An IoT Middleware Template Proposal

Gustavo S. Cardoso, Luiz C. B. C. Ferreira, Fernando Bauer N., Eduardo R. Lima, Luís G. P. Meloni and Paulo Cardieri

***Abstract*— There are several different IoT middleware platforms available for IoT systems. However, most of these platforms require high computational hardware to be implemented, making their use in low-cost and computationally restricted devices difficult. This motivated us to develop an IoT architecture and template suitable for computationally restrained hardware to be a tool that can facilitate the development of new IoT middlewares. We tested the proposed architecture in two application scenarios, using hardware with low computational power. The results, in terms of RAM and CPU usage rates and response time, show the proposed architecture makes good use of the available resources.**

***Keywords*— Middleware, Internet of Things, Frameworks.**

I. Introduction

In the IoT scene, the basic principle of operation consists in things communicating with each other. Currently, there are a lot of IoT devices from different manufacturers available in the market [1]. This diversity brings one of the greatest issues in IoT, which is the need for interoperability among several products from different manufacturers that use different protocols of communication [2].

In this context, the middleware platforms have a crucial role for interoperability between systems and devices. The middleware acts as a layer of software (SW), which interconnects devices and applications in an IoT solution. It also offers a standardized way of access to data and services. Therefore, details like device implementation and configuration, such as network protocols and hardware (HW) definitions, are abstracted from the application developer's point of view. This abstraction helps the middleware to bring interoperability for the development of the system [3].

Another important issue in the IoT scenario is the role played by computationally restrained hardware platforms/devices used in IoT applications. These devices are generally low-cost, which enable the deployment of large-scale systems [4]. The important role played by this kind of hardware is further reinforced by the tendency of using an edge computing approach in some IoT cases [5], [6], [7]. This approach has many advantages, such as reduction of dependency from external communications and the response time between parts of the system [5].

A great number of middleware platforms are created for Cloud or Fog computing paradigms [8]. These approaches limit or preclude their use by restrained hardware. Furthermore, the lack of middleware for edge computing creates a scenario where developers need to develop their own solutions for a specific application [5], [6], [7], [9], [10].

In light of the scenario discussed above, this work proposes an edge IoT middleware architecture and template, aimed to facilitate the development of edge middleware solutions in IoT. The architecture was designed to be suitable for computationally restrained hardware. Additionally, two similar middleware platforms using Python and the Django REST framework were developed. These middleware platforms follow the proposed architecture and template requirements and serve as a proof of concept of this work.

The proposed Middleware Template is part of 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 this work is organized as follows: Section II presents a literature review and related works on edge middleware for IoT. Section III describes the proposed middleware architecture and template. Section IV details the experiments and the results achieved. Finally, in Section V the main conclusions are presented.

II. Literature Review and Related Works

In an IoT environment, the middleware acts as a translator, being responsible by making applications and/or devices (APPs) exchange data with each other exclusively through the middleware. Therefore, a middleware abstracts the complexities of communications. A scenario without a middleware makes the communication between different APPs a tricky task, as every time a certain APP makes a requisition to other APP it will have to communicate through specific protocols of this other APPs. This process generates a situation in which the developer needs to create not only the applications, but also ways for them to communicate with other APPs. [11]. Figure 1 illustrates the scenarios with and without middleware.

Therefore, middleware is a layer of software that enables different systems to interact with each other and work together. Thus, the role of a middleware in IoT scenarios is essential for the creation of fast, scalable, and secure solutions, enabling simplicity and modularity [11], [12].

In [11], [12], the authors discuss general requirements for IoT middleware development, using distinct approaches. In [11], the authors classify the requirements as functional and non-functional; in [12], the authors view the requirements as architecture and services-related, the latter being further divided into functional and non-functional. The main requirements found in these works are: scalability, interoperability, security

Gustavo Cardoso, Luiz Ferreira, Luís Meloni e Paulo Cardieri, Departamento de Comunicações, Universidade Estadual de Campinas, Campinas-SP, e-mail: {gustavo_16a@hotmail.com, carlinho@decom.fee.unicamp.br, meloni@unicamp.br, cardieri@unicamp.br}; Fernando Bauer, Copel Distribuição, Curitiba-PR, e-mail: fernando.bauer@copel.com; Eduardo Lima, Instituto de Pesquisas Eldorado, Campinas, SP, eudardo.lima@eldorado.org.br.

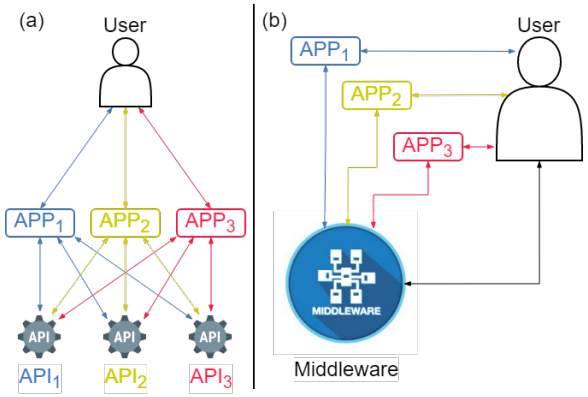

Fig. 1. Communication among APPs: (a) without middleware (b) with middleware [11]

and privacy, data management, real-time operation and context awareness.

There are several different IoT middleware proposals available in the literature, but they generally have minimum hardware requirements, which makes their use almost impossible in an edge computing scenario. Even in specific middleware solutions using an edge computing approach, most of the works employ hardware with minimal computational restrictions. However, this issue is relevant and works found in the literature show the importance of middleware proposal for computationally restrained hardware [10], [7], [5]. Table I lists relevant works in the area of middleware for edge devices applied in several different scenarios, with the respective specification of the hardware used to implement each one.

TABLE I
USED HARDWARE IN EDGE MIDDLEWARE APPROACHES IN LITERATURE

Ref.	Hardware
[5]	Raspberry Pi3 - ARM Cortex-A53 4-Core 1.2 GHz 1GB RAM
[8]	Intel Xeon E5-1620 v3 4-Core 3.5GHz 32GB RAM
[6]	Raspberry Pi3
[7]	AMD Athlon (tm) 64 X2 2-Core 2.21GHz 1GB RAM
[9]	Raspberry Pi3

In [5], the authors proposed a modular and scalable architecture based on microservices and docker for all IoT layers (Edge, Fog, Cloud) at the same time. The Raspberry Pi3 was used as edge devices, being the hardware with minimum computational power used.

In [8], the authors presented metrics for comparing 11 different middleware proposals. All those middlewares were tested in a system with high computational power. Metrics like packet size, response-time, error rate in requests were used to compare the performance of each middleware.

The authors of [6] proposed a 3-layer architecture to support IoT Wireless applications. The Raspberry Pi3 was the minimal hardware used with this middleware.

In [7], a comparison was made between five different middlewares. Data like CPU usage, RAM usage, and bytes send by requisition were used for comparison. The Open DDS middleware achieved the best results.

In [9] is proposed a middleware architecture using docker and microservices running on a Raspberry Pi3 to limit the amount of sent data in different communication protocols.

TABLE II
MINIMUM REQUIREMENTS FOR POPULAR IOT MIDDLEWARES

Ref.	Requirements
oneM2M [13]	It is a set of specifications. One of the most famous applications, the Eclipse OM2M needs an environment of execution JAVA 1.7 and Maven 3
dojot [14]	For a 500 devices setup with intervals of 15s. Dojot lighter version needs: 1 CPU x86-64 3,5 GHz, 1GB RAM 10GB of free space on disk
Kaa [15]	SO 64-bits; 256MB RAM third-party remotely. or 4 GB RAM third-party on the same device.
Zabbix [16]	Setup with MySQL InnoDB for 500 devices suggests a dual core CPU and 2GB RAM and at least 256 MB of free-space on disk
EcoDif [17]	CPU 2,5 GHz; 4GB RAM; 100GB of disk; SO: Ubuntu 12.04 or superior or Windows 2003 Server / XP (or superior)

Table II shows the minimum hardware necessary for ready-made popular middleware platforms. These platforms were designed to run on powerful devices with high computational capacity. The platforms in this table and those in Table I reveal the necessity of a tool that could ease the development of edge middleware platforms with low minimum hardware requirements to run on low-cost and computationally restricted platforms.

III. MIDDLEWARE ARCHITECTURE AND TEMPLATE PROPOSAL

The middleware template and architecture were created for a scenario of data aggregation and command sending to IoT devices, which take measurements.

The middleware architecture was developed based on some requirements in the presented literature. Other requirements were proposed due to the context in which the proposal was created. For a data aggregation edge middleware, some of the main requirements found in the related works are: Data management, scalability, interoperability and security. In addition to these requirements, other that helped the architecture development were: low resources consumption, high portability to use in HW and SW, ease of maintenance and new functionalities aggregation.

From these requirements, the modular architecture was created (see Figure 2) to supply the demands of requirements and allow the Template implementation. It is worth mentioning that a given requirement does not create a particular module in the architecture, but rather a module works to partially or fully fulfill a requirement, thus guaranteeing the middleware operation according to the requirements

The middleware is composed by seven components, which can communicate with each other. The idea of this modularized architecture is to facilitate the additions of new modules and the location of operation faults. Each of these modules must perform tasks independently of others, increasing the middleware operation reliability and contributing to its scalability. Each element and its basic functionality are presented next.

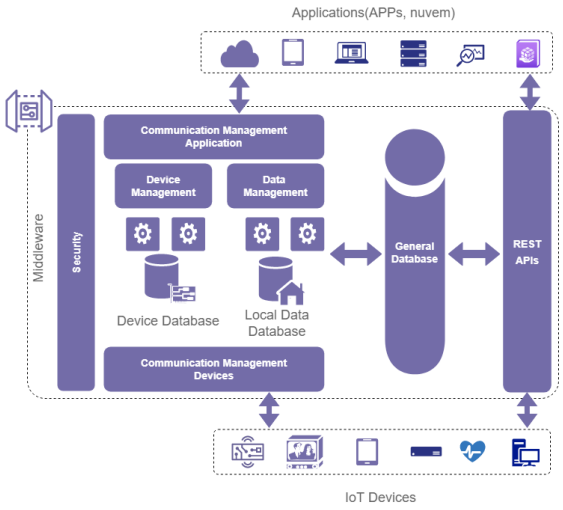

Fig. 2. Middleware Architecture proposed

- Communication Management - Application: This module handles middleware communications with external applications for data transfer or reception via protocols such as MQTT and CoAP. Furthermore, it manages the requisitions of data solicitation made by these applications.
- Data Management: This module manages and stores data collected from devices.
- Device Management: This module is responsible for managing the connected devices via low-level protocols (e.g., RS232 and Wi-SUN), as well as devices that use high-level protocols (REST APIs). It is also responsible for providing status and the available resources of devices.
- Communication Management - Devices: It is responsible for the low-level communications with devices. The module must provide methods for data reception, communication protocol configurations, devices packets registration.
- Security: Security is a critical component and must be present in all modules. There is no single security tool capable of handling all the system, thus it is necessary to use different techniques by the middleware modules.
- REST APIs: The middleware has REST APIs that provide functions for sending and receiving data, changing middleware operating parameters, among others functions. The REST APIs provide a standardized method for requests via HTTP protocol in the middleware, providing interoperability with a wide range of devices.

The Middleware Template, which is a set of UML diagrams, ease the implementation of the architecture in any technology, and creates a path to follow during all the development of the solution. Using the template as reference and modifying it according to the application simplify the process, reduce the time of development and facilitate fault location.

Figure 3 shows the general use case diagram for a data aggregation edge middleware. This diagram presents the basic functions that the middleware should perform. The middleware must provide this functions to the user/developer who will

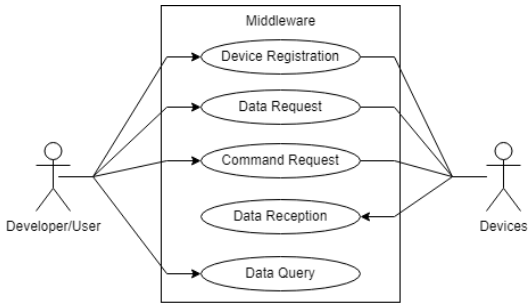

Fig. 3. Middleware Template general use case diagram

utilize them for retrieving data and sending commands from and to devices. The use cases will be described next.

- Device Registration: The user can (un)register a device in the middleware.
- Data Request: The user requests data from a particular device to the middleware.
- Command Request: The user requests a certain command from a device to the middleware.
- Data Reception: The device returns the information from a requested command or measurement to the middleware.
- Data Query: The user queries the available data on middleware via some way of access.

Figure 4 shows the middleware modules and the relationships among them. The Middleware Template is basically composed of six modules and the REST APIs.

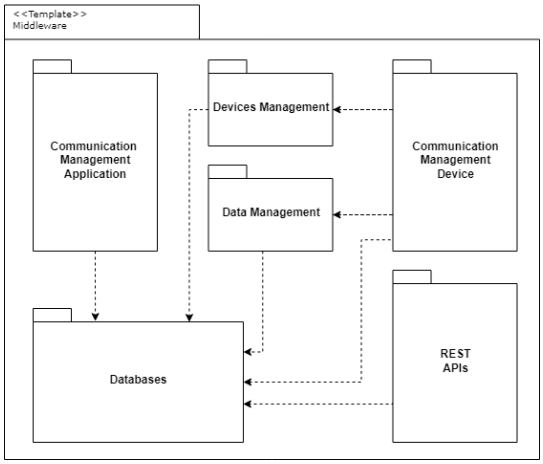

Fig. 4. Middleware Template Package diagram

The device communication module is responsible for receiving data and sending commands to devices. This module is also modularly implemented, as it allows different communications protocols, including new ones, to be used, such as Wi-Fi, Bluetooth, Zigbee, TCP/IP, RS232/485, Wi-SUN FAN/HAN. This facilitates the use of heterogeneous devices, meeting the interoperability requirements.

Figure 5 shows the Middleware Template main use case diagram. This diagram presents the interactions of the user/developer and the devices directly with the modules use cases. The user communicates with the middleware in the REST APIs and the Manage Communications Application use

cases. The devices uses the APIs and the Manage Communications Device use case to communicate with the middleware. All modules use cases are managed by the Middleware Template orchestrator use case, which makes all these modules operate in harmony with them and with the system

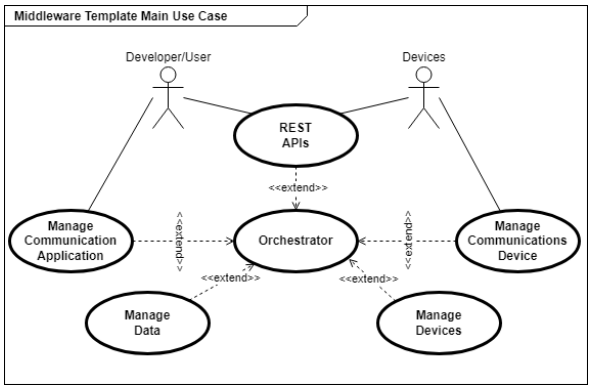

Fig. 5. Middleware Template: main use case diagram.

.

IV. Experiments and Outcomes

Two case studies were developed in order to proof the concept of the proposal. Two middlewares with similar structures were developed for the tests. One for a **smart outlet system** and another for a **biomedical devices system**. The middlewares modules for these two cases were developed in Python, with the databases implemented in SQLite3 and the APIs were implemented using the Django REST framework. The tests were carried out in a NXP i.MX6ULL hardware platform, which is equipped with a single-core Cortex-A7 ARM CPU and 512MB RAM, being a very resource limited hardware, even more than the Raspberry Pi3. The Operational system was based on Yocto Linux. Therefore, the platform on which the middlewares ran had low computational power.

The metrics used in the experiments were *average response time*, *CPU Usage* and *RAM usage of the Middlewares processes*. These metrics were chosen to evaluate how the middlewares perform regarding to time response and resources consumption.

The Jmeter software was used to perform the tests, whose objective was to stress the middlewares, sending data to them from devices. Since the number of real devices were limited (there were only six real outlets in the smart outlet system, and two devices in the biomedical devices system), simulated devices were also used. The Jmeter was used to simulate up to 400 devices. The tests consisted of each simulated and real device sending data to the middleware and measuring the time elapsed until the confirmation that the data was received by the middleware. The total response time for one round of test is the sum of the elapsed times of the devices. We carried out ten rounds of tests and computed the average response time.

The first scenario is the **smart outlet system**. The communication protocols used in this case were the Wi-SUN HAN for the real outlets and the HTTP through the REST APIs for the simulated devices. In the latter case, the number of simulated devices were 10, 50, 100, 200, 350, 400. The real devices sent data to the middleware every 5 seconds, while the simulated ones were tested with two intervals: 0.5 and 5 seconds. Each package has a total size of 392 bytes of data. Table III show the results for this scenario.

The second case is the **biomedical device system**. The communication protocols used in this case were the RS-232 for the real devices and the HTTP for the simulated ones. In this case the number of simulated devices were 10, 25, 40. The real and simulated devices sent a package of data to the middleware every 100 milliseconds. The package size was 20 bytes (100 bytes if the headers for the HTTP protocol is taken into account). Table IV shows the results for this scenario.

TABLE III
Results for the smart outlets case

Case	Interval of 5s				
Devices (Sim + Real)	Error (%)	Avg. Time (seconds)	Sys. Usage (%)	Midd Usage (%)	RAM Usage (%)
10 + 5	0	5.94	63.37	47.19	23.93
50 + 5	0	27.44	47.92	35.57	27.63
100 + 5	0	50.92	56.54	44.49	25.24
200 + 5	0	93.97	67.88	49.75	26.30
350 + 5	0	188.19	82.38	67.28	26.15
400 + 5	4.38	194.51	80.50	64.95	25.33
Case	Interval of 0.5s				
Devices (Sim + Real)	Error (%)	Avg. Time (seconds)	Sys. Usage (%)	Midd. Usage (%)	RAM Usage (%)
10 + 5	0	4.71	54.31	42.55	28.55
50 + 5	0	26.84	46.02	32.97	27.82
100 + 5	0	38.73	65.03	48.20	28.22
200 + 5	0	147.96	60.46	49.97	28.94
350 + 5	0	165.65	82.87	63.21	26.51
400 + 5	2.56	150.26	82.74	67.13	27.53

TABLE IV
Results for the biomedical devices system

Devices (Sim + Real)	Error (%)	Avg. Time (seconds)	Sys. Usage (%)	Midd. Usage (%)	RAM Usage (%)
10 + 2	0	6.14	75.68	38.19	33.13
25 + 2	0	15.56	73.07	43.38	31.92
40 + 2	0	28.21	76.32	46.20	34.58

The results in Table III show that the middleware can handle up to 350 devices without errors in the Smart Outlet scenario. Although the average times presented for this scenario are high, especially for more than 50+5 devices, these results show the proposed middleware's ability to handle a large number of devices without errors. A more powerful hardware system must be used if a shorter response time is required. On the other hand, Table IV (biomedical devices scenarios) shows that the middleware can handle up to 40 devices without errors and with an average time response below 30 seconds, which is maximum response time to consider that a device is presenting data in real-time [18].

The results also showed that, for both systems, (*i*) the system CPU usage did not exceed 85%, (*ii*) the Middleware RAM Usage was under 35%, and (*iii*) the Middleware usage was under 68%. Therefore, the proposed middleware makes

good use of the available resources, even when handling a considerable amount of devices.

Concerning the ease of development using the base case Template, which is the Template in its pure form, the only additions made to the Middlewares were: (1) implementation of the Wi-SUN FAN protocol for communication with the outlets, for the smart outlet system; (2) increase of the RS-232 protocol in the communications management - devices module, for the biomedical devices system. These modifications to the Middleware are what allow it to communicate with different devices.

The changes made to the system are only possible because of the developed modular architecture and because when the first communication between middleware and devices occurs, the latter must send packets containing the description of how data and commands are transmitted to and from these devices.

Therefore, according to the results, the developed middlewares work with acceptable performance in a mixed communication protocol environment with multiple devices. It also is suitable for different usage scenarios, as the test cases were different from each other. The usage in distinct cases is a consequence of the support and easiness of development provided by the Template. This facilitates the employment of new devices through the addition of new communications protocols as sub-modules of the Template modules. Thus, the Template and architecture can be used as a flexible starting point tool that might need only minor changes to run in other scenarios.

V. Conclusions

This work presented the issues for edge middleware development. Also it has shown that the popular available platforms have high computational requirements. Thus, developers need to create their own platforms. This showed that a tool for Edge middleware development could help to accelerate and facilitate this process.

In this study, a middleware architecture and Template were developed to address part of the aforementioned issues. The architecture is composed by seven elements and is modularized, which makes the architecture flexible, as it can be changed according to the usage if needed. The Template is a set of diagrams to follow and how to code the the solution.

The developed middlewares based on the architecture and Template have low computational hardware requirements. The results showed that both are suitable to develop low requirements middlewares. The results also showed that the middlewares performance are acceptable in both cases, which demonstrates that the architecture and the Template could be used in different scenarios without difficulties regarding the implementation.

Therefore, according to the results presented, the proposed work could be used as a tool to ease edge middleware development and it also could help the beginning of the development of another architecture.

Acknowledgments

We are thankful to Companhia Paranaense de Energia (COPEL) for support and financial assistance in this research project, Project Copel ANEEL PD-02866-0508/2019.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001

References

[1] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5187–5192.

[2] O. Bello, S. Zeadally, and M. Badra, "Network layer inter-operation of device-to-device communication technologies in internet of things (iot)," *Ad Hoc Networks*, vol. 57, pp. 52–62, 2017.

[3] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of middleware for internet of things: A study," *International Journal of Computer Science and Engineering Survey*, vol. 2, no. 3, pp. 94–105, 2011.

[4] A. Polianytsia, O. Starkova, and K. Herasymenko, "Survey of hardware iot platforms," in *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*. IEEE, 2016, pp. 152–153.

[5] 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 Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018.

[6] 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, 2017.

[7] I. Ungurean, N. C. Gaitan, and V. G. Gaitan, "A middleware based architecture for the industrial internet of things," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 10, no. 7, pp. 2874–2891, 2016.

[8] M. A. da Cruz, J. J. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, "Performance evaluation of iot middleware," *Journal of Network and Computer Applications*, vol. 109, pp. 53–65, 2018.

[9] A. S. Gaur, J. Budakoti, and C.-H. Lung, "Design and performance evaluation of containerized microservices on edge gateway in mobile iot," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 138–145.

[10] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Mosden: An internet of things middleware for resource constrained mobile devices," in *2014 47Th hawaii international conference on system sciences*. IEEE, 2014, pp. 1053–1062.

[11] M. A. da Cruz, J. J. P. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, "A reference model for internet of things middleware," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 871–883, 2018.

[12] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of things journal*, vol. 3, no. 1, pp. 70–95, 2015.

[13] M. B. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "Om2m: Extensible etsi-compliant {M2M} service platform with self-configuration capability," *Procedia Computer Science*, vol. 32, no. 0, pp. 1079 – 1086, 2014, the 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914007364

[14] Dojot, *Installation Guide - dojot v0.7.0 documentation*, 2021. [Online]. Available: https://dojotdocs.readthedocs.io/en/latest/installation-guide.html#hardware-requirements

[15] Kaa, *System installation - Kaa*, 2021. [Online]. Available: https://kaaproject.github.io/kaa/docs/v0.10.0/Administration-guide/System-installation/

[16] Zabbix, *Zabbix Manual - 2 Requirements*, 2021. [Online]. Available: https://www.zabbix.com/documentation/current/en/manual/installation/-requirements

[17] P. F. Pires, F. C. Delicato, T. V. Batista, B. C. C. Costa, T. A. Barros, and E. R. S. Cavalcante, *GT-EcoDiF–Ecossistema Web de Dispositivos Físicos, Manual Técnico*, 2013. [Online]. Available: https://silo.tips/download/gt-ecodif-ecossistema-web-de-dispositivos-fisicos-manual-tecnico

[18] ISO 80601-2-61:2017, "Medical electrical equipment – Part 2-61: Particular requirements for basic safety and essential performance of pulse oximeter equipment," International Organization for Standardization, Geneva, CH, Standard, 2017.