IoT Manager: a Case Study of the Design and Implementation of an Open Source IoT Platform

Luca Calderoni
Dept. of Computer Science and Engineering
Università di Bologna
Cesena, Italy
luca.calderoni@unibo.it

Antonio Magnani
Dept. of Computer Science and Engineering
Università di Bologna
Cesena, Italy
antonio.magnani@unibo.it

Dario Maio
Dept. of Computer Science and Engineering
Università di Bologna
Cesena, Italy
dario.maio@unibo.it

***Abstract*—IoT represents one of the most insightful trends concerning ICT. As we assist to a growing diffusion of heterogeneous sensor networks deployed all over urban areas, one of the main concerns is to provide the community with versatile and easy-to-go frameworks capable to serve and to organize data collected by these sensors. However, as we may notice, the world largest information technology companies tend to release user friendly IoT platforms and services avoiding to reveal the know-how concerning design and implementation details of these products. As a consequence of this business strategy, a common trend for universities and other teaching institutions is to use these mainstream IoT platforms during their classes in a 'as a service' fashion, omitting to unveil technical details and design strategies these platforms rely on. In this paper, we present IoT Manager, a general framework designed for sensor networks management which was completely designed and implemented within the University of Bologna. Our main aim is to provide the scientific community with a detailed implementation strategy concerning a specific IoT platform in order to disseminate such topics in a more precise and understandable way, both for research and teaching purposes.**

***Index Terms*—Internet of Things, IoT platform, IoT framework, Software engineering**

I. Introduction

THE Internet of Things (IoT) is rapidly growing and spreading in several aspects and scenarios of everyday life [14]. The pervasive presence of different objects spread around us is the main idea behind this paradigm: this vision is increasingly tangible and sensor networks are becoming a reality in urban and suburban contexts [2], [3], [10]. However, one of the possible drawbacks of this rapid evolution is the uncontrolled proliferation of heterogenous sensor networks lacking interoperability features, even when they are deployed within the same context. For instance, how may we enable an easy and integrated solution to combine data generated by the well-known *Santander sensor network*[1] with those produced by devices of the *Smart Citizen platform*[2] or originating from any other device? Such a model raises several issues: on the one hand, those derived by the different nature of involved actors and business models, on the other hand those represented by technical challenges. The variety of architectural models for sensor networks, the etherogeneous nature of sensors and the lack of shared methodologies and best practices concerning their deployment [15] are good examples to describe the complexity behind IoT solutions. Several companies have released their own solutions concerning IoT. These platforms are delivered in a ready-to-go fashion and offer users a number of advantages as, for instance, a native cloud computing integration for each component of the infrastructure they rely on. The main concern with respect to these platforms is their lack in transparency and implementation details [12]. Predictably, majors are not interested in divulging key technical aspects as this could reveal too much about their technologies and strategies. This condition thus implies some drawbacks for the scientific community, as each IoT solution relying on this mainstream platforms is in some sense dependent on a number of black boxes. The goal of this paper is to propose an utterly homemade solution relying on open source technologies; in particular, the realization of a flexible middleware that allows the rapid interfacing of heterogeneous sensor networks, similar to the conceptual model of *Global Sensor Network* detailed in Aberer et al. [1]. Moreover, this paper discusses design and implementation details at each layer of the stack our platform is built upon, enabling researchers and practitioners to fully understand what lies behind a IoT solution. Several academic institutions have proposed their own IoT platforms, as per the case of Castellani et al. [7], where the focus is posed on indoor environments. Our solution is conversely oriented to outdoor sensors while it preserves the aim to offer a testbed designed for teaching and research purposes within the IoT domain, similarly as proposed in [8], [11]. The aforementioned framework, named *IoT Manager*, also sports several features designed to improve sensor networks interoperability. This condition offers several benefits such as the availability of different types of data, the opportunity for a citizen to take part in sensing activities adding nodes to a sensor network [9] or the ability for a user to increase or reduce the granularity of sensed data within a particular area of interest. IoT Manager pursues these goals and offers users the ability to couple sensing devices in an agile way.

[1] http://maps.smartsantander.eu/
[2] https://smartcitizen.me/

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

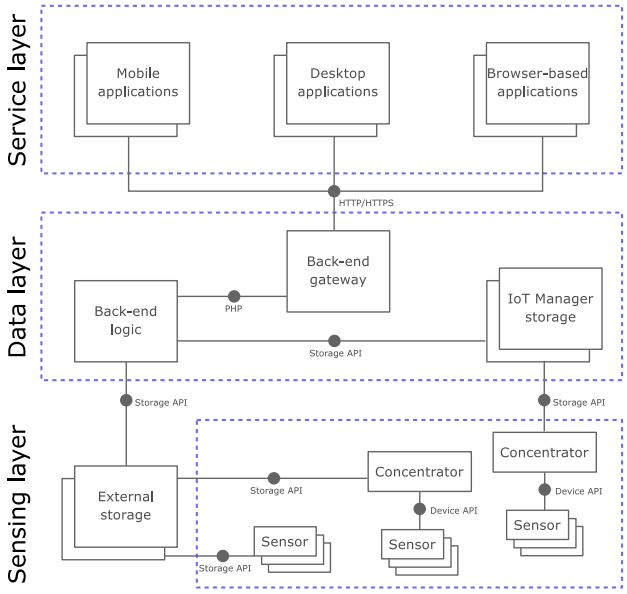

Fig. 1. A high-level diagram showing IoT Manager architecture.

II. IoT Manager

In [6] Calderoni et al. proposed a general ICT architecture designed to manage several subsystems in urban contexts. *IoT Manager* represents an evolution of this model and implements its main features. Throughout this section we want to clearly explain how this home made platform was designed and implemented, in order to provide the scientific community with a tangible example of a fully open IoT stack. It is indeed common for IoT players not to reveal any detail about sensors to be handled through their platforms as well as about those services relying on data which their platforms are able to deliver. All the discussion is usually directed at the infrastructure hosting the platform and at the related capabilities. In the following we will instead investigate each aspect concerning the proposed platform. Following a top-down approach, we first describe IoT Manager high-level design and we proceed with an in-depth discussion about specific sensors, the integration middleware and the connected client applications. Specifically, the integration middleware shares several architectural aspects with the platforms developed by major companies.

In [13] a taxonomy is proposed in order to classify IoT platforms in relation to the corresponding area of application: following this taxonomy, our solution refers to the *Application Enablement Platforms*. From a high-level architectural point of view (see Figure 1 for reference), this system is composed of three layers, similarly as discussed in [5].

The **sensing layer** consists of a number of heterogeneous sensor networks. These subsystems may be deployed all around the globe and are responsible for raw data collection. IoT Manager supports a two-level taxonomy with respect to these sensors. Specifically, within the sensing layer, each object may be treated as a *simple sensor* or as a *concentrator* connecting a number of simple sensors. This feature plays a key-role with respect to user requests to access sensors information from within a client application. In fact, IoT Manager is a fully geo-referenced platform and is able to retrieve those sensors which lie within a specified range from the end user. Although this feature may be useful in a number of scenarios (for instance when a maintenance engineer needs to check out the position of a compromised sensor), it is not enough in order to represent the logical connection which exists between a set of sensors and the related concentrator. Thanks to the two-level taxonomy, the *back-end gateway* allows for requests which only address the set of simple sensors connected to a given concentrator. Sensors and concentrators communicate raw data through some *storage API* (depending on the storage engine). These data may be stored both in IoT Manager internal storage or through external storage services. This is another key-aspect of our platform: as the *back-end logic* is able to retrieve raw data from internal and external storage, it is possible for third parties to connect their sensor network directly to IoT Manager clients allowing the back-end logic to access their repositories through a set of predefined APIs. Within Section II-A we describe some sensors which are already handled by IoT Manager and we also detail the procedure used to build one of this devices from scratch.

The **data layer** represents the back-end of the system and is responsible for two main features: on the one hand, it serves as a repository for all of the sensed information, on the other hand, it provides several API which may be called by client applications in order to query those data and retrieve them in a properly arranged format. This layer is responsible to preserve compatibility across different subsystems, providing clients with an efficient and transparent way to access data. The *back-end logic* (see Figure 1) represents the more sophisticated component of the system and acts as an integration middleware. Specifically, this component is able to retrieve raw data produced by sensors and concentrators using a set of predefined APIs which allow it to query different storage engines. Additionally, data retrieval is enabled both for internal and external storages. Although raw data may be retrieved from a variety of different repositories, the back-end logic is able to rearrange these records in order for them to comply with a specific format, in accordance with the *back-end gateway* dispositions. The back-end gateway is another key component of this layer. It exposes HTTP/HTTPS APIs to enable communication with client applications. It is also responsible for requests translation (in a set of jobs handled by the back-end logic component) and for final response formatting (JSON). An in-depth discussion about the back-end gateway and the back-end logic is provided in Section II-B.

The **service layer** contains a number of client applications communicating with the data layer through the back-end gateway APIs. As these APIs rely on HTTP and HTTPS protocols, it is quite simple to integrate them in any desired end-user application (desktop, mobile, browser-based applications etc.). Within Section II-C we provide a detailed design of one of these client applications, which has been developed for

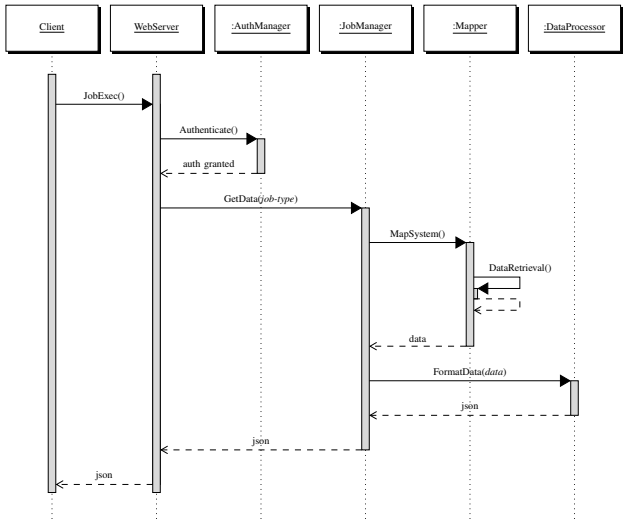

Fig. 2. IoT Manager requests processing from the back-end perspective.

Android mobile devices. Clients are subject to a specific access policy and handle geo-referenced data.

A. Sensing layer: some examples

Our solution deals with different types of sensors, one of which consists in a low-cost weather station. This prototype relies on a UDOO Neo Extended board. This board includes an I^2C (Inter Integrated Circuit) connector used to plug sensor modules (UDOO bricks). One of the main features concerning UDOO bricks is the ability to work through a cascade configuration: it is allowed to connect several sensor modules using the sole I^2C interface on the board. Of course, it is also allowed to connect sensors directly to the Arduino socket provided by the board [4]. In our experiment, we used three different sensor modules: a *Barometer-Temperature brick*, a *Light brick* and a *Humidity brick* providing relative humidity percentage. Data received from each sensor are collected by the UDOO operating system and then sent to an external storage via HTTP/S API. In order to comply with IoT Manager specifications, the payload also includes some mandatory information (*sensor identifier*, *sensor name*, *subsystem identifier*, *status*, *latitude* and *longitude*). These fields are introduced in Section II-B.

Besides weather stations, the sensing layer is composed of a number of other devices such as *ArLu* and *Lamps*. An ArLu, representing a lighting cabinet, acts as a *concentrator* and is logically connected to a set of simple sensors (Lamps) allowing a full lighting system management. This two-level taxonomy enables a logical partition even when ArLu and Lamps are not physically connected one each other.

B. Data layer: the back-end logic

When a client application queries the back-end for data, the data layer acts as outlined in Figure 2.

The client application delivers a request over a HTTP/HTTPS post channel. The web server, implementing the back-end gateway, handles this request and, first of all, checks for user authentication. This operation is performed by the *AuthManager* class, a specific software component which addresses authentication queries to the central IoT Manager storage. Thanks to a complete integration with prepared statements, this module preserves the framework from being affected by SQL injection. On authentication granted, the back-end gateway instantiates a *JobManager*: this module checks for the type of the handled request and instantiates in turn a *Mapper* object in order to retrieve data. The set of job types supported by our framework along with each request parameter are reported in Table I.

TABLE I
IoT MANAGER INPUT PARAMETERS AND JOB TYPES DERIVED FROM THE HTTP SERVICE CONTRACT EXPOSED BY THE BACK-END GATEWAY.

Parameter	Description
user, pwd	Username and password for authentication.
filter	Susbsystem identifier (0: all subsystems).
id	Single sensor or single city identifier, depending on the job.
minLon, maxLon	Longitude bounding values.
minLat, maxLat	Latitude bounding values.
job	Job identifier, as outlined below.
Job	**Description**
1	Returns a list of sensors lying within a specific bounding box specified by the calling application. Depending on the *filter* parameter, it is possible to address this request to a specific subsystem (a specific set of sensors) or to each subsystem.
return	[*id, name, subsystem, longitude, latitude, status*]:list
2	Returns a single sensor and all of its related information in a key-value fashion. The identity of the sensor is provided in the request through the couple *subsystem, id*.
return	[*attribute name, value*]:list
3	Returns the list of subsystems handled by IoT Manager.
return	[*subsystem, name*]:list
4	Returns the list of known cities in the back-end atlas.
return	[*city, name*]:list
5	Returns a single city and all of its related information.
return	*city, nation, name, longitude, latitude, gmt*
6	Returns the list of sensors connected to a specific concentrator (uniquely identified by *subsystem, id*).
return	[*id, name, subsystem, longitude, latitude, status*]:list
7	Returns a key-value list exposing a semantic description of each attribute for each specific subsystem.
return	[*attribute name, description*]:list

While jobs 3, 4, 5 and 7 depend on meta data and affect IoT Manager storage only, jobs 1, 2 and 6 may also affect a number of external storages. In fact, as previously discussed (see Figure 1 for reference), our framework is able to retrieve raw data both from its own storage and from a number of external sources. As we may notice, each of these jobs is completely transparent with respect to the calling application concerning real data location. Thanks to a set of back-end APIs, the Mapper object connects to each subsystem and retrieves each relevant record. Desired records are thus collected by the JobManager object and prepared for being returned to the calling application by the *DataProcessor* (see Figure 2 for reference). The latter class is responsible for data formatting in compliance with the service contract through JSON notation.

Our back-end logic thus relies on several APIs for data retrieval. It is important here to point out that each retrieved

record may belong to a separate subsystem, each holding specific features. As a consequence, data may contain a large number of heterogeneous attributes. This is the reason why we defined a restricted set of attributes which subsystems need to exhibit as a mandatory requirement for them to be connected to IoT Manager. Specifically, these attributes shall represent a *sensor identifier* (unique within its own subsystem), a *sensor name* (or description), the *identifier of the subsystem* they belong to, a *status* information and a couple of fields specifying the *longitude* and *latitude* coordinates of the sensor. It is meaningful to note that these data do not need to be stored under a single or predefined column name. For each external source, the Mapper queries IoT Manager meta data in order to know which column or columns contain each mandatory information and which names represent those columns within the external storage schema. This mapping feature provided by the back-end logic allows for a proper implementation of jobs 1 and 6 which, as should be noticed, produce a list of compliant information derived from heterogeneous subsystems. This allows client applications to easily handle sensor lists throughout each part of the user interface where sensor-specific details are not required. Conversely, when a calling application would require something specific about a single sensor, a different mapping principle applies. This is indeed the case of job 2. The back-end logic access the aforementioned meta data and search for column mapping concerning sensor and subsystem identifiers. Through the proper connection API, the Mapper queries target storage for each data related to the sensor and blindly collect them. Sensor-specific data are then JSON formatted and returned through the HTTP service in a key-value fashion. The calling application is thus responsible for data interpretation. In order to build a proper user interface and to correctly show meaningful data, end-user application developers may rely on job 7, which provide the client with a human readable description of each returned field. Finally, a couple of words about georeferencing. IoT Manager natively supports positional data. Mobile services built against the IoT Manager framework may use GPS coordinates to enrich their queries with bounding box information. However, when a client application is not aware of its location, or when the hardware it is executed on is not equipped with any form of location sensing device, job 4 and 5 may be used to simulate user's position as derived from the framework atlas.

C. Service layer: a client application

As previously discussed (see Figure 1 for reference), the service layer is an ensemble of applications designed to interact with IoT Manager data layer. Within this section we explore this layer through a real application which was designed and implemented by our research team. The service we are about to discuss consists of a mobile application built over Android OS.

The main aim of this application is to sense location information through GPS and network hardware and to display sensors which lie within a given distance with respect to the device itself. The user is also allowed to displace his

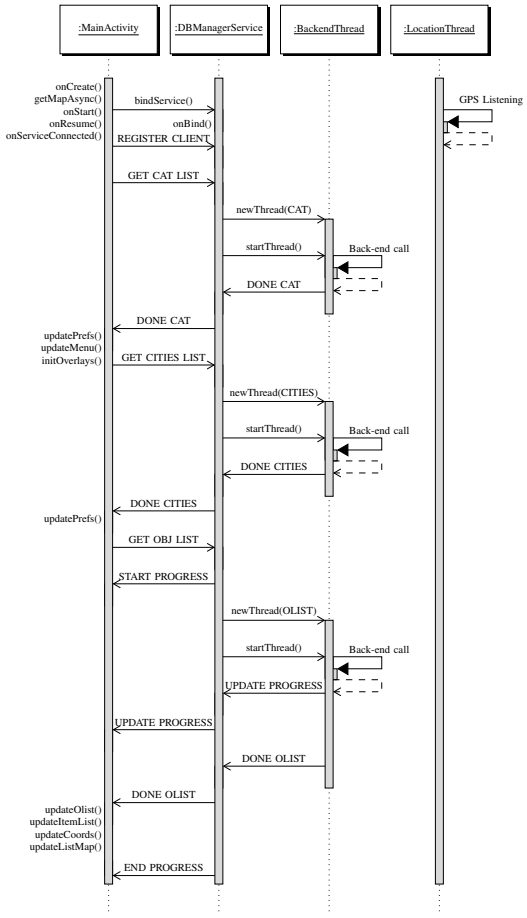

Fig. 3. Main activity starting sequence. Here we assume login information has been already filled in the application preferences and no specific city from the back-end atlas was selected instead.

position using one of those provided by the back-end atlas. As this application is intended to be used in the IoT domain, it is designed with multi-threading and asynchronism in mind. Each client *activity* relies on a shared Android service in order to obtain positional data as well as each kind of external data to be downloaded through IoT Manager HTTP API.

When the client application boots up, a *welcome activity* is initially started along with a *service* running on a separate thread. The activity first checks for user permission concerning GPS and Network and, on permission granted, asks the service for location coordinates. The service then starts a dedicated thread which implements several primitives provided by Android OS able to deal with GPS and Network sensing. When a fresh position is sensed, the location thread sends a message to the service, which in turn sends these new coordinates to each connected activity. As the welcome activity receives coordinates, the program control passes to the *main activity* which immediately binds to the service. The main activity first checks for authentication information within the application preferences. Figure 3 shows its starting sequence assuming these credentials were already provided by the user.

While the Android service and the location service keep on

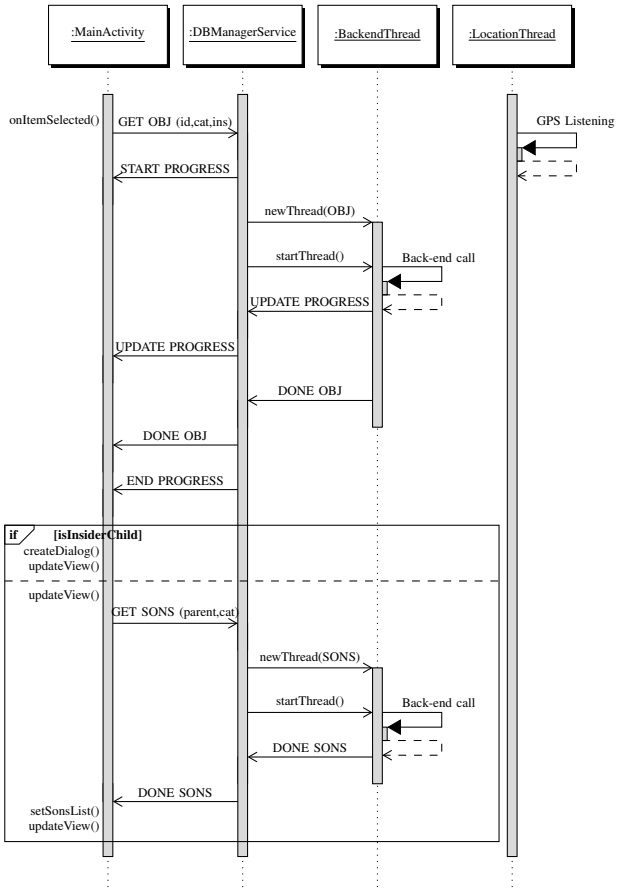

Fig. 4. Sensor details request. The collected information replaces the map layout container or, when the request arises from that container itself (*isInsiderChild* = *true*), it is shown in a dedicated dialog.

running on their own threads, this sequence diagram shows a new type of thread which has been designed to handle back-end calls. During its starting sequence, main activity asks the service for a number of external data. For each task, the service instantiates a single thread implementing the IoT Manager communication service and propagates the request to the endpoint through HTTP/S. Specifically, it first asks for the complete list of subsystems handled by the back-end (Table I, job n.3). Then it requests the list of cities stored in the back-end atlas, used to populate a specific combo box within application preferences (Table I, job n.4). Finally, it asks for a list of sensors (belonging to any subsystem) which lie within a predefined range from the user (Table I, job n.1).

As we may see, each back-end call is handled by a specific thread and does not affect the application responsiveness at all. Please note that each *back-end call* depicted in Figure 3 may be exploded with the sequence diagram provided in Figure 2. When these calls are completed and information is returned to the calling activity, the application GUI is updated with sensors data. A sorted list of sensors (with respect to the distance to the user) is populated on the left, while a map showing an overlay icon for each sensor is proposed on the right. It is meaningful to point out that, at this stage, no detailed sensor information is required. In order to populate list and map is enough to know basic information as those returned by job 1 or 6 (see Table I for reference). Consequently, our GUI is subsystem-independent and is able to deal with heterogeneous sensors with no need for specific personalization.

When the user clicks or taps on a specific sensor, a request for sensor's details is propagated to the back-end, as depicted in Figure 4.

This sequence implements the call for job n.2 (see Table I for reference). When the download process terminates and the information is delivered to the main activity, the GUI is properly updated. Again, as the given sensor could be a concentrator, another back-end call (job n.6) is propagated in order to show the list of related sensors. Conversely as per the sequence proposed in Figure 3, the information to be shown is sensor-specific and, thus, a specific layout needs to be designed to arrange it. Our Android client is conveniently designed to this purpose and it is provided with a *class factory* which instantiates the proper object on a subsystem basis. A simple class diagram showing some specializations of the abstract class implementing a single sensor is provided in Figure 5.

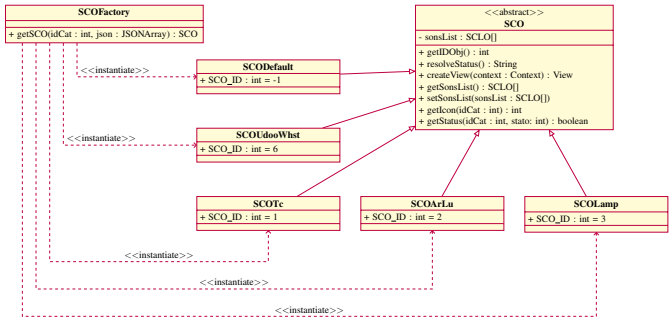

Fig. 5. The Android client class factory, the abstract class representing a single sensor and some of its specializations.

Each sensor class need to specialize an abstract method *createView()*. This method should contain those instructions used to render a proper layout for the sensor. Consequently, when we need to show some sensor-specific detail within the GUI, it is sufficient for us to call this method on the object representing the given sensor, without any other knowledge about its features.

It is finally important to stress that IoT Manager is designed with research and teaching purposes in mind. We released an open distribution of this client application on *GitHub*[3]. This approach allows students and researchers to synchronize their IDE with IoT Manager's repository and to develop their own IoT solutions against the framework.

III. Case Study and Future Improvements

Currently IoT Manager includes four different types of sensors: in addition to the already mentioned ArLu, Lamp and Weather Station (see Section II-A) a sensor called *Traffic Controller* (TC) is also included. This sensor is based on a

[3]https://github.com/smartcitylabunibo

TABLE II
GEOGRAPHICAL DISTRIBUTION AND QUANTIFICATION OF THE VARIOUS TYPES OF SENSORS CURRENTLY PROVIDED IN OUR CASE STUDY.

Sensor	Quantity	Distribution
ArLu	$\simeq 50$	Europe
Lamp	$\simeq 500$	Europe
Traffic Controller	$\simeq 30$	Europe and Morocco
Weather Station	$\simeq 10$	Italy

smart camera that continuously monitors a road section using some virtual spires placed on the lanes. The TC is responsible for counting, categorizing and estimating the speed of vehicles crossing the virtual coils that are placed in strategic points of the roadway. These four types of sensors have reached a good pervasiveness in the European continent (see Table II and Figure 6) and represent an excellent case study for both industrial and research purposes. Data collected by these sensors were derived from an agglomeration of corporate databases and research outcomes as the result of a number of collaborations between the University of Bologna and other institutions.

Our research and teaching team is constantly working on IoT Manager platform. Several modules were implemented during the recent years in order to expand the data layer capabilities as well as to extend the set of subsystems handled by the framework. Several efforts have been also carried out in order to improve the service layer. As one of the main concern of IoT Manager is interoperability, we will devote our attention to the platform's APIs. Two are the main challenges with respect to this subject: first, a wider set of communication protocols should be exposed by the back-end gateway. As an example, several IoT platforms accept connection from MQTT or WebSockets protocols, which are not handled by our middleware at the moment. Second, the back-end mapper should be provided with a wider set of external storage engine APIs. This condition would indeed lead to an easier connection of pre-existing subsystems. Specific attention should be posed on NoSQL databases and column-based storage engines.

IV. CONCLUSION

In this paper we introduced IoT Manager, a full stack IoT platform relying on open source technologies. As a lot of research and teaching projects within this field rely on hidden details which private companies do not tend to unveil, our main aim was to provide the scientific community with a tangible implementation of such a solution, along with a detailed description of our design strategies at each level of the stack.

ACKNOWLEDGMENT

This work has been funded by Italian Ministry of Economic Development (MSE), within the framework of the Program Agreement MiSE-CNR "Ricerca di Sistema Elettrico".

REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *8th ICMDM 2007*. IEEE, 2007, pp. 198–205.

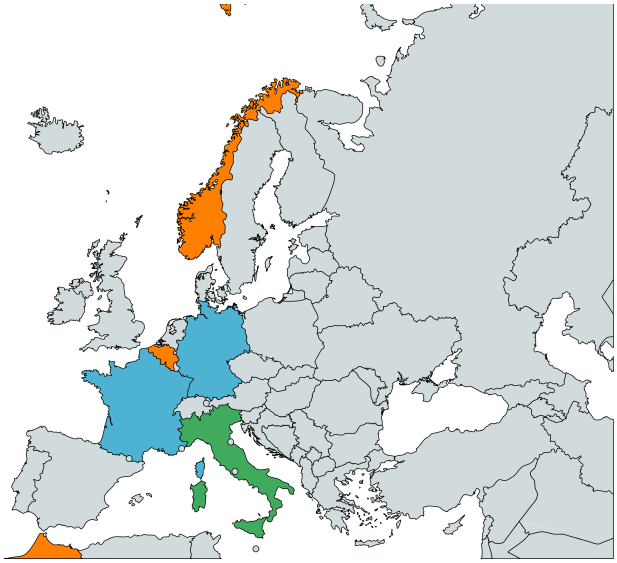

Fig. 6. Distribution of the various types of sensors that are part of the IoT Manager sensing layer. In blue, the countries in which ArLu and Lamp sensors are located. In orange, the countries in which sensors of type TC are deployed. In green (Italy), all types of sensors are located.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[3] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, "Convergence of manet and wsn in iot urban scenarios," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3558–3567, Oct 2013.
[4] G. Borrello, E. Salvato, G. Gugliandolo, Z. Marinkovic, and N. Donato, "Udoo-based environmental monitoring system," in *APPLEPIES 2015*, ser. Lecture Notes in Electrical Engineering, vol. 409. Springer, 2015, pp. 175–180.
[5] L. Calderoni, D. Maio, and P. Palmieri, "Location-aware mobile services for a smart city: Design, implementation and deployment," *JTAER*, vol. 7, no. 3, 2012.
[6] L. Calderoni, D. Maio, and S. Rovis, "Deploying a network of smart cameras for traffic monitoring on a "city kernel"," *Expert Syst. Appl.*, vol. 41, no. 2, pp. 502–507, 2014.
[7] A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi, "Architecture and protocols for the internet of things: A case study," in *8th IEEE ICPCC, PerCom 2010*. IEEE, 2010, pp. 678–683.
[8] A. L. Chan, G. G. Chua, D. Z. L. Chua, S. Guo, P. M. C. Lim, M. Mak, and W. S. Ng, "Practical experience with smart cities platform design," in *4th IEEE WF-IoT 2018*. IEEE, 2018, pp. 470–475.
[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Comp. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
[10] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 112–121, 2014.
[11] S. Latré, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester, "City of things: An integrated and multi-technology testbed for iot smart city experiments," in *IEEE ISC2 2016*. IEEE, 2016, pp. 1–8.
[12] P. P. Ray, "A survey of iot cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1, pp. 35 – 46, 2016.
[13] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.
[14] K. S. Yeo, M. C. Chian, T. C. W. Ng, and A. Do, "Internet of things: Trends, challenges and applications," in *ISIC 2014, Singapore*. IEEE, 2014, pp. 568–571.
[15] A. Zanella, N. Bui, A. P. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.