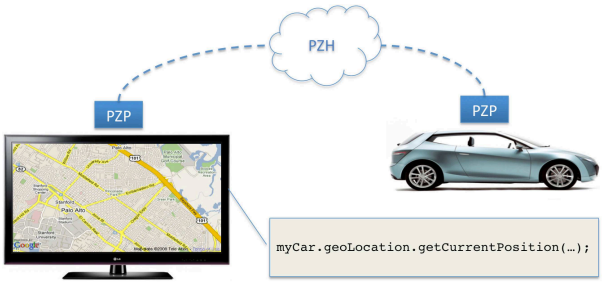

Figure 3.1: *An example of using "API as service"*

In particular, we are going to describe how *webinos* can be adopted as a platform to allow mobile service to be created and shared among users. The main advantages of *webinos* compared to other platforms will be discussed.

3.4 *Webinos* as a platform for User-Provided Mobile Services

Webinos introduces new scenarios for the generation and sharing of mobile services. Figure 3.1 shows a use-case where user has registered a personal computer and a car inside his PZ. Each of these devices has a PZP, which implements and exposes the *webinos* geo location API. In the case of the example, a user is watching his car's position through an application running on his PC, which uses the geolocation API provided by the car. Thus, each *webinos* API implemented by a PZP can be considered as a service provided by a device. The PZP then turns each device in a server able to accept requests from other devices

Webinos provides both the mechanism for dynamic registration of new services and for discovering these services by searching the devices able to provide them. For example when a new device is added to a user's personal zone, the PZH registers all the services exposed by this new device and makes them discoverable, or not, according to the security policy set by the user. All services provided by devices registered inside a PZ could be retrieved using the *webinos.discovery* API. We have said in the previous section that *webinos* provides the possibility to connect each other multiple PZH. Each PZH represents a user and his devices. Linking together multiple PZH means that when a user search for a service (for example the geolocation service) his PZH will query not only devices inside his PZ but also those devices belonging to linked PZs. *M:Ciudad* project considers only user generated services provided by smartphones, *webinos* instead takes into account different domains such as automotive, home-media devices and even smart objects belonging to the domain of Internet of Things. Especially in the case in which more PZHs are mutually connected, a mechanism for controlling access to services is of fundamental importance. Each PZP in fact, has an access control module based on XACML [26] specifications, which checks whether the request from an external device to a certain API may or may not take place.

Besides the possibility of calling APIs as services provided by other devices, *webinos* offers the possibility to create applications that can communicate with other applications installed on different devices. The *webinos* App2App messaging API specification defines interfaces to create, send and receive messages between applications in the *webinos* system. It provides generic messaging primitives, which can be applied in different application scenarios. The messaging is indirect,

meaning that applications do not directly address each other but use a channel to route the messages to connected applications. A unique namespace (within a PZ) is used as a key to find and connect to channels. This API can be used by third-party application developers to implement custom message-based protocols by taking advantage of the features offered by the *webinos* message handling system and overlay networking model. The App2App API represents a starting point to allow the creation of new applications in the form of services, realized as a mashup of existing other services.

The possibility offered by *webinos* application to call an API exposed by another device may give rise to some problems of content management. Suppose that an application running on Alice's tablet was able to access the *webinos* Contacts API provided by Bob's smartphone to read and save locally Bob's contacts. In this case, which assumes that Bob had given access control rights to Alice, privacy concerns may arise if a third person, such as Carol, uses the Contacts API provided by Alice's tablet to read Bob's contacts.

Our future work will be exploiting the potential of *webinos* and in particular of the App2App API in order to make it possible for users to create and share *webinos* services obtained from the composition of services provided by multiple devices. In particular, we would like to

- Extend the registration and discovery mechanism to ensure that each new service created is associated with semantic information.
- Extend the current security mechanism in order to solve problems related to data handling and privacy of contents.

CHAPTER

FOUR

THE WEB OF THINGS: DEALING WITH EVERYDAY OBJECTS

4.1 User-Objects Interaction

With the increase of sensors and actuators connected to the Internet and the spread of technologies such as RFID, NFC or visual tags, the interaction between users' devices and real-world objects is gaining more and more attention. In a not too distant future, every object will have its virtual counterpart in the web that will provide services and augmented information. Although the increase in the number of mobile devices, such as smartphones and tablet, seemed disruptive in recent years, it will be nothing compared to the spread of smart objects that will occur in the coming years. The number of things connected to the Internet will be soon larger than the number of people which use them. We are immersed in a continuous flow of data generated by

users' devices and common objects that will bring soon new and more complex interactions.

Through the introduction of more mature and cheap technologies, the spread of IoT will be prominent in the domains of Smart Cities and Home / Building Automation. An often cited example is the smart fridge which will be able to: check the status of foods, take an inventory of what the fridge contains and send a notification with a shopping list for the missing products.

'Internet of Things (IoT) is a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'things' have identities, physical attributes and virtual personalities and use intelligent interfaces and are seamlessly integrated into the information network." [1] According to the Internet of Things (IoT) 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: i) unlike a website, only a few people should access objects inside the house, ii) 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, iii) the web pages are made with standard technologies, the real objects instead are produced by different manufacturers and with different specifications. The main scenario of the Internet of Things is one in which the objects of everyday life can be controlled by the user via the REST web services. These objects are then defined "smart objects" because they must be provided with a minimal

[1] Definition by ITU and IERC-Internet of Things European Research Cluster

processing capability and at least a communication interface.

A fundamental aspect of IoT has always been 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 object.

The objects that we consider are those that can be found in the domestic environment: such as ovens, air conditioners, TVs, media players. 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 such as house hold 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. In a future where smart objects can be controlled through web services, it will be crucial to find the best way by which users can interact with these objects. 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.

Considering smart objects as service providers may lead to some new issues never considered before. The possible huge amount of heterogeneous smart objects connected to the Internet would complicate the discovery and use of the services they expose. Traditional web search engines represent a limit for discovery of smart objects since they require a mechanism for understanding their capabilities and

functionalities which users can exploit. Another important aspect is to consider 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 Internet 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 objects: 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 [27]. If a user has a smart object, it should be himself to decide who can access to it. So, in a IoT environment it's 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 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. Each webinos enabled device can expose its capabilities as services. What we propose in this paper is an extension of the webinos platform by introducing a new API for controlling smart objects through web applications. We finally present a testbed application which, using the augmented reality and the proposed API, allows user to identify and control real objects considering the home automation scenario.

4.2 Related Work

The scientific interests about the Internet of Things have been going on for more than ten years, since in the 1991 Mark Weiser introduced for the first time the concept of ubiquitous (or pervasive) computing [28]. From that moment many things have changed and many efforts have been made to accomplish what Weiser had only thought.

Research on IoT leads to talk about "things" equipped with sensors for data acquisition, actuators to perform some operations and low computational capacity. Over the years, many researchers have discussed on which features a smart object should have and which activities it should be able to run. Obviously many of these definitions are strictly dependent on technological progress. Lev Manovich [29] describes smart objects as "objects connected to the Net; objects that can sense their users and display 'smart' behavior".

Thompson *et al.* [30] list the some requirements that a smart object should provide.

- communications: to send and receive queries or commands;
- identity and kind: every smart object should be able to be identified in order to understand what are its skills and should be able to self-describe its capabilities when the other smart objects ask its to identify itself;
- memory and status tracking: the smart objects should have persistent memories;
- sensing and actuating: to understand what happens in the environment and to act accordingly;

- reasoning and learning: a form of intelligence which could be not necessarily sophisticated.

Kortuem *et al.* [31] have identified three canonical smart object types that represent fundamental design and architectural principles: activity-aware objects, able to collect data about the environment or their own use but don't provide interactive capabilities; policy-aware objects, able to interpret events and activities with respect to predefined policies; and process-aware objects, that are the most complete of the three object types because they create a context-aware workflow model that defines timing and ordering of activities.

Around the concept of smart objects, many architectures, that allow users to use different types of smart objects, were proposed. Guinard *et al.* [32] contributed to a step towards the definition of the Web of Things by creating RESTful APIs to integrate the services offered by devices and objects in the real world such as wireless sensor networks, embedded devices and household appliances with any other Web content. They propose two ways to integrate devices to the Web using REST: direct integration and a Smart Gateway-based approach for resource-limited devices. Mingozzi *et al.* [33] propose the BETaaS platform for virtualization of real things as services. The platform exposes to applications a unified service-oriented interface to access physical objects regardless of their physical location and physical model. Estrada *et al.* [34] have realized UbiSOA (from Ubiquitous Service-Oriented Architecture), a platform for building smart environments using IoT technologies, and sentient visors, which are systems comprised of user devices and specialized services that together allow users to interact with their environments. UbiSOA provides three

basic mechanisms: discovery of services, for the detection and identification of components at runtime; common messaging with semantic contents; and event notification, to allow an application to respond to changes in the environment. Sentient visor uses the concept of augmented reality to enable users to interact with objects and obtain relevant information superimposed on the screen of user device. In our previous work [35], we proposed the adoption of Webinos [16] as a viable platform for WoT. We have shown how Webinos enables virtualization of real objects in the form of services that can be used by web applications using the API. In particular, in that paper we have explained the webinos APIs for generic sensors and actuators.

Son *et al.* [36] highlight the need to use semantic communication between smart objects to discover and identify who they are and what they can do so that they can be able to collaborate together. Semantic communication should be leveraged with visual identification which lets to know what the user is aiming at. Information about objects displayed by augmented reality is enriched with contextual information that provide personalized contents to users.

4.3 A webinos API for smart objects

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: i) belongs to the same personal zone, ii) 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, geo location, 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 it is based webinos platform, 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:

- 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.

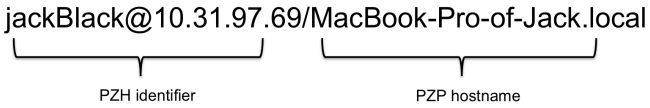

Figure 4.1: *Webinos Service Address Composition*

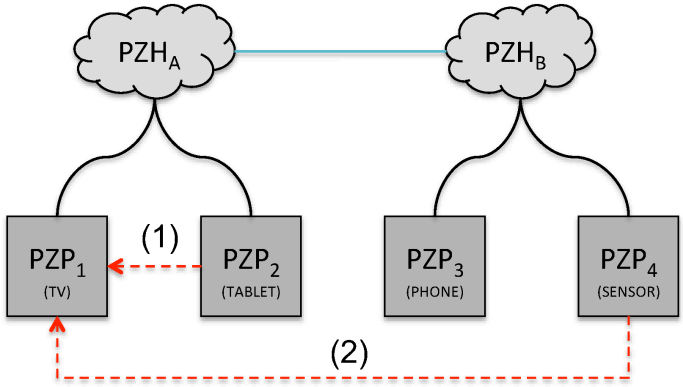

Figure 4.2: *Intra and Inter Zone communication*

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 each 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.

In [35] we discussed how the webinos platform can well suit the IoT world. We argued about the APIs webinos provides to deal with generic sensors and actuators. Every sensor or actuator is virtualized 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.

The interaction between users and smart objects will enable very important scenarios in the coming years. The reduction of the cost of technology will facilitate the deployment of smart objects, catching on the home environment. Some companies including Samsung, Qualcomm, Ericsson are moving in this direction by developing proprietary protocols for their products. Defining the functional requirements that a platform for smart object must offer will constitute an important research activity in the near future.

In the first part of this section we are going to describe the main features that make webinos able to support the interaction between users and smart objects. To achieve this, we have introduced a new API to allow web applications to communicate with objects. This API will be described in the second part of the section.

4.3.1 Why is webinos 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 it will include smart objects such as the fridge or the 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. 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 authorize 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.

4.3.2 Smart Object API

In the previous section we have shown how webinos well suits the IoT domain by defining generic APIs for sensors and actuators. However, what we propose in this paper is something different: in order to bridge the gap between real objects and their virtual instances we have a support for smart objects. The webinos platform is fully extensible by adding new APIs (or even new drivers) to support new devices. Hence, one possible approach to handle smart objects such as household appliances, 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 two public methods:

- *getMethods*: returns a description of all the functionalities provided by the object in a well defined JSON format. The description comprises information about the method itself (name, human readable description), and information (type, accepted values, etc.) about all the input and output parameters which the method expects.

- *callMethod*: is used to invoke a specific method (functionality) of the object.

- *callAsyncMethod*: is used to invoke a specific method (functionality) of the object. The result of the operation will be sent back as a callback function's parameter.

For the sake of clarity, let's think about a smart calculator which implements the webinos API for smart objects. The manufacturer should provide some information about the object in a JSON array with the description of its methods:

```
exports.name = ‘‘calculator’’;
exports.description = ‘‘a simple calculator’’;
exports.type = ‘‘office.calculator’’;
```

```
exports.definition =
{
  "swagger": 2,
  "info": {
    "version": "1.0.0",
    "title": "Smart Calculator API",
    "description": "An API to control a smart calculator",
  },
  "host": "http://localhost:3000",
  "basePath": "/calculator",
  "schemes": ["http"],
  "consumes": ["application/json"],
  "produces": ["application/json"],
  "paths": {
    "/moduloMath": {
      "get": {
        "description" : "Finds the remainder of division of
            one number by another",
        "parameters" : [
          {
            "name": "dividend",
            "in": "query",
            "description": "The dividend number",
            "type": "number"
          },
          {
            "name": "divisor",
            "in": "query",
            "description": "The divisor number",
```

```
          "type": "number"
        }
      ],
      "responses": {
        "200": {
          "description": "The result of the modulo
              operation",
          "type": "number"
        }
      }
    }
  }
}
```

Each smart object must provide a private implementation of all the exposed methods:

```
exports.moduloMath = function(params){
  try{
      var res = params.dividend % params.divisor;
      return {
        results: [
          {
            moduleResult: res
          }
        ],
        status:{
          code : "200",
          type : "success",
```

```
        message : "Operation has been successfully
            completed"
      }
    }
  }catch(e){
    return {
      results: [
        {
          moduleResult: undefined
        }
      ],
      status:{
        code : "400",
        type : "error",
        message : "Invalid operation"
      }
    }
  }
};
}
```

In this example we suppose that the calculator has a method called "moduloMath" which accepts two input parameters ("dividend" and "divisor") and returns as output a JSON object which contains the result of the operation. A web application which wants to invoke the modulo operation from this smart calculator has to discover the "calculator" service through the webinos Discovery API, and then has to call the desired functionality using the *callMethod* method:

```
// calculator refers to the smart calculator object
```

```
var params = {dividend: 5, divisor : 2};
calculator.callMethod('moduloMath', params);
```

Summarizing, the proposed smart object API allows applications, and therefore users, to control all the real objects that implement the API. The objects capability to expose their functionality as services, can be used not only to make them controllable by users, but also, in the future, to ensure that multiple objects can cooperate among themselves and autonomously carry out their tasks.

4.4 Proposed Application

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 services and functionalities that object is able to provide. This description is provided by the smart objects API in the form of a JSON object. Using a proper transformation algorithm, the application uses the in-

formation received from the object 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 an easier and more 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 is capable of providing, and use them, without the burden of reading complex instruction manuals. Using augmented reality we want to simplify the interaction between user and real objects and provide the same user-experience of web applications. Azuma *et. al* [37] 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. The application uses an ArUco [38] marker to identify objects. Each marker is a 5x5 matrix where each row is composed by 2 bits of data (in green on the

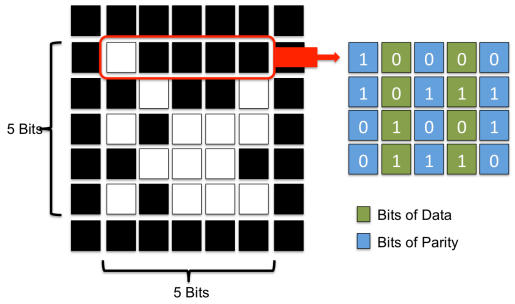

Figure 4.3: *ArUco Marker*

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 to the example of the smart calculator described in the

previous section, we show in the 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

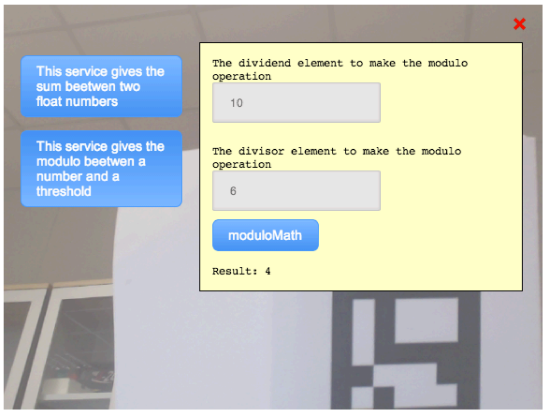

Figure 4.4: *Graphical user interface for modulo operation implemented by a smart calculator*

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 interpret 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 "callMethod" function is called on the calculator's service, passing the name of

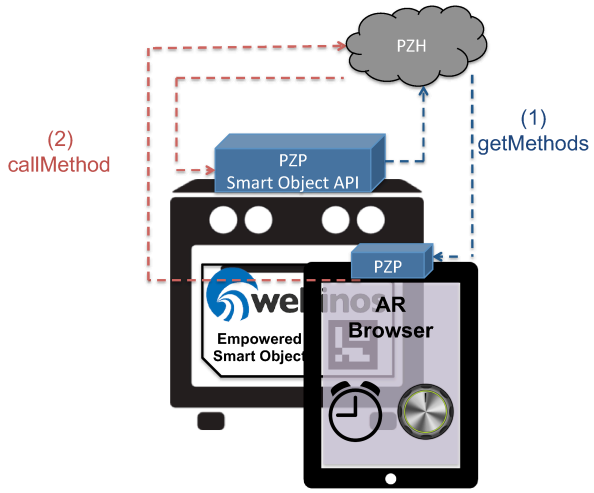

Figure 4.5: *The proposed AR application to control smart objects*

the method to call and an object which contains the input parameters.

```
calculator.callMethod('moduloMath', {dividend: tf1.value,
    divisor: tf2.value}, successCB, errorCB);

function successCB(output){
  if(output.status.type=="success")
    divResult.innerHTML=(output.results.moduleResult);
}
```

Once the required operation has been carried out by the object, a callback method (success or error) is executed. The success callback receives as input parameter an object which contains the final result of the operation and information about the outcome. If the status has no errors, then the current value of the operation is shown to the user.

The application can be used for more concrete and real scenarios, such as the domotic. 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 stuck 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 object which is currently framed. Figure 6.2 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.

4.4.1 Improving the scalability using Vuforia SDK

Currently add a new object means to include it in the personal zone, print qrcode with the id assigned to the object and place the qrcode on the object itself. This approach is not very scalable and requires

a commitment on your part. For this reason we have improved the application by using an augmented reality framework called Vuforia.

Vuforia™ is the software platform that enables the best and most creative branded augmented reality (AR) app experiences across the most real world environments, giving mobile apps the power to see. The Vuforia platform uses superior, stable, and technically efficient computer vision-based image recognition and offers the widest set of features and capabilities, giving developers the freedom to extend their visions without technical limitations. With support for iOS, Android, and Unity 3D, the Vuforia platform allows developers to write a single native app that can reach the most users across the widest range of smartphones and tablets. Vuforia first detects feature points of the target image [Web-based target management] and then uses the data to compare the features in target image and the frame received by the camera. In addition Vuforia provides support for Unity 3D framework which can be used to build attractive UI which can both help and involve users in the way they interact with smart objects.

CHAPTER

FIVE

THE COGNITIVE INTERNET OF THINGS: HOW THE ROLE OF THE USER IS GOING TO CHANGE

5.1 New Kinds of User-Objects Interaction: the case of Machine to Machine

The first concepts of pervasive Internet were developed in the early 90s by the pioneer Marc Weiser at Xerox PARC in Palo Alto, but it is only in the few last years that the interests of the academic world and the most important technological players have been focusing towards this topic, thanks to the emerging of a new context which goes under the name of Internet of Things (IoT). The IoT paradigm is based on the concept of URI (universal Resource Identifier): this is a way to uniquely identify an object in a network structure. Internet of Things

was initially focused on technologies for identification and tracking of real objects in the industrial field such as RFID but it is quickly spread over a wide range of sectors, from domotic to industrial automation. Moreover, recently, Bluetooth low energy (BLE) and Near Field Communication (NFC) technologies, has opened completely new scenarios.

Over the years the scientific progress on the miniaturization techniques for integrated circuits and the relative cost reduction for electronics components has been fostered the appearance on the market of the first microcontrollers which can host small web servers and then provide RESTful services: the objects become part of the Web and can be controlled by the user remotely through REST technology. The IoT paradigm evolved in the Web of Things (WoT) whose transition was undoubtedly accelerated by the appearance on the market of easy to use and low cost prototyping boards. One of the most famous of these board is Arduino, an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. Arduino can be used to create applications for home automation that are managed by the user's smartphone. Today the interest around WoT is constantly growing: several companies are beginning to put on the market proprietary systems which allow users to interact with everyday objects. However, the absence of a common international standard has made it difficult to adopt a single platform to guarantee the interoperability with objects produced by different manufacturers.

A further IoT evolution which is gaining more and more interest is the Machine to Machine (M2M), where objects are able to interact each other without a direct intervention by the user[39, 40, 41]. The machines typically considered in M2M applications are often simple

sensors or actuators, but from the point of view of the present paper, we identify the machines involved in M2M systems with the so-called "smart objects". Smart objects can be defined as autonomous, physical/digital objects augmented with sensing/actuating, processing, storing, and networking capabilities. Common objects from daily living such as TV, air conditioners, oven and in general all the home appliances inside our homes will become in the near future smart objects, and enable new application scenarios where they will interact with each other, under user control or autonomously. The main application fields of M2M are related to industrial automation, however today we are assisting to the transition of the technology from factory to the smart homes. The home automation involves areas such as lighting control, energy management, remote home management, assisted living. Although this field is not new, it has not been successful due to the high cost for installing a domotic system, which is probably caused by the lack of a de facto standardization. Several attempts to realize an easy, scalable, simple to use system was made over the years, but there isn't till now a solution universally accepted by the major players.

in the future,the smart objects will surely be Home automation protagonists. In facts, thanks to their ability to interact with the physical world they let our homes to become "smarter" and help us to improve our life quality. In this sense, the role of users and the way they interact with objects will change: while currently they use to control the surrounding objects (e.g. through smartphones), in the future they will only supervise objects interactions, which will take place automatically i.e. the user merely expresses a "goal", leaving to objects the responsibility to coordinate themselves to reach the given

goal.

In this paper, we will consider the adoption of the M2M into the Smart Home scenario, showing the main issues that affect the design of a M2M home automation system, and proposing a cloud-based goal-oriented architecture which, by exploiting semantic and location awareness, helps smart objects to autonomously carry out complex tasks.

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. Therefore each object must provide a description of all its capabilities, in a machine-readable form to be understood by the other objects. In this context, a good approach is the adoption of techniques which use semantic for describing smart objects. Semantic technologies have been introduced in the last years on the web to give a semantic meaning to the contents, with the aim of converting the current web, where data are mainly unstructured or partially structured, in a web 3.0 where data are highly structured and can be processed by computers. In the smart objects context, it is possible to use these techniques - by defining ontologies - to describe the meaning of the services that each object is capable of delivering. Associating each object's functionality to an ontology is very important, since these functionalities will be used by the objects themselves to communicate each other in order to perform

complex tasks. For example, if a smoke detector has to notify that the smoke level is above the threshold, it should contact an object which can alert the user (for example a buzzer or whatever is able to emit a sound). Using a semantic reasoning, it is possible to autonomously understand which objects have the capability of "alerting".

Another important aspect to take into account concerns the position of the user (which is, for our purposes, identified by the smartphone) inside the environment. Being able to figure out when the user is inside a particular section of the environment (e.g. inside a particular room) is very important for the system to be able to meet a goal which is related to the user position. Moreover, in a goal-oriented architecture, it is not necessary an user indoor localization system which is "always on": it is the system that determines when the position is needed, and activates the user localization process in order to perform a particular task. This kind of approach lets the system to reduce power consumption.

In the following we give a brief summary of several indoor user localization techniques which we examined for this article:

- *Audio Systems:* exploit some characteristics of sound signals to locate a smartphone inside a building. They are low-cost systems, and usually they do not need to deploy infrastructures inside the building[42, 43, 44].

- *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) by measures the RSS values and compares 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 natively support the bluetooth low energy. 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 micro-location in an accurate way: when the device gets within 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. Many manufacturers are currently producing devices iBeacon-enabled.

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. It was found out that some search engines allow to find on the Internet all those objects that expose a Web server and do not have any form of protection. In this way it is possible for an attacker to access, for example, to a private webcam placed in the homes of unsuspecting users which can be spied. 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 problem in relation to the safety (non only the security) of users. While a privacy fault in a social network can lead to the inadvertent spread of contents, on the other hand, a wrong security configurations of the objects inside a smart home can allow strangers to attempt the safety of the

users. It becomes clear that is absolutely necessary that these objects implement some sort of security mechanisms (eg. firewall whitelist or by exchange of documents) to avoid unwanted intrusions, but most of the time the object has limited resources in terms of memory and processing capability, therefore it is impossible for it to handle complex algorithms such as cryptography. 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

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 assumption that users interacts (almost) directly even with a Smart Home management system to fulfill 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 [45], SmartThings [46] and Staples [47]) 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 [48] or LG [49] promise to give a seam-

less 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 [50] 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 [51] 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 [52, 53, 51, 54, 55]. 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 [56] 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 [57] 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 players 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: "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 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 [58] from Microsoft Research Lab and Linux MCE [59] these projects represent the growing interest in this area.

5.4 Architecture Description

The architecture proposed in this thesis 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. Generally speaking, a smart object can be defined as an item equipped at least with:

- Sensing and/or actuating capabilities: these allow the smart object to interact with the physical world, by sensing something from the environment and by doing something else when certain conditions occurs.
- Microprocessor: it enables the smart object to elaborate the data received from sensors and to send commands to actuators in order to perform some tasks.
- Power source: most of the time is a battery, and it is used to power the electric circuits.
- Communication device: it is typically a low power radio communication system, that give to the smart object the ability to

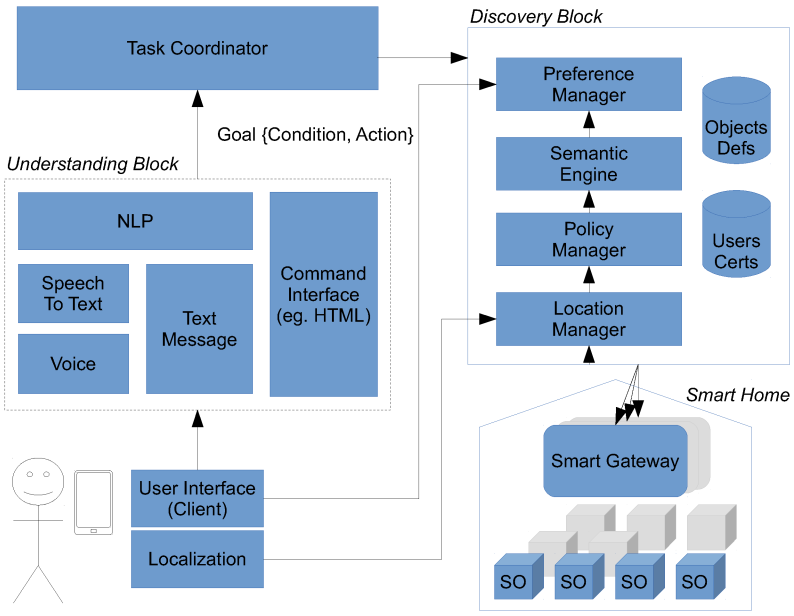

Figure 5.1: *Enabling M2M in smart spaces: the proposed architecture*

communicate each other, with the internet, or with a gateway.

We can identify, according to their roles, two kind 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 gateway objects: are deployed one for each room in a known position, and are a sort of central units which, when the indoor positioning module is waked up, perform the advertising

operation, in order to communicate their position to the smartphone (and then the user) that is inside the room.

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 it 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 understanding block contains a Natural Language Processing block (NLP). Otherwise data are translate through a command interface (e.g. HTML).

The outdoor/indoor localization information collected by the smartphone are sent to the discovery block and elaborated by the location manager which is responsible for enabling the indoor localization system to locate the user inside the house. The indoor localization process is better explained in section XXX. Once the location manager obtains the information about the position of the user inside the smart home, it can communicate it to the

The cloud blocks, namely “understanding block”, “discovery block” and “task coordinator block” 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 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. 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, selction 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" block, 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 Stanford Parser[2], the command "*Start the washing machine with the program J at 8:00 PM if the washing machine is full load*" is split into its main components:

```
(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 output provided by the understanding block to the task coordinator is a JSON object which contains information about the goal, split into an *action* (A) and a *trigger* (T).

```
A: {
    verb: 'set',
    what: 'conditioner',
    where: 'living room',
```

[1]http://www.abisource.com/projects/link-grammar/
[2]http://nlp.stanford.edu:8080/parser/index.jsp

```
    how: '25 C',
    for: '1h'
}

T:{
    combine: 'and'
    matches: [
        {
            match: {
                what: 'washing machine',
                where: 'bathroom'
            },
            func: '=',
            value: 'loaded'
        }
    ]
}
```

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.

The part of this system that cares about "understanding" provides the task coordinator with some structured information that describe user defined goals. 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).

There are three main types of triggers:

- Time-dependent: the action will be carried out at a certain time (eg 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 and the methods and parameters useful to retrieve them. 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 method to invoke (e.g. getTemperature) and an array (empty in this case) containing the types of the needed parameters.

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

3. Requires from the Discovery Block, when the condition associated with the trigger occurs, the object that can carry out the action and the information to invoke it.

4. Performs the action on the latter object by calling the appropriate method.

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 control units placed in every room, can notify the Task Coordinator of his presence in a specific room.

5.7 Discovery Block

The role of the Discovery block is to provide to the Task Coordinator those objects able to perform a certain task. Since we are considering a smart space, the selection of these objects depends on several factors: the position where the object is located, the possibility that an object to perform an operation,

The *Bootstrap* phase occurs when an object for the first time makes its entry into the system. This phase requires user interaction: once

activated, the object must be configured through a direct wifi connection between user's smartphone and the object itself. The configuration will consist of specifying the credentials to access the wifi network of the smart space and assigning the room in which the object is located. During the rump bootstrap, the object notifies the system of the operations that is able to fulfill: these operations are described by means of RDF ontologies of which will be discussed later. The user then has the fundamental task of configuring the new object and then "accept" in the system.

5.7.1 Location Manager

The location manager is the block which is responsible for locating the user, who 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 an 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 $500mt$ 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. $500mt$), it put itself in the discovery mode and send this information to the location manager which elaborate it and activate the indoor localization system of the smart home, by enabling the smart gateway objects (if they are not already enabled because of another user of the smart home) to advertising its known ID and position. 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. The user enter inside the room for which the task was set. Because the smartphone is in discovery mode, and the smart gateway objects are in advertising mode, the smartphone can sense the smart gateway object of the room in which it is inside. It send the information to the task coordinator which is responsible for performing the previously set task and related to the user position (e.g. turn on the lights on the room).

Another responsibility of the location manager is to determine if a smart object it is available at the moment of the task execution, or if it is not reachable because it is turned off or it 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 eliminate it from the list of smart objects which are involved in the task execution and which must be passed to the semantic manager for the next steps.

Speaking about the smart homes of the future, it is clear that the problem of the indoor localization of the user is one of the major challenges to be addressed: as we said above, knowing the user position let the domotic system to perform position-related tasks such as turn on lights in the room on which the user is inside, or turn off particular devices, etc. Moreover, in order to save energy, the home automation system should be "smart", by activating the indoor positioning system of the smart home just when it is needed and by deactivating it when the task is over.

Think about the architecture proposed in the present paper, we need an indoor localization system which is capable to detect when a user enter inside a room. Because in our approach the user is identified by the smartphone, the problem become to understand when the the smartphone is inside the 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, e.g. 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: the smart gateway object can act as iBeacons and advertise his ID (which is associated to a known room) when the smartphone is in the set proximity range of the smart home. The smartphone can put itself in discovery mode and sense the iBeacon signal.

5.7.2 Semantic Engine

The semantic engine is the most complex part and at the same time the most important discovery of the macro block. It acts in two different phases:

- during the object's bootstrap: the semantic engine is responsible to keep the ontology updated. Whenever a new object performs the bootstrap, the semantic engine is notified with information about the description of its services and its relative position in home, set by the user through the appropriate application of control;

- during a user's request to perform a task: the task coordinator needs to know what object can perform the goal or unlock the condition. The semantic engine is interrogated by the task coordinator during the research phase of smart objects.

For building the ontology is needed information about the features of the objects and the room in which they are located inside the house. In the proposed architecture, every object has got a file that contains a description of its services enriched with semantic information. The file has to be built directly from the manufacturer and follow the guidelines analyzed below. For example, we suppose a user has got a washing machine that exposes a service that lets us know if the object contains clothes to be washed, i.e. if the washing machine is loaded. In order for this method can be invoked, the semantic engine has to know that it exists, what it does and on which physical variables it acts. Therefore a description of the service in a standard way that can be analyzed and interpreted by a machine is necessary.

The code shows the format of the service description we have used:

```
"paths": {
  "/temperature": {
    "post": {
      "description" : "Sets the temperature to achieve",
      "parameters" : [
        {
          "name": "temperature",
          "in": "body",
          "description": "The temperature to achieve",
          "required": true,
          "schema" : {
            "type" : "number",
            "minimum" : 14,
            "maximum" : 35
          },
          "what": "NS:Temperature"
        }
      ],
      "verbs": ["set", change", "alter"],
      "what": "NS:Temperature"
    }
  }
}
```

All the descriptions from objects are transformed by the semantic engine to RDF tuples which are merged to build up an ontology for the smart space. Figure 5.2 shows a graphic representation for a smart space's ontology.

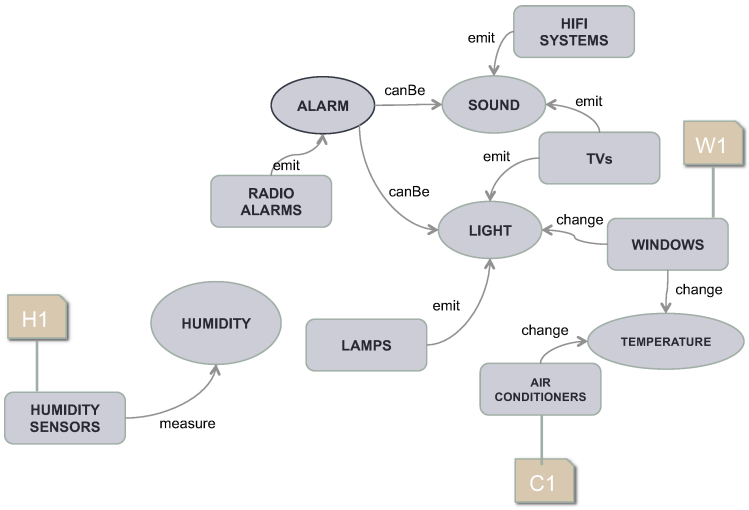

Figure 5.2: *An excerpt of a smart home ontology*

The ontology is used by the semantic engine to derive information about what objects can directly or indirectly do. Lets think at a radio alarm whose main functionality is to wake up the user. An *alarm* could be either a *sound* or a *light*. According to the ontology, an HiFi system and a TV con emit sounds: therefore they can be used to wake up the user when the radio alarm is not available. The ontology helps the system to find out objects' indirect capabilities.

5.7.3 User Preferences

As already mentioned, in typical CIoT scenarios the user has a different role than the one he holds in the Web of Things: he does not directly control objects but supervises their work. This supervision firstly includes the generation of a goal that objects will have to com-

plete by interacting with natural language or through ad hoc UI. We have seen how this is carried out bu the understanding macro block. Apart from the generation of a goal the user can help the system to choose the most suitable objects to perform a given task. These preferences can be taken into account characteristics such as efficiency (the time required by an object to perform its tasks), effectiveness (the quality with which the task is executed) or criteria for access control (not all users of the smart space can have access to all objects). The two blocks described below are used to support the user to express the criteria with which the objects are selected at the moment in which they will be called upon to perform the task.

Policy Manager

The policy manager is a fundamental building block to ensure controlled access to objects that are part of a smart space. Such environments are in fact characterized by being multi-user: it is therefore necessary to be able to choose which user can control which object. The user who for the first time registers the object in the system becomes the owner. He can specify fine-grained policies deciding which other users can use the object (that is, if the object can be used by the system to perform the tasks for that user), specifying the decision for any transaction which is the subject able to accomplish. How a policy manager works will be widely described in Chapter 6.

Preference Manager

The preference manager contains user preferences about the object to be selected by the system to perform a certain task. For example, if

the goal expressed by the user is to change the temperature in a certain room, and in this room there are two objects that can accomplish the task (for example, the window and the air conditioner) the preference manager allows the user to establish a priority with which these objects will be selected.

In summary, when the Task Coordinator requires from the Discovery Block the objects that can perform a certain task, a series of filtering is done on objects available in the smart space (Figure 5.3). The first filter is applied by the Location Manager that selects only the 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 preferences for a given action, which choose the object to utilize among multiple objects with similar capabilities.

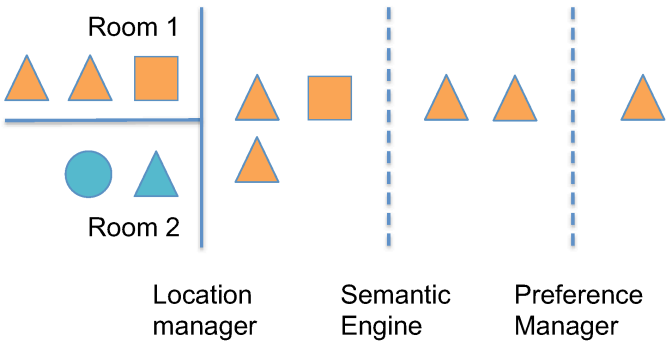

Figure 5.3: *Discovery steps in the proposed M2M architecture*

5.8 Secure Communication Among Blocks

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 cryptography 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 5.4.

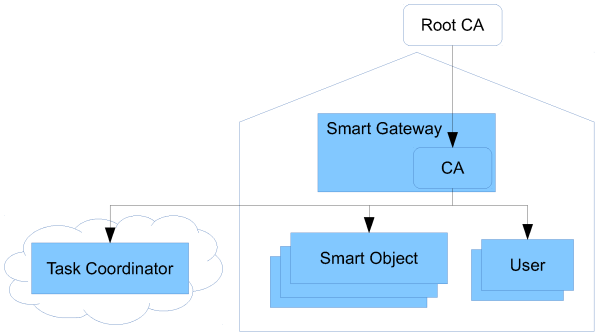

Figure 5.4: *PKI for the proposed system.*

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 [60] [61] 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.

CHAPTER

SIX

SECURITY AND PRIVACY ISSUES IN THE SMART CITY

6.1 Access Control for Context-Aware Services

Mobile devices became in last decade the central hub for our personal information due to the amount of private data they might control locally (e.g. address and phone books, texts and emails) as well as the capability to manage remote information and systems (e.g. again emails, social networks, cloud storage and enterprise systems). In addition, smartphones are one of the main sources of personal information that can feed remote systems like social networks. Examples of this information are the photos and videos recorded on the devices and the multitude of data that might be gathered by sensors like GPS, accelerometer, etc. To manage all these data and obviously for other

purposes, users make an extensive use of applications: pieces of software that may also substitute operating system modules for some core operations and functionalities (e.g. configuration, keyboard). One of the factors that contributed to the success of some mobile operating systems like Android and iOS is the availability of application markets, which provide a very large number of applications. Malicious applications - and the ones that do not take care about users' privacy - may access private data, manipulating and spreading them uncontrollably. These applications may also access system's functionalities impacting on user's bill without his conscious consent (e.g. using SMS, call and Internet services, especially while roaming). Modern mobile operating systems commonly prompt the user with an authorization dialog showing the list of functionalities which an application will have access to. This prompt appears only during the application's installation process and it is not possible to define access control policies to be enforced at run-time. In addition, the user can only decide to install or not a certain application without any degree of flexibility. It is not possible, for instance, to decide to install an application and then constraint the access to some functionalities during run-time, in the case some specified conditions occur.

Modern mobile operating systems allow users to install applications in an easy way. This feature opens big chances for users that are now able to customize devices following their needs and moods but on the other side, it poses strong security and privacy concerns. Smartphones other than handling personal and reserved data, provide also services which have a cost for users. The resources that require protection can be distinguished into:

- User's personal/reserved data: address and phone books, call lists, stored messages, passwords, codes, etc.;
- Applications' reserved data: every application has files related to its configuration or its data, that should be reserved (like passwords). An unwanted access can modify the behavior of the application, its configuration and its stored data;
- System's and device's reserved data: configuration and all the information related to the device (e.g. IMEI, model);
- System functionalities: making calls, sending messages, connecting to Internet, localizing the user are just examples of functionalities that the system provides to applications and that can affect user's privacy or can lead to unwanted expenses. These functionalities have to be provided only to those applications that effectively required them and only following user consensus;

We will cover in detail how Android manages the security of its applications. In short, the choice adopted by Android is that each application states, during its installation process, all the system's permissions that it will require at run-time. The user can give his assent to allow the application to use requested functionalities or stop the installation. In other words, Android bases its security permission granting system on a "prompting" approach at installation time. In addition, the user does not have a centralized control (thus offered by the system) to change the behavior of applications while they are running. This type of control can be only provided independently by single applications. The ability to grant or to reject authorizations to an application at

run-time goes against one of the principles of usability that Android developers have decided to follow:

> Android has no mechanism for granting permissions dynamically (at run-time) because it complicates the user experience to the detriment of security[1].

We agree that subjecting the user to repeated grant requests could seriously damage the device usability. However, we think that a policy-based mechanism to specify a priori the behavior of applications should be introduced. In [62], authors examined the efficacy of privacy signaling provided by Android during installation process. As a result of their research they found that the average user does not pay much attention to warning messages, as he often is not able to understand the consequences that may arise allowing a set of permissions to an application. Conversely, the "download count" information related to an application has a strong effect on the users' decisions. We believe that the Android security model may be enriched with new functionalities. First of all, more and more applications offer a lot of functionalities and then require a large number of permissions during their installation. Users may be interested only in a subset of these functionalities, which might require a limited number of permissions. In the case, for example, of a photo editing application, the user may be only interested on storing photos and not on geo-tagging functions. In this scenario the user is forced to consent to all of the requirements of the application, otherwise the system would prevent the installation. Moreover, latest generation of mobile devices are able to obtain information about user's context. Information from sensors such as GPS, accelerometer,

[1] http://developer.android.com/reference/android/Manifest.permission.html

gyroscope can be used together with information coming from social networks or provided by OS, to determine user's context in a specific instant. We believe that all this user's information can be used by the system to allow the user to decide how his applications should behave according to what his context is. As an example, user might specify the following policies

- Application X does not have the rights to access Internet if the device is roaming
- Service Y must be halted if battery level is lower than 10%

These context-aware policies could be defined by users for each installed application. Another interesting scene could be the one where context-aware policies are not chosen by the user but are imposed to him by other authorities. As an example, we could consider an IT department of an enterprise that may like to impose some policies on employees devices (for instance only during work hours or only to some applications or only when the device is within company site): in this case we need a very flexible policy language and enforcer in order to express more complex and context-aware polices. The needed flexibility becomes even more pushing if we consider the same context, where a Bring-Your-Own-Device [63] policy is in place, and we need to adapt policy based on type of applications, context - private or enterprise - and source of applications. In the consumer segment, other examples can be a policy enforced by a museum to its guests to prevent them using smartphones' camera inside some rooms, or again policies which constraint access to sensors and network communication, etc..

All this personal information that can be obtained from smartphones characterize the context of the user and allow the use of

context-aware services. In a near future where users will be surrounded by a large number of services, many of them may be malicious and manipulate user data for malicious purposes. The smartphone, will then become a hub of services and mobile operative systems will necessarily provide users the possibility to control accesses to these services.

In this Chapter, we therefore propose an extension of the Android security framework that is able to:

- apply security policies at run-time (other than during installation-time)
- enforce security policy and user choices taking into account some contextual information.
- use well-known standard languages for the definition of such policies
- allow the user to choose and change the behavior of such policies

We will also present the main research areas, which can address and improve our proposal.

6.2 Related Work

The growing of context-aware services has accelerated academic interest toward context-aware policies especially for those scenarios which put mobile devices under the spotlight. Some important contributions in this field have been directed through the definition of policy models suitable for context-aware scenarios [64]. Access control systems

should be able to support and understand any new context information in order to address access control requirements. To make this possible, Cheaito et al. presented an extensible access control solution based on XACML making it able to understand new attributes data types including the functions that are used in the policy to evaluate users' requests [65]. Another interesting work is [66] where Li et al. proposed an access control policy model based on context and role that can be appropriate for web services. The model takes context as the center to define and perform access control policies. It uses the contexts of user, environment and resource to execute dynamic roles assignment and constrain the authorization decision. Another interesting work, which addresses conflict problems in context-aware policies, is [67] where authors propose a framework where authorization for a particular access request is decided dynamically based on context information. They further support dynamic conflict resolution where current policy is chosen at run time based on context information. Finally, the emerging of context aware services and relative mobile applications is making it necessary to redesign actual mobile OS's security frameworks. Current smartphone systems marginally allow users to specify the behavior of the applications through the presence of contextual information, Conti et al. propose CRêPE [68], an Android security extension which allows context-related policies to be set (even at runtime) by both the user and authorized third parties, locally or remotely. Policies which can be defined in CRêPE are based on the status of variables sensed by physical (low level) sensors, like time and location. Apex [69] is an extension of the Android permission framework which allows users to specify detailed runtime constraints to restrict the use of sensitive resources by applications. The user

can specify his constraints through a simple interface of the extended Android installer introduced by authors called Poly. Within Android, Google provide a so-called "Device Policy for Android" environment that allows to set policies to enforce use of PIN or password and screen lock on the device and allow an administrator to wipe the device remotely. This framework has the only aim of preventing physical access to information on a device which is not under direct user control. It has no relation with the behavior's control of a certain application. In the field of improvements to the security of standard operating systems, it is well-know the case of Security Enhanced Linux [70] where on top of Linux an amended security framework was introduced to enhance the capabilities of the operating system. SELinux became part of the standard Linux distribution once adopted by the community. This article goes in the same direction, introducing a more featured policy system on an already available platform.

6.3 Access Control in Mobile Operating Systems

Mobile operating systems protect their data and functionalities using several approaches. Digital signature and certification, for example, guarantee authenticity and integrity for the applications which are being installed. Using a central repository, like an Application Market, besides providing to users the possibility to find every kind of applications, allows them to know information about the application's author and to read other users' comments about the application, which might reveal possible bugs and unexpected or malicious behaviors. Another

fundamental method to guarantee applications' security is the isolation: every application is executed in a restricted execution context where it can access only a minimum set of functionalities: it cannot access reserved data and reserved functionalities, it cannot communicate with other applications or access their data. The way in which the main OSs such as iOS, Android and Windows Phone manage the security of their applications is undoubtedly one of the factors that have contributed to their success.

On the iOS platform, each application is signed with an Apple-issued certificate. Applications can be installed only from iTunes market. To publish an application, an author must be registered and its application is first deeply tested. Every third-party application is sandboxed and can access exclusively to its directory. Accessing to protected data or functionalities and communicating with other applications is possible only using system APIs. An application declares protected functionalities that it needs during market registration phase, so security compliance is established by Apple in registration phase. A user can be advised of which functionalities an application needs during the installation phase. Apple's *App Store* acts as a gatekeeper for the applications uploaded by developers. The Apple's staff checks the source code of uploaded applications and retains the possibility to refuse them if not compliant with the security criteria.

As reported in [2] "The Windows Phone 7 and 8 security model introduced the chamber concept, which is based on the principle of least privilege and uses isolation to achieve it; each chamber provides a security boundary and, through configuration, an isolation boundary

[2] "Windows Phone 8 Security Overview" - 2012

within which a process can run. Each chamber is defined and implemented using a policy system. The security policy of a specific chamber defines what operating system capabilities the processes in that chamber can call. Every app on Windows Phone (including Microsoft apps and non-Microsoft apps) runs in its own isolated chamber".

Android is based on Linux Kernel and uses its mechanisms to ensure security. Each application run in a different process as a different user with an UID (User ID), with a different home directory and no root privileges. So application's isolation is done at kernel level and for each application, included system and pre installed applications. Communication between apps or between apps and system is done with IPC mechanisms that extends Linux kernel, like Binder, or are implemented at higher level, like Services, Intents, and Content Providers. Android has a centralized repository, *Google Play Store*, where application can be published, but can also install applications from other sources. Each application is certified and the certificate is verified during installation. Unlike the App Store, Google Play exercises no control over the source code of applications uploaded by users.

6.4 Android Security Framework

Protected data and functionalities are provided by Android to applications using the Android Permission System. Each kind of protected data or functionality is associated to one permission. Android provides a standard set of permission that includes all functionalities provided by the system. An application can define its own permission to provide

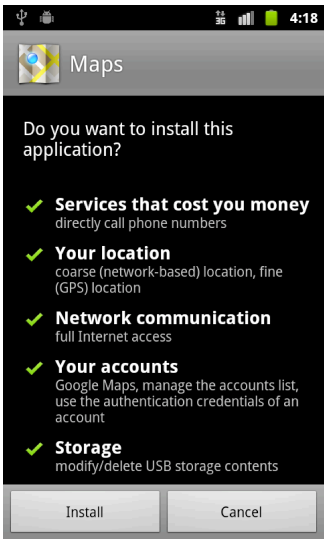

Figure 6.1: *Android installation process*

and protect its functionalities from other applications. An application lists permissions it needs using a Manifest file, which is read by the system during installation. Android has four protection levels which differ for their relevance: *Normal*, *Dangerous*, *Signature*, *SignatureOrSystem*. Each protection level has to be specified by a developer when he defines own permissions.

During installation, a dialog box listing all the permissions at protection level *Dangerous* requested by the application that the user is trying to install, is shown. User then reads and decides whether to install the application or not. If the application is installed, all permissions it declared in its manifest are registered and associated to the UID that identifies the application itself. Every time an application tries to access to protected functionalities or data, the system

controls if the permission is registered for the application's UID. If the answer is positive, the permission is granted, else the application cannot get access. Permission are granted once at install time at cannot be revoked afterwards: the only chance is to uninstall the application. Is not possible to revoke a permission during application's execution. It is possible making a choice only during installation. Moreover all decisions taken are static. There is no connection between policy's decision and user's context: battery level, position, connection, and any other variable are not taken into account during evaluation. Google justifies this static mechanism saying that it would be too boring for users answering to security prompts every time an application carries out an operation, the user would end to ignore advice, granting permissions without reading. So Google prefers to concentrate all user attention at install time, where he has much focus on application, its functionalities and problems. In this way, Google takes no account of the correlation between security decisions and context.

Android's security framework checks for permissions when one of the following situations occurs. i) An application want to access to a particular functionality protected by a permission (e.g. GPS information), ii) An application tries to start an activity of another application, iii) Both when an application sends and receives broadcasts, iv) An application tries to access and operate on a content provider and v) When binding to or starting a service.

With our work we try to take into account all of these cases. We have studied the Android source code in order to establish which are the most suitable points to place our run-time policy's check.

6.5 Policy Model

We've customized the XACML[3] policy model to suit the Android system. The policies we consider, contain information about subject, resources and context. More precisely, in our model: i) A *Subject* represents the application to which the policy will be applied. ii) A *Resource* represents the Android's permission protected by the policy. iii) *Context* is a set of information which characterize user's or device's context.

Each of the above elements, according to the XACML standard, is identified by a series of attributes.

A Subject can be specified through several attributes which are taken from application certificate. The most important are:

- *ID*: The name of the application's package,
- *AUTHOR-KEY-CN*: Author's common name,
- *AUTHOR-KEY-FINGERPRINT*: Author's signature.

Resources are expressed with "api-feature" attributes which are mapped to standard Android permissions, and with "uri" in case of accessing a content provider.

Our context model is made up of the following information:

- Location data: indicate user position eg. latitude and longitude with a radius, or higher level information like city and nation;
- Device battery data: indicate battery level and if it is charging or not;

[3]https://www.oasis-open.org/committees/xacml

- Time data: eg. time interval or day of week information;

- Connection type used by device at that moment: this information allow to distinguish between connections that costs money to the users and those free of charge;

Our policy model is based on a subset of the XACML grammar for policy definition. Mainly, a policy is composed of a *target* that identifies the entity to which the policy will be applied (in our case the subjects are applications) and one or more *rules*. Each rule contains an *effect*, which can assume values "permit", "deny" or "prompt", and a set of matches for the resources and the environment. An example of security policy which could be defined inside SecureDroid is the following:

```
<policy-set combine="deny-overrides">
  <policy combine="deny-overrides">
    <target>
      <subject>
        <subject-match attr="id"
            match="com.example.exampleApp"/>
      </subject>
    </target>
    <rule effect="deny">
      <condition>
        <resource-match attr="api-feature"
            match="android.permission.INTERNET"/>
        <resource-match attr="uri"
            match="http://blockedsite.org*"/>
        <context-match attr="connection-type"
```

```
            match="mobile-roaming"/>
      </condition>
    </rule>
    <rule effect="permit">
  </policy>
</policy-set>
```

The above policy sets the behavior for an application (in this case "exampleApp") denying Internet access toward "http://blockedsite.org*" if the current connection is of type "mobile-roaming". The policy contains "deny-overrides" as rule's combination algorithm and it is composed of two rules which are evaluated in the same order they are written. The first rule contains itself three matches which are evaluated with *and* logic (as default) since the *condition* element does not have a specified value for its *combine* attribute. The first match indicates the resource required by the application (android.permission.INTERNET), the second one specify a filter for the URI ("http://blockedsite.org*") and the last one specifies the device context ("mobile-roaming"). According to deny-overrides algorithm, the PDP will answer "deny" if "exampleApp" requires Internet access while it is roaming, otherwise it will answer "permit".

6.6 SecureDroid Layer

Controlling the way in which an application works during run-time means changing the normal operation carried out by Android's security framework. A naïve solution to mediate access from applications to Android services would be to extend the Activity redefining its *get-*

SystemService method. All applications based on this "safe" version of Activity will then be subject to a security check during run-time whenever they try to access a service. Although it may seem a good solution, using this method the control during run-time would take place only for those applications which make use of this extended version of Activity. All the other applications, such as those in the Android market, would execute without any security control. For these reasons, our choice is to extend the framework based on Android by changing the components which are responsible for granting permissions to applications.

The security engine, presented in this paper, has been developed taking into account one main principle: making access management policies dynamically dependent on the context. The decision process is carried out at each attempt to access a protected resource and its result is derived from the evaluation of a set of policies. The flexibility and modularity of the evaluation method is strongly dependent on these policies, their structure and content. The rules which compose our policies contain a set of attributes dependent on applications and available resources but also some context-dependent. In facts, the huge availability of context information provided by modern mobile devices makes it possible to effectively assess the context in which a user and his device are immersed, allowing to define policies that take into account the variables that characterize surrounding environment and user activity.

By setting up a context-aware security system, it is possible to automate the decision process to reduce the amount of needed user/device interactions, making the whole process transparent to the user (that in this case would be simply notified about decisions taken by the

security engine). The definition of context-dependent policies therefore allows to exceed limitations, like those supposed for example by Android's designers, to the usability of a system that continuously prompts a user for resources' access authorizations. It also allows moving away from the equally limited mechanism of security decisions taken only during an application's installation process. To maintain a high dynamics of the security module proposed, it is also important providing the capability to update during run-time, in a fast and easy way, the access control policies. In addition, we must pay particular attention to operations performed on policy files like editing and management in general. In order to prevent unauthorized, accidental or malicious access to the policies, they are protected, as well as any other resource in the operating system, by the Android's permissions mechanism. In this way, maintaining coherence with system approaches (the basic principle taken into account during the design and implementation of the whole module) it is possible to get at the same time the protection level offered by the operating system to other private resources and the ease of access for authorized entities. It is important to point out that the security engine is an extension module for the Android security framework and it preserves the underlying features and capabilities. Our security engine might grant the requested authorizations to an application after the standard security control made by the operating system is performed, in order to avoid a privilege escalation. An application cannot access to a secured feature without having first declared in its manifest file.

6.6.1 Policy Evaluation Order

Despite the fact a smartphone belongs to a user, there are several parties that can control its operations, both in a static and dynamic way. Manufacturers, in either the case that they are owners of the operating system (e.g. Apple) or simply customize it to suit their own devices (e.g. Samsung), could set policies which restrict access to some functionalities of their devices. A first version of iOS, for example, limited the use of bluetooth to connect only earphones avoiding to exchange of files. Google retains the ability of modifying some aspects of Android devices over the air. Android platform allows not only remote application's uninstallation (via the *REMOVE_ASSET* intent), but also the installation of new applications (via the *INSTALL_ASSET* intent). In addition, Google can push a *REMOVE_ASSET* message down to all the Android phones in order to remote kill a particular application deemed malicious. Also the operator may specify policies on a terminal, which restrict access to certain features or do not allow the use of a device with a SIM card of another operator. The same applies to a third-party, which may require the installation of its own policy on a device in order to limit some functionalities (as in the case of the museum in Section 1). For these reasons, we have provided the opportunity of evaluating multiple policies during the enforcement phase. The policies' evaluation order is quite important because it reflects the priority given by the system to each policy. SecureDroid considers the following kinds of policies:

Manufacturer $\longrightarrow$ *Operator* $\longrightarrow$ *Third-parties* $\longrightarrow$ *User*

The First to be evaluated is the Manufacturer's policy because it is the most relevant to the system. The last policy to be evaluated is the

one that the user can specify for his applications. It should be pointed out that the permissions to add, remove or edit a certain policy have to be granted by an upper level policy. For example if the museum requires adding a policy to the user device, this operation must be allowed at least by the Manufacturer's or the Operator's policies. SecureDroid may be adopted also for Bring Your Own Device scenarios, where companies permit employees to bring personally owned mobile devices to their workplace, and use those devices to access privileged company information and applications. In fact, companies may install on these devices a third-parties policy to avoid unnecessary costs or an incorrect use of devices that can be made by employees. All the policies handled by SecureDroid are stored in a system folder, which can not be accessed by normal applications. The only way to modify or add a new policy is through a system service we have developed which is described in Section 6.7.

The mechanism introduced by SecureDroid starts working only after the standard security check provided by Android. If Android notices that an application is attempting to access a resource for which permission has not been declared in the Manifest, it denies access directly without going through SecureDroid. The control carried out by SecureDroid is more complex than that performed by relying exclusively on the Android permissions defined in the Manifest. In the worse case, for each request SecureDroid has to check N policies, where N is the number of parties which may have control of the device: from the manufacturer to the user. Each of these policies may be more or less complicated on the basis of the number of rules it contains. For all these reasons, placing SecureDroid after the standard Android control avoids unnecessary overheads due to all the applications that would

be eventually stopped by the system, for example for the absence of a permission in the Manifest file.

6.6.2 SecureDroid Architecture

SecureDroid acts at framework level in the Android system, its basic architecture is depicted in Figure 6.2.

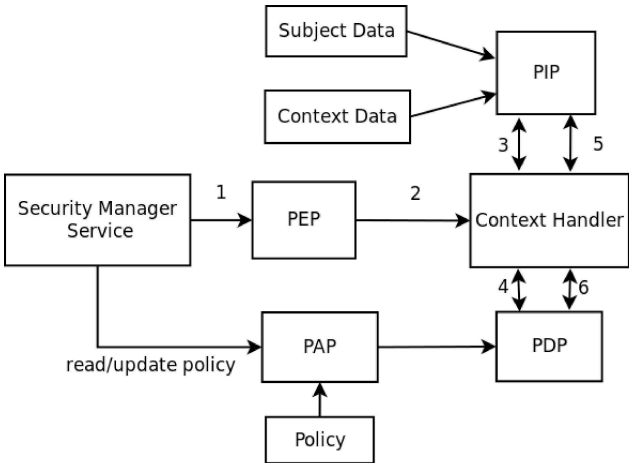

Figure 6.2: *SecureDroid Architecture*

- *Security Manager Service* is the system entry point for our security engine. It's a system service that offers three functionalities: read policies, write policies and make requests. Security Manager maps XACML responses into pre-defined system's responses.

- *Policy Enforcement Point* (PEP) is the enforcement point of policy access control. PEP is responsible for collecting both

information about UID of the application which requested access and the permission that is being requested. Finally PEP creates and sends a request to Context Handler.

- *Context Handler* (CH) links all the components, handles both requests from PEP and information from PIP, finally forwards requests to PDP;

- *Policy Information Point* (PIP) collects information about subject, resource and context attributes using system Package Manager and device's sensors;

- *Policy Decision Point* (PDP) is the point that reads and evaluates policies. It has principally two functions: given a subject, it reads policy and returns attributes that are necessary for the request's evaluation and, given a complete request, it evaluates that and and the policy returning a decision;

- *Policy Administration Point* (PAP) is responsible for managing the policies. It securely store policies and give them to PDP. It is the only one that can have access to policies, so it is invoked by Security Manager to modify, to read and to store policies.

Security Manager deals with creating and maintaining a reference for all the parts making up the engine. Particular attention is required during the creation of the PAP, which controls if there is a previous policy stored in the filesystem and, if so, loads it. In case of absence or corruption of a policy file, PAP uploads a default policy and passes it to the PDP that will be able to use this policy to perform its functions. When an application requests a resource, such as a system capability

or some information from a content provider, its request is checked at run-time by both ActivityManager and PackageManager which are defined in the Android security framework. After the PackageManager has run the standard checks on permissions declared in the application's Manifest, it starts running SecureDroid calling the *doRequest* method provided by the Security Manager. The request which arrives to Security Manager Service contains three information: 1) the UID of the application which is requesting access, 2) the permission the application is asking for and, 3) an optional URI indicating the resource that is being requested (e.g. a contact, a picture).

Since Android checks only if some permissions have been previously granted to an application, it doesn't provide a way to propagate, trough ActivityManager and PackageManager, information about requested resources (URIs). We have extended this mechanism carrying URIs information up to the Security Manager. In this way, SecureDroid can provide a fine-grained control not only based on the permissions declared in the Manifest but also based on the resource itself.

Assuming that user has defined a policy (for example the one in Section 6.5) for a certain application, the execution flow of SecureDroid is shown in Figure 6.2. When the application requires access to a resource (for example, when it attempts to open a connection to a URL) the PackageManager checks if the permission *android.permission.INTERNET* has been granted to the application by the user, during installation. If not, PackageManager immediately denies the access, otherwise it invokes the method *doRequest* provided by the Security Manager which calls the PEP. The PEP collects information about the request (the UID of the application, the resource to which the application wants to access and, where the requested URI)

and sends them to the Context Handler. This sends the UID to the PIP which returns additional information about the subject (i.e. the application) such as package name, author's signature, etc. At this stage, CH sends these subject information to the PDP which, knowing the policies, returns to CH a list of context information needed by the PDP to make a decision (e.g. "Is the device roaming?"). CH then requires to the PIP the current values for these context information. The PIP returns the current context status (e.g. "Current connection type is roaming"). At this point, CH owns all the information it needs (subject, resource and context). It sends this information to the PDP which evaluates the policy and takes a decision which is propagated up to the PEP, which eventually enforces it.

6.6.3 Decision handling

The PDP takes a decision only once a request from an application arrives, this decision is then propagated to the Security Manager that is responsible for enforcing it. According to policy specification, a decision could be one of: PERMIT, DENY, PROMPT or UNDEFINED.

In the case of PERMIT, Security Manager communicates the positive outcome to the Package Manager, which continues its normal execution, including assessment of the request in the standard mode provided by the system and returns its standard answer.

In the case of DENY, the Package Manager returns a negative response to the application which asked for a permission. SecureDroid returns a value different from that returned by the standard control system when a request from an application is not granted. In this way, the Security Manager Service can differentiate whether a request

has been denied by the Android standard security framework or by SecureDroid.

In the case of PROMPT, a dialog is displayed asking the user to grant or not a permission to the application. According to policy definition, SecureDroid handles three different kinds of prompt: i) *PROMPT-ONESHOT*: the dialog appears every time the application tries to access a function protected by the policy, ii) *PROMPT-SESSION*: the choice made by the user is saved until the system reboot or the device is turned off. iii) *PROMPT-BLANKET*: the choice made by the user is stored in the filesystem and remains in effect until the user decides to remove it. Figure 6.3 shows some screenshots of the dialogs which the system launches when the decision taken by PDP is PROMPT.

SecureDroid defines a new exception called *PolicyDenyException* which is an extension of standard *SecurityException* provided by the system. Applications which are aware of the presence of SecureDroid are able to manage instances of denied permission. These applications can, in fact, catch PolicyDenyException and understand when SecureDroid denies a permission. In this way applications can modify their behavior according to which permission has been denied. Other applications, which do not handle PolicyDenyException, in the case of having access to resources denied by SecureDroid, are treated by the system as those applications that try to access a system's capability without having the correct permission in the Manifest. As we have seen, Android grants permissions during the installation phase. So once permission has been given in the Manifest file, an application will always have access to the permissions declared. So it is unusual for applications to see a permission denied during its execution. There

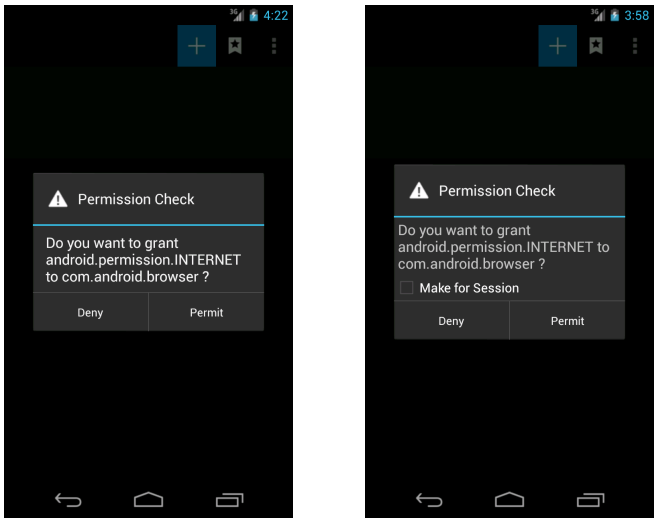

Figure 6.3: *SecureDroid dialogs in the cases of PROMPT-ONESHOT and PROMPT-SESSION*

is no common way to manage these situations and there are not guidelines from Google. It is not possible to predict *a priori* the behavior of an application when a permission is denied: probably if the application does not handle this occurrence it will crash.

6.6.4 Comparison with other security frameworks

Studying other modern solutions to this problem, we found two approaches which attempt to solve the same problems that were mentioned. In particular, we have seen that the works that most resemble SecureDroid are APEX and CRêPE.

APEX is an extension of the Android security framework which allows the user to have more control during application installation.

At this stage, in fact, the user can specify security options for each permission required by the application being installed. APEX defines a new syntax for the policy, we have preferred to rely on an already well-established standard such as XACML, also used by another ongoing project called *webinos* [71]. APEX also allows users to specify the policy with static attributes (for example, to restrict the use of a resource to a maximum number of times per day, or to deny access to the resource after a certain time of day). From our point of view it is important allowing the user to specify policies that depend on his context.

CRêPE has more than one point in common with SecureDroid. Like our solution, CRêPE offers the ability of specifying parameters related to user's context while defining policy. Also CRêPE acts at the system level but, unlike SecureDroid, performs its checks before the standard permission control offered by Android. We preferred to place our control after that standard to avoid repeated checks on requests made by the system itself. These kinds of requests should be always accepted by SecureDroid in order to not destabilize the whole system. In addition, it might be pointless to grant or not a particular permission at SecureDroid level (thus evaluating the entire chain of policy that we discussed) if we eventually find out that the application did not declare such a permission inside its Manifest file. Another difference between our framework and CRêPE is that the latter takes into account both access control policies and obligation policies. An obligation policy defines an action that the system will perform under certain conditions. For example it could be possible to automatically disable certain features of the system (such as the camera) when the battery charge drops below a certain threshold. The obligation pol-

icy, in the case of CRêPE, can be activated when certain conditions, which depend on the user's context, are met. To support obligation policies CRêPE must perform an onerous procedure. Initially CRêPE gets information about those parameters that characterize context and thresholds that would trigger the action. Subsequently CRêPE starts monitoring these parameters to detect, moment to moment, if one of the conditions related to the obligation policy is met and, if that happens, it triggers the action. This continuous monitoring activity could be computationally heavy and can effect the energy consumption of the device. Keeping always on the accelerometer, GPS or WiFi etc. can drastically reduce the battery life. Unlike CRêPE, SecureDroid does not perform a repetitive control of context variables. This check is performed only at the instant when an application requests access to a resource. In this way we simplify the architecture of the system and reduce energy consumption. In addition to these considerations on energy saving, one of the reasons why we decided to do not take into account obligation policies is because we believe that they should be handled at application level and not at system level.

6.7 Policy Management

Security Manager Service is a system service, accessible to applications through a client interface. It exposes two methods: *readPolicy* and *writePolicy*. Each of these methods is protected by an Android permission so it cannot be used by applications which do not have proper rights. Security Manager Service defines also *READ_POLICY* and *WRITE_POLICY* permissions which must be declared by an applica-

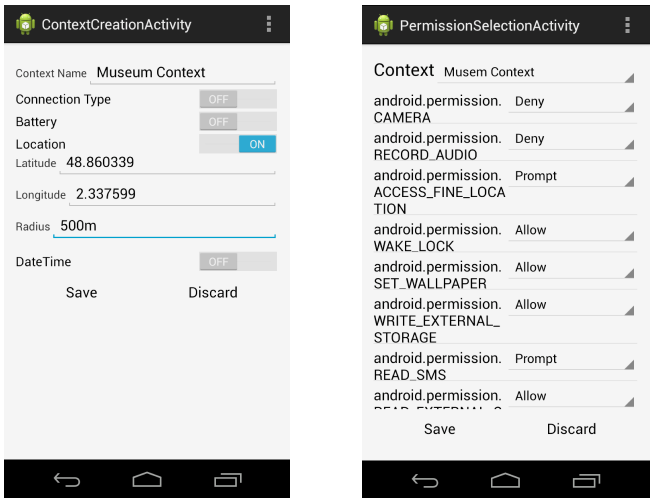

Figure 6.4: *Context and Policy Management*

tion if it intends to read or edit system policies. These permissions are defined with a *signatureOrSystem* protection level, so only system application can get access to them. We wanted to give SecureDroid the possibility to set policies both from a user input and from the external. A user interface is provided through an application by which users can create context specifications and choose which permission to grant and which others to deny for each context (Figure 6.4). Users can define several contexts specifying some parameters such as time constraints, battery levels, location, connection, etc. Subsequently, user can specify for each application which permissions have to be allowed when a particular context is active. SecureDroid provides also the possibility to set policies from the external using a system service which listens to various interfaces (NFC, SMS, Internet, Bluetooth) for policy updating requests. Each request contains information about

the policy provider's identity and the policy itself. A policy provider may be a manufacturer, an operator or whatever third-party and it is identified by a signature. The identity of the policy provider indicates the priority that will have the policy in the chain of evaluation defined in section 6.6.1. This means that if the policy provider is a third-party, his policy will be integrated into the system only in the case both manufacturer and operator policies allow this operation. A policy update may regard a single application as well as all applications: it is possible, for example, to block the permission *android.permission.CAMERA* for the application *com.android.camera* or to block *android.permission.CAMERA* for all applications.

CHAPTER

SEVEN

CONCLUSIONS

The smart cities will replace the cities we know today in a future that is becoming increasingly close. The smart city will be a container of services offered to their citizens, in order to simplify their lives and make more effective and efficient processes that still make up the weak point of many cities. The objectives of the new smart city will be to: i) improve public transportation in order to reduce the traffic, ii) reduce energy consumption, iii) let citizens participate in the administration of the city. Current technologies and those that will spread in the coming years in the fields of electronics, IT and telecommunications, will enable and accelerate the transformation of cities in smart cities.

In this scenario, the user, and therefore the citizen, will be the focus of services ecosystem which will be a key part of the economy of these new cities. These services, which will be gradually more and more, will be delivered to users in an intelligent way in order to fit their needs. Services of the smart city will be proposed to users with

the aim to adapt to what is the physical/logical context in which the user is located. Therefore services of the smart city will be proposed to citizens based on their context, which can be characterized by: position, movement, social interactions, etc.

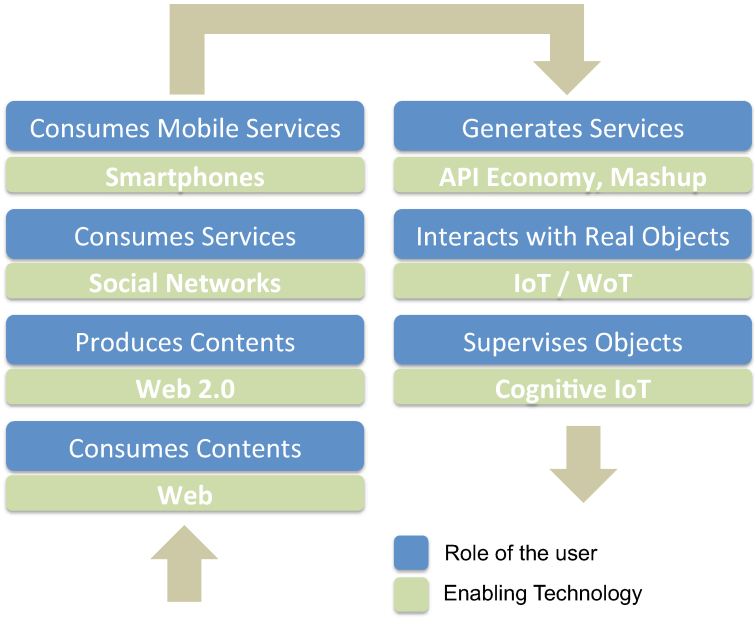

Figure 7.1: *The evolution of Web's users*

Figure 7.1 shows the evolution of the user over the past years. In particular, at the dawn of the Web user was a simple passive consumer of contents available on Web pages. Along with the introduction of Web 2.0 and the emergence of first social networks and blogs, the user turned in the so-called prosumer, that can both use and generate contents. With years to come and the spread of the API economy, the number of services that users have been able to use grew exponentially. These services were at the first stage used statically by desktop com-

puters and later in mobility using smartphones. Moreover, the spread of high level tools for the services mashup have enabled users not only to generate new contents but also new services that may be used by other users. Tools for high and low level mashup are increasingly narrowing the skills required to create new services and make them available on the Web. Within a few years more and more users will be able to generate services and this will constitute a substantial boost to the API which will be the mainstay of the smart city economy. Another important phenomenon that has been treated in this thesis is the Web of Things, which was certainly helped by advances in electronics and the availability of virtualization platforms for creating virtual copies of real objects and then considering such items as service providers. In the Web of Things, users interact with real objects, but they do so in a completely transparent with respect to their normal interactions with the content of the Web. In the cities of the future users, as well as their friends, will follow on the social networks also items such as the refrigerator (which notifies when the food it contains is expiring), their plants (which will communicate their health status), and so on. There will not exist, therefore, a clear separation between user and object, and between real and virtual world. Finally it was shown an evolution of the concept of IoT that has led to the Cognitive Internet of Things (CIoT): a future scenario in which the objects (supported by appropriate platforms) come with an intelligence and are able to self-organize themselves with the aim of satisfying the goals expressed by users. The CIoT requires users to be surrounded by smart objects that can understand who among them is able to perform a certain task. This is a further development of the role of the users who no longer have to interact directly with objects but will be limited to ex-

press the objectives to be achieved, then the objects around him will coordinate themselves in order to accomplish the tasks.

The role of the user is therefore evolving and goes hand in hand with technological developments. Cities of the future should benefit from this new ability of users but at the same time cities must support their citizens in their daily lives through services tailored to them. As the title says, in this thesis the author has addressed some important issues to provide users with secure access to context-aware services in a smart city, providing contributions in the form of solutions or suggestions to these problems. The content of research in this thesis have benefited from a research, design and implementation that the author has actively played [6] within the European FP7 *Webinos* project. The platform Webinos was therefore taken as a reference for the study of the issues addressed during the period in which the author has done his research. In particular, in Chapter 3 the author has proposed to adopt the webinos platform to allow users to generate and provide services to other users, using their mobile devices. Using webinos, smartphones become tiny servers and can host services enabling scenarios like crowdsensing and mobile distributed computing. Chapter 4 has addressed the problem of interaction between user and heterogeneous objects, namely, that have different modes of interaction and various control procedures. 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 allow him to control the object. For this purpose the author proposes the extension of the webinos platform by introducing of a new API for smart objects: through this API applications may require a description of the capabilities offered by the object, and

use this description to dynamically generate the graphical interface. Chapter 5 has described a scenario in which objects of a smart space can work together to accomplish the tasks determined by the user. In this context, the author proposes a cloud architecture and methodologies to enable the implementation of such scenarios. The concept of Cognitive Internet of Things discussed in this chapter is a new and challenging topic which embraces many disciplines such as semantic, artificial intelligence, machine learning. Finally, Chapter 6 was focused on the security and privacy issues that may arise in scenarios where users deal with context-aware services because the context is characterized by personal and private information that users do not want to disclose. Since users access these services from their mobile devices, the author proposes an extension of the Android mobile operative system in order to allow users to specify security access control policy to discriminate which personal information can be disclosed and which services are allowed when the user is in a certain context.

Writing this thesis, the author has not got the claim to propose definitive solutions to topics that today are under study and standardization. The aim of this thesis is to show how over the years, due to technological developments on many fronts, skills, and the ability of users have evolved and brought them to be the next citizens of smart cities. The author hopes that the reading of this thesis may give rise to constructive criticism and valuable insights to undertake new and promising research activities.

BIBLIOGRAPHY

[1] R. Mason, "Welcome to the api economy," 2014.

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

[3] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *Services Computing, IEEE Transactions on*, vol. 3, pp. 223–235, July 2010.

[4] G. L. Torre, "Contributed code repositories: https://github.com/glatorre," 2013.

[5] webinos Project Team, "webinos repositories: https://github.com/webinos," 2013.

[6] webinos Project Team, "webinos apps repositories: https://github.com/webinos-apps," 2013.

[7] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[8] Appuntisoftware.it, "Semantic web services," 2010.

[9] P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. Ter Hofstede, "Analysis of web services composition languages: The case of bpel4ws," in *Conceptual Modeling-ER 2003*, pp. 200–215, Springer, 2003.

[10] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, *et al.*, "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, pp. 2007–04, 2004.

[11] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, E. Oren, *et al.*, "Web service modeling ontology (wsmo)," *Interface*, vol. 5, p. 1, 2005.

[12] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, *et al.*, "Semantic web services framework (swsf) overview," *World Wide Web Consortium, Member Submission SUBM-SWSF-20050909*, 2005.

[13] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, A. Sheth, and K. Verma, "Web service semantics-wsdl-s," 2005.

[14] C. Doukas and F. Antonelli, "Compose: Building smart & context-aware mobile applications utilizing iot technologies," in *Global Information Infrastructure Symposium, 2013*, pp. 1–6, IEEE, 2013.

[15] J. L. Pérez, Á. Villalba, D. Carrera, I. Larizgoitia, and V. Trifa, "The compose api for the internet of things," in *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pp. 971–976, International World Wide Web Conferences Steering Committee, 2014.

[16] C. Fuhrhop, J. Lyle, and S. Faily, "The webinos project," in *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, (New York, NY, USA), pp. 259–262, ACM, 2012.

[17] webinos, "webinos travel," Feb. 2013.

[18] Statistic Brain, "Social networking statistics," Feb. 2013.

[19] TG Daily, "Internet-enabled devices to outpace pc shipments by 2013," Feb. 2013.

[20] Yahoo, "Pipes: Rewire the web," Feb. 2013.

[21] Z. Zhao, N. Laga, and N. Crespi, "A survey of user generated service," in *Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, pp. 241 –246, Nov. 2009.

[22] C. S. Jensen, C. R. Vicente, and R. Wind, "User-generated content: The case for mobile services," *Computer*, vol. 41, pp. 116–118, Dec. 2008.

[23] J. Tacken, S. Flake, F. Golatowski, S. Prüter, C. Rust, A. Chapko, and A. Emrich, "Towards a platform for user-generated mobile services," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 532–538, april 2010.

[24] D. Werth, A. Emrich, and A. Chapko, "An architecture proposal for user-generated mobile services," in *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pp. 142 –147, Jun. 2012.

[25] A. Emrich, A. Chapko, and D. Werth, "Context-aware recommendations on mobile services: The m:ciudad approach," in *Smart Sensing and Context* (P. Barnaghi, K. Moessner, M. Presser, and S. Meissner, eds.), vol. 5741 of *Lecture Notes in Computer Science*, pp. 107–120, Springer Berlin Heidelberg, 2009.

[26] OASIS, "Oasis extensible access control markup language (xacml) tc," Feb. 2013.

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

[28] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[29] L. Manovich, "The poetics of urban media surfaces," *First Monday*, vol. 0, no. 0, 2006.

[30] C. Thompson, "Smart devices and soft controllers," *Internet Computing, IEEE*, vol. 9, no. 1, pp. 82–85, 2005.

[31] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the internet of things," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 44–51, 2010.

[32] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009.

[33] E. Mingozzi, G. Tanganelli, C. Vallati, and V. Di Gregorio, "An open framework for accessing things as a service," in *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pp. 1–5, 2013.

[34] P. E. Estrada-Martinez and J. A. Garcia-Macias, "Semantic interactions in the internet of things," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 13, pp. 167–175, July 2013.

[35] V. Catania, G. La Torre, S. Monteleone, D. Patti, S. Vercelli, and F. Ricciato, "A novel approach to web of things: M2m and enhanced javascript technologies," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pp. 726–730, 2012.

[36] H. Son, S. Han, and D. Lee, "Contextual information provision on augmented reality with iot-based semantic communication," in *Ubiquitous Virtual Reality (ISUVR), 2012 International Symposium on*, pp. 46–49, 2012.

[37] 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, pp. 34–47, 2001.

[38] S. Garrido-Jurado, R. MuÒoz-Salinas, F. Madrid-Cuevas, and M. MarÌn-JimÈnez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, no. 0, pp. –, 2014.

[39] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. Biswas, and K. Moessner, "Enabling smart cities through a cognitive management framework for the internet of things," *Communications Magazine, IEEE*, vol. 51, pp. 102–111, June 2013.

[40] G. Niezen, "Ontologies for interaction: Enabling serendipitous interoperability in smart environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 5, no. 1, pp. 135–137, 2013.

[41] Q. Wu, G. Ding, Y. Xu, S. Feng, Z. Du, J. Wang, and K. Long, "Cognitive internet of things: A new paradigm beyond connection," *Internet of Things Journal, IEEE*, vol. 1, pp. 129–143, April 2014.

[42] 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.

[43] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background spectrum," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 155–168, ACM, 2011.

[44] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Demo: indoor localization without infrastructure using the acoustic background spectrum," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 385–386, ACM, 2011.

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

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

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

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

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

[50] 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*, pp. 70–70, IEEE, 2007.

[51] 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*, pp. 402–409, IEEE, 2009.

[52] 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.

[53] 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, pp. 1208–1213, 2009.

[54] 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.

[55] 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.

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

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

[58] 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.

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

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

[61] 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.

[62] K. Benton, L. J. Camp, and V. Garg, "Studying the effectiveness of android application permissions requests," in *Fifth International Workshop on SECurity and SOCial Networking*, 2013.

[63] G. Thomson, "Byod: enabling the chaos," *Network Security*, vol. 2012, no. 2, pp. 5–8, 2012.

[64] H. Yahyaoui and M. Almulla, "Context-based specification of web service policies using wspl," in *Digital Information Management*

(ICDIM), 2010 Fifth International Conference on, pp. 496 –501, july 2010.

[65] M. Cheaito, R. Laborde, F. Barrere, and A. Benzekri, "An extensible xacml authorization decision engine for context aware applications," in *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pp. 377 –382, dec. 2009.

[66] H. Li, Y. Yang, Z. He, and G. Hu, "Context-aware access control policy research for web service," in *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, pp. 529–532, oct. 2011.

[67] A. Mohan and D. M. Blough, "An attribute-based authorization policy framework with dynamic conflict resolution," in *Proceedings of the 9th Symposium on Identity and Trust on the Internet*, IDTRUST '10, (New York, NY, USA), pp. 37–50, ACM, 2010.

[68] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "Crêpe: A system for enforcing fine-grained context-related policies on android," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 5, pp. 1426–1438, 2012.

[69] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, (New York, NY, USA), pp. 328–332, ACM, 2010.

[70] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the linux operating system," in *Proceedings*

of the FREENIX Track: 2001 USENIX Annual Technical Conference, (Berkeley, CA, USA), pp. 29–42, USENIX Association, 2001.

[71] J. Lyle, S. Monteleone, S. Faily, D. Patti, and F. Ricciato, "Cross-platform access control for mobile web applications," in *Policies for Distributed Systems and Networks (POLICY), 2012 IEEE International Symposium on*, pp. 37–44, 2012.