Atmosphere, an Open Source Measurement-Oriented Data Framework for IoT

Riccardo Berta, *Member, IEEE*, Ahmad Kobeissi, Francesco Bellotti, and Alessandro De Gloria

***Abstract*—The ever more extensive data collection from Internet of Thing (IoT) devices stresses the need for efficient application development tools. State-of-the-art IoT cloud services are powerful, but the best solutions are proprietary, and there is a growing demand for interoperability and standardization. We have investigated how to develop a nonvendor-locked framework, which exploits state of the art data management technologies, and targets effective and efficient development in this article. Focusing on the concept of measurement, we abstracted an architecture that could be applied in a variety of domains and contexts. We tested the framework and its workflow in four use cases analyzing data and enabling new services in health, automotive, and instruction. Our experience showed the benefits of the development tool, which is not tied to a commercial platform, nor requires the huge set-up times needed to start a project from scratch. The tool is released open-source, particularly supporting collaborative research.**

***Index Terms*—Application programming interface (API), autonomous driving, cloud, e-health, e-mobility, Internet of Thing (IoT), NoSQL database, open source, representational state transfer (REST)ful, smart city.**

I. Introduction

WIDE availability of efficient application development tools is key to the success of any digital ecosystem, as it supports a virtual cycle with a community of developers (e.g., [1]). In the Internet of Thing (IoT) ecosystem, data collected from the field fuels a variety of applications (e.g., monitoring, prediction, maintenance, surveillance, etc.) in multiple industrial domains [2]. According to a Gartner's [3] forecast, 25 billion connected things will be in use by 2021, and the share of IP traffic generated by machine-to-machine modules is estimated to rise from 3.1% in 2017 to 6.4% by 2022 [4]. This traffic typically consists of real-time data generated by sensors and IoT devices, bringing a large amount of context information.

Manuscript received February 12, 2020; revised April 8, 2020; accepted May 4, 2020. Date of publication May 14, 2020; date of current version November 20, 2020. This work was supported in part by the European Union's Horizon 2020 research and innovation programme under Grant 723051 and FP7 Transport, GA 605405, and in part by the Italian Government Ministero dello Sviluppo Economico, SCN_00558. Paper no. TII-20-0709. *(Corresponding author: Francesco Bellotti.)*

The authors are with the Department of Electrical, Electronics, and Telecommunication Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy (e-mail: riccardo.berta@unige.it; ahmad.kobeissi@elios.unige.it; franz@elios.unige.it; alessandro.degloria@unige.it).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2020.2994414

Cloud services are needed to support data access and management. These services are developed and maintained by taking advantage of platform-as-a-service frameworks. Despite the use of common components (e.g., databases, application programming interfaces (APIs), protocols), the development and deployment process is challenging and time consuming [5].

Commercial companies (e.g., Amazon, Microsoft, Google) have established efficient IoT ecosystems based on powerful cloud services, but they rely on proprietary technologies, with very limited interoperability and development opportunities for third parties.

In a significant effort to rationalize and further extend the field [6], the IEEE P2413 working group, including industry giants, has recently released—as a draft standard—a cross-domain architectural framework providing a reference model that defines relationships among various IoT verticals (e.g., transportation, healthcare, etc.) and common architecture elements. It also provides a blueprint for data abstraction. A reference architecture covers the definition of basic architectural building blocks and their ability to be integrated into multitiered systems [7].

In this article, we therefore investigate the research question on how to develop a nonvendor-locked, interoperable framework, which exploits state of the art data management technologies, and is able to support effective and efficient development for a variety of relevant IoT applications.

As a fundamental design choice, we focused on the concept of measurement, as this type of data is very common in IoT. This choice delimited the target coverage, but made it easier and more effective the process of abstraction, which is needed in order to support efficient application development in a variety of domains and operational contexts.

The remainder of the article is organized as follows. Section II analyzes the state-of-the-art. Section III concerns system architecture, presenting requirements, design, and implementation. Section IV presents the deployment of the framework in four use cases (three industrial research project and one close-to-market application). Finally, Section V concludes this article.

II. Related Work

The literature offers a wide coverage of IoT frameworks. Cheruvu *et al.* [8] provide a survey categorizing such tools according to a consumer, industrial, or manageability focus. Atamani *et al.* [9] provide a review of several available IoT frameworks and platforms, analyzing such criteria as security, data analytics, and support of visualization.

Powerful solutions are currently available on the market for different types of IoT services. The Amazon Web Services for the IoTs (AWS IoTs) [10] cloud service features architecture modules for data management, device connectivity and control, and analytics and event detection. A stack of functions is available on the back-end: DynamoDB, Kinesis, Lambda, S3, SNS, SQS, etc. Since AWS is a cloud service provider, multiple data processing services are already integrated. Application integration in the platform is supported through various messaging and queing services. AWS IoT does not distinguish among sensors, actuators, and devices, as it focuses on the concept of things. However, there are limited custom attributes for things, which challenges manageability and increases latency [11]. The Microsoft Azure cloud platform [12] enables developers to create cloud-based programs using a software as a service commercial platform. Azure Sphere contains tools for edge software development kits (SDKs) [13] to enable secure edge-to-cloud connectivity. Other commercial frameworks [14], like Google Cloud and Bluemix, have a variety of IoT cloud services. Despite their differences in performance and efficiency in practice, Zúñiga-Prieto *et al.* [5] indicate long deployment times as a shared issue among such frameworks. We argue that a better-focused content structure could facilitate developers, especially for the applications dealing with measurements.

Several IoT data frameworks have been presented in the literature to deal with specific IoT application domains. Recent examples concern forensics [15], smart homes [16] and smart cities [17]. In a more general approach, Jiang *et al.* [18] proposed a framework dealing with typical IoT challenges (large volume of data, different data types, rapid generating data, complicated requirements, etc.). For structured data, they propose a database management model that combines and extends multiple databases and provides unified access APIs. For unstructured data, the framework wraps and extends the hadoop distributed file system based on the file repository model to implement version management and multitenant data isolation. A resource configuration module supports static and dynamic data management in terms of the predefined meta-model. Thus, data resources and related services can be configured based on tenant requirements. More recently, Cai *et al.* [19] presented a functional framework that identifies the acquisition, management, processing and mining areas of IoT big data, and several associated technical modules are defined and described in terms of their key characteristics and capabilities. Fu *et al.* [20] deal particularly with IoT data storage efficiency and security, proposing a framework that keeps time-sensitive data (e.g., control information) on the edge and sends the other (e.g., monitoring data) to the cloud.

Similarly to Atmosphere, [21] proposes a framework, which supports developers in modeling smart things as web resources. The framework supports resource-type definition and design, general-purpose software for operations on web resources, a mapping between web resources and data sources, and programming and publishing tools. User evaluation concerned the use of InterDataNet (IDN)-studio (a graphical interface web application for designing and managing web resources) to customize and enhance the representation of a Point of Interest within the mySmartCity application, and evaluate its rendering performed by IDN-viewer. Sharma and Wang [22], single out four main characteristics of IoT data in cloud platforms: multisource high heterogeneity, huge scale dynamic, low-level with weak semantics, inaccuracy. These characteristics are important, as they highlight key features that should be provided by an effective IoT data framework (e.g., source characterization, variety of source data configurations/aggregation, outlier computation [23]), that we kept into account in the design of Atmosphere.

III. System Design and Implementation

This section presents the design and implementation process, starting with requirements up to the supported user workflow.

A. Requirements

Based on the literature analysis and our experience in industrial research projects (e.g., [24]) we elicited a need for an IoT domain-independent data framework to support efficient development of IoT applications dealing with measurements. The solution should support easy problem and context modeling and facilitate collaboration between developers, customers, and stakeholders. Once the data model is defined, developers should be able to quickly configure the framework accordingly. After the deployment, the edge devices should be able to upload data, and (third-party) applications to seamlessly access them.

From an architectural point of view, requirements concern scalability (in terms of both data size and number of remote connections), ease of deployment, preservation of data integrity, large reusability, standardized IoT terminology cross-domain, and user management.

B. Platform Choices

Based on the above requirements, we opted for designing a framework leveraging cloud principles to support scalability and ease of deployment, but without locking into nonportable proprietary technologies. The framework integrates APIs implementing representational state transfer (REST) services [25], that provide a platform-independent hypertext transfer protocol (HTTP) interface (e.g., [13]). A generic web service accommodates numeric-type data in compliance with REST guidelines in a reusable fashion. A RESTful API separates the user interface (UI) from the server and data storage, which improves portability, scalability, and independent development. For data storage, we adopted a document-based database management system (DBMS), such as MongoDB. NoSQL databases provide a series of features that relational databases cannot provide, such as horizontal scalability, memory, and distributed index, dynamically modifying data schema, etc. [26]. MongoDB also supports sharding, a method for distributing data across multiple machines. A certain difficulty in coding complex queries brings up the cost of such a choice. However, this is hidden to the user of the framework, who access data through predefined routes to the modeled resources, according to the REST API design principles [25]. REST supports the mapping of HTTP verbs (GET, POST,

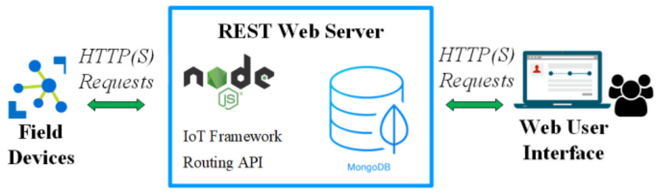


Fig. 1. Atmosphere high-level block diagram.

PUT, DELETE) to the classical CRUDdatabase actions (create, read, update, delete).

Unlike relational DBMSs, MongoDB does not impose the prerequisite of defining a fixed structure. Models in MongoDB allow hierarchical relationships representation, with the ability to modify the structure of the record. Furthermore, MongoDB recognizes data in javascript object notation (JSON) [27], a natural JavaScript format, which means that no conversion is required on a Node.js server. JSON facilitates the exchange of data between web apps and servers in a compact and human-readable format. Fig. 1 depicts the high-level block diagram of the system. Node.js provides the advantages of full stack JavaScript development [28]. A usual server-side approach (e.g., in PHP, ASP.net, Ruby and Java) involves multithreading. Node.js avoids the multithreading burden by employing a nonblocking single-thread pattern and is able to efficiently serve multiple concurrent clients by operating asynchronously, employing the event-loop mechanism [29], which is particularly suited for microservices architecture [30].

C. Modeling

Atmosphere was designed to represent the target context and its elements as interrelated software objects, onto which to build applications. These objects are modeled as resources, with their own schemas and functionalities, accessible through the API routes. Defining resources to expose the interface is thus a key design step and requires abstraction in order to support flexibility, extendibility, and scalability. Not only do the choices concern the terminology, but also the semantic of each resource.

We identified five essential resources: *thing*, *feature*, *service*, *device*, and *measurement* (see Fig. 2 and 3). A *thing* represents the subject of a *measurement*. a *feature* describes the (typically physical) measured quantitity/ies. Each quantity in a *feature* is an *item*, with a name and a unit. A *Service* denotes the type of the target IoT deployment. A *device* is a tool providing measurements regarding a *thing* (or an actuator that acts within a thing to modify its status). A *measurement* represents a sample of a *feature* measured by a *Device* for a specific *thing* in the context of a certain *Service*. Other resources include: *alert*, *user*, *provider*, *subscription*, *log*, *login*, *script*, *tag*, *constraint*, and *computation*.

The concept of *measurement* abstracts the samples posted to and retrieved from the database. The structure of a *measurement* must match its *feature*. A *measurement* can contain one or more homogeneous *samples*, the latter being the typical case when sampling signals (see Fig. 4). Each *sample* contains a vector of *values*. *Values* are not necessarily homogeneous. For instance, a *sample* could represent a set of statistical information on a quantity (e.g., average, stdev, etc.). Each *value* can be a scalar (e.g. a temperature), a vector (e.g. the orientation in space) or a tensor of numbers (e.g., general multidimensional data points). The *feature* resource is used to check the integrity of each received *measurement*. The size of each *mesasurement sample value* must match the corresponding *dimension* attribute stored in the corresponding *feature item* (see Fig. 2).

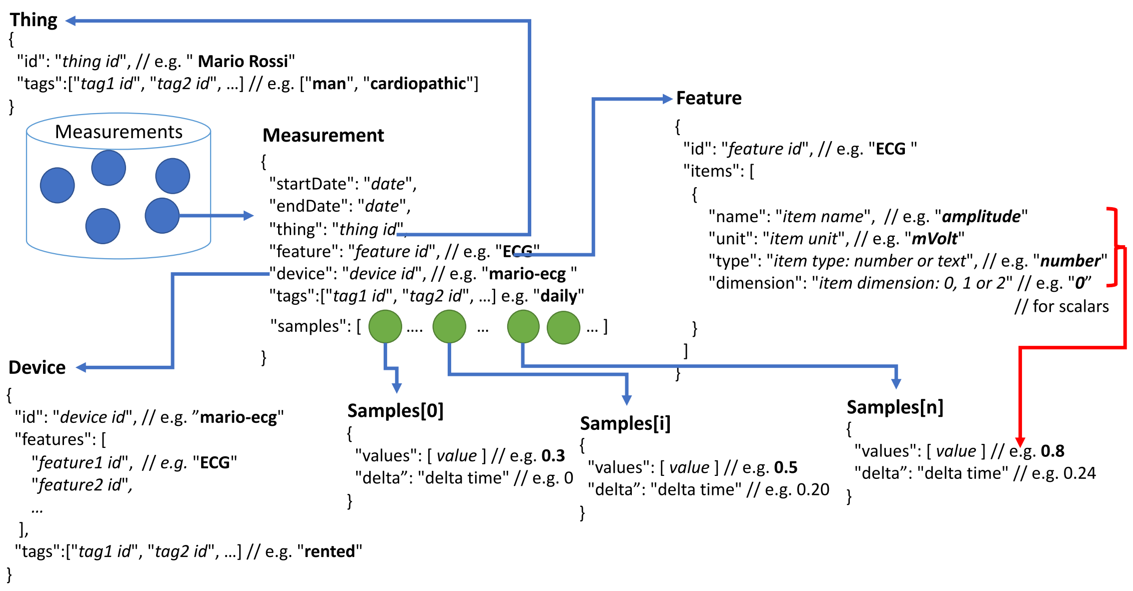
We also defined the *computation* resource, which performs postprocessing calculation on measurements, exploiting the cloud server capabilities. The result of a *computation* is structured and stored as a *measurement*, thus allowing further processing. A set of (typically statistical) computation types are available, identified by the "code" attribute. Currently, the following codes are supported: *stat* (including maximum, minimum, average, median, standard deviation, variance), *quartiles* (first quartile, third quartile), *histograms*. A custom computation is also available, which executes a custom script uploaded by the user. In the case of *stdev* and *var*, the engine offers two variations: population-based and sample-based [31]. Another type of computation concerns outlier detection, which is key to guarantee the quality of data series.

Another abstraction that we defined to support the possible needs of a data consumer service, is the *constraint*, which allows defining relationships between different resources. We modeled this as a resource in order to support the maximum flexibility in adding associations/dependencies between resources while avoiding hard-coding them inside the resources themselves. As a use case, this allows building a drop-down menu in a UI which is filled with its proper options by dynamically querying the database (DB).

D. Implementation

The framework guarantees that all its exposed resources can be manipulated through the previously mentioned HTTP methods. Standard response codes were defined for each method like success codes (2xx) and client error codes (4xx), with suited information in the response message. We have implemented the RESTful API services in Node.js (JavaScript-based) within the Express.js framework. This open-source framework offers easy integration of third-party middleware, particularly the MongoDB database and its Mongoose persistence layer.

In storage, resources correspond to collections in the database. Each resource involves a number of fields, some of which are mandatory, while others are optional. Within the API, resources are implemented through two stages: schema and controller. The schema—which is needed for data-checking, given the schemaless nature of MongoDB—defines the resource structure, while the controller implements the resource functionality. A resource schema prescribes the fields, including their types, default values, and references to other resource fields. Fields that refer to other resources have cross-validation functions implemented within the resource schema. The schema also includes plugin definitions as well as indexing options. The controller defines the HTTP methods that are supported by the resource (typically: GET for fetching resources, POST for inserting, PUT for updating, DELETE for removing, either permanently or softly).



```
Thing
{
  "id": "thing id", // e.g. " Mario Rossi"
  "tags":["tag1 id", "tag2 id", ...] // e.g. ["man", "cardiopathic"]
}

Measurement
{
  "startDate": "date",
  "endDate": "date",
  "thing": "thing id",
  "feature": "feature id", // e.g. "ECG"
  "device": "device id", // e.g. "mario-ecg "
  "tags":["tag1 id", "tag2 id", ...] e.g. "daily"
  "samples": [ .... ... ... ]
}

Feature
{
  "id": "feature id", // e.g. "ECG "
  "items": [
    {
      "name": "item name",  // e.g. "amplitude"
      "unit": "item unit", // e.g. "mVolt"
      "type": "item type: number or text", // e.g. "number"
      "dimension": "item dimension: 0, 1 or 2" // e.g. "0"
                                                // for scalars
    }
  ]
}

Device
{
  "id": "device id", // e.g. "mario-ecg"
  "features": [
     "feature1 id",  // e.g. "ECG"
     "feature2 id",
     ...
  ],
  "tags":["tag1 id", "tag2 id", ...] // e.g. "rented"
}

Samples[0]
{
  "values": [ value ] // e.g. 0.3
  "delta": "delta time" // e.g. 0
}

Samples[i]
{
  "values": [ value ] // e.g. 0.5
  "delta": "delta time" // e.g. 0.20
}

Samples[n]
{
  "values": [ value ] // e.g. 0.8
  "delta": "delta time" // e.g. 0.24
}
```

Fig. 2. Atmosphere resources outlook.

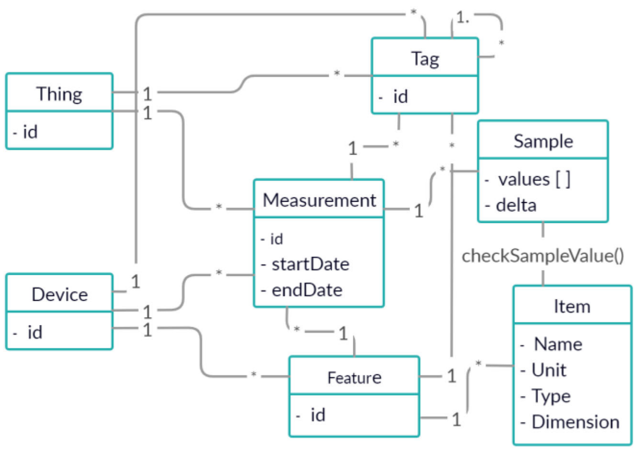


Fig. 3. UML metamodel of the main classes of Atmosphere.

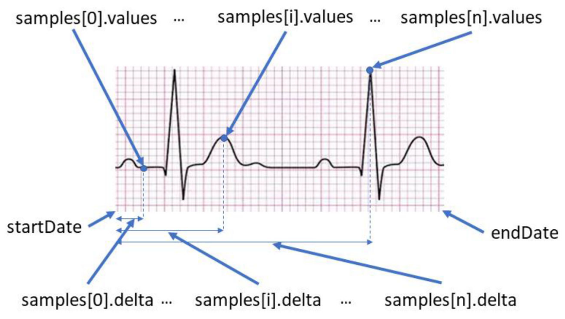


Fig. 4. Measurement and Feature match.

The computation controller performs incremental calculations in order to avoid exhausting the system's resources (e.g., memory), as computations are typically performed on huge quantities of data. Computations are obtained in a two-step process, where the client first issues a computation request, which opens a WebSocket through which the client can get information from the system about the progress of the execution.

According to the best practice in software engineering, the framework includes an automatic test suite for all routes and methods, supported by the javascript package manager. This is key to ensure a correct working of the API operations.

E. Working System

When the API is first initialized, it connects to the storage server and creates the database "atmosphere-DB" with one collection inside. That collection is the "users" collection, and it is essential to have one admin user in order to create other users and other collections. This first instance of a user is predefined within the source code. The "admin" user can create two other types of users: "provider" and "analyst." While the "admin" has full access to the DB (all methods unrestricted—including those for managing the application DB (ADB), which is described in the next sub-section), the"provider" user has restricted access that depends on ownership (allowed to POST measurements and to GET only those records that are owned by the user), and the "analyst" user can only access measurements (GET methods). The authorization mechanism supports also a finer grain control, as described in the L3Pilot use case. Another security measure is authentication. It is based on a JSON web token (JWT), a security pass that expires after a configurable timeout. To get a JWT, users must POST to the "login" resource with their given credentials. The API will reply with a JWT with the specific authorization level of the requesting user. The JWT must then be used as a header attribute for any further requests to the API.

F. Supported Workflow

Atmosphere has been designed in order to support an efficient workflow for preparing different measurement-based data-rich applications. The first step consists of the *domain modeling*, where the field objects are mapped to the Atmosphere API's resources. In this phase, the IoT application designer has to define *features* (i.e., types of measurements), *devices* (i.e., measurement instruments), *tags* (i.e., labels that can be attached as attributes to other resources— typically measurements, features, things and tag themselves), and *constraints* (i.e., relationships between elements in the DB). In the *configuration* (or *deployment*) step, the above designed model resources are straightforwardly encoded in a *.json* file (e.g., see fragments in Fig. 2), which is POSTed to the framework APIs, so to create the ADB structure. In the *regime phase*, the framework manages the ADB, allowing dynamic insertion/update of users, things, field measurements and computation requests; and retrieval of results in terms of things, measurements and computation outcomes. The ADB structure can be updated during the operation as well, by POSTing/PUTting/DELETEing features, devices, and tags. All these actions happen only through the exposed resource routes, with the well known advantages of the RESTful approach in terms of scalability, encapsulation, security, portability, platform independence, and clarity of terminology and operations.

IV. Use Cases

Atmosphere has been employed in three industrial research projects, one in the health/smart home sector and two in automotive that has put Atmosphere in challenging experimentation set-ups, with different settings. The projects have also provided the opportunity to improve and extend the framework. We also report on a case applying the system for a closer to market application for health-care instruction. In all cases, Atmosphere was deployed on a commercial cloud server, namely an AWS elastic cloud computing (EC2) T2.micromachine (1 virtual CPU, 1 GB Ram) hosting Linux OS, without exploiting proprietary-specific APIs nor services.

A. Health at Home (H@H)

H@H, a domestic project funded by the Italian Ministry of economic development and aimed at supporting the elderly with chronic health failure [32], developed a complete IoT cloud service consisting of the home monitoring sensor system (front-end), the home gateway (middleware), and a remote cloud (back-end). Through the gateway, several physiological quantities (electrocardiogram signal, heart rate, breathing waveform, breathing rate, oxygen saturation, blood pressure, glycemia, etc.) are collected and delivered to the cloud. Through a web UI, a clinician can view the measurements, and modify the pharmacological therapy according to the symptoms. Atmosphere acted as the backbone of the application in order to implement the H@H API described in [33]. Atmosphere had to accommodate the needs and usage variations of different service providers, as the front-end (edge) and the gateway (fog) were provided and maintained by third parties in the health industry.

The mapping of the H@H quantities onto the Atmosphere resources was straightforward. For each physiological signal, we create a corresponding "feature" and for each sensor a "device," in order to collect information acquired on patients as "measurements" in Atmosphere. The mapping between H@H concepts and Atmosphere resources is given in Table I. The "operator" entity, a new instance of the user resource, was required, with a special set of behaviors and permissions. We implemented this as an extension to the main "user" resource exploiting the object-oriented programming paradigm.

In H@H, several service providers offer various e-health services (e.g., postsurgery rehabilitation support, daily activity monitoring, pain self-assessment, etc.). This required a management framework to organize the services on both the patient and the provider side. We thus implemented a publish/subscribe

TABLE I
Atmosphere Resource Mapping to Project Models

Resource	H@H data	Fabric data	L3Pilot data
Measurement	A record containing the value of a medical indicator (e.g., breath rate, heart rate, body temperature)	A record containing the vehicle-coil alignment estimation, or the samples of the charging process	A record containing all the samples of a Driving Scenario Instance (DSI) Datapoint, or of the Performance Indicators (PI) per Trip, or of the PI per SI
Feature (type of a Measurement)	A single medical indicator. For instance: breath rate, heart rate, body temperature, etc.	Measured quantity. For instance: vehicle-coil displacement, charging parameters (with dimensions: current, voltage, speed, etc.)	Measured quantity (e.g., Trip PI, DSI Datapoint). Dimensions include: average speed, time headway at maximum speed, percentage time in each type of driving scenario, etc.
Device (Measurement tool)	Different types of medical devices (e.g., heart pulse sensor, blood pressure sensor)	Vehicle-coil alignment system, Dynamic Wireless Charger (DWC) road and vehicle side	The L3Pilot data toolchain
Thing (subject of a measurement)	Patient	Passage of an electric vehicle in the charging road lane	Trip (i.e., the unit of the information source processed by the source data toolchain)
Tag (attribute of a measurement)	-	Charging conditions (e.g., preparing, charging, fault)	Driving scenarios (e.g., traffic jam, cut-in), road types (e.g., urban, motorway), experimental conditions (e.g., baseline), UI components (e.g., drop-down menus)
Constraint	Association between things and services	Association between features and services	Association between tags and features.
Computation	Outlier detection, Stats	Outlier detection, Power, Energy	Outlier detection
Service	Specific health monitor services (e.g., post-surgery rehabilitation, monitoring)	Vehicle-coil alignment, Wireless charging, Billing, Lane keeping	Data analysis

(pub/sub) pattern, through a supplementary *subscription* resource. Atmosphere takes advantage of asynchronous WebHooks to provide automated real-time callbacks. WebHooks broadcast any service update by the service providers to all subscribers. The new resource couples the users with their subscribed services (the ones that are supported by their installed edge sensory system). One field (IsActive) specifies whether the payload delivery is enabled, while another one (endpoint) specifies the uniform resource locator address to which the service and later updates must be published.

H@H has been successfully piloted in Oderzo, a small town in Veneto Region, where 13 older persons living in 8 apartments have been monitored using different sensors (e.g. passive infrared (PIR) sensors and thermostats in several rooms), and measurements stored in an Atmosphere ADB. A second demonstration is ongoing in the facilities of Fondazione Don Gnocchi, one of the largest non state-owned hospital infrastructure in Italy.

B. Fabric

In fabric, a seventh Framework Programme European Industrial Research Project which implemented an on-road testbed for dynamic wireless charging (DWC) of electrical vehicles [34], we realized a charging process metering service for the vehicles passing through the charging lane [35]. The system senses and computes on the edge information about the charging process and stores it on Atmosphere's cloud server to support new electro-mobility (e-mobility) services (e.g., billing, energy-aware car navigation) which can be implemented by relevant companies (e.g., energy providers, navigation providers).

The road-side DWC subsystem sends to Atmosphere data representing the state of each consecutive charging grid. The vehicle-side DWC subsystem generates a stream of measurements recording the vehicle-coil alignment (which is key to power transfer efficiency and was supported by a vision system [36]) and the charging parameters and battery status. In an approach suitable to the nature of each data stream, we implemented the edge-computing module to process data in two different ways. The road-side uploads to the cloud when a change in measured samples occurs. The vehicle side averages the data within a predefined period. The edge processor also included a local buffer to deal with an intermittent connection between the edge and the cloud.

The stored samples represent the electrical charging process in current and voltage. Fig. 4 shows the measured samples of the current in ampere of a charging session. The mapping of the Fabric objects onto the Atmosphere resources is given in Table I. Further parameters are required to contribute to the metering and billing services such as the electrical power transfer, energy stored, and final electrical bill. Acquiring these data was made possible through the *computation* resource. Power transfer is computed directly from the stored measurements. But computing energy requires integrating the result of previous power computations, that are stored as measurements.

The whole end-to-end deployment was tested in-lab and (the vision module for vehicle-coil alignment) on the test site inside the MotorOasi Piemonte safe drive track in Val di Susa, Italy. Through enabling edge computing functionalities, the edge device managed to drop data upload size by approximately 96% from 720 to 27 MB/h. Based on stored data, we performed tests in lab tests for a simple prototype billing service. In a three-step procedure, starting with power (KW), then computing energy (KWH), we were able to obtain an estimate of the electrical bill for a specific charging lane passage. In all cases, concurrent and consecutive HTTP requests were maintained at a stable 40 ms delays in response time on a low-end AWS EC2 T2.micromachine.

C. L3Pilot

L3Pilot is a Horizon 2020 research project aimed at assessing the impact of automated driving (AD) on public roads, testing the society of automotive engineers level 3 (and some level 4) functions [37]. The pilots systems are being exposed to variable conditions in hundreds of trips on vehicles by 13 owners (ten original equipment manufacturers, two suppliers, one research facility), across ten European countries. The project uses the field operational test support action methodology [38], driven by a set of research questions and hypotheses on technical aspects, user acceptance, driving and travel behavior, as well as the impact on traffic and safety. In order to answer such research questions, the project has developed a data toolchain, that translates the proprietary vehicular signals to a shared format [39], and processes them to estimate the driving scenarios (e.g., "lane change," "cut-in") [40]. Data from all the pilot sites is now being analyzed for an overall impact assessment. Atmosphere provides the shared data storage back-end [41].

The mapping of the L3Pilot objects onto the Atmosphere resources is given in Table I, with a *thing* (i.e., the subject of a *measurement*) corresponding to every single experimental trip. Stored measurements—produced by the abovementioned toolchain—are not the original signal time-series, but meaningful aggregations (called "Datapoints" and various types of "indicators," with such items as: avg speed, minimum longitudinal distance, time headway at minimum time to collision, percentage of driving times in the various scenarios, etc.). Thus, such "datapoints" and "indicators" are the *features* in the L3Pilot installation. Different driving scenario types have different datapoint structures. All these features are tagged as datapoint to facilitate data retrieval according to the jargon. Tags were very useful also to specify driving scenarios, experimental conditions (baseline, system available, system active, etc.) and road types (e.g., motorway, major arterial, local road, etc.), that are all used to segment a trip's data.

The *constraint* resource was introduced to define abstract relationships (e.g., dependencies) between two documents in the DB. The web-browser-based UI— that was developed by another project partner [40]—exploits *constraint* documents in order to allow the data analyst user to select the available measurement types from dynamically filled drop-down menus. To this end, constraints were used for relating tags among each others. For instance, the "datapoint" UI tag, that is one of the items selectable for querying measurements, points to a group of actual features (that finally identify the measurements), one

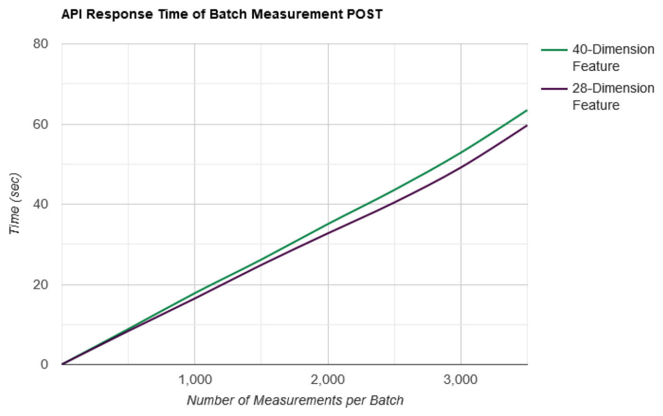


Fig. 5. Response times to batch uploads with two types of measurements (one with 28 and one with 40 dimensions) in the L3Pilot Atmosphere ADB.

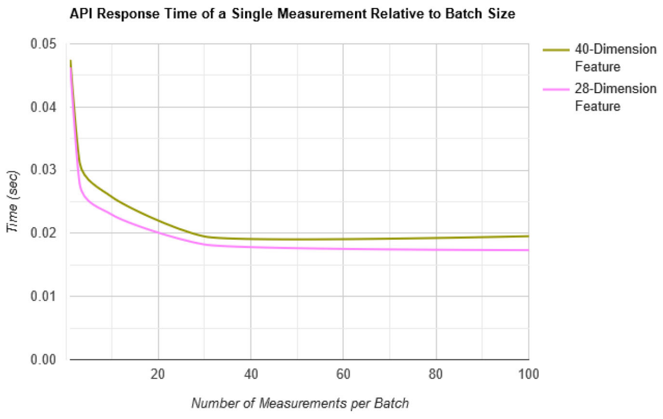


Fig. 6. Response times per single measurement relative to different sizes of batch uploads (range 1 to 100) in the L3Pilot Atmosphere ADB.

for each driving scenario. Other UI tags identify point to other groups, according to the logic of the application. This allows automatic creation of hierarchical query forms.

The L3Pilot toolchain processes offline the data gathered during a pilot vehicle's trip and POSTs the resulting measurements in batches to the DB. Batch sizes differ from one trip to another, and we monitored the effect of batch sizes on the API response time. As Fig. 5 shows, the response time is almost linear with respect to the number of measurements per batch. The size of the sample vector (feature's items) has a minor impact. Examining the effect of the batch approach on the single measurement response time, we observed a significant improvement in response times down to 17 ms on batches with a larger number of measurements. Fig. 6 shows a decline in response time, which settles at batches with 30 measurements. The addition of the support of HTTPS had only a minor impact, even while enabling SSL certificate verification. We observed a +/- 4 ms difference at most between encrypted and unencrypted requests, with a slight peak of 1% in CPU load upon HTTPS connection, on our AWS EC2 T2.micromachine.

To POST measurements to Atmosphere, vehicle owners are given "provider" credentials, through which they can also acces their own measurements only. Analyst users were initially granted access to all measurements. Then, to better protect confidentiality, the consortium required to restrict the analyst access, requiring a finer grain right management. Three groups of features were identified, and analysts are now given read access rights to these single groups.

The L3Pilot ADB is now being used by the analysts of all the pilot sites, that have started processing data inserted by the 13 vehicle owners to answer the project's research questions on L3 autonomous driving. The last requirement we elicited has been the need to implement the text/csv multipurpose internet mail extensions type, so to reduce the message size and speed-up the file preparation on the client side (analysts typically work on .csv files obtained by downloading the result of a query).

D. Emergency Room Educational Simulator

While the three above presented use cases concern industrial research prototypes, we also tried to check the system as is for commercial applications. To this end, we organized a presentation of Atmosphere with the development team of a university spin-off company engaged in rich-data applications. It took the developers two days to come up with an ADB model implementation for the use case of a three-dimensional virtual reality simulator from emergency room personnel instruction. The very limited time-frame required for designing and deploying an effective solution was considered a key merit. Atmosphere was used to achieve the goal of assessing the performance of a doctor by evaluating the effects of his interventions on the various patients. To this end, the ADB was designed to record the state of a patient, that is characterized by a set of time-evolving parameters; the actions performed by the doctor (e.g., how he interacted with the available simulation tools); and the events in the simulation (e.g., changes in medical equipment availability). To speed up the queries and facilitate the post-processing, measurements should be recorded both as and as blobs aggregating data with the same time. Time series data will then be retrieved to compute correlations and sophisticated machine learning so to identify relationships between the doctor's avatar actions and the evolution of the patients.

E. Analysis of Results

The deployment of Atmosphere in the above presented diverse IoT applications showed its effectiveness, flexibility, and ability to support an efficient workflow. The system was used also by third parties (universities, research institutes, and companies), with satisfaction. Additionally, we were able to progressively integrate new functionalities in an abstract manner, keeping the reusability objective valid across all resources and methods. Table I summarizes the mapping between Atmosphere and the three project models.

While the other two projects are finished, L3Pilot is in progress, and a full account of Atmosphere's deployment will not be available before the project concludes in 2021. However, requirements by L3Pilot partners and pilot tests have been satisfied efficiently by Atmosphere, which showed a great deal of versatility, also considering that we dealt with statistically preprocessed data, with complex semantic structures. This challenged our design, and required some improvements, for instance to allow different dimension sizes for each value in measurement samples, as L3Pilot datapoints, that are recorded at each detected driving scenario instance (DSI), contain quite different preprocessed quantities. When upgrades were needed, we managed to keep the deployment delay upon structure change relatively low (a couple of days at most). L3Pilot allowed testing and upgrading Atmosphere to the use case in which different IoT data source providers share postprocessed data, typically in batches. The structural data checks implemented in Atmosphere allowed detecting bugs in the complex data preparation toolchain (particularly in terms of feature value dimensions), which saved significant development time.

Across the four use cases, not only did the deployment environment change, but the manner of data streaming to the cloud as well. In H@H, we experienced a steady stream of raw data, in fabric uploads contained high frequency of aggregated data, and in L3Pilot we implemented batch upload of preprocessed data. Upon examination of each use case, we extracted the relevant parameters that give an insight into the efficacy of Atmosphere's deployment. We spotted a remarkable difference in API response time to stream POSTs (one measurement with a single or multidimension value vector) versus batch POSTs. Stream POST requests resulted in an average 40 ms response time on our low-end AWS EC2 T2.micromachine. While such latency was suitable for our use cases, other types of workload and applications (e.g., streaming for real-time remote control) would need much more powerful servers than our low-end T2.micromachine, and/or exploiting computation scaling functionalities, such as MongoDB sharding. Batch POSTs contained up to 4000 measurements, with multiple dimension (5 to 40) value vectors. Fig. 5 and 6 show the response times of two batch uploads to Atmosphere, with different dimensions. Each batch POST costed around one minute, for 3500 measurements with 40-sized value vectors, resulting in a per measurement cost of 17 ms on average. We argue that the reason for this contrast in response time per measurement between stream and batch POSTs is partly because of the enabling of the response compression using the "npm compression" middleware and partly because of the overhead avoidance of HTTP request/response header processing upon batches.

F. Lessons Learned

The most apparent lesson we learned from our use case experience is that the framework must be easy to deploy and use. We also prepared a Docker image, which greatly simplifies the process. Creation of an ADB (i.e., configuration of an Atmosphere server instance for an application) can be automated through a Postman [42] script, which offers a turnkey solution for deployment, that was particularly appreciated in L3Pilot, where the ADB had to be deployed in several pilot sites for preliminary local analysis.

Postman is an excellent tool for testing as well. But actual deployment required us to develop tailored tools to support insertion and query of measurements. In fact, while, the abstract design of Atmosphere allowed its deployment in a variety of contexts, an effective usage required some domain specific customizations. Particularly, we have developed web UI and command line interfaces, that introduce a business layer between the user and the cloud platform (e.g., processing of input files, dealing with possible duplicates and missing documents, periodic back-up, provision of messages customized to the specific application). These tools are project-tailored, but their porting between different ADBs is relatively straightforward, greatly benefiting from the Atmosphere abstractions. Expressing customizations with domain-specific languages is left as a future work [43].

Collaboration with the client application development teams is key, particularly in the ADB design phase. To this end, documentation and examples are necessary to present the Atmosphere abstractions, that are powerful, but need to be understood. Particularly important are the concepts of samples, for time series (nontime-series have a single sample), and of values (inside a sample), each one of which can have different dimensions (scalar, vector, and tensorial). This gives a high flexibility in accommodating different types of data collections and organizations. Tags and constraints are useful to define lightweight relationships among resources, without hard-wiring them in the resources themselves, which would unnecessarily stiffen and weigh down the software and model structure. Tags are particularly important in supporting the automatic building of query forms for UIs.

Our experience showed the importance of understanding developers' needs. If they require new product features, solutions should be designed to be generic (i.e., application independent), inline with the abstractions that make the framework flexible and seamlessly usable in different contexts. Generally speaking, frameworks should not be "opinionated." They should support a clear workflow, but not prevent users from performing other desirable actions. The Atmosphere design aimed at achieving flexibility, within a focus on the concept of measurement. As a consequence, for instance, measurement features are very flexible, and fine grain user rights are available for measurements, while access to other resources (e.g., the ADB configuration resources) is reserved to an admin.

An advantage stressed by all the developers we have worked with is that, using Atmosphere, they could customize and exploit a reliable system, which was already tested not only through automatic testing, but also by other developers, in the pervious project deployments. We consider this as an important acknowledgment of the robustness of the system and, overall, validity of its design abstractions.

V. Conclusion

As IoT technologies are increasing the capabilities of collecting huge quantities of data from the field, it was ever more important to have tools for creating new, data-rich applications. In this article, we investigated how to develop a nonvendor-locked, interoperable framework, which exploits state-of-the-art data management technologies, and was able to support effective and efficient development of relevant IoT applications. We developed Atmosphere, a cloud-platform independent framework for managing smart things in the IoT ecosystem. Our original contribution consists in an edge-to-cloud computing model using abstract IoT web resources. Atmosphere was designed as a deployment-ready IoT data storage and computation support service, which was not locked into proprietary cloud technologies. It focuses on measurement data and exposes resources (including *Computations*) to support measurement-rich applications

As our target users were mainly IoT developers and service providers, and in order to support the IoT developer community, we have released Atmosphere open source on GitHub: https://github.com/Atmosphere-IoT-Framework. We believed that the tool could be particularly useful for shared research. Our experience in three industrial research collaborative projects (use cases 1 to 3)—quite various in nature—showed that Atmosphere can seamlessly support a variety of IoT applications, providing benefits in terms of efficiency and effectiveness, as its resources support a structured and modular approach to application modeling and development. Atmosphere does not tie the development to a proprietary commercial platform, nor requires the huge set-up times needed to start from scratch a solution. Also based on the feedback from the fourth, closer-to-market, use case we estimated that Atmosphere has a technology readiness level between 7 and 8.

Atmosphere was designed to support an efficient application development workflow starting with domain modeling, where the field objects are to be mapped to the Atmosphere API's resources. In the configuration step, the designed model resources were POSTed to the framework API, so to create the ADB structure, that can be efficiently deployed in the cloud or in a local facility. In the regime phase, the framework manages the ADB, allowing to insert and retrieve measurements and computations. The system implements a rigorous RESTful architecture, exploiting its well-known advantages. The tests indicated that Atmosphere supports also the start up of new generation e-mobility data-driven services for metering and energy-awareness.

Future works will involve enriching the existing set of computations, also with machine learning processing, for instance for time-series prediction and automated clustering. Moreover, we would like to integrate psychometric measurements and user survey/questionnaire data, for supporting user acceptance studies in the field. Finally, it will be interesting to set up an empirical interventional study to assess the causal impact of usage of Atmosphere by IoT application developers.

Acknowledgment

The sole responsibility of this publication lies with the authors. The authors would like to thank all the project partners for their cooperation and valuable contribution.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet Thing J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[2] L. D. Xu, "Enterprise systems: State-of-the-art and future trends," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 630–640, Nov. 2011.

[3] Gartner, *Gartner Identifies Top 10 Strategic IoT Technologies and Trends*. Barcelona, Spain, Gartner Press Reslease, Nov. 7, 2018.

[4] M. Cooney, *Cisco Predicts Nearly 5 Zettabytes of IP Traffic Per Year by 2022*, Netw. World Press Release, Nov. 18, 2018. [Online]. Avialble: https://www.networkworld.com/article/3323063/cisco-predicts-nearly-5-zettabytes-of-ip-traffic-per-year-by-2022.html. Accessed on: May 29, 2020.

[5] M. Zúñiga-Prieto, J. González-Huerta, E. Insfran, and S. Abrahão, "Dynamic reconfiguration of cloud application architectures," *J. Softw., Pract. Experience*, vol. 48, pp. 327–344, 2016.

[6] V. P. Singh, V. T. Dwarakanath, P. Haribabu, and N. S. C. Babu, "IoT standardization efforts — An analysis," in *Proc. Int. Conf. Smart Technol. Smart Nation SmartTechCon*, 2017, pp. 1083–1088.

[7] "*Standard for an Architectural framEwork for the Internet of Things (IoT),*" IEEE P2413, 2019. [Online]. Avialble: http://grouper.ieee.org/groups/2413/

[8] S. Cheruvu, A. Kumar, N. Smith, and D. M. Wheeler, "IoT frameworks and complexity," in *Proc. Demystifying Internet Things Secur.*, 2020, pp. 23–148.

[9] A. Atmani, I. Kandrouch, N. Hmina, and H. Chaoui, "Big data for internet of things: A survey on iot frameworks and platforms," in *Advanced Intelligent Systems for Sustainable Development* (Lecture Notes in Netw. Syst.), vol. 92. M. Ezziyyani, Ed. Cham, The Netherelands: Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-33103-0_7

[10] Amazon Web Services AWS IoT, 2019. [Online]. Available: https://aws.amazon.com/iot/solutions/industrial-iot/?nc = sn&loc = 3&dn = 2

[11] W. Tarneberg, V. Chandrasekaran, and M. Humphrey, "Experiences creating a framework for smart traffic control using AWS IOT," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, New York, NY, USA, 2016, pp. 63–69.

[12] Azure IoT, Microsoft, Redmond, WA, USA, 2019. [Online]. Available: https://azure.microsoft.com/en-us/overview/iot/

[13] S. Jiong, J. Liping, and L. Jun, "The integration of azure sphere and azure cloud services for internet of things," *MDPI J. Appl. Sci.*, vol. 9, no. 13, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/13/2746

[14] T. Pflanzner and A. Kertesz, "A survey of IoT cloud providers," in *Proc. 39th Int. Conv. Inf. Commun. Technol., Electron. Microelectron.*, 2016, pp. 730–735.

[15] H. Chi, T. Aderibigbe, and B. C. Granville, "A framework for IoT data acquisition and forensics analysis," in *Proc. IEEE Int. Conf. Big Data*, Seattle, WA, USA, 2018, pp. 5142–5146.

[16] J. Jung, K. Kim, and J. Park, "Framework of big data analysis about IoT-Home-device for supporting a decision making an effective strategy about new product design," in *Proc. Int. Conf. Artif. Intell. Inf. Commun.*, Okinawa, Japan, 2019, pp. 582–584.

[17] Ş. Kolozali *et al.*, "Observing the pulse of a city: A smart city framework for real-time discovery, federation, and aggregation of data streams," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2651–2668, Apr. 2019.

[18] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An IoT-oriented data storage framework in cloud computing platform," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1443–1451, May 2014.

[19] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: perspectives and challenges," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 75–87, Feb. 2017.

[20] J. Fu, Y. Liu, H. Chao, B. K. Bhargava, and Z. Zhang, "Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4519–4528, Oct. 2018.

[21] F. Paganelli, S. Turchi, and D. Giuli, "A web of things framework for RESTful applications and its experimentation in a smart city," *IEEE Syst. J.*, vol. 10, no. 4, pp. 1412–1423, Dec. 2016.

[22] S. K. Sharma and X. Wang, "Live data analytics with collaborative edge and cloud processing in wireless IoT networks," *IEEE Access*, vol. 5, pp. 4621–4635, 2017. [Online]. Available: https://doi.org/10.1109/ACCESS.2017.2682640

[23] T. Yu, X. Wang, and A. Shami, "Recursive principal component analysis-based data outlier detection and sensor data aggregation in IoT systems," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2207–2216, Dec. 2017.

[24] F. Bellotti *et al.*, "TEAM applications for collaborative road mobility," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 1105–1119, Feb. 2019.

[25] S. S. Solapure and H. Kenchannavar, "Internet of things: A survey related to various recent architectures and platforms available," in *Proc. Int. Conf. Adv. Comput., Commun. Informat.*, 2016, pp. 2296–2301.

[26] J. Guo, L. Xu, G. Xiao, and Z. Gong, "Improving multilingual semantic interoperation in cross-organizational enterprise systems through concept disambiguation," *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 647–658, Aug. 2012.

[27] "The JavaScript Object Notation (JSON) Data Interchange Format," Internet Eng. Task Force, Fremont, CA, USA, Dec. 2017.

[28] A. German, S. Salmeron, W. Ha, and B. Henderson, "MEAN web development: A tutorial for educators," in *Proc. 17th Annu. Conf. Inf. Technol. Educ.*, New York, NY, USA, 2016, 128–129.

[29] A. Nandaa, "*Beginning API Development With Node.js: Build Highly Scalable, Developer-Friendly APIs for the Modern Web With JavaScript and Node.js*. Birmingham, U.K.: Packt Publ., 2018.

[30] J. Lewis, M. Fowler, "Microservices," 2014. [Online]. Available: https://www.martinfowler.com/articles/microservices.html

[31] T. D. V. Swinscow, *Statistics at Square One*, 9th ed. London. U.K.:BMJ Publ. Group, 1996.

[32] A. Monteriù *et al.*, "A smart sensing architecture for domestic moniotring: methodological approach and experimental validation," *Sensors*, vol. 18, no. 7, Jul. 2018. https://doi.org/10.3390/s18072310.

[33] L. Pescosolido, R. Berta, L. Scalise, G. M. Revel, A. De Gloria, and G. Orlandi, "An IoT-inspired cloud-based web service architecture for e-health applications," in *Proc. 2nd IEEE Int. Smart Cities Conf.*, Sep. 2016, doi:10.1109/ISC2.2016.07580759.

[34] V. Cirimele *et al.*, "The Fabric ICT platform for managing wireless dynamic charging road lanes," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 2501–2512, Mar. 2020.

[35] A. Kobeissi, F. Bellotti, R. Berta, and A. De Gloria, "Towards an IoT-enabled dynamic wireless charging metering service for electrical vehicles," in *Proc. 4th AEIT Int. Conf. Elect. Electron. Technol. Automot.*, Turin, Italy, 2019, pp. 1–6.

[36] A. H. Kobeissi, F. Bellotti, R. Berta, and A. De Gloria, "Raspberry Pi 3 performance characterization in an artificial vision automotive application," in *Proc. Appl. Electron. Pervading Ind., Environ. Soc.*, Pisa, Italy, Sep. 2018, pp. 1–8.

[37] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2016. [Online]. Availble: https://www.sae.org/standards/content/j3016_201806/

[38] Y. Bernard, S. Innamaa, S. Koskinen, H. Gellerman, E. Svanberg, and H. Chen, "Methodology for field operational tests of automated vehicles," *Transp. Res. Procedia*, vol. 14, pp. 2188–2196, 2016.

[39] J. Hiller, E. Svanberg, S. Koskinen, F. Bellotti, F. Osman, and N, "The L3Pilot common data format – enabling efficient autonomous driving data analysis," in *Proc. 26th Int. Techn. Con. Enhanced Saf. Veh.*, 2019.

[40] J. Hiller *et al.*, "The L3Pilot data management toolchain for a level 3 vehicle automation pilot," *Electronics*, vol. 9, 2020.

[41] F. Bellotti *et al.*, "Designing an IoT framework for automated driving impact analysis," in *Proc. 30th IEEE Intell. Veh. Symp.*, Jun. 2019, pp. 1111–1117.

[42] Postman, San Francisco, CA, USA. [Online]. Availble: https://www.postman.com/. Accessed on: May 29, 2020.

[43] I. Portugal, P. Alencar, and D. Cowan, "A preliminary survey on domain-specific languages for machine learning in big data," in *Proc. IEEE Int. Conf. Softw. Sci., Technol. Eng.*, 2016, pp. 108–110.

Riccardo Berta (Member, IEEE) received the M.Sc. degree in electronic engineering and the Ph.D. degree in electrical engineering from University of Genoa, Italy, in 1999 and 2003, respectively.

He is an Associate Professor with ELIOS Laboratory, Department of Electrical, Electronics, and Telecommunication Engineering and Naval Architecture, University of Genoa, Genoa, Italy. He is Founding Member of the Serious Games Society. He authored about 100 papers in international journals and conferences. He has been involved in the Editorial Team as a Section Editor of the *International Journal of Serious Games* and in the Program Committee of the GALA Confernce. His current main research interest include applications of electronic systems, in particular in the fields of serious gaming, technology enhanced learning, and Internet of Things.

Ahmad Kobeissi received the B.Sc. and M.Sc. degrees in computer engineering from Beirut Arab University, Beirut, Lebanon, in 2012 and 2015, respectively. He is working toward the Ph.D. degree in big data architectures for IoT and cloud computing at the University of Genova, Genova, Italy, in Joint Cotutering with the Lebanese University, Beirut, Lebanon.

His current research interests include the Internet of Things, education technology, cloud computing, and machine learning.

Francesco Bellotti received the M.Sc. degree in electronic engineering (cum laude) and the Ph.D. degree in electronic engineering from University of Genoa, Italy, in 1997 and 2001, respectively.

He is an Associate Professor with the Department of Electrical, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa, Genoa, Italy, where he teaches cyber–physical systems and edge computing with the M.Sc. program in electronic engineering. He has been the WP Leader of several European and Italian research projects. His current main research interests include edge computing, machine learning, and cyber–physical systems.

Alessandro De Gloria received the M.Sc. degree in electronic engineering from University of Genoa, Italy, in 1980, (cum laude), and the specialization degree in computer science at the University of Genoa, in 1982.

He is a Full Professor of Electronic Engineering with the University of Genoa, Genoa, Italy. He is the Leader of the ELIOS Laboratory with, Department of Electrical, Electronics, and Telecommunication Engineering and Naval Architecture, University of Genoa. He is the founding and Emeritus President of the Serious Games Society. He sits in the Directive Board of University of Genoa SIMAV center for advanced simulation. He has led and participated in more than 20 research projects in the last 10 years, in the fields of serious games, technology enhanced learning and automotive. His current main research interests include IoT, serious games, virtual reality, computer graphics and HCI.