IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 September 2022, accepted 9 October 2022, date of publication 13 October 2022, date of current version 20 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3214229

RESEARCH ARTICLE

Edge Computing and Microservices Middleware for Home Energy Management Systems

LUIZ C. B. C. FERREIRA[1], ANDREZA DA ROSA BORCHARDT[1], GUSTAVO DOS SANTOS CARDOSO[1], DIMAS AUGUSTO MENDES LEMES[1], GABRIEL RODRIGUES DOS REIS DE SOUSA[1], FERNANDO BAUER NETO[2], EDUARDO RODRIGUES DE LIMA[3], GUSTAVO FRAIDENRAICH[1], PAULO CARDIERI[1], AND LUÍS GERALDO P. MELONI[1]

[1]Department of Communications, School of Electrical and Computer Engineering, University of Campinas (FEEC), Campinas, São Paulo 13083-852, Brazil
[2]Companhia Paranaense de Energia, Curitiba, Parana 81200-240, Brazil
[3]Department of Hardware Design, Instituto de Pesquisa Eldorado, Campinas, São Paulo 13083-898, Brazil

Corresponding author: Luís Geraldo P. Meloni (meloni@unicamp.br)

This work was supported by the Companhia Paranaense de Energia (COPEL) through the Research and Development Project ANEEL PD-02866-0508/2019.

ABSTRACT A middleware software can be seem as an abstraction layer between hardware and user applications, that facilitates the development and deployment of services in various scenarios, such as those found in Home Energy Management Systems (HEMS). There are several middleware proposals for HEMS, with most of them taking the cloud computing approach. This approach is unconcerned about computing resources but raises a dependency on external connections. This paper presents a middleware for energy management systems, based on the concept of edge computing for smart homes. The paper presents a reference model for the proposed architecture, considering specific requirements for this type of application. The proposed architecture employs the concept of microservices for data access and system configuration. The proposed middleware is designed to work with embedded systems under computational constraints, such as processing capability and storage, to reduce costs and allow its application closer to the user. The middleware is open and customizable to meet the developer's needs. The proposed solution was implemented and tested in a university laboratory, as well as at the Eldorado Research Institute to confirm the effectiveness of the middleware. The proposal stands out from others found in the literature as it can be implemented using low cost hardware. In addition to using microservices concepts, the proposed middleware is a valuable option for applications that need an edge computing approach. A performance analysis was carried out, using low cost hardware with limited resources. The results show that the proposal can handle a significant number of devices, offering low latency and low error rate, and consuming few processing resources and memory.

INDEX TERMS Home energy management systems, middleware, Internet of Things.

I. INTRODUCTION

One of the challenges that the society faces nowadays is to meet the growing demand for electrical energy using sustainable and environment-friendly solutions. One way to address this issue is based on using energy management systems. Home Energy Management Systems (HEMS) are technological solutions designed to manage the use of electrical energy in homes or commercial buildings, by measuring and analyzing the data consumption and/or controlling energy production.

Typical requirements of the HEMS solutions include implementation in low cost hardware platforms with low computational capacity, edge processing capability, low

The associate editor coordinating the review of this manuscript and approving it for publication was Vyasa Sai.

latency, continuity of operation in case of Internet connection interruption, and reliable security requirements. There are several IoT middleware proposed in the literature that, in principle, could be employed in HEMS solutions. However, the requirements of HEMS solutions are more stringent. This situation has motivated the proposition of a novel middleware architecture focused on HEMS applications.

HEMS applications can be considered within the IoT context, which involve several areas of knowledge, such as computing, communications, energy, data analysis, and microelectronics [1]. The variety of products and manufacturers of IoT devices, the lack of standardization, and the multidisciplinary nature are key characteristics of any IoT deployment. This heterogeneous and dynamic environment complicates the development of solutions (such as HEMS), leading to a dependency on platforms that provide interoperability of systems, devices, people, and data [1].

In this context, middleware platforms emerge, which operate as a link connecting distinct layers. The middleware works as an intermediary abstraction layer between the devices (such as smart outlets, sensors and controllers, in the case of HEMS) and the user applications. Thus, the middleware facilitates data acquisition and processing by providing high-level APIs that abstract the complexity of the implementation.

A scenario also related to the IoT and HEMS contexts is the emergence of low cost hardware platforms [2], opening up new possibilities for application. These low cost hardware platforms make it possible, for instance, to deploy a large-scale HEMS system in a city, serving thousands of customers with affordable hardware and acceptable performance. However, this lower cost is typically associated to low computational power (i.e., processing, storage and communication), requiring special attention when designing reliable communication, storage and data processing.

A trend observed in the literature and in the market of IoT is to move some of the services offered by IoT solutions closer to the end-user, using an edge computing approach [3], [4] [5]. In the context of HEMS, this trend means a middleware platform operating closer to the user and addressing the requirements of a HEMS application. Figure 1 depicts this scenario, with the middleware implemented in the edge.

There are several middleware platforms for IoT available nowadays. In general, these platforms were not developed for low computational power hardware, preventing them to be used in an edge computing approach.

Typically, middleware platforms work under the concept of cloud computing, providing scalability, high processing capability, and nearly unlimited data storage capability [3]. However, the middleware can also work under the fog computing paradigm, acting as a gateway, gathering data from multiple users and pre-processing them before sending them to the cloud [6]. The cloud and fog computing approaches are characterized by the availability of large computing resources, including processing and storage. However, these approaches

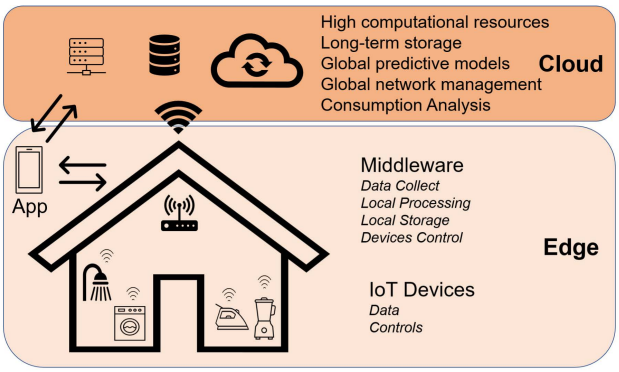


FIGURE 1. The scenario adopted for HEMS in the middleware proposal.

add delays to the communication with end-devices [3], potentially affecting the end-user experience and increases the cost.

A software development approach that has received a great deal attention in the last few year is the microservices approach, which consists of small, independent services communicating via well-defined APIs [7]. The use of microservices to expose the middleware functionalities provides scalability and accelerates the development of the middleware, enabling innovation and reducing the time to introduce innovative solutions.

This work presents a middleware platform for a home energy management system, based on microservices and designed to run on low-cost, low computational power hardware, following the edge computing approach. These characteristics distinguish this work from others found in the literature, where middleware platforms work predominantly under the cloud computing approach and use development architectures based on conventional Web services. The proposal presented here was developed for limited capacity, low-cost hardware, and, for that reason, considers the consumption of computational resources.

The main contributions of this work are:

- We present a literature survey of middleware for IoT, focusing on the hardware requirements and architecture.
- We propose a middleware architecture for HEMS following the edge computing approach, providing the main HEMS functions closer to the user. This approach removes the requirement for external connections (from internet or network operators) and decreases the response time for the end-user. The proposed middleware was designed to run on low-cost hardware, and was implemented using the Microservices architecture as a method of access to data and middleware configuration.
- We present implementation details and performance analysis results of the proposed middleware.
- It is an open middleware, which the APIs are available at Github project (RT-DSP/SHArM), alongside its complete documentation.

The proposed middleware is part of an energy management system being developed in the research project "Open Middleware and Energy Management System for the House of the Future." This project is carried out via a partnership involving the University of Campinas, the Instituto de Pesquisas Eldorado, and the Brazilian energy provider Companhia Paranaense de Energia (COPEL).

The remainder of the paper is structured as follows: Section II presents a literature review on middleware architectures for IoT; Section III introduces the proposed middleware architecture; Section IV shows the middleware implementation; Section V presents the results and Section VI presents the conclusions of the work.

II. MIDDLEWARE IoT - REVIEW AND APPROACHES

In the IoT context, the middleware resides between the applications and the subjacent infrastructure (communication, processing, storage, etc.), offering a standardized method of accessing data and services via a high-level interface. The middleware provides an abstraction for developers, concealing complexities of adjacent hardware components, network protocols, characteristics of operating systems, among other implementation details [8].

Several HEMS architectures have been proposed in the literature, such as those presented in [9], [10], [11], [12], [13], and [14]. All these works present architectures and systems for HEMS, but only the work in [11] mentions the use of a middleware.

The work in [15] presents a comparison among 20 open systems for Smart Homes. However, none of these solutions uses a middleware to facilitate the development and implementation of new features, thus increasing flexibility, interoperability and system scalability.

There are few works in the literature about middleware for HEMS and Smart Homes, even though middleware is an important component in these systems, as it can be used to deal with interoperability, flexibility and scalability issues of these systems.

The literature shows that some energy management applications use a middleware platform [16], [17], [18], [19], [20]. These works seek to meet the requirements of HEMS applications. In general, the middleware platform is based on cloud or fog computing, where processing and control functions are performed in the cloud or in an intermediate environment, in-between the cloud and the edge.

As HEMS systems and Smart Homes can be seen as IoT applications, IoT middlewares could be used in these applications. Several middleware platforms are available for IoT in the literature. A survey of these platforms is presented in [21], which shows that most proposals are based on the concept of Platform as a Service (PaaS). This concept relies on cloud development with almost unlimited computational resources, making a middleware platform developed under this concept appropriate to be used in applications with different requirements in terms of computational capabilities [22].

The main requirements of IoT middleware are [21], [27]:

- Scalability: The middleware must be scalable to allow allocation and release of computational resources as needed, keeping the system in operation at a consistent and adequate level of performance;
- Real-time: Due to the nature of HEMS operations, which involve collecting and processing energy consumption information, the delay when sending and processing data must be kept low;
- Interoperability: The interoperability among the various components of an IoT application (devices, applications, services, etc.) is a key requirement for an IoT middleware platform;
- Security and privacy: Any middleware block containing personal or sensitive information must preserve user privacy;
- Data management: In addition to its ability to process large volume of data, the IoT-oriented middleware platform needs to provide data management, including data store, verification, and processing;
- Context Awareness: The middleware must understand, analyze, and react according to context changes, both past and present, in an attempt to infer potential future actions.

An emerging trend in the IoT middleware context is to move data processing and device control closer to the end-user. This approach reduces latency and the dependency on external communications, maintaining the system functional even in the case of interruptions of Internet access. Moreover, this approach allows an additional security layer to be implemented between the edge and the cloud, leading to an overall performance improvement of the services offered [3], [5] [4].

In order to guarantee a massive deployment of energy management systems, the end-user devices must be low cost, which means devices with low computational capability. Therefore, providing services with good performance in energy management systems may be a challenging task. In this case, offering these services through a middleware using the edge computing approach may be a good solution [18].

The edge computing approach requires special attention to hardware requirements. In the works mentioned here, this problem is not addressed, since the processing and control take place in the cloud or fog, in which computational constraints is typically not an issue.

In [23], several IoT middleware platforms were compared, using qualitative and quantitative metrics. All of the analyzed platforms use the PaaS approach, and hardware requirements are not discussed. Furthermore, the tests were carried out using high computational power hardware.

However, hardware requirements of middleware in the IoT context can be a concern [28], [29], [30]. For instance, Perera et al. emphasized the importance of designing middleware solutions for low computing power devices to achieve the IoT vision. This indicates the importance of approaches

TABLE 1. Comparison of the proposed middleware and approaches available in the literature.

Ref.	Hardware	Microservices	HEMS Focused
Proposed middleware	**NXP i.mx6 - 396 MHz ARM Single Core 512 MB RAM**	**Yes**	**Yes**
[3]	Raspberry 3 - ARM Cortex-A53 Quad Core 1.2 GHz CPU 1 GB RAM	Yes	No
[11]	High Performance Servers (Not Specified)	No	Yes
[23]	Intel Xeon E5-1620 v3 Quad Core 3.5GHz 32GB RAM	No	No
[24]	Raspberry Pi 3 - 1.2 GHz ARM Single Core 1 GB RAM	No	No
[25]	AMD Athlon (tm) 64 X2 Dual Core 2.21GHz 1GB RAM	No	No
[26]	Raspberry 3 - ARM Cortex-A53 Quad Core 1.2 GHz CPU 1 GB RAM	No	No

that address hardware requirements, as the one proposed in this work.

As far as the middleware development architecture is concerned, the literature review shows a predominance of the REST architecture. For instance, in [23] several solutions are discussed, and all of them use REST.

In [3] the authors discuss the use of microservices as development architecture. In fact, the use of microservices in the middleware context is a promising approach since it standardizes access to the middleware functionalities and allows any entity operating with the HTTP standard to interact with it, improving the interoperability degree of the middleware platform. The microservices are independent from each other, providing scalability and resilience to the middleware.

Table 1 lists several middleware approaches found in the literature, summarizing their hardware configurations and development architectures. This table also presents the hardware specification of our proposed solution, showing that it requires a simpler hardware. Furthermore, our approach uses microservices as a development architecture, providing more flexibility and ease of implementation of new features and modules.

It is worth mentioning that several other middleware approaches found in the literature are based on cloud and fog computing, in which the computational capability is almost unlimited. Other studies are conceptual, lacking proof of concept or practical testing.

III. MIDDLEWARE ARCHITECTURE

Several middleware solutions are already available in the literature and are widely employed in many designs. Most of these solutions were developed for platforms with almost no restrictions regarding hardware computational capability. The HEMS being developed in the project ''Open Middleware and Energy Management System for the House of the Future'', on the other hand, is expected to be implemented at low computation capability (and, therefore, low cost) hardware. This scenario led us to develop a new middleware architecture suitable for low-cost hardware.

The proposed middleware was designed to operate within the home/commerce of the end-user's premises, running on the edge portion of the HEMS solution. The advantage of the edge approach is to bring processing and control tasks of devices closer to the end-user. Only the most complex processing tasks that require higher computing capability are performed in the cloud.

In the scenario envisioned in the project, smart outlets collect appliances' electrical consumption data and actuators may control the operation of appliances, solar inverter and charger, among other devices, allowing an efficient home energy management. The middleware is responsible for gathering and storing data, processing data, and controlling the attached devices. In addition, the middleware sends data to remote repositories located in the cloud, where the most complex processing tasks involving big data and machine learning techniques are performed. The data generated through these more complex processes are accessible to the end-user through client applications and websites.

Moving some of the middleware operations to the edge reduces the delays observed by the end-user when accessing services for device control, consulting, and information registration. This approach also reduces the need for Internet access to monitor the appliances, since there is a local connection between the client application and the middleware.

The proposed middleware architecture considered the following aspects of the HEMS project:

- Cost: The solution must use open-source tools and open languages and be free from expensive intellectual property controls;
- Low computational available capacity: The solution is expected to operate in simple hardware, aiming to provide low-cost systems running on the edge portion of the system;
- Software requirements: The middleware must address the requirements of the remaining components of the overall system, such as cloud software and applications;
- Standardized access: All services offered by the middleware must be accessible through REST APIs, employing the concept of microservices.

The middleware offers all required services for its configuration, including parameter and interface adjustments and updates. The information is accessible through applications and administration systems designed for the electric energy provider personnel.

The proposed middleware architecture is based on the reference model shown in Figure 2. As depicted in the figure, the middleware operates by connecting two ends: the ''HEMS Devices'', i.e, smart sockets and local access devices, and the ''Applications'', i.e., software programs in the cloud and remote applications.

The middleware contains a core, which consists of the basic modules needed to implement the primary functions

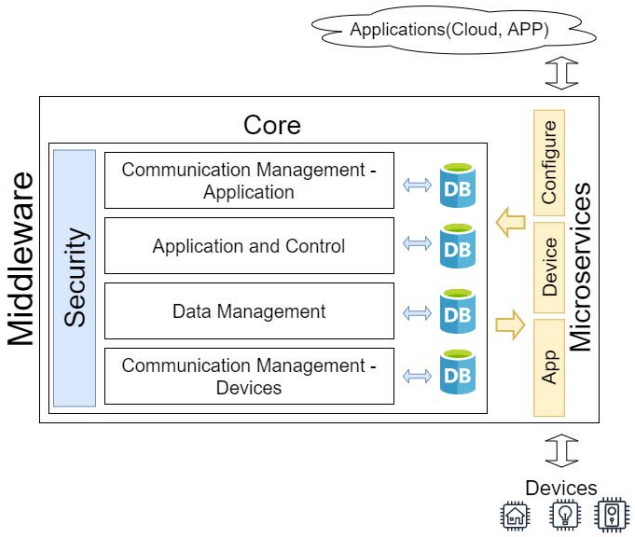


FIGURE 2. Proposed Middleware architecture reference model.

for its operation. The modules are independent processes, performing specific and well-defined tasks.

Every function offered by these basic modules is accessible through microservices using REST APIs. Each microservice has specific and objective functions providing complete control and configuration of the middleware.

In the following, the components of the proposed architecture are described:

- Microservices: A feature of the proposed middleware is the use of microservices via a local server to interact with other components. The use of REST APIs standardizes the usage of the middleware. Any entity using HTTP can communicate with the middleware, either by using data or changing behavior. The microservices paradigm is attractive for the IoT context since it provides resilience, scalability, and deployment agility. Furthermore, the use of microservices architecture simplifies the middleware development due to the independence among the microservices, allowing developers to create new services regardless of the existing ones. The microservices independence improves middleware's resiliency and performance, reducing concurrent and parallel accesses with independent databases. Another factor is the ability for services to continue to operate normally in case a service failure.
- Communication Management - Application: Once the data are collected and handled, it must be sent to a remote repository, processed, and displayed to the end-user. Services are performed in the cloud, such as load disaggregation, discussed in Section IV-B, and detection of abnormal behavior of appliances. Several tools and protocols are available for sending local data to remote repositories. Many of these options are available in this module. It is possible to send local data via Message Queuing Telemetry Transport (MQTT) protocol, which is widely employed in the IoT scenario. Additionally, it is possible to employ HTTP using the Constrained Application Protocol (CoAP). There are also proprietary solutions, such as Microsoft (Azure), AWS, Google, and others. These options are feasible to adopt once the proposed middleware is open and flexible.
- Control and Application Management: The middleware must provide an environment for user interaction with connected devices in the local network, based on simple programming routines, either specified by the user or available from equipment manufacturers. This module includes services for device control, such as ON/OFF control, dimmer, and energy controllers. It is also possible to implement intelligent applications, such as load disaggregation, fault detection, and temperature control. In summary, the middleware allows the development of various applications for HEMS and smart home environment.
- Data Management: This modules processes incoming data from HEMS devices. Functions such as storage and filtering belong to this component. The focus of this module is local data storage. Several appliances can generate data in HEMS scenario, requiring a reliable storage system. The size of the database must be appropriately managed to avoid performance degradation of the middleware. Large databases working on hardware with low performance leads to delays and failures in queries and recordings. To address this issue, the local database was designed as a circular queue, where replacing older data with newer data. The size of the circular queue is configurable. The local database stores the information for a short period of time, as all data are sent to remote repositories to be consolidated and processed using advanced techniques. Several database types are supported, such as SQL, NoSQL, time-series, and others.
- Security: This component addresses the security issues found in all middleware components. However, there is no single security technique that covers all components, and a set of techniques was employed: (i) at the communication layer among HEMS devices, used to send data to the client applications, security is provided by the communication protocols employed, such as Wi-SUN, Wi-Fi, and Zigbee, which already have their own security layers; (ii) at the operating system level, security is based on access control, encryption, and hash algorithms for reliable and confidential data storage; (iii) application protocols used to send local data to remote repositories, such as HTTP, MQTT, and CoAP, also have their own security mechanisms; (iv) finally, the microservices architecture encapsulates the functions, making the source code inaccessible to users.
- Communication Management - Devices: This component manages the communication between HEMS

controller and local devices. It contains multiple communication interfaces and is accessed and configured via microservices. There are several communication protocol options available nowadays. The choice of communication protocol is based on issues related to the communication network, such as the distance between nodes, transmission capacity, and the required link quality. The middleware should be flexible enough to support different protocols, such as Wi-SUN (HAN and FAN), Wi-Fi, Zigbee, and Bluetooth, which are standards commonly adopted. The Communication Management - Devices module allows the implementation of routines to support these standards. However, since it is an open middleware, other interfaces and modes of communication with appliances can be implemented.

IV. MIDDLEWARE IMPLEMENTATION

The proposed middleware was implemented using the following tools and technologies:

- Python Programming Language: Python is a widely used scripting language featuring easy portability over architectures and operating systems;
- SQLite Database: this is a relational database that is lightweight and robust, and widely employed in embedded systems;
- Django REST Framework: This is framework for developing microservices, employing the Python language;
- Hardware: NXP i.Mx6 hardware platform, equipped with a 396 MHz processor, 512MB RAM and 32 GB storage, running a customized Linux operating system, based on the Debian distribution for embedded systems.

As presented in Figure 2, the modules communicate among them indirectly through the databases of the system. Each module operates independently from the others, updating the databases.

The independence of the modules is crucial, as it provides flexibility and scalability to the middleware, allowing the addition of new modules without interfering the operation of other modules.

To achieve the independence of the modules in the core, the concept of multiprocessing was employed, wherein each module is treated as an independent process. Therefore, the main middleware file, named orchestrator (mdw-orq.py), is responsible for creating and managing these processes. The orchestrator initiates and manages the parallel execution of the middleware modules (see Figure 3).

Microservices perform a key role in the proposed middleware. They provide functions enabling the consumption of the generated data, in addition to saving and modifying information in the databases in a independent and scalable way. The APIs provide the capability of changing the behavior of the middleware in a standardized way via the REST architecture.

Figure 4 shows the diagram of the middleware main use case. This diagram presents the interactions of

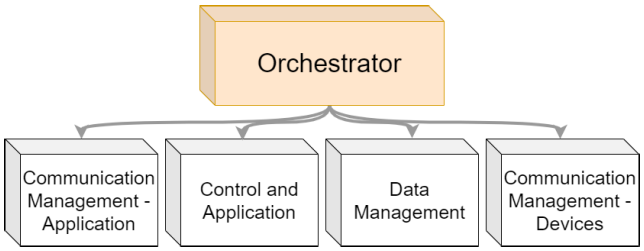


FIGURE 3. Middleware orchestrator.

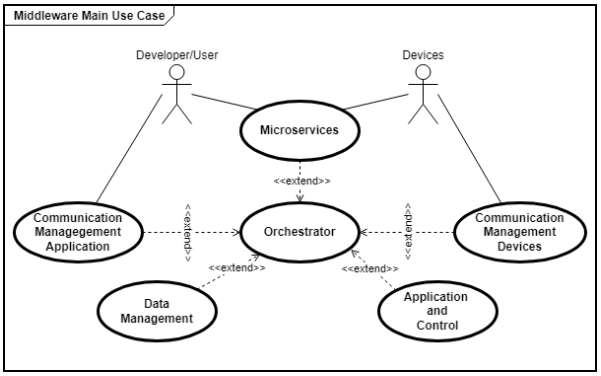


FIGURE 4. Middleware: main use case diagram.

users/developer and the devices directly with the modules use cases. The user communicates with the middleware in the Microservices and the Manage Communications Application use cases. The devices uses the Microservices and the Manage Communications Device to communicate with the middleware. All modules are managed by the Middleware orchestrator use case, which makes all these modules operate in harmony with them and with the system.

Another important aspect related to middleware regards its distribution, management and configuration. In this context, container technology has been considered by several developers if the hardware resources allow it [31]. In an edge computing approach, there may be operational restrictions, such as the operating system and limited hardware, which do not favor the use of container due to additional complexities. The container technology was not used in this project due to computational restrictions of the employed hardware.

In the following sections, we describe some implementation aspects of the modules.

A. COMMUNICATION MANAGEMENT - DEVICES

The Device Communication Management Module contains functions for interface configuration and scheduling of data reading and storage.

The middleware communicates with devices (e.g., smart outlets and smart meters) according to a template defined for each appliance. Different templates are available for implementation, allowing communication via several protocols. In our work, the data are collected from the files generated

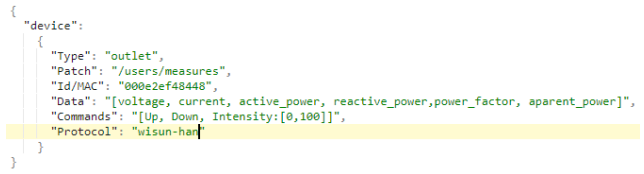


FIGURE 5. Device template.

by the smart outlets. The microservice allows the registration of the template, containing the required information to access the device data. The template is presented in JSON format, as depicted in Figure 5.

When a device adopts the CoAP protocol or supports the HTTP standard, an alternative way to collect data is based on the microservice named receive_data. In this case, the device sends the data via a POST or PUT request. This alternative solution provides interoperability between the middleware and the devices.

The functions of the module are:

- Device_Template(): This function stores and captures the format of the information provided by the devices, besides the information from the device itself.
- Time_Request(): It sets the time between data requests to the device.
- Config_Interface(): It stores and captures information regarding the communication interfaces.
- Receive_data(): It checks for incoming data from devices.
- Store_data(): This function stores the received information in the database.

All functions are accessible via microservices. Therefore, using an mobile App or Web interface, an administrator is allowed to alter the middleware configuration.

B. CONTROL AND APPLICATION MANAGEMENT

This module runs routines using the data stored in the database to perform specific functions or applications, whose results are stored in the database and are accessible via microservices.

An example of application that uses the data stored in the database is load disaggregation. This application uses artificial intelligence algorithms to determine the devices connected to outlets, using information of aggregate energy consumed by the house. More details on this application are found in [32].

C. DATA MANAGEMENT

This module implements the management of the middleware databases. By default, there are four databases for these types of information: (i) raw data collected from the devices; (ii) information genereted by processing the raw data; (iii) configuration data for middleware operation; and (iv) HEMS identification data.

These databases are accessed by all middleware modules, being connecting points among the modules. Therefore, the modules are responsible for the exchange of information among the module and are accessible via microservices.

The database responsible for storing the raw data collected from the devices is implemented using a circular buffer format to achieve an ordered expansion and avoid exceeding a pre-specified database size.

The microservices provide functions to handle the data, enabling data collection and changing middleware operation parameters, affording flexibility and standardized access.

D. COMMUNICATION MANAGEMENT - APPLICATION

This module implements functions to send data to the cloud computing software, through an Internet connection. Presently, two options are available for transferring data: (i) sending to an IoTHub (a paid online repository supported by Microsoft); or (ii) sending to any MQTT Broker. Hence, the following functions are implemented:

- Search_data(): This function is responsible for database queries for preparing the information to be sent to the cloud computing software.
- Send_IotHub(): Responsible for sending the data to the IoT Hub.
- Send_MQTT(): Responsible for sending the data to the MQTT Broker.
- Resend_data(): This function is responsible for searching the data whose transmissions failed, and attempting to resend them.

The destination of data is predefined by default, although it is possible to modify it via the APIs. The data is periodically uploaded to the cloud computing software, with the upload period adjustable via the corresponding microservice.

E. SECURITY MANAGEMENT

The middleware security is implemented through the security protocols provided by the solutions and protocols employed in communication, data storage, and cloud computing. The security strategy consists of

- Access Control: Only registered users are allowed to access the APIs,
- Data Sending Authentication: All data are sent to the cloud computing software using authentication at the MQTT broker,
- Microservices: The microservice architecture inherently encapsulates the functions, not allowing direct access to any portion of the system.

F. MICROSERVICES

Microservices offer the functions provided by the middleware based on the REST standard. Complete documentation on the APIs is available to users and developers. The use of REST standard allows any entity to interact with the middleware.

Each microservice performs specific functions and maintains a different database. The microservices are organized as follows:

- App: Responsible for the communication with cloud computing software and the client applications,

- Devices: Responsible for the communication with the devices (outlets),
- Configure: Responsible for the middleware configuration.

A brief description of each microservice is presented in the following paragraphs.

1) APP MICROSERVICE

This microservice is responsible for the communication with the client applications. It contains four entities: Device, Hems_sys, Zone, and outlet. The Device entity is the table responsible for registering devices and relies on three fields in Django: the device name, whether the device is active or not, and whether it is a generating or consuming device. The Hems_sys entity contains information of the user that must be stored, both in the cloud and in the middleware, providing easier access to it. Therefore, this entity includes the following fields:

- hems_reg_date: Date of HEMS registration,
- userName: User name,
- priceKWh: kWh price,
- hems_last_update: HEMS latest update,
- homeCity, homeStreet, home Neighbour, and homeComplement: Complete HEMS address.

The entities Zone and Outlet refer to the household area where the outlet is installed (living room, kitchen, etc.) and a set of information on the registered outlets, respectively. These set o information includes the outlet type (relay or dimmer), which appliance is connected to, allowing the identification of the outlet.

2) DEVICES MICROSERVICE

It is a microservice for communicating with the devices and contains two different entities: Data and Root Data. The Data entity addresses the information proceeding from the smart outlets. It includes the following fields:

- dev: device identification,
- voltage: Voltage reading by the outlet,
- current: Current reading of the outlet,
- active_power: Active power reading of the outlet,
- reactive_power: Reactive power of the outlet,
- power_factor: Power factor of the outlet,
- device_energy: Total power of the outlet,
- time: Timestamp of the measures.

The Data entity is accessible via a GET or POST request, allowing the devices to send the data using the CoAP standard or any other standard that supports HTTP requests. This feature provides interoperability with multiple existing devices.

The Root Data entity contains smart meter information such as time (date and time of the measurements) and power (the total kWh value). It is noteworthy that all APIs provide the collected information from the outlet organized in JSON format, allowing flexibility when manipulating the variables, via either the application or the cloud computing software.

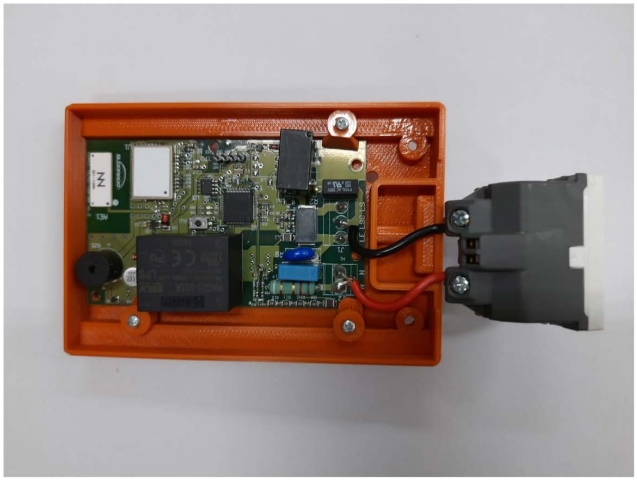


FIGURE 6. Smart outlet used in the proof-of-concept.

3) CONFIGURE MICROSERVICE

It is used to configure the middleware and contains the following entities:

- Data request time: it is the interval within which measurements of the devices are requested;
- Send-to-cloud method: it configures the destination and the method to send the data to the cloud;
- Update: used to configure the repository where the periodic monitoring for middleware updates is performed;
- Time-to-cloud: it sets the time interval for sending information to the cloud.
- Username and password for MQTT broker: it sets the credentials for the use of MQTT broker.
- MQTT Address: it sets the address of the MQTT Broker.

Each microservice features specific functions and accesses a unique database. This characteristic of microservices ensures better resilience and performance of the middleware since the reading and writing processes occur independently, thus avoiding a large number of simultaneous accesses to a single database. Furthermore, new services can be added, without interfering other services already implemented.

V. RESULTS AND DISCUSSION

A. TEST SCENARIO

The proof-of-concept of the middleware was implemented and tested in laboratory and at the university campus, with real devices. Six smart outlets were used to connect an LCD TV, a tube TV, a radio, an air conditioner, and a microwave oven. Fig. 6 shows the smart outlet used, while Figure 7 shows some devices plugged into it.

The smart outlets employ the Wi-SUN HAN protocol to communicate with the controller board, which hosts the middleware. Fig. 8 displays the controller board. Measurements of voltage, current, power factor, active, reactive and apparent power are transmitted by the outlets to the controller every 5 seconds. The middleware uploads the collected

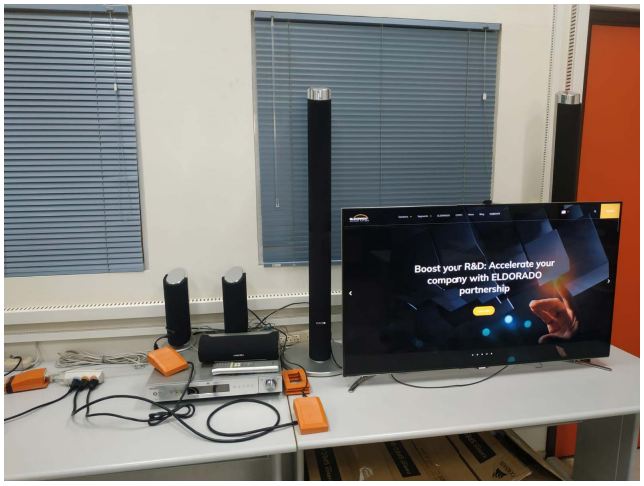


FIGURE 7. Devices connected to smart outles.

FIGURE 8. Controller board.

data every 1 minute to the cloud software via the MQTT protocol.

Figure 9 shows the communication topology used in the test scenario's. The outlets communicate with the controller via WISUN-HAN. In the controller, the Middleware receives and processes the data, as shown in Figure 10. This information is made available to client applications (App and Cloud Software) via Microservices, using the REST standard and the MQTT protocol. Figure 11 shows the App displaying the data obtained from an outlet over a period.

The implemented version of the middlewae requires 250 kB of disk space, excluding the databases. The total disk space required is around 111 MB, including the Python installation, the libraries required for operation, and the latest version (1.21) of NGINX web server.

The most relevant database in the tests was the one used to store the device measurements, as it receives a large amount of data and can affect the middleware performance. The default size of this database is adjusted to store data collected in a period of five days. Tests in the laboratory

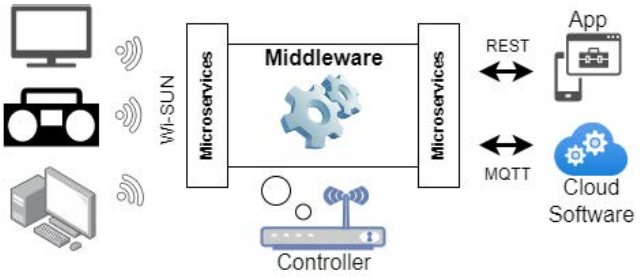


FIGURE 9. Communication topology.

```
\\*****Devices Communication Management Module******\\
['001D129100035AB3', '001D129100035EED']
[(3,)]
3
dev_id: 3
date: 2022-08-11 21:36:44
voltage: 127.2679214477539
current: 0.06432999670505524
power factor: 0.5580000281333923
active power: 4.576000213623047
reactive power: -2.171999931335449
apparent power: 8.20199966430664
novo consumo: 0.2128390375400988
[(6,)]
6
dev_id: 6
date: 2022-08-11 21:36:48
voltage: 218.0417938232422
current: 1.4074499607086182
power factor: 0.9800000190734863
active power: 300.9670104980469
reactive power: 60.3129997253418
apparent power: 307.0639953613281
```

FIGURE 10. Outlets data received in middleware.

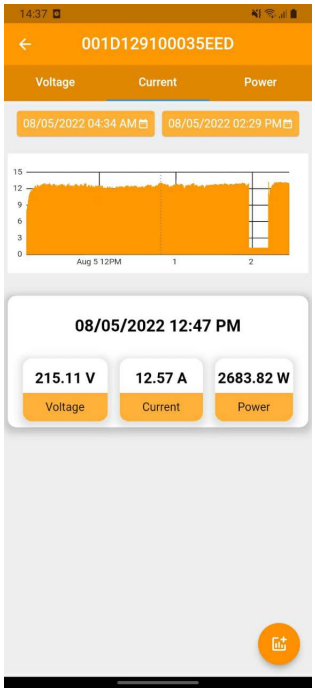


FIGURE 11. App showing outlet data.

and in campus trials showed that 17 MB is enough for this period length. However, the database size can be configured

TABLE 2. Results of Scenario 1.

	Scenario 1					
Time Between Requests	**0.5s**			**5s**		
Simultaneous Requests (Devices)	**Error (%)**	**Average Time (seconds)**	**Median (seconds)**	**Error (%)**	**Average Time (seconds)**	**Median (seconds)**
10	0	4,52	4,66	0	4,15	4,37
30	0	9,78	12,88	0	3,26	3,61
50	0	6,23	6,54	0	5,77	6,23
100	0	22,64	15,78	0	17,76	13,12
250	0	44,86	33,93	0	43,06	33,52
400	4.80	65,30	58,48	4.20	74,15	85,50
600	38.50	39,21	47,94	36.20	48,85	50,69

TABLE 3. Results of Scenario 2.

	Scenario 2					
Time Between Requests	**0.5s**			**5s**		
Simultaneous Requests (Devices)	**Error (%)**	**Average Time (seconds)**	**Median (seconds)**	**Error (%)**	**Average Time (seconds)**	**Median (seconds)**
10	0	3,07	1,69	0	1,10	1,69
30	0	4,48	4,64	0	4,09	4,46
50	0	10,56	7,79	0	6,87	7,46
100	0	21,32	15,71	0	17,16	15,
250	0	50,11	38,84	0	49,46	39,18
400	4.30	82,46	93,02	4.20	86,71	92,64
600	36.20	54,56	61,04	37.80	46,41	58,70

via the microservice. A circular buffer was employed to delete older data after five days, keeping the size of the measurement database constant.

Hence, the total disk size of the middleware is approximately 130 MB. The disk footprint is an issue that requires attention for implementation on hardware platforms with limited storage space.

During the laboratory tests, control logs was generated to identify faults and interruptions in the system. In a test period of 30 days, emulating a real scenario, the middleware performed continuously with no faults.

B. MIDDLEWARE PERFORMANCE ANALYSIS

The middleware performance were tested using the approach presented in [26] and the JMeter framework [33].

No standard metrics exist to analyze the performance of a middleware, and therefore metrics are defined according to the application context [26]. In this work, the chosen metrics aim to examine whether the proposed middleware provides the services required for HEMS application, even when the computational resources of the hardware platform are limited. In this sense, the following metrics were chosen:

- CPU usage: The percentage of CPU usage by user software (including the middleware) and system software;
- RAM Memory Usage: Used memory and free memory;
- Error Rate: Percentage of failed service requests;
- Average Response Time: Time elapsed to respond to the requests addressed to the middleware.

The microservice Receive_data was used in the performance tests, as this microservice is responsible for receiving data from the devices via a POST operation.

TABLE 4. Scenarios.

	Scenario 1	Scenario 2
Time between requests	0.5s, 5s	0.5s, 5s
Simultaneous requests	10, 30, 50,100, 250, 400, 600	10, 30, 50, 100, 250, 400, 600

Besides receiving the data, the microservice saves the information in the local database and subsequently sends a response code to the answered HTTP request. The size of each data package is 392 bytes.

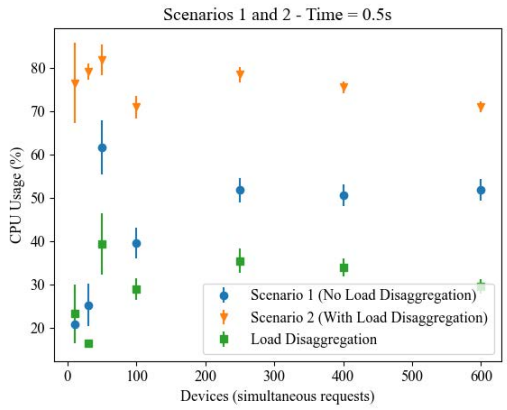
Two different scenarios were tested:

- Scenario 1 (S1): The middleware operates following its basic structure, with no additional applications;
- Scenario 2 (S2): The middleware operates with load disaggregation as an additional application.

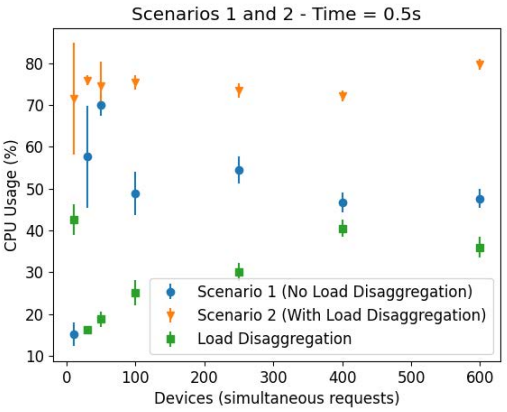
The objective is to evaluate and compare the middleware performances under a default configuration (S1) and when additional computational resources are required (S2). For both scenarios, different numbers of simultaneous requests (which is related to the number of outlets) and time intervals between requests were tested, as shown in Table 4.

The tests for the 0.5 second interval aimed to overload the target microservice and check the middleware performance in a stress scenario. On the other hand, the 5-second interval is considered to be a more realistic scenario. Ten rounds of tests were performed for different numbers of simultaneous requests: 10, 30, 50, 100, 250, 400, and 600.

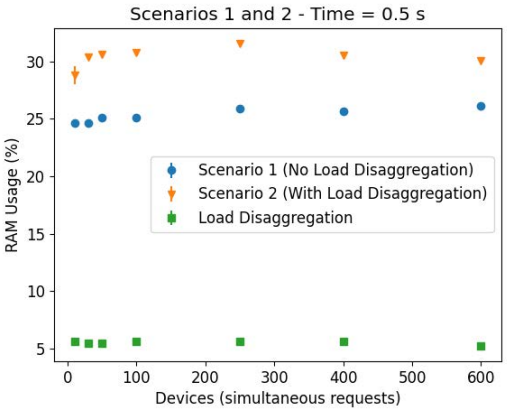
The results of the tests for Scenarios 1 and 2 and metrics error rate and average and median time to respond all requests are presented in Tables 2 and 3. We can see that no error occurred up to 250 devices, for both time intervals



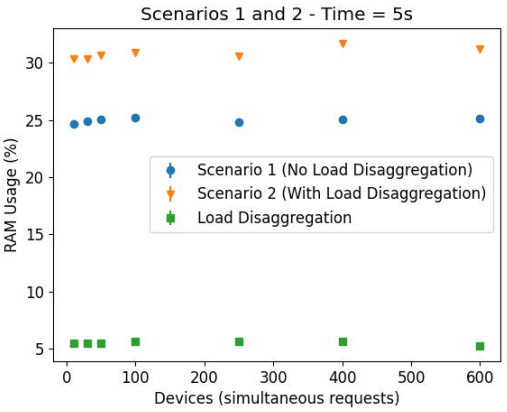
(a) CPU usage for 0.5 s time interval between requests.



(b) CPU usage for 5 s time interval between requests.



(c) RAM usage for 0.5 s time interval between requests.



(d) RAM usage for 5 s time interval between requests.

FIGURE 12. **Result of CPU and RAM tests.**

TABLE 5. **Middleware minimum requirements.**

Software	CPU	RAM	Hard Disk
Linux-based OS Python 3.8 Django 3.2.4 with DRF SQLite 3 Nginx with uwsgi	Arm Cortex single core 396 MHz	512MB	8 GB

tested. We can then assert that the middleware can serve up to 250 devices in parallel with no errors. For 600 devices or more, the error rate considerably increases due to the overload of requests and the limited hardware and software resources.

The average response times presented in Tables 2 and 3 are the time to respond to the total number of requests tested, i. e., to serve 10, 30, 50, 100, 250, 400, and 600 requests. The operation requested by these requests is the most costly one, involving receiving the HTTP request, writing to the database, and sending the HTTP response code. Therefore, the response times presented in Tables 2 and 3 are suitable and fulfill the requirements of most applications.

An important conclusion from the tests was that both scenarios yielded remarkably similar results, demonstrating that the execution of an additional application on the middleware (e.g., load disaggregation in our case) did not affect performance.

Figure 12 shows the CPU and RAM consumption results for Scenarios 1 and 2. This figure also shows the consumption of the load disaggregation application. Figures 12 (a) and (b) show that, as expected, the highest consumption occurred in Scenario 2 (when load disaggregation is considered), but the maximum consumption did not exceed 85% of the available CPU capacity. Moreover, no significant difference was observed between the results for 0.5s and 5s time intervals between requests. In scenario 1, consumption did not exceed 65% usage. The disaggregation script alone consumes approximately 40% of the CPU capacity in the worst case.

Graphs Fig. 12(c) and (d) depict the RAM consumption for both scenarios. In Scenario 2 the middleware consumes approximately 30% of the available memory, while in Scenario 1 this figure reduces to approximately 25% of the resources. The disaggregation script consumes

approximately 5% of RAM. Notably, no significant difference is observed between the scenarios with 0.5 s and 5 s time intervals between requests.

These results show that the proposed middleware can respond to up to 250 simultaneous requests with no errors, with a reasonable response time, and consuming reasonable computational resources, regardless of employing limited hardware/software. Even in the stress test with 400 simultaneous requests, the error rate is under 5%. Therefore, the proposed middleware can provide reliable service for up to 400 devices in a real scenario.

Therefore, the specifications of the hardware and software components used in these tests (shown in Table 5) are the minimum requirements to guarantee a good performance of the proposed middleware.

The estimated cost of the hardware components shown in 5, using information from websites of major hardware distributors at low volumes, is under US$ 40. Hence, the proposed middleware is an affordable option for large-scale applications, providing all the features required for application in HEMS.

VI. CONCLUSION

This work introduced a middleware for a Home Energy Management System (HEMS). The architecture is based on established concepts in the literature and was developed considering specific requirements for a HEMS application, providing solid grounding and a good perspective for further usage.

The middleware was designed following the edge computing perspective, then providing services closer to the end-user. Besides, the proposed middleware aims at low-cost hardware platforms with computational constraints in terms of storage, processing, and communication.

The concept of microservices was employed to offer the middleware functionalities by using standard forms to access user and configuration data. Additionally, the independence of the services, intrinsic to the use of microservices, provides greater robustness, performance, and interoperability.

Field trials were performed to simulate real application scenarios as a proof of concept. The middleware proved to be functional and fulfilled the requirements, achieving good performance concerning the proposed tasks.

Furthermore, performance tests were carried out with the middleware that showed its capability to support a large number of devices and, therefore, its ability to be used in small, medium, and large applications. The minimum hardware and software requirements were determined to guide potential users and developers based on the performance tests. Moreover, the proposed middleware is open at a Github project (RT-DSP/SHArM), promoting the specifications for HEMS applications.

The middleware presented in this paper is set to be improved and tested in new scenarios and applications on field trials, continuing the research work. As it was developed for low cost hardware, using flexible architecture such as microservices, this middleware can be a valuable contribution for developers of IoT solutions, especially those related to rational use of resources.

REFERENCES

[1] S. Nižetić, P. Šolić, D. L.-D.-I. González-de-Artaza, and L. Patrono, "Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future," *J. Cleaner Prod.*, vol. 274, Nov. 2020, Art. no. 122877.

[2] A. Polianytsia, O. Starkova, and K. Herasymenko, "Survey of hardware IoT platforms," in *Proc. 3rd Int. Sci.-Practical Conf. Problems Infocommun. Sci. Technol. (PIC ST)*, Oct. 2016, pp. 152–153.

[3] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using Docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.

[4] T. Vaiyapuri, V. S. Parvathy, V. Manikandan, N. Krishnaraj, D. Gupta, and K. Shankar, "A novel hybrid optimization for cluster-based routing protocol in information-centric wireless sensor networks for IoT based mobile edge computing," *Wireless Pers. Commun.*, vol. 116, pp. 1–24, Jan. 2021.

[5] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "IoT design patterns: Computational constructs to design, build and engineer edge applications," in *Proc. IEEE 1st Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, Apr. 2016, pp. 277–282.

[6] Y. K. Teoh, S. S. Gill, and A. K. Parlikad, "IoT and fog computing based predictive maintenance model for effective asset management in industry 4.0 using machine learning," *IEEE Internet Things J.*, early access, Jan. 11, 2021, doi: 10.1109/JIOT.2021.3050441.

[7] V. Velepucha and P. Flores, "Monoliths to microservices–Migration problems and challenges: A SMS," in *Proc. 2nd Int. Conf. Inf. Syst. Softw. Technol. (ICI2ST)*, Mar. 2021, pp. 135–142.

[8] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of middleware for Internet of Things: A study," *Int. J. Comput. Sci. Eng. Surv.*, vol. 2, no. 3, pp. 94–105, 2011.

[9] J. Han, C.-S. Choi, and I. Lee, "More efficient home energy management system based on ZigBee communication and infrared remote controls," *IEEE Trans. Consum. Electron.*, vol. 57, no. 1, pp. 85–89, Feb. 2011.

[10] J. Han, C.-S. Choi, W.-K. Park, I. Lee, and S.-H. Kim, "Smart home energy management system including renewable energy based on ZigBee and PLC," *IEEE Trans. Consum. Electron.*, vol. 60, no. 2, pp. 198–202, May 2014.

[11] A. R. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. Alikarar, "A smart home energy management system using IoT and big data analytics approach," *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 426–434, Nov. 2017.

[12] S. Zhou, Z. Wu, J. Li, and X. Zhang, "Real-time energy control approach for smart home energy management system," *Electr. Power Compon. Syst.*, vol. 42, nos. 3–4, pp. 315–326, 2014.

[13] H. Shareef, E. Al-Hassan, and R. Sirjani, "Wireless home energy management system with smart rule-based controller," *Appl. Sci.*, vol. 10, no. 13, p. 4533, Jun. 2020.

[14] P. Pawar and M. TarunKumar, "An IoT based intelligent smart energy management system with accurate forecasting and load strategy for renewable generation," *Measurement*, vol. 152, Feb. 2020, Art. no. 107187.

[15] B. Setz, S. Graef, D. Ivanova, A. Tiessen, and M. Aiello, "A comparison of open-source home automation systems," *IEEE Access*, vol. 9, pp. 167332–167352, 2021.

[16] S. Nagalakshmi, M. Prabha, R. Senthamarai, and G. Rohini, "Design and implementation of Aurdino based smart home energy management system using renewable energy resources," *Int. J. ChemTech Res.*, vol. 10, no. 6, pp. 696–701, 2017.

[17] E. D. L. Oliveira, R. D. Alfaia, A. V. F. Souto, M. S. Silva, C. R. L. Francês, and N. L. Vijaykumar, "SmartCoM: Smart consumption management architecture for providing a user-friendly smart home based on metering and computational intelligence," *J. Microw., Optoelectron. Electromagn. Appl.*, vol. 16, no. 3, pp. 736–755, Sep. 2017.

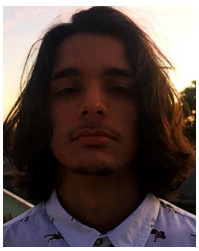
[18] J. Rodríguez-Molina and D. M. Kammen, "Middleware architectures for the smart grid: A survey on the state-of-the-art, taxonomy and main open issues," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2992–3033, 4th Quart., 2018.
[19] B. Mahapatra and A. Nayyar, "Home energy management system (HEMS): Concept, architecture, infrastructure, challenges and energy management schemes," *Energy Syst.*, vol. 13, pp. 643–669, Aug. 2022.
[20] I. Machorro-Cano, G. Alor-Hernández, M. A. Paredes-Valverde, L. Rodríguez-Mazahua, J. L. Sánchez-Cervantes, and J. O. Olmedo-Aguirre, "HEMS-IoT: A big data and machine learning-based smart home system for energy saving," *Energies*, vol. 13, no. 5, p. 1097, Mar. 2020.
[21] M. A. A. D. Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, "A reference model for Internet of Things middleware," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 871–883, Apr. 2018.
[22] C. M. Mohammed and S. R. Zebaree, "Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review," *Int. J. Sci. Bus.*, vol. 5, no. 2, pp. 17–30, 2021.
[23] M. A. A. D. Cruz, J. J. P. C. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, "Performance evaluation of IoT middleware," *J. Netw. Comput. Appl.*, vol. 109, pp. 53–65, May 2018.
[24] P. Bellavista, C. Giannelli, S. Lanzone, G. Riberto, C. Stefanelli, and M. Tortonesi, "A middleware solution for wireless IoT applications in sparse smart cities," *Sensors*, vol. 17, no. 11, p. 2525, Nov. 2017.
[25] I. Ungurean, N. C. Gaitan, and V. G. Gaitan, "A middleware based architecture for the industrial Internet of Things," *KSII Trans. Internet Inf. Syst.*, vol. 10, no. 7, pp. 2874–2891, 2016.
[26] A. S. Gaur, J. Budakoti, and C.-H. Lung, "Design and performance evaluation of containerized microservices on edge gateway in mobile IoT," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 138–145.
[27] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 1–20, Feb. 2017.
[28] A. Palade, C. Cabrera, G. White, M. A. Razzaque, and S. Clarke, "Middleware for Internet of Things: A quantitative evaluation in small scale," in *Proc. IEEE 18th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2017, pp. 1–6.
[29] R. Ramakrishnan and L. Gaur, "Smart electricity distribution in residential areas: Internet of Things (IoT) based advanced metering infrastructure and cloud analytics," in *Proc. Int. Conf. Internet Things Appl. (IOTA)*, Jan. 2016, pp. 46–51.
[30] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen, "MOSDEN: An Internet of Things middleware for resource constrained mobile devices," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, Jan. 2014, pp. 1053–1062.
[31] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the Internet of Things," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–6.
[32] D. A. M. Lemes, T. W. Cabral, G. Fraidenraich, L. G. P. Meloni, E. R. De Lima, and F. B. Neto, "Load disaggregation based on time window for HEMS application," *IEEE Access*, vol. 9, pp. 70746–70757, 2021.
[33] JMeter. (2021) *JMeter*. Accessed: Jun. 10, 21. [Online]. Available: https://jmeter.apache.org/

LUIZ C. B. C. FERREIRA received the B.Sc. degree in computer science and the master's degree in electrical engineering. He is currently pursuing the Ph.D. degree in electrical engineering with the State University of Campinas—UNICAMP, Brazil. He is also a Lecturer at the Federal Institute of Education, Science and Technology of South Minas Gerais, Brazil. His research interests include wireless sensor networks, the Internet of Things, communication protocols, and middleware for IoT applications.

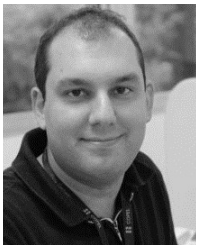
ANDREZA DA ROSA BORCHARDT received the B.Sc. degree in electrical engineering from the Federal Institute of Education, Science and Technology of Rio Grande do Sul (IFSul), Brazil, in 2020. She is currently pursuing the M.Sc. degree in electrical engineering with the State University of Campinas (UNICAMP) focusing in middleware for IoT and microsservices.

GUSTAVO DOS SANTOS CARDOSO received the B.Sc. degree in electrical engineering from the Federal Institute of Science and Technology Sul-Rio-Grandense (IFSul), Brazil, in 2020. He is currently pursuing the M.Sc. degree in electrical engineering with the State University of Campinas (UNICAMP), Brazil, focusing in IoT and middleware for IoT applications.

DIMAS AUGUSTO MENDES LEMES received the B.Sc. degree in telecommunications engineering from the Federal University of São João Del-Rey (UFSJ), Brazil, in 2014, and the M.Sc. degree in electrical engineering from the State University of Campinas (UNICAMP), Brazil, in 2018, where he is currently pursuing the Ph.D. degree in electrical engineering focusing in machine learning, load disaggregation, and MIMO systems.



GABRIEL RODRIGUES DOS REIS DE SOUSA is currently pursuing the bachelor's degree in electrical engineering with the State University of Campinas (UNICAMP). He is also a Technician in maintenance electrician and a Mathematics Teacher at the Popular Cram School. He has knowledge in high level desktop and web software development focused on end user and also in the embedded systems programming.



FERNANDO BAUER NETO received the degree in electrical engineering from the State University of Santa Catarina (UDESC) and the M.B.A. degree in project management from Getúlio Vargas Foundation (FGV). He worked as a Technology Engineer in a multinational industry and currently works as an Electrical Engineer with the Innovation Management Department, Companhia Paranaense de Energia—COPEL, acting as a Project Coordinator for the Energy Efficiency Program (PEE) and the Research and Development Program (R&D), both regulated by the National Agency of Electric Energy (ANEEL). He received the title of Specialist in energy efficiency from the Federal Technological University of Paraná (UTFPR).

EDUARDO RODRIGUES DE LIMA received the degree in EE from the University of São Paulo State—UNESP and the Ph.D. degree from the Technical University of València—UPV, Spain. He is currently the Research and Development Manager of the Exploratory Hardware Design Department, Eldorado Research Institute, and a Visiting Professor at UNICAMP, Campinas, Brazil. His current research interests include the implementation and theoretic aspects of physical layers of wireless and wired communications systems. He has more than 20 years of experience in telecommunications systems. Currently, he coordinates several research and development projects related to microelectronics, embedded system, smart grid, and IoT. He is also a MCTI/CNPq Fellow of Technological Productivity.

GUSTAVO FRAIDENRAICH was born in Pernambuco, Brazil, in 1977. He received the Graduate degree in electrical engineering from the Federal University of Pernambuco (UFPE), Brazil, and the M.Sc. and Ph.D. degrees from the State University of Campina—UNICAMP, Brazil, in 2002 and 2006, respectively. From 2006 to 2008, he worked as a Postdoctoral Fellow at Stanford University (Star Lab Group), USA. Currently, he is an Assistant Professor at UNICAMP. His research interests include multiple antenna systems, cooperative systems, radar systems, and wireless communications in general. He has been an Associate Editor of the *ETT* journal for many years. He was a recipient of the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) Young Researcher Scholarship, in 2009. He has published more than 50 international journal articles and more than 100 conference papers of the first line. He is the President of the Technical Board of Venturus Company, a branch from Ericsson Company.

PAULO CARDIERI received the B.S. degree from the Mauá School of Engineering, Brazil, in 1987, the M.Sc. degree from the State University of Campinas, Campinas, Brazil, in 1994, and the Ph.D. degree from the Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, all in electrical engineering. He is currently an Associate Professor at the School of Electrical and Computer Engineering, State University of Campinas (UNICAMP). Prior to joining the Faculty of UNICAMP, he was with CPqD Foundation, Campinas, Brazil, where he was involved with several research projects on communications, including satellite and wireless communications. From November 1991 to August 1992, he was a Visiting Researcher at the Centro Studi e Laboratori Telecomunicazioni, Turin, Italy. His current research interests include wireless *ad-hoc* networks, sensor networks, and modeling of communication systems.

LUÍS GERALDO P. MELONI received the B.E. degree in electrical engineering/electronics and the M.Sc. degree in electrical engineering/automation from the Unversity of Campinas—UNICAMP, Brazil, in 1980 and 1982, respectively, and the Ph.D. degree in automation/signal processing from the Université of Nancy I, France, in 1985. He is currently a Professor at the School of Electrical and Computer Engineering (FEEC), UNICAMP. He has vast academic and industrial experience in telecommunications and worked previously in several telecommunication companies. He was a member of the Council of the Brazilian System for Digital Television Forum, from 2012 to 2016, for coordinating works of standardization of technologies for interactive channel at the Brazilian Association of Technical Standards (ABNT). He was the Coordinator of the Faculty Outreach at FEEC for four years. He was a Lecturer at the University of Brasilia (1990–1993). He is also the Director of University Outreach at UNICAMP. His current research interests include new technologies for wireless communications, hypercomplex algebra, artificial intelligence, middleware architectures for home energy management systems, the Internet of Things, and software-defined radio technologies. He has many publications in international journals and symposia and has also been an instructor in continuing education programs for telecommunication professionals.