	Research Activity in the CNR Area in Pisa, Italy **Involved CNR Institutes: ISTI, IIT, IFC**
	ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO" Istituto di Informatica e Telematica IFC CNR
Project Title	**SMART AREA OF CNR IN PISA, Italy** **Coordinator: Erina Ferro, CNR-ISTI**
Date	**November 30, 2015**
Authors	**Erina Ferro, Raffaele Conte, Raffaele Bruno, Claudio Vairo, Davide Moroni, Paolo Barsocchi, Andrea Marchetti, Francesco Potortì, Franca Delmastro, Carlo Meghini, Abraham Gebrehiwot, Mario Marinai, Antonino Crivello**
Web Site	***www.smart-applications.area.pi.cnr.it***

CNR Research Area, Via Moruzzi 1, 56124 Pisa, Italy
erina.ferro@isti.cnr.it
+39-050-315.3070
+39-347-2643126

TABLE OF CONTENTS

LIST OF PICTURES

1 THE CNR SMART AREA AS A PART OF THE SMART CITY PROJECT (Rif. Erina Ferro, ISTI)

Most of the large cities of today have to face with immense problems in terms of development, traffic, social services, security, pollution, climate, health, and many more. Many European cities are currently developing strategies towards becoming "smart cities"; such strategies are based on an assessment of the future needs of the cities, an innovative usage of the ICT technologies, the development of new applications, and a multidisciplinary approach to the solution of the problems. An important aspect of this innovation process is the high level of active involvement of the citizens in the process itself, being them the final users of the complex ecosystem in which the smart cities are evolving.
Since the end of 1990s, many cities have initiated smart initiatives. In the Digital Agenda of the European Commission, cities are considered as innovation drivers in areas such as environment, inclusion, health and business.
But what is a "smart city"? According to the definition of [Caragliu et al, 2009][1], a city is defined smart when "*investment in human and social capital and traditional (transportation) and modern (ICT-based) infrastructure fuel sustainable economic growth and a high quality of life, with a wise management of natural resources, trough participatory government*".
The deployment of digital solutions relies on a series of information, communication, and programming technologies, most of them becoming available in the last two decades. The evolution of these technologies (which has been extremely rapid), together with the explosion of Internet for global information dissemination has driven the creation of smart cities. Most of these changes fall in four categories: 1) the diffusion of the world wide web ant the related technologies, 2) the increase in communication bandwidth, 3) the embedded systems and wireless networks, and 4) the diffusion of the smart phones.
In smart cities, transportation, energy, bridges, hospitals, offices, schools, roads, parking places, airport, etc. offer a wide domain of experimentation for the Internet of Contents & Knowledge, the Internet of Services, and the Internet of Things.

In the smart cities context, in the second half of 2013 CNR launched a project entitled *Renewable Energy and ICT for Energy Sustainability* (Energia da Fonti Rinnovabili e ICT per la Sostenibilità Energetica). The project was based on the widespread use of renewable energy sources (and related storage technologies and energy management) and the extensive use of ICT technologies for an enhanced management of the energy flows, thus making the energy services more efficient by adapting them to the demand (and, therefore, encouraging the energy saving and the energy rational use), with the informed involvement of citizens. One group of researchers in the CNR area of Pisa was involved in a part of this wide project, and on

[1] Caragliu A., Del Bono C., Nijkamp P., "Smart Cities in Europe". Series Research Memoranda 0048. Free University of Amsterdam, Faculty of Economics, Business administration and Econometrics, 2009.

October 31 2014 they presented a preliminary experimentation in the area. However, as already said, a smart Energy diffusion and management is only one aspect, among many others, of a smart city, and the CNR area in Pisa (the largest CNR area in Italy) is, in fact, a small city where smart technologies and applications can be experimented before to be reversed in a smart city.
The area experiences every day many problems related to parking, transportation, surveillance, maintenance, energy, localization, and so on…..thus, why not to develop smart applications to transform the area in a smart area. These applications are useful to the people working in or visiting the area but they also are of interest for the public administration of the Pisa city.
Thus, we used the CNR area as a laboratory where to redesign some infrastructures and where to test some applications related to parking, transportation, navigation in the area, energy management in buildings and access registration to the area.
The added value of this large effort, sustained by a consistent group of researchers and technicians, is twofold: on one side, all the developed applications are CNR-marked, that means that we can "put the hands" in the applications as we like (no commercial solutions have been chosen); on the other side, each application is autonomous but it shares its data with the data of all the other applications, putting them in a common database on a cloud. The data collected are open to any smart application of the area that wants to access them; each application can access the data of the other applications thus inferring cross-information that allows the enhancement of the application itself and the development of new applications.

As a last (but not least!) point, I would like to underline the human success derived from the smart area initiative, i.e. that people of different institutions collaborated with enthusiasm for a common purpose, which is to make the area more liveable, to the benefit of all people.

In the following Sections, the current level of smartness of the CNR area in Pisa is described.

The new infrastructures we installed, the new applications developed and those we have in mind to develop in a close future are the initial steps towards the creation of a Smart Area Living Lab, where companies can be inspired to test and develop innovative solutions as products or services in the ICT technological field: urban planning, mobility, energy, sensorization, and so on. Most of the technologies we are developing or we will develop could be transferred to the local administrations for making Pisa a smart city. This is our final goal!

2 THE SMART BADGE (Ref. Raffaele Conte, IFC)

***In brief**: Smart Badge implements on smartphone a badge for recording staff attendance and access to the workplace, through a QRCode generated on the basis of a one-time password, read by a Raspberry Pi and a Camera. Application and reader utilize a SAML infrastructure for initialization and synchronization of user data. The application can also be used for remote stamping in case of off-the-job, by sending the server "now" and "location" of the user.*

The Smart Badge (SB) application has been developed by a group of researchers of the Institute of Clinical Physiology (IFC) to manage the entry/exit of the staff through an Electronic Badge (Smart Badge), as an alternative to the current system based on magnetic badge. The SB system, by means of a smartphone-based application, allows the users to checking/checkout and to exploit all the recent technological advances to provide a useful, easy-to-use, and user-friendly solution to all potential end-users.

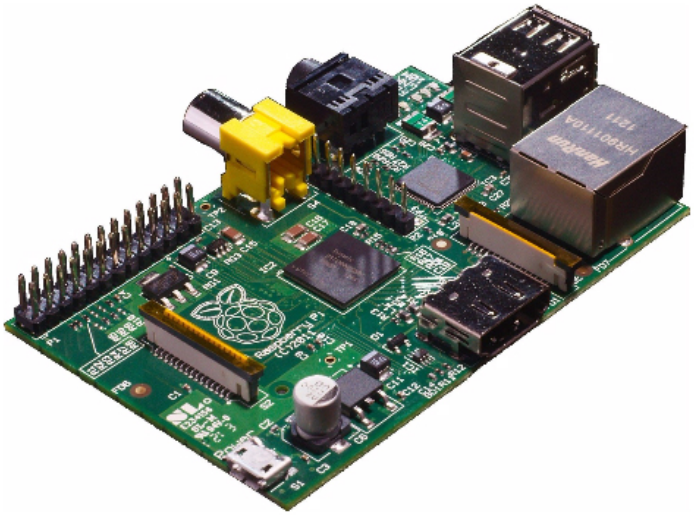


Figure 1: The Raspberry PI board evaluation system

Furthermore, the tool developed aims at representing a low-cost solution, whose reduced design and development cost, robustness and full integration with the existing infrastructures of CNR are among the main features. At the moment, the Smart Badge application integrates with the CNR-IFC Identity Manager and Authentication system (called "*idea*"). Integration with the other identification & administration systems used by the various CNR Institutes is in progress; the work is quite content as many institutes federate on the same identification and administration system.

The Smart Badge system is composed by three main elements: 1) the Smart Badge Service Provider: a server, which coordinates the communications among the components and with the different *check-in/check-out management systems* inside the different institutes, 2) one or more *badge readers*, realized through a micro-PC (Raspberry PI, Figure 1) equipped with a camera, and 3) a smartphone mobile app, currently based on Android 2.3 or higher OS (future work will transport the application on other OS). All components and the architecture of the whole systems are represented in Figure **2**.

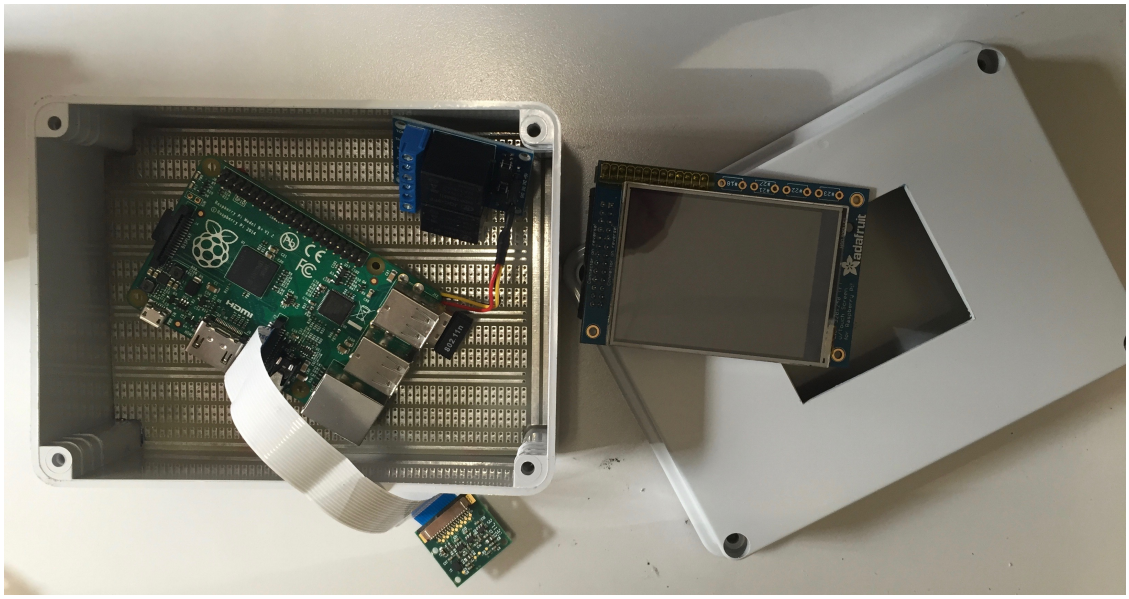


Figure 2: The Raspberry PI with the display, the camera, the relays board and a "hand-modified" enclosure

2.1 SYSTEM ARCHITECTURE

The Smart Badge System was designed as a standalone module that can be integrated with the existent systems for managing the access/exit records. Currently, it is integrated with the Identity Manager, named "*idea*", of the IFC Institute and the relevant check-in server. The interfacing with *idea* is intended to verify the login credentials and the access permissions of the users to the system (user recognition and access permission to a specific service) by means of the SAML (Security Assertion Markup Language) protocol, as well as to provide the list of the users enabled to use the SB System.
The only hardware component of the system is the Badge Reader, being the user supposed to have his own Smartphone. All servers involved in the system (the LDAP Server, the SAML Identity Provider, the Check-in Module Server and the Smart Badge Service Provider) might be implemented in an existent virtualized infrastructure and do not require large computational resources. The components of the whole systems are described below and shown in Figure **3**.

2.1.1 The Identity Manager

The Identity Manager contains the database, a Ldap Server (implemented by OpenLDAP), where the user information is stored and realizes the backend of idea, the whole Identity Manager. It is employed as a data repository for authentication and personal information of the users. It allows a good flexibility for the organization's information. The authentication protocol and the access to the user information can happen in three ways: 1) the LDAP protocol (deprecated), 2) the Radius protocol (for example with 802.1x protocol used for WiFi), or 3) the SAML protocol (for all the Web applications). For the aim of the Smart Badge system, the SAML Identity Provider is used. It manages the SAML protocol, implemented with Shibboleth, to share authentication and authorization data among different services.

Moreover, it manages the dispatch of users' data to the different Service Providers for their proper management.

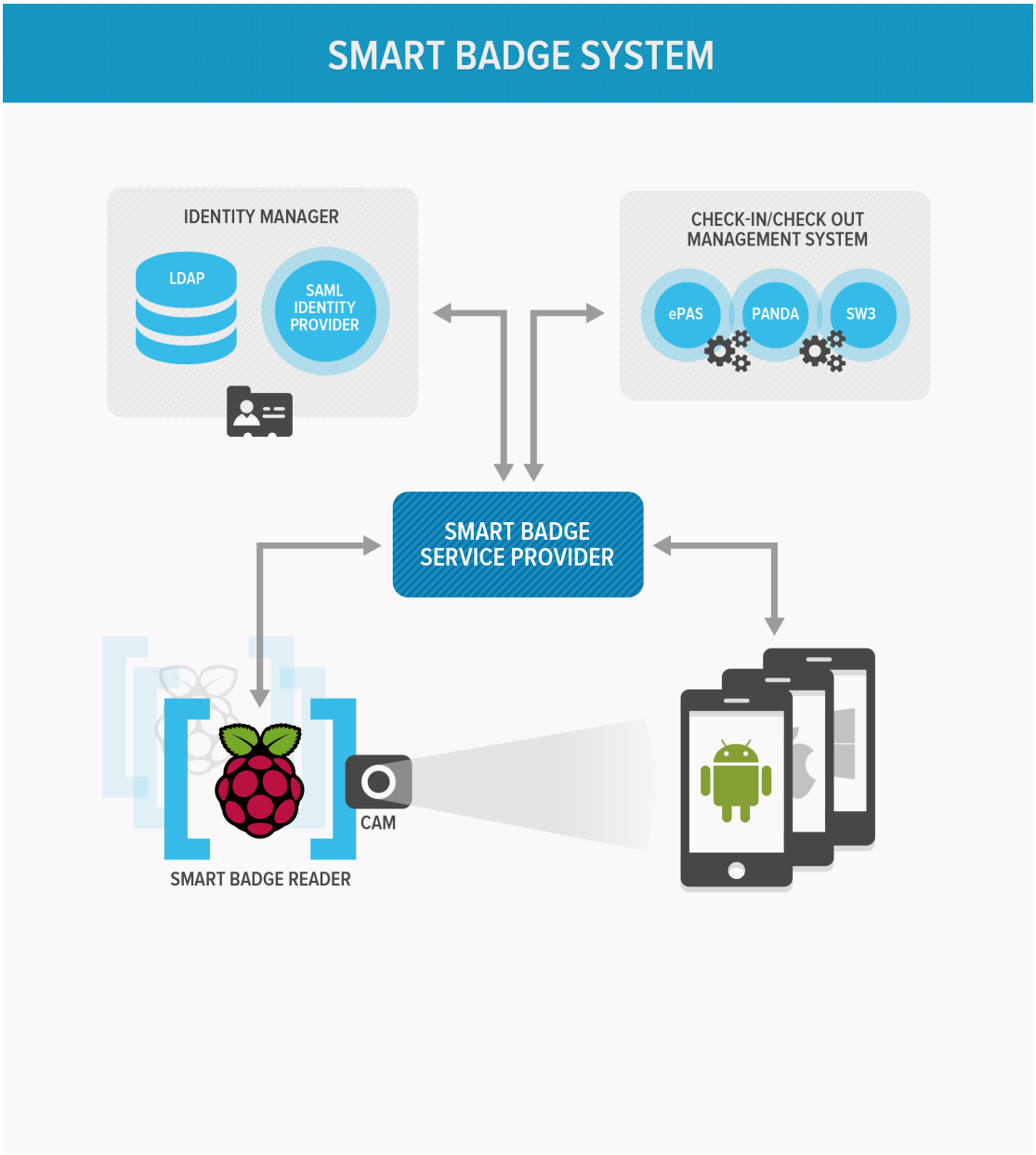


Figure 3: Smart Badge System architecture

2.1.2 The Check-in/check-out Management System

It is an external system (regarding the Smart Badge System) that archives the operation of check-in and checkout for the different users and interacts with the Smart Badge Service Provider. Each Institute autonomously manages this type of data, so the system can be different for each of them.

2.1.3 The Smart Badge Service Provider

It implements the Service Provider entity for the SAML protocol, which interacts with the SAML Identity Provider in the Identity Management System, manages and archives the users' data and the readers' data, manages and temporary archives the check-in/check-out users' data.

Moreover, it exposes the RESTful Web-Services and allows the abstraction of the Badge Reader and Mobile App on the smartphone. It gets all the requests from the Mobile App and Badge Reader and modifies them accordingly in order to get them interfaced with *idea* and the API of the Check-in/check-out Management System. It allows a univocal access to the Mobile App, participating to the authentication phase of the user towards the Identity Manager and managing the identification and authentication of the Badge Readers. The Web-Services can be classified into:

- Web-Services towards Mobile App:
  - user/getInfo for users information;
  - check-in/insert to send a remote check-in;
  - check-in/last to retrieve the last 10 check-ins for a given user.

- Web-services towards Badge Reader:
  - login/readerId to retrieve the reader's information and its own seat;
  - user/list/readerId to retrieve the list of users enabled for check-in;
  - check-in/insert to send the check-ins locally stored to the Service Provider.

From a logical point of view, the software architecture of the Smart Badge Service Provider can be divided in three main parts: the Core, the Database Abstraction Layer, and the Database. The Core interacts, on one side, with the Mobile App or with the Badge Reader through the http protocol, and on the other side with the Database. It holds three common modules concerning the most frequent activities required:

- *Check in*: here the input/output event recording occurs together with the query about the historical check-ins;
- *User*: it provides information to the Mobile App about the current user, and to the Smart Badge about a subgroup of all active users;
- *Reader*: it provides setup information to the Reader.

The Database Abstraction Layer is used to increase the flexibility of the project and to free the Core from a given database. Specifically, the Doctrine MongoDB ODM was employed, with a good integration with PHP 5.5.3+ and with Silex, through a Service Provider able to expose its functionalities within the framework itself.
The Database is a NoSQL model, the MongoDb, a non-relational, document-oriented database, chosen for the freedom to add and delete features within each document, for the efficient search feature (thanks to the indexing in each field of the document), for the data good reliability, and for the horizontal scalability (thanks to the sharding).

2.1.4 The Smart Badge mobile App

With the aim of making the app available on the greatest number of smartphone, the technology identified as more suitable for the interaction with the Badge Reader was the QR Code generated within the mobile app and read by the SB reader. Other technology (as NFC) could be added over the time. Technologies that do not require the proximity between the smartphone and the reader, hence do not require a voluntary action of the user were discarded for obvious reasons. With these minimum requirements, the app can run from the 2.3 (Gingerbread) version of Android OS. **The Smart badge application is downloadable from the Smart Area Web Site, clicking on *Smart Badge*.**

Figure 4: QRCode generated within the Mobile App

Figure 5: The remote (with GPS) check-in/check-out

The SB app offers three main functions: 1) an *on-site check-in/check-out*, for the entrance or exit to the workplace (Figure **4**), 2) a *remote registration*, for occasional out-of-site work (Figure **5**), and 3) the *list of the last records* (Figure **6**). In order to implement the remote

check-in/check-out, the only hardware subsystem required to the smartphone is the GPS system, available on almost all the smartphones used today.

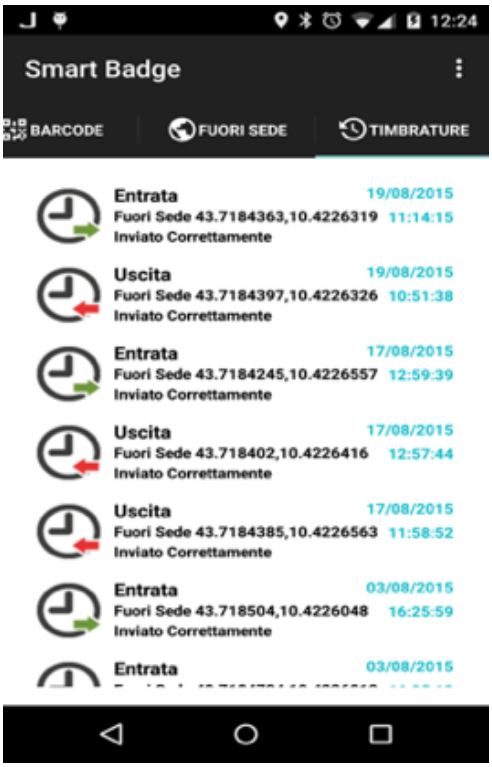


Figure 6: The last check-in/checkout for a specific user

2.1.5 The Smart Badge Reader

By the hardware point of view, it is composed of the Raspberry PI micro-PC, an LCD display, to interact with the end-users, and a PI Camera to detect and recognize a QR Code, used for the check-in/check-out. It also holds a LAN network board attached to the mainboard to allow the Internet connection, also possible through a WiFi board that can be linked through USB interface. The Badge Reader allows recording of the entrance or exit requests of the users. Realized under Raspbian OS, it holds a local database to store the user' list and the relative records for entrances and exits.

2.2 USE CASES

There are five possible use cases forecasted for the use of the system: 1) initialization and configuration of a reader; 2) download and initialization of the SB app; 3) on-site check-in/check-out; 4) remote check-in/check-out; 5) access to the last registrations.

The main use cases with the components of the system involved in each use case are represented in Figure **7**.

2.2.1 Initialization of the Smart Badge App

Once downloaded, the app on the smartphone must be initialized to be ready for the use. For the initialization phase, the only operation required to the user is to authenticate himself.

Thanks to the SAML authentication mechanism, after the user authentication, the Service Provider automatically receives all the needed user's attributes (username, name, surname, location of the institute's section with the relative GPS coordinates and others) by the identity Provider. Then, the same attributes are available for the App together with the last check-ins of the users. Using the SAML protocol, the initialization only requires the user's authentication.

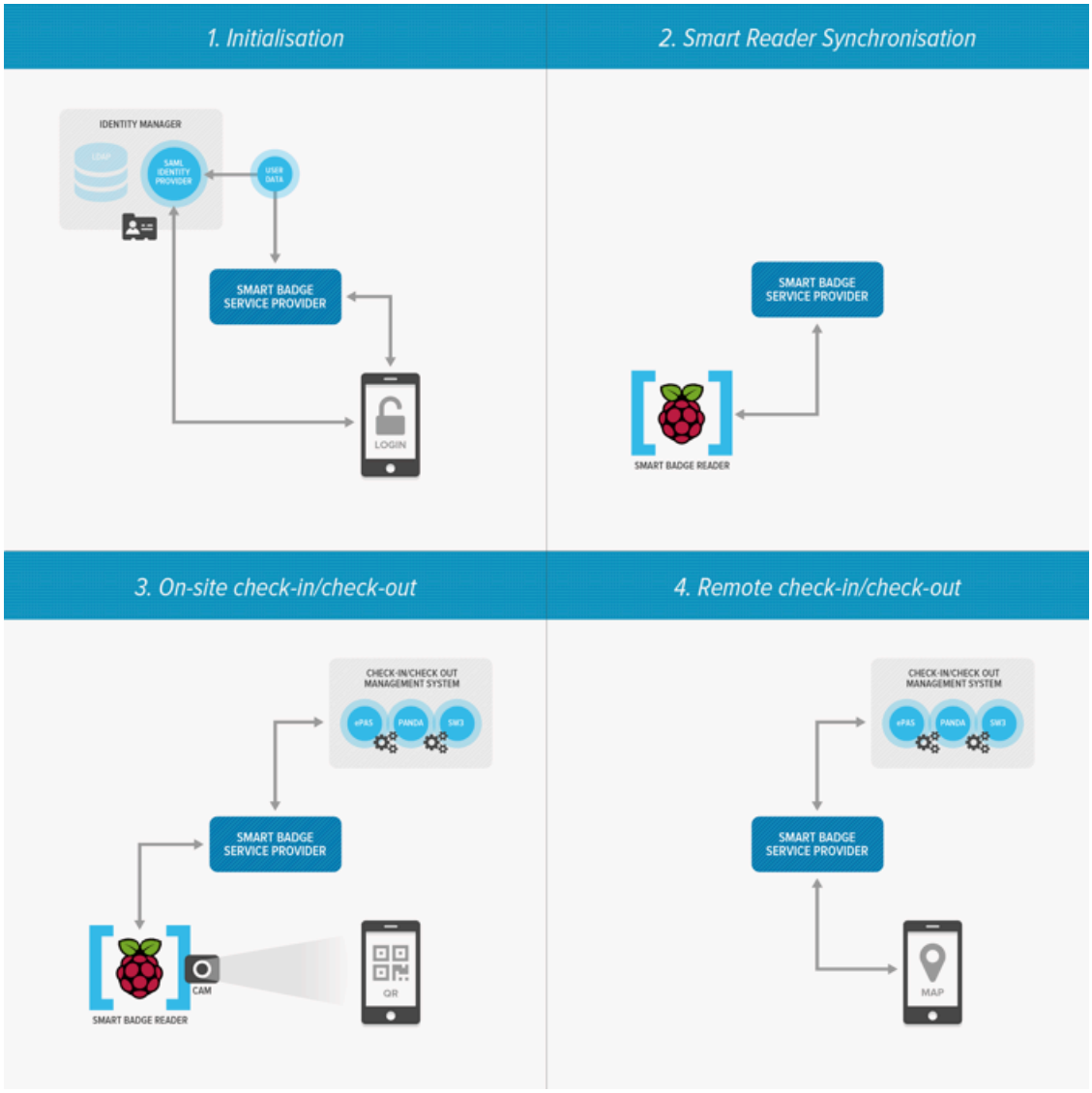


Figure 7: The Smart Badge system components involved in the use cases

2.2.2 Initialization and synchronization of the Smart Badge Reader

With the initialization operation, the reader gets the enabled users' list and the private tokens employed in the control of the One Time Password stored within the QRCode. After the initialisation, the reader is ready for the check-in/check-out through the reading of the QRCode visualised on the smartphone.

After reading the QR Code, the reader shows a message to the users as acknowledged and sends, if possible, the data to the Service Provider for the storage. If the Service Provider is not reachable, data are locally stored and sent after a while.

2.2.3 On-site registration

The app allows checking-in locally by generating a QR Code read by firm readers positioned both at the entry and at the exit of the building. The QR Code contains some attributes needed for identifying the user. For security reasons, the QR Code is dynamically created and contains a code generated as a One-Time Password (OTP). The code must match with the corresponding code generated on the reader. Moreover, the code contains the type of record (entry or exit), deduced by some elements (last record, position, and so on) and proposed to the user. The user himself may change the type of the record.

To speed up the registration operation, a proximity notification is also inserted into the app, so that the mobile can check the presence of a structure equipped by a Badge Reader and shows a banner, in the home screen, that allows the user to quickly access the check-in function. This is the reason why the mobile app downloads by the Service Provider the location of the Institute's sections with the relative GPS coordinates.

2.2.4 Remote registration

In some circumstances, the personnel could work out of their workplace (for collaborations with other institutions, universities and so on); in these situations, the user can autonomously declare its work time (self-declaration). The SB app allows avoiding the self-declaration by means of the remote registration function. In this function, together with the registration time, the geographically position of the user, obtained by the GPS, is also sent (Figure **5**).
The GPS check-in is inserted into a local database inside the app and sent to the server once the Internet connection is enabled.

2.2.5 Last check-in/check-out registration

Another function of the app is the list of the last records of the specified user. Figure **6** shows a screenshot of the app with the records and relative types (entrance or exit).

2.3 THE REST API

Within the REST API approach, the type of the exposed resources and their representation are particularly important. Concerning the Smart Badge, the resources needed to interact with are the users, the check-ins, and the readers. Only the check-in resource is required to allow record generation and storage within the Database (the only POST operation), while the other two resources are classified as "read-only" (GET). The response format for data exchange is JavaScript Object Notification (JSON), easy to use through a browser and with languages like PHP and JAVA, employed for the realization of the application.

User resource

- GET /api/v1/m/getInfo: it is necessary to the mobile client if, after the login action, it needs to update the users info, the offices list or any other information about the authentication session;
- GET /api/v1/r/users/{readerId}/list: returns the list of the users associated to a particular section of the Institute and, consequently, a specific reader. It is used by the reader when it needs an update of the enabled users;
- GET /api/v1/r/{readerId}/list/{updatedFrom}: used by the reader for incremental synchronisation;

- GET /api/v1/r/{readerId}/users/{userid}: used by the reader in the case of unknown user (userid).

Checkin resource

- GET /api/v1/m/checkin/last: through this url, the mobile client could retrieve the list of the last "n" check-in with information about office, time, entry/exit status, and coordinates;
- POST /api/v1/m/checkin/insert and POST /api/v1/r/checkin/insert: to check-in entry/exit through the mobile app (remote check-in/check-out) and reader (optical recognition), respectively.

Reader resource

- GET /api/v1/r/login/{readerId}: login url. It gets the setup information, including address and coordinates of the required office, used for check-in operations.

3 THE SMART PARKING (Ref. Raffaele Bruno, IIT)

In brief: *The Smart Parking application allows visitors and CNR employees to check in real-time the availability of single vacancies in the parking stalls of the area, to get detailed statistics on the parking area utilization, to predict the future availability of free parking slots or to book a place in case of need. This is made possible thanks to a system of parking management that integrates data, collected by two different types of innovative monitoring cameras and by occupation road parking sensors, with web and mobile applications to access online services.*

It is estimated that about 30% of the cars circling a city at any given time are doing so as drivers look for parking. Looking for a parking space (called "cruising") creates additional traffic congestion, increased carbon emissions, additional delays. In central areas of large cities, cruising may account for more than 10% of the local circulation as drivers can spend 20 minutes looking for a parking spot [1]. Clearly a lack of parking is not the sole reason for traffic congestion; it is, however, a major contributing factor.

In order to save time for cruising, the driver should be notified when and where empty spots are available, rather than have to search for them by himself/herself. *Automatic monitoring of parking lots is one of the most relevant problems for Intelligent Transport Systems in urban scenarios*. Urban traffic monitoring is made possible by the deployment of pervasive technologies, such as Wireless Sensor Networks (WSN)[3][4]. Current monitoring technologies, such as weigh-in-motion sensors, inductive loop detectors or magnetic sensors, are very reliable. However, they are embedded in the pavement and the cost of individual sensors becomes a limiting factor for large parking areas with many spaces to monitor. Thanks to computer vision techniques, fully automatic video and image analysis can be used to solve this task. Compared to sensor-based parking space detection, which requires the installation and maintenance of networks of sensors, vision-based systems are cheaper and less intrusive. In addition, a suite of online tools is also needed to use the information gathered by these monitoring technologies and to enable drivers to easily locate vacant parking spaces in real time.

In the rest of this Section, we describe the implementation of an intelligent car parking management system, called "Smart Parking" application, which is designed for the purpose of supporting intelligent parking management in the CNR Area in Pisa. Specifically, we overview the system architecture, the used technologies, the provided functionalities (e.g., data fusion, storage and analysis), web tools, and how this application is integrated in the open CNR cloud platform.

3.1 SYSTEM ARCHITECTURE

The main features of the Smart Parking application can be summarised as follows:

- *Low-cost solution*: it does not require that each parking lot is equipped with one sensor node, but it mainly leverages on low-cost image processing technologies to precisely monitor the occupation status of a large parking area;
- *Multi-technology platform*: it leverages on a common REST-based data access model that allows to easily integrating different information sources. In addition, the intelligent fusion of data coming from multiple monitoring technologies that have overlapping monitoring areas allows to detect more precisely the occupation status of the parking lots;
- *Interoperability*: it is deployed on an open cloud platform and it fully embraces the Open Linked Data paradigm to facilitate the sharing of structured parking-related data;
- *Flexibility*: it facilitates the re-use of the monitoring infrastructure to support multiple services, such as surveillance and navigation;
- *Utility*: By using the management system, both the users and the managers will be able to get real-time information about the parking field, as well as aggregate statistics and historical trends. This facilitates the development of more efficient parking policies and it can promote the adoption of more sustainable mobility behaviours.

The functional architecture of our intelligent car parking management system is illustrated in Figure **8**.

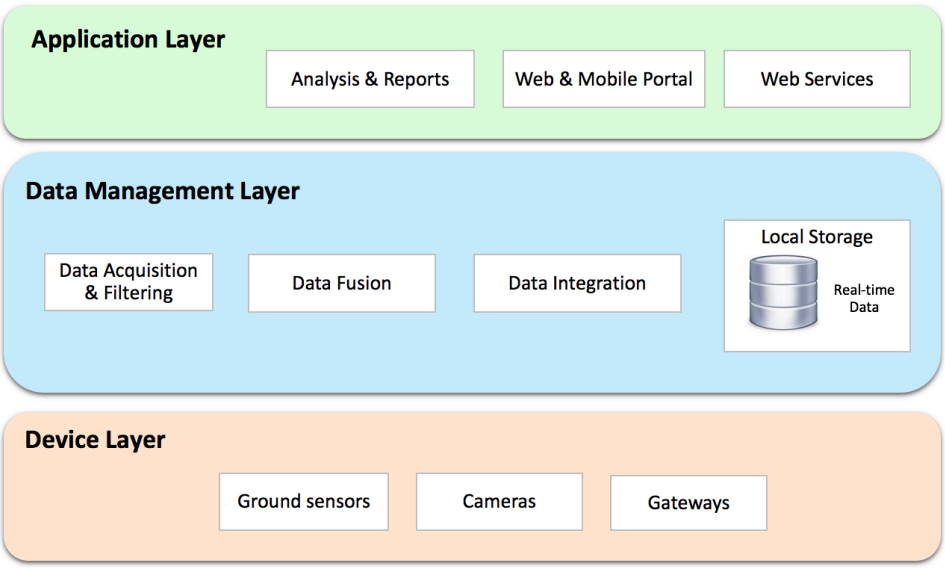


Figure 8: Functional architecture of our intelligent car parking management system.

As in any control system based on sensor networks, the bottom layer of the system architecture consists of the *Device Layer*. In our system, the devices are sensors and data acquisition boards that gather information on the occupied car-parking spaces. As explained in mode detailed in the following sections, in our system multiple parking monitoring technologies have ben integrated, ranging from ground sensors with magnetic/IR sensing modalities for highly accurate and reliable vehicle detection, to cameras that can detect the presence of many cars at once. The collected information can be transmitted to a gateway via wireless/wired communication among the sensor nodes. Beside data transferring, gateways allow the sensor

nodes to accept control and configuration commands from the upper layer application. However, it is important to point that our functional architecture also supports the direct communication of the sensor devices with the data management layer without the use of intermediate gateways and middleware technologies. As better explained in the following this is achieved by using a REST-based transfer protocol that leverages on HTTP functionalities and universal resource identifiers. The *Data Management Layer* is the core of our functional architecture. This layer is responsible for data filtering, pre-processing, aggregation, transfer and local storage. Data may need to be pre-processed to handle missing samples, remove redundancies and aggregate data from different sources into a unified schema before being committed to the local data storage and the remote archive. Note that archiving refers to the offline long-term storage of data that is not immediately needed for the system's ongoing operations. This capability relies on the persistent data storage that is provided by the cloud platform.

Finally, the upper layer of our intelligent car parking management system is the *Application Layer*. This layer consists of the software components and tools that operate on top of the database system. The application layer focuses on the business logic of the parking administration and the processing/visualization of the collected information stored in the database system. For instance, the analytic/forecasting module generates various reports to help users and administrators to understand the running status of the parking field. The application sub-system includes the graphical user interfaces (GUIs) that allow users and managers to interact with the system and visualize both real-time and historical data. Finally, our application sub-system implements the web services that are needed to communicate with the devices and other applications.

In the following sections we briefly describe the characteristics and the operations of the various components in the three layers of our parking management system.

3.2 HARDWARE COMPONENTS OF THE DEVICE LAYER

In the device layer of our system, we have integrated three different sensing technologies. The first two sensing technologies are based on sensors that are equipped with cameras and video processing techniques. These sensors have been developed by two research groups of CNR-ISTI. The third technology is based on ground sensors, which support both magnetic and infrared proximity sensors. This technology is developed by the Kiunsys start-up[2] and it is used as a reference benchmark to evaluate the detection performance of the cameras.

3.2.1 High-Performance Smart Cameras (Ref. Claudio Vairo, ISTI)

In this Section we describe in detail the design characteristics of the high-performance smart camera prototype that we used in our intelligent parking management system. To develop our smart cameras for acquiring and analysing video information about the occupancy of parking

[2] http://www.kiunsys.com/

lots, we used a Raspberry Pi 2 model B, i.e., a card-sized single-board computer, equipped with a camera module (see Figure **9**). More precisely, the Raspberry Pi hardware includes a:

- BCM2836 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM DDR2
- 32GB micro SD card for storage

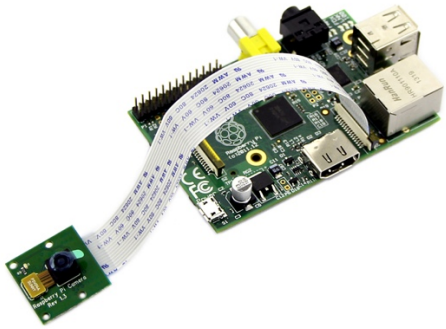


Figure 9: Raspberry Pi with camera module.

The camera module is a 5MP fixed-focus camera that supports 1080p30, 720p60 and VGA90 video modes, as well as still captures. The view angles of the camera are 53.50° horizontally and 41.41° vertically. The allowed resolutions are reported in Table **1**.

Table 1: Raspberry Pi camera module resolutions.

Size	**Aspect Ratio**	**Frame Rate**
2592x1944	4:3	1-15fps
1920x1080	16:9	1-30fps
1296x972	4:3	1-42fps
1296x730	16:9	1-49fps
640x480	4:3	42.1-60fps

We put the Raspberry Pi in outdoor boxes (Figure **10**) that have been installed on top of the roof of the building in front of the parking lots. Each box is provided with a heater/blower temperature control system, and each Raspberry Pi is connected to the network by an Ethernet cable.

We deployed nine cameras in order to cover half of the parking lot of the rear part of the CNR area. Other smart cameras installed by another group working to the project cover the other half of the parking lot.

Figure 10: Outdoor box.

3.2.1.1 Image processing

We used the OpenCV library to process the images acquired by the cameras and a deep learning approach to determine whether a parking slot is available or not. Then, we communicate the changes of the occupation status for each parking slot using a RESTful web service, which is responsible for storing the updates in the cloud platform. In the following, we explain in detail the video processing algorithm.

Deep Learning (DL)

Deep Learning is a branch of Artificial Intelligence that develops techniques that allow computers to learn complex perception tasks, such as seeing and hearing at human levels of performance. It provides near-human level accuracy in image classification, object detection, speech recognition, natural language processing, vehicle and pedestrian detection, and more.

The traditional approach to the classification problem uses ad-hoc functions to extract particular features from the image that are considered to be indicative of certain objects. For example, hard corners and straight edges might be believed to indicate the presence of manmade objects in the scene. The outputs of these feature extraction functions are then fed into a classification function, which determines if a particular object has been detected in the image. However, this approach leads to weak and false-alarm prone detectors and present the following problems:

- It is very hard to think of robust, reliable features, which map to specific object types.
- It is a massive task to find the right combination of features for every type of object to classify.
- It is very difficult to design functions that are robust to translations, rotations and scaling of objects in the image.

All these problems make very hard developing high accuracy object detectors and classifiers for a broad range of objects. On the contrary, the DL approach exploits a large number of ground-truth labelled images to discover which features and combinations of features are most discriminative for each class of objects to be recognized and builds a combined feature extraction and classification model (this phase is called training the model). The model thus obtained can be deployed and it is not only able to classify specific objects it was trained on, but also it is able to recognize previously unseen similar objects. The general DL approach is shown in Figure **11** (courtesy of the NVIDIA Deep Learning web course).

A large set of images, which compose the training set, is fed to a neural network composed of a possibly large number of hidden layers (whence the term "deep"). Each hidden layer performs a mathematical computation on the input and produces an output that is fed as input to the following layer. The final outputs of this network are the classes for which the network has been trained on.

The training phase is usually extremely computationally expensive and it may take a long time. On the other hand, once the network has been trained and the classifier has been initialized accordingly, the run time phase of prediction is quite fast and efficient.

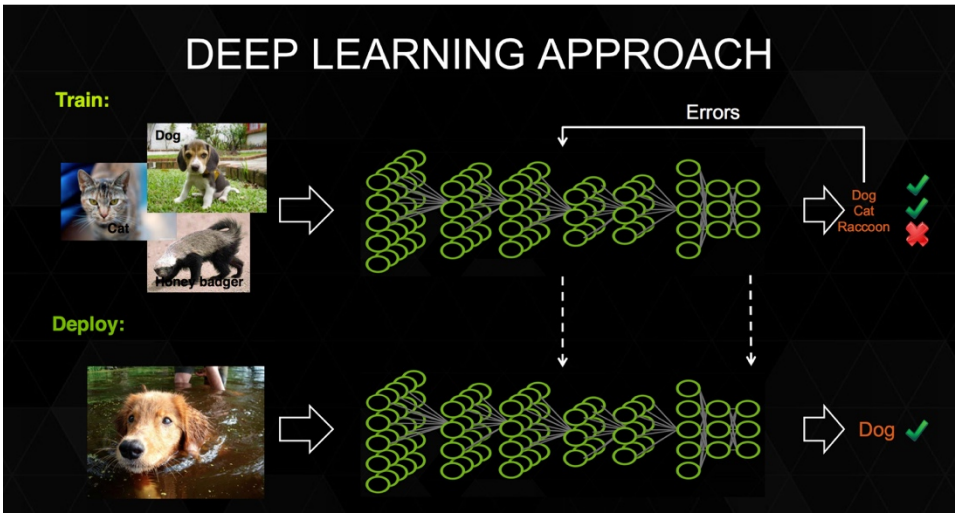


Figure 11: Deep Learning Approach.

Our solution

We collected a large number of screenshots of the parking lot in different days, from different points of view, with different occlusion patterns, and in different weather conditions. Then, we built a mask for each parking slot in order to split the original image in several smaller images (we refer to these small images as patches), one for each parking slot. Each of these patches is a square of size proportional to the distance from the camera, the nearest are bigger then the farthest.

We built a dataset of about 10K patches and this constitutes the training set of our deep learning network. The training set captures different situations of luminosity, including partial occlusion due to obstacles (lamps, trees or other cars) and partial or global shadowed cars (see Figure **12**). This allows the generated classifier to be able to distinguish almost every situation that can be found at run time. Of course, total occlusions and nightly situations cannot be addressed with this approach.

We used the Caffe framework for Deep Learning to train the neural network and initialize the classifier. Caffe is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. It is very fast due to its highly optimized C/CUDA backend, which integrates GPU acceleration. It provides command line, python and MATLAB interfaces and it is very easy to use since it allows defining the network and the training/testing phases by

writing simple human-readable text files. The neural network that we adopted is presented in Figure **13**. It is composed of 14 levels, 5 of which trainable (3 convolutions and 2 fully-connected). It is a smaller network compared to the typical Deep Learning neural networks, since we need the classifier to be able to predict only two classes: busy and available

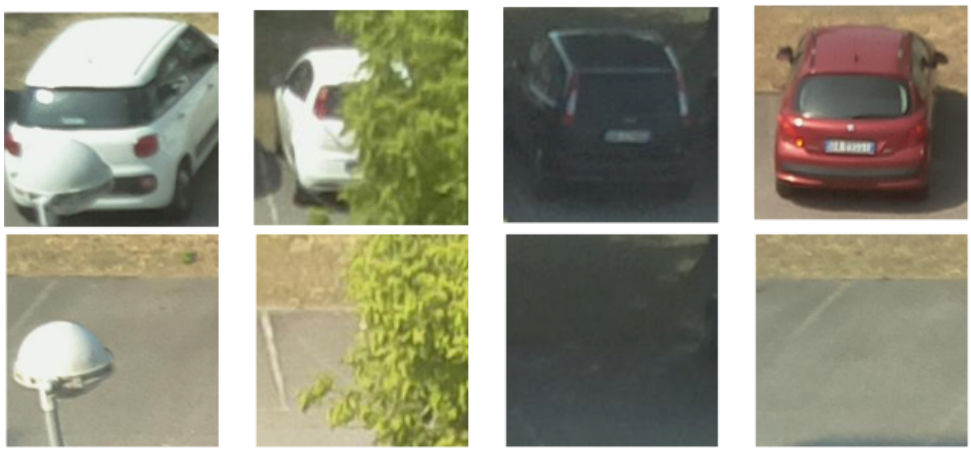


Figure 12: Training set samples

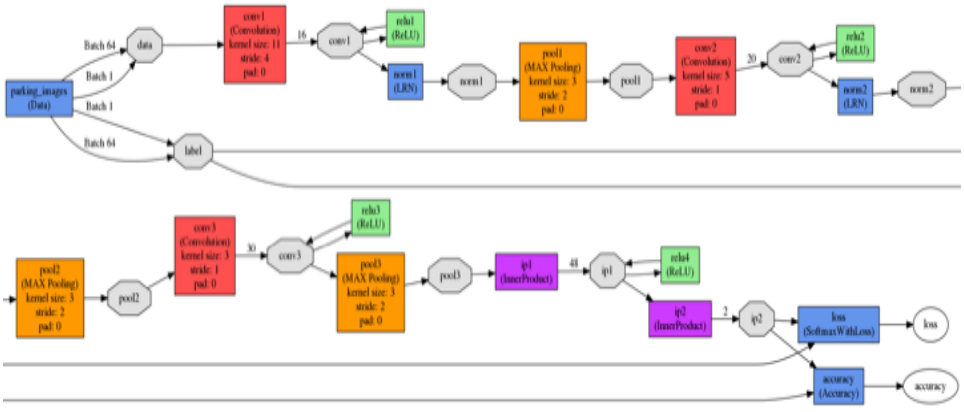


Figure 13: The neural network used in our solution

The training phase generates the weights of the neural network that we used to initialize the classifier provided by Caffe according to our application scenario. To this purpose, we implemented a JNI interface to use the C interface provided by Caffe in order to instantiate and use the neural network. We then created a library containing the classifier initialized with the trained neural network, and we integrated it in our software that has been eventually deployed on the Raspberry Pi.

The software we implemented first loads the trained neural network generated by Caffe, then it loads the masks to detect the individual parking slots and to split the image in the patches. Please note that each camera has its own set of masks because each camera sees a different portion of the parking lot with different angles of view. However, this is a one-time process to be done only when the camera is installed and we implemented a semi-automatic software tool

to generate the masks. With this tool, the user can load an image captured by the camera and by simply clicking in the centre of each parking slot, the corresponding mask will be automatically generated with the proper size.

The rest of the computation is organized in two main run-cycles, one small with duration of 15 seconds, called slot monitoring cycle, and one large with duration of one minute, composed of four slot monitoring cycles, called slot update cycle.

In the slot monitoring cycle the status of each slot is measured and compared to the previous consistent state. The slot update cycle determines if there has been an actual slot status change according to the values read in the four slot monitoring cycles. In particular, if a slot status change is detected in at least three slot monitoring cycles, then we can conclude that the slot status has been actually updated. This is done to avoid failures due to temporary manoeuvring cars, or detection errors.

1. An iteration of the slot monitoring cycle performs the following operations:
2. Capture the current frame.
3. Split the frame in the set of patches according to the masks for that camera.
4. Perform the prediction for each patch by interrogating the classifier.
5. As a result, a matrix, containing the score for both the busy and available classes for each slot, is returned.
6. Compare the occupancy level for each slot with the status of the previous consistent state for that slot. If a parking slot status change is detected, a counter (called counterUpdate) for that slot is incremented.

If the slot update cycle determines a change in the status of a parking slot (i.e. the counterUpdate has a value greater or equal to 3), an update message for the slot status is sent to the RESTful web service responsible of storing the information about the parking lot in the cloud platform. The format of this message is described in detail in Section 3.3.3.

Validation

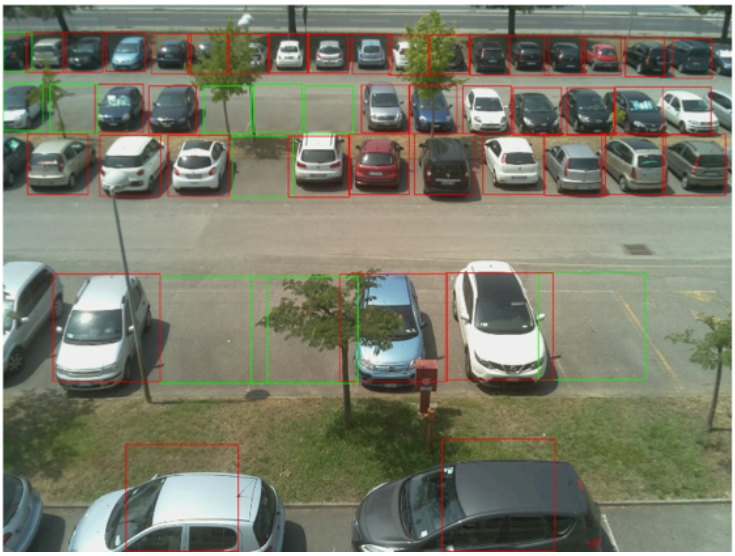


Figure 14: Parking slot status detection example from roof

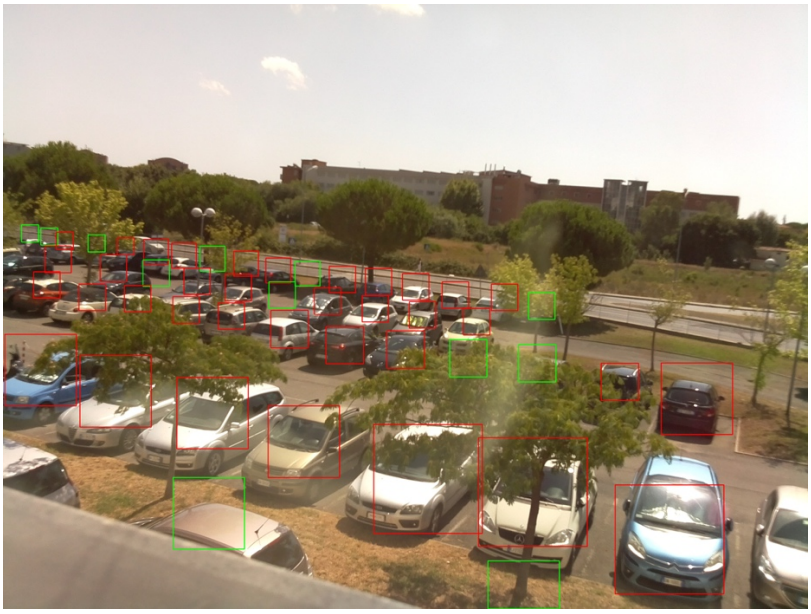


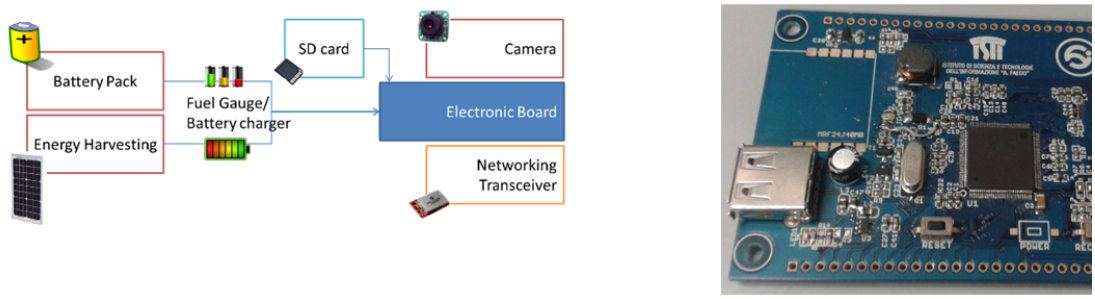
Figure 15: Parking slot status detection example from office.

We performed some preliminary tests by placing the camera both on the roof of the building (Figure **14**) and behind the window of an office (Figure **15**) with a lower angle of view and an oblique perspective. In both cases, we obtained quite good prediction results from the deep learning classifier, considering that trees, lamps, other cars and shadows occlude some cars.

3.2.2 Low-power Embedded Smart Cameras (Ref. Davide Moroni, ISTI)

In this Section, we describe in detail the design characteristics of the embedded smart camera prototype that we used in our intelligent parking management system. One of the most important design principles we adopted in the development of our prototype was to use low-cost and low-power technologies. Specifically, it is essential to use low-cost sensors and electronic components so that, once engineered, the device can be manufactured at low cost in large quantities. The single sensor node has a main board that manages both the vision tasks and the networking tasks thanks to an integrated wireless communication module (RF Transceiver). Other components of the sensor node are given by the power supply system that controls charging and permits to choose optimal energy savings policies. The power supply system includes the battery pack and a module for harvesting energy, e.g. through photo-voltaic panel (see Figure **16**.a). We designed and realized a custom vision board with an embedded Linux architecture for providing enough computational power and ease of programming. The printed-circuit board (PCB), which is shown in Figure **16**.b, was designed in order to have the maximum flexibility of use while maximizing the performance/consumption ratio. A good compromise has been achieved by using a Freescale CPU based on the ARM architecture, with support for MMU-like operating systems GNU/Linux. This architecture has the advantage of integrating a Power Management Unit (PMU), in addition to numerous peripherals interface, thus minimizing the complexity of the board. In addition, the CPU package of type TQFP128 helped us minimize the layout

complexity, since it was not necessary to use multilayer PCB technologies for routing. Thus, the board can be printed also in a small number of instances.



(a) Sensor architecture (b) PCB design

Figure 16: Our sensor prototype.

The choice has contributed to the further benefit of reducing development costs; in fact, the CPU only needs an external SDRAM, a 24MHz quartz oscillator and an inductance for the PMU. It has an average consumption, measured at the highest speed (454MHz), of less than 500mW. The board has several communication interfaces including RS232 serial port for communication with the networking board, SPI, I2C and USB. For radio communication, a transceiver compliant with IEEE 802.15.4 has been integrated in line with modern approaches to IoT. A suitable glue has been used to integrate the transceiver with the IPv6 stack, also containing the 6LoWPAN header compression and adaptation layer for IEEE 802.15.4 links. Therefore, the operating system is well capable of supporting ETSI M2M communications. For the integration of a camera sensor on the vision board, some specific requirements were defined in the design stage for providing easiness of connection and to the board itself and management through it, and capability to have at least a minimal performance in difficult visibility condition, i.e. night vision. Thus, the minimal constraints were to be compliant with USB Video Class device (UVC) and the possibility to remove IR filter or capability of Near-IR acquisition. Moreover, the selection of a low cost device was an implicit requirement considered for the whole sensor node prototype. The previously described boards and camera are housed into an IP66 shield. Another important component of the node is the power supply and energy harvesting system that controls charging and permits to choose optimal energy savings policies. The power supply system includes the lead (Pb) acid battery pack and a module for harvesting energy through photo-voltaic panel.

3.2.2.1 Image processing

In this Section, we describe the operations of our proposed Smart Camera Network (SCN), i.e., a special kind of WSN in which each node is equipped with an image-sensing device. Our SCN is used to estimate the available parking lots and it is based on a lightweight computer vision pipeline. In particular, only data in predefined Region of Interests (RoIs) are processed to detect the presence of a vehicle. Each camera sends in real-time information about the spaces it monitors. The beliefs of all sensors monitoring a certain space are fused together to provide a final interpretation of the scene (see Figure **17**).

Background Subtraction

Lightweight detection methods are used to classify a pixel as changed (in which case it is assigned to the foreground) or unchanged (in which case it is deemed to belong to the background). Such decision is obtained by modelling the background. Several approaches are feasible. The simplest one is represented by straightforward frame differencing. In this approach, the frame before the one that is being processed is taken as a background. A pixel is considered changed if the frame difference value is bigger than a threshold. Frame differencing is one of the fastest methods but has some cons in ITS applications; for instance, a pixel is considered changed two times: first when a vehicle enters and, second, when it exits from the pixel area.

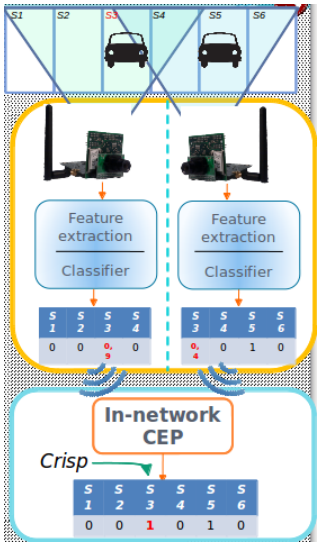


Figure 17: The beliefs of all sensors are fused together to provide a final interpretation of the scene

In addition, if a vehicle is homogeneous and it is imaged in more than one frame, it might not be detected in the frames after the first. Another approach is given by static background. In this approach, the background is taken as a fixed image without vehicles, possibly normalized to factor illumination changes. Due to weather, shadow, and light changes the background should be updated to yield meaningful results in outdoor environments. However, strategies for background update might be complex; indeed, it should be guaranteed that the scene is without vehicles when updating. To overcome these issues, algorithms featuring adaptive background are used. Indeed, this class of algorithms is the most robust for use in uncontrolled outdoor scenes. The background is constantly updated fusing the old background model and the new observed image. There are several ways of obtaining adaptation, with different levels of computational complexity. The simplest is to use an average image. In this method, the background is modelled as the average of the frames in a time window. Online computation of the average is performed. Then, a pixel is considered changed if it is different more than a threshold from the corresponding pixel in the average image. The threshold is uniform on all the pixels. Instead of modelling just the average, it is possible to include the standard deviation of pixel intensities, thus using a statistic model of the background as a single Gaussian distribution. In this case, both the average and standard deviation images are computed by an

online method on the basis of the frames already observed. In this way, instead of using a uniform threshold on the difference image, a constant threshold is used on the probability that the observed pixel is a sample drawn from the background distribution, which is modelled pixel by pixel as a Gaussian. Gaussian Mixture Models (GMM) are a generalization of the previous method. Instead of modelling each pixel in the background image as a Gaussian, a mixture of Gaussians is used. The number k of Gaussians in the mixture is a fixed parameter of the algorithm. When one of the Gaussian has a marginal contribution to the overall probability density function, it is disregarded and a new Gaussian is instantiated. GMM are known to be able to model changing background even in cases where there are phenomena such as trembling shadows and tree foliage [1]. Indeed, in those cases pixels clearly exhibit a multi-modal distribution. However, GMM are computationally more intensive than a single Gaussian. This method is the one we implemented in our application. Several ad hoc procedures can be envisaged starting with the methods just described. In particular, one important issue concerns the policy by which the background is updated or not. In particular, if a pixel is labelled as foreground in some frame, we might want that this pixel does not contribute in updating the background or that it contributes to a lesser extent. Similarly, if we are dealing with a region of interest (ROI), i.e. a portion of an image that you want to filter or perform some other operation on, we might want to fully update the background only if no change has been detected in the ROI; if a change has been detected instead, we may decide not to update any pixel in the background.

(a) Asphalt detection: small asphalt samples on the driveways are used to remove the image's parts with different values of hue and saturation

(b) Canny edge detection is computed on the input image to determine if anything is inside the parking lot

Figure 18: Image processing.

Real-time free space detection

Each camera is required to send real-time information about the spaces it monitors. This should be a vector of probability values of the occupancy of each parking lot. Based on the

consideration that an empty lot should appear as plain asphalt we perform two different image analysis to reach a strong belief of the status of the parking lot. First we use a Canny operator to obtain a very neat image of the vehicles contours (see Figure **18**.b): performing the edge detection of the frame acquired at time t we guess if a vehicle occupies the ROI R_k calculating the index $e_k(t)$ which is proportional to the ratio of edge pixels with respect to the total number of pixels in R_k, i.e.:

$$e_k(t) = \#(\text{edge pixels in } R_k)/ \#(\text{pixels in } R_k)$$

An empty lot has an index near to zero, while one with some edge activities in it gives a value in the range 5%-12% (experimental results).

The second index is based on the "asphalt detection". We check periodically small rectangular "asphalt samples" on the drive way (we use the background image for that, so no moving vehicle is on the region of interest) and we look in the rest of the image for similar hue and saturation values. The result is shown in Figure **18**.a. Similar to previous calculation referring to frame t and ROI R_k, the second index $a_k(t)$ is proportional to the ratio of asphalt pixels with respect to the total number of pixels in R_k, i.e.:

$$a_k(t) = \#(\text{asphalt pixels in } R_k)/ \#(\text{pixels in } R_k)$$

In this case, an empty slot (many asphalt pixels) has a value near to 1, while has a value in the range 0%-20% with a vehicle.

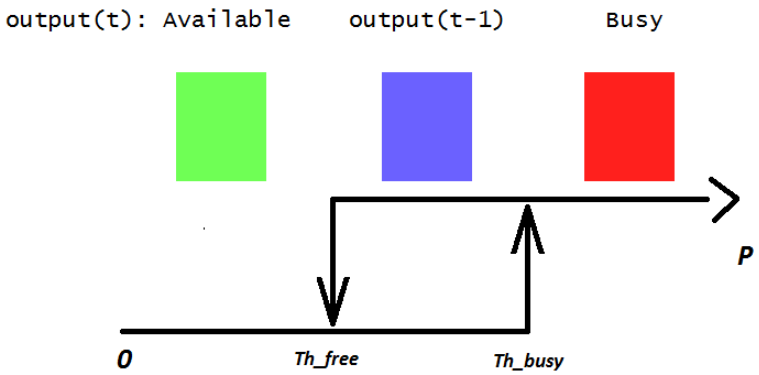
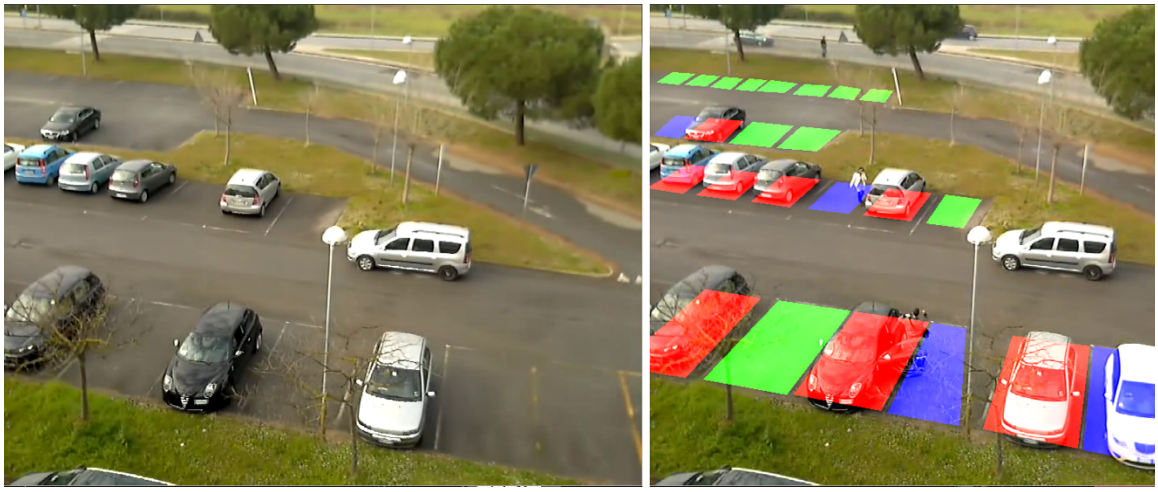


Figure 19: We use two experimental thresholds to determine the status of the parking lot: busy or available

The combination of the two index create the final belief of the sensor which indicates the probability of the occupation of the parking lot.

$$P_k(t) = e_k(t) * (1-a_k(t))$$

A hysteresis function with two experimental thresholds is used for the status change condition (see Figure **19**): in this way there is no trembling results because all the uncertain condition fall in the middle zone where the previous status is maintained. An illustrative example of the output of the above-described vision-based monitoring technique is shown in Figure **20**.



(a) GMM background at time t

(b) Real-time output at time t: red is busy, green is available, blue is uncertain due to partial occupation or vehicle not integrated in the background yet (output is the same of previous frame)

Figure 20: An example of image processing

3.2.3 Ground sensors

In our intelligent parking management system we have also integrated ground sensors that are produced and commercialized by the Kiunsys start-up. In particular, this technology consists of battery-powered ground sensors, which are equipped with magnetometers, IR proximity sensors and temperature sensors, and data collection units (DCUs). The ground sensors send data about the status of the monitored parking lot to a DCU every 30 seconds. The DCU must be placed on a pole at least 3 meters high and it can manage up to 35 ground sensors. A DCU sends the collected data through a configurable TCP Socket Stream over a WLAN every N seconds (configurable parameters) to a multi-threaded java application, called PSSJSON. This application performs a preliminary filtering on the gathered data and it implements the algorithm to analyse the data sent by the sensors for determining the state of a stall according to the values of the signals sent by the sensor. For instance, the application may determine the state of the parking lots based on the last three sensor readings in a range of time to eliminate spikes due to external interference. It can also discard duplicate packets received in a configurable interval of time from more than a DCU. Furthermore, it also sends to an external web service the value of state (when changed), the date of the event, the temperature and the battery voltage. Finally, the application can store a maximum number of records if requested and delete a specific number of records after maximum number of records is reached. For the sake of completeness, in Figure **21** we show a typical deployment scenario for the ground sensors and the DCU.

The goal of integrating these ground sensors in our intelligent parking management system is twofold. First, they are used as a benchmark reference to compare the detection accuracy of the prototype sensors that have been developed based on image-processing technologies. Second, they increase the technology heterogeneity of the system and this allows us to validate more

sophisticated mechanisms for data fusion between sensor measurements with different characteristics.

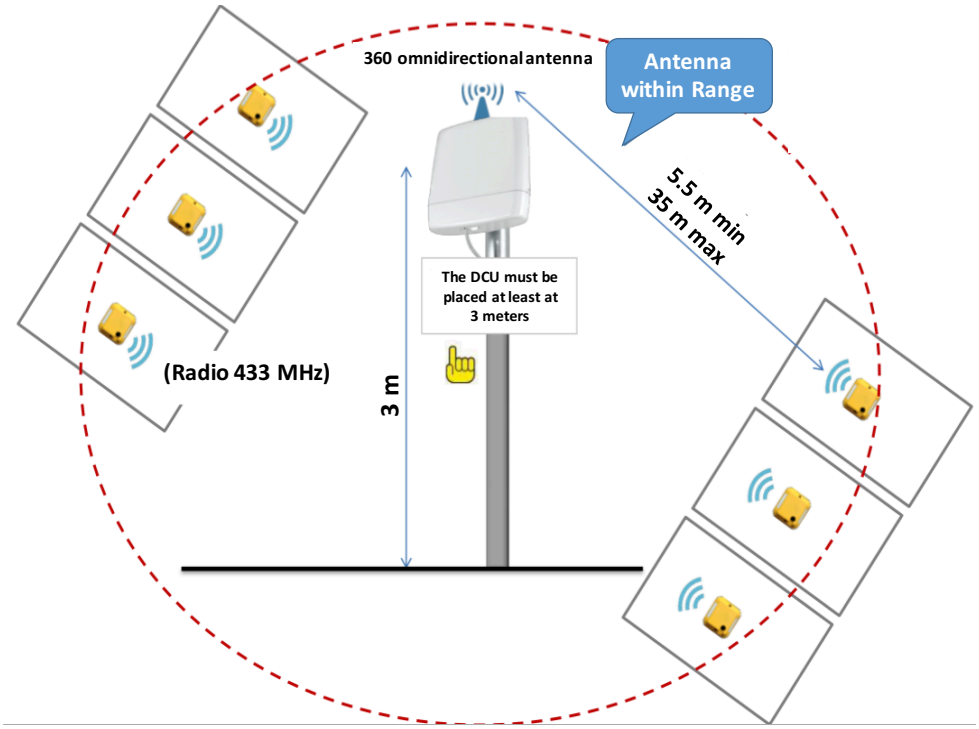


Figure 21: Ground sensors and deployment architecture

3.3 DATA MANAGEMENT LAYER

The data management layer consists of several modules and in the following we explain in detail each module.

3.3.1 Data Acquisition & Filtering

This module is responsible for receiving the data from the sensors that are deployed in the parking field. This module has been implemented as a RESTful Web Service. The camera-based sensors communicate with this Web Service over HTTP. In particular, our RESTful Web Service accept HTTP POST methods that contain JSON messages. Two types of JSON messages have been developed for the smart cameras. The first message is CONF, which is used to register a new sensor and to pass to the management system some basic information on the area that is monitored by each sensor. The second message is EVENT, which provides information on status changes for each parking lot. An example of a CONF message is illustrated in Figure **22**.

{"MSG_TYPE":"CONF",

"MAC":"b827eb962ec8",

"CAMERA_TYPE":"1",

"COORDINATES":"longitude: 11.204242199999953;latitude: 43.8071809;altitude: 13.9",

"POSITIONS":[230,372,190],

"WEIGHTS":[0.5, 0.9, 1],

"STATUS":[0, 0, 0]

}

Figure 22: JSON CONF Message

More precisely, the meaning of the fields in the CONF message are the following:

- MSG_TYPE: the type of message (CONF or EVENT)
- MAC: MAC address of the sensor
- CAMERA_TYPE: sensor technology (1=high-performance smart camera, 2=low-power embedded smart camera)
- COORIDNATES: geolocation of the sensor
- POSITIONS: identifiers of the parking lots that are monitored by the sensors
- WEIGHTS: a real number between 0 and 1, one for each monitored parking lot, which indicates the goodness of the sensor to monitor that parking stall (1= the best, 0=the worst)
- STATUS: the status of the parking slot at the sensor initialization (0=empty, 1=occupied)

An example of an EVENT message is illustrated in Figure **23**.

{"MSG_TYPE": "EVENT",

"DATE": "2015-09-10 12:05:07 +0200",

"STATUS": "0",

"MAC": "b827eb962ec4",

"POSITION": "4",

"CONFIDENCE": "62",

}

Figure 23: JSON EVENT Message

More precisely, the fields in the CONF message have the following meaning:

- MSG_TYPE: the type of message (CONF or EVENT)
- DATE: timestamp of the message
- STATUS: the status of the parking slot at the sensor initialization (0=empty, 1=occupied)
- MAC: MAC address of the sensor
- POSITION: identifier of the parking lot which this event refers to

- CONFIDENCE: a integer number between 0 and 100 that estimates the reliability of the reported status (0=the lowest, 100=the highest)

3.3.2 Data Fusion

The Data Fusion module is at the heart of our intelligent parking management system. Indeed, it is responsible for fusing the data received from different sensors and to decide whether the parking slot is empty or occupied. It is important to point out that the sensor measurements may have different reliability, or a sensing technology may be more accurate than another technology (e.g., magnetic sensors are typically the most precise). In addition, the data collection is event-driven and the reception of status update is asynchronous (e.g., a sensing technology may be faster than another technology to detect the occupation status of a parking lot). Finally, sensor measurements can be conflicting and a conflict resolution algorithm is also needed. The data fusion module is responsible for providing this conflict resolution capability.

Figure **24** shows a flow diagram of our data fusion algorithm. First of all, we adopt a threshold-based mechanism to decide the degree of reliability of a sensor measurement about the occupation status of a parking lot. More precisely, let R be the reliability associated to a sensor measurement that reports a change of occupation status (i.e., from empty to occupied or vice versa). Then, we assume that:

- $R < a$: reliability is low and the change of occupation status is *unlikely*;
- $a < R < b$: reliability is intermediate the change of occupation status is *uncertain*;
- $R > b$*:* reliability is very high and the change of occupation is status is *certain*.

Now let us assume that at time t the data fusion module receives and EVENT message for a parking lot k that reports a change of occupation status from S_0 (= empty) to S_1 (= occupied), being S_0 the current occupation status. Then, let R denote the reliability of the EVENT, T the type of the sensor technology and W the weight of the sensor measurement[3]. If $R>b$ we immediately change the occupation status for the parking lot k to S_1. If $R< a$ we discard the EVENT message. Finally, if $a<R< b$ we activate an observation timer t_W during which we collect other EVENT messages that may be sent by other sensors that are monitoring the same parking lot k. When this timeout expires we collect the list of received EVENT messages. If any of these messages reported a change of occupation status with $R>b$ we confirm the change of status. Alternatively we count all the uncertain events that confirm the change in occupation status, say n_1, and the sensors that did not report a change of status[4], say n_0. Then, we apply a simple majority criterion to make a decision: if $n_1 > n_0$ then a change of status is confirmed otherwise if $n_1 < n_0$ we keep the current status. The uncertain condition is when there is a tie. In this case, we search for each technology T only the sensor measurement with the highest

[3] The weight measures the goodness of a sensor to monitor a specific parking lot. For instance, in the case of camera-based sensor the weight depends on the view angle of a camera for a specific parking lot.

[4] It is important to point out that if a sensor that is monitoring the parking lot k has not reported a change in the occupation status, this is implicitly assumed as an uncertain measurement because we already know that there are other sensors with a different perception of the parking status.

weight and we again apply the majority criterion. If there is still a tie, then we take a decision according to the technology with the highest priority[5].

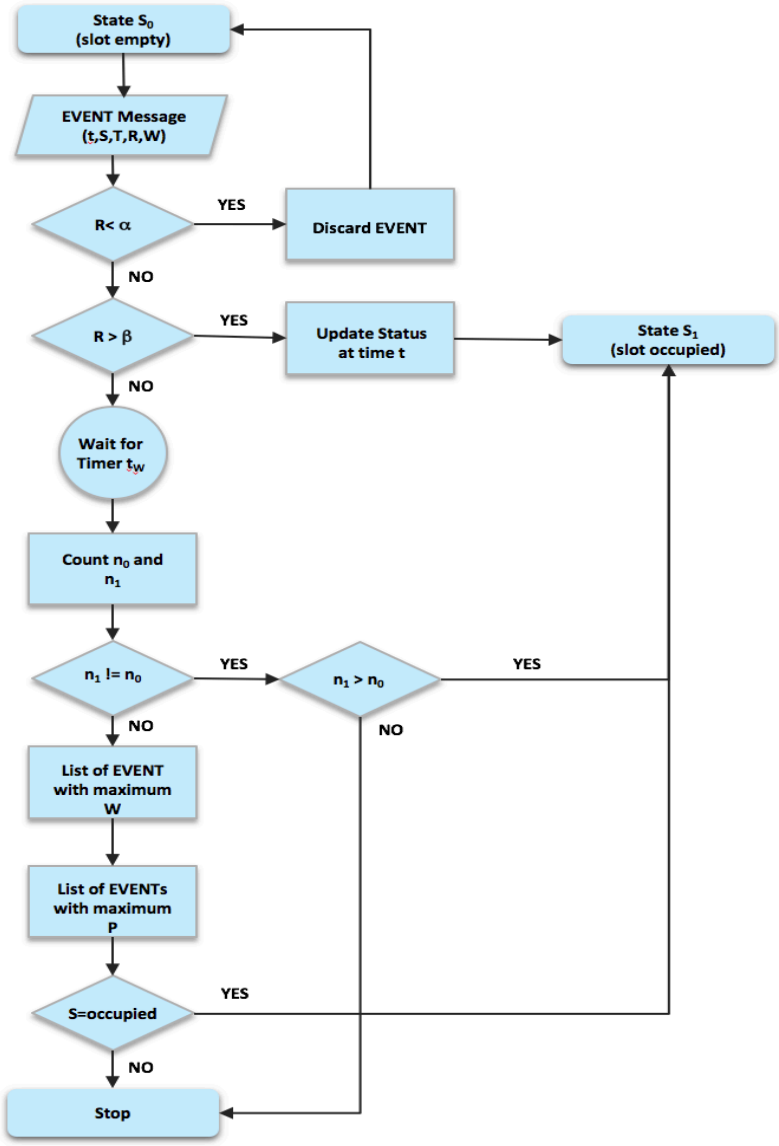


Figure 24: Flow diagram of the conflict resolution algorithm

3.3.3 Data Integration

The Data Integration module is responsible for providing the integration of the system with the Open Data in the cloud platform. This module is implemented as a RESTful Web Service over HTTP that interacts with the Open Data Web Service using classical HTTP methods (GET, POST, PUT, DELETE, etc.). The main tasks implemented by this module are briefly described in the following.

[5] We remind that each monitoring technology has a priority level based on its accuracy. We decided to assign to the ground sensors the highest priority level.

Application Registration

This function implements the registration of the application on the Open Data platform. The registration method contains the URL of the Smart Parking application, http://parking.smart-applications.area.pi.cnr.it:8080/SmartParkingWeb/

Sensor Registration

This function implements the registration of a sensor on the Open Data platform.

An example of this REGISTRATION message is illustrated in Figure **25**.

{"idLocalSensor":"513",

"characteristic":"Raspberry;Raspberry;Raspberry;b827eb962ed0",

"geoPosition":"longitude: 11.204242199999953;latitude: 43.8071809;altitude: 13.9",

"idApp":"SmartParking"

}

Figure 25: JSON REGISTRATION Message.

where idApp is the identifier of the application, geoPosition contains the geographical coordinates of the sensor, and idLocalSensor is the sensor identifier that is used in the application (this is needed to create a correspondence list in the Open Data platform), and characteristic is the list of sensor attributes.

Association Sensor to Parking lot

A single sensor can monitor multiple parking lots. This function creates in the Open Data platform the correspondence matrix between sensors and parking lots. An example of this ASSOCIATION message is illustrated in Figure **26**.

{"idApp":"SmartParking",

"idLocalSensor":"513",

"stalliList":[184,521,245]

}

Figure 26: JSON ASSOCIATION Message.

where stalliList is the list of identifiers of the parking lots as used in the smart parking application.

Data Archiving

This function implements the transfer of a bundle of measurements to the Open Data platform for data archiving and post-processing. An example of this DATA message is illustrated in Figure **27**, which contains the occupation time intervals for each monitored parking lot.

{"idApp":"SmartParking",

"datiSensoriList":[{"idSlot":40,"startRest":"2015-10-08T08:40:07Z","endRest":"2015-10-19T15:55:11Z"},{"idSlot":11,"startRest":"2015-10-19T07:31:51Z","endRest":"2015-10-19T15:53:25Z"},{"idSlot":29,"startRest":"2015-10-19T07:45:53Z","endRest":"2015-10-19T16:08:56Z"},{"idSlot":44,"startRest":"2015-10-19T08:38:49Z","endRest":"2015-10-19T16:23:11Z"}]

}

Figure 27: JSON DATA Message

3.3.4 Local data storage

For the sake of completeness in Figure **28** we show the SQL scheme of the database that is used for local storage and real-time processing of the sensor measurements.

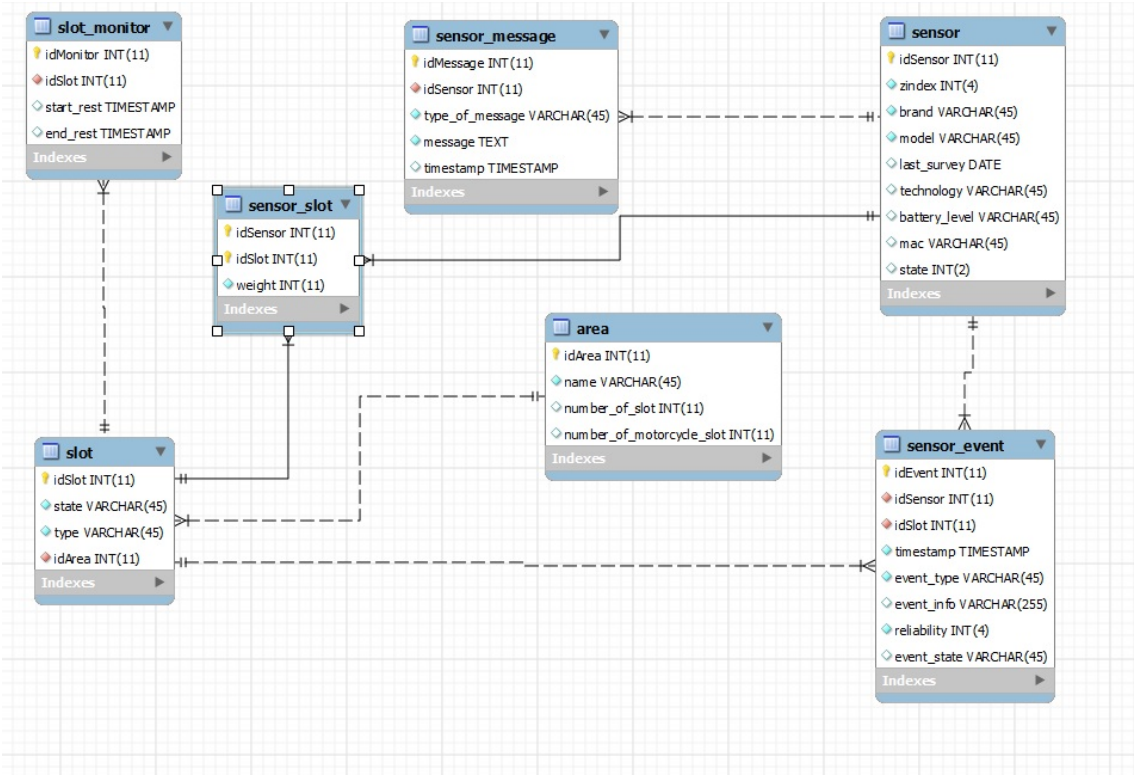


Figure 28: SQL scheme of the database that is used for local storage and real-time processing of sensor measurements

As shown in the figure, separated tables are used to maintain information on the monitored parking slots, the deployed sensors, the events generated by the sensors and the occupation status of each parking lot.

3.4 APPLICATION LAYER

The Application Layer provides the online tools that allow the users to interact with the intelligent parking management system. These interactions are supported through a dedicated web application, called Smart Parking. It is out of the scope of this document to provide a detailed description of the structure of the web application and the diagram of the uses cases. In this section we give a brief description of the main functionalities and we show screenshots of the main sections of the web application. Specifically, the Smart Parking application provides:

- The graphical visualization of a map of the parking fields of the CNR Area with a visual representation of the occupation level of each parking sub-area;
- The real-time information on the occupation status of each parking lot;
- Historical information on the use of the parking areas;
- The capability of reserving parking lots;
- Administration tools.

Figure **29** shows a screenshot of the Smart Parking home page. In this site the user can access the various application functionalities. Furthermore, there is a panel with summary information on the amount of free parking lots (normal and for disabled people). Finally, it is also shown a map of the CNR area with a heat map of the occupation status of the monitored parking sub-areas.

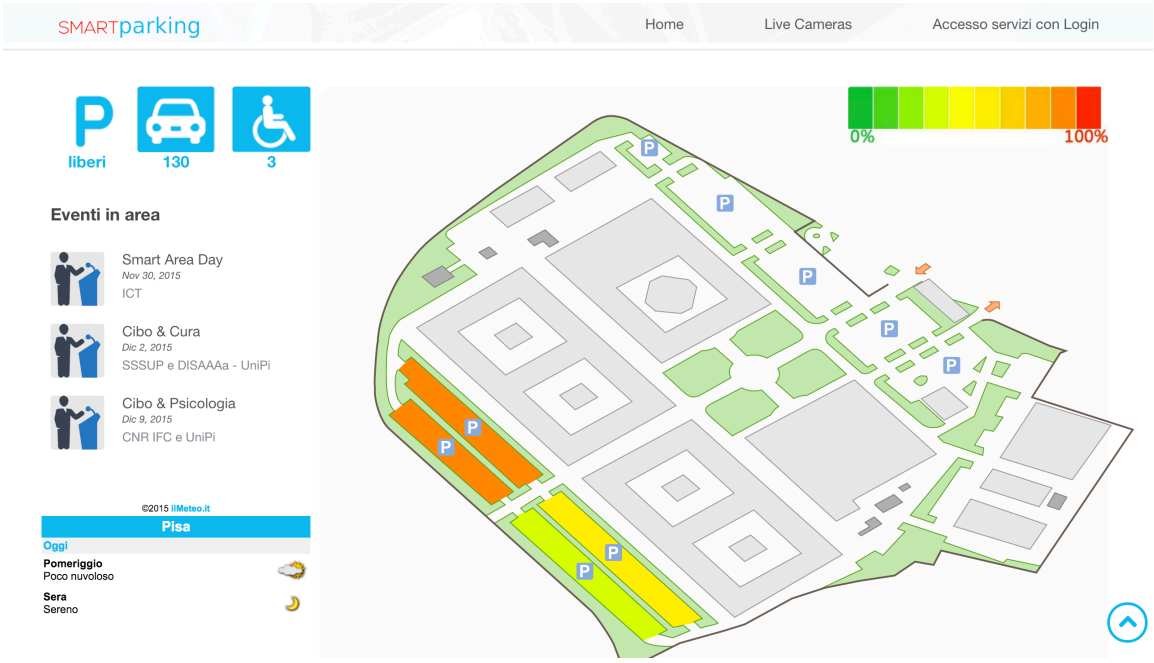


Figure 29: Screenshot of the Smart Parking home page

Figure **30** shows a screenshot of the web page dedicated to each parking sub-area[6]. In this section we report a schematic map of the parking layout with real-time information on the occupation status of each parking lot. In addition, a graph with the last-month history of the overall occupation level of the parking area is also shown.

Figure **31** shows a screenshot of the web page dedicated to statistics and historical information. In particular, a first diagram reports the average occupation level of the different parking sub-areas in the last month for each day of the week, and six different time intervals. Furthermore, a second diagram reports the average number of occupied parking lots in each parking sub-area during the entire week for six different time intervals.

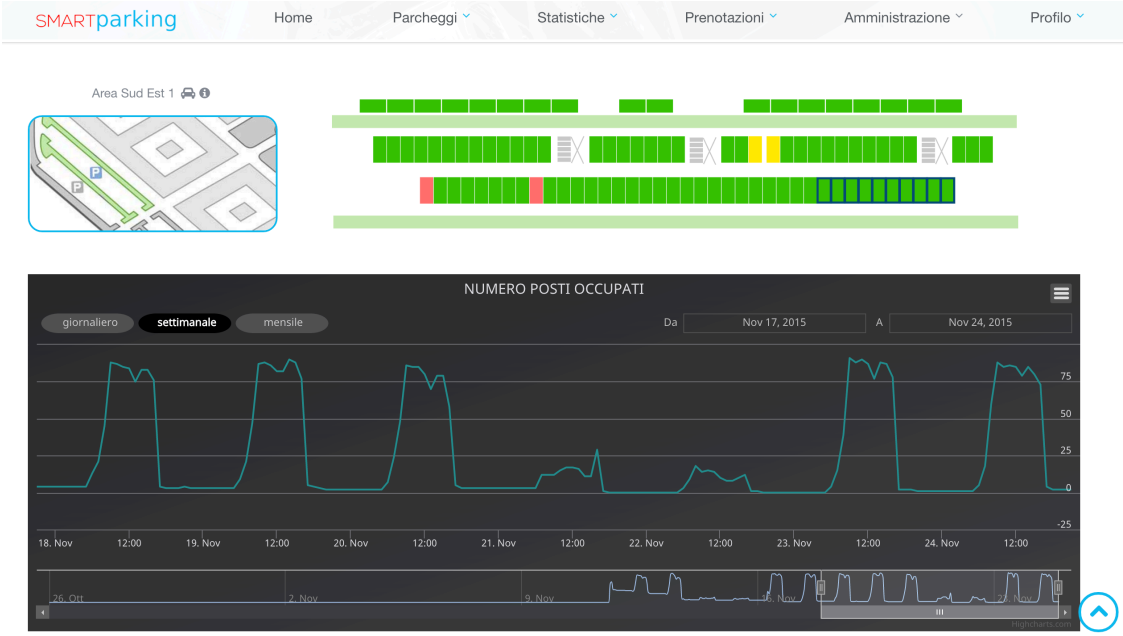


Figure 30: Screenshot of the section related to single parking areas

Figure **32** shows a screenshot of the web page dedicated to the management of the deployed sensors. Various information is displayed, such as the type of sensor, its identifier, the last sensor measurements, etc.

Finally, Figure **33** shows the screenshot of the web page dedicated to the management of the parking reservations. In particular, a subset of parking lots can be managed by online bookings. Limitations are defined on the number of reservation requests a single user can insert. A booking confirmation is sent by the administrator to the requester the day before. It is important to point out that the parking reservation is also used as an incentive for the Smart Shared Mobility application. This feature will be explained in more details in the chapter dedicated to the Smart Shared Mobility application.

[6] We remind that four parking sub-area are monitored in the southern part of CNR Area.

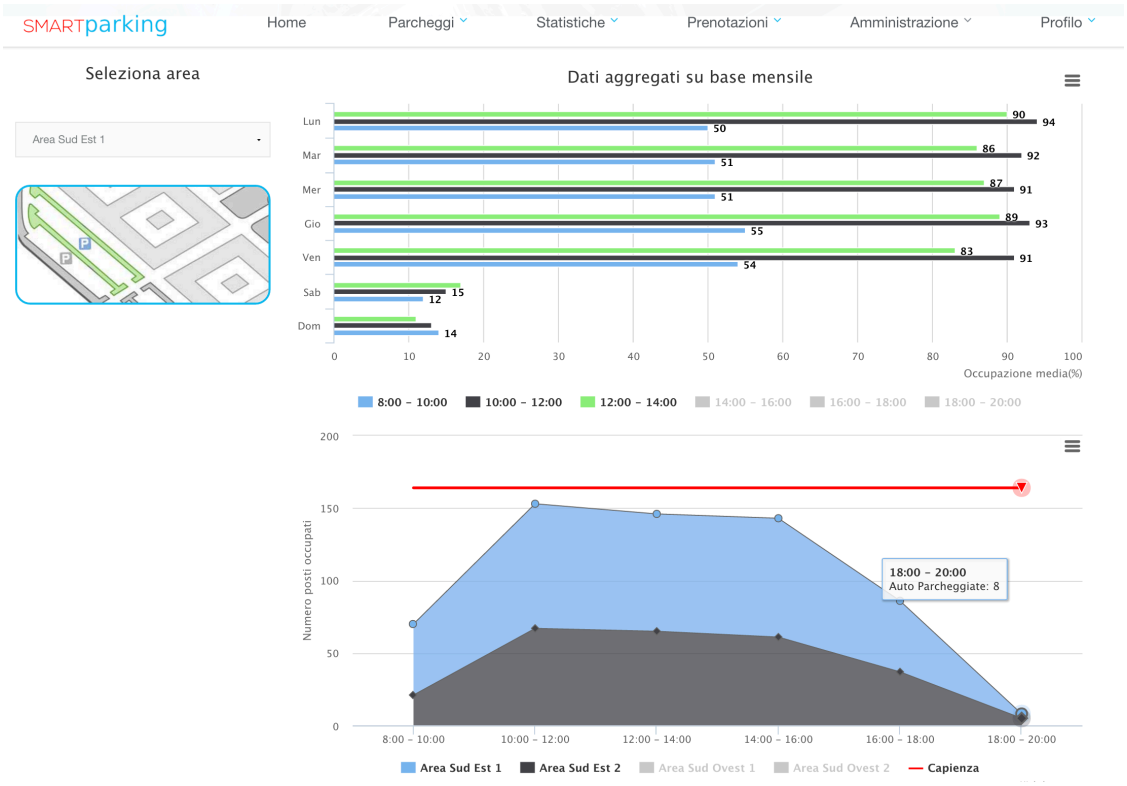


Figure 31: Screenshot of the section related to statistics and historical information

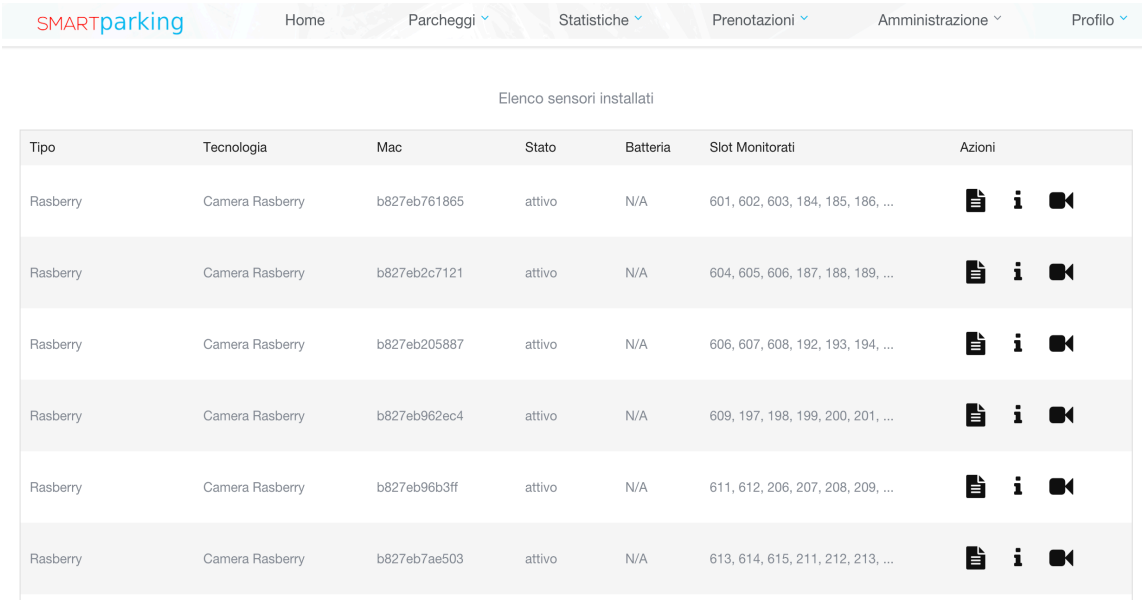


Figure 32: Screenshot of the section related to system administration.

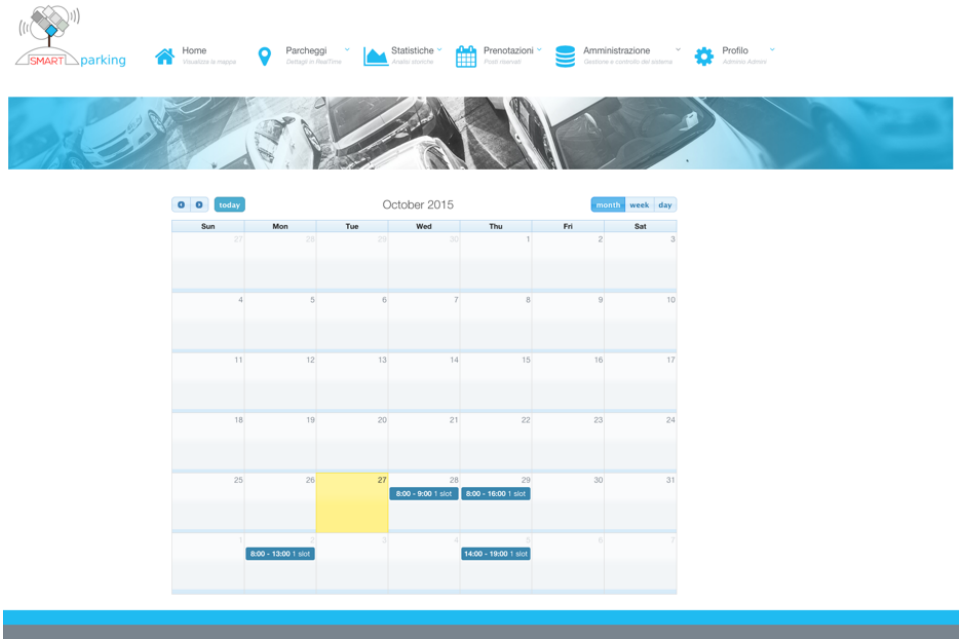


Figure 33: Screenshot of the section related to parking reservations

3.5 References

[1] IBM, "Global Parking Survey: Drivers Share Worldwide Parking Woes", 2011

[2] C. Stauffer, and W. E. Grimson, "Adaptive background mixture models for real-time tracking," Proc. CVPR, Fort Collins, CO, USA, vol. 2, pp. 246-252, 1999.

[3] N. Buch, S. A. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," IEEE Trans. ITS, vol. 12 (3), pp. 920-939, 2011.

[4] M. Magrini, D. Moroni, G. Pieri, and O. Salvetti, "Real time image analysis for infomobility," Lecture Notes in Computer Science, vol. 7252, pp. 207 – 218. 2012.

[5] P. Remagnino, A. I. Shihab, and G. A. Jones, "Distributed intelligence for multi-camera visual surveillance,". Pattern Recognition, 37(4):675–689, 2004.

[6] J. Lopes, J. Bento, E. Huang, C. Antoniou, and M. BenAkiv, "Traffic and mobility data collection for real-time applications," IEEE Conf. ITSC, pp.216 -223, 2010.

[7] K. Kim, T. Chalidabhongse, D. Harwood, and L. Davis, "Background modeling and subtraction by codebook construction," IEEE International Conference on Image Processing, ICIP-2004, pp. 2-5, 2004.

4 THE SMART BUILDING (Ref. Paolo Barsocchi, ISTI)

***In brief**: A network of wireless sensors, appropriately designed and developed, installed in a building, allows: i) the monitoring of the energy consumption of one or more environments, ii) the use of remote control techniques to reduce such a consumption, iii) an easy maintenance of the electrical system, and iv) the recognition of the presence of people within the monitored environment (for security reasons). This sensor network can communicate with other networks of sensor nodes, installed in other environments and using heterogeneous technology.*

The Smart Building application intends: (i) to provide long-term energy consumption monitoring, (ii) to enable energy savings through automated control and configurable policies, (iii) to provide end-users with access to tools and data, thus fostering the energy awareness. To achieve these goals, we developed a wireless sensor network (WSN), where each node is equipped with transducers able to measure the power consumption; the nodes are installed in selected offices of the CNR-ISTI and CNR-IIT Institutes, in the CNR area. The requirements of the system were the low cost of the sensors, the versatility of the system, its easily extendibility, the non-invasiveness of the installations, and the integration with other possible existing equipment. We chose the ZigBee technology since it is a standard-based wireless technology designed to address the low-power requirement, and it also has a rich model for the representation of the device capabilities. Each node of the WSN is connected to a Zigbee sink, which provides connectivity to the Internet of Things (IoT) through the IPv6 addressing.
Measured power values are then made available to applications through a middleware platform and data query services. The platform provides a solid starting point for the development of location-based services, as it delivers an information service for device power consumption data (appliances, lights).
Each in-building sensor is characterized by a physical location (e.g.: office rooms, corridors, common areas, or GPS coordinates).
Activity classification algorithms can be developed by exploiting the in-building sensors, allowing the identification of the in-office activities or habits. Moreover, applications that allow office workers to manage their in-office energy consumption by means of mobile devices can be delivered once that power-switch actuators are put in place and made available on the communication platform for the automated remote control.

4.1 CURRENT TRENDS

Recent studies have shown that in-building infrastructures for energy monitoring lead to interesting conclusions on what is happening in the observed environment and provide relevant information on the resource consumption. For example, there are systems that monitoring the electrical noise within the power-lines of a house and exploiting that each appliance introduces a unique noise signature identify the used appliance. These solutions focus on the monitoring of residential power consumption, representing a good starting point for an in-depth analysis of activities related to the utilization of appliances and their locations. Our solution follows this path developing a pervasive infrastructure represented by the middleware used to build

location-based services and used to integrate different kinds of sensors thanks to its modular nature. A different approach is used by commercial sensors like Kill-A-Watt7 and Watts UP8, which directly measure the consumption of an appliance connected to a particular power outlet. Such a technology lacks in the possibility of aggregating data from multiple sockets and hard-wired lights into the power distribution network, limiting in its ability to offer a common infrastructure to build context-aware and location-based services.

4.2 THE INTEGRATION FRAMEWORK

4.2.1 Wireless Sensor Network

In this Section, we present the chosen hardware solution, how the real power consumption has been estimated and, finally, how we developed the ZigBee device.

1) Hardware Solution. In order to measure the real power consumed by the user in his room, the current waveform is not sufficient. Indeed, when the resistive loads (such as incandescent light bulbs, electric water heaters, etc...) are switched-on, the current and the voltage waveforms are in phase (i.e. the only knowledge of the current waveform is sufficient) and the energy flows to the load. As a consequence, the power is the product of the voltage and the current, at a given time. However, fridges, electric heaters, washing machines and others have inductive or capacitive components in addiction to the resistive part. Most of the energy flows to the load, while the rest is stored and released back afterwards into the mains supply. The current and the voltage waveforms are out of phase and the instantaneous power has a negative part. Moreover, the phase delay is hardly predictable and depends on the specific appliance. In this case, the only knowledge of the current waveform is not sufficient, unless knowing the phase difference. Therefore, in order to measure the current and the voltage values at the same time instant, we use two transformers, namely current and voltage transformers (CT and VT). The former is SCT-013 split core CT by YHDC, the latter is 77DE-06-09 AC to AC adapter by Ideal Power Ltd. The choice of these voltage and current transformers was suggested by an open source project called OpenEnergyMonitor[9] and it is driven by the need to operate within existing buildings without the possibility of changing the existing electrical appliances.
The ZigBee module chosen is the board Open2530, composed by a core module (Core2530) and a support board (ZB500), produced by WaveShare Electronics[10], based on the Texas Instruments System-on-Chip (SoC) CC2530. This SoC provides some functionality, including a sigma-delta 12 bits ADC (Analog Digital Converter). Both current and voltage signals must be transformed and conditioned in order to be correctly interpreted by the ADC (Figure **34**). The current conditioning circuit consists of the burden resistor and a biasing voltage divider: the resistor turns the current signal into a voltage signal, while the divider brings the sinusoid upon positive limits. The voltage conditioning circuit consists of a first divider, which scales down the waveform, and a successive biasing voltage divider, which adds an offset to eliminate negative components.

---

[7] http://www.p3international.com/products/p4400.html
[8] 4http://www.wattsupmeters.com/
9 http://openenergymonitor.org/
10 http://www.wvshare.com/

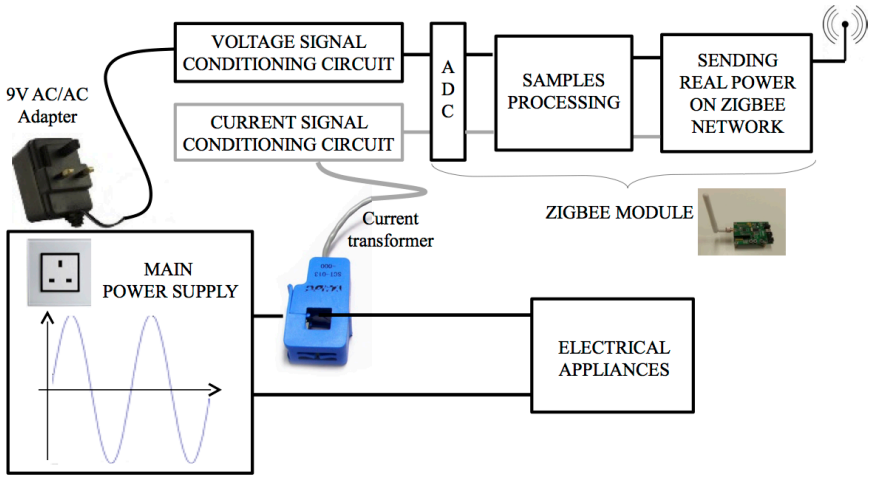


Figure 34: Power consumption evaluation system

2) Real Power Evaluation: The power consumption evaluation system shown in Figure **34** introduces many sources of error. In particular:

- all resistors used in the conditioning circuit have resistance tolerances of 5%,
- VT has an output voltage tolerance of 5%,
- CT has a non-linearity of 3% with respect to its ideal output characteristic.

In order to reduce the introduced error, it was necessary to introduce a calibration phase both at the beginning of the sample processing and also during the processing itself. The measured voltage signal (V) and the measured current signal (I) are achieved by using the following equation:

$$V(ti) = KV * (Vb(ti) - Vm)$$
$$I(ti) = KI * (Ib(ti) - Im)$$

where V_b and I_b are the ADC output signals at *b* bits, V_m and I_m are the mean values of V_b and I_b, respectively, and K_V and K_I are the voltage and current constant, respectively.
The values of these constants depend on the ADC features, on the conditioning circuits and on the current and voltage transformers. The values of V_m and I_m are achieved by using two different numeric filters that update these values at every sample. Measuring the time difference between the current sampling and the voltage sampling we measured that this value is greater than 220us only one time every thousand measures. Therefore, the error produced estimating the real power (as a product between current and voltage) is less than 0.5%.

3) The ZigBee Devices Development: In order to send the measured real power to the ZigBee network, we developed a ZigBee device exploiting the TI Z-Stack, a fully ZigBee-PRO compliant stack that offers the support to Smart Energy Profile and Home Automation Profile, two particular Application Profiles (AP) defined by the ZigBee Alliance. The characteristics of the developed ZigBee are listed in Table **2**.
We used the ”report attributes” command, defined in the ZigBee Cluster Library, to report the value of the Real power to the coordinator (sink). The “Current Summation Delivered” attribute (ID 0x0000), belonging to the Metering cluster, has been configured for this purpose. A reporting message is generated and sent when the reporting timer expires and, at the same time, the attribute undergoes a change of value, in positive or negative, equal to or greater than a delta. Timer expiration and delta value are configurable by the user at run time.

Device Type (ID)	Application Profile (ID)	Clusters List
Metering Device (0x0501)	Smart Energy (0x0109)	Basic Key Establishment Identify Metering

Table 2: Characteristics of the developed ZigBee device

4.2.2 Middleware Architecture

The proposed system provides tools and mechanisms that cover several needs: integration of ZigBee sensor network devices in Java applications, exchange of sensor data and actuators control in distributed environments, applications for long term monitoring and decisional processes support. The software system architecture has been divided in three parts: 1) a sensor network integration layer, 2) a distributed communication platform, and 3) top end applications.
In the following, a brief description of these components is provided.

1) Sensor network integration layer
ZigBee for OSGi (ZB4O7) is a software stack, implemented as a set of OSGi bundles, that allows to easily integrate ZigBee devices in Java applications. It provides a Java object representation of sensors and actuators, relying on the modular software model and service-oriented design approach fostered by the OSGi platform. A basic device representation software module discovers devices on a ZigBee network thanks to a ZigBee dongle and its driver module. A basic device representation layer maps discovered devices into a Java objects with a low-level interface. These objects are then registered in the OSGi service registry as services with descriptive properties. ZigBee Profile specific software modules wrap Basic ZigBee devices in Profile specific Java objects. The latter representation allows to efficiently exploiting all features offered by the physical bottom-end devices with an higher level interface, promoting minimal integration effort. Java modules rely on the whiteboard pattern registry provided by an OSGi container in order to discover ZigBee devices with specific properties by registering listeners on the container registry; notifications on target devices life cycle events are thus enabled: registration, un-registration, and properties modification.

2) Communication platform
The system uses ZB4O to manage sensors and actuators available in the ZigBee networks. In our scenario, ZigBee devices are spatially grouped in separate sensor networks, each one provided with a gateway node, hosting an OSGi container with the ZB4O stack (Figure **35**). The gateway node provides access to the discovered sensors through an IP network and a communication platform (called *Sensor Weaver*), which is an evolution of the software developed in the context of the FP7 GiraffPlus[11] project. The main goal of Sensor Weaver is to provide a secure communication platform for the exchange of sensors data and actuators control in a distributed sensor network environment. Moreover, Sensor Weaver also provides tools and applications that enable long-term sensors monitoring and automated actuators control. Following a top-down approach, the communication platform is constituted by the

11 http://www.giraffplus.eu/

following components (Figure **35**): a client communication module, a messaging client module, and a communication Middleware.
These components are available both for the OSGi desktop platform and for Android mobile platform. Sensor Weaver Communication API offers a simple and technology independent abstraction for representing both sensors and actuators: a Sensor Weaver Service, that is, a metadata rich object that may periodically publish data (like a sensor) and may provide some methods (like actuators) for remote control. Services can be either physical components (like sensors), with a spatial and time location, or abstract, like software modules that provide processed data based on real sensor measurements. Java clients are provided with a Java object-based language for the description of services and an API that allows announcing them, publishing sensor data, discovering services and controlling them.

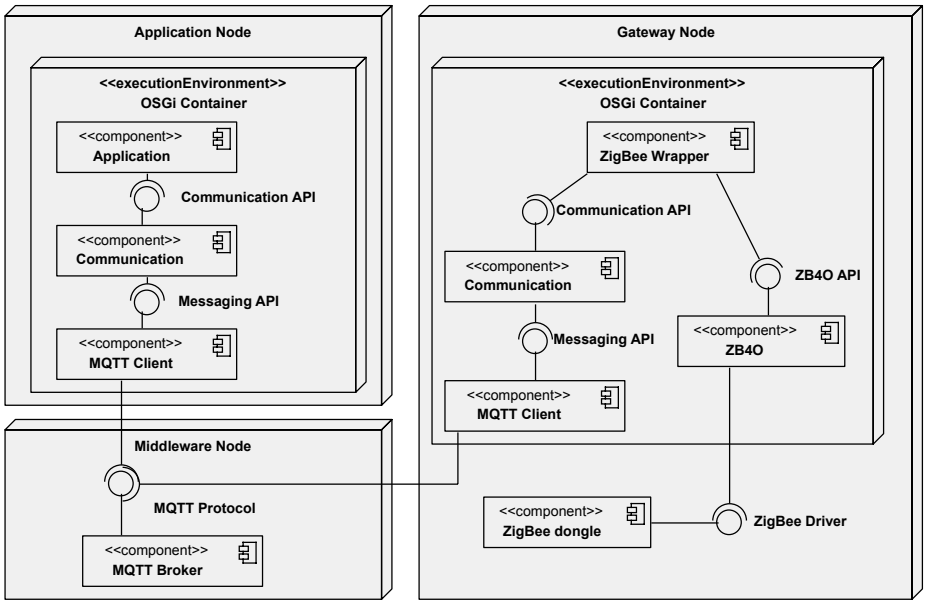


Figure 35: Gateway and Application nodes deployment diagram

The communication module ensures that clients have limited access to data flowing on the communication buses, providing means for limiting access to messages and resources with configurable scopes of visibility. Bus access is regulated with certificate-based authentication mechanisms and authorization policies. The communication module leverages a Message-oriented Middleware system based on the IoT /M2M MQ Telemetry Transport**12** protocol, an extremely lightweight publish/subscribe messaging transport. The communication module directly connects to a MQTT broker (Mosquitto**13**) by means of a Java client library with connection failover and message buffering mechanisms, preventing data loss when connectivity issues arise on the IP network. In order to separate communication concerns on the Middleware, several communication buses have been envisioned, each one aiming at a specific concern (

Figure 36): (i) a Service Bus for service life-cycle events, (ii) a Context Bus for sensor measurement updates, (iii) a Control Bus for actuators methods invocations. The Mosquitto

12 http://mqtt.org/

13 http://mosquitto.org/

instance is configured to authenticate incoming clients, to establish encrypted connections only and to ensure that dynamically configured topic access policies are applied. Moreover, the system architecture can be easily **scaled to address wide area monitoring scenarios, as multiple cooperating messaging brokers can be deployed.**

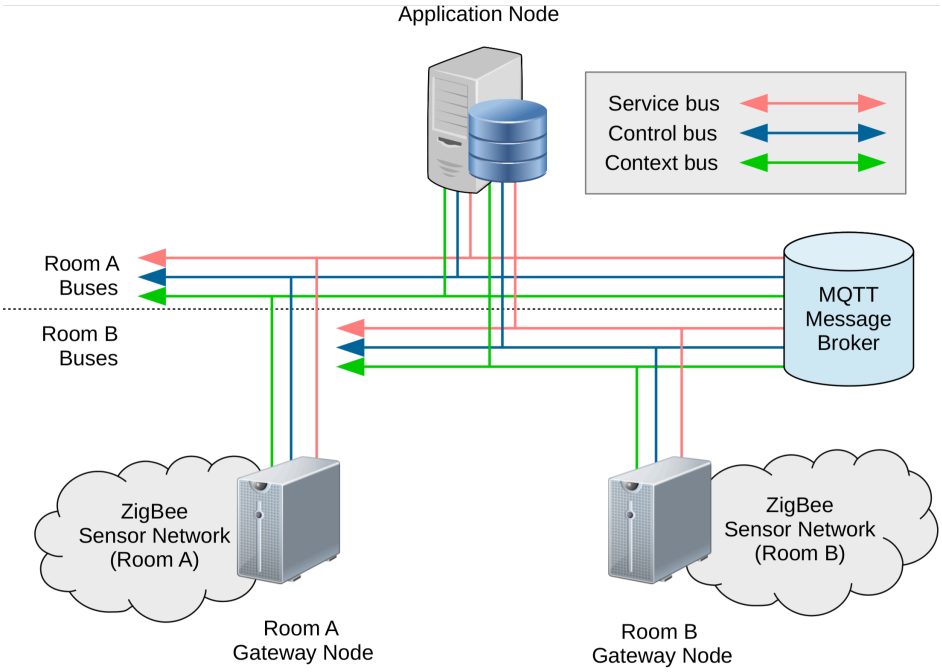


Figure 36: Sample scenario with Message-oriented Middleware bus scoping

3) Applications: Top-end applications can leverage the capabilities of the communication platform to provide decisional algorithms based on sensor data and actuators control (Figure **35**). Those components may need to rely both on near real-time sensor data and on stored data. Sensor Weaver provides tools and interfaces for sensor data persistence and access. Since data incoming from sensor networks are big in volume, heterogeneous by nature, and are produced and consumed according to known profiles, the database technology must comply with these aspects. Sensors typically transmit few bytes of sensed values with a given frequency or at irregular time intervals, whenever a monitored event happens. Produced data are often consumed in chunks of multiple values measured by one or multiple readers. In our opinion, a NoSQL database is the best fit for storing sensor data as it guarantees efficient management of big data with horizontal scaling techniques and better write/read performance, when compared to SQL solutions, for the aforementioned usage scenarios. After a brief evaluation phase based on related works, we decided to adopt MongoDB as the technology for the sensor measurements persistence. A software (Data Recorder) module, running on an Application node (Figure **35**) can access sensed data and write them on the database, in the meantime also keeping track of the sensors' life cycle events. Data coming from a single sensor can then be organized as chunks of homogeneous data (data offerings), each one characterized by a specific time interval and measure properties (e.g. unit of measure). A REST service allows other components to discover available data offerings and to query for raw or aggregated data, offering a query interface based on the sensor representation model. The exposed middleware

architecture provides a backbone communication platform for the exchange of sensor data. The API delivers a technology-agnostic model for the representation of sensors and actuators. This aspect opens the platform to the integration of heterogeneous sensor network technologies (Konnex, Bluetooth LE, etc.) and sensor types (HVAC, lightning, and monitoring).

4.3 THE INSTALLATION

In order to monitor the electric consumption in an office during a working day, we deployed a sensor board over the office's ceiling to monitor the lights' power consumption and the loads from electric outlets. Figure 37 shows the main page of the developed application able to show the reporting values of the deployed WSN. Figure **38** shows one of the results of our infrastructure in terms of long-term monitoring of the office power consumption offering the possibility to build services able to infer working activities and movements in office buildings. The first red line is the power consumed by the lights in the monitored room, while the second line is the Passive Infrared (PIR) output activated when a movement of the monitored room is detected.

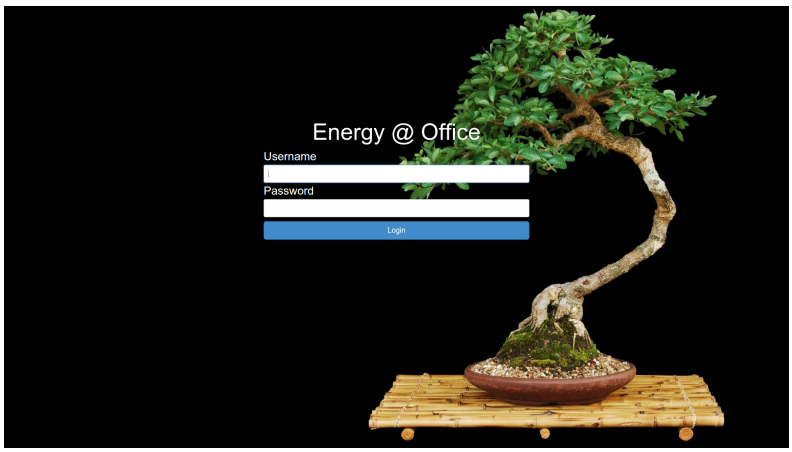


Figure 37: The main page of the smart building application

The power consumed in a room during a working day can be leveraged by location-based services. Indeed, the power consumption can be used to provide the position of a person (in front to the personal computer or inside/outside the office) and also to recognize whether or not the user activity in that moment is energy-consuming.

The possibility offered by this infrastructure to run middleware instances also on mobile platforms paves the way for the development of personalized services that allow the user to remotely control its office appliances and their status, thus promoting an increased involvement of the user in the process of the energy consumption reduction. Furthermore, fusing the energy consumption context information with the localization data coming from the user's smartphone, the infrastructure can autonomously apply an intervention plan to avoid inefficient use of electric loads. In future works we plan to integrate different modules in order to address the usage of new platforms (e.g. embedded controller like Arduino[14] or emerging technologies like Intel POEM) and execution environments (e.g. javascript web engine like node.js).

14 www.arduino.cc/

The proposed system is able to control if the electrical appliances, such as the neon, work or not. Indeed, knowing that the power consumption of each neon is around 35W the system is able to report if the light in an office (composed by 2 neon) correctly works.
Finally, the system by exploiting the PIR, noise and electrical power consumption sensors, is able to detect if a person occupies the office or not, and automatically switching off the lights or notify the energy manager when the room is identified as empty. This feature is particularly useful for security reasons.

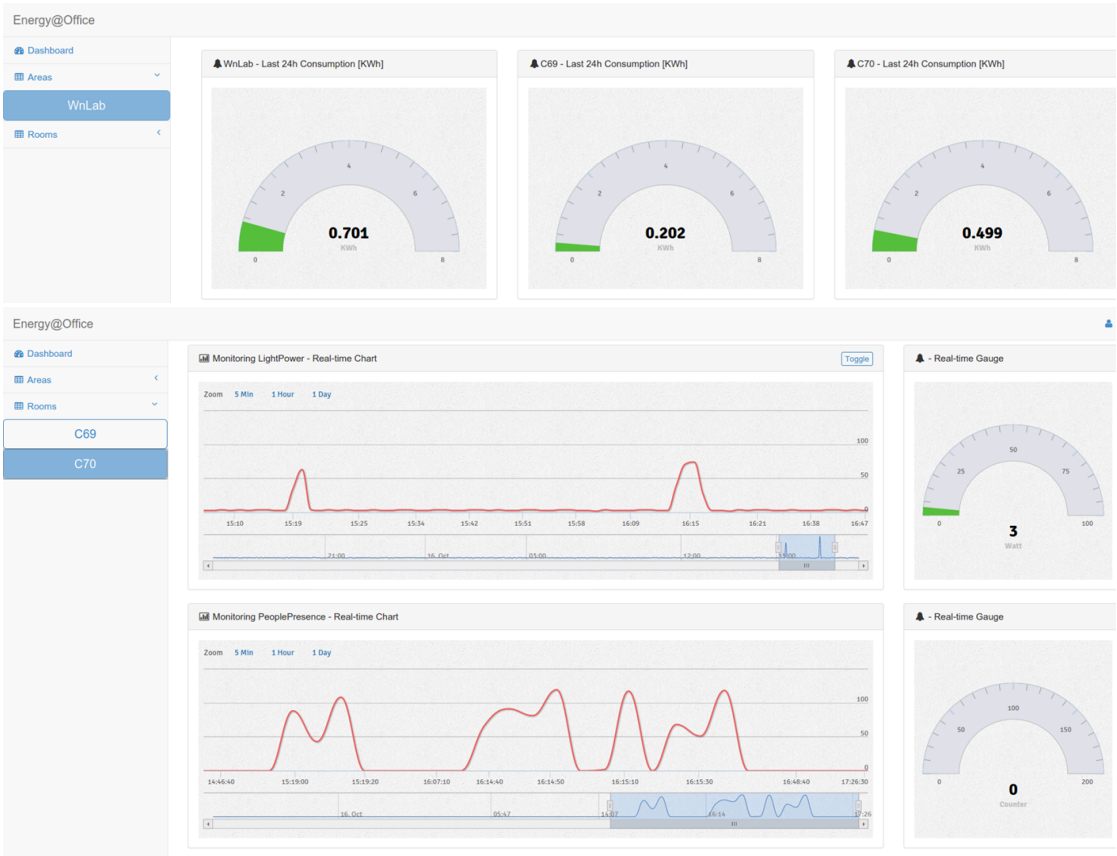


Figure 38: Web application of the power consumption of a room. The first line is the lights' power consumption, while the second one is the PIR sensor.

5 THE SMART NAVIGATION (Ref. Andrea Marchetti, IIT)

***In brief**: Smart Navigation allows staff and visitors to consult from their own computer or smartphone a complete map of the area of the CNR in Pisa. The map can be interrogated to find offices or people and be guided to the desired final destination by means of graphic indications. Thanks to the indoor location, the user can also utilize the map in combination with an Android app to display in real time his location within the area.*

The smart navigation application (prototype available at http://wafi.iit.cnr.it/smartarea/) is a system consisting of an interactive, HTML5-based map of CNR and an indoor positioning service. The map is the entry point for users, and it can be used with or without relying on the indoor positioning service. If the service is used, the map is able to show the user's position inside the building. In any case, the map can be used to see the user's own position outside the building, explore the building floors, search for rooms or people, or obtain directions to a specified destination.

5.1 HTML5 INTERACTIVE MAP

This part of the system is a web application consisting in a rich interactive map of CNR, available to users with modern desktop or mobile browsers. The application is designed to be used by both CNR personnel and visitors that are not familiar with the building.
No external proprietary services such as Google Maps or similar are used at runtime by the web application, which relies only upon a collection of open source Javascript libraries (three.js and d3.js).

5.1.1 Map creation pipeline

The smart navigation application leverages a collection of tools and procedures to build and maintain the map. Currently, it consists in a combination of 3D editing software (AutoDesk's AutoCad (file format only used as import) and Trimble's Sketchup), and web rendering technologies (the already mentioned Javascript libraries).
The AutoCad DWG files containing the floor plans of CNR main buildings are imported into SketchUp, to support an easy 3D editing process. The imported floor plans are then used to manually create a 3D model of the buildings: the most relevant features of the maps are traced and included in the 3D model. The model is georeferenced, in order to match real world coordinates to the coordinates of the 3D representation.
The model is then segmented into a *hierarchy of groups*. The first level of the hierarchy is used to group the different floors (ground, first, second floor, and rooftop). Walls are in separate groups in order to be able to hide them when the user needs to see inside the building. Rooms, which are interactive elements in the map, are separated from inactive elements at the second level of the group hierarchy. Each room is finally given an identifier at the third level of the hierarchy.
Whenever the model is edited, the editor must export it from SketchUp as a Collada model (DAE), which is then loaded at runtime for rendering by the DAE import facility of three.js.

5.1.2 User Interface

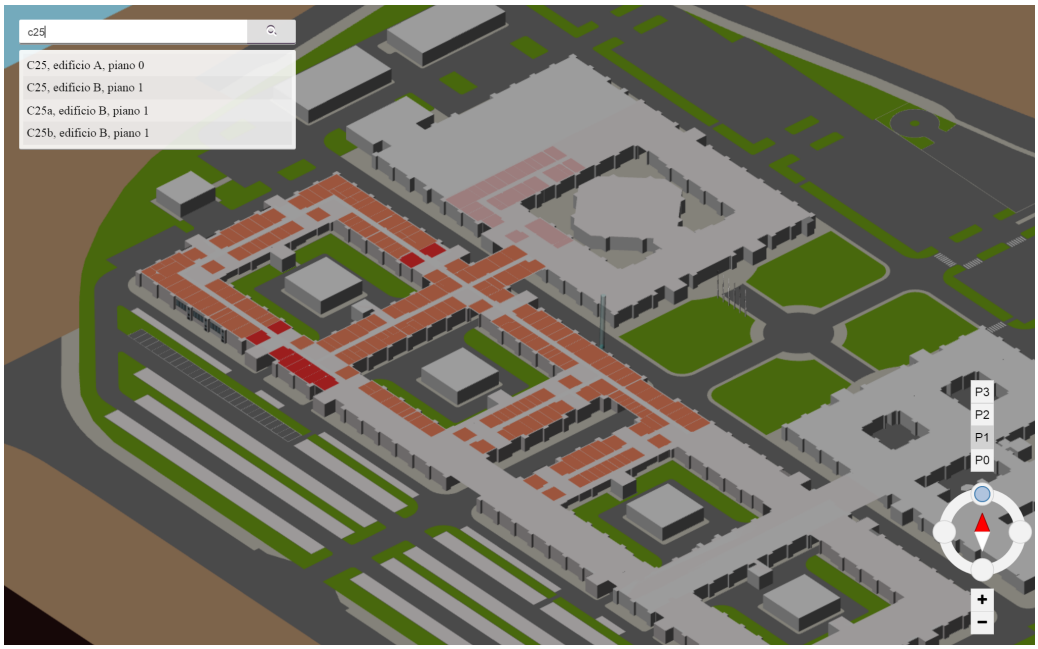


Figure 39: The map application's user interface. The *map* view takes the entire screen space, while other boxes and panels are displayed above it. A *search box* is presented at the top left corner, along with a *result box* below it. A *navigator* panel is displayed at the bottom right corner

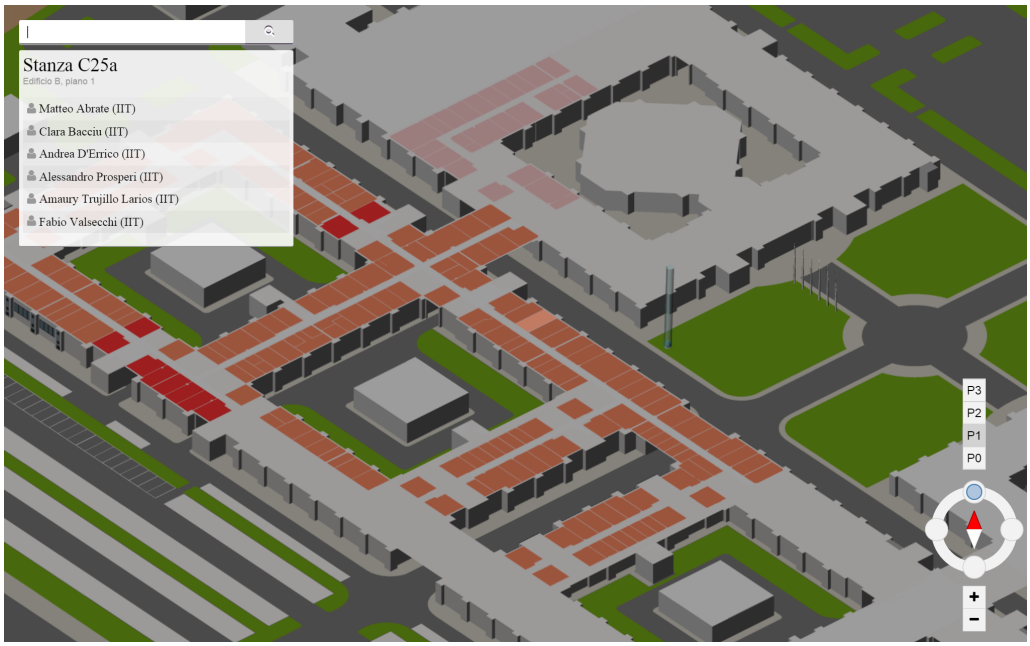


Figure 40: When an object is selected, an *infobox* is shown in place of the result box

The *map* view is the main part of the interface. An isometric projection keeps the presentation simple, while giving a 3D effect that is helpful to include relevant details that make the map easier to remember.
When the application starts, the map shows the whole CNR area, including the parking lots, and a portion of the surrounding roads, to help the user getting oriented. At the beginning, the

map is oriented with the geographic north up, and the buildings are shown complete with the rooftop.
The user can zoom and pan the view using either a mouse or the keyboard, or pen or fingers on touch devices. The rooms inside the building can be selected with a click of the mouse or a tap in order to obtain details about their label and occupants. Clickable items have a size that is suitable for use on touch devices.
A series of commands are available in the navigator panel on the right:

- A **selector for floors**.
  It lets the user select which floor they want to explore, making all the floors above it invisible. At the beginning P03 is selected, so that all the floors, including the rooftop, are shown. If another floor is selected the rooms it contains become visible.
- A **rotation handle**.

It lets the user rotate the view. A compass in the middle always shows the position of the geographic north.

- **Outdoor/indoor positioning buttons.**

The user can choose to have his position traced outdoor or indoor. When one of those functions is active, an avatar, always centered in the view, shows the user's position on the map.

- **Zoom in / zoom out buttons**.

They increase or decrease zoom relative to the center of the view.

Keyboard input can also be used to control the map's position (arrow keys), zoom (+/-) and the displayed floor (number keys).
The user can also search for a room name or for the name of a person using the *search box* on the top left corner. The results are shown in a panel below the box (*result box*). If the user clicks on one of the results, the corresponding room is highlighted and the view is centered to show it. A new panel, called *infobox*, is shown in place of the result box to display a description of the selected room.
In the same panel, the user can click on the arrow on the top right to see the path to be followed get to that room, starting from the entrance of the area or from the current user's position.

5.1.3 Application architecture

The application code is organized in several modules that communicate by sending and reacting to events. Modules can play different roles, such as acting as controllers implementing business logic, storing the application status as models, or presenting user interface elements to the user in views (Figure **41**).

Figure 41: The different roles a module can play, along with the two types of interaction between modules

Modules are also collected in different logical configurations, each one taking care of a specific feature of the application.
The configuration concerning the **navigation** of the map takes care of reacting to user's inputs from the keyboard, the navigator buttons or the map itself. Whenever the user acts, an abstract camera is translated, rotated or brought near to/far from the building. Whenever the camera changes status, the map is updated and asked to render the 3D model again (Figure **42**).

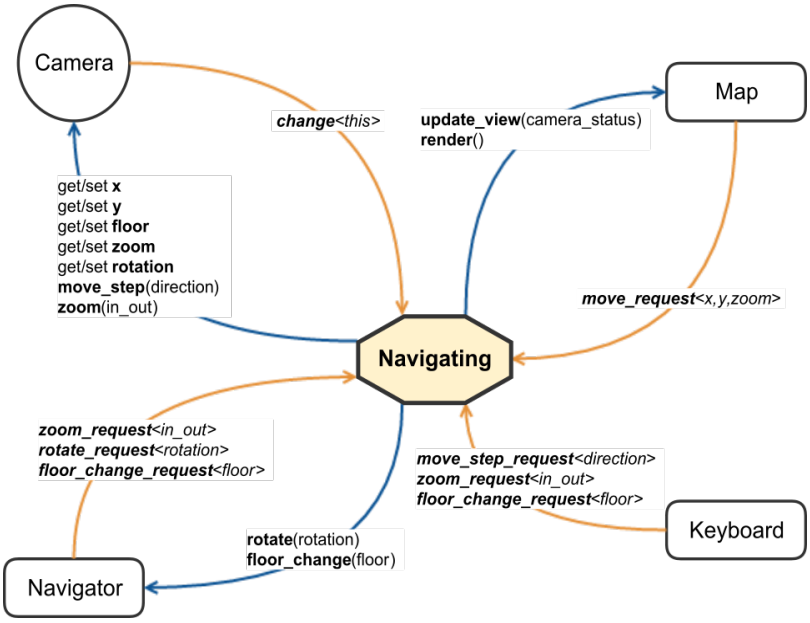


Figure 42: Exchange of events and method invocations between modules, related to the navigation of the map

A second configuration is dedicated to managing the user's ability to **select** an object on the map, which can be for instance a room or a parking spot. The selection action can be initiated either by clicking on the object displayed on the map, or by clicking a reference to the object from the search results displayed in the result box (see below for more details about search). Selecting an object: i) highlights it on the map, ii) causes the camera to center the object in the view, and iii) opens up the infobox describing the object. The selection can be emptied by clicking on the map outside any objects (Figure **43**).

Searching for rooms or people is made possible by a third configuration, which is responsible of reacting to the search box command by issuing a query to a database containing information about rooms and personnel. When ready, the result is made available in the result box, from which the user can select one object (see above) (Figure **44**).

Another configuration of modules obtains and visualizes the estimated **position** of the user on the map. By acting on the navigator buttons, the user can enable or disable both the outdoor and the indoor positioning services. When enabled, a positioning service displays an *avatar* on the map, and instructs the camera to track its movements. The user can disable tracking by manually moving the map, or disable the positioning service by acting again on the navigator buttons. A geographical module is also defined, taking care of translating coordinates between WGS84 and model reference systems (Figure **45**).

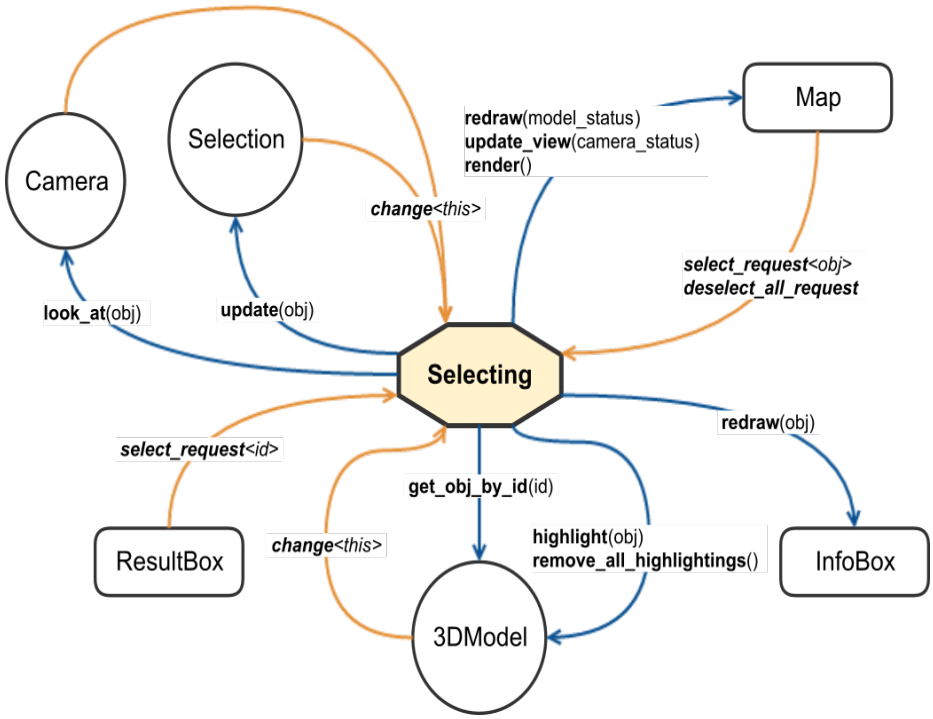


Figure 43: Exchange of events and method invocations between modules, related to the selection of objects

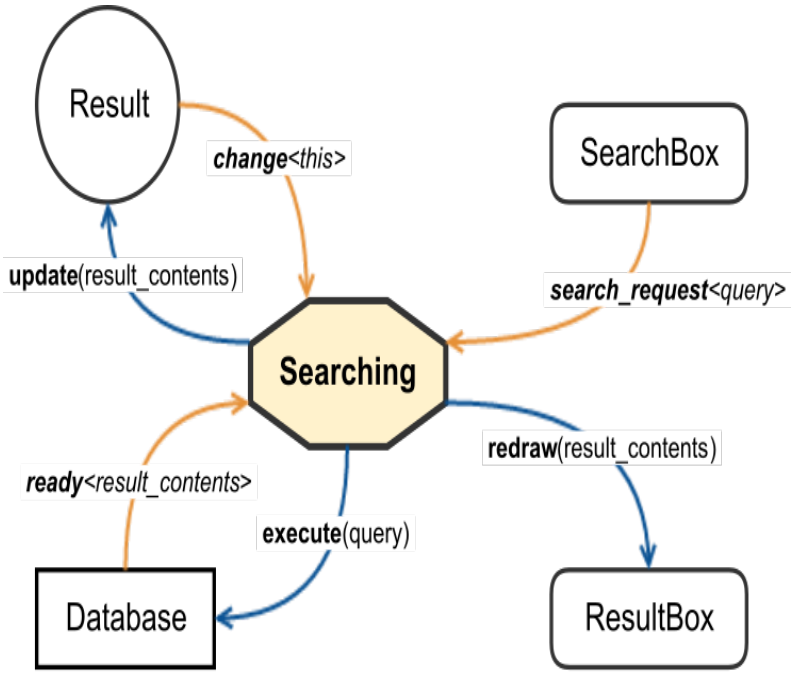


Figure 44: Exchange of events and method invocations between modules, related to the search feature

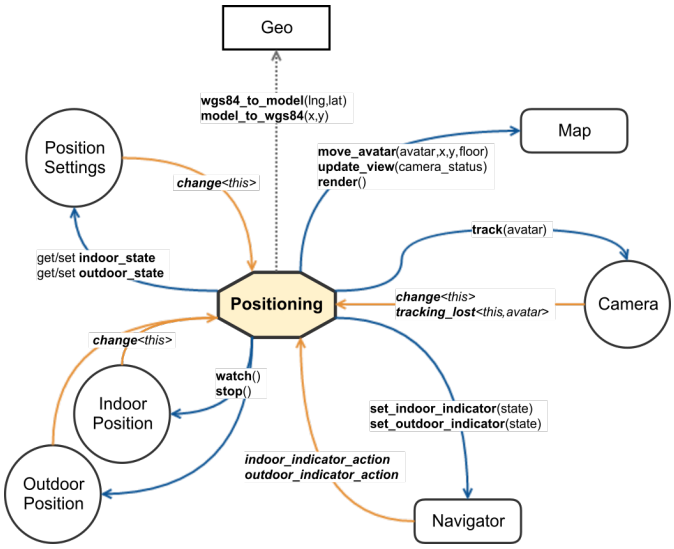


Figure 45: Exchange of events and method invocations between modules, related to the positioning services.

5.1.4 Map embedding

Besides the main application, the map is also available in a stripped-down version for embedding into other web pages. Positioning and searching are unavailable, while navigating and selecting are present as simpler behaviors. The hosting web page should load the embed URL as the source of an <iframe> element, and it can specify navigation or selection parameters in the query string.

5.2 THE INDOOR POSITIONING FUNCTION

(Ref. Francesco Potortì, ISTI)

Another functionality provided by the system is the ability to indicate the user's position inside the buildings. While showing the position outdoors is simple, due to the availability of GPS, the problem of indoor localization is still in its research phase. Several solutions exist, but even the best ones do not provide accurate positioning unless tuned to the relevant environment. In practice, only complex systems using a variety of methods fused together are able to pinpoint the user's position with a precision to the room level.

For example, services like those provided by Google maps, or the Android and OSX library calls usually have positioning errors of 10-50 meters, which in an indoor environment can be very useful (inside a big metro station) or almost useless (inside an office environment). Additionally, for multi-floor buildings, the available general services require the user to indicate the floor.

However, the ability to know one's position is a worthwhile addition to a map service, and even more to a navigation service. In the latter case the functionality indoor is similar to what we are accustomed when using an outdoor street navigator, which is able to show the path to follow along with the current position on the road.

5.2.1 Methods

The most used method for indoor positioning in office environment is based on Wi-Fi fingerprinting. The idea is to exploit the ability of the Wi-Fi receiver of a smartphone to scan

the existing Wi-Fi networks and, for each, measure the RSS, that is, the received signal strength. Since the received power is sensitive to the distance from transmitting AP (access point) and to the intervening obstacles in a generally non-obvious way, each position in the area of interest is likely to exhibit a unique combination (a *vector*) of visible APs and relative RSS. This method needs no infrastructure to be purposely installed, as it is based on the already existing Wi-Fi access points, nor does it require any specialized receiver, as any Wi-Fi receiver, like those present on a smartphone, is able to provide the necessary measurements.
The practical implementation, however, has two main problems, which limit the usefulness of the idea. The first problem of Wi-Fi fingerprinting is structural: it is necessary to provide an initial vector data base for the method to work, which implies an initial, time-consuming survey of the whole area, and a periodic, time-consuming, update of the data base. This problem can be alleviated by the use of collaborative methods, which however add significant complexity to the system.
The second problem of Wi-Fi fingerprinting is algorithmic: RSS readings are affected by random error, seasonal variation, AP infrastructure updates, AP power management, receiver heterogeneity, limited scanning frequency and variable receiver positioning with respect to the user. A vast literature exists on methods to tackle these problems, but no general method exists that is able to be precise enough to identify the position at the room level without external aid, with the best systems giving errors exceeding 10 m in practical situations, and not being reliable on floor identification. The solution to the above problems is to use multiple sources of information, rather than Wi-Fi scanning only. The information is *fused* by using one of several algorithms, like pure Bayesian systems, non-trivial variations on Kalman filters or particle filters.
The main additional source of information is a map of the building, which is often used with particle filters. It prevents errors like paths crossing a wall, a ceiling, or the positions falling outside of the building.

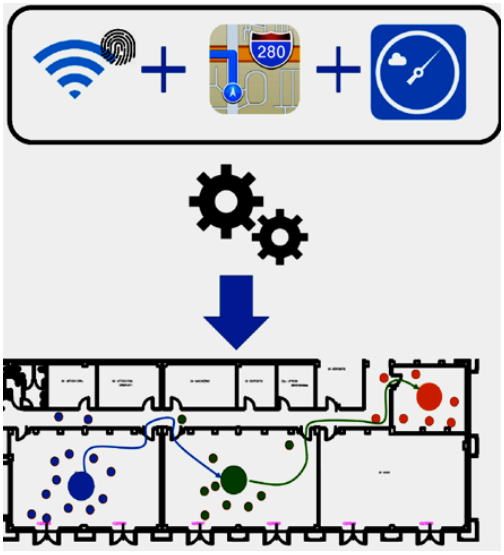


Figure 46: Four sources of information and three position estimates

In Figure **46**, four sources of information are depicted on top: (from the left) Wi-Fi, fingerprinting, map of the area, barometer. They feed a fusion system based on particle filtering. The map below shows three position estimates in three consecutive moments, in blue,

green and orange. Each estimate is computed as the center of a cloud of particles that move randomly, attracted by the information provided by the three sources.

Using a particle filter allows easy addition of more sources of information, like those provided by barometers, which with specific filtering functions can greatly reduce the problem of floor identification, even if they are currently only available on high-end smartphones.
One more source of information comes from sophisticated filters based on inertial sensors (accelerometers and gyroscopes) which, when properly trained, are able to give hints about the user's movement speed. The use of a magnetic sensor (compass) may add the ability to identify the movement direction. This line of research is very promising but highly specialized, and not yet ready for non-experimental usage.
Instead of the very general and very powerful fusion provided by particle filters, one interesting possibility in the case of navigation systems is to start from the already available navigation graph. The advantage is that the graph has knowledge of the points of interest, contains information on the path followed and the expected future movements, and is much more compact and easy to use than a map. Moreover, fusion using a graph only requires a time-varying pre-built transition matrix rather than the high-intensity processing power needed by particle filters (Figure **46**).

5.2.2 Implementation

The positioning service is implemented as a public server, accessible over the Internet with a REST interface, independent of the navigation service. The server hosts the fingerprinting data base, the maps and the algorithms that compute the expected position given the Wi-Fi scans provided by clients.
Clients (the user smartphones) access the server by sending to it a Wi-Fi scan. After the first scan is sent, a client can ask the server for position estimates. More scans help the server to refine and update the estimates that are sent upon further request to the client.
In practice, the smartphone running the smart navigation application also runs a background application that continuously performs a Wi-Fi scan and sends it to the server. The smart navigation application, on the other hand, periodically queries the server to get a position estimate, and shows it on the map as a red dot. This architecture completely separates the smart navigation application from the positioning service, which thus is viewed as an optional feature and can be developed and updated independently of the smart navigation application.

5.2.3 Current status and perspective

Currently, the server is up and running, using a fingerprint database limited to the area used for the demo. The fingerprint database is filtered by semi-automated methods to select only the most significant APs. The implemented algorithm is a simple k-NN, and no fusion is performed with other information sources. Given the small size of the database, the performance is good: approximately the error is below 5 m 75% of the time.
The next steps are selecting and implementing a method for a completely automatic selection and update of the APs in the data base, enlarging the data base with fingerprint measurements of the whole ISTI and IIT areas, possibly replacing the k-NN algorithm with a stochastic one**,** and choosing a fusion method between particle filtering working on maps or transition matrix based on navigation graphs. Adding the ability to work with barometer data for floor detection would be a worthwhile addition.

6 THE SMART SHARED MOBILITY (Ref. Franca Delmastro, IIT)

In brief: *The application of Smart Shared Mobility allows visitors and employees of the CNR to share a trip to or from the CNR area according to particular interests and habits. The application, in the form both web and mobile, provides incentives and customized recommendations to improve the travel and the sharing experience, thus also getting better the mobility to and from the CNR area.*

Nowadays there is a lot of car pooling and sharing solutions developed to improve urban mobility conditions, reduce the impact of traffic on air quality and improve the use of parking systems. Customised solutions have been proposed for different target users and for different use. BlaBlaCar [1] is the most used solution for long distance trips with million of users, allowing them to share the cost of the trip. JoJob [2] is one of the first solutions designed explicitly for the industrial community. Employees of the same company can use it to share their commuting trip, receiving an appropriate reward as an incentive. CarmaPool [3] enables its users to find nearby people with which they can share their trip. Other solutions have been developed for car sharing, and specifically for taxi sharing, especially for big cities. Uber [4] and CabWithMe [5] are a couple of examples. A lot of other solutions are currently available on mobile app stores, all with similar features. In the framework of the Smart Area project we decide to propose a new ride sharing system since the mobility in area represented a big issue in the last few years, seeing the increasing number of visits to the area and the limited parking area. In addition, the potential user community presents common behavior that could be facilitated by this type of solutions.

Smart Shared Mobility (SSM) has been designed explicitly for CNR users (both visitors and employees) and the main idea is to provide users with a simple application *to share trips to and from CNR Area in Pisa (but not only), customized on their preferences and requirem*ents. With respect to the other available solutions, SSM has several advantages. It provides personalized suggestions both after an explicit user request (i.e., search operation) but also based on a proactive feature of the system, which selects the most appropriate solution based on the user history in the app usage and on reminder features (set in case the current solutions found by the systems do not satisfy the user request). In addition, SSM is completely integrated with the Cloud of the Smart Area, through which it is able to access data of external services (like Smart parking) to provide additional features, and to push its own data to the Cloud, in order to enrich the entire system.

The system has great potentialities also from the research point of view. In fact, it can provide a detailed analysis of the mobility patterns to and from the area, in addition to the definition of *new social communities*, not only related to the sharing mobility features, but also to the use of a set of technological solutions that can be integrated through the cloud (e.g., users that exploit the sharing mobility and maintain a low profile in the energy consumption of their office). In this way we could study behavioral models both on the single solutions and on their possible integrations, opening the way to the definition of new personalized solutions.

In this document we present the technical description of Smart Shared Mobility applications and its architectural design.

6.1 THE SYSTEM ARCHITECTURE

The system is characterized by a typical centralized architecture in which we can identify three main components: (i) the clients (i.e., personal computers and smartphones/tablets), with which users can access the service, (ii) the web service, which process the clients' requests and provides the required information, and (iii) the spatial database in which are stored the data about users and trips (Figure **47**).

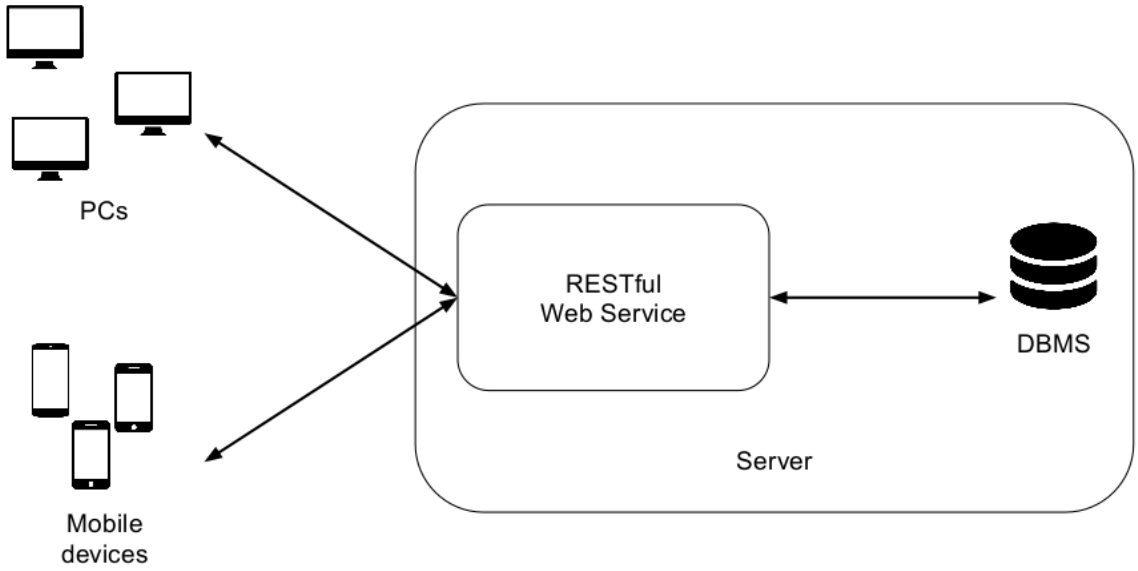


Figure 47: The system's main components

The web service represents the main component of the entire architecture. It was created according to the *Representational State Transfer (REST) [8]* architectural pattern. By exploiting the standard HTTP methods (e.g., GET, PUT, POST or DELETE), the web service exposes a set of web APIs to allow clients to perform the various operations (e.g., to offer new rides to be shared with other users, to find potentially interesting rides, to send messages, to leave feedbacks to drivers, etc..). To perform geographical and spatial queries about the trips, the web service uses a spatial and geographic object-relational database, also needed to store the resources offered to the clients (e.g., rides, messages, users, etc..).

6.1.1 Server side

This Section describes the definition of the backend features used to implement the main functionalities of the Smart Shared Mobility (SSM) service.

Specifically, as far as the data management is concerned, the system defines a trip as mainly characterized by geographic information such as: a starting point *A*, a destination point *B* and the route chosen by the driver to go from *A* to *B*. Each geographic point is composed of two coordinates (i.e., latitude and longitude) and the route is nothing more than a list of geographic

points where the first one is the start point *A*, and the last one is the destination point, *B*. In order to manage this kind of information and perform queries on them, we decided to exploit the potentialities of *Geographic Information System (GIS) [1]* combined with relational databases. Specifically, we decided to use a specific *Data Base Management System (DBMS)* such as the geographical and spatial extension of *PostgreSQL*[15], *PostGIS*[16].

6.1.1.1 Searching a ride

Since this particular version of SSM is designed for employees and visitors of the CNR area of Pisa, we defined that the starting point or the destination point of each trip is limited to the CNR area. This constraint is also applied when a user searches for useful rides, and each query is characterized by the following parameters:

- $\mathbf{q_{start}}$: the point at which the user requests to the driver to be picked up
- $\mathbf{q_{dest}}$: the point that the user wants to reach
- $\mathbf{q_{st}}$: the time in which the user wants to start the trip
- $\mathbf{m_{wd}}$: the maximum distance that the user is willing to travel on foot to reach the driver or her destination point
- $\mathbf{w_t}$: the maximum time that the user is willing to wait to meet the driver.

When a search is performed, the database runs two different queries depending on the fact that CNR is the requested starting or destination point. Figure **48** depicts the first case: the selected rides are all those whose departure time is between q_{st} and $q_{st} + w_t$, and whose distance between the route and q_{dest} (destination point of the searching user) is not greater than m_{wd}.

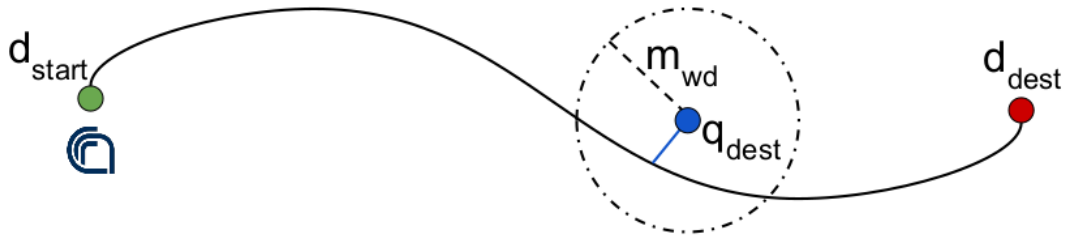


Figure 48: An example ride from the CNR.

Otherwise, if the CNR is the destination point, the service selects all those trips whose distance between the route and q_{dest} is not greater than the specified m_{wd}, and those for which the driver will be at the intersection point i_p (i.e., the end of the shortest path between the

15 http://www.postgresql.org

16 http://postgis.net

requested q_{start} and the trip's route) at the departure time q_{st}. This case is depicted in Figure *49*.

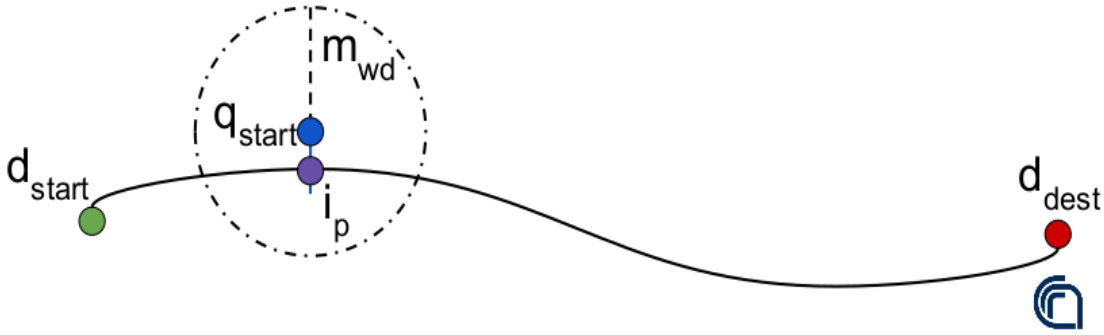


Figure 49: An example ride to the CNR

6.1.1.2 The web service

The web service implements the application logic of the entire system and it was built with *Jersey*[17], a popular open source framework for RESTful Web services.

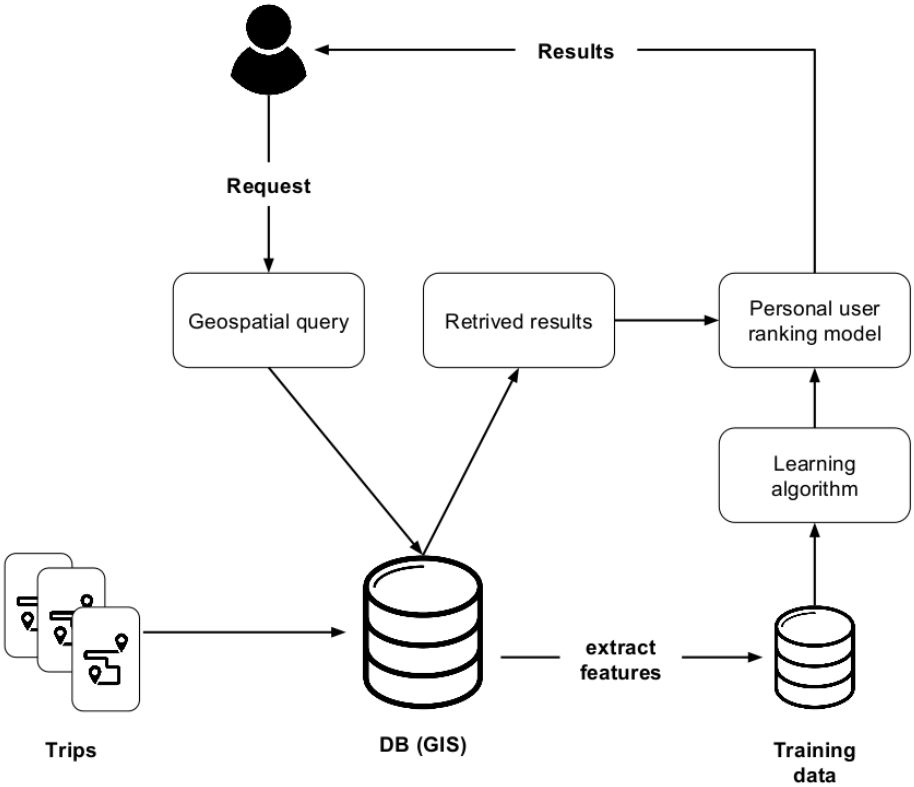


Figure 50: The Learning to Rank architecture implemented by the web service

17 https://jersey.java.net

All the communications between clients and web service are performed with standard HTTP requests and the exchanged data are encoded using the *JSON*[18] format. Through these technologies the web service is able to serve different kinds of clients, with different hardware and operating systems.

The web service's features can be grouped into two main categories, proactive and reactive, depending on the type of interaction of the final users with the application. The first, concerning the system's ability to autonomously propose new rides to the users, without any explicit request. The recommended rides are selected by the system based on the users' preferences and requirements expressed in the past during their search and offer operations. Instead, the reactive features concern the ability of the system to quickly react to an explicit request of a user (e.g. search available rides, get users' feedbacks, etc...). In Figure **50** we describe the service architecture used to learn how to rank the results of a user's search of available rides, based on the user's preferences and her past interactions with the system.

6.2 THE GoTogether MOBILE APPLICATION

Since all the car pooling/sharing system today have a mobile app, developed for Android or IOS (or both in some cases), we decided to develop a dedicated app for the SSM and to define a more incisive name. We called it GoTogether. The app is developed for Android OS and it is available through the PlayStore:

https://play.google.com/store/apps/details?id=it.cnr.iit.smartmobility.

In addition, the application has been also developed as a web application linked by the Smart Area web page (*http://www.smart-applications.area.pi.cnr.it*). Details on the web version are explained in Section 6.3.

Since the application requires the user identification and the possible specification of some user preferences, the app starts with a login/registration page and the definition of the user profile (for details on the privacy terms and conditions for the management of the user personal data, please see Section 6.4). The registration page is the same for the mobile and web apps. Everybody can register to the application, both CNR employees and visitors. Depending on the user role, different information is requested in the personal profile.
In the following, we describe the main features of the mobile app as a tutorial, to briefly explain how to interact with the app and what a user can expect from it.

6.2.1 User Interface (UI)

The user interface has been developed following the *material design* guidelines provided by Google. These guidelines define graphical elements and their role, to create a consistent UI that can seamlessly integrated with other application, providing a better user experience on the Android platform.

18 http://www.json.org

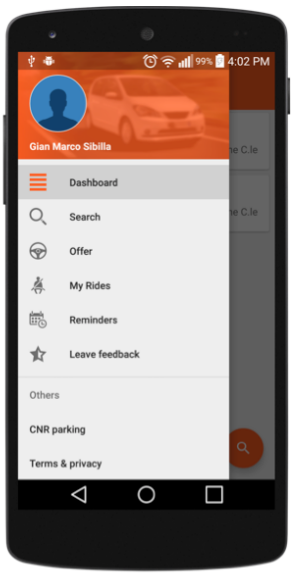
Elements

Below are reported the main material design elements used in the UI of the application.

- **Toolbars:** group the current view title and actions in a ribbon placed at the top of the screen and above any other component.
- **Tabs:** provides an easy way to switch between different categorized contents hosted in the same view.
- **Cards:** wraps unique related data.
- **Snackbars:** provides lightweight feedback about an operation by showing a brief message at the bottom of the screen, containing an optional action.
- **Bottom sheets:** sheet that slides up from the bottom edge of the screen after a user action.
- **Floating Action Buttons** or **FAB**: buttons usually placed in the bottom right corner of the screen, identifying the main action of the current view.

Navigation

The main navigation has been implemented through *navigation drawer*, a sliding panel (accessible from the left edge of the screen) displaying navigation options (Figure **51**). The drawer is displayed in the main *activity*, which loads different Fragments when the user makes a selection.



.

Figure 51: The Navigation drawer

Dashboard

Dashboard is the first view displayed when the application is opened (Figure **52**). It contains some recap information on how the user should use the app and the current situation of the monitored parking area in the CNR. This data is obtained from the Smart Parking web service and consists in the total number of free parking slots, including those for disable people.

The user information consists of:

- the next upcoming trip (if any), either offered of accepted by the user;
- user reputation;

- available and total parking coins obtained by offering trips;
- total km the user travelled through the app.

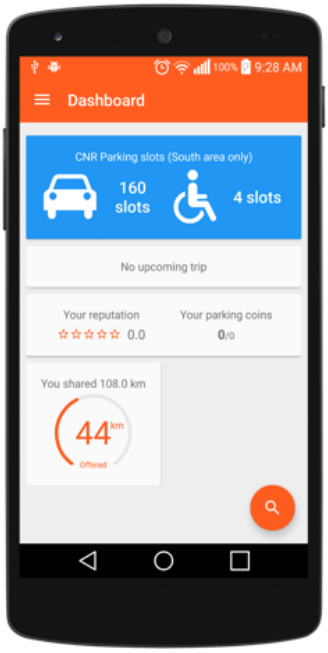


Figure 52: The Dashboard

Searching and Accepting a ride

In the *Search* view the user can search for a trip by setting the starting point or the destination (the other end of the travel has to be the CNR), the starting date and time, the maximum acceptable walking distance and waiting time. The activity to choose an address provides a list of fixed places (including the user home address if set in the user profile) to speed up the query. In any case, the address is auto-completed from Google places.

After all these parameters are set the query is automatically submitted and the results are shown to the user. The two categories, “smart result” and “default results” show the most relevant results for the user, in terms of search criteria and current user’s preferences, and the most relevant results according to the default user profile inserted by the user, respectively. In case no result is available, the application allows the user to set a reminder for future available trip matching the search.

Once the user selects a result entry, the “ride's details” activity is launched showing trip details (Figure **53**). This view contains a map displaying ride's path, starting and destination point and the intersection point between the user and the driver. Once the user scrolls the view -- either by dragging it or using the FAB -- she can access details on available seats and driver's preferences on the trip as well as the button to reserve a seat. By pressing the “accept this ride" button, the user is added as a passenger of the trip and a notification is sent to the driver.

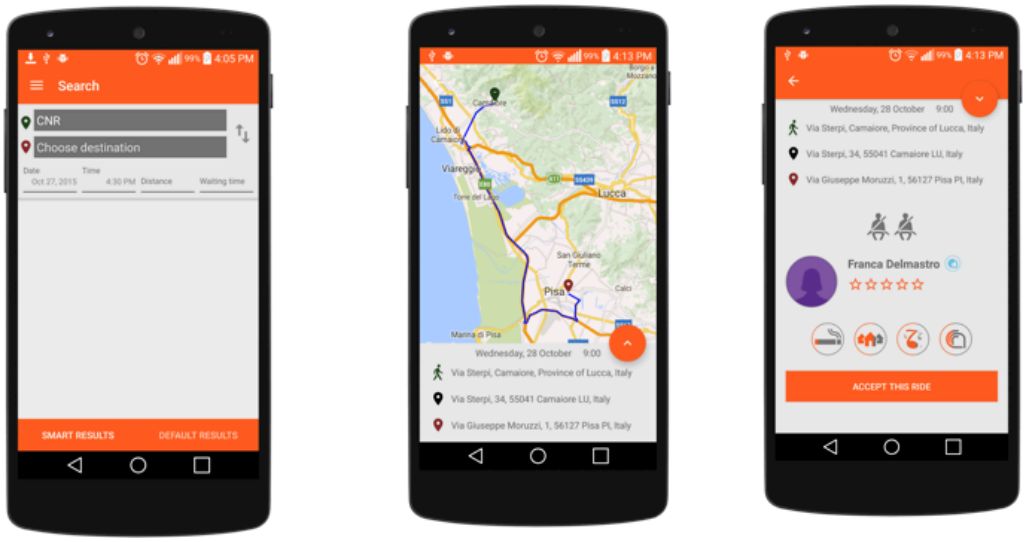


Figure 53: Search for a ride and ride details

Offer a ride

By the "New trip" activity it is possible to insert an offer for a new trip specifying the starting point, the destination, the starting time and the available seats (Figure **54**).

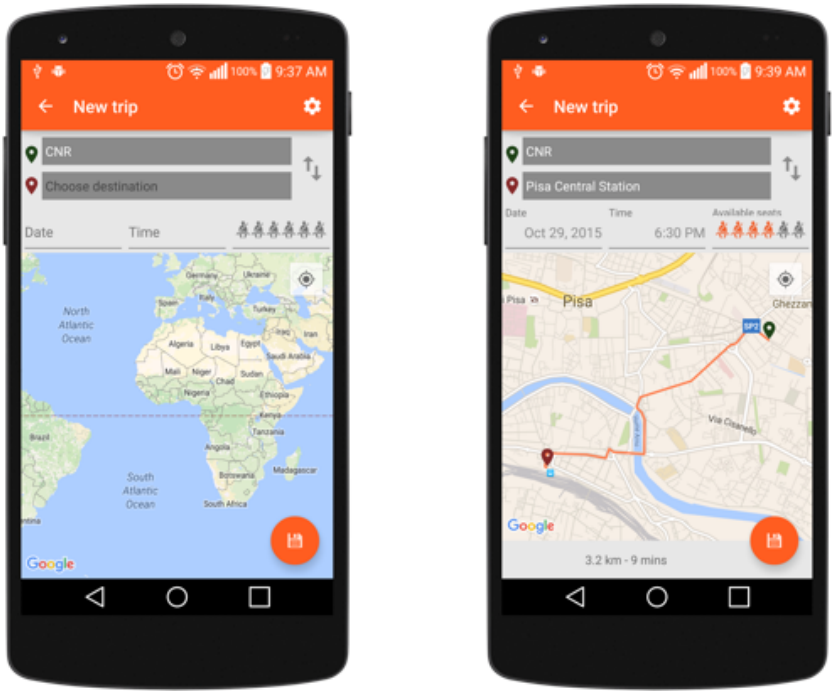


Figure 54: Ride offer details

The address selection procedure is the same explained for *search view*, while the date picker is different, since it allows the selection of multiple days as a periodic trip. Once saved, the trip is immediately available in other users search results.

My rides

The “My Rides" view lists all the user rides, categorized in trips for which the user is a passenger and those in which the user is the driver (Figure **55**).
By clicking on a list item, an activity is launched showing trip details (i.e., end points info, driver and passengers list). For accepted trips the intersection point between the user and the driver is added in the points list (if different from the starting point), while an entry with the intersection point is added for each passenger for the offered trips. By this activity it is possible to open the ride's chat room, i.e. a conversation between the ride passengers and the driver (Figure **56**). When a user sends a message, it's immediately delivered to all the attendees with a push notification

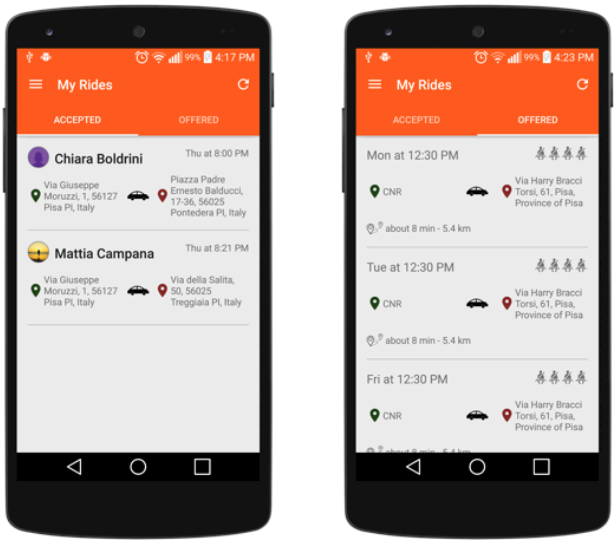


Figure 55: My rides section (accepted and offered)

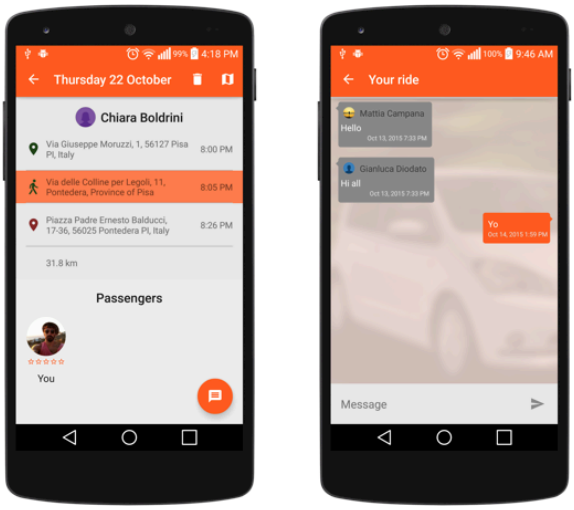


Figure 56: Ride's details and the ride's chat room

Reminders

Reminder consists in a *future query* that the system tries to fulfil upon new trip insertion (Figure **57**). A user can set a reminder after a search query, whether it produced a result or not. A list of active reminder can be accessed from the relative entry in the main navigation menu; if the user is no more interested in a reminder, he can swipe it away from the list to dismiss it.

Feedback

User reputation inside the app is computed through *feedback*, that passengers have to leave to the driver after a trip (Figure **57**). From the navigation drawer the user can access the "Leave feedback" view, which lists all the feedbacks the user has to leave for attended trips. To leave a feedback, the user must fill in a form that is composed of a compulsory score (from 1 to 5) and an optional text to describe the driver performances.

User profile

By clicking in the header view of the navigation drawer a user can access her personal profile (Figure **57**). In this view, all the information the user entered during the registration process are displayed, along with a short recap of the last feedback received and travel preferences. Preferences can be edited on-the-fly, while to enable editing of other profile information or the profile picture the user must press the edit FAB. The same view is shown, in a *read-only* mode, whenever the users asks for details of another user -- e.g. when a user wants to know more information about the driver of an offered trip.

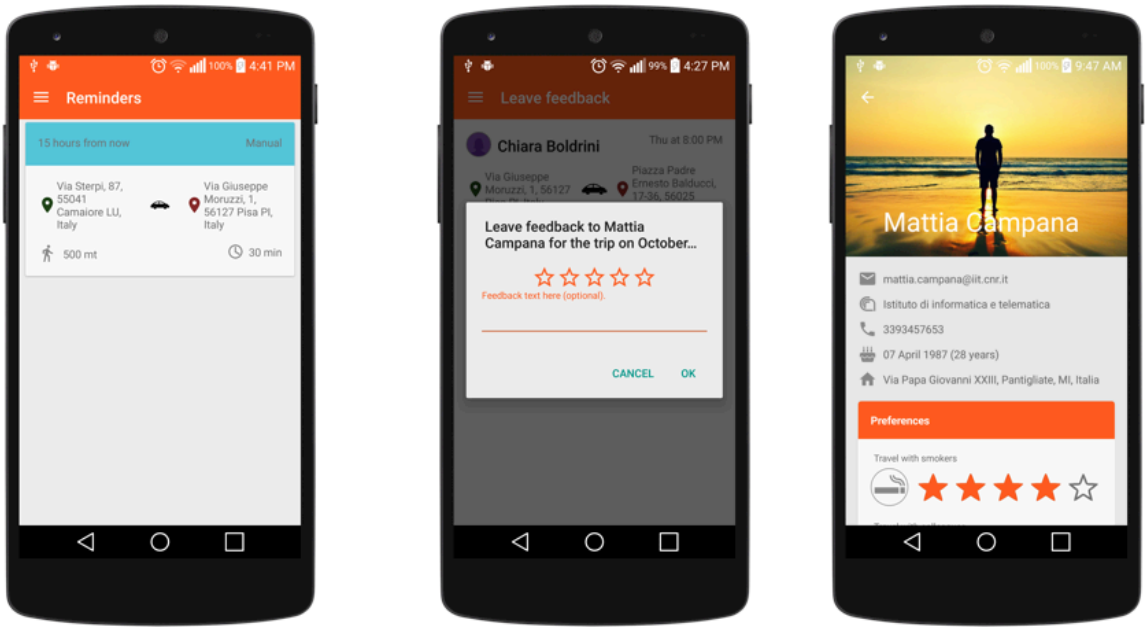


Figure 57: Reminders, Feedbacks, and User profile

6.3 THE GoTogether WEB APPLICATION

The application has been initially developed as a web app linked by the Smart Area web page. The first interaction with the application is related to the user registration and the definition of the personal profile (Figure **58**). Depending on the user role, different information is requested in the personal profile.

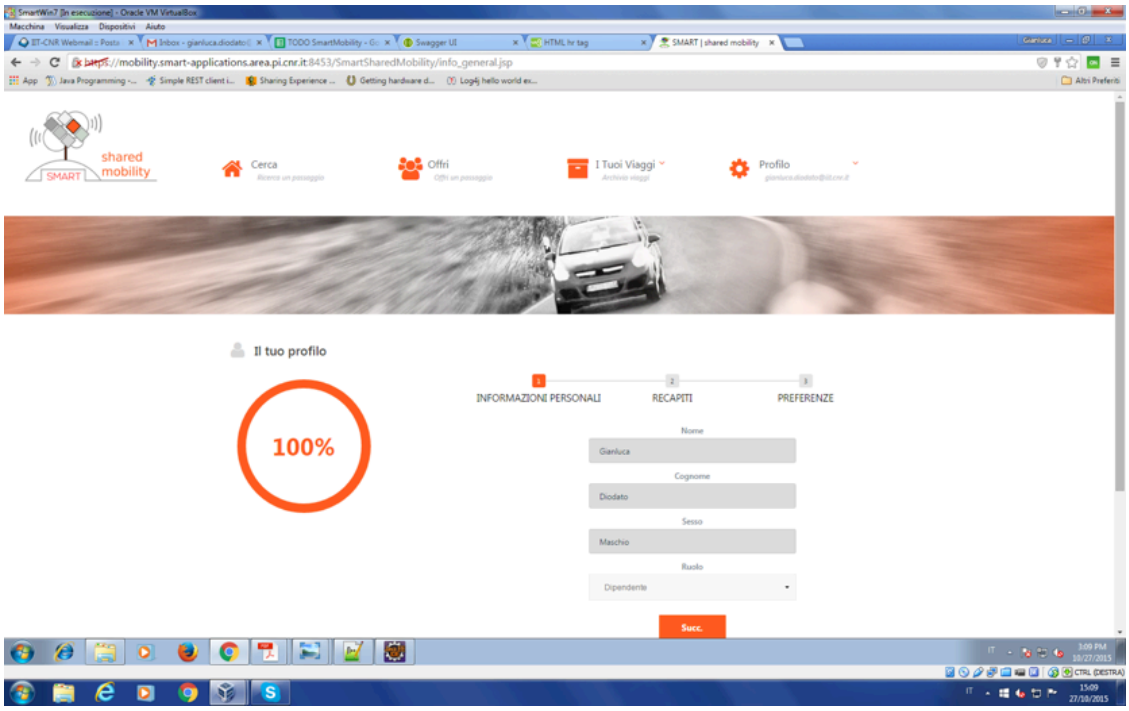


Figure 58: Web App. The user profile

The user profile specifies both personal information of the user and the user preferences used to customize the user experience while sharing a ride (e.g., travel with colleagues and/or with neighbours, preference for non-smoking passengers, music etc.).

Once the profile is complete and saved into the system, the user can start offering or searching a ride. Figure 59 shows the user interface for a ride offer. It is a simple interface asking the user to select the starting and ending point of the trip, number of available seats and a calendar to set the date or the range of dates in case of a periodic offer. Since CNR often hosts technical and public events, we decided to add the possibility of associating a trip with a specific event. In the future, the event information will be directly got from the CNR Area web page, so that they will be constantly updated. For the moment, event information is optional and added by the user. In addition, the user can specify the preference for the specific offer, otherwise the preferences of his personal profile will be associated to the ride.

The map displayed on the right side of Figure 59 shows the trip selected by Google maps. The user can modify the path by moving the trip line directly on the map. When the trip selection is finalized, the user must save the trip through the button on the bottom-left right.

Figure 60 shows the results of a ride search. On the left side, the panel is similar to the offer page, specifying starting and ending point of the trip, and adding the tolerance parameters selected by the searching user in terms of maximum delay and walking distance with respect to those originally selected. Then, on the right side, the app shows a list of trips as the results of the search query executed on the server. As in the mobile app, here the system provides to the user two sets of suggestions: “smart results” and “default results'”. They show the most relevant results for the user, in terms of search criteria and current user’s preferences, and the most relevant results according to the default user profile, respectively.

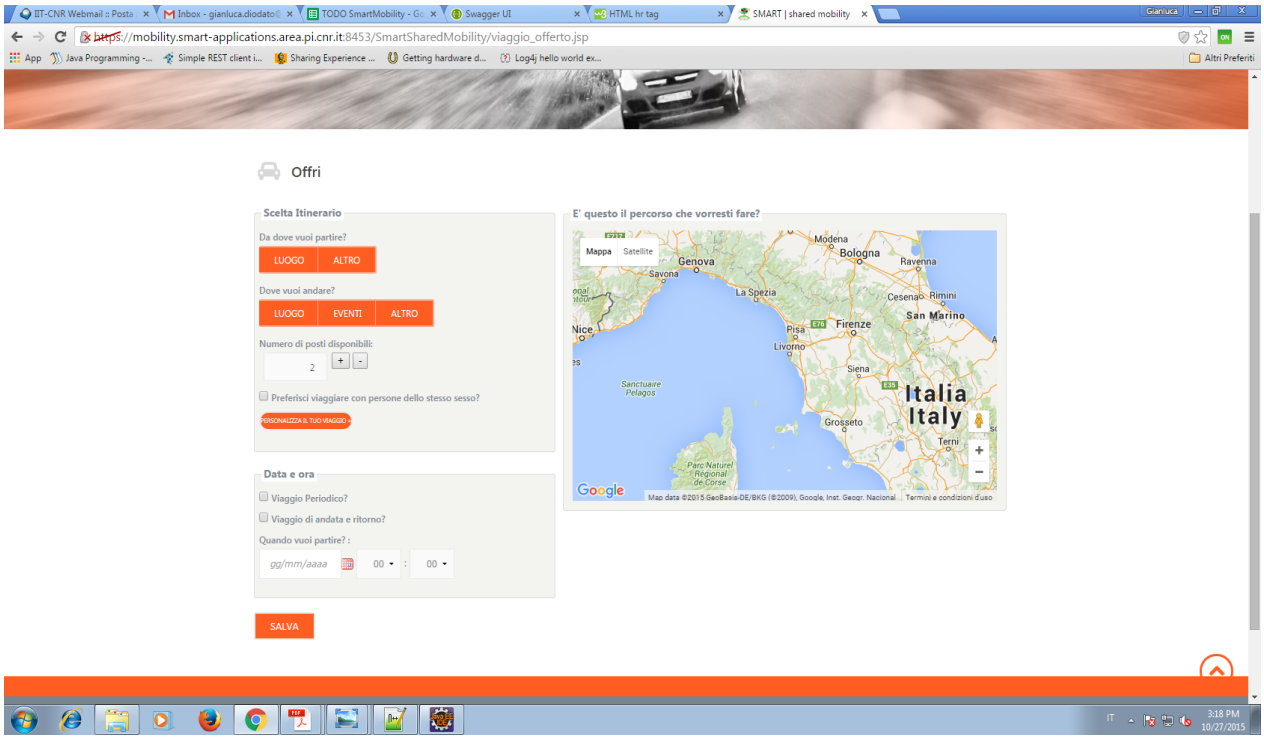


Figure 59: A ride offer

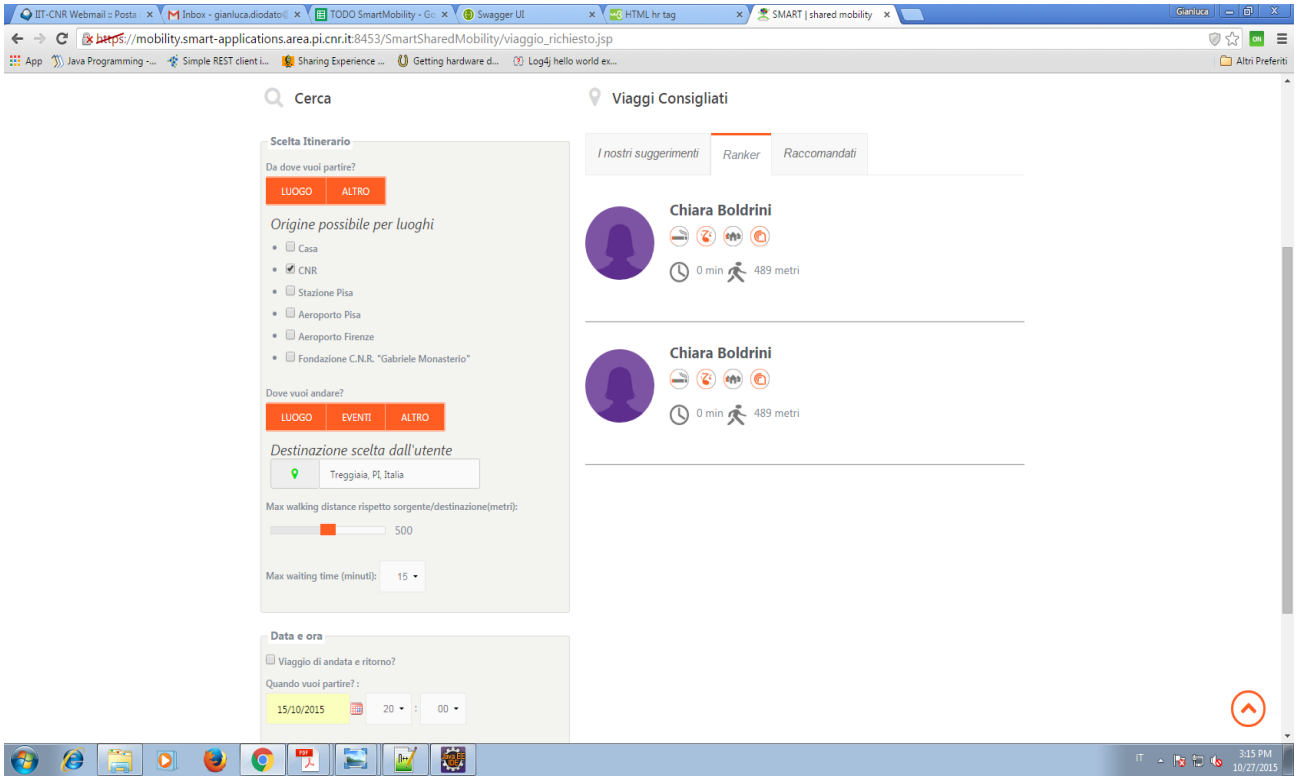


Figure 60: A ride search

Finally, the web app presents a summary of the accepted and offered trips in a dedicated page, specifying on the left side the evaluation of the local user as a driver and his preferences (Figure 61).

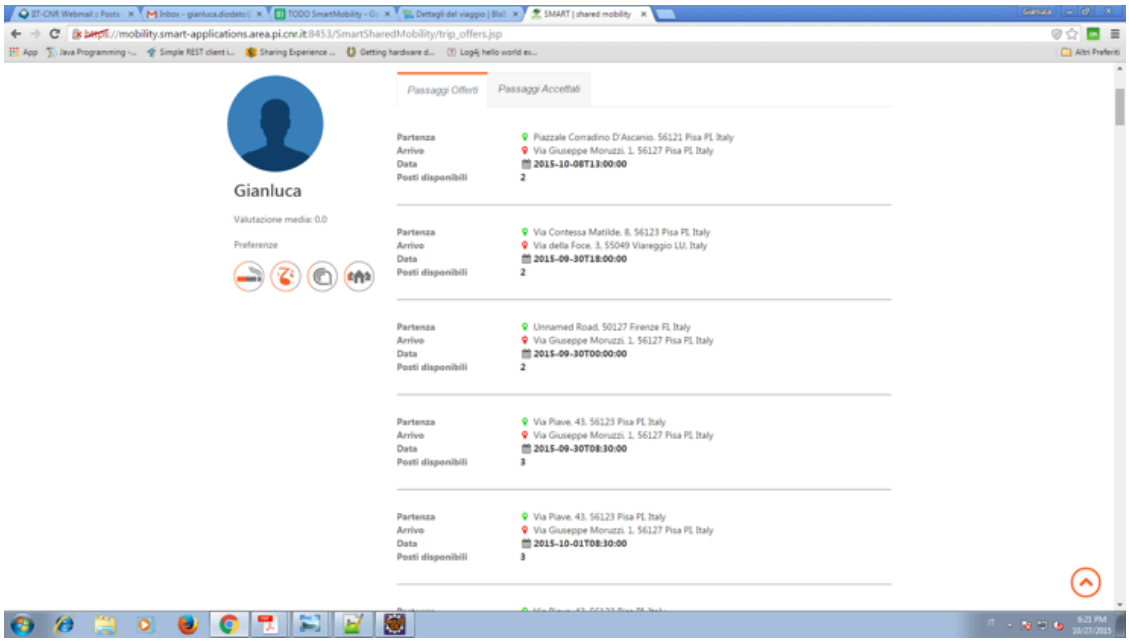


Figure 61: Summary of trips

6.4 PRIVACY TERMS AND CONDITIONS

Since the Smart Shared Mobility applications collects several information characterising the user, including localization information and movement habits, we defined an appropriate module describing Privacy terms and conditions of the apps. It derives from the general form used for Smart Area web site and the other apps, but we specified in detail the two types of data managed by the apps: (i) navigation data and (ii) personal information. The collection of these data does not require the authorisation of the privacy guardant, but it is essential that the user know the types and finality of the collected data. The correct functionalities of the apps are based on the use of such data, and their lack can cause damage in the apps functionality. The user is requested to agree to these terms before launching the apps. Both the mobile and web apps include a link to the detailed instructions related to privacy issues and management.

6.5 REFERENCES

[1] www.blablacar.it
[2] www.jojob.it
[3] https://carmacarpool.com/
[4] https://www.uber.com/

[5] http://www.welcome.cabwith.me/
[6] Michael F Goodchild. Geographic information system. In Encyclopedia of Database Systems, pages 1231–1236. Springer, 2009.
[7] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. Information Retrieval, 16(1):63–90, 2013.
[8] Michael Jakl. Representational state transfer, 2005.

7 THE SMART AREA PLATFORM

(Ref. Carlo Meghini, ISTI)

In brief: *The Smart Area Platform (SAP) is a hardware and software architecture that supports the Smart Area applications. SAP can be seen as consisting of two main parts: the Smart Cloud (SC) system, which offers computational resources managed as a cloud, and the Smart Open Data (SOD) application, which offers storage and access services on the data produced by the applications of the Smart Area (SA) project.*

In the following, the main features of the Smart Cloud (SC) and the Smart Open Data (SOD) applications, which are both under development, are illustrated.

7.1 THE SMART CLOUD

The Smart Area project infrastructure is an OpenStack - Juno version - based IaaSi[19] cloud architecture. OpenStack was chosen as cloud middleware because it is an Open Source, widely adopted and developed project, with strong support by the users community and ICT industry.

The OpenStack software architecture is shown in Figure **62**. All of the shown components have been fully deployed, with the exception of the Object Store.

Before the actual infrastructure deployment, a set of services has been set up to assist the installation and management of the cloud nodes. A machine offering PXE[20], TFTP[21], DHCP[22], VPN[23], NTP[24], Proxy[25] and NFS[26] services was the first to be configured. The whole infrastructure has been deployed in a closed environment, without direct Internet access.
Most of the functionalities of the OpenStack framework have been split on separate nodes and deployed as Virtual Machines, on a server running XenServer as Hypervisor[27]. The actual list of the virtualized nodes is composed by:

---

[19] IaaS: "Infrastructure as a Service", a cloud service model.

[20] PXE: "Preboot eXecution Environment", a standardized client-server environment that boots a software retrieved from a network.
[21] TFTP: "Trivial File Transfer Protocol", allows a client to get from or put a file onto a remote host. Used in PXE enabled environments.
[22] DHCP: "Dynamic Host Configuration Protocol", a standardized network protocol used on Internet Protocol networks for dynamically distributing network configuration parameters.
[23] VPN: "Virtual Private Network" extends a private network across a public network, such as the Internet.

[24] NTP: "Network Time Protocol" is a networking protocol for clock synchronization between computer systems.
[25] A proxy server is a server that acts as an intermediary for requests from clients seeking resources from other servers.
[26] NFS: "Network File System", a distributed file system protocol allowing a user on a client computer to access files over a network much like local storage is accessed.
[27] Hypervisor: a piece of computer software, firmware or hardware that creates and runs virtual machines.

- a cloud Controller node, running the SQL database service, the Message Queue, the Identity service, the Compute and Block Storage management services.
- a Network node, running Neutron[28] with Open vSwitch/GRE configuration in ML2 plugin, offering services like DHCP for tenants networks, metadata server, router, vpn, firewall.
- an Image Service node, running Glance[29].
- a Dashboard node, running Horizon[30] for cloud services management using a web GUI.

The compute and storage nodes have been deployed on physical servers, consisting of:

- 4 compute servers running XenServer hypervisors, achieving together 192 VCPU and 384 GB of RAM. They mount shared storage, allowing live migration of instances.
- 2 block storage servers hosting virtual disks. One of them is used as Root disks storage, offering 3.6 TB of shared NFS space. The other one is used as Ephemeral volumes storage space with 14.3 TB offered as ISCSI[31] targets. They are also used as cloud images repository by Image Service node.

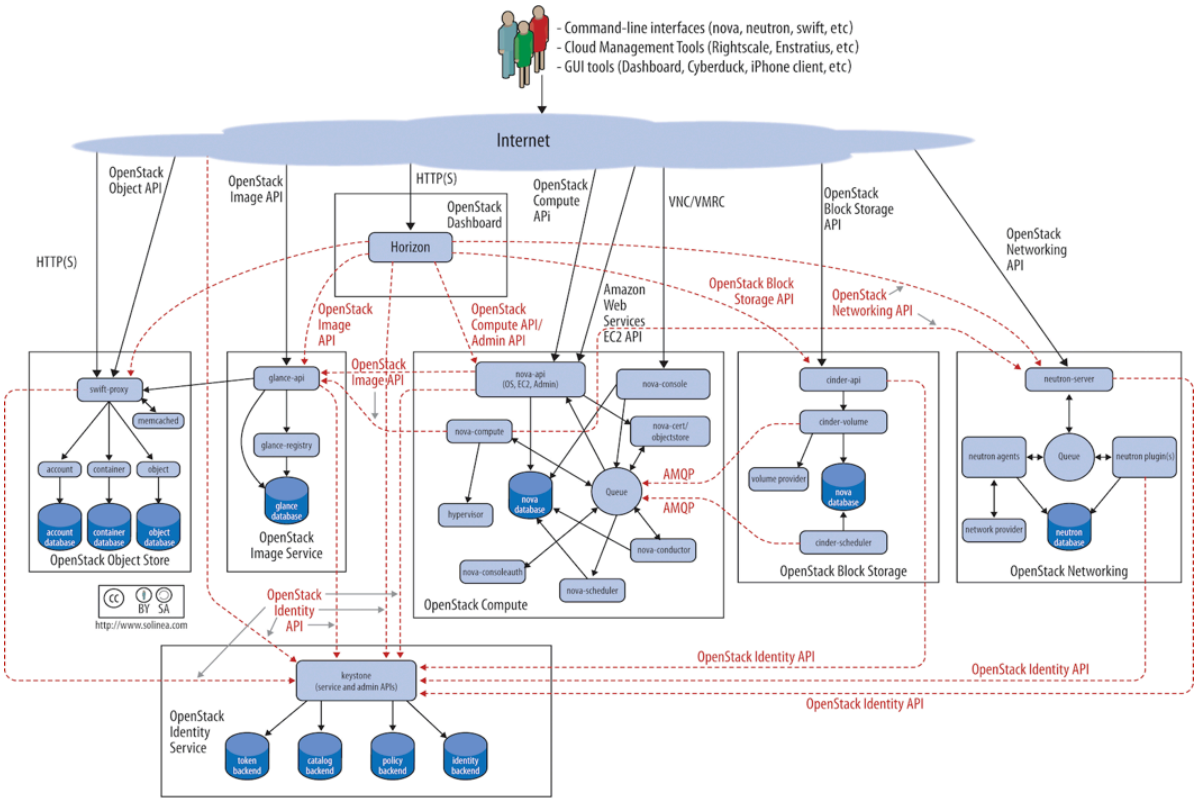
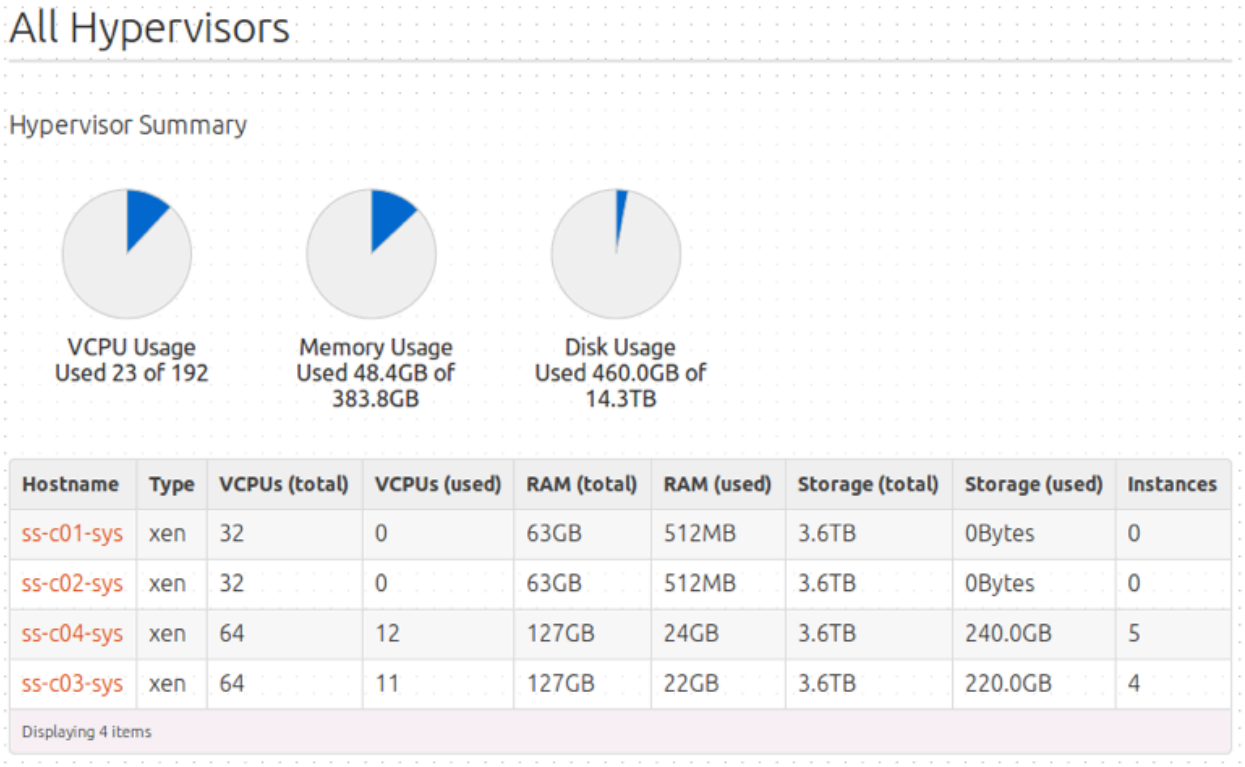


Figure 62: The Open-stack architecture

[28] Neutron, a software part of the OpenStack platform.
[29] Glance, a software part of the OpenStack platform.
[30] Horizon, a software part of the OpenStack platform
[31] ISCSI: "Internet Small Computer System Interface", an Internet Protocol based storage networking standard for linking data storage facilities.



Hostname	Type	VCPUs (total)	VCPUs (used)	RAM (total)	RAM (used)	Storage (total)	Storage (used)	Instances
ss-c01-sys	xen	32	0	63GB	512MB	3.6TB	0Bytes	0
ss-c02-sys	xen	32	0	63GB	512MB	3.6TB	0Bytes	0
ss-c04-sys	xen	64	12	127GB	24GB	3.6TB	240.0GB	5
ss-c03-sys	xen	64	11	127GB	22GB	3.6TB	220.0GB	4

Displaying 4 items

Figure 63: Hypervisors deployed

Figure **63** shows the actual Hypervisor Summary page from the Dashboard web interface.

Also physical are the network switches used to connect servers and nodes, specifically

- 2 stacked switches on an isolated subnet for SAN[32] traffic,
- 2 stacked switches for traffic generated by management VLAN[33], tenant tunnels VLAN and Network node Internet access.

The logical layout of all of the servers and their connections is shown in Figure **64**.

The deployment of the infrastructure is still in progress, next major steps will concern the database services (based on Trove[34]), monitoring and metering services, and the Object Store service (based on Swift[35]).

Other OpenStack features we are interested in are a setup that uses multiple network nodes and the ability to link external Identity Providers for users authentication using SAML[36] protocol.

---

[32] SAN: "Storage Area Network", a network that provides access to consolidated, block level data storage.

[33] VLAN: "Virtual LAN", any broadcast domain that is partitioned and isolated in a computer network at the data link layer. LAN is an abbreviation of local area network.

[34] Trove, a software part of the OpenStack platform.

[35] Swift, a software part of the OpenStack platform.

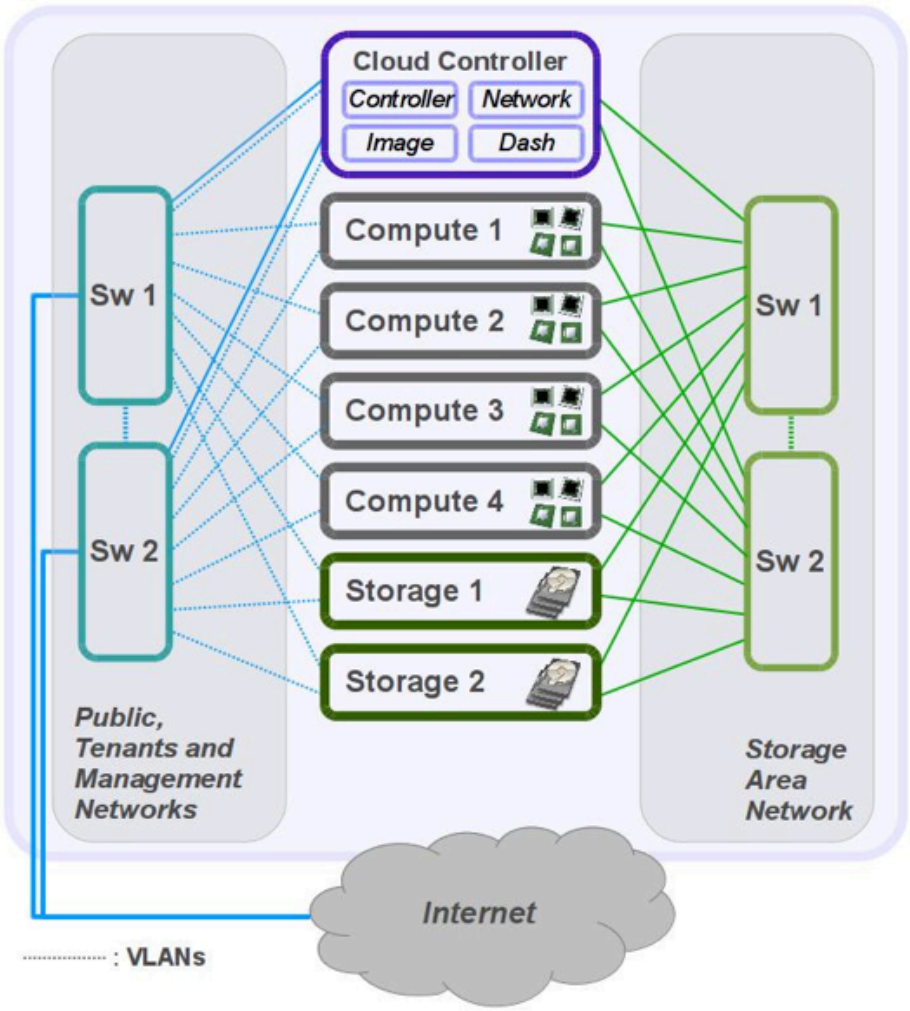


Figure 64: Logical layout of the servers in the Open-stack deployment

7.2 THE SMART OPEN DATA APPLICATION

The primary objective of the SOD activity is to support the semantic interoperability of the applications that are developed in the context of the SA project, allowing these applications to interact with one another in an intelligent way, based on the sharing of the semantics of the exchanged data. In order to achieve this objective, the SOD application relies on ontology of the involved entities, and supports the other applications of the project in mapping their data

[36] SAML: "Security Assertion Markup Language", an XML-based, open-standard data format for exchanging authentication and authorization data.

towards this ontology. Needless to say, the sharing of the data is fundamental to the development of smarter and smarter applications, able to perform sophisticated tasks by exploiting a wider range of data and knowledge.

A secondary but realted objective, is to make the data collected within the SA project available to external consumers, who can take advantage of the data to design and implement other smart services, or to experiment with new algorithms and techniques. To this end, the SOD application develops a Linked Data access point, based on well-known web standards for maximal interoperability also towards the outer world.

The SOD application consists of three main components:

- an integrated semantic repository (ISR) of the knowledge and the data produced by the applications developed within the SA project;
- a set of software modules for acquiring the data to be fed to the ISR from the producing applications; the acquisition includes also the mapping from the original data to the SOD ontology;
- a set of software modules for accessing the data stored in the ISR. These modules are exposed to the humans via a web application called "SMART | OpenData". The application allows the user to (1) explore the data space of the Smart Area via an intuitive browsing facility, and (2) to query the ISR and possibly download the obtained data in CSV format for further processing. Both functionalities are under development. Figure **65** shows the result of a query requesting the events about parking slot occupation in a given time range.

Figure 65: A query result

In the rest of this Section, we will focus on the description of the ontology underlying the ISR, which is the conceptual backbone of the SOD application and at the moment is the most developed component of the SOD.

7.2.1 The Smart Open Data ontology

Conceptually, the SOD ontology consists of three main models:

- the *architectural* model, which is a symbolic representation of the geographic area covered by the SA applications. The architectural model includes classes such as building, room, parking slot, and the like. The architectural properties described relationships of inherence, containment and connection amongst such classes;
- the *sensor* model, which provides a representation of the sensors and the applications deployed in the SA, and relates applications to the sensors that they use, and each sensor to a type and to a physical quantity or event monitored by the sensor;
- the *status* model, which provides a representation of the status of the area, composed of the events and the time series that are observed by the sensors or inferred by applications. The population of the status model produces Big Data.

Correspondingly, the integrated semantic repository consists of three databases: the architectural, sensor and status database (see Section Implementation below).

In this Section, we describe all classes and associations of the Smart Open Data ontology. For each class, we give:

- the informal semantics of the class;
- the superclass(es), if any
- the associations that have the class as domain, if any.

For convenience, the classes have been grouped in six groups, named after the typology of classes included:

1. Architectural classes,
2. Space classes,
3. Digital Data classes,
4. Event classes,
5. Agent classes, and
6. Other classes.

For each property, we give:

- the informal semantics of the property;
- the super-properties, if any;
- the domain and range of the property;
- the cardinality constraints on the property.

The ontology is graphically expressed as a UML class diagram, divided in three diagrams for readability, shown in Figure **66**, Figure **67**, and Figure **68**.

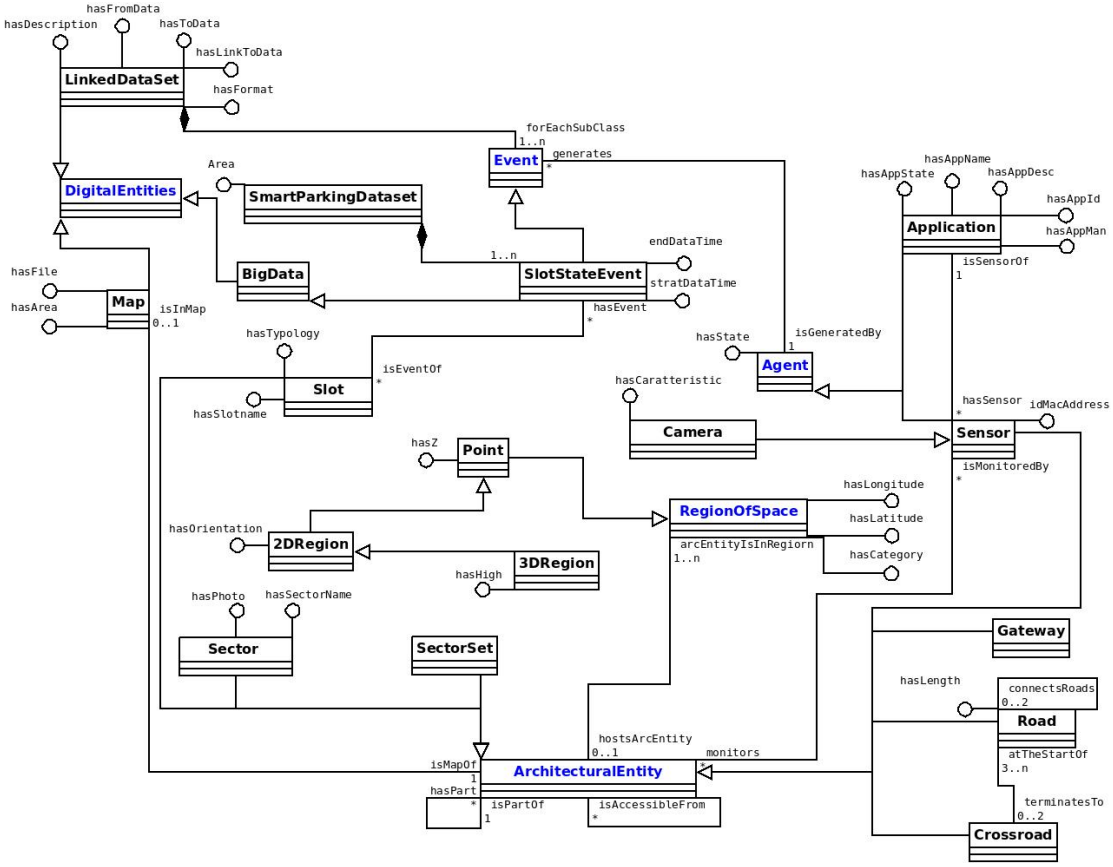


Figure 66: SOD ontology (first part)

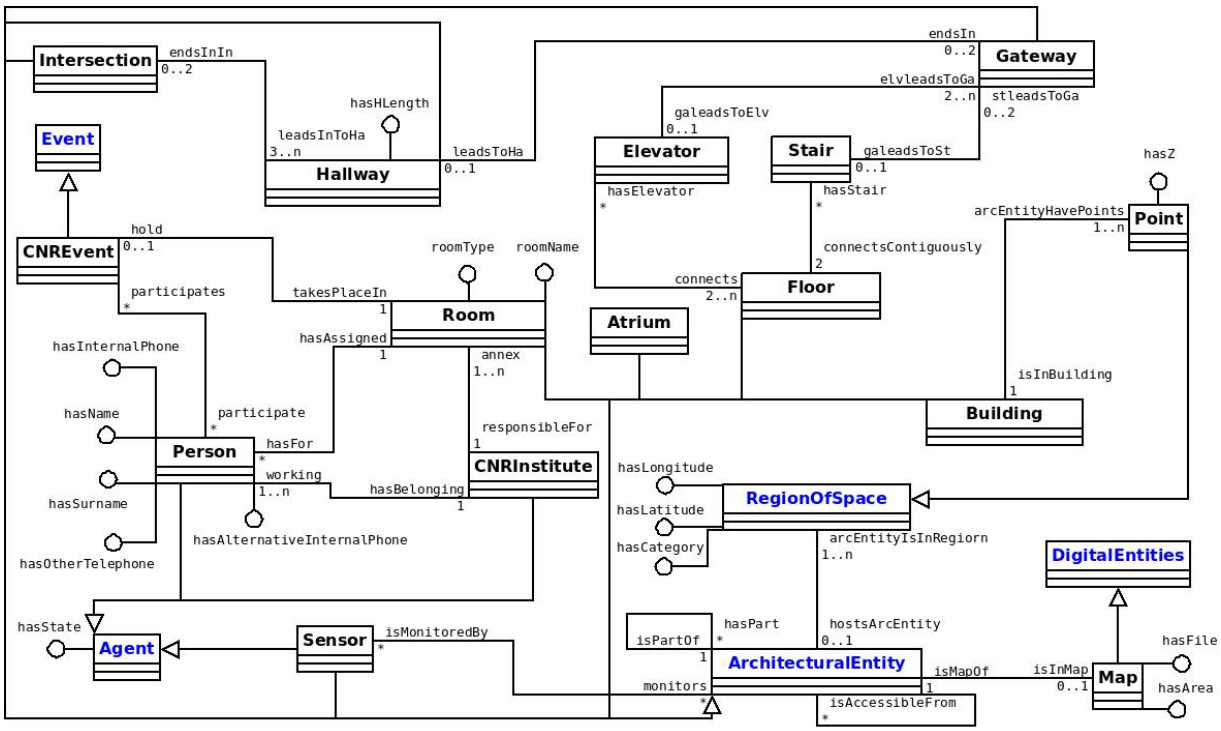


Figure 67: SOD ontology (second part)

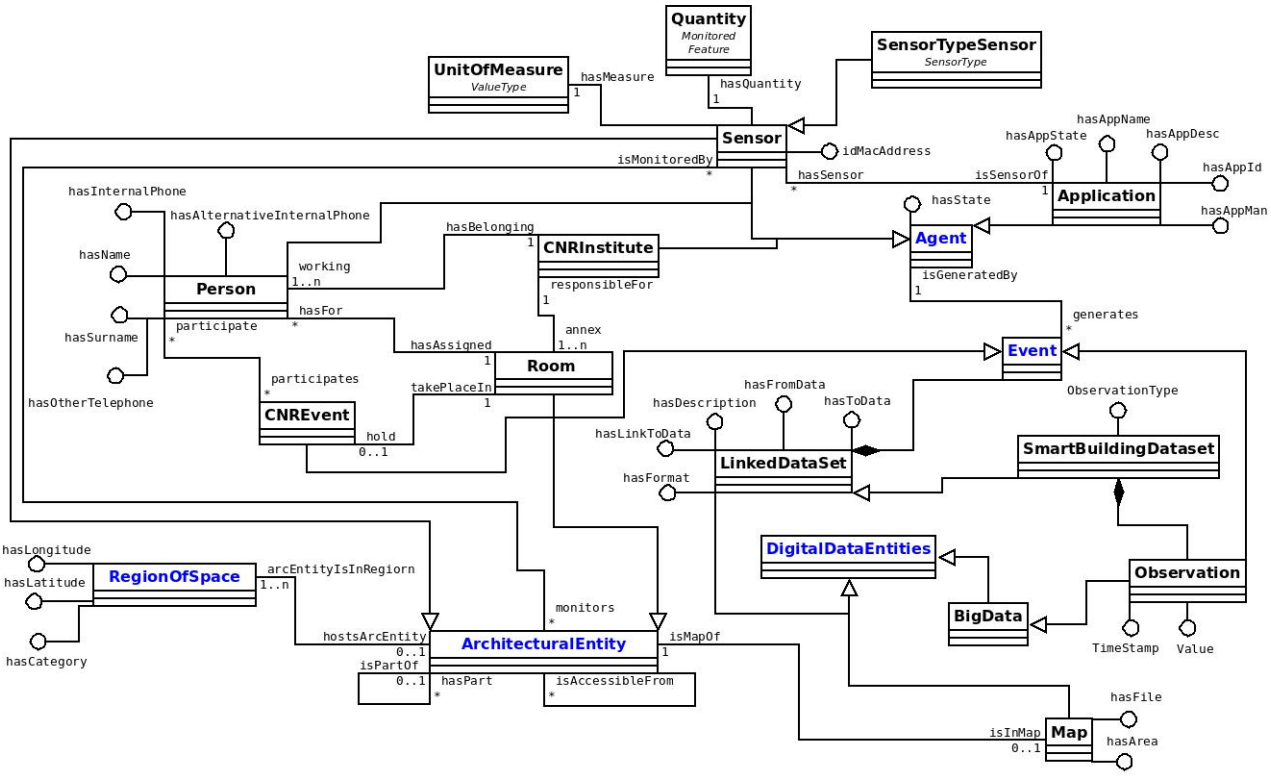


Figure 68: SOD ontology (third part)

7.2.2 Classes

7.2.2.1 Architectural classes

ArchitecturalEntity: has as instances all the architectural entities in the area of interest.
Properties:

- **hasPart**
- **isPart**
- **inMap**
- **arcEntityIsInRegiorn**
- **isMonitoredBy**

Gateway: has as instances all the passages that physically connect two architectural entities.
Super-classes: **ArchitecturalEntity**
Properties:

- **galeadsToSt**
- **galeadsToElv**
- **leadsToHa**

Slot: has as instances the individual parking stalls.in the area of interest.
Super-classes: **ArchitecturalEntity**
Properties:

- **hasEvent**
- **hasTypology**
- **hasSlotName**

Sector: has as instances the maximal contiguous sequences of parking slots.

Super-classes: **ArchitecturalEntity**
Properties:

- **hasPhoto**
- **hasSectorName**

SectorSet: has as instances groups of sectors that are architecturally contiguous**.**
Super-classes: **ArchitecturalEntity**

Road: has as instances the streets within the area of interest**.**
Super-classes: **ArchitecturalEntity**
Properties:

- **connectsRoads**
- **terminatesTo**
- **hasLength**

Crossroad: has as instances architectural entities that are the intersection of three or more **Roads**.
Super-classes: **ArchitecturalEntity**
Properties: **atTheStartOf**

Building: has as instances all buildings in the area of interest
Super-classes: **ArchitecturalEntity**
Properties: **arcEntityHavePoints**

Intersection: has as instances all indoor intersections in the area of interest.
Super-classes: **ArchitecturalEntity**
Properties: **leadsInToHa**

Room: has as instances all rooms in the area of interest that are not bathrooms, *e.g.* laboratories, offices, and so on.
Super-classes: **ArchitecturalEntity**
Properties:

- **hold**
- **hasFor**
- **responsibleFor**
- **hasName**
- **hasType**

Hallway: has as instances all areas within a building that are surrounded by rooms, intersections, or gateways.
Super-classes: **ArchitecturalEntity**
Properties:

- **endsIn**
- **endsInIn**
- **hasHLength**

Atrium: has as instances atriums or lobbies, which generally are large areas within buildings that give access to different areas within the same buildings or outside**.**
Super-classes: **ArchitecturalEntity**

Elevator: has as instances all elevators.
Super-classes: **ArchitecturalEntity**
Properties:

- **elvleadsToGa**
- **connects**

Stair: has as instances stairs connecting the floors of the same building**.**
Super-classes: **ArchitecturalEntity**
Properties:

- **stleadsToGa**
- **connectsContiguously**

Floor*:* has as instances all floors within any building of the area of interest**.**
Super-classes: **ArchitecturalEntity**
Properties:

- **hasStair**
- **hasElevator**

7.2.2.2 Space classes

RegionOfSpace: has as instances all space regions.
Properties:

- **hasLongitude**
- **hasLatitude**
- **hasCategory**
- **hostsArcEntity**

Point: has as instances all individual space points that are relevant to a smart application.
Super-classes: **RegionOfSpace**
Properties: **hasZ**

2DRegion: has as instances all 2D shapes that can be derived from instances of the class **RegionOfSpace**.
Super-classes: **Point**
Properties: **hasOrientation**

3DRegion: has as instances all 3D space regions that are occupied by an entity of interest for the smart data, such as a sensor or an architectural entity.
Super-classes: **RegionOfSpace**
Properties: **hasHigh**

7.2.2.3 Digital Data Classes

DigitalDataEntity: the most general digital entity class, having as instances all data classes.

Map: has as instances digital maps used for visually illustrating some architectural entity.
Super-classes: **DigitaDataEntity**
Properties:

- **isMapOf**
- **hasFile**
- **hasArea**

BigData: A big data class is a class that can have a very large number of instances. This class is the most general big data class.
Super-classes: **DigitaDataEntity**

LinkedDataSet: has as instances all the Linked Data Datasets that store data of some smart application, typically events generated or processed by the application.
Super-classes: **DigitaDataEntity**
Properties:

- **hasDescription**
- **hasFromData**
- **hasToData**
- **hasLinkToData**
- **hasFormat**

7.2.2.4 Event Classes

Event:This includes all individuals that represent events, where an event is something that happens and is of interest to a smart application.
Properties:

- **isGeneratedBy**

CNREvent: has as instances all scientific or social events within CNR.
Super-classes: **Event**
Properties:

- **participate**
- **takesPlaceIn**

SlotStateEvent: has as instances all state transition events of each parking slot.
Super-classes: **Event**
Properties:

- **isEventOf**
- **endDataTime**
- **startDataTime**

Observation: has as instances special kinds of events, consisting in the observation of the quantity value in a specific time instant.
Super-classes: **Event**

- **timestamp**

- **value**

7.2.2.5 Agent Entities

Agent: the class of entities that act or have the power to act (Dublin Core)
Properties: **hasState**

Person: has as instances all persons of interest for the smart applications.
Super-classes: **Agent**
Properties:

- **participates**
- **hasAssigned**
- **hasBelonging**
- **hasInternalPhone**
- **hasName**
- **hasSurname**
- **hasOtherTelephone**
- **hasAlternativeTelephone**

CNRInstitute: has as instances all CNR institutes of interest for the smart applications.
Super-classes: **Agent**
Properties:

- **working**
- **annex**

Sensor: has as instances all sensors as devices whose purpose is to detect events or make observations in the environment where they are deployed. Each type of sensor used by a Smart Application is defined as a sub-class of this class.
Super-classes: **Agent**, **ArchitecturalEntity**
Properties:

- **isSensorOf**
- **monitors**
- **idMacAddress**

Camera: has as instance all video-cameras deployed as sensors in the area of interest.
Super-classes: **Sensor**
Properties: **hasCaratteristic**

SensorTypeSensor: For each specific sensor will have a new subclass that identifies it.
Super-classes: **Sensor**

Application: has as instances the smart applications that produce or consume smart data.
Super-classes: **DigitaEntity**
Properties:

- **hasSensor**
- **generates**

7.2.2.6 Other classes

Quantity: the class of the physical entities measured or observed by sensors, e.g. temperature, electrical energy, etc.
Properties: **descriptionQ**

Unit of Measure: the class of units of measures
Properties: **description**

7.2.3 Properties

generates: Associates an agent with any event that the specific agent can produce.
Inverse property: **isGeneratedBy**
Domain: **Agent**
Range: **Event**
Cardinality: an agent may generate zero, one, or many events. An event is generated by exactly one agent.

hasEvent: Associates a parking slot with any event that is relative to that slot.
Inverse property: **isEventOf**
Domain: **Slot**
Range: **SlotStateEvent**
Cardinality: a slot may have zero, one or more events. A parking slot event is an event of exactly one parking slot.

hasSensor: Associates an application with any sensor that the application uses.
Inverse property: **isSensorOf**
Domain: **Application**
Range: **Sensor**
Cardinality: an application may have zero, one or many sensors. A sensor may be the sensor of exactly one application.

monitors: Associates a sensor with any architectural entity that the sensor monitors.
Inverse property: **isMonitoredBy**
Domain: **Sensor**
Range: **ArchitecturalEntity**
Cardinality: a sensor may monitor one or many architectural entities. An ArchitecturalEntity may be monitored by zero, one, or more sensors.

hasPart: Associates an architectural entity with any architectural entity that is a direct part of it, *i.e.*, this property is not transitive.
Inverse property: **isPartOf**
Domain: **ArchitecturalEntity**
Range: **ArchitecturalEntity**
Cardinality: An architectural entity can have zero, one or more parts. An architectural entity may be part of zero or one entity.

arcEntityIsInRegiorn: Associates an architectural entity to the region of space occupied by the entity.
Inverse property: **hostsArcEntity**
Domain: **ArchitecturalEntity**
Range: **RegionOfSpace**
Cardinality: an architectural entity is in exactly one region of space and one region of space may host zero or one architectural entity.

isInMap: Associates an architectural entity with the map that displays it.
Inverse property: **isMapOf**
Domain: **ArchitecturalEntity**
Range: **Map**
Cardinality: An architectural entity is in zero or one map and a map may be the map of zero, one, or more architectural entities.

isAccessibleFrom: Associates an architectural entity to any other adjacent architectural entity that can be accessed from it. This is a symmetric property.
Inverse property: **isAccessibleFrom**
Domain: ArchitecturalEntity
Range: ArchitecturalEntity
Cardinality: An architecture entity is accessible from zero, one or many architectural entities.

connectsRoads: Associates a road with another read that is the continuation of it. This is a symmetric property.
Inverse property: (reflexive property)
Domain: **Road**
Range: **Road**
Cardinality: a road may connect to zero, one or two other roads.

atTheStartOf: Associates a crossroad with the roads that start at that crossroad.
Inverse property: **terminatesTo**
Domain: **CrossRoad**
Range: **Road**
Cardinality: a crossroad may be at the start of tree or more roads. A road may terminate to zero, one or two crossroads.

arcEntityHavePoints: Associates with a building several points of particular interest for some application (eg. smart wayfinding)
Inverse property: **isInBuilding**
Domain: **Building**
Range: **IndoorPoint**
Cardinality: A building may have zero, one or many point of interest. A point can belong to one building only.
Super-property: **arcEntityIsInRegiorn**

connects: An elevator connects and goes through the floors of a building.
Inverse property: **hasElevator**
Domain: **Elevator**
Range: **Floor**
Cardinality: An elevator may connect two or more floors of a building. A floor may have zero, one or more elevators.

connectsContiguously: The stairs of a building connecting two consecutive floors of the building itself.
Inverse property: **hasStair**
Domain: **Stairs**
Range: **Floor**
Cardinality: Stairs connect two floors of a building. A floor may have zero, one or more stairs.

galeadsToSt: Have the same meaning of the property "**isAccessibleFrom**" but the cardinality changes because a stair has exactly two gateway.
Inverse property: **stleadsToGa**
Domain: **Gateway**
Range: **Stair**
Cardinality: Stairs have exactly two gateways. A gateway may be associated with zero or one stair.
Super-property: **isAccessibleFrom**

galeadsToElv: Have the same meaning of the property "**isAccessibleFrom**" but the cardinality changes because an elevator connecting two or more floors has two or more gateways.
Inverse property: **elvleadsToGa**
Domain: **Gateway**
Range: **Elevator**
Cardinality: An elevator has two or more gateway. A gateway may be associated with zero or one elevator.
Super-property: **isAccessibleFrom**

endsIn: An hallway may end with a gateway.
Inverse property: **leadsToHa**
Domain: **Hallway**
Range: **Gateway**
Cardinality: An hallway ends in zero, one or two gateways. A gateway can lead to zero or one hallway.

endsInIn: An hallway may end with an intersection.
Inverse property: **leadsInToHa**
Domain: **Hallway**
Range: **Intersection**
Cardinality: An hallway ends in zero, one or two intersections. An intersection can lead to tree or more hallways.

hold: The CNREvents takes place in rooms.
Inverse property: **takesPlaceIn**
Domain: **Room**
Range: **CNREvent**
Cardinality: An event takes place in one room. A room can hold zero or one event.
Super-property: **Event**

participates: An event involves the peoples who have organized it.
Inverse property: **participate**
Domain: **Person**
Range: **CNREvent**
Cardinality: A person may participate to many events and an event can involve many people.

hasAssigned: A person working in a room.
Inverse property: **hasFor**
Domain: **Person**
Range: **Room**
Cardinality: A person working in exactly one room and many people may work in a room.

hasBelonging: A person working for an institute.
Inverse property: **working**
Domain: **Person**
Range: **CNRInstitute**
Cardinality: A person working in exactly one institute and many people working in an Institute.

responsibleFor: One room is the responsibility of a particular institute.
Inverse property: **annex**
Domain: **Room**
Range: **CNRInstitute**
Cardinality: A room is in the responsibility of one institute, and an institute annexes many rooms.

7.2.4 Implementation

The SOD ontology is expressed in OWL using the RDF/XML syntax[1], defined by the W3C. The architectural and the sensor databases, being of a relatively small size, are encoded and persisted into RDF format, using the Jena Apache[2] library. Jena is a free and open source Java framework for building Semantic Web and Linked Data Application. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract model. The data loaded into the model are stored into a Triple Store database OpenLink Virtuoso[3], ready to be queried using the SPARQL[4] language.

For the persistence of the status model, we are currently designing a cloud-based storage based on a no-SQL database. Current solutions being considered are Mongo DB [5], Apache CouchDB [6] and Apache Cassandra [7].

7.3 References

[1] http://www.w3.org/TR/rdf-syntax-grammar/

[2] https://jena.apache.org/

[3] http://www.w3.org/TR/rdf-sparql-query/

[4] http://virtuoso.openlinksw.com/

[5] https://www.mongodb.org/

[6] http://couchdb.apache.org/

[7] http://cassandra.apache.org/

---

8 THE INSTALLATIONS IN THE PISA CNR AREA (Ref. A. Gebrehiwot, M. Marinai, IIT)

In brief: *To realize the smart area applications described in the previous Sections, installations were required. The activity described in this Section is composed of two parts. The first part describes the installation process of 11 new outdoor Access Points and the activity carried out for federating two distinct indoor wireless infrastructures of the Research Area of Pisa. While the second part of the chapter describes the installations required by the Smart Parking Application.*

8.1 THE OUTDOOR WIFI INSTALLATION

Within the framework of the Smart Area activity, it was necessary to realize a single wireless LAN infrastructure covering the indoor and outdoor spaces of the Research Area of CNR in Pisa. There are two indoor wireless LAN infrastructures in the Research Area of Pisa. The first one, covering most of the area, is based on CISCO Wireless LAN Controller (WLC) and is managed by Institute of Informatics and Telematics (IIT) while the second wireless infrastructure, covering a smaller zone, is based on Columbris Networks Wireless Tecnology and is managed by Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" (ISTI).

The outdoor WiFi installation is a totally new infrastructure and is based on the CISCO WLC technology using the outdoor unit Cisco Aironet model *AIR-CAP1552E-E-K9* Access Points. The *Telematic Networks* group, which belongs to the Institute of Informatics and Telematics of CNR (CNR-IIT), is in charge of managing the shared networking infrastructures and network services of the Research Area of CNR and has carried out the installation process the outdoor APs.

Actions taken during the installation process of the outdoor access points:

- Outdoor coverage planning based on the map of the Research Area
- Selection of the appropriate outdoor AP model being based on technical characteristics of the outdoor CISCO AP models (Figure **69**)
- A site survey making measurements of the WIFI coverage to calculate the exact number APs units needed for the appropriate deployment. For this purpose we used a demo Cisco Aironet 1552S outdoor Access Point provided for one month by CISCO Systems
- Defined the structured cabling and power supply project for the purchased APs
- Installation of the Structured Cabling
- Installation and configuration of the 11 Cisco Aironet *AIR-CAP1552E-E-K9* Access Points.

The map of the Research Area of Pisa in Figure **70** describes the exact location of the equipment installed, indicated with red dots, and the corresponding intermediate distribution facility where the cabling system of the Access Points are terminated,

indicated with a blue square sign. The location of the Access Points has been chosen to minimize the length of the installed cabling system by putting them close to the wiring closet.

Figure 69: Selected CISCO outdoor AP model "AIR-CAP1552E-E-K9"

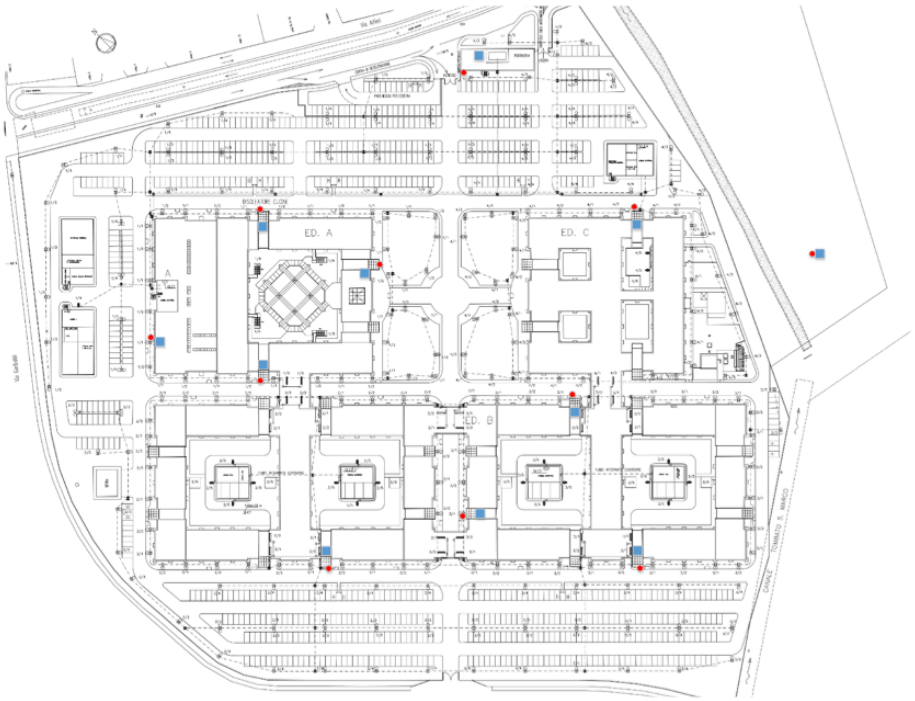


Figure 70: Map of the CNR area in Pisa

The first 10 Access Points have been installed using a wall mounting kit and an omni-directional antenna in the correspondence of the outer body of the building, which corresponds to the entrance doors as shown in Figure **71**. These 10 APs are positioned at approximately 4 meters above the ground level while the last access point has been installed using a pole mounting kit and a directional antenna at the main entrance of the Research Area. The quality of the structured cabling system used is a composite UTP category 5E + 2x0,75 providing both the Ethernet and two additional wires to provide the power supply to the APs. The characteristic of the composite cable is indicated in Figure 72.

Figure 71: The wall installation

UTP 5e

Codice	Formazione	Resist. DC (Ohm/km)	Diam. esterno (mm)	Peso (kg/km)
BNUTP5E	UTP 5e	-	5,25	32,0
BNUTP5E DG*	UTP 5e doble funda	-	6,55	43,0
BNUTP5E05	UTP 5e + 2x0,50	37,7	8,90	85,8
BNUTP5E07	UTP 5e + 2x0,75	24,6	9,28	95,9

*Doppia guaina Duraflam + PE per resistenza meccanica superiore.
Guaina esterna: Duraflam LSZH Blu
Tipo di posa: interna / esterna

Figure 72: The characteristic of the composite cable

In the wiring closet side the network termination has been connected to a patch panel while on the AP side the network termination has been directly connected to the AP. Surge protection units have also been installed for both the data transmission lines and the power supply units.

8.2 FEDERATING THE TWO INDOOR WIRELESS INFRASTRUCTURES OF THE AREA

Within the Research Area of Pisa there were two wireless LAN infrastructures, the first one managed by CNR-IIT and based on CISCO System's Wireless LAN Controller, with a WiFi coverage made by 78 APs (indoor and outdoor units), the second infrastructure managed by CNR-ISTI and based on Columbris Networks Technology and made of 26 indoor APs, with a total of 104 APs. Prior to the federation, the two infrastructures were managed as two completely independent entities, so users were experiencing the following problems:

- Moving between networks did drop connections
- Users close to the borders of the two infrastructures experienced instability of the wireless network
- Users where forced to repeat the authentication process when passing from one infrastructure to another

Within the Smart Area project it was necessary to guarantee the mobility of authenticated WiFi users within the whole Research Area of Pisa. Our goal was to solve the problems listed above by authenticating end users only once and allowing the mobility without loosing network sessions during the migration from one infrastructure to another.

Before the unification of the two WiFi infrastructures, distinct and separate VLAN IDs and IP address spaces were used, so it was evident that active TCP and UDP sessions were interrupted when passing from one infrastructure to the other. It was also evident that, even if the WLAN configurations were the same within the two infrastructures, as in the case of the ERUROAM WiFi network, since the two infrastructures were using two separate VLAN IDs (48 for IIT and 107 for ISTI) and separate IP addressing space, even if the automatic re-authentication was successful the network sessions used to fail due to the change of the IP address of the wireless nodes. So during mobility of users from one infrastructure to another, end users were supposed to disconnect and reconnect to the WiFi infrastructure to start a new network sessions. Being based on what has been reported by users and also on our own evaluation, the objectives of the activity have been the following:

- Users need to authenticate only once to access the network using a federated wireless account
- Users should be able to move in the whole coverage area of both infrastructures without loosing connectivity and network sessions
- The same SSIDs with the same authentication mechanisms, the same VLAN IDs and IP addressing space should be present on both infrastructures to guarantee network continuity.

For these reasons it was necessary to find a way to manage the two WiFi infrastructures as a unique federated WiFi network. To resolve the problem at no cost we started collaboration between the two institutes, IIT and ISTI, by reconfiguring the network equipment as follows:

- The same SSID has been defined on both infrastructures using the same authentication mechanism
- Every SSID defined on both infrastructures should be mapped on the same shared VLAN ID (ex. For SSID EDUROAM, both infrastructures are configured to use VLAN ID 48) using the same IPv4/6 addressing space (146.48.48.0/21 - 2a00:1620:c0:30::/64).

EDUROAM is a federated WiFi network that relies on 802.1x authentication mechanism. For the EDUROAM WiFi network, the above configuration resulted sufficient to unify the two infrastructures. Tests has been done to verify the correct functionality of the solution.

To give access to guest users visiting the Research Area of Pisa, for those who do not have EDUROAM account, WEB based authentication is used. When an end user tries to access a WiFi network with a WEB based authentication, an authentication page is popped up requesting a valid username and password. Since the two wireless infrastructures managed by IIT and ISTI were completely separate end users were supposed to have separate WiFi accounts to access each of the two WiFi infrastructures.

To solve the problem of a federated access to the two WiFi infrastructures, the network has been configured as follows:

- Open SSID “wifi-guest” has been defined on both WiFi infrastructures
- A new VLAN ID 900 has been defined to map the “wifi-guest” users.
- A unique MikroTik wired captive portal, also acting as a DHCP server, traffic shaper and default gateway for the wireless users has been installed (Figure 73).
-

Figure 73: MikroTik router CCR1009-8G-1S-1S+

The overall functionality of federating the two WiFi infrastructures is as follows; when an end user selects the SSID “wifi-guest”, being an open network, access is granted to the user and public IP address will also be assigned by the DHCP server. When the first time the end user starts to navigate on the network, the captive portal installed on the MikroTik router will pop up the authentication page asking for user credentials. If the end user provides valid credentials, access will be granted. Since the two wireless infrastructures are configured as an open network with identical SSID and a unique IP addressing space end users are granted to access the two network infrastructure and able to move from one infrastructure to another without loosing connectivity and network session. For end users without valid credentials, users are allowed to define a valid account by filling a web form and providing a valid phone number and the defined account will be delivered to the end user using our SMS gateway.

8.3 INSTALLATIONS REQUIRED BY THE SMART PARKING APPLICATION

As part of the Smart Area project, it was necessary to realize the structured cabling system necessary for the installation of 19 outdoor cameras to be used for surveillance and parking lot detection on the side of via Luzzatto street. The activities described in this chapter are: the realization of the structured cabling system for data transmission, the power supply units and the pole mounting kits necessary for the installation of the outdoor cameras on the roof of the building.

The map in Figure 74 shows the left side, looking at the exterior of the building of the Research Area on the side of via Luzzatto, and indicates the first 10 locations of outdoor cameras (red squares in the map). The cabling system of these 10 cameras is terminated in a single network cabinet, marked with the light blue square. The other 9 locations are installed on the right side of the building of the Research Area on the side of via Luzzatto, cabling system is terminated to a different network cabinet, the installation is identical to the first one.

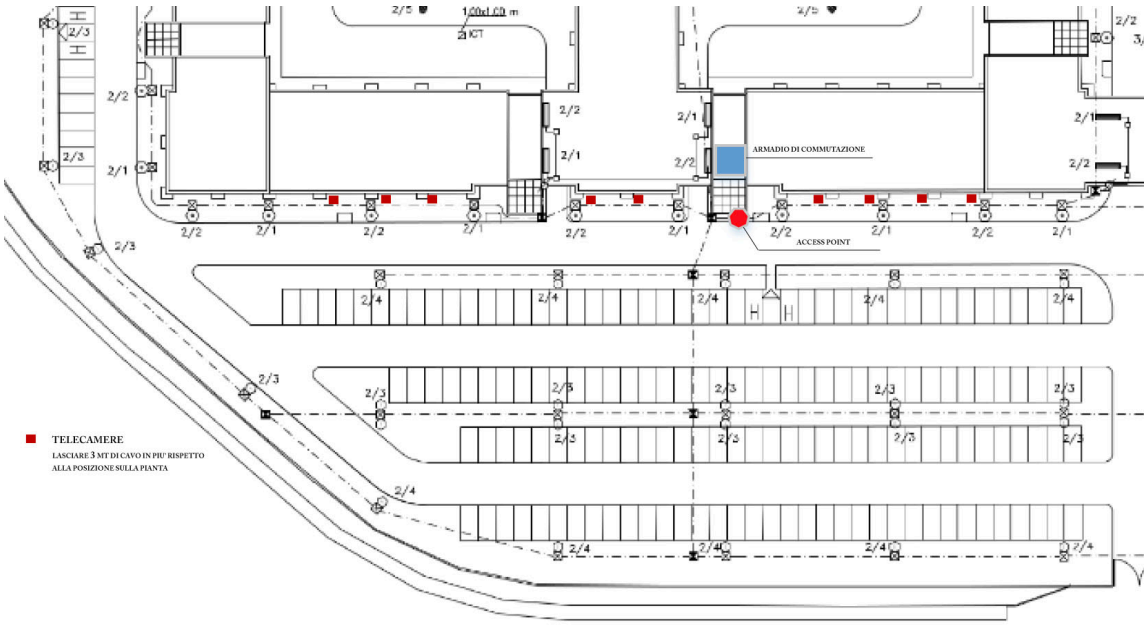


Figure 74: Left side, at the exterior of the building, indicating 10 positions of cameras

In each network cabinet a 24 port Cat 5E RJ45 Patch Panel is installed and is used to terminate the structured cabling coming from the outdoor cameras. In each of the two network cabinets a 25A Circuit breaker switch has also been installed, to protect equipment in case of overvoltage or lightning. In addition a total of 19 Differential Switch Breaker C6, one for each outdoor cameras, has been installed. The characteristics of the cable used is a composite UTP category 5E + 2x0,75 providing both the Ethernet and the two additional cables used for the power supply of the cameras. A pole-mounting tool has also been provided for each camera.

9 THE SMART AREA WEB SITE

(Ref. Antonino Crivello, ISTI)

The web site for the Smart Area (Figure 75) has been developed to allow users a full description of the developed applications, to direct hyper-links to the applications, to link the Facebook page and to receive feedbacks and suggestions from the applications' users to improve both the web and the applications and to suggest new applications. The web link is: ***www.smart-applications.area.pi.cnr.it***

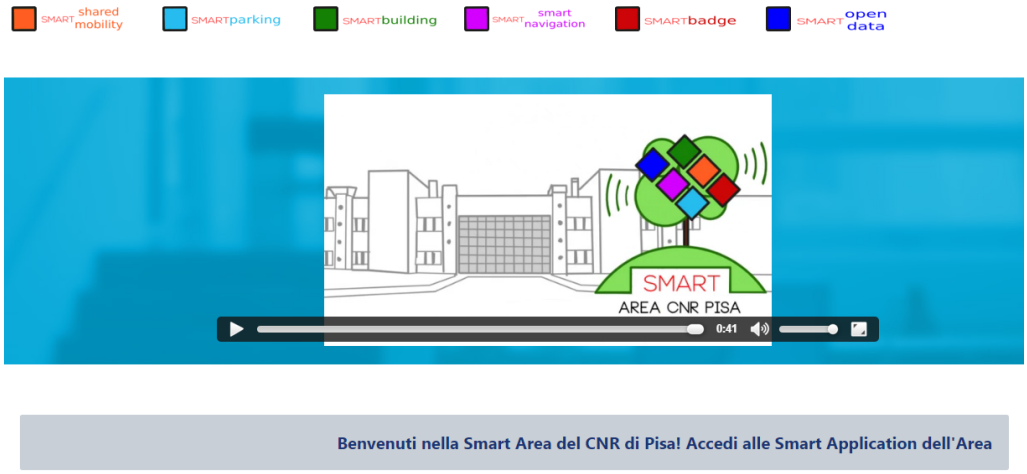


Figure 75: The Smart Area Home Page

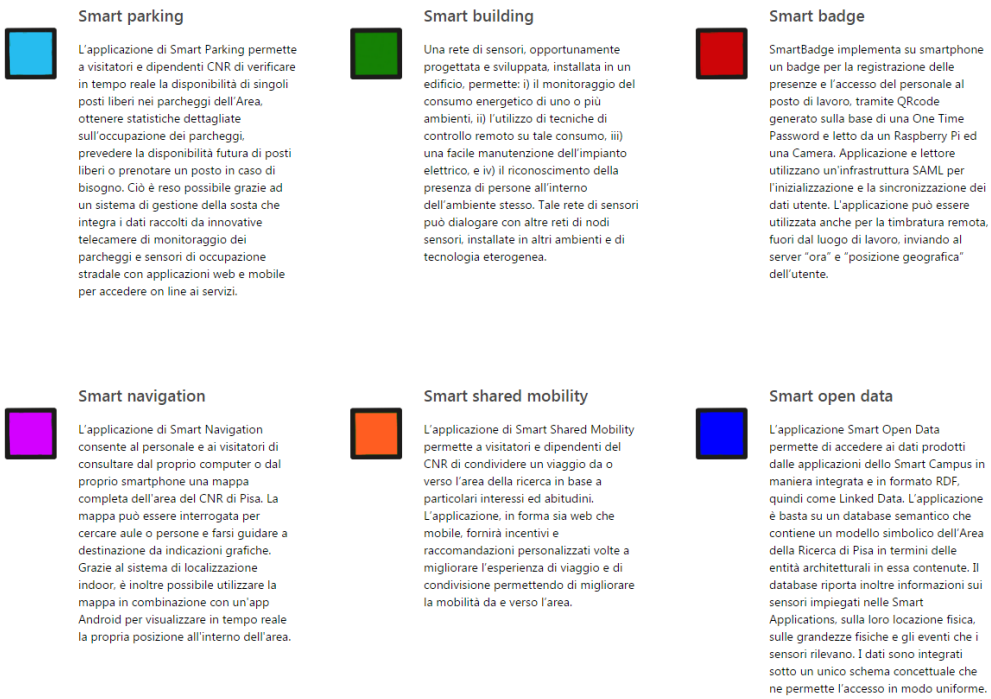


Figure 76: The Application Descriptions

Figure 75 shows the upper part of the web site. At the top, the home page shows a top bar with six hyperlinks to the six applications developed. A user can find the right applications without scrolling down the page. The user can easily recognize the applications by the use of different colours (green: smart building; red: smart badge; heavenly: shared parking; orange: smart shared mobility; blue: smart open data; violet: smart navigation).
A short video shows and explains the applications developed for the Smart Area; below the video, the user can find a short description of the applications and the relevant links to enter the applications and use them (Figure 76).

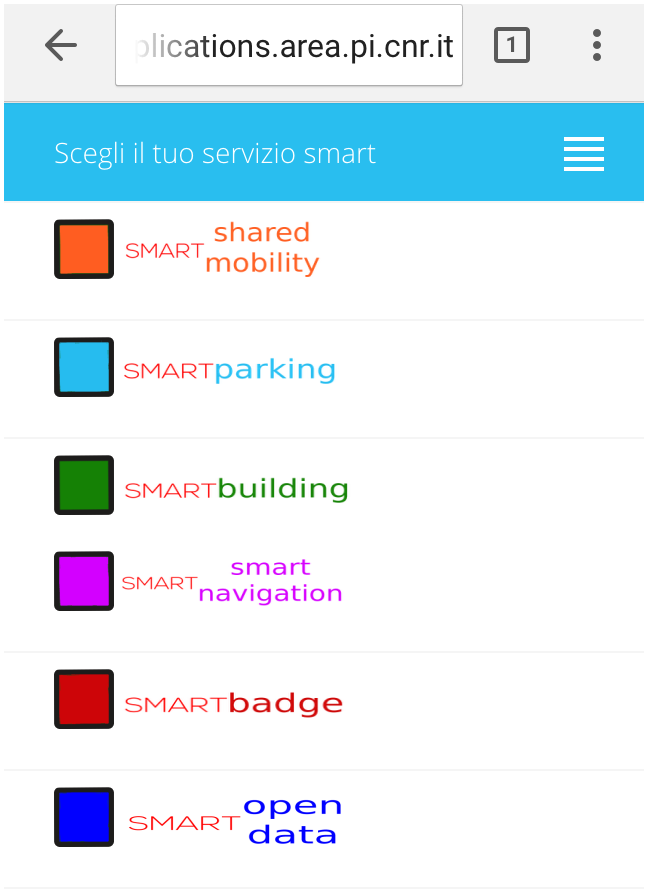


Figure 77- Mobile views

The smart area web home page gives the same functionality even in mobile/tablet navigation (Figure **77**). Currently the home page is in Italian language; in the future, the English version will be available.

The Web Site also contains:

- a link to Facebook: *https://www.facebook.com/smartareacnrpisa*
- a link to Twitter: *https://twitter.com/SmartAreaPisa*
- a link to download technical documents
- a link to send suggestions and to propose ideas for new applications
- a link to follow the events related to the smart area activities.

10 THE PEOPLE INVOLVED IN CNR SMART AREA 2015

Smart Area 2015 was possible thanks to the collaborative work of the following "smart" people belonging to the ISTI, IIT, and IFC Institutes of CNR located in the Pisa Research Area:

Matteo Abrate, Giuseppe Amato, Clara Bacciu, Paolo Barsocchi, Filippo Benedetti, Raffaele Bruno, Mattia Giovanni Campana, Fabio Carrara, Luciano Celi, Antonino Crivello, Raffaele Conte, Franca Delmastro, Andrea De Vita, Gianluca Diodato, Erina Ferro, Luigi Fortunati, Vincenzo Galella, Abraham Gebrehiwot, Claudio Gennaro, Paolo Gentilini, Francesco Gianetti, Michele Girolami, Hanna Kavalionak, Giuliano Kraft, Giuseppe Riccardo Leone, Alessandro Lucaferro, Massimo Magrini, Alessandro Mancini, Andrea Marchetti, Mario Marinai, Maurizio Martinelli, Fabio Mavilia, Carlo Meghini, Davide Moroni, Francesco Napoli, Francesca Nicolini, Sergio Palumbo, Gabriele Pieri, Luca Pisani, Francesco Potortì, Giancarlo Riolo, Luca Trupiano, Ovidio Salvetti, Gianmario Scanu, Gian Marco Sibilla, Marco Tampucci, Calogero Todaro, Anna Vaccarelli, Claudio Vairo, Fabio Valsecchi, Loredana Versienti

Thanks also to:

- *Dott. Marco Conti, Director of CNR DIITET Department*
- *Ing. Ottavio Zirilli, Director of the CNR Area in Pisa*
- *Dott. Domenico Laforenza, Pisa CNR Area President and CNR-IIT Director*
- *Dott. Claudio Montani, Director of CNR-ISTI*
- *Dott. Giorgio Iervasi, Director of CNR-IFC*
- *All the CNR Area Supervision personnel*