# Exposing IoT Objects in the Internet Using the Resource Management Architecture

Carlos Eduardo Pantoja, Heder Dorneles Soares, Jose Viterbo, Tielle Alexandre, Arthur Casals, Amal El Fallah-Seghrouchni

# A Resource Management Architecture For Exposing Devices as a Service in the Internet of Things

Carlos Eduardo Pantoja
CEFET/RJ
Universidade Federal Fluminense
pantoja@cefet-rj.br

Heder Dorneles Soares and
José Viterbo and Tielle Alexandre
Universidade Federal Fluminense
hdorneles,viterbo@ic.uff.br, tiellesa@id.uff.br

Arthur Casals and
Amal El-Fallah Seghrouchni
Sorbonne Universités UPMC, LIP6
Universidade de São Paulo (USP)
amal.elfallah@lip6.fr, arthur.casals@usp.br

***Abstract*—This work proposes an architecture for sharing devices' resources in the Internet of Things providing real sensor data for its users. The main idea is based on the fact that users such as developers and researchers do not always have access to the necessary hardware and resource sharing should impact these persons activities. Taking advantage of the Sensors as a Service model, we propose an architecture where several sensors and actuators can be coupled to environments and they also are represented virtually in a web system becoming available to be consumed by users and platforms. The architecture is composed of three layers and a model representing devices, the cloud, and clients, and how they interact with each other. A study case for testing the whole approach is also presented.**

***Index Terms*—Internet of Things, Embedded Systems, Ubiquitous Computing**

## I. Introduction

The number of computing devices used in daily tasks, the internet coverage as well as how to handle data, are increasingly being used to improve people's lives and together with the Internet of Things (IoT), they are the key to the development of a wide range of applications [1]. These large number of components open up relevant issues, such as sharing data in heterogeneous environments once sensors deployment often employs high costs. Then, mechanisms for resource sharing should play a major role in this scenario.

These applications gather data from several sensors and the number of devices can make the project unfeasible. There is an emerging cloud computing model named Sensors-as-a-Service, where users can make public their own sensors' data to be consumed by other clients [2]. However, it is difficult to integrate and deploy such systems due to the heterogeneity of devices and communication technologies [3], [4]. Considering this, middleware for IoT play an important role in these systems since they can deal with the heterogeneity of hardware, data distribution, and communication protocols.

So, the objective of this paper is to propose an architecture for the management of resources and to expose them as a service in an IoT network. This Resource Management Architecture (RMA) is divided into three layers of abstractions where devices are responsible for keeping the real resources independently from the core of RMA, which deals with the virtualization, registering and updating of data from devices. Then, clients can consume this data as a service using exposed web services that access the virtualization layer. The aim of RMA is to provide an integrated solution to be used in any domain since its general architecture is not bounded to any application provided and to work as a repository of devices where researchers can make their data consumable to other persons who might be interested in them.

The RMA is built over instances of a robust middleware for IoT in its layers, which treats the connectivity and scalability of devices [5]. Besides, we also propose a structure for the development of devices able to self-configuration in the RMA. Finally, the RMA can be accessed by web services or other web solutions. The rest of this paper is structured as follows: in section II, we present the related works; section III presents our proposed architecture and their components; in section IV, it is described the implementation of RMA; section V the RMA is evaluated based on software engineering methods. Finally, conclusions are discussed in section VI.

## II. Related Works

During the last years, several approaches try to deal with shareable resources in different domains. For example, the research in Ambient Intelligent has been searching for technological strategies to improve the people life quality and solve problems caused by population growth in urban areas. A lot of papers explore technologies for improving citizen services and how to add value to public administration in Smart Cities [6]. In these cases, there are data coming from people, social networks, household, organizations (public and privates), and so on. Devices are often used in such approaches however, issues as interoperability, communicability, and how to turn data public and consumable are not in their scope. In this paper, we present an architecture that can be used to provide a layer of consumable devices that can be applied in any domain and deal with those issues.

When considering systems that use sensors as a service, there is an interoperable system [7] that offers a web service description language interface designed for users to access sensors. However, its main focus is on low-power wireless sensor nodes and energy profile. The SOCRADES middleware [8] is a solution for business integration that offers a web service-oriented architecture that integrates different types of intelligent objects. It uses an enterprise business solution

DOI reference number: 10.18293/SEKE2019-187

system to provide the ability of users to construct services based on physical objects using the Web of Things [9] and using REST applications [10]. These approaches tie the design of the device to the web system restraining new devices to be added at runtime. In our proposed approach, devices have their own mechanism for dealing with the core system at runtime.

## III. The Resource Management Architecture

In this section, we present the Research Management Architecture (RMA), an architecture responsible for the virtualization of devices that act as IoT objects, exposing their sensors and actuators to be accessed and consumed by who might be interested in their functionalities. These resources' data are maintained at runtime in a model that can be publicly visualized and subscribed for several purposes. Exposing devices as a service that can be virtually consumed brings some advantages since users do not need to have their own devices, reducing eventual costs in creating new ones. In this way, users can provide a shareable resource component that can be exploited for several purposes. The proposed architecture (Figure 1) is composed of three different layers:

1) **Device**: in this layer, the devices have an embedded system enhanced with a processing cycle where (i) it connects and registers in the Cloud layer at the very first time; (ii) it gathers data from all its sensors to be sent to the cloud layer and; (iii) it receives from the Cloud layer actions that must be executed by the device's actuators.
2) **Cloud**: it is capable of maintaining updated information from devices hosted in environments and running over an IoT middleware. The Resource Management Component (RMC) manages (i) the registering process of devices; (ii) the sensors' data updating process, and; (iii) the actions that must be executed by devices. Besides, it also capable of exposing RESTful web services for several purposes.
3) **Client**: it is responsible for the visualization of devices and environments. In this paper, it is represented as a web solution. It implements a publish and subscribe mechanism for allowing developers and researchers in obtaining data from real sensors without possessing them and it offers an infrastructure for those who want to provide sensors as a service.

In the RMA, a device is an IoT object equipped with a micro-controller (hardware) where sensors and actuators are plugged in. The devices host an embedded system capable of connecting and registering to the Cloud layer for providing current data to be consumed by users. The embedded system dynamically registers itself in the RMC by sending all its functionalities (available resources, data and action commands) and the environment it is situated. Once connected, the device receives a confirmation and it starts to send data gathered from its sensors directly to the RMC, which keeps the most recent value available to be consumed or visualized in a web panel in the Visualization layer. Besides, the embedded system deals with the action commands coming from the RMC that need to be executed by the device's actuators.

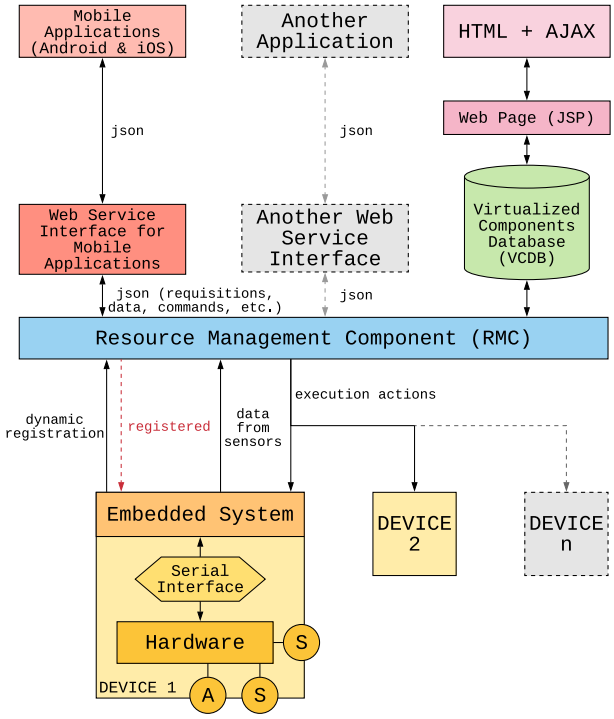


Fig. 1. The Resource Management Architecture's (RMA) components.

The RMC is the main component in the RMA and it maintains information about devices and environments to be consumed as a service by other layers. It has a mechanism for registering devices in given environments, storing their information in the Virtualized Components Database (VCDB). The VCDB keeps updated the data coming from devices by considering only the new values that have changed since the last data message was received. In addition, this layer exposes web services for providing different services such as visualization layers for mobile applications or allowing new technologies to interact and access the VCDB. Web services use a RESTful architecture for system interoperability and it uses Json files for exchanging information (data, requisitions, and execution commands) between the Cloud and Client layers.

The actions that need to be executed by devices are managed by the RMC. Any user that desire to perform an action in any available actuator must send an action command by accessing one of the provided services. The RMC redirects the action directly to the specific device. The technical information about hardware and actuators is transparent to the end user, that just need to know the available commands of the device he or she wants to interact. While one device is being used, the RMC blocks the specific actuator or the entire device for avoiding conflicts in the lower layer. The respective resource is unlocked after the execution or after a timeout boundary.

The RMC is a server-side solution running an IoT middleware instance where the devices must connect to. The middleware provides the connectivity and communicability necessary for devices to interact with the RMC layer and it

should guarantee the scalability of the system. These issues are not the focus of this work, however more technical details about the chosen middleware are provided later in Section IV.

For facilitating the environments managing, this layer provides a web page where all the environments can be accessed at real-time and users can create their own public or private environments. This web system shows all the public environments allowing users to select and see their available information and subscribe to have access to a specific and personalized group of information. In this work, an environment is defined by a logical representation of a physical space where several devices can co-exist sharing information in an IoT network.

## IV. Implementing the RMA's Layers

In this section, it is presented the implementation of the Device, Cloud and Client layers considering technological components and how these layers communicate with each other. For both Device and Cloud layers, we employ the ContextNet middleware [5], which is an IoT middleware for context reasoning and data sharing in a large scale environment. It is a context-providing service for stationary and mobile networks, which already addresses data communication issues such as fault tolerance, load balancing, node disconnect support (handover), and security. It uses the OMG DDS [11] protocol for handling messages between clients.

The Device layer uses a low coupled serial interface [12] for communicating with the micro-controllers that hosts the sensors and actuators. This interface isolates the high-level programming from the low-level using serial commands for activating users' pre-defined functions, which controls actuators or gathers data from sensors. The Client layer employs a web system capable of managing environments, showing their available resources, and it has a publish and subscribe mechanism for users interested in specific resources. All layers consider a model where these resources are part of devices situated in pre-existing environments in the architecture. For instance, the environments have to be manually registered by the user to facilitate their management and for security issues. This model is represented as a class diagram (Figure 2) shared between the three layers. The classes that can be found in our solution are described as follows:

- **Action**: the action that is executed at the low-level hardware in a device. Every command sent by the the Client and Cloud layers becomes an action in the Device layer.
- **Command**: it is the representation of commands available to be executed if the resource is an actuator. Sensors do not have commands because they are data providers.
- **Cycle**: it is the functioning cycle of the embedded system hosted in devices. It is responsible for synchronizing the device activities of sending data and executing actions.
- **Device**: it is the device representation used in the Client, Cloud and Device layers. It keeps the identification, name and description of a device and it is composed resources.
- **Embedded Client**: it is the ContextNet client instance responsible for receiving messages from the Cloud layer and other clients, and sending the data from sensors to the Cloud layer. It also maps the components of the configuration file into its respective components.
- **Environment**: the virtual representation of environments in all layers. Each environment is composed of devices.
- **Main**: the main class that starts a device.
- **Resource**: it represents both sensors and actuators in all layers. The resources keep information about the serial port where the resource is connected, the available commands to be executed, and the availability of the resource.
- **Resource Management**: it is the main class of the cloud layer and it is a server instance of the ContextNet. It keeps a list of environments mapped and the devices registered for each one. It is responsible for the process of registering and updating devices and resources. It also exposes the web servers and the interfaces to the database.
- **Serial Communication**: the serial interface between the micro-controller and the system hosted at devices.

### A. The Device Layer

The Device layer comprises devices running an embedded system with enough processing power interfacing sensors and actuators. There is a physical and logical architecture for clients where the first one uses hardware technologies and it is composed of a tiny mobile board with Bluetooth and WiFi connections (e.g. Raspberry Pi) connected to one or more micro-controllers using serial communication for accessing sensors and actuators. The logical architecture of the Device layer comprises both micro-controllers' programming and the embedded system. They are able of gathering the raw data from the sensors and then send it to the embedded system. After that, the embedded system sends the data to the Cloud layer or receive commands to be executed by actuators.

The micro-controller is programmed in a loop for verifying if there are messages coming from other layers. So, it accesses all the sensors and mounts a string to be sent by serial communication to the embedded system. Otherwise, the execution action is verified and if exists an equivalent programmed, an action is executed in the respective actuator.

**Algorithm 1** Device's Processing Cycle

```
1:  procedure CYCLE(configurationFile)
2:      mountDevice(configurationFile)
3:      intervalTime ← getIntervalTime()
4:      loop
5:          if isRegistered() then
6:              getheredData ← dataFromSensors()
7:              sendToCloud(gatheredData)
8:              wait(intervalTime)
9:              actions[] ← getReceivedActions()
10:             executeNextActions(actions[])
11:         else
12:             registerDevice()
```

The embedded system controls the microcontroller and it uses a ContextNet client able to communicate with the Cloud layer. For this, there is a cycle (Algorithm 1) for synchronizing

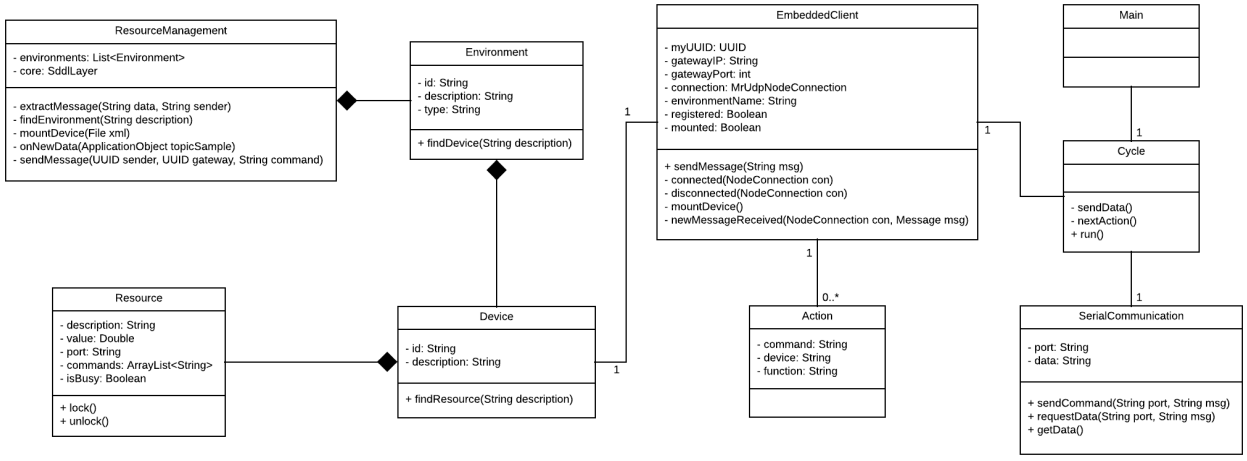


Fig. 2. The class diagram of the overall architecture comprising the Client, Cloud and Device layers.

the reception of data coming from sensors to be sent to the Cloud layer and the actions that need to be sent and executed in the micro-controller because both cannot execute at the same time for avoiding undesired conflicts. So, the actions received from the Cloud layer are put in a queue of actions to be executed one step after the data from sensors are collected through serial ports. It is also capable of performing a self-configuration and registration at the Cloud layer when it starts running and there is a cloud server available. For this, the device keeps an XML file describing all the available resources, commands, and the server configuration information. Once the file is correctly filled, the system performs everything automatically. The following information must be provided:

(i) **Server**: the gateway of the server and the port must be provided in order to connect to the Cloud layer where the server instance of the ContextNet is installed. Besides, the time interval of sending data to the server must be set.
(ii) **Environment**: for instance, it is necessary to inform in which environment the device is inserted into. For this, the identification number of the environment must be declared in the configuration file. The environment's identification is generated when users register an environment in the web system.
(iii) **Resources**: all the resources available must be mapped in the configuration file. All the resources have the serial port where it is connected, its name, and a non-mandatory description. If the resource is an actuator, it also has the available execution commands.

The embedded system architecture is composed by the *Cycle* class, which instantiates an instance of ContextNet client (represented by the *EmbeddedClient* class) capable of exchanging message with other clients and the cloud server. Besides, the *EmbeddedClient* is responsible for the serial interface and it deals with a queue of actions received from the Cloud layer that has to be executed. The *SerialCommunication* and *Action* are the classes responsible for these behaviors.

### *B. The Cloud Layer*

The cloud layer is a server instance of the ContextNet running a core system responsible for the virtualization of environments, devices and available resources. The devices register at the cloud informing their resources and environment where they are situated. This process starts when the core receives the file containing all the information of the device. Then, the system extracts the necessary information from the file and it registers the device at the system in the informed environment. Afterward, it sends back a message to the device authorizing the beginning of sending data from sensors.

Once the devices start sending data, the core system registers the new values and updates the old ones in VCDB (Figure 1). For every device, there are resources that can be sensors or actuators. In the case of sensors, if it is the first time that a new value of a resource is received, the system inserts this new value at the VCDB. Otherwise, if the value has changed since the last data reception, this new value is updated in VCBD. In the case of actuators, there is no data to be stored but it is kept the information about the availability of the resource. For example, the system informs if a certain actuator is being used or it is free for executing commands.

The core system deals with commands requisitions coming from the Client layer (a web system or by exposed web services) to be sent and executed at devices. The Client layer does not need to know technical details of hardware or even in which device the command will be executed. The commands are specific for a resource and the core system avoids duplicated resources and commands. Besides, it also redirects a received command to the respective device registered at the system by sending a message using ContextNet.

In the Cloud layer, there is also a locking process for avoiding conflicts in executing actions when two or more clients try to execute commands in the same actuator. For this, every time that a client needs to use a specific actuator, the core system locks this resource until the client informs that

is no longer using it, the action was performed, or a timeout boundary is reached. During this process, the locked resource is unavailable for all clients. It is important to remark that sensors are not part of the locking process because the nature of sensors is to provide data to be consumed. If the data is considered confidential, the environment can be set as private.

As stated before, web services can be exposed for extending the functionalities of the RMA in the Cloud layer. For creating RESTful web services it is used as an embedded servlet container and web server named Jetty. The available web services are a requisition service for agents applications for using resources with contextual planning; an execution service for activating or deactivating the actuators based on available commands; and a web service for providing data access to mobile applications.

The Cloud layer's model is composed of the *ResourceManagement* class where the ContextNet server is instantiated. Besides, it hosts a list of *Environments* with their *Devices*, *Resources*, and *Commands* classes.

### C. The Client Layer

The idea behind the Client layer is to provide a layer capable of showing environments' resources in any kind of platform such as web pages, web services or mobile applications. Besides, it is responsible for providing some basic mechanisms that are not available in previous layers such as environments creation and, publish and subscriber mechanisms for example. The Client layer is represented as a web page for showing all the resources of environments and some basic functions for interacting with the resources. The user is able of creating environments to virtually host his devices and to expose them to be consumed by other users. Besides, the actuators have commands that users can activate by interacting with the Client layer's web system. The technologies employed are relational databases, Java web pages and Ajax.

Besides that, users can choose to follow some of the resources without the need to access the environments every time that he needs to get those values. The publish and subscriber mechanism allows users to access values always that a change is perceived by the core system in the Cloud layer. It allows the user to set basic rules such as defining a desired value to be announced when reached. All existing modules can co-exist without interfering in each other since all functions are managed by the Cloud layer. They all have to connect to the Cloud layer's database or use an exposed web service to access information.

## V. Evaluation

In this section, it is presented an initial case study evaluation in the assisted environment domain using the proposed RMA using a software engineering based approach. Case studies are employed because they are suitable for evaluation of software engineering methods involving development, operation, and its maintenance and artifacts [13].

The scenario will be held in a hypothetical Smart City where the government has access to a hospital where the RMA is implanted. Some rooms in the hospital building have devices for controlling the temperature and luminosity (as sensors), and light lamps of the room (as actuators), and other devices for measuring some of the patients' information such as heartbeat frequency. Therefore, every room in the hospital endowed with devices is considered an environment in the RMA and its devices' resources are exposed as a service for the board of directors, government and everyone interested in them. It is important to remark that, even in this hypothetical scenario, there is no real personal contact of patients available.

Based on this, two devices were prepared for a room, named `Room 403`. Both devices use a Raspberry Pi Zero connected to an Arduino board. The first device is connected to a temperature and a luminosity sensor for the basic sensing of the room. The second one is a device with a light lamp connected to the Arduino working as an actuator and informing if the lamps are on or off. Besides, virtual devices were simulated in order to stress the RMC functioning. For this, the serial interface between the embedded system and the hardware were disabled, and several resources were simulated for each virtual device, which sends random data to the Cloud layer. So, one device for monitoring the heartbeat frequency of a patient and devices identical to the real ones above were simulated in each room. In general, 20 environments were prepared where the environment `Room 403` has two real devices and one simulated, and the other 19 have three simulated resources.

The case study approach is divided into four steps: case study design, preparation for data collecting, the data collecting and data analysis. The following Table I shows the details and aim of the descriptive case study.

TABLE I
The Case Study Design

| Design | Description |
|---|---|
| **Objective** | A descriptive analysis of the behavior of the RMA functioning. |
| **Case** | The asynchronous process of transferring information from devices to the Cloud layer to be consumed by clients using web solutions. |
| **Questions** | Is the device connects correctly to the Cloud layer? Is the communication process between all layers works? Is the Client layer showing the correct data? |
| **Method** | Qualitative data analysis using negative case analysis and observation method. |

Some tests were conducted trying to deny the research questions above. The preparation for the data collecting consisted in store both dynamic registration at the RMC and the answer that devices receive before starting sending data from sensors. Afterward, it is verified if these data arrives at all layers properly by analyzing the transferring process between hardware and device, device and cloud, and cloud and client. A string of data that comes out from the hardware is collected and compared if the data read arrived correctly at the embedded system. Between Devices and the Cloud layer, all devices should keep sending data to be stored at the VCDB in the RMC. Finally, between Cloud and Clients, these same

data should be read by clients when data update occurs.

The data collecting and analysis were performed in an arithmetic progression from 1 to 20 devices. Firstly, the server instance was running properly to verify the effectiveness of RMA and then it was disabled. Once there is no server instance available, they should not send data. Then, the data should be properly stored and read by Cloud and Client layers. Table II shows the resumed results from tests.

TABLE II
DATA COLLECTION AND ANALYSIS

| Test | Description | Hit (%) |
|---|---|---|
| **Hardware** | Data is correctly transferred to the embedded system. | 100 |
| **Connection with Server** | Device registered at RMC and registered message received. | 100 |
| **Connection without Server** | Device registered at RMC and registered message received. | 0 |
| **Data Updated and Stored** | Data correctly stored at VCDB. | 100 |
| **Data Read** | Data correctly read by clients. | 100 |

The communication between hardware and the embedded system is done using a serial interface, which guarantees no losses in the data transferring. None errors were observed in this process. As expected, when the server instance is disabled, it is observed that devices try to connect to the Cloud layer but there is no response from the server and no data is sent from any device. Otherwise, the device is registered and receives a confirmation to start sending data to the Cloud. All devices work properly considering an available server instance.

The most recent information available coming from devices is updated in the VCDB. Considering that the VCDB is implemented as a relational database, there are no big deals in this process not even in the visualization of the devices by the Client layer. This case study focused on observing the communication and the correctness of data flowing through the architecture. More experiments focusing on performance and a proper formalization were left for future efforts.

## VI. CONCLUSION

In this work, it was presented a low-coupled three-layer architecture for exposing devices as a service to be consumed by clients. The devices are able of connecting to an IoT server, registering their resources (sensors and actuators) in a core system, which turns all the public data available that can be accessed by clients using web services and a web platform. A case study was proposed and evaluated integrating the Device, Cloud and Client layers for monitoring an environment.

RMA architecture employs different technologies in its layers. Devices are autonomous and uncoupled from the Cloud layer because they are built using hardware platforms enhanced with wi-fi connections and it uses a serial interface for communicating with micro-controllers. These technologies provide the necessary autonomy, heterogeneity of hardware employed, and communicability to the Cloud layer. The ContextNet middleware provides client and server instances and it is used in the architecture because of the middleware guarantees connectivity, communicability, reliability, and scalability, in addition to using an industrial market standard protocol.

The Client layer offers web solutions for managing environments and for the visualization of their respective resources. Besides, it is possible to subscribe specifically to resources that one might be interested in. Moreover, the RMA aims to provide an architecture for exposing devices as a service to be consumed by persons that do not have access to these kinds of resources either for the cost or complexity of creating from scratch an architecture for that purpose.

Nowadays, the designer of the device should program how data is mounted and captured by the micro-controller and sent to the embedded system. As future works, it is important to create an automatized plug-and-play way of configuring the device in low-level. Besides, as micro-controllers are connected to the serial port of the tiny computer of the device, a similar process for the identification of serial ports by the embedded system is also interesting. The environment has to be set manually at the device's configuration file for security and control reasons, nevertheless, it is possible to identify and register autonomously the environment based on access points.

## REFERENCES


[1] E. Santos, P. H. V. Penna, I. M. Coelho, H. D. Soares, L. S. Ochi, and L. Simonetti, "Logistics sla optimization service for transportation in smart cities," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2018.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a Service Model for Smart Cities Supported by Internet of Things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 3, pp. 294–307, 2014.

[3] M. Weiser, "Some computer science issues in ubiquitous computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, pp. 12–, July 1999.

[4] C. Pantoja, H. Soares, J. Viterbo, and A. Seghrouchni, "An architecture for the development of ambient intelligence systems managed by embedded agents," in *The 30th International Conference on Software Engineering & Knowledge Engineering*, (San Franscisco), 2018.

[5] M. Endler and F. S. e Silva, "Past, present and future of the contextnet iomt middleware," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 7–23, 2018.

[6] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano, "Current trends in smart city initiatives: Some stylised facts," *Cities*, vol. 38, pp. 25–36, 2014.

[7] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pp. 253–266, ACM, 2008.

[8] L. M. S. De Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, "Socrades: A web service based shop floor integration infrastructure," in *The internet of things*, pp. 50–67, Springer, 2008.

[9] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web, in proceedings of International World Wide Web Conferences, Madrid, Spain*, vol. 15, 2009.

[10] B. Ostermaier, F. Schlup, and K. Römer, "Webplug: A framework for the web of things," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 690–695, March 2010.

[11] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pp. 200–206, IEEE, 2003.

[12] N. M. Lazarin and C. E. Pantoja, "A robotic-agent platform for embedding software agents using raspberry pi and arduino boards," *9th Software Agents, Environments and Applications School*, 2015.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.