ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

A Collaborative Mobile Crowdsensing system for Smart Cities

Relatore:
Chiar.mo Prof.
Luciano Bononi

Co-relatori:
Dott. Luca Bedogni
Dott. Federico Montori

Presentato da:
Alain Di Chiappari

Sessione II
Anno Accademico 2015/2016

My crime is that of curiosity. My crime is that of judging people by what they say and think, not what they look like.

[The Hacker's Manifesto, The Mentor]

Contents

List of Figures

Chapter 1

Introduction

Nowadays words like Smart City, Internet of Things, Environmental Awareness surround us with the growing interest of Computer Science and Engineering communities. Services supporting these paradigms are definitely based on large amounts of sensed data, which, once obtained and gathered, need to be analyzed in order to build maps, infer patterns, extract useful information. Everything is done in order to achieve a better quality of life.
Just to illustrate the problem, a large number of people around the world are exposed to high levels of noise pollution, causing serious physical illnesses and influencing negatively productivity and social behavior. Countries, such as the United Kingdom and Germany, have started monitoring noise pollution [1] to face the problem starting from data.
The advantages that could come from an intelligent approach to sensing, gathering and analyzing environmental or social data are numerous and heterogeneous. They ranging from intelligent transportation system, using traffic congestion monitoring and real-time parking maps; environmental monitoring, with air and noise pollution level sensing; targeted and location-based advertising, using GPS or other location systems; furthermore, other interesting solutions are becoming possible, such as the construction of detailed and real-time maps upon WiFi and telephone cell coverage.

A successful society and city management relies also on efficient monitoring of urban and community dynamics for decision and policy making [2]. In order to achieve this,

traditional sensing techniques like Wired or Wireless Sensor Network (WSN), need an intensive usage of distributed sensors to acquire real-world conditions. Despite various and growing research, the high installation and maintenance cost, the low spatial coverage, as well as hard algorithmic issues [3], these techniques have never been largely deployed in urban contexts.

An enormous help predictably comes from mobile devices like smartphones, tablets and other more recent technological gadgets such as smartwatches and fitness bands. In fact, over the past ten years the evolution of mobile phones (mainly), have originated new solutions and services [4]. These devices are equipped with several sensors such as (two or more) camera, microphone, GPS, accelerometer, digital compass, light sensor, proximity sensor, temperature, barometer, humidity and it will be early the normality to have health-monitoring sensors as well [5].

Crowdsensing is a paradigm aiming to gather sensing measurements directly from people's

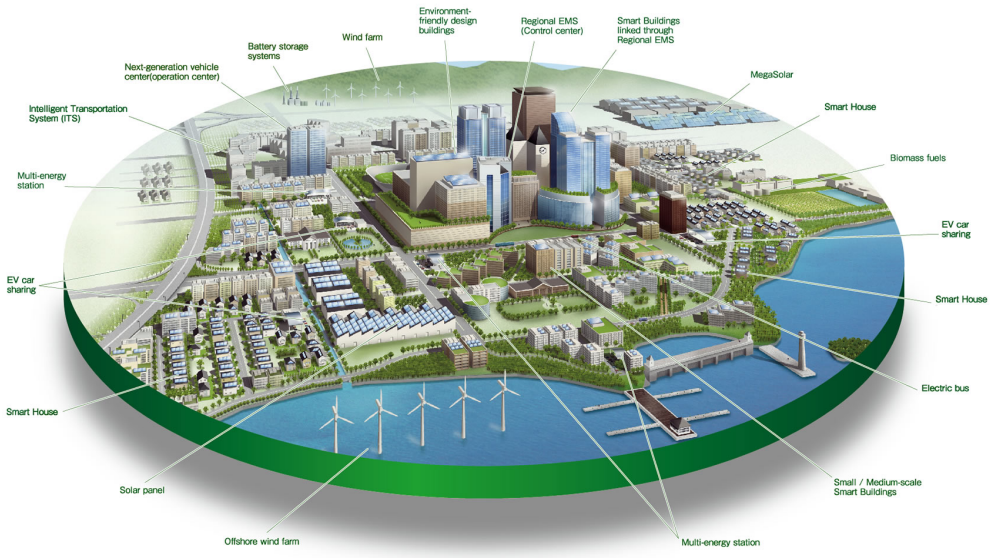

Figure 1.1: A Smart City portrayal [6]

mobile devices in order to provide clients (people themselves, institutions, governments, companies and so on) with advanced services. Mainly these data will be used for analysis and information extraction [4]. The looser energy constraints, the greater memory and storage and the extremely higher computational and network communication power of mobile devices compared to the standard little WSNs sensors, are remarkable fea-

tures, leading crowdsensing along the way of success. Furthermore, geofencing and activity recognition often represent, among the others, essential technologies supporting this emergent paradigm.

Toward meeting these challenges, in this document, I propose SenSquare, a Crowdsensing approach based on smartphone and a central coordination server for time-and-space homogeneous data collecting.
The rest of the document is organized as follows. In Chapter 2 the whole set of main technologies and the paradigms that leverage this project will be briefly described. Chapter 3 introduces many related MCS works both on classification (and taxonomy) and applications in different scenarios. In Chapter 4 we will see the system architecture, especially how it have been envisaged and the main concepts on which, SenSquare relies on. After that, in Chapter 5, the implementation has been described more in detail, with related examples for each part of the system. In Chapter 6, it is briefly illustrated a simple test physically carried on, to evaluate the performances comparing SenSquare with an ideal non-smart approach. Finally, Chapter 7 introduces many future improvements and Chapter 8 makes a brief overview conclusion on the whole work.

Chapter 2

Background

2.1 Environmental monitoring

Sensing and environmental monitoring were born with the necessity to collect data to observe and control environmental, productivity, technical and social parameters, mainly for military and ecosystem monitoring purposes.
As mentioned earlier, a common solution adopted so far is the deployment of a sensor network, especially in the recent form of Wireless Sensor Networks, also thanks to protocols and technologies such as 6LoWPAN, ZigBee, Ultra Wide Band and so on.
WSN, for his conception, has the need to link tens to thousands of radio transmitting sensors and cover areas that are difficult to wire for charging and interconnecting. The challenges in designing this kind of network involve the management of resources. One of the main areas of application unquestionably is the environmental monitoring for personal, institutional and scientific purposes. Crowdsensing comes in handy to extend and improve different kind of sensing applications and services where an infrastructure support is difficult or expensive to maintain.

2.2 CoAP

*The Constrained Application Protocol (**CoAP**) is a transfer protocol for constrained nodes and networks, such as those that will form the Internet of Things. Similarly to its older and heavier cousin HTTP, CoAP uses the REST architectural style. Based*

on UDP and unencumbered by historical baggage, however, CoAP aims to achieve its modest goals with considerably less complexity [7].
CoAP provides a request/response interaction model between application endpoints, supports the built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, strongly low overhead, and simplicity for constrained environments [8].

In these two statements, we can find all the main CoAP core's characteristics. Let us see more about this protocol and why a section of this document is dedicated to it.

CoAP was conceived mainly for IoT devices, that are battery-operated, constrained in computational power, RAM and storage capacity. Often protocols and applications are not designed to be optimized for running under these requirements, hence there are many performance issues.
Especially in wireless environments, many packet losses will occur, getting worse the

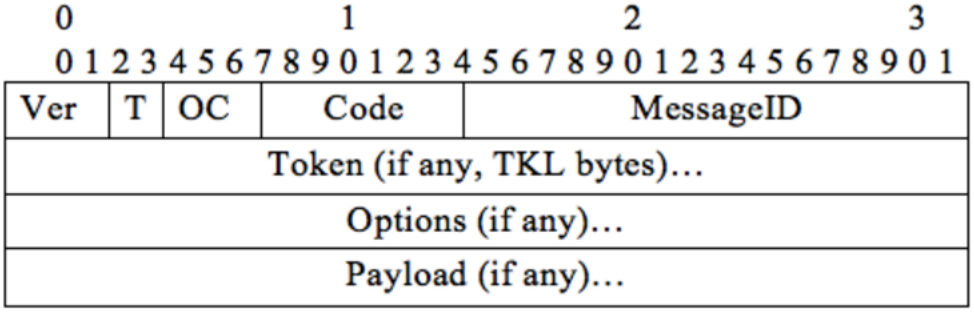

Figure 2.1: CoAP packet [9]

already critical power consumption of mobile devices. Furthermore, wireless standards need fragmentation, aggravating packet loss and adding other weight for headers.
Wireless devices and applications need power-efficient protocols, but existing protocols have typically been designed without power-efficiency criteria. In low-power wireless systems, the radio transceiver is typically the most power-consuming component [10]. Usually, in WSNs, a first and common approach is the tuning of the radio duty cycle,

sometimes with the protocols support. These methods allow to maintain a low power consumption, organizing the send/receive tasks all in the on-period, and in the idle time turning off (or decreasing power of) the radio transceivers.
Several devices are becoming connected, in order to build automation systems, mobile personal gadgets, cellular terminals and smart grids, considering that under the push of IoT, it is expected over the next decade, a growing to trillions of embedded devices [7]. Under this consideration, it has to be said that **6LoWPAN** and **ZigBee** are a remarkable breakthrough on the network side.
Unfortunately, networking alone is not enough to support the modern approach based on the Web Architecture with **REST** paradigm, especially in the form of **HTTP** protocol, which provides through simple URIs, both information and services.
CoRE WG[1], an IETF Working Group, has been working in these years with the purpose of making possible REST architecture for devices and networks with constrained requirements. Their main result is the Constrained Application Protocol, CoAP [7, 8].
REST architecture is so crucial that represent the *Web* as we know it today. Key features of REST are the following. First of all, beside the central *client-server* model, is the *resources* concept, everything is accessible through a URI. *Synchronous request/response* and *stateless* approach. *Layered system* and possibilities for deploying of *caching*, *proxying* and *redirecting*, seamlessly for clients. HTTP implements this concept through synchronous request/response methods such as GET, PUT, POST and DELETE to access on-server resources.

CoAP is conceived to keep all the handy REST features in HTTP, decreasing the cost for network and memory usage. In order to save batteries and to be lightweight CoAP has a fresh approach to the REST paradigm with a limited amount of requested resources. To do that, CoRE WG has built the protocol on the following bases.
Primarily, CoAP uses UDP with a simple message layer for lost packets retransmission and a little four bytes (binary) header within it. A common request employs from ten to twenty bytes for the header [7].
Clearly, all the main request methods of HTTP are kept (GET, POST, PUT, DELETE) and the response codes as well, but encoded in one byte only. CoAP uses URIs to identify resources, with the schema *coap[s]://*{*scheme-specific-part*} as HTTP does.

[1] https://trac.tools.ietf.org/wg/core/trac/wiki

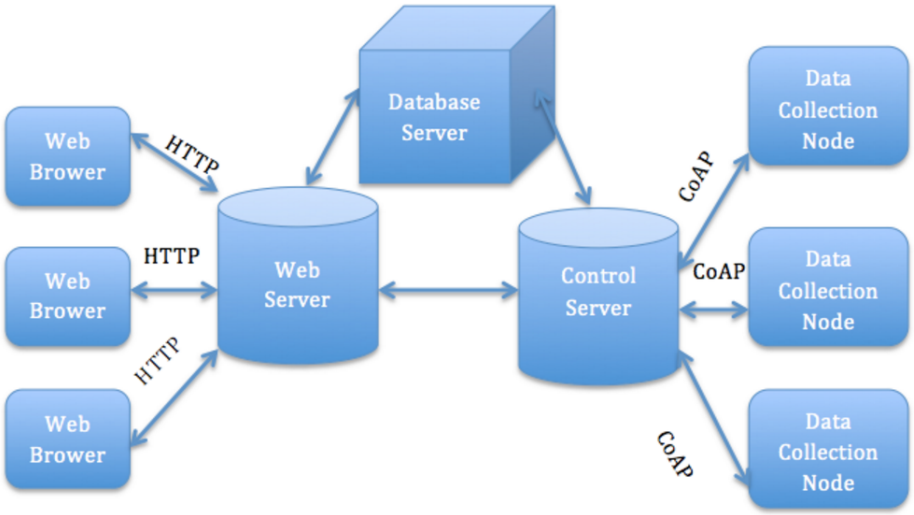

Figure 2.2: A typical architecture using CoAP [9]

Basic CoAP messages work better for small payloads, for instance for sensing data and similar [7]. Anyway, if the application requests larger payloads, UDP supports it through IP fragmentation, but CoAP prefers to exploit "Block" options dividing the data into multiple packets and keeping the server stateless.
An immediate advantage of this HTTP-like approach is the possibility to develop a CoAP IoT, or similar, architecture, connected with an existing HTTP service, through a simple proxy as a translate-middleware, "speaking" CoAP on the client side, and HTTP on the server side.
Further CoAP characteristics are, first, the Observer pattern, where the client can specify an interest in a GET request, afterward, the server will push information requested asynchronously without establish a whole publish and subscribe mechanism. Second, CoAP supports Machine to Machine communication, through resource discovery following a new IETF approach with the well-known resource path */.well-known/core* (RFC5785). Today, security on CoAP is demanded to the use of the protocol on the top of DTLS, similarly to SSL/TLS for HTTP.

In SenSquare I used CoAP mainly to address the problem of high power consumption, due to the huge amount of requests for each sensing data. The kinds of requests used in this project perfectly fit with the CoAP purposes. In fact, as we can see in a

moment, there are only little fragments of json both for data sending and little more features. Furthermore, in future developments, it is expected to use homogeneously the same approach for development on Arduino and other embedded platforms, keeping the examined advantages.

2.3 Geofencing

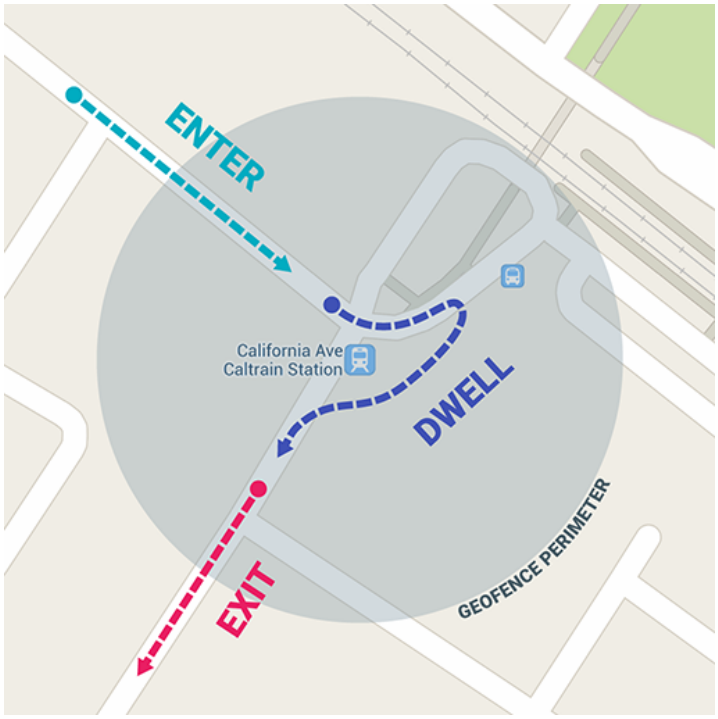

Figure 2.3: Geofencing modalities [11]

The core of SenSquare is conceived using CoAP for communication purposes and, among the others, geofencing technology to support sensing *spatial homogeneity* on the client, as described in the next chapters. This section wants to introduce and describe this location-based technology and the offered possibilities. In this document I am going to refer with *geofencing* according to the Android implementation [12], given that it has been used in SenSquare development.

In [13] we can find that **context-aware** computing is referred to *mobile computing in*

which users employ many different mobile, stationary and embedded computers over the course of the day. In this model computation does not occur at a single location in a single context, as in desktop computing, but rather spans a multitude of situations and locations covering the office, meeting room, home, airport, hotel, classroom, market, bus, etc.
The challenge is to exploit the user life's dynamic to devise a new class of application, in which the core is based on the users' current context. These new systems have to recognize the correct context in which they are, triggering the right behavior based on space, time, relationships, device and so on.
Context-aware applications, mainly, have to react to the environment answering the questions: *Where you are. Who you are with. What resources are nearby.* [13]. With geofencing we can answer the first question, moving forward to a new kind of location-based, context-aware applications.

A geo-fence is a virtual delimited area, in two-dimensional space, generally shaped as a circle identified by a central position (latitude/longitude located) and a radius. The major utility, using this technology, is the possibility to set user/developer-defined actions to trigger in one or more of the following situations:

- **ENTER**. When the user enters a geofence.

- **EXIT**. When the user gets out of a geofence.

- **DWELLING**. When the user is moving inside a geofence for a defined continuous amount of time.

Thanks to these features location-based services become proactive supporting smart notifications, action triggering and so on. Mobile devices, in this way, are able to act within an intelligent environment supporting the user by adapting its capabilities towards the situational needs [14], or in our case, creating an automatic system for trigger transmission of sensing data based on the user geographic position.

2.4 Military Grid Reference System

SenSquare space homogeneous sensing system relies on MGRS (UTM) coordinates. Let us see, in this paragraph, what is this reference system and why it has been chosen for SenSquare, instead of using the common geographic coordinate system.

For our purposes, independently from the granularity level and the size of the area, we do not need, primarily, the exact (centimeter-accuracy) position whence the data come from. It is crucial, instead, the possibility to associate a sensing data to this area, according to the kind of sensor we are monitoring and the granularity is requested from stakeholders. In other words, for instance, if we are monitoring the pressure level in *hPa*, we are not looking for the exact latitude-longitude identified position, we are, rather, aiming to estimate the pressure level in a kilometer-sized region.

GPS or other position provider services (WiFi fingerprinting, Telephony Cells estim-

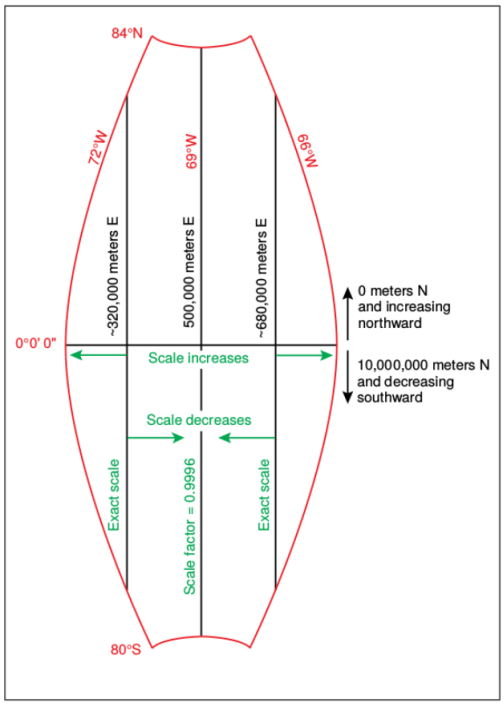

Figure 2.4: UTM grid example with zone 19 [15].

ate) give us position information in terms of latitude and longitude (and seldom height as

well). For most purposes, this kind of system is quite enough, however, when we need to plot position information on maps or we have to make further calculations using them, it is definitely useful to work with the corresponding grid on a map projection [15]. In our case, it has been chosen the Universal Transverse Mercator (UTM), one of the most widely used map projection and grid system. According to the U.S. Department of Defense decision, I used the Military Grid Reference System (MGRS) applied on the UTM grid.
UTM exists since 1947 when the United States Army adopted the ellipsoidal transverse Mercator projection and the associated grid UTM system in order to get rectangular coordinates on large-scale military maps covering almost the whole world [15]. UTM became a widely and popularly system used not only for military purposes but for civil uses as well, such as general surveying and mapping and navigation.

UTM divides the Earth into 60 zones, each generally 6 degrees wide in longitude. Omitted minor exceptions, the zones are numbered from 1 to 60 starting at 180 degrees longitude and proceeding east. The effort in creating the grid was in keeping zones narrow enough to lower distortion and scale variation within an acceptable level.
MGRS comes from UTM grid system (and UPS, Universal Polar Stereographic grid system for polar regions). MGRS is used to map the entire planet and uses a different labeling system from the UTM one.
To better explain MGRS, let us start with an example.
We can take the valid MGRS address ***33TUF 81384 62754***, which identifies a position in the ancient Medieval Caste of Gaeta, in Italy, as we can see in Figure 2.5. It is composed as follows:

33T	Grid Zone Designator.
UF	100km square identifier.
81384 62754	location 1m resolution, 81384 easting, 62754 northing.

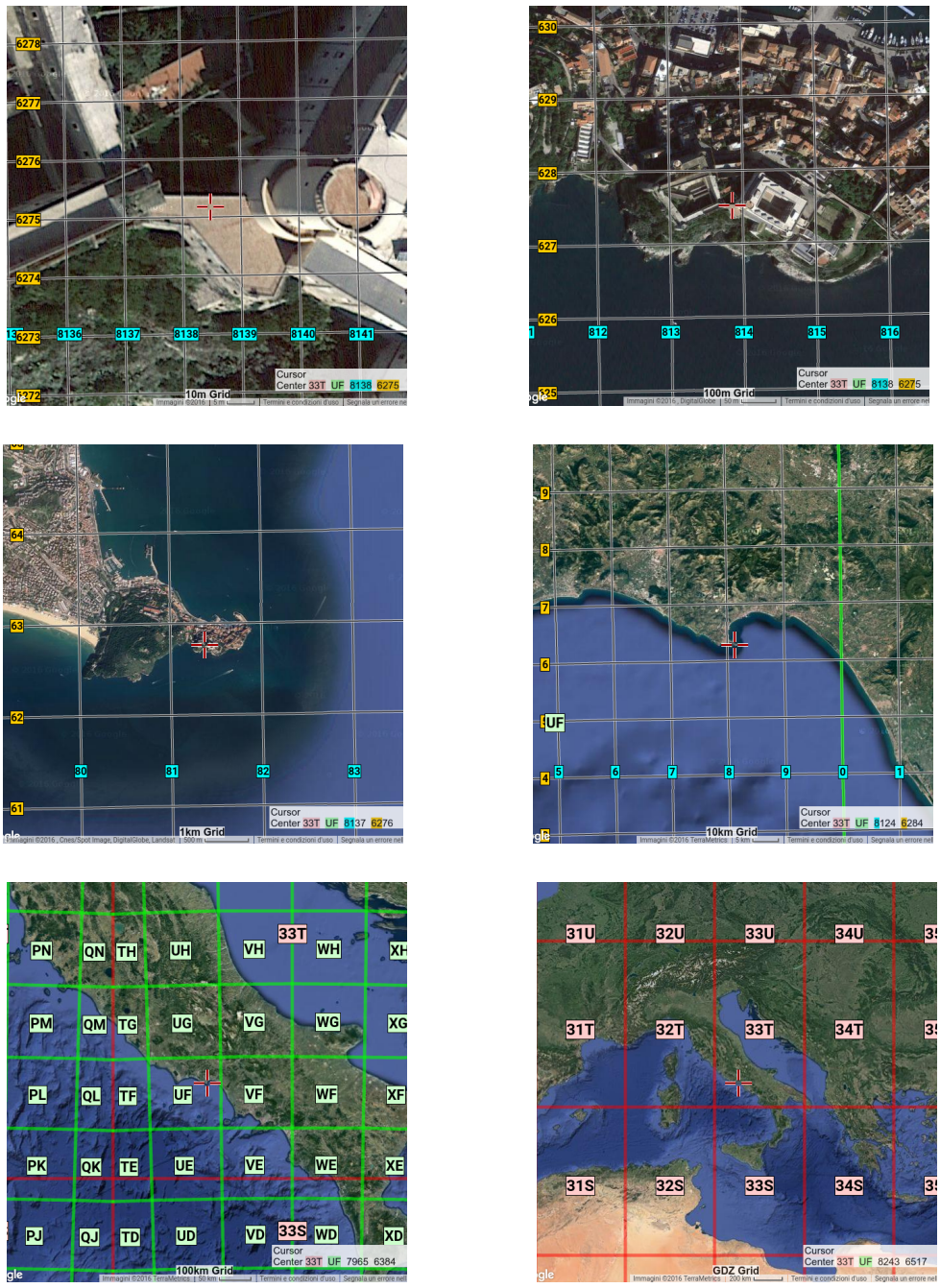

Figure 2.5: In left-right and top-bottom order. MGRS grid with 10m, 100m, 1km, 10km, 100km and GDZ grid on the same location.

33TUF	GZD and 100km precision grid square ID.
33TUF 8 6	10km precision.
33TUF 81 62	1km precision.
33TUF 813 627	100m precision.
33TUF 8138 6275	10m precision.
33TUF 81384 62754	1m precision.

The main advantage coming from MGRS is due to the infinite possible GPS coordinates, especially when calculus or manipulation is involved on position values. UTM divides the Earth into squares, and MGRS approximates each position identifying it according to different levels of accuracy. This is quite useful for sensing task, as we mentioned earlier.

MGRS, hence, divides the earth surface into 10km, 1km, 100m, 10m and 1m side length square, conforming to the specified precision, therefore the digits of the last part of the address could be 0, 2, 4, 6, 8, 10.

Due to the application of 2-dimensional geometry on 3-dimensional space itself, some adjacent square to a grid junction is clipped, in these cases area are better described by a polygon.

Finally, it is needed to know that when it is lowering on precision scale, MGRS needs truncation (on both easting and northing) rather than rounding, and that the southwest corner of the square, or polygon in general, is the identifying point for the whole area, even when the polygon is clipped [15].

Chapter 3

Related Works

3.1 Mobile Crowdsensing

To enter the world of Crowdsensing, it could start summarizing the more general concept of crowdsourcing, a paradigm impelling people to collaborate for a common project, with or without a monetary profit and without the necessary legal belonging to a company or association. Classic examples of crowdsourcing are open-source programming projects, the famous Wikipedia platform, the growing universe of crowdfunding for finance independent (and not) projects and so on.

Mobile Crowdsensing (MCS) is a paradigm extending the vision of participatory sensing in a crowdsourcing vision by leveraging both participatory sensory data from mobile devices and user-contributed data from mobile social networking services [2], fusing machine and human intelligence in this collaborative process. It allows the increasing number of mobile device users to get and share local knowledge information to collect for real-time or future analysis.
From a pragmatic perspective, in [16] we can find that MCS is *the ability to acquire local knowledge through sensor-enhanced mobile devices - e.g., location, personal and surrounding context, noise level, traffic conditions, and in the future more specialized information such as pollution - and the possibility to share this knowledge with the social sphere, healthcare providers, and utility providers.*
According to a formal definition in [2], MCSC (Mobile Crowd Sensing and Computing)

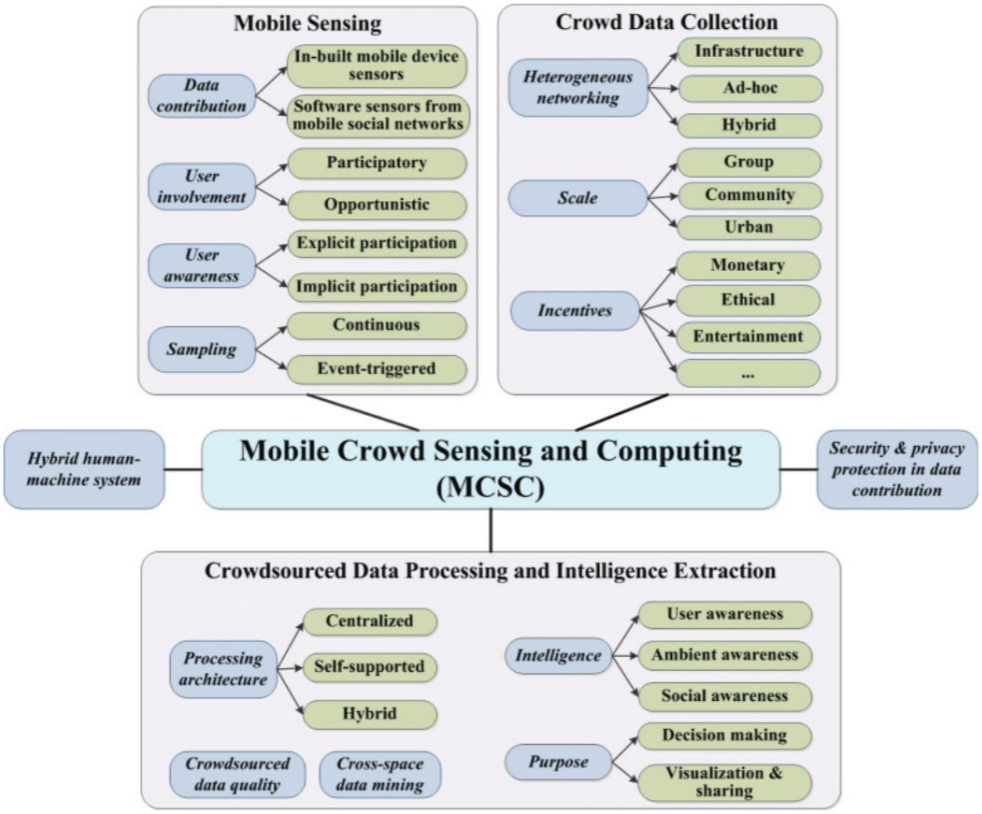

Figure 3.1: MCSC taxonomy [2]

is *a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices and aggregates and fuses the data in the cloud for crowd intelligence extraction and human-centric service delivery*.
Concluding this review around the possible definitions, in [17] we have a human-centric perspective of MCS, which is *Mobile crowd sensing is a new paradigm that takes advantage of pervasive mobile devices to efficiently collect data, enabling numerous large-scale applications. Human involvement is one of the most important features, and human mobility offers unprecedented opportunities for both sensing coverage and data transmission. [...] Compared to traditional sensor networks, human mobility offers unprecedented opportunities for both sensing coverage and data transmission*. Similarly in [4] *Mobile crowdsensing refers to a broad range of social and community-based sensing paradigms employing mobile devices and wireless networks. Different from conventional sensing solutions using specialized networks of sensors, mobile crowdsensing aims to leverage human intelligence to collect, process, and aggregate sensing data using individuals' mobile devices, so as to realize a higher quality and more efficient sensing solution*.

It is possible to discern different types of MCS according to different points of view.
A first kind of classification is done upon the user *consciousness* in collecting data [18]:

- **Participatory sensing** is referred to an MCS paradigm in which users are completely involved in the application running lifetime, deciding how, where, when and what to sense or get data for the system.
- **Opportunistic sensing** require, instead, an unconscious usage of the application, which always runs in the background, opportunistically getting data without user involvement.

Looking at the kind of transmission support, we can find another classification [17, 18]:

- **Infrastructure-based transmission** uses infrastructure support for Internet connection like cellular networks (3G/4G). In this MCS paradigm, there are the classic problems related to the constraining economic cost and bandwidth usage.
- **Opportunistic transmission** enables, instead, transmission of data through short-range radio communication support, such as WiFi, Bluetooth, ZigBee. Opportun-

istic transmission avoids the costs related to telephony company subscription but suffers intermittent connection like any other best-effort service.

In [19] we can find a categorization related to analysis time:

- **Offline** analysis, the most common and older solution provide for the input analysis process in deferred time. Classic examples are transportation activities in urban space, health, and physical assessment, environmental parameters monitoring.
- **Online** analysis, on the other hand, provide a real-time analysis, this is pretty useful in scenarios like traffic monitoring, public safety, and collaborative searching.

Scale	Data Quality	Collaboration	Social Tie Strength	Key Techniques	Major App Areas
Group	Medium	Close	Weak or strong	Opportunistic connection and group formation	Local area sensing, local event replay
Community	High	Close	Medium	Community creation/detection	Scientific research, community-specific services
Urban	Low	Loose	Weak	Data quality, trust maintenance	Urban dynamics sensing, environment monitoring

Figure 3.2: MCS Categorization [2]

In [2] other classification are identified according to:

- **Data generation mode**:
  - **Mobile Sensing** exploits the sensing functionalities (included antennas, microphones etc.) of the device, providing a data completeness related to its technological level.
  - **Mobile Social Networking** approaches leverage on user spontaneous contribution, through direct intervention or through other platforms such as social networks.

- **Sensing style**:
  - **Explicit Sensing** is the sensing style in opportunistic/participatory modalities, where the user is aware to sensing task and data collection is the main goal of the service. In this case, the sensing task is explicit to the user.
  - **Implicit Sensing** performs a kind of hidden data collection, the user is focused on the usage of the platform for social interaction. In this case, the data generation is a secondary task and so it is implicit for the user.
- **Volunteer organization**:
  - **Group sensing** is referred to a modality in which user are loosely or opportunistically organized (for instance for space or time criteria) in order to address a data collection for a shared problem. An example for this could be a group of random people passing, over the time, in a precise zone of the city which needs a sensing task for a set of environmental parameters.
  - **Community sensing** is based on the "community" concept, according to the Cambridge Advanced Learner's Dictionary community is *the people living in one particular area or people who are considered as a unit because of their common interests, social group or nationality*. It expected from this modality a high-quality of data, for the nature of community itself, who perform tasks in a higher interactive and spontaneous way. For example, a college community can be treated as a unity in the task of monitoring their school quality, supposing the existence of a common goal, the better life in their structure.
  - **Urban sensing** has often a lower quality level of data compared to the previous models, despite the broad spread. This is due to a low fostering in using the service and to a scarce sense of belonging to community.

SenSquare is designed as a mixed form of participatory and opportunistic sensing; users can, independently, take decisions about which sensor data they want to share and which stakeholder can benefit of. Regarding transmission type both the modalities described are used, in order to achieve a better coverage sensing map, anyway, on cutting-edge

devices is possible to autonomously choose which kind of connection technology allows for each single application. The project is also designed to support both offline and online analysis, considering that all the data are immediately sent to the server and directly accessible through specific API or REST calls. The data generation belongs to the first class, exploiting all the available sensors. The sensing style is explicit and the system is designed for an urban audience, because of the Smart Cities nature itself.

In [20] the mobile cyber-physical systems (CPS) are discussed as *integrated computing and communication systems that process and react to sensing data from the external physical environment and transform the way humans interact with the physical world*, enhanced with mobile features and exploiting strongly the Crowdsensing paradigm. They also identify a classification upon the kind of MCS application:

- **Vehicular Social Networking** which uses the regular and predictable drivers' routes to collect data in order to measure traffic congestion, identify the real-time best route and detect accidents supporting a quicker medical intervention.

- **Environmental Monitoring**, not only in an implicit way but also exploiting the awareness, leveraging on the user's cooperativeness, especially when the system is deployed for communities.

- **Disease Report and Crisis Management** is one of the most interesting and appropriate field for MCS spreading, considering the richness of sensors on mobile devices, for instance, *the Ministry of Health in Cambodia uses GeoChat, a crowdsourced sensing interactive mapping application, for disease reporting and staff alerts which enable rapidly escalated responses to potential outbreaks. Also, Ushahidi has been used to crowdsource and map crisis information from multiple sensing data streams in real-time through mobile devices, so as to coordinate field teams' activities and provide remote support from outside an earthquake zone* [20–22].

We can also find a further classification, as described in [5]:

- **Smart Cities** exploit the high density and heterogeneous population to *improve city efficiency by deploying smarter grids, water management systems and ulti-*

mately the social progress. [...] People can actively participate in sensing campaigns to make their cities safer and cleaner.

- **Road Transportation**, beyond the real-time traffic information, uses MCS, in particular through GPS, accelerometer, and gyroscope in order to quickly identify and repair uneven roads.

- **Healthcare & Wellbeing** can help both for real-time first aid and long-term epidemiological or medical research thanks to collecting a big amount of data, also linked to activity recognition systems as well. Besides this, data collection and analysis has interesting applications for fitness and wellbeing in general [23, 24].

- **Marketing/Advertising** is a cutting-edge field of study for the impressive economic returns. MCS fused with activity recognition and location management (context-aware solutions) generate mobility routes and patterns incredibly useful for companies and vendors in order to adapt commercial strategies to social behaviors.

Let us move on which are the key features, and related issues, affecting MCS design and development. Outstanding obstacles in the growing MCS expansion, are identified for instance in [16] and [19]. Let us proceed in a short review about them.
First of all, the **heterogeneous** mobile devices and sensing hardware population leads to enormous difficulties from designing to deployment. Just the different operating systems installed on the devices, both for the brand and version itself, make up a hindrance for a larger spread of applications, dealing with incompatibility and portability problems. An application lifecycle is already a hard task in itself, support and maintenance for so different platforms is onerous in time and, often, in money as well.
Later, another issue regards the **burden** of an MCS installed app for the users. Without considering that for each MCS system, today, a user has to install a separate and specific application, it is clear that battery lifetime is a well-known open problem for smartphone and similar devices. Users are also conscious that each application steals processing and memory capacity to other apps; modern operating systems, on the other hand, are designed for self-contained products, making inter-application data sharing difficult because of justifiable security reasons.
Finally, the **increasing network bandwidth** demand is a big issue for a large scale

MCS. Emerging applications, not only for sensing, require more and more data to send and receive; this is primarily due to the nature of the data, often multimedia streaming such as video and music, and later for the intensive network usage requested for better accuracy, performance and so on.
In [19] the proposed solution relies upon three principles: separation of data collection and sharing from application logic, removal of application installation from the deployment and decentralization of processing and data aggregation.
My solution through SenSquare adopts this last principle, combined with a lightweight communication protocol and an intelligent server logic.

I decided to briefly debate the crucial issue of **User Motivation** individually. This problem is faced in many works such as [16], [17], and deeply in [25], which treats the economic incentive for the MCS user (as data provider) and in [26], in which the authors build up a detailed dissertation upon the fostering problem.
Mainly when users' devices have a limited amount of resources (battery, memory, computation), or the information requested are particularly sensible (position, photos, medical parameters), users properly demand a kind of remuneration, incentive or benefit to installing and use the MCS application. For these reasons construct a fostering system or an economic model is crucial for the system success, mainly because, for definition, Crowdsensing request a large number of "volunteers".
Often an economic incentive is a right way, especially when the volunteer organization model is the Urban one. The incentive could be directly monetary, sale-based, company fidelity-based (in SenSquare this is the designed way) and so on. Marketing in this field has no limits.
Another fostering model is related to **Gamification**[1]. The user application, in this case, could be developed using the main Gamification principles, for instance providing badges, levels, goals, ranking. This fostering modality mainly attracts the youngest segment of population, and has a great success on workplaces although is, currently, a recent research field yet [27–29].

[1] Gamification is the application of game-design elements and game principles in non-game contexts. Gamification commonly employs game design elements which are used in so-called non-game contexts in attempts to improve user engagement, organizational productivity, flow, learning, employee recruitment and evaluation, ease of use and usefulness of systems, physical exercise, traffic violations, and voter apathy, among others. [from Wikipedia, *https://en.wikipedia.org/wiki/Gamification*]

User participation represents one of the most important elements in MCS. Considering the consumption of resources, the personal data transfer and the exposing to potential threats, the fostering modality and the decision of the called *minimum price* is a well-known problem to address properly.

As we saw so far, WSN or other sensor networks require specific hardware, protocols, and applications to cover a large area for monitoring of the environment, roads, social behaviors and so on. Performances are heavily dependent on the number of sensors and a scarce coverage leads to a low quality of the sensing task results [25]. Smartphones and mobile devices in general are born with embedded sensors, growing in number and quality as time passes. Wireless and 3G/4G connectivity, GPS, activity recognition, provide with sensors a perfect mixture for an MCS exponential growing. Crowdsensing is the perfect tool for address sensing task in an urban environment to support a more general Smart City environment, considering problems and characteristics that make it unique.

3.2 Applications

Collaborative Internet of Things is, always more, a growing field of study, gaining interest as solution breaking down the barriers of isolated IoT ecosystems.
As I mentioned before, like it happens for MCS, also, in general, IoT-based architectures suffer a low inter-architecture interoperability. In [30] a general comprehensive paradigm is proposed. MCS is a perfect candidate to become part of this revolution, thanks to a little amount of requested resources, an easier deployment and a large population, which under the correct stimulus, can collaborate with restrained costs.
We saw that MCS is a recent concept that exploits the sensing (and not only) capacity of personal mobile devices, such as smartphones, in-vehicles facilities, and wearables, giving us a potentially huge amount of device (or manually) produced data. When this data, after the appropriate gathering and analysis, become information, the common interests on it are definitely remarkable.
Although the above considerations upon the theory of MCS, the term is common referred, in literature, on the two paradigms described earlier: **participatory sensing** (crowdsourcing in an user-awareness way) and **opportunistic sensing** (not involving

the user in the sensing tasks). MCS approach is conceived and implemented to map phenomena of common interest, which need a significant amount of data, in a heterogeneous way regarding the different points of view to describe events/situations, and in a homogeneous way regarding the as larger as possible sensing sampling. This has been demonstrated to be efficient in several fields of application where the IoT is already the key technology set that tackles the most challenging tasks [31].

Most of the remarkable applications of MCS are designed to support **Smart Cities** concept. By now it has become straightforward that the increasing density of population with related problems make big cities management a strenuous assignment for institutions. Solutions for Smart Cities are heterogeneous and range on the entire spectrum of city and personal life.
Environmental monitoring is the natural application of MCS paradigm applied to Smart Cities, considering that most devices natively support environmental sensors. Several applications, such as SecondNose [32] and studies like [33] and [34], have been proposed in order to improve the environment conditions, starting from data sensing collecting, sometimes trying to sensitize directly the interested population. In *Park Here!* an example of advanced smart parking systems is provided, through which users can be informed in real-time about the presence of vacant parking spots close to their destinations, in order to reduce the high level of stress and decreasing air pollution. Often, activity detection is used for this purpose as well, based on gyroscopes, accelerometers [35], sonars [36], magnetometers [37] and on-board cameras. Other crowdsensed user's transportation mode detection in smart cities is also interestingly addressed in [38,39] regarding activity-recognition system itself and in [40] implementing a location and activity-based mobile advertisement service.
Especially in Smart Cities, MCS can reduce the costs associated with large-scale sensing, providing at the same time additional human-related data [5].

As just mentioned with *Park Here!* **Road Transportation** is an highly potential field of application for many MCS applications [41]. Institutions and transport authorities can strongly benefit of real-time traffic information [42], constructing fine and large grain maps in order to support traffic engineering, roads and cycle lanes construction and so on.

Healtcare has even found advantages in MCS, such as in [43], in which patients are collecting measurements about their daily activities in order to provide a significant dataset to be analyzed by doctors. This study, in particular, is focused on a specific annoying disorder called *tinnitus*.

Finally, **Marketing & Advertising** is relying on MCS for new and effective kinds of strategy. As mentioned above, Geospot [40] is a perfect example of mobile advertising system which uses sensing for activity recognition and geofencing. Stakeholders as vendors and companies, in general, are definitely interested in human traces and patterns to implement context-aware solutions.

The common approach in designing an MCS architecture is to divide the system into two interdependent pieces according to a client-server paradigm. A cloud backend normally expects data in a particular format from its respective sensing client, which, this last, can only talk with its respective server. In this way, there is any kind of vertical interoperability among different application, neither a common protocol is established in order to achieve a similar result.
Is straightforward what kind of waste this approach causes. Some architectural attempts have been proposed, for instance, McSense [26], a centralized MCS system that exploits monetary rewards to make the backend entity assign sensing tasks to the users. In this document, through SenSquare, we are going to address these issue as well, in order to give a better solution for this interoperability purposes.

As addressed in the previous section, we know that social and monetary incentive is essential to push the user toward a better collaboration [26, 44]. It is clear, indeed, that MCS is effective and efficient if and only if the penetration of the application is as wide and permeating as possible.
Exploiting monetary rewards (refunds and actual payments) is an immediate way to foster a massive collaboration, in [45] it is described an approach based on **micro-payments**, whereas in [25] is based on reverse auction **dynamic price** with virtual participation credit and recruitment algorithm. In this reverse auction, participants want to sell their sensed data to an auctioneer, which, wants to buy the least expensive meas-

urements. To make this process equal, after such cycle winners raise their prices and losers lower theirs.

Like gamification does, other approaches address the fostering issue relies on the personal interest of users, or anyway, on their entertainment using the application.

Chapter 4

System Architecture

This chapter illustrates the SenSquare architecture, describing how it has been conceived and outlining the modules which compose the whole system.
SenSquare is designed to be a star topology system, in which **sensing clients** can only communicate with a central server, hence there is no link of any kind among them. The central server is called **Central Coordination Unit** (CCU). CCU is the only element in the system designated to have reasoning capabilities or any kind of intelligent behaviour.
As just mentioned, client device entities are the sensing elements of SenSquare. They rely on a minimum set of sensing hardware, one or more current location retrieval (and geofencing) mechanisms and network connection to communicate with CCU. Such devices can be smartphones, tablet, wearable and in the near future, embedded devices. This is the basic components of MCS SenSquare system.

Take part in this scenario also the **stakeholders**, a different type of client in SenSquare, we can call them **interested clients**. Stakeholders orchestrate the meaning of the sensing data, in order to achieve their singular and specific purposes. To do that, they can push **rules** to the CCU, specifying sensor, time and space related parameters as described in detail, shortly. Users, equipped with their personal sensing client devices, can subscribe to one or more stakeholders offering data in exchange for some kind of revenue, previously set among the parts and depending on the respective needs.
CCU main task consists in gather and store, in a database, sensing measurements com-

ing from sensing clients. Crucial, right after this incoming direction part, is the process to determine the correct answer for the client. What sensing clients expect from the server, after a measuring sending, are the time and space update criteria, related to that sensor.
More in detail, a sensing client continuously gets measurement updates from its sensors. Whenever it sends this values (one communication for each sensor update), CCU replies to them with a *time interval update condition* (time after the new measurement for that sensor has to be re-sent) and a *space update condition* (a zone, in which the client currently is, outside which the new measurement for that sensor has to be re-sent). It is clear that the space condition is worthless for non-mobile devices.
It is needed to better specify the time and space conditions mentioned above, consisting in the core aspect of MCS SenSquare intelligence. These conditions are based on the set of stakeholders (or system default) inserted rules. How the set is composed, that is, which of all rules stored in the CCU's database, belong to a set, depends on kind of sensor, user's position, current time, user's subscription. Let us see why.
First of all, a rule is defined by a tuple, defined as follows:

$< stakeholderID, sensorType, addrMGRS, areaMGRSGran$
$spaceSampleGran, timeSampleGran, timeExpire, countExpire >$

- **stakeholderID** is the identifier of the stakeholder who inserted the rule and that owns it.
- **sensorType** is type of sensor, requested to monitor.
- **addrMGRS** is a valid MGRS address, it has no meanings without the next field.
- **areaMGRSGran** gives a semantic to the previous field. If *addrMGRS* identifies a specific 1m precision location, this field is the **mask** to apply on it, giving so the zone which is requested to monitor. Making an example, if the address is *33TUF 81384 62754* and the mask is *2*, the MGRS area within the rule is valid is *33TUF 81 62*.
- **spaceSampleGran** defines the size of the area within a single measurement for a rule is valid. The sensing client will receive a zone for geofencing exit trigger

related to this value, hence, it will have to send the new measurement for the given sensor, once exited the zone it currently is in. If *spaceSampleGran* is *4*, sampling within this area has to be 10m sized. Obviously, this field can not contain a value lesser than *areaMGRSGran*, because the sampling would concern a bigger square than the monitored area itself.

- **timeSampleGran**, similar to the previous field, defines a time period in seconds, after which the sensing client has to send the new measurement for the given sensor.
- **timeExpire** defines until when, as timestamp, the rule have validity.
- **countExpire** defines up to how many measurements, the rule have validity.

The **Sensing Infrastructure** is defined as the set of all mobile and static devices participating actively in this scenario. Being crowdsensing, part of the general crowdsourcing concept, it can be assumed that sensing clients are mostly private users, sharing data and consequently battery, computation, memory, and storage for our purposes. Users, in this architecture, can submit voluntarily their data, choosing also the stakeholders that can take benefit from them. As mentioned at the beginning of this chapter, the voluntaries expect a reward from this exchange, on the other hand, stakeholder clients request a minimum of reliability from the data that users provide. As described in the related works chapter of this document, there are many strategies to face these problems and we leave them as future enhancements, complementaries to this project although as interesting as it.
Depending on what and how many stakeholders the users subscribe (instead of just keeping the default ones), device resource consumption could be onerous. Thereby, it is needed, some mechanism to avoid this problem, especially with numerous subscriptions and stakeholders' rules in the database. CCU is delegated to handle this task.

Before explaining how this problem is addressed, let us consider an example.
An internet service provider decides to develop a crowdsourcing city WiFi for the subscriber clients. Each customer, accepting the terms and conditions, will share a part of its home connectivity with other customers of the same provider. On the other side, these customers can take benefit of this sharing when they are not at home, using the

other customers' connectivity. To deploy this system in the first city, the provider company, first want to map all the territory in terms of WiFi coverage. They, hence, need the average wireless throughput with a precision of 10 meters and a rate of few hours in all the zones they want to map. They also want the BSSID and the RSSI of the sensed access points, to evaluate the signal strength in as many points as possible. After inserting the rule on the CCU's database and created an effective fostering plan to get subscriptions, they have to wait for the proper amount of data to take strategy decisions.

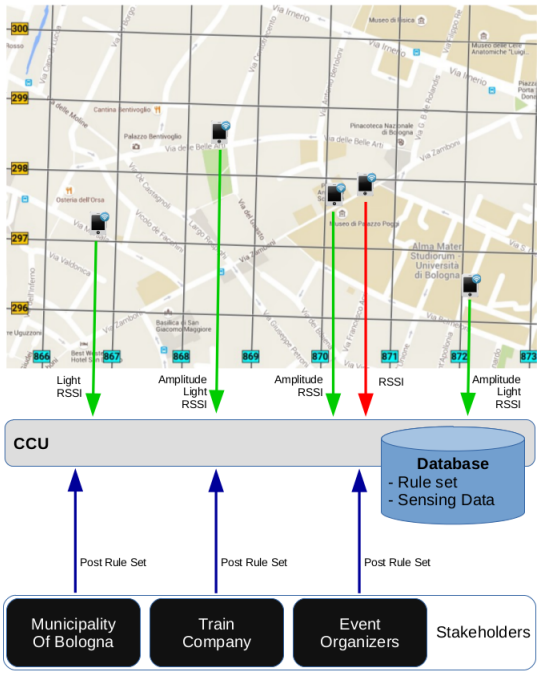

Figure 4.1: System architecture. The red arrow represent the redundancy avoidance through the rules in the same area.

4.1 Central Coordination Unit

One of the core goals of SenSquare, as seen so far, is to reduce as much as possible the energy consumption of the sensing clients. On the computational side, it has acted concentrating most of the burden generated by the ecosystem on the CCU, giving thus, the complete control of the scenario to it. I developed the python CoAP/REST server together with a MySQL database, for rules and sensing data, as the CCU.

Most of the computational cost is due to responses that clients expect to receive after a measurement sending. This response includes a timer and an area, that, as we stated before, characterize when and where the next update for the given sensor is required.

Timer Different sensor measurements need to be updated according to different timers. It is clear that the ambient temperature or pressure value, do not require frequent updates, being a slow changing number. Instead, for noise level, it is needed to have always fresh data in order to monitor, conveniently, frequently changing noisy locations.

Zone Assuming mobility on sensing clients, and the MGRS zone labeling introduced above, SenSquare uses zone constraint to achieving a homogeneous sampling of the territory. We know the MGRS hierarchically divides the word into squares (or polygons) uniquely identified with the encoding seen earlier. A great advantage of MGRS comes from exploit this encoding applying a mask to scale or descend in the space hierarchy. Considering an amount of digits (on both easting/northing last parts), according to the mask, identifies different sized square in the hierarchy, always containing the full encoded location (1m square zone).

For instance, Bologna belongs to the part of Italy having Grid Zone Identifier *32T*, and the city itself is assigned the 100km sided square *32T PQ*. A device sends to CCU a latitude/longitude position that server translates into the MGRS identifier *32TPQ 57314 29575* (always the maximum precision after translation). We can infer that the device is in Bologna, and that it belongs, among the other, to the 10km sided square *32TPQ 57 29* ($mask = 2 => 1km$).

When the CCU calculates which zone has to be sent to the client in response, acts as follows. First, it takes all the rules related to the right sensor, both default ones and stakeholders subscribed too. Among them are taken only those contain the client position. Finally, considering that there still could be multiple rules satisfying these criteria and that we want to avoid useless energy consumption on the client side, is reckoned just one zone. The zone chosen is always the smallest sticking out from the rules, that is the one which satisfies, with the finest grain, all the other rules requesting that sensor measurement.

Similarly, the shortest timeout (finest grain in time) is chosen to fit all the others.
Let us define more formally the tricky task of zone calculus. CCU apply for each rule the operation $addrMGRS||areaMGRSGran$ (where $||$ is the mask operation), to obtain the validity area in these rules. Then, it identifies if the client position belongs to them and searches for the littlest granularity requested in these rules, identifying so, the square in the hierarchy satisfying all the rules, consequently. Granularity goes from 0 to 5, where 0 means "remove all the digits east/north" and keep a 100km sided square, and 6 means "keep all the digits east/north" and keep a 1m sided square.
Summarizing, two rules *R1* and *R2* are both applicable to the same measurement if:

$$\begin{cases} \{R_1, R_2\}.\textit{timeExpire} \geq \textit{currentTime} \\ R_1.\textit{sensorType} = R_2.\textit{sensorType} \\ R_1.\textit{validityArea} \subseteq R_2.\textit{validityArea} \end{cases},$$

where *validityArea* is the result of mask operation and $\subseteq$ the geographical inclusion between square areas.
If a set $R_1, \ldots, R_n$ of rules are all applicable in this way, server can answer the client with the area defined by $max\{R_1.\textit{spaceSampleGran}, \ldots, R_n.\textit{spaceSampleGran}\}$ and timer
$min\{R_1.\textit{timeSampleGran}, \ldots, R_n.\textit{timeSampleGran}\}$.
Once obtained the minimum square, as described, is calculated a circular area, that is the circumscribed circle for this square, using the half of the diagonal as the circle's radius. The center (in latitude and longitude coordinates) and the radius (in meter, of this circle) is finally sent to the client.

4.2 Mobile Application

Smartphones, body gadgets, but also car and other IoT devices, are increasingly a powerful set of environmental sensors too. They compose a perfect scenario for an MCS scenario. Here, I am going to present my deployment for a demonstrative mobile application sensing client, developed in Android environment.
Once started, the application checks the availability of the sensors we want to use for our crowdsensing purposes. Then, it starts all the routines to obtain constantly the necessary

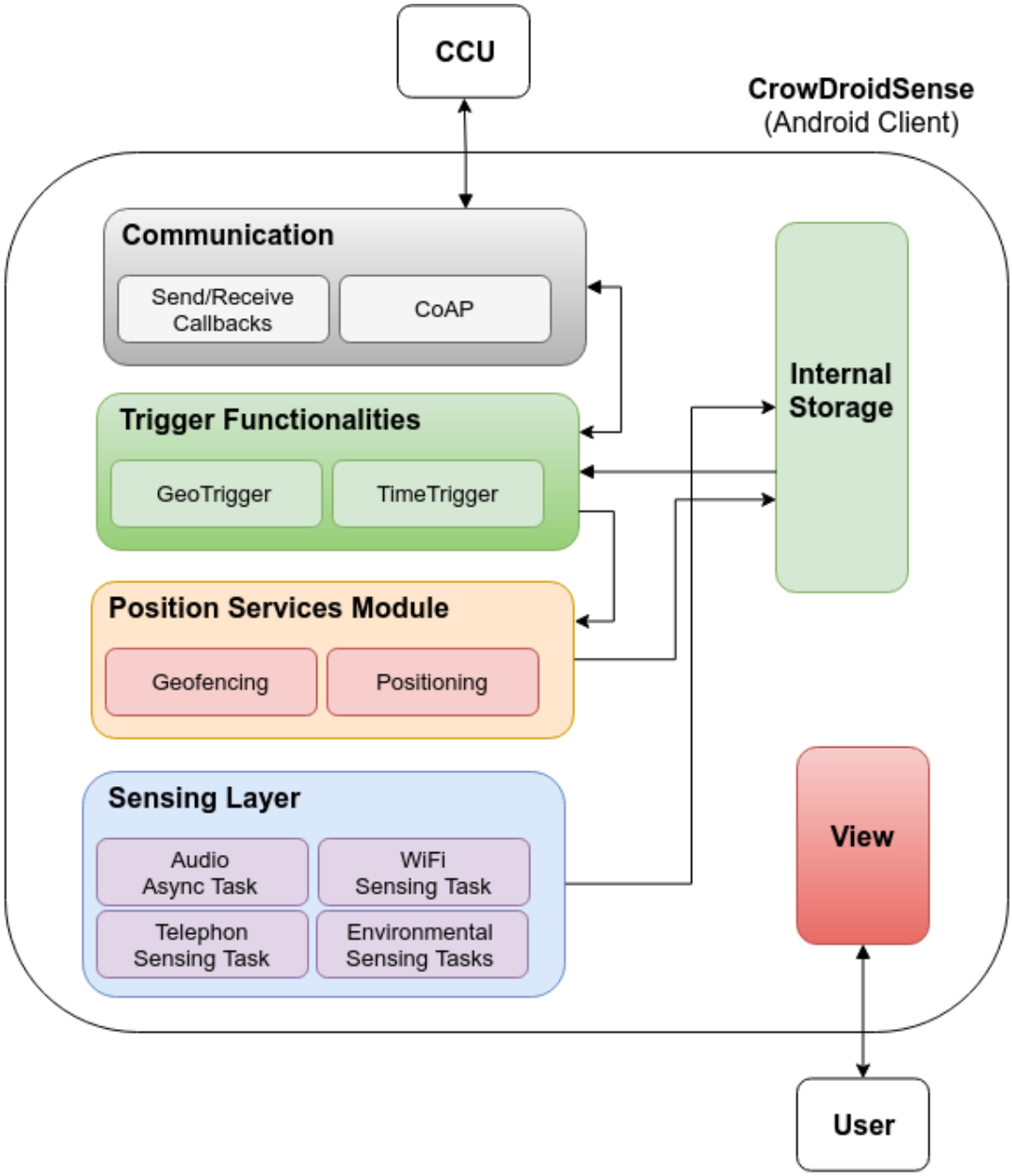

Figure 4.2: Mobile Application architecture

measurements of these sensors, most of which, are approachable only through asynchronous requests. For debugging purposes (nowadays temperature/humidity sensors are seldom included) I also developed a *stub* to provide mock measurements for unavailable sensors in order to fully test the system. When a measurement is available, it is sent to the CCU through a lightweight CoAP request encapsulate in a JSON format, including other information, as we are going to see in the Implementation chapter. When the application performs the first start, all the data are sent almost simultaneously.
When the CCU receives a request containing the JSON encoded data, as we saw in the previous section, it calculates a timer and a circle sized area identified through a center and a radius. Another JSON with these data is sent, as CoAP response, to the client. The client now can decode the JSON extracting timer to set the re-sending of the sensor measurement, and similarly, the geofencing area outside the which the re-sending will be triggered as well. We know that this circle area is calculated taking the finest grain rule matching the conditions as seen. Once taken the designed MGRS square delimiting the validity of the measurement, the center will be the MGRS 1 meter precision address (translated into latitude/longitude coordinates) and the radius will be the half of the diagonal. Specifying, geofencing uses circle areas, hence CCU gets the circumscribed circle for each of the MGRS square areas, taking their half diagonals as radius for the client.

The sensors we consider in SenSquare are relative humidity, temperature (both hard to find on existing devices), barometer, light. Other measurements are ambient noise (through microphone), RSSI/BSSID/SSID of personal and public WiFi infrastructural networks the user is connected to, operator/RSSI/throughput of the cellular network the user is connected to. The latter is a kind of OpenSignal[1] crowdsource stations monitoring. Throughput measuring, differently from the most of other measurements which happen constantly or asynchronously by the operating system, is done only when requested, reducing the power consumption and network utilization, that can, also, be monetarily expensive.

Each user can also subscribe one or more stakeholders. If the user does not choose to do that, CCU will answer to its requests only using the default rules (for example

[1] https://opensignal.com/about/

those inserted by the institution owning the system). Currently, each user can only choose to concede its sensing data in full to stakeholders subscribed. One of the future enhancement will be the possibility to choose each single sensor measurement for each stakeholder, giving to the user a finer grain permission setting. For instance, a user is willing to concede WiFi and cellular network data to its telephone company in exchange for some benefits, but he will not provide other kinds of measurements for the least privilege principle.

Chapter 5

Implementation

As we have seen in this document, SenSquare is composed of two main parts, a Central Coordination Unit (a MySQL database and a running Python server) and, currently, an Android sensing client. First of all, it has been designed and developed a basic Android application able to get all the requested measurements in different ways. Once this "draft" application was ready to run, the focus has been moved to the database implementation, with tables for rules, stakeholders, measurements and so on. The next step it has been the Python server, core of the system. It links the client measurements to the semantic logic and the database storing and retrieving. Finally, it has been developed the communication modules for both client and server in CoAP as debated along this document.

5.1 Central Coordination Unit

5.1.1 Database

The database, initially based on a simple SQLite to start the first test, it has been finally developed in MySQL and is, currently, composed of the following seven tables.

- **sensors** contains all the sensors we are interested in, comprehending their related information, such as a numeric identifier (mostly corresponding to the android sensor ID), name, unit of measure.

Sensor	ID	Unit
Light	5	lux
Pressure	6	hPa
Humidity	12	%
Temperature	13	°C
Ambient Noise	100	ND
WiFi	101	dB
Telephone	102	dB

- **all_sensor_data** contains all the client sensing data coming from real environmental sensors (light, temperature, relative humidity, pressure) and microphones. This data are all homogeneous in terms of fields, they have a unique identifier, a user and sensor type identifier, a location (both in latitude/longitude and in MGRS), a timestamp and obviously, a value.

- **all_tel_data** contains all the data relative to telephony cells sensed by clients. They have the same field of the previous table, besides specific value like operator name, the strength of the signal, the technology used (2G, 3G, 4G) and throughput.

- **all_wifi_data** have the same characteristic of *all_tel_data*. but the values are the SSID, BSSID, the signal strength relative to the user current connection (if connected to a WiFi network).

- **stakeholders** contains all the information about entities, authorities, companies involved in the system as clients of the sensed raw data.

- **rules** contains all the rules as described in the previous chapter.

- **subscription** contains, finally, the references of the client subscriptions to the stakeholders. In a future development, this table will be enhanced with a finer grain filtering on the specific sensor to monitor.

5.1.2 Server

The whole server is developed in Python 3.4.3. It has been decided to rely on this language to evade from the common choices as PHP is, keeping a stable and high performance server-side. Python (especially in the chosen version 3) provides also a huge amount of libraries, and finally, supplies an easier way to rapidly build, even complex, architectures, with small constructs and highly readable code.
The server relies on many specific libraries, which have been chosen to address in an easier way the following issues.

- **MySQL Connector** is a standardized database driver for Python platforms and development[1]. This library is used to interface the server with the MySQL database in the following easy manner:

```
import mysql.connector
conn = mysql.connector.connect(user="user", password="passw",
    database="db", host="host")
cursor = conn.cursor()
res = self.cursor.execute("SELECT * FROM table")
```

- **MGRS coordinate conversion for Python** is one of the rare libraries to deal with the Military Grid Reference System according to the UTM grid. This library is a wrapper for Python, actually written in C, compatible with the commonest and widest used CPython implementation (for Python3). This is an example, from the official documentation[2] about how it works in a nutshell:

```
>>> import mgrs

>>> latitude = 42.0
>>> longitude = -93.0

>>> m = mgrs.MGRS()
>>> c = m.toMGRS(latitude, longitude)
>>> c
'15TWG0000049776'

```

[1] https://dev.mysql.com/downloads/connector/python/
[2] https://pypi.python.org/pypi/mgrs

```
>>> d = m.toLatLon(c)
>>> d
(41.999997975127997, -93.000000000000014)

>>> y = '321942.29N'
>>> yd = m.dmstodd(y)
32.328414

>>> d, m, s = m.ddtodms(32.328414)
>>> d, m, s
(32.0, 19.0, 42.290400)
```

- **Aiocoap**[3] is a great Python implementation of Constrained Application Protocol for our communication purposes, the fundamental part of this project to respect low battery consumption constraints. Aiocoap uses **asyncio**[4] as the module that *provides infrastructure for writing single-threaded concurrent code using coroutines, multiplexing I/O access over sockets and other resources, running network clients and servers, and other related primitives*.
- Other common and standard libraries concern random, time, json and strings operations.

The server consists of four modules, each of which fulfills specific duties, let us see more in detail.

Module - Query.py

This module provides an easy interface to database for other modules. It will be needed just to instantiate an object of **Query** class to rapidly get access to all the possible queries, from the basic CRUD operations to the most complex rule filtering calls.

```
### WHICHEVER MODULE ###
from Query import *

queryObj = Query()
subStakes = list(queryObj.getSubscribedStakeholders(data['user']))
queryObj.close()
```

[3] https://github.com/chrysn/aiocoap
[4] https://docs.python.org/3/library/asyncio.html

When a **Query** object is instanced, a connection with the database is established as follow:

```
### CLASS QUERY ###
class Query:
    conn = None
    cursor = None

    def __init__(self):
      # Read connection informations from a file
      db, user, passw, host = tuple(map(lambda x:x.strip("\n").split(
    ":")[1], open("auth.txt", "r").readlines()))
      self.conn = mysql.connector.connect(user=user, password=passw,
    database=db, host=host)
```

Now we can interface with the database through this object, finally, it is important to call the *close()* method, which terminates the connection with the database, closing the cursor and committing possible updates.

```
### CLASS QUERY ###
def close(self):
  try:
    self.cursor.close()
    self.conn.commit()
    self.conn.close()
  except mysql.connector.Error as err:
    print("DB ERROR: {}".format(err))
```

This module contains many queries mostly for selecting, inserting, deleting rules, stakeholders, measurements and other complex specific queries functional for other modules. This is the most in-constant-evolution module of the project.

Module - utils.py

This module contains all the functions useful for other modules. Most of these functions regard tricky calculations for spatial and time constraints. MGRS conversion, spatial-belonging check, zone and timer calculations to answer the clients are implemented here.
For instance, the following is an interesting function, illuminating in understanding how the *MGRS mask*, debated above, is implemented to check sensing client belonging.

```
### MODULE UTILS ###
def belongsto(zone, coord, granularity):
    # Division in "32T PQ 12345 12345"
    gridsq_zone, bigsq_zone, x_zone, y_zone = zone[:3], zone[3:5],
    zone[5:10], zone[10:15]
    gridsq_coord, bigsq_coord, x_coord, y_coord = coord[:3], coord
    [3:5], coord[5:10], coord[10:15]
    return gridsq_zone == gridsq_coord and \
           bigsq_zone == bigsq_coord and \
           x_zone[0:granularity] == x_coord[0:granularity] and \
           y_zone[0:granularity] == y_coord[0:granularity]
```

Module - routines.py

This module, currently, only contains a routine function to remove all the old rules. This means that all the rules with an expire time minor than the current timestamp will be deleted from the database. In the same way, all the rules with a number of related measurements greater than the requested amount in the *countExpire* field, will be deleted.
It has been planned, as future works, a series of different routines implementing a machine learning system. This system will be in charge of optimize the entire system performance, operating mainly on the rules, adapting in this way the sensing client behavior in an intelligent and efficient way.

Module - server.py

This is the main module of the server, it is the point of access for sensing clients and the brain of the whole system, coordinating resources, database access, receiving and sending data.
As we have seen, first of all, it have been created the resource tree, identifying the exposed services:

```
### MODULE SERVER ###
def main():
  [...]
    # Resource tree creation
    root = resource.Site()
    root.add_resource(('sensing_send',), SensingSend())
```

```
7     root.add_resource(('get_subscriptions',), GetSubscriptions())
8     root.add_resource(('update_subscriptions',), UpdateSubscription()
      )
9     root.add_resource(('calc_throughput',), CalcThroughput())
10    asyncio.async(aiocoap.Context.create_server_context(root))
11    asyncio.get_event_loop().run_forever()
```

For each resource, it has been developed a class providing the service requested. All the calls are made through POST CoAP.

- **SensingSend** accept a JSON from the client containing a sensor measurement with the related location, time, user and sensor identifiers fields. After the rule extraction mechanism described previously, in this class it will be crafted a response with zone and timeout parameters for the next client update.
- **GetSubscriptions** answers the client with the list of stakeholders previously subscribed, given the user identifier. In future developments, this class will also add to the response which sensors for each stakeholder the user accepted the data access.
- **UpdateSubscription** accept a JSON containing the list of stakeholders the user subscribed. Once obtained this information, the database will be consequently updated.
- **CalcThroughput** only contains a mock method to accept a certain amount of nonsense data and answer an empty response. The semantic of this call is expressed on the next *sensing_send* call for a cellular sensing measurement, which also will contain the value of the throughput estimated on the client-side, knowing the size of the sent packet (header plus body) and the time within which the process is completed.

5.2 Sensing Client Android

As mentioned in the previous chapters, my mobile application implements a mixed form of participatory sensing because of user's possibility to choose the stakeholders (and in future the single measurements too) to subscribe, and opportunistic sensing because

of the background running nature of the applications itself. A user, once subscribed the preferred stakeholders, can forget the application running in background, while it senses and communicates with the CCU seamlessly.
Currently, the sensing client is available on Android platforms, compiled with SDK version 23, which also is the minimum target version. As we can in see in the *build.gradle* file it have been used many libraries.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.google.android.gms:play-services:9.0.2'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'de.uzl.itm:ncoap-core:1.8.3-SNAPSHOT'
    compile 'org.slf4j:slf4j-android:1.7.21'
    compile 'com.android.support:multidex:1.0.0'
}
```

Just mentioning two libraries among them, **Google Play Services**, has been fundamental to obtain easy access to location services like device position and Geofencing, and **SpitFirefox**[5], based on CoAP Java implementation **nCoAP**[6] that allows us, instead, to implement the CoAP client functionalities for Android applications.

Android development highly relies on asynchronous calls and tasks. Many methods are invoked through an Intent call where a specific event is triggered or a preparatory task is completed. This is true especially in this case, where most of the work is done, for instance, when a CoAP request is answered, a sensing measurement is ready and it is given to us from the system, when a Geofencing event is triggered because the user is entered/exited/dwelling into/from/within an area. Navigate into the code, also because of this intrinsic aspect could be tricky, let us more in detail how the application is structured.

5.2.1 The View

As a case of study, the application is composed of two simple screens.

[5]https://github.com/okleine/spitfirefox
[6]https://github.com/okleine/nCoAP

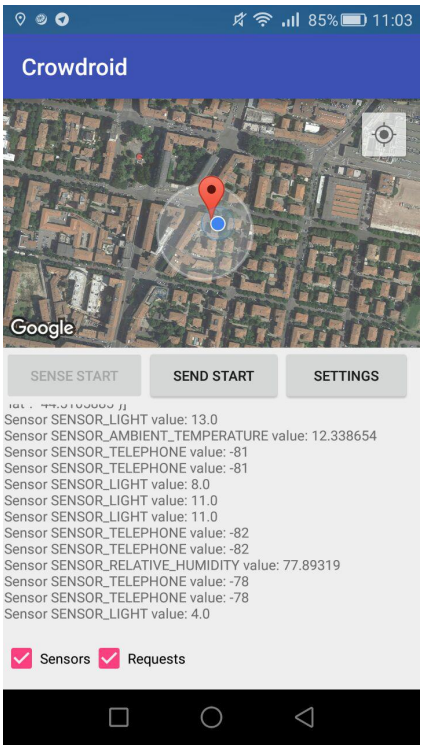

Figure 5.1: MainActivity

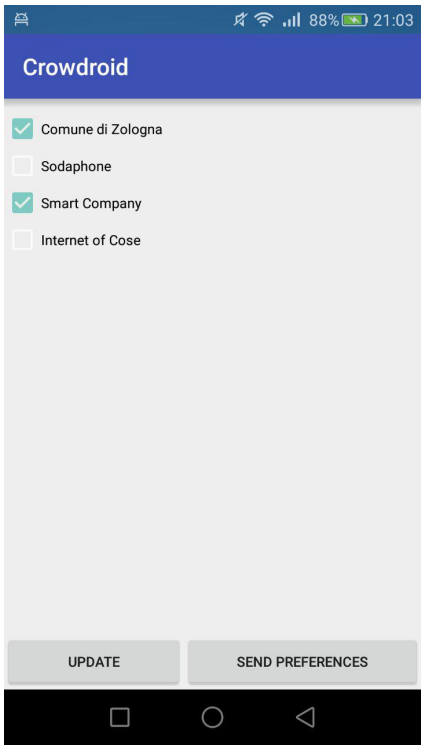

Figure 5.2: StakeholdersActivity

- **MainActivity**, the first one, in Figure 5.1, is the main activity that starts with the application, it mostly has a debugging purpose in our prototypal scenario. It includes a map, showing the user position and the geofences active in the current moment. The three buttons under the map allow us to start the sensing routines and asynchronous tasks, to start the CoAP sending of the measurements, and finally to switch the screen with the settings one, which allows to (un)subscribe the stakeholders. Moreover, there is a log screen, that shows useful information about sensing tasks and connection with the CCU. The user can selective choose to log both, one of, or none of this information.

- **StakeholdersActivity**, in Figure 5.2 is currently the Settings activity and allow the user, as mention earlier, to choose which stakeholder subscribe. Two debug-purpose buttons are needed to retrieve the stakeholders subscribed and to update the new choices on the server.

5.2.2 Sensors Measurements

As we have seen so far, the sensing client has to gather measurements from real sensors, included the microphone, and from antennas for WiFi and Telephone technologies. Most of the information that we need, especially on Android platforms, request a special treatment. Once the designated button is clicked, a Service and few Intent Services to deal with sensors and position are started.

```
1  /*** MAINACTIVITY ***/
2
3  // Start service for PhoneListener
4  Intent phoneListIntent = new Intent(getApplicationContext(),
       NeverSleepService.class);
5  startService(phoneListIntent);
6
7  // Start intent service for update position
8  Intent posintent = new Intent(getApplicationContext(),
       PositionIntentService.class);
9  startService(posintent);
10
11 // Start intent service for update sensors
```

```
12 Intent sensorintent = new Intent(getApplicationContext(),
       SensorsIntentService.class);
13 sensorintent.setAction(Constants.INTENT_START_SENSORS);
14 startService(sensorintent);
15
16 // Start intent service for update amplitude sensing
17 Intent amplintent = new Intent(getApplicationContext(),
       SensorsIntentService.class);
18 amplintent.setAction(Constants.INTENT_START_AUDIOAMPLITUDE_SENSE);
19 .startService(amplintent);
```

NeverSleepService is a very simple Service, that always runs in the background (it is also the only one and with very limited functions) to keep a landing point for *TelephonyManager* services. In this case, we only need it for a *PhoneStateListener* in order to get asynchronously the received signal strength variation from telephony cell the user is connected to. **CustomPhoneStateListener** extends *PhoneStateListener* and retrieves all the useful information, such as the company provider, the technology adopted and, obviously, the signal strength. Finally, it stores them in a SharedPreferences file. **PositionIntentService** instead is started to bind the **GoogleAPI** and receive the asynchronous position update, stored automatically in the SharedPreferences to make them read when useful, for instance before a measurement sending. We can specify a level of accuracy and a time range for updates, balancing performances and energy consumption as we want.

SensorsIntentService deals with all the environmental sensors that the device provides us, with a subscribe and asynchronous update similar to the positioning mechanism. Again, we have also to monitor the ambient noise level, in practice, we have to measure the amplitude of the sound incoming to the microphone. To do that, we need to start an AsyncTask, in the class **GetAmplitudeTask**, which records a few seconds from the microphone and calculates the amplitude average. In this case, finally, the value is sent to the SensorsIntentService which saves it in the SharedPreferences, ready to be read.

As regards the WiFi data retrieval, we do not need asynchronous calls or tasks, so all the methods relative to these information and other utils are gathered in the **RadioUtils** class. Here we can accede to WiFi RSSI, BSSID, SSID, user MAC address to calculate a unique identifier and so on.

5.2.3 CoAP and Communication

Let us see now, something more on the communication side. The classes that mostly accomplish the communication task are **SendIntentReceive** and the **SendRequestTask**, extending AsyncTask and using the SpitFirefox library callbacks.
When the user starts for the first time the application, or when a timer or a geofence callback is triggered, SendIntentReceive is awakened trough an Intent. This IntentService also deals with the other kind of requests, relative to subscriptions and throughput estimation. For each kind of Action (an Android parameter for Intent) is implemented a specific method to manage it.
If the task is a sensing measurement sending, all the useful data are taken from the SharedPreferences (if retrieval is asynchronous) or calling the right method. Now, for each request Action, is triggered an asynchronous call, which is a CoAP POST request, implemented in the SendRequestTask. Once a request is successfully done, with the same mechanism, SendRequestTask notifies with an intent the SendIntentReceive.

5.2.4 Timer and Geofencing

At this point, SendIntentReceive has the information obtained from the CCU. If the call regards subscriptions or throughput estimation, it is just needed to store the new values in the SharedPreferences and eventually update the View.
Otherwise, if the call was a sensing measurement update, all the information in the JSON response are extracted to set the timeout for time sampling and the geofence trigger on the **GeofenceIntentService**, a specific class for the landing of geofencing intents provoked.

```
1 /*** SENDINTENTSERVICE ***/
2
3 private void handleActionReceivedData(String response) {
4   // Extract JSON information
5   [...]
6     Intent gi = new Intent(getApplicationContext(),
    GeofenceIntentService.class);
7     gi.putExtra(Constants.EXTRA_GEOFENCE_SENSORTYPE, sensor);
8     gi.putExtra(Constants.EXTRA_GEOFENCE_LATITUDE, latitude);
9     gi.putExtra(Constants.EXTRA_GEOFENCE_LONGITUDE, longitude);
```

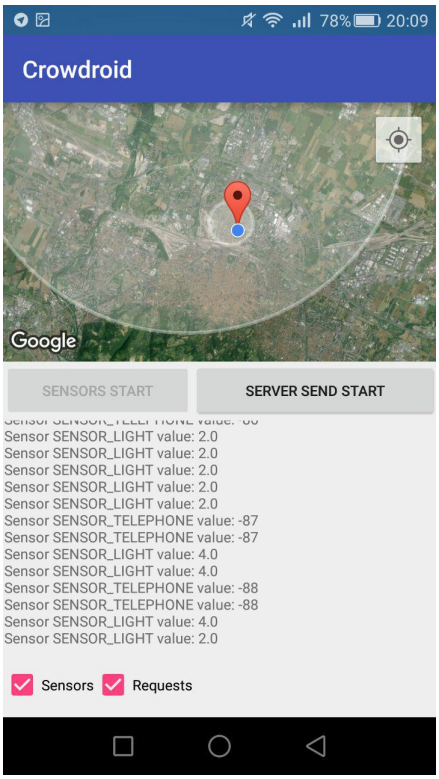

Figure 5.3: Multiple geofences active for the user position.

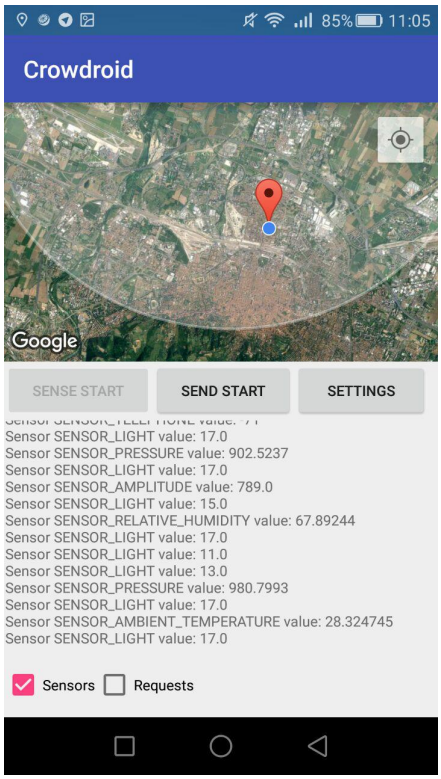

Figure 5.4: The same situation with only one geofence active.

```
10      gi.putExtra(Constants.EXTRA_GEOFENCE_RADIUS, String.valueOf(
        radius));
11      gi.putExtra(Constants.EXTRA_GEOFENCE_EXPIRE_MILLISEC, String.
        valueOf(timeout*1000));
12      startService(gi);
13
14      // Set timeout based on server response
15      alarmMgr = (AlarmManager)getApplicationContext().getSystemService
        (Context.ALARM_SERVICE);
16      Intent intentAlarm = new Intent(getApplicationContext(),
        SendIntentService.class);
17      intentAlarm.setAction(Constants.ACTION_SENDDATA+sensor);
18      intentAlarm.putExtra(Constants.EXTRA_TYPE_OF_SENSOR_TO_SEND,
        sensor);
19      alarmIntent = PendingIntent.getService(getApplicationContext(),
        0, intentAlarm, 0);
20
21      int seconds = timeout;
22      alarmMgr.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
23              SystemClock.elapsedRealtime() +
24                      seconds * 1000, alarmIntent);
25  }
```

Chapter 6

Test and evaluation

The following test case was led, in order to evaluate the system in a real urban environment. It has been used city of Bologna as perfect example of heterogeneous and complex municipality in which the deployment of Smart City concept, as debated in this document, is applicable and fertile.

First, there have been identified two MGRS 10km sided squares including all the city, one identified by **32TPQ 8 2** (the southern one) and the other by **32TPQ 8 3** (the northern).

There have been set up two rules in the database, for the light sensor. One rule requested *timeSampleGran = 300* and *spaceSampleGran = 0* (100km side square sampling) over the all the considered space. Then a mock stakeholder has been added, which requested through another rule, the same light sensor monitoring, this time over the only northern half of the city and with a finer space grain, *spaceSampleGran = 3* (100m side square sampling). In figure Figure 6.1 we can see the variation of the update timer over a time span of about 10 hours for a test user's smartphone. The x-axis shows the updates in chronological order, labeled with an increasing index. On the y-axis, instead, we have the number of seconds elapsed from the previous update.

As in the diagram is clear, while the user is within the southern square, the updates are triggered at the regular interval of about 300 seconds according to the relative rule. Connection failures (high spikes) and sporadic border crossing (low spikes) are probably responsible for the few irregularities present on this trend.

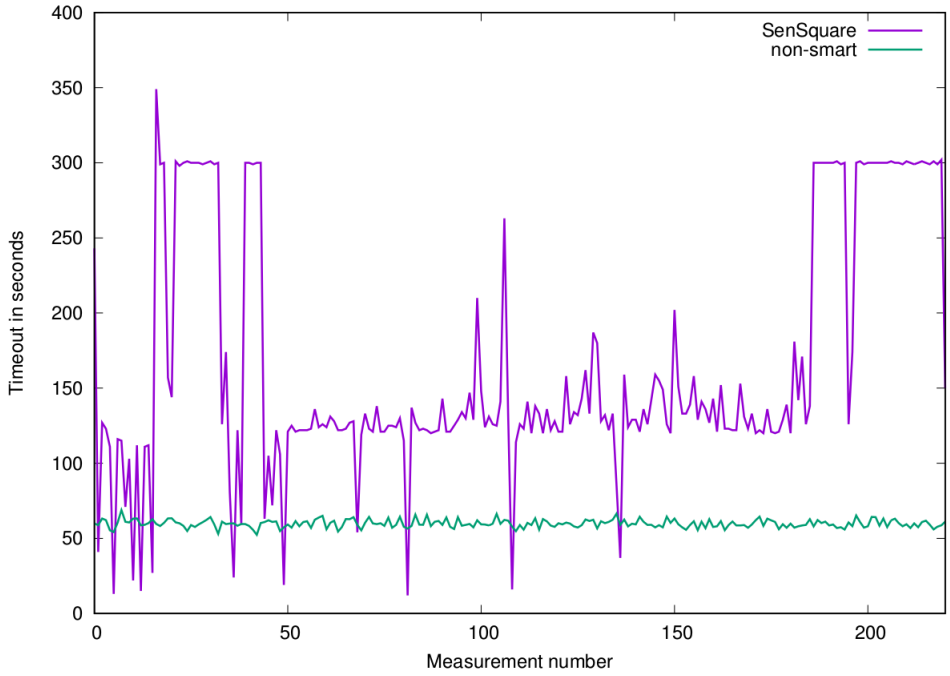

Figure 6.1: The update time variability over 230 measurements compared with the ideal non-smart approach.

While the user, instead, is within the northern square of the city (interval $[40, 180]$), the spatial constraint of the second rule seems to be more effective, making the device send updates more frequently, as the user is moving. During this period, high spikes are probably due to slow or standing still user, while low spikes denote a user walking fast and/or crossing a small lateral portion of a 100 meter MGRS sided square.

Assuming a non-smart application, which sends measurement updates at a constant rate of 60 seconds, we can see a huge difference with the SenSquare approach. The non-smart application guarantees a regular and constant updating rate but it is straightforward that are performed much more updates than required, with the consequent energy and resource waste. In fact, with a 60 seconds timer for the same number of measurements this approach cover only 3.5 hours of monitoring.

Another test is showed in the histogram in Figure 6.2. This test is performed on two

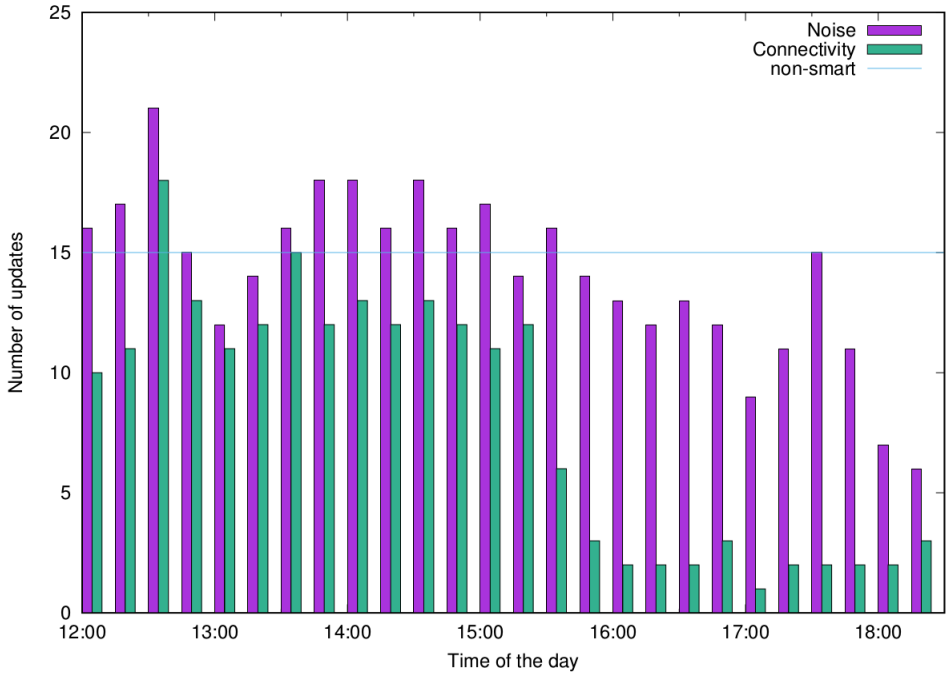

Figure 6.2: Number of updates required by SenSquare for two different sensors, grouped in 15 minutes time slots.

different measurements in the south square of Bologna. A rule identifies a measurement

for the noise sensor with *timeSampleGran = 80*, and another rule requires the monitoring of cellular connectivity with *timeSampleGran = 300*. Two stakeholders' rules require the same sensor sampling, for both the measurements with *spaceSampleGran = 4* (spatial constraint of 10 meters). As we can see, initially it is clear that the update rate is due to the spatial constraint, because of the user movements, such as continuous walking, while when the user stands still, in the second part of the time span, the update rate is more time-driven. A non-smart approach would require the same update rate regardless of the spatial conditions, asking for useless measurements, with battery, computation and network consumption.

More and different kinds of test are planned as future work. Among them, it could be interesting a stress test with a relevant amount of sensing clients, in order to test the server implementation efficiency, what kind of tuning is requested on the *countExpire* field in rules for each sensor and to study the urban mobility in a real usage test. An analytical or simulation-based study could be useful to show, in a less expensive way, the system behavior in various conditions. Furthermore, specific tests with different sensing client devices, such as Arduino platforms and other smart devices could be appropriate to give a complete evaluation of the system.

Chapter 7

Future developments

As future works, many improvements are planned to expand this project in order to make it a complete MCS system, which can be able to exit from the academic environment, being competitive with other solutions.

7.1 More and different devices

First of all, it is needed to develop the same client functionalities (or most of them) on other operating systems, such as iOS and Windows, as well as design and develop sensing clients for platforms like Arduino/Genuino and ESP8266-01. This is fundamental to widen the entire scenario, getting great advantages from different devices with various and numerous sensors, especially for embedded platforms. An implementation for these devices would also allow covering homogeneously specific zones in an urban environment with a lightweight "infrastructure" supporting the mobility sensing client population.

7.2 Performances and consumption

Staying on the client side, as we mentioned in the previous chapters, besides specific parameters tuning tests, it is planned a complete performance enhancement, through analytical and simulation studies. The goal is to achieve a very low cost on and users' devices, mainly in terms of computation and power consumption. It is possible to act on

the client implementation, on the CCU, and on the whole system as well. For instance, sensing clients could enrich their CoAP requests with battery/CPU/memory constraint, in order to give to the CCU the possibility to balance the sensing tasks, not only on the basis of stakeholders' rules, but adding them a new reasoning on this constraints (for example treating them as new rules as well).
It is also, envisaged a future work that includes machine learning techniques in order to establish the needed number of measurements in a specified zone. There will be calculations using the data variance and the number of users, therefore the rules will change dynamically, outlining a self-adapting elastic ecosystem.

7.3 Scalability

Moreover, another future work is related to the *zone-specific CCU*. This simply means that the scalability will be ensured also by a division of the scenario in different zones, each of which is automatically served by a different CCU. A user in travel, who changes multiple city environments in few hours or minutes, will not know that his/her device is connecting with different CCU because it will be seamless. Many different implementations are possible in this sense, from proxying to an application-driven solution with a kind of "almanac" like in GPS technology [46].

7.4 Privacy, security, and reliability

Many future developments are related to ensure more control for the user above privacy issues. These enhancements engage the whole system and involve strongly architecture design and development effort.
The easiest feature, shortly implementable, is the possibility for the user to choose with a finer grain which sensor measurements concede to which stakeholders, as we mentioned in "Mobile Application" section. In this case, the work on the client is minimal, but on the server, it is requested a more complex filtering on the described rules choosing the procedure. Further improvements are related to the following issues. Data anonymization is one of them and it is one of the most studied fields in the era of big data. Currently, users are identified by an ID on the CCU's database, this is useful to link

them with their subscriber, in order to implement the rule filtering and fostering mechanisms as well. A great enhancement could be the implementation of data aggregation beside data encryption.
Furthermore, data reliability is crucial. As the SenSquare system will be available for a large audience, it will be necessary a strict control over the data quality. It is clear, in fact, that people live their life independently from their crowdsensing application running on their devices. This obvious, but not banal fact, is to consider implementing mechanisms able to identify situations in which the data coming on the CCU are junk. For instance, a person could keep their phone in a pocket invalidating the light sensor measurements, but not completely the noise sensor one.

7.5 Stakeholders and Marketing

A web and a mobile platform are also necessary to make the stakeholders able to insert their rules, developing a personal fostering and marketing campaign, and clearly, to get and see in handy ways the data they have collected. Finally, it could be also interesting to design a potential business model, as seen in [25, 26, 45], in order to reach an as wide as possible participation and make the project attractive for companies and institutions.

Chapter 8

Conclusions

In this document it has been introduced the crowdsensing paradigm, as a specific form of crowdsourcing for wide environment sensing monitoring. We covered the technological background, upon which SenSquare is designed, starting from the environmental monitoring, passing through MCS solution paradigms, the CoAP lightweight protocol and technologies like Geofencing besides MGRS reference system as geographical support.
We have also made a panoramic over the state of the art in MCS research field, in different application scenarios. Finally, it has been presented the SenSquare system, our proposal for a new MCS point of view, joining different aspects existing in other related projects.
The architecture, in addition to CCU and a mobile sensing client implementation, were illustrated, in order to better understand principles and reasonings behind every choice. Strong advantages come from this MCS system. First, the usage of MGRS allows an easiest and efficient way to make calculus on the server, helping the scalability and keeping simple the location management. Second, the rule filtering system allows a low energy consumption on the client side, reducing also the network usage, avoiding multiple, useless or redundant measurements sending, just reasoning on spatial and temporal semantics. Furthermore, the usage of a lightweight and portable application protocol allows to drastically reduce the resource waste in terms of memory, computation, network, and power, enabling the same client design on different platforms, including the embedded ones.

In SenSquare, there is no effort for users and stakeholders to collaborate for collecting huge amount of sensing measurements. Stakeholders only have to ensure an effective rewarding campaign and inserting strategic rules on the CCU. On the other hand, users only have to accept one or more stakeholders sensors monitoring proposal, doing nothing else that carrying around their "sensing" devices during the day.
The implemented working system, described in this thesis, just represents the first step in a long road to a complete and competitive work. SenSquare, hopefully, with the illustrated enhancements, can easily become a milestone in the Mobile Crowdsensing scenario.

Bibliography

[1] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Ear-phone: an end-to-end participatory urban noise mapping system," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2010, pp. 105–116.

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

[3] A. D. Chiappari, "Wireless sensor network in environmental monitoring," 2015. [Online]. Available: http://alaindev.it/docs/wsnenv.pdf

[4] F. J. Villanueva, D. Villa, M. J. Santofimia, J. Barba, and J. C. López, "Crowdsensing smart city parking monitoring," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 751–756.

[5] M. Talasila, R. Curtmola, and C. Borcea, "Improving location reliability in crowd sensed data with minimal efforts," in *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP*. IEEE, 2013, pp. 1–8.

[6] (2016) Image. [Online]. Available: https://www.linkedin.com/topic/smart-city-concept

[7] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, p. 62, 2012.

[8] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Tech. Rep., 2014.

[9] (2016) Image. [Online]. Available: http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/

[10] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power coap for contiki," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*. IEEE, 2011, pp. 855–860.

[11] (2016) Image. [Online]. Available: https://developer.android.com/training/location/geofencing.html

[12] A. Developers. (2016, aug) Creating and monitoring geofences. [Online]. Available: https://developer.android.com/training/location/geofencing.html

[13] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 1994, pp. 85–90.

[14] S. Rodriguez Garzon and B. Deva, "Geofencing 2.0: taking location-based notifications to the next level," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 921–932.

[15] R. B. Langley, "The utm grid system," *GPS world*, vol. 9, no. 2, pp. 46–50, 1998.

[16] B. Guo, Z. Yu, X. Zhou, and D. Zhang, "From participatory sensing to mobile crowd sensing," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*. IEEE, 2014, pp. 593–598.

[17] H. Ma, D. Zhao, and P. Yuan, "Opportunities in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 29–35, 2014.

[18] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, and A. T. Campbell, "Urban sensing systems: opportunistic or participatory?" in *Proceedings of the 9th workshop on Mobile computing systems and applications*. ACM, 2008, pp. 11–16.

[19] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan, "Lowering the barriers to large-scale mobile crowdsensing," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*. ACM, 2013, p. 9.

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

[21] InSTEDD. (2006) Geochat. [Online]. Available: http://instedd.org/technologies/geochat/

[22] (Accessed: 2013) The ushahidi platform. [Online]. Available: http://ushahidi.com/products/ushahidi-platform

[23] Garmin, edge 305. [Online]. Available: www.garmin.com/products/edge305/

[24] Mit news. [Online]. Available: http://web.mit.edu/newsoffice/2009/blood-pressure-tt0408.html

[25] J.-S. Lee and B. Hoh, "Dynamic pricing incentive for participatory sensing," *Pervasive and Mobile Computing*, vol. 6, no. 6, pp. 693–708, 2010.

[26] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola, "Fostering participaction in smart cities: a geo-social crowdsensing platform," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 112–119, 2013.

[27] M. Witt, C. Scheiner, and S. Robra-Bissantz, "Gamification of online idea competitions: Insights from an explorative case," *Informatik schafft Communities*, vol. 192, 2011.

[28] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: defining gamification," in *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*. ACM, 2011, pp. 9–15.

[29] S. Dale, "Gamification making work fun, or making fun of work?" *Business Information Review*, vol. 31, no. 2, pp. 82–90, 2014.

[30] F. Montori, L. Bedogni, and L. Bononi, "On the Integration of Heterogeneous Data Sources for the Collaborative Internet of Things," in *2nd International Forum on Research and Technologies for Society and Industry (RTSI)*, 2016.

[31] M. Talasila, R. Curtmola, and C. Borcea, "Mobile crowd sensing," 2015.

[32] C. Leonardi, A. Cappellotto, M. Caraviello, B. Lepri, and F. Antonelli, "Secondnose: an air quality mobile crowdsensing system," in *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*. ACM, 2014, pp. 1051–1054.

[33] M. Zappatore, A. Longo, M. A. Bochicchio, D. Zappatore, A. A. Morrone, and G. De Mitri, "Mobile crowd sensing-based noise monitoring as a way to improve learning quality on acoustics," in *Interactive Mobile Communication Technologies and Learning (IMCL), 2015 International Conference on*. IEEE, 2015, pp. 96–100.

[34] D. Oletic and V. Bilas, "Design of sensor node for air quality crowdsensing," in *Sensors Applications Symposium (SAS), 2015 IEEE*. IEEE, 2015, pp. 1–5.

[35] R. Salpietro, L. Bedogni, M. Di Felice, and L. Bononi, "Park here! a smart parking system based on smartphones' embedded sensors and short range communication technologies," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 18–23.

[36] R. Liao, C. Roman, P. Ball, S. Ou, and L. Chen, "Crowdsourcing on-street parking space detection," *arXiv preprint arXiv:1603.00441*, 2016.

[37] F. J. Villanueva, D. Villa, M. J. Santofimia, J. Barba, and J. C. López, "Crowdsensing smart city parking monitoring," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 751–756.

[38] L. Bedogni, M. Di Felice, and L. Bononi, "By train or by car? detecting the user's motion type through smartphone sensors data," in *Wireless Days (WD), 2012 IFIP*. IEEE, 2012, pp. 1–6.

[39] B. L. Bedogni L., Di Felice M., "Context-aware android applications through transportation mode detection techniques," *Wireless Communications and Mobile Computing*, 2016.

[40] D. B. A. Di Chiappari, G. Cinelli, "Geospot, a location-based and activity-based mobile advertisement prototypal system," jul 2016. [Online]. Available: http://alaindev.it/docs/geospot.pdf

[41] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, and W. Trappe, "Parknet: drive-by sensing of road-side parking statistics," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 123–136.

[42] (2016) Mobile millennium project. [Online]. Available: http://traffic.berkeley.edu/

[43] R. Pryss, M. Reichert, J. Herrmann, B. Langguth, and W. Schlee, "Mobile crowd sensing in clinical and psychological trials–a case study," in *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*. IEEE, 2015, pp. 23–24.

[44] L. G. Jaimes, I. J. Vergara-Laurens, and A. Raij, "A survey of incentive techniques for mobile crowd sensing," *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 370–380, 2015.

[45] S. Reddy, D. Estrin, M. Hansen, and M. Srivastava, "Examining micro-payments for participatory sensing data collections," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, 2010, pp. 33–36.

[46] I. A. Getting, "Perspective/navigation-the global positioning system," *IEEE spectrum*, vol. 30, no. 12, pp. 36–38, 1993.

[47] R. Liao, C. Roman, P. Ball, S. Ou, and L. Chen, "Crowdsourcing on-street parking space detection," *arXiv preprint arXiv:1603.00441*, 2016.

[48] A. Sheth, S. Seshan, and D. Wetherall, "Geo-fencing: Confining wi-fi coverage to physical boundaries," in *International Conference on Pervasive Computing*. Springer, 2009, pp. 274–290.

[49] D. R. Sanqunetti, "Implementing geo-fencing on mobile devices," Apr. 13 2004, uS Patent 6,721,652.

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

[51] W. Sherchan, P. P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke, and A. Sinha, "Using on-the-move mining for mobile crowdsensing," in *2012 IEEE 13th International Conference on Mobile Data Management*. IEEE, 2012, pp. 115–124.

[52] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha, "Piggyback crowdsensing (pcs): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2013, p. 7.

[53] G. Chen, D. Kotz *et al.*, "A survey of context-aware mobile computing research," Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Tech. Rep., 2000.

[54] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web services for the internet of things through coap and exi," in *2011 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2011, pp. 1–6.

[55] B. L. Di Chiappari, Montori and B. L, "Sensquare: a mobile crowdsensing architecture for smart cities," in *submitted to 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT) (WF-IoT 2016)*, Reston, USA, dec 2016.