Sustainable Computing: Informatics and Systems 41 (2024) 100951

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Towards rapid modeling and prototyping of indoor and outdoor monitoring applications

Alessandra Rizzardi, Sabrina Sicari *, Alberto Coen-Porisini

Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via O. Rossi 9 21100 Varese, Italy

ARTICLE INFO

Keywords:
Internet of Things
Prototyping
Monitoring application

ABSTRACT

Nowadays, the capability to remotely monitor indoor and outdoor environments would allow to reduce energy consumption and improve the overall management and users' experience of network application systems. The most known solutions adopting remote control are related to domotics (e.g., smart homes and industry 4.0 applications). An important stimulus for the development of such smart approaches is the growth of the Internet of Things (IoT) technologies and the increasing investment in the development of *green* houses, buildings, and, in general, heterogeneous environments. While the benefits for the humans and the environment are evident, a pervasive adoption and distribution of remote monitoring solutions are hindered by the following issue: modeling, designing, prototyping, and further developing the remote applications and underlying architecture require a certain amount of time. Moreover, such systems must be often customized on the basis of the need of the specific domain and involved entities. For such reasons, in this paper, we provide the experience made in addressing some relevant indoor and outdoor case studies through IoT-targeted tools, technologies and protocols, highlighting the advantages and disadvantages of the considered solutions as well as insights that can be useful for future practitioners.

1. Introduction

Applications for indoor and outdoor remote monitoring are nowadays adopted in different domains, ranging from smart homes to smart offices, agriculture and greenhouses, and for many scopes. Among them, saving energy consumption and improving the overall management, and users' experience of remote monitoring systems, are the most fundamental requirements. In fact, the capability to remotely control, at any time, facilities and plants would allow handling the involved resources better (e.g., energy, land, water and materials), also minimizing possible local mismanagement and wastage.

Moreover, in order to reduce the negative influences of buildings on the environment, *green* buildings, which are also known as *sustainable* buildings, began to spread, aiming at creating a better indoor environmental quality for occupants with less natural resources consumption [1]. There are many different definitions of *green* building. However, it is generally accepted as the planning, design, construction, and operations of buildings with the maximum conservation of resources (e.g., energy, land, water and materials), environmental protection, pollution reduction and providing people with healthy and comfortable indoor space. The same considerations can be done about smart agriculture and greenhouses applications, where the use of resources must be optimized to promote the growth of plants without wastage.

Different technologies concur with the realization of remote monitoring applications, which are strictly related to the Internet of Things (IoT) paradigm. They potentially include, in the same scenario, more than one of the following technologies and communication protocols: Wireless Sensor Networks (WSN), Wireless Multimedia Sensor Networks (WMSN), Near Field Communication (NFC), Radio-Frequency IDentification (RFID), cameras, actuators, which represent the technologies; Message Queue Telemetry Transport (MQTT), ZigBee, Constrained Application Protocol (CoAP), 6LowPAN (IPv6 over Low-Power Wireless Personal Area Networks), and so on [2], which represent the communication protocols. The basic idea behind the IoT paradigm is the possibility of acquiring heterogeneous kinds of information, both scalar and multimedia data [3], from the environment where IoT devices are placed in. Such devices embed both sensing and actuating capabilities, making them "smart" and enabling them to interact with the surrounding environment. Such features allow the IoT system to share throughout the network and the Internet a huge amount of information, which can be used to provide customized services to the interested users in a remote manner [4].

To achieve such a goal, the numerous technologies and communication protocols mentioned above should, in many cases, cooperate, in order to enable the realization of efficient IoT infrastructures and

* Corresponding author.
E-mail address: sabrina.sicari@uninsubria.it (S. Sicari).

https://doi.org/10.1016/j.suscom.2023.100951
Received 16 July 2023; Received in revised form 25 November 2023; Accepted 16 December 2023
Available online 20 December 2023

to regulate the information exchange process. Hence, issues related to interoperability, scalability, security and privacy [5] naturally emerge, thus compromising the rapid development of efficient solutions. In fact, the time required for designing and prototyping could generally prevent a pervasive diffusion of indoor and outdoor remote monitoring applications. As a consequence, tools (like a simulator or a testing-platform) for supporting the realization of such IoT infrastructures from the design towards the development phase are needed, with the following main goal: representing all the components acting within the envisioned environment, so as to give an overview of the whole system before real deployment.

In this direction, this paper proposes the use of different supporting tools targeted to the IoT, providing representative cases studies of indoor and outdoor monitoring applications, ranging from the application logic to the physical devices. More in detail, existing technologies and protocols will be used to develop four case studies, which will be discussed taking into account advantages and disadvantages in the design and performance evaluation of such solutions. The potentiality of the proposed examples resides in the integration with different technologies and protocols, which range from software to hardware straightforwardly. Instead, simulators usually run within an environment far from real deployments; for example, the direct connection with real devices or with software modules on a gateway are neither supported nor integrated. The scope of this work is to demonstrate how proper tools can be closer to reality just from the design phase. Moreover, it is worth noting that many approaches and architectures are often evaluated in a single application domain employing a single technology or protocol [6]. At the same time, it is important to dispose of tools that facilitate the simulation of different scenarios, enabling the interoperation among different technologies/protocols, also accepting data formats and information coming from existing data-sets, possibly running a real-time environment. Only testing a partial representation of the whole architecture is a limitation, since it does not allow to thoroughly understand the implications and possible side effects of a designed infrastructure before real deployment. In contrast, the possibility of early (i.e., in a testing environment) analyzing and running the entire system could save time before the final development and avoid mistakes. To reach such a goal, this paper envisions the adoption of Node-RED[1] tool and OpenHAB[2] platform. They are both used for the implementation of the presented case studies, since they reveal to be valid candidates for modeling IoT networks. As it will emerge from the analysis, Node-RED is more intuitive and easier to set up with respect to OpenHAB. Hence, the adoption of Node-RED facilitates the prototyping of IoT systems by providing an integrated set of technologies and by abstracting low-level details. Instead, the integration of OpenHAB with sensors and front-end applications is more complex. A scope of this work is to provide to the scientific community an overview and a comparison of these available technologies in the prototyping of heterogeneous IoT applications, revealing advantages and disadvantages, as well as some performance indices (i.e., latency, packet delivery ratio, energy consumption, resources' utilization). Moreover, this work provides a discussion about emerging network infrastructures, based on fog computing, edge computing and mist computing concepts, in contrast with the traditional cloud systems. Finally, security and privacy issues will be analyzed.

The remainder of this paper is organized as follows. Section 2 investigates the actual state of the art about the tools and methods used by the researchers for validating remote monitoring systems, thus revealing our motivations. Section 3 presents the technologies and tools adopted for investigating the case studies, which are detailed in Section 4. Section 5 points out a discussion about the technologies, communication protocols, and databases adopted for the case studies, the related performance, in terms of latency, packet delivery ratio, CPU load, and energy consumption, possible threats and security solutions. Finally, Sections 6 and 7 end the paper, providing a discussion about the conducted analysis and drawing some hints for future research.

2. Related works and motivations

The growth and diffusion of remote monitoring systems were favored by the availability of sensor devices, able to acquire, in real-time, information from the surrounding environment and transmit them throughout the network towards a sink point, which is usually in charge of collecting and processing all the gathered data from a specific application [7]. This is the basic behavior of a WSN, which represents one of the enabling IoT technologies, together with actuators, NFC, RFID, cooperating by means of heterogeneous and lightweight protocols, such as MQTT, ZigBee, CoAP, 6LowPAN.

In this section, we aim to provide an overview of the most relevant indoor and outdoor scenarios, which have been analyzed in the literature. The selection process includes those papers which effectively present a design and a deployment phase towards the realization and, possibly, the evaluation of the investigated environment. In general, the works available in literature try to validate and test the performance of the proposed remote monitoring approaches by means of different programming languages and tools and typically refer to particular case studies, which aim to provide a concrete application context (even if it is only simulated). Note that researchers often use datasets to operate with real data (even if not in real-time) during the performance evaluation. A well-investigated field in remote controlling is e-health [8,9], ranging from the monitoring of chronic diseases to vital signs current status monitoring, and, finally, to the triage prioritization of patients. The survey, presented in [10], points out the evolutions of healthcare IoT systems, even if it does not focus on adopted technologies, in terms of tools, protocols, sensors, or platforms for realizing the discussed architectures and solutions. In this paper, telehealth solutions will not be covered, since the literature already offers a broad spectrum of examples. For such a reason, this work will focus on approaches targeted to indoor and outdoor environments, with particular attention to implications regarding the sustainability of the solutions themselves. Such a feature can be granted thanks to the technological means coupled with a clever internal logic, which aims to control the running system's overall energy and resources consumption. An adequate allocation of resources (e.g., energy, land, water and materials) surely prevents wastage and allows you to keep under control the costs [1].

What emerges is the need for a tool (or a set of inter-operating tools), which is able to represent the whole remote monitoring architecture closer as much as possible to the future working system, in order to provide to designers and developers a complete view of the final architecture and underlying logic, before its real deployment. As emerging in [11], novel validation methods and experimental tools are needed to study smart object networks in vivo, new software platforms are required in order to operate smart objects efficiently, and innovative networking paradigms and protocols are required in order to interconnect smart objects. Conducting experiment-driven research is fundamental to study software and protocol design for IoT use cases. Experiments can be used to verify (or disprove) perceived insights gained from theoretical models or simulations. Moreover, results from such experiments can serve to generate valid input parameters for the model or simulation-driven research. Besides implementing the examined approach itself, a software platform to operate the IoT devices, frameworks or middleware software, and tools to schedule, execute, control, and evaluate the experiments are necessary. Such a role has been played by WSN's simulators/emulators for many years [12], but, with the advent of IoT, new systems must be adopted, due to the heterogeneity of the involved devices and to the different services

[1] Node-red, Flow-based programming for the internet of things. https://nodered.org/

[2] Openhab, open-source automation software. https://www.openhab.org.

provided. Hence, the main goal of the work presented herein is to adopt tools and methodologies able to represent and validate indoor and outdoor IoT scenarios.

Table 1 summarizes the adopted technologies, the scope and the limitations of related works concerning smart building applications, which include smart homes and smart offices, smart manufacturing-related systems (e.g., those related to industry 4.0), and outdoor scenarios. Concerning this last point, smart agriculture surely represents the favorite example of application. A comprehensive overview of IoT devices and sensors, network and communication protocols adopted in the smart agriculture domain is provided in [13]; this paper surveys approaches (i.e., platforms, architectures, methods, models, frameworks, or applications) in the realization of smart agriculture solutions from 2006 to 2019, pointing out their purpose, the adopted technologies, and the open challenges. What emerges is that the existing solutions are not built over high-level tools, like those proposed in this paper, but, on the other side, they are mainly based on sensors and are targeted to certain communication protocols. The absence of modularity in the definition of such solutions does not guarantee high flexibility. Especially, smart agriculture and, in general, environmental monitoring scenarios, is moving from the traditional WSN-based architectures [14] to new innovative ones, which are based on the dynamic IoT [15,16]. For such a reason, the design and prototyping environments must meet the need for interoperability and flexibility required by the IoT.

As emerges from Table 1, some works tailored to indoor or outdoor case studies provide prototypical implementation (i.e., [17–25] or use real data sets to perform a preliminary evaluation of the proposed architecture and mechanisms (i.e., [26]). Most of them do not offer a quantitative investigation, but only the architecture/platform is presented (i.e., [27–34]), possibly with a running example (i.e., [35–40]). Often, the whole system is not tested, but only a part of it, and such an aspect represents the main lack in the previous studies and solutions. Such an issue arises because the adopted evaluation methods cannot provide the required instruments to embrace all the aspects and life cycles of the analyzed infrastructures. Note that it is due to multiple communication protocols, hardware and software technologies, which are difficult to put in action together in a single test bench. As a consequence, researchers often conduct their analysis only on the part of the envisioned architectures (usually the core of the application) and omit other interacting parties, such as end-users, data sources, brokers, trust authorities, ad so on. However, such parts' response times and behavior may significantly impact the application's core performance. Hence, it is really important to try simulating the behavior of all the components of the desired IoT system, in order to understand the performance better and have a complete view of the final solution. Another important reason to foster the availability of integration systems is related to the amount of different technologies and protocols acting into IoT environments. As pointed out in Table 1, the cited works make use of: (i) MQTT, ZigBee, Bluetooth, and Modbus as communication protocols; (ii) Node-RED, OpenHab and Ebbits as IoT platforms; (iii) MongoDB and InfluxDB for the data storage; iv) Raspberry Pi and Arduino as smart devices. The cooperation among such different elements is a complex-prone task, mainly in the prototyping phase of a new system. Finally, other limitations include the lack of a performance evaluation of the envisioned solutions, maintainability issues, complexity in the configuration and deployment, and incompatibility with the resource-constrained IoT scenarios. Such issues emerge from the column *Limitation* in Table 1. Hence, the differences among the available solutions and this proposed work can be summarized as follows:

- The case studies presented in this paper include both indoor and outdoor application domains.
- With the adopted tools (i.e., Node-RED and OpenHAB) the analyzed applications are totally prototyped and tested (i.e., not only a part of them is considered).
- Different technologies and communication protocols are integrated.
- Performance evaluation is carried out for to infer about the feasibility of the proposed solutions.

Trying to fill the emerged gaps, in Section 4, some relevant indoor and outdoor scenarios are investigated and modeled by means of emerging tools tailored to the IoT. Note that other researchers are recently interested in such a topic. For example, the authors of [41,42] present *RapIoT* along with *Tiles*, which represent a toolkit approach for rapid prototyping of IoT applications. The project is still evolving to allow end-users with no programming knowledge to adopt it. More in detail, RapIoT supports the development of collaborative applications by enabling the definition, implementation and manipulation of high-level data type primitives, where an input primitive is a discrete information sensed by an IoT device, while an output primitive is an action that can be performed by the IoT device. Such primitives act as loosely coupled interfaces among embedded devices and one or more application logics. The role of primitives is twofold. On one side they provide an event-driven approach to programming, on the other side they facilitate collaboration among developers working on different IoT layers by providing simple constructs to be used to describe the data exchanged among embedded devices and applications. Furthermore they allow non-experts to think in terms of high-level abstractions without dealing with hardware complexities. RapIoT is event-driven and it is built on top of Arduino platform and MQTT and CoAP protocols. Its main limitations are that its architecture does not comprehend any coded application logic embedded into IoT devices, and network latency issues. RapIoT supports the *Tiles* toolkit, which provides further functionalities to ease the utilization of RapIoT by non-expert users. Note that Node-RED ideally requires the same expertise from users to be used. Instead of primitives, Node-RED makes use of black boxes to connect the network's elements, functions, and real devices. Moreover, it is already adopted by companies to pursue design goals for the development of their systems, and the research community provides more support consisting in documentation and online forums.

3. Technologies and tools

Before detailing the case studies of interest, the involved technologies and tools are introduced herein.

3.1. Node-RED

Node-RED is an open-source project developed by IBM over the Node.js framework. It is a flow-based programming, web-based and event-driven tool, which can be deployed on a physical smart device (e.g., Raspberry Pi[3] or Arduino platform[4]). Node-RED makes a large set of nodes and flows available, mainly developed by the open-source community. Some nodes can act as cloud points. Hence, the behavior of the application to be designed/developed can be represented as a network of black boxes, which communicate with each other and regulate the flow of information within the envisioned system. The visual representation is particularly user-friendly.

In three of the case studies presented in Section 4, Node-RED will represent the core of the application logic. Besides Node-RED, several other web-based platforms have emerged to ease the development of interactive or near real-time IoT applications by providing a way to connect things and services together and process the data they emit using a data flow paradigm [44]. Among these, WoTKit Processor[5] is

[3] Node-RED and Raspberry Pi, https://nodered.org/docs/getting-started/raspberrypi.

[4] Node-RED and Arduino, https://nodered.org/docs/faq/interacting-with-arduino.

[5] WoTKit Processor, https://wotkit.readthedocs.io/.

Table 1
Previous studies and the research gap on indoor and outdoor scenarios.

Work/year	Technologies	Scope	Limitation
This work 2023	Raspberry Pi, Node-RED, OpenHab, MongoDB, InfluxDB, MQTT, Ignition	Design and prototyping of indoor and outdoor scenarios (**smart agriculture, planting system, smart office, domotics**)	No large data-sets from real world scenarios adopted in the performance evaluation
[35] 2022	XML formalism	Development of a case of study about a **domotic** IoT system application	No performance evaluation provided
[27] 2021	Raspberry Pi, OpenHab, relays modules	Definition of an **home automation** platform	No performance evaluation provided
[17] 2021	Arduino, Node-RED, InfluxDB, MQTT	Real-time monitoring and controlling of internal parameters for **smart buildings**	No performance evaluation provided, Node-RED only used for connecting entities and not as application logic
[37]–[36] 2020–2021	Smart home appliances, sensors, smart plugs, gateways, machine learning	Controlling and monitoring of **home automation** systems to detect behavioral patterns	Lack of real and large scale testing experiments
[38]–[39] 2020	Smart home appliances, sensors, smart plugs, gateways, ZigBee, machine learning	Energy consumption monitoring and controlling in a **smart home**	Implementation depends on the chosen devices and mobile app
[18] 2020	Raspberry Pi, mobile app, Microsoft Azure Cloud	Definition of a secure **home automation** system	Lack of a middleware layer as an intermediate in large-scale IoT environments, custom software implementation
[40] 2020	OpenHab	Definition of a **smart home** platform supporting decentralized adaptive automation control	Only running example provided
[19] 2019	Raspberry Pi, Node.js[a], MQTT, *Apache* web server	Monitoring of plants into the pots within a **smart planting system**	Adoption of a pure Node.js framework, which presents maintainability issues
[20] 2018	Arduino, Node-RED, MQTT, ZigBee	Prototype of a **smart home** application	No performance evaluation provided, complex installation
[31] 2018	Node-RED, Modbus, MQTT	Definition of a wireless industrial communication system (**industry 4.0** domain)	The target is limited to Industry 4.0 applications
[26] 2018	Raspberry Pi, Node.js for the custom IoT middleware, MongoDB[b]	Monitoring of home appliances and security policies enforcement, use of data-sets into a **smart home**	Adoption of a pure Node.js framework, which presents maintainability issues
[21] 2018	TLSensing platform, Raspberry Pi, Node.js for the custom IoT middleware, IP camera	Environmental monitoring and intrusion detection in a **smart building**	Adoption of a pure Node.js framework for the IoT middleware, which presents maintainability issues
[32] 2018	Minimal hardware to keep the cost low	Smart industrial automation and remote control system (**industry 4.0** domain)	No performance evaluation provided
[22] 2017	Raspberry Pi, Bluetooth	Security policies enforcement in a **smart home**	Complex installation
[15] 2017	WSN and cameras	Monitoring temperature and humidity in a **smart agricultural field**	The network is not suitable for the IoT
[30] 2016	Node.js, MongoDB	Definition of a **smart building** network architecture	Adoption of a pure Node.js framework, which presents maintainability issues, no performance evaluation provided
[28] 2016	Custom middleware	Virtualization of supply chain, simulation and decision support based on on-line operational data (**industry 4.0** domain)	Mainly customized solution
[29] 2016	ZigBee	Smart home control system	Implementation with a custom C# program
[23] 2016	*Mica2* hardware, TinyOS	Monitoring and controlling the micro-climate factors of a greenhouse (**smart agriculture** domain)	Implementation of the application logic with a custom Java program
[24] 2015	Mobius[c]	Realization of an integrated semantic service platform (ISSP) to support ontological models in **smart offices** and IoT-based service domains in general	The required ontology is difficult to configure
[25] 2013	Arduino, ZigBee, cloud services	Measuring home conditions, monitoring home appliances, controlling home access (**smart home** domain)	Strict dependence to chosen technologies
[33] 2013	Ebbits IoT platform[d]	Gathering real-time data (i.e., energy consumption and water usage) along with the manufacturing processes (**industry 4.0** domain)	No performance evaluation provided
[34] 2012	ZigBee, *NS2* simulator	Smart home control system	The simulator is not suitable for the IoT
[14] 2011	WSN	Environmental monitoring in **smart agriculture**	The network is not suitable for the IoT
[43] 2010	WSN, Binary Web Service (BWS), TinyOS	Environmental monitoring and localization, regulated by roles and authorizations (**smart building** domain)	Low-level configuration of the network, limited to sensors

[a] Node.JS. http://nodejs.org/.
[b] MongoDB. http://www.mongodb.org/.
[c] Mobius oneM2M, "oneM2M-compatible IoT service platform". http://wiki.onem2~m.org/index.php?title=Open_Source.
[d] FP7 ebbits project site, http://www.ebbits-project.eu.

almost famous as Node-RED. We decided to use Node-RED tool in this work since, unlike WoTKit Processor, which is server-based and does not provide access to local sensors, it can be deployed on a smart device, due to the lightweight nature of Node.js and the simplicity of the execution engine. Such features allow Node-RED flows to execute with good performance on devices such as the Raspberry Pi.

3.2. MQTT

MQTT[6] is a publish–subscribe network broker-based messaging protocol. In the presence of constrained scenarios, usually including embedded devices on non-TCP/IP networks, such as Zigbee, it is better to adopt the MQTT-SN protocol,[7] which is the MQTT for Sensor Networks. MQTT is largely used in the IoT domains due to its robustness and power-saving communication. One-to-many message distribution based on $topics$ is performed; another feature consists in the decoupling of information of sources and consumers.

In all the four case studies presented in Section 4, MQTT protocol plays a central role in message passing, due to its efficiency for the investigated scenarios. The broker is implemented with Mosquitto,[8] but other solutions are also available, such as HiveMQ.[9] Mosquitto has been chosen with respect to other solutions because it is totally open-source and very easy to use.

3.3. InfluxDB and MongoDB

InfluxDB[10] and MongoDB[11] are databases belonging to the NoSQL family. InfluxDB has been specifically developed for managing time-series data, thus making it an ideal choice for periodically logging sensor information. MongoDB is cross-platform and document-oriented. It makes use of JSON-like documents stored in collections, the data structure can change over time, and the document model is mapped to objects in the application code. Both InfluxDB and MongoDB will be adopted in the case studies presented in Section 4, since their data structure perfectly fits the need of heterogeneity dictated by IoT environments.

3.4. OpenHAB

Open Home Automation Bus (OpenHAB) is an open-source project home application platform used to run smart homes. It natively supports many devices and allows the user to further extend its capabilities by installing modules and plugins. Moreover, OpenHAB allows to write custom logical rules, which can be triggered using deployed sensors and perform user-defined actions (e.g., turn on lights at a given time or when a motion sensor is activated). OpenHAB will only be adopted in the case study related to domotics, due to its intrinsic features, as described here.

3.5. Ignition

Ignition[12] is a commercial, server-based cross-platform software, supporting a modular structure so that its deployment can be tailored for every specific requirement. Ignition includes a *Human Machine Interface/Supervisory Control And Data Acquisition (HMI/SCADA)*, which can be built, in a customized way, depending on the intended purpose. Ignition makes use of different kinds of *tags*: (i) *OPC Tags* use the Open Process Connectivity (OPC) standard to communicate and read/write values directly to the Programmable Logic Controller (PLC); (ii) *Memory Tags* hold and store information; (iii) *Expression Tags* are driven by a user-defined expression, such as a mathematical operation, a logical operation, and so on; (iv) *Query Tags* pool their value from an SQL statement; (v) *Reference Tags* refer to other tags to fetch their value. Ignition, and the next Grafana tool, will be coupled with OpenHAB to create a simple yet real domotics system with the support of real devices, as presented in Section 4.

3.6. Grafana

Grafana[13] is an open-source web-based tool for data visualization and analysis. It allows to model custom dashboards, based on the required use cases and supports different data sources, like InfluxDB, Microsoft SQL Server, PostgresSQL, AWS CloudWatch, etc. Also, it allows the development and installation of custom modules/plugins which can expand its capabilities.

4. Case studies and prototyping

In this section, four case studies will be designed and their prototypical implementation will be detailed, concerning the technologies and tools described in Section 3. Such case studies concern: a smart agriculture scenario, a hydroponic planting system, a smart office, and a cross-domain domotics example.

4.1. Smart agriculture

In the considered smart agriculture scenario, the goal is to create a system capable of managing an outdoor cultivated field and a greenhouse. The greenhouse is divided into four different areas, so that the user can have the opportunity to choose a species of plant to be placed in each area; moreover, the same plant can be placed in different areas, but each area can only contain a maximum of one plant. Through the adoption of specific sensors, the system is able to autonomously take care of the different plants' needs, considering the information reported in a proper database. Note that the system also guarantees to maintain ideal conditions about temperature and humidity, also in case of plants' diseases, by means of proper pesticides. More specifically, the sensors on the cultivated field measure the following parameters: soil temperature and soil humidity. Instead, the following parameters are monitored inside the greenhouse: air temperature, air humidity, soil temperature in each area, soil humidity in each area, leaf wetness (in the range [1:15], such a parameter indicates the presence of humidity and, therefore, of diseases on the surface of the leaves), and light.

A control application, written in Java programming language, simulates and keeps track of the behavior of the just mentioned sensors. Then, values from the sensors are requested and transmitted using HTTP methods (i.e., GET, $POST$, PUT, $DELETE$) for RESTful services over the TCP/IP stack. The logic of the application is structured in rules, which are managed and implemented inside the Node-RED tool:

- If, in the cultivated field, the soil temperature exceeds 25 °C or the humidity is lower than 30%, the irrigation system is activated.
- If, inside the greenhouse, the air temperature exceeds 27 °C or the air humidity is higher than 65% the fan is activated, where the fan has three different speeds.
- If the leaf wetness sensor detects a data greater than 9, pesticides are activated in the greenhouse' area.

[6] MQTT OASIS standard, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt.

[7] MQTT-SN OASIS standard, https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf.

[8] Mosquitto, open-source MQTT v3.1/v3.1.1 broker. http://mosquitto.org.

[9] HiveMQ MQTT broker. https://www.hivemq.com.

[10] InfluxDB, time-series platform. https://www.influxdata.com/.

[11] MongoDB. http://www.mongodb.org/.

[12] Ignition software. https://inductiveautomation.com.

[13] Grafana, interactive visualization tool. https://grafana.com.

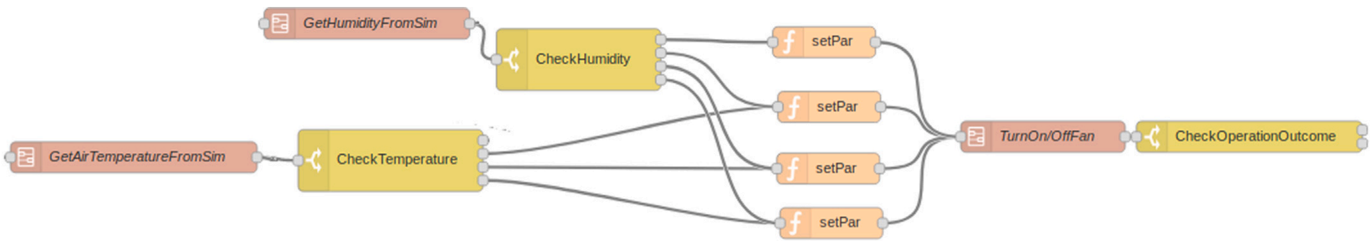


Fig. 1. Smart agriculture - greenhouse's flow, temperature.

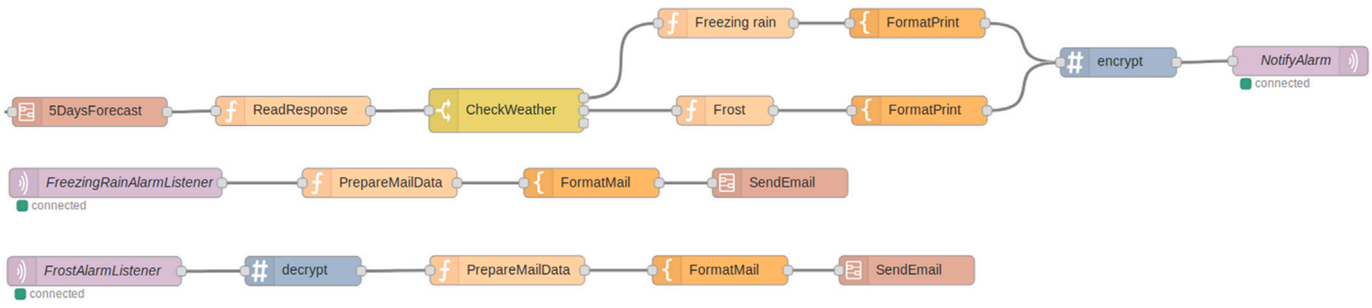


Fig. 2. Smart agriculture - vegetable garden's flow.

- If temperature or humidity do not fall within the thresholds set for the chosen plant (such values are gathered from the system's database), irrigation is activated in the concerned greenhouse's area.
- If the light level inside the greenhouse is low, the lights are switched on to illuminate the plants.
- If rain is expected in the next few hours, the irrigation system is not activated; note that such information is gathered from a specific component, named *OpenWeatherMap*, made available by Node-RED.
- Proper alarms are sent in case of very low temperatures or freezing rain, to an email configurable by the user. An email will also be sent to the user at the beginning of the harvesting time for each plant inserted in the greenhouse.

Note that, for the transmission of alert messages, the system uses the MQTT protocol to notify all the brokers. The hierarchy of MQTT topics is includes: *greenhouse/alarm/frost* and
greenhouse/alarm/freezingrain. In this way, a broker listening on the *greenhouse/alarm* topic is able to receive both the alarms.

The flows implementing the rules mentioned above are defined in Node-RED, as follows:

- A flow for the management of the greenhouse: Fig. 1 represents the temperature management. Note that temperature is the more complex metric considered by the rules. In fact, the flow shows that two parameters (i.e., temperature and humidity) are taken into account to decide if activate or not the fan inside the greenhouse.
- A flow for the management of the vegetable garden: Fig. 2 contains the flows to manage the irrigation system and is based on the information obtained by both the weather forecast and the data collected by the sensors, as shown in Fig. 1. Note that, the system takes into consideration the forecast for the next 5 days. An alarm is triggered in case of frost or freezing rain. The alarm system is managed in an encrypted way via MQTT notifications. Such notifications generate an email to be sent to the administrators of the greenhouse, in order to take some actions to preserve vegetables against harmful atmospheric agents.
- A flow for the management of the web pages: it implements the services offered through *POST* and *GET* requests to change the settings regarding the whole system (e.g., location, email, greenhouse's areas configuration).
- A flow for the management of the dashboard: it consists of four tabs that contain the flows for creating the four sections included in the dashboard which are related to weather, monitoring, plants, and logs, as shown in Fig. 3 for the part related to the monitoring dashboard. Such a dashboard can be used by an administrator to monitor the data collected by the sensors, display the weather forecast for the current day and the next five days, access the system logs (i.e., the events that occurred in the system), and insert new plants species in the system. Moreover, the state of each greenhouse's area can be monitored and controlled from another dashboard shown in Fig. 4.

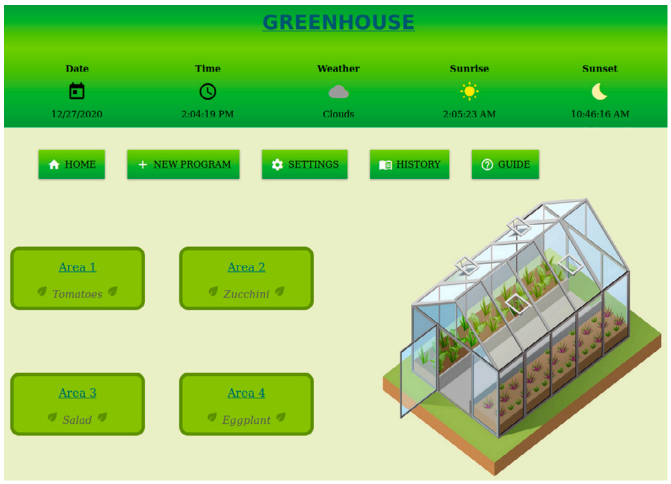
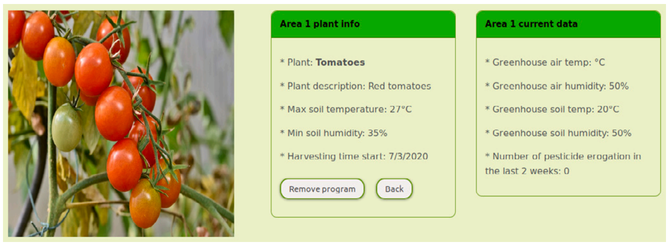


Fig. 3. Smart agriculture - dashboard.

MongoDB is adopted for the management of the data collected and generated by the system; among the collected data, we have: (i) the temperatures and humidity detected by the sensors, along with place (i.e., greenhouse or cultivated field), area, date, and time; (ii) all the information, such as name, description, harvesting period, humidity,



Area 1 plant info

* Plant: **Tomatoes**

* Plant description: Red tomatoes

* Max soil temperature: 27°C

* Min soil humidity: 35%

* Harvesting time start: 7/3/2020

Area 1 current data

* Greenhouse air temp: °C

* Greenhouse air humidity: 50%

* Greenhouse soil temp: 20°C

* Greenhouse soil humidity: 50%

* Number of pesticide erogation in the last 2 weeks: 0

Fig. 4. Smart agriculture - dashboard related to a greenhouse's area.

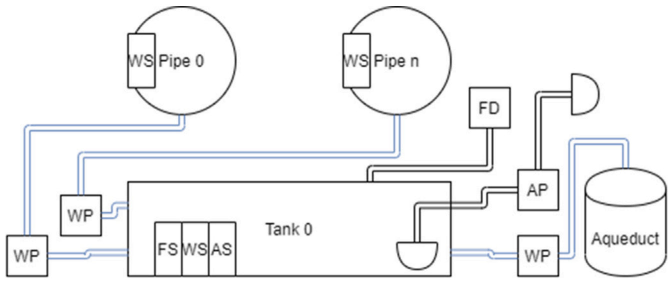


Fig. 5. Hydroponic planting system - plantation structure.

temperature and image's URL, of the available plants. The stored information could be useful to generate statistics over the time, in order to plan future strategies to better manage the vegetables inside the greenhouse or into the cultivated field.

4.2. Hydroponic planting system

Hydroponics is a method of growing a plant without soil, where the root of the plant is directly exposed to a water fertilized solution [45]. Root takes all the needed nutrients directly from water, so the importance of keeping water suitable for the crops' growing is fundamental. Such a system aims to innovate intensive farming, by reducing water consumption and allowing, with a lamp, to stack the crop vertically. Hydroponics' harvest is allocated at the top of a pipe with a hole, which allows the root to reach the water. Water continuously flows inside the pipe and drops into another bigger tank that works as water storage; some pumps have to push water through the pipe in order to let the roots take what they need.

The environment of a hydroponic plantation is summarized in Fig. 5 and can be divided into three subsets, which are divided by scope, as follows:

- *Plantation subsystem*, composed by a tank, works as a reserve of fertilized water that is sent to several pipes; on the top, this pipe has a series of holes that allow the roots of the plant to reach the water and absorb all needed nutrients. The water reaches the pipe by means of a pump (WP) that connects the tank to the pipe; each pipe has its own pump, and, at the end of the pipe, there is another smaller pipe that allows the water to return into the tank slowly. A WP is also inside the water tank since its goal is to keep the water reserve always above a certain threshold for all the connected pipes. An air pump (AP) keeps the water with the proper oxygen saturation and a fertilizer dispenser (FD) drops the nutrients the plant needs into the water itself.
- *Sensor subsystem*, composed by a series of sensors and actuators attached to the plantation subsystem and communicating with the control subsystem using MQTT protocol; those sensors can be divided into two sets: (i) the former enables the measurement of the water (WS) in each pipe, the water (WS) in the tank, the oxygen saturation (AS) and the level of nutrients contained in the water (FS) inside the tank; the latter enables the remote control of the pumps (WP, AP) and the dispenser (FD). Essentially, the *sensor subsystem* includes both telemetric sensors (which are divided into water, oxygen and fertilizer sensors) and actuators coupled to WP, AP, and FD, respectively, in order to switch on/off the pumps or drop the fertilizer into the water.
- *Control subsystem*, composed of technologies and applications that enable the control and the interaction with the plantation entities. More in detail: (i) an MQTT broker manages the message from and to the remote sensors; (ii) a MongoDB database stores the measure of remote sensors and the current configuration of the plantation; (iii) a web page allows the visualization of data gathered by sensors and the management of configuration settings. Hence, the status of the system is available both in real time and into the storage, for future analysis.

The MQTT protocol plays a fundamental role in such a case study, since the whole system's behavior is regulated by MQTT message passing. In fact, MQTT is used by sensors to publish their values and by the control subsystem for managing the actuators keeping the system always in the right status. The adoption of MQTT for message exchanges lightens the whole system's communication infrastructure and enables a very easy management of information thanks to the use of precise topics. The topic structure is as follows: */target/target's id/scope/element/field*, where:

- *target* identifies if the value is referred to a tank or a pipe component.
- *target's id* is a progressive number to identify pipes and tanks.
- *scope*, which identifies three kinds of information: (i) *status*, if the node indicates the value published by sensors; (ii) *config*, if the node indicates the configuration value of the sensors; (iii) *machine*, if the node identifies the actuator; hence, publishing under this node permits the control of all pumps and dispensers.
- *element* which represents the target/source of a message; according to its scope, this node can be used for different purposes, as described above, while the node name can be: (i) *water*, if in "status" publishes the water level, or in "machine" is used to control the pump; (ii) *air*, if in "status" publishes the oxygen saturation, or in "machine" is used to control the pump; (iii) *fertilizer*, if in "status" publishes the fertilizer level, or in "machine" is used to control the dispenser.
- *field*, which is only available under *config* scope, since it contains different values used to change the configuration of the planting system. In particular, it allows to set values related to: (i) the water level of pipes or tanks; (ii) the oxygen saturation of tanks; (iii) the fertilizer.

The *control subsystem*, implemented in Node-RED, is subscribed to all topics so that, according to the settings previously defined, it can activate the devices, shown in Fig. 5, through the connected actuators. The main flows related to the described scenarios refer to water pump's and tank's management. Fig. 6 represents the water pump's management (task's management is very similar). Here, we can see the publish and subscribe operations happening throughout the whole application, as motivated above. Moreover, a fundamental Node-RED block, named *report-by-exception (RBE)*, guarantee that the node blocks unless the incoming value changes. Hence, the system avoids to send multiple times the same information, thus saving bandwidth and reducing network delays.

Such a case study includes both initialization and normal running phases. The initialization depends on the types of crops included in the hydroponic planting system and can be made throughout a proper web interface, which is depicted in Fig. 7. Then, once the plantation ends the initialization, users can monitor its status using the web page, shown in Fig. 8 concerning the tank's status (similar one are available for the pumps). If some values are not compliant, the user can investigate about the motivations and fix the issue, for example by changing the setting values of single devices or the fertilizer thresholds.

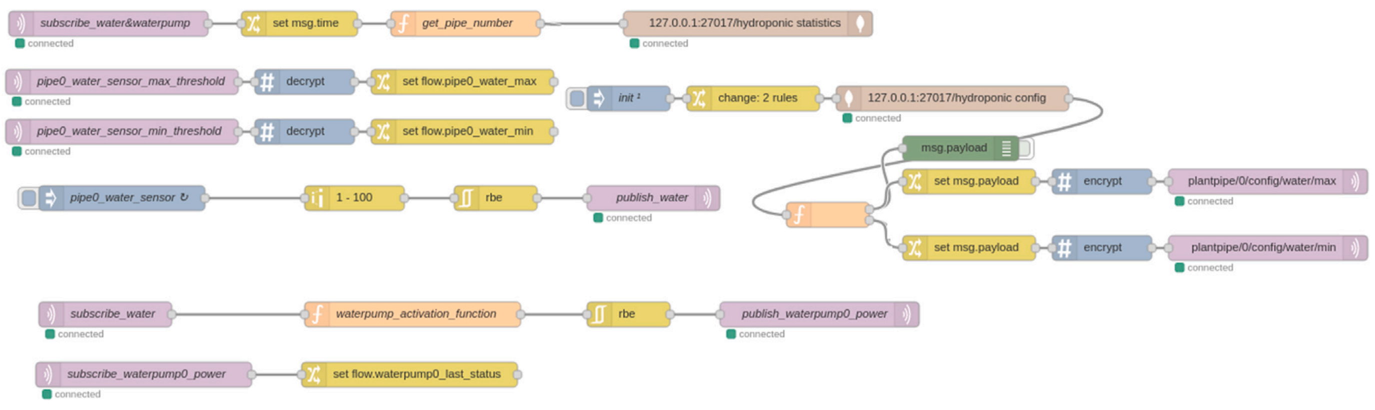
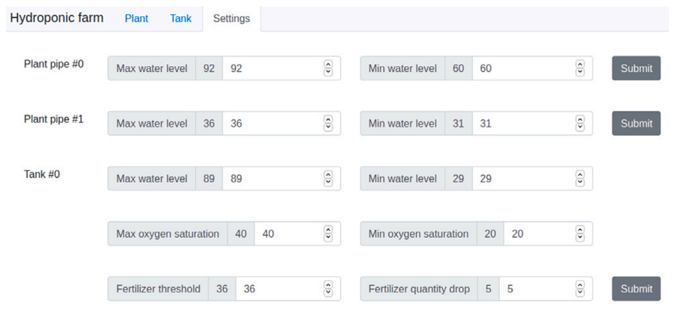


Fig. 6. Hydroponic planting system - water pump's management flow.


Hydroponic farm Plant Tank Settings

Plant pipe #0	Max water level 92 / 92	Min water level 60 / 60	Submit
Plant pipe #1	Max water level 36 / 36	Min water level 31 / 31	Submit
Tank #0	Max water level 89 / 89	Min water level 29 / 29	
	Max oxygen saturation 40 / 40	Min oxygen saturation 20 / 20	
	Fertilizer threshold 36 / 36	Fertilizer quantity drop 5 / 5	Submit

Fig. 7. Hydroponic planting system - settings' web interface.

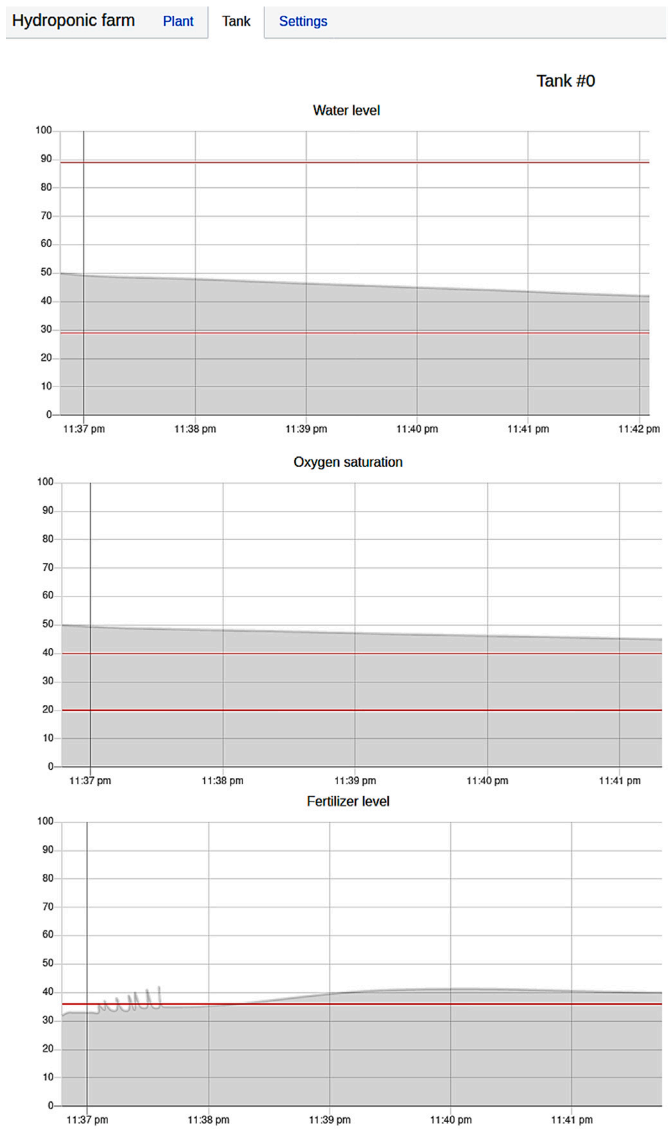


Fig. 8. Hydroponic planting system - tanks' monitoring web interface.

4.3. Smart office

A *smart* office should be able to automatically and remotely manage components, such as windows, doors, lights, heaters, air conditioning, and intrusion alarms, in response to the current state of the environment itself (e.g., time of the day, presence of people, weather). The envisioned system should give the users the capability to set up their office based on real situations, for example configuring the windows' number. Hence, proper sensors will be in charge of measuring the presence of people, the temperature, and the air quality; while other ones will act as actuators to change the status of the door/windows (i.e., open/close), of the lights/heaters/air conditioning (i.e., switch on/switch off), and of the intrusion alarm (i.e., activated/disabled). Fig. 9 shows the interface related to the Java simulator, which represents all the components and current settings of the smart office.

Based on user configurations, the system behaves in one way rather than another. For example, if the user inserts zero as windows' number, and if the air quality exceeds a certain threshold, then the system will open the door; while, if there is at least one window (and the weather it is different from "rain"), the system will open the window/windows. If the user selects the time slot "6 p.m. to 6 a.m.", and the presence sensor discovers someone in the office, then the system will turn on the intrusion alarm; otherwise, the simulator will turn on the light. In order to put in act such rules and, in general, the overall functionalities related to the smart office, the application logic is performed by five Node-RED flows:

- The *weather* flow (Fig. 10) is used to retrieve the current weather and save it along with the current external temperature on MongoDB for further generating a pie chart (Fig. 11) in the dashboard that represents the occurrences of some weather conditions during the monitored period. Such information is gathered from *OpenWeatherMap* block (as shown in the figure), which is made available by Node-RED and makes use of an online service to provide the required data. Moreover, checking the current weather and external temperature is also useful to properly set the status

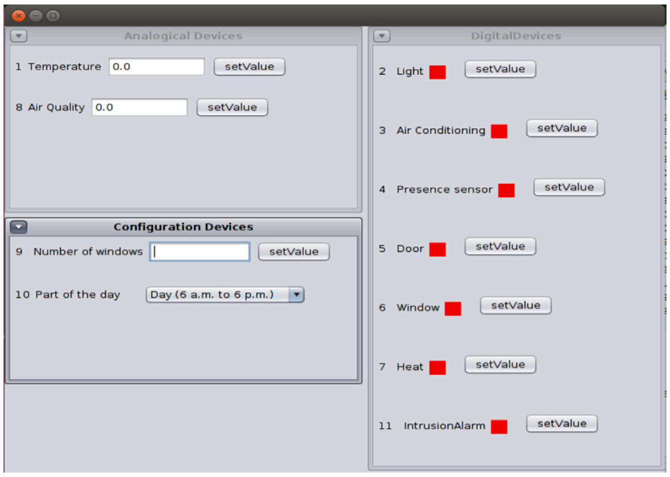

Analogical Devices
1 Temperature 0.0 setValue
8 Air Quality 0.0 setValue
Configuration Devices
9 Number of windows setValue
10 Part of the day Day (6 a.m. to 6 p.m.)
DigitalDevices
2 Light setValue
3 Air Conditioning setValue
4 Presence sensor setValue
5 Door setValue
6 Window setValue
7 Heat setValue
11 IntrusionAlarm setValue

Fig. 9. Smart office - control simulator.

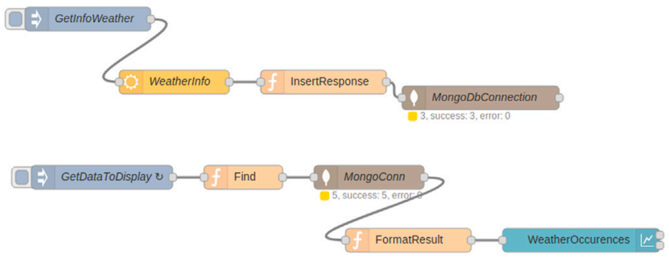


Fig. 10. Smart office - weather flow.

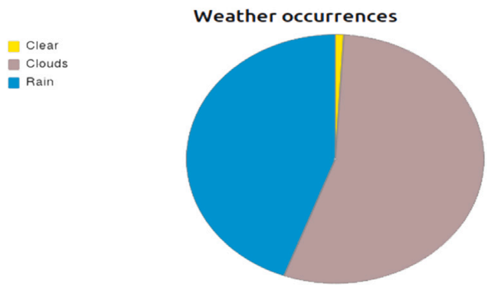


Fig. 11. Smart office - weather conditions.

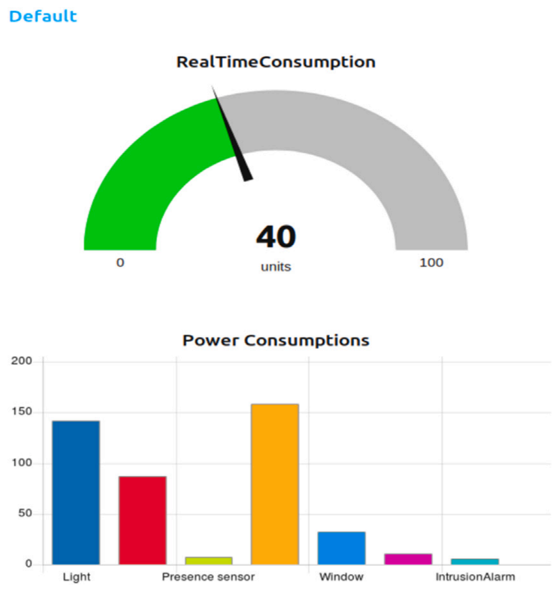


Fig. 12. Smart office - power consumption.

of heaters, air conditioning, windows and door (e.g., if it rains outside, the windows are not opened).

- The *light* flow (Fig. 13) is used to verify the presence or absence of someone inside the office and, depending on the time of day, determine whether to turn the light on or off or to trigger the intrusion alarm. Both alarms and logs are managed by means of MQTT communications and topics. This represents a very simple and light way to propagate control information about the system, since subscribers, which will receive the notifications, can actuate proper countermeasures and analysis about the condition of the smart office. Note that both alarms and logs are encrypted in order to prevent possible confidentiality or integrity attacks.
- The *temperature* flow controls the temperature and the air conditioning/heating status, considering different thresholds and the time of the day, whose status is notified via MQTT broker by means of dedicated topics (i.e., *smartoffice/alarm/temperature* and *smartoffice/log/temperature*).
- The *air quality* flow (Fig. 14) controls the air quality value so that:
  – If *air quality* <= 100 the system notifies through MQTT broker on the topic *smartoffice/log/airquality* an encrypted message saying that the air quality is suitable and the system closes both door and windows.
  – If *air quality* > 100 the system notifies via MQTT broker on the topic *smartoffice/alarm/airquality* an encrypted message saying that the air is not clean; then, it checks the current weather and, if it is raining, it opens the door and closes the windows; otherwise it opens both door and windows.
- The *dashboard* flow provides the functionalities to realize and run the dashboards related to: (i) the current notified alarms; (ii) the web form to remotely turn on or off lights, air conditioning, heating and set the status of the intrusion alarm; (iii) the power consumption, which is empirically calculated on the basis of hourly energy consumption of each sensor/actuator, as shown in Fig. 12.

It is worth remarking that the historical data (i.e., related to the energy consumption, logs, and so on), stored in proper MongoDB collections, can also be used to further analyze the energy consumption of the smart office and, in general, of a whole smart building, during the time to improve the management and save energy, thus reducing costs.

4.4. Domotics

The previously described case studies, i.e., smart agriculture, hydroponic plating system, smart office, are mainly focused on the application logic, which is implemented in the Node-RED tool; while the behavior of real sensors is delegated to simulators (e.g., Java programs or MQTT external devices). Certainly, this is the best approach to follow if designers/developers are primarily interested in understanding and prototyping the behavior of a complex system; while the hardware will be considered later. Instead, in the case study about domotics, presented herein, the focus is just on real devices. A RaspberryPi and some sensors are directly connected to software applications and tools, able to change their status. Adding such a scenario, related to the domotics domain, gives a complete overview of prototyping methods.

The idea is to build a system using the Ignition software SCADA solution to control its real devices. The list of devices includes: (i) a

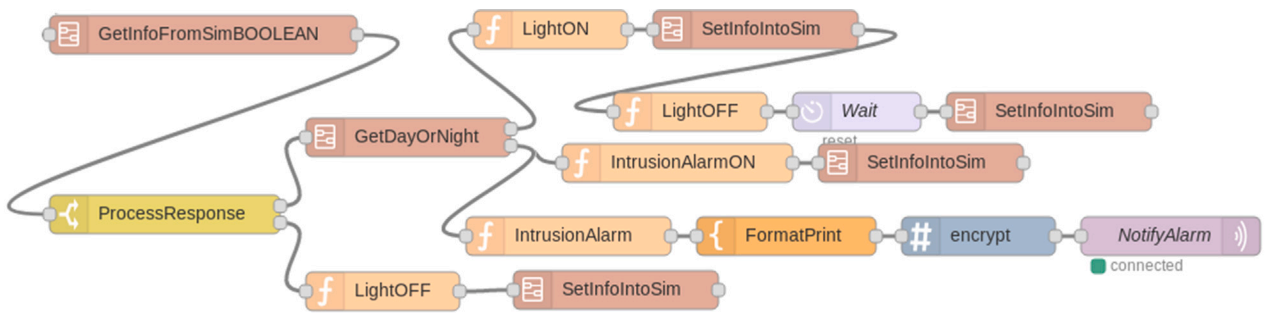


Fig. 13. Smart office - light flow.

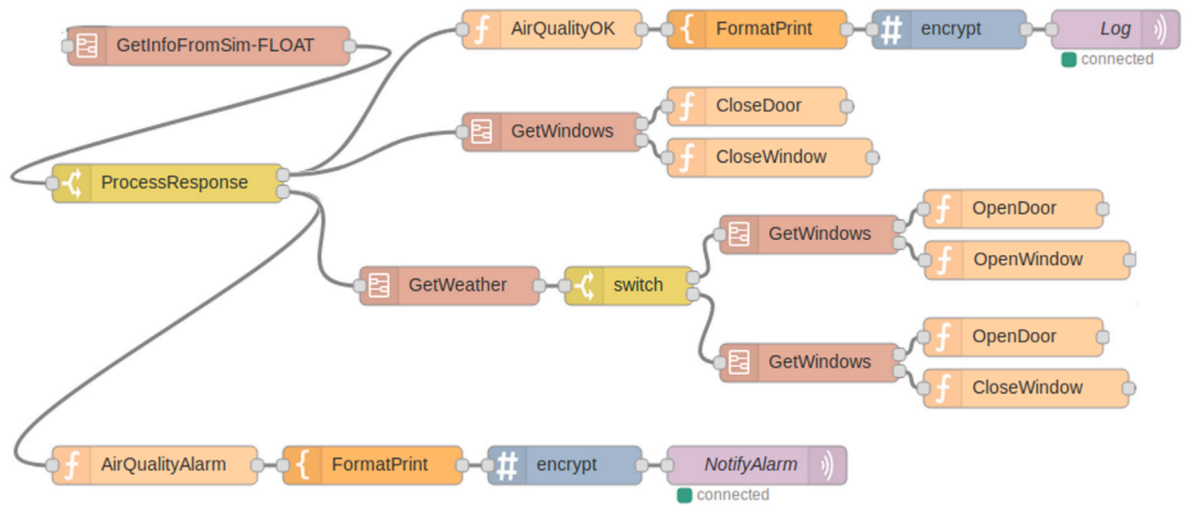


Fig. 14. Smart office - air quality flow.

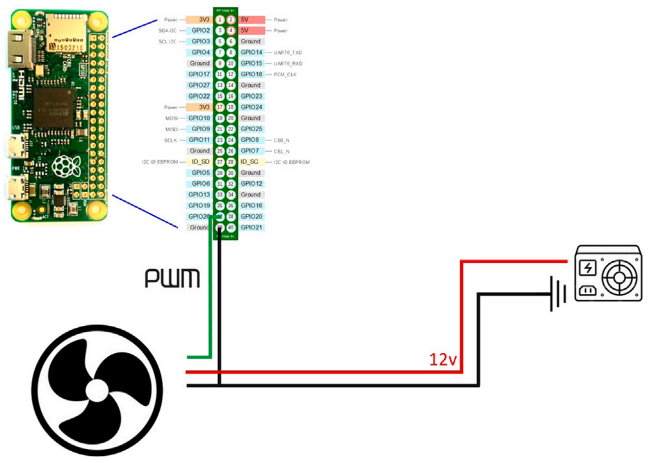


Fig. 15. Domotics - fan's connection schema.

RaspberryPi Zero W[14] with an attached computer fan; (ii) three Xiaomi room temperature and humidity sensors; (iii) two Xiaomi smart plugs. Also, different host systems are involved: (i) Virtualbox to host the SCADA system; (ii) home NAS to host the MQTT broker (deployed using $Mosquitto$); (iii) Cloud VPS to host historical database and data visualization UI.

The RaspberryPi Zero W behaves like an IoT-enabled PLC. PLC devices are equipped with sensors (to gather information) and actuators (to perform actions). In this case, CPU temperature information feeds the sensor's data. Instead, concerning the actuator, an external fan is used and controlled through the GPIO pins using Pulse Width Modulation (PWM). PWM is a method used to control devices that require power or electricity. It essentially makes use of a digital signal, which is periodically turned on or off to modulate the connected device. In particular, PWM controls the fans motor; the larger the time frame between pulses is, the slower the motor turns. The envisioned scheme is shown in Fig. 15. The RaspberryPi communication is managed through the usage of MQTT, where:

- Sensor's information is published to the topic *rpi/cpu/temperature*.
- Actuator's information is fetched by subscribing to the topic *rpi/fan/speed*.

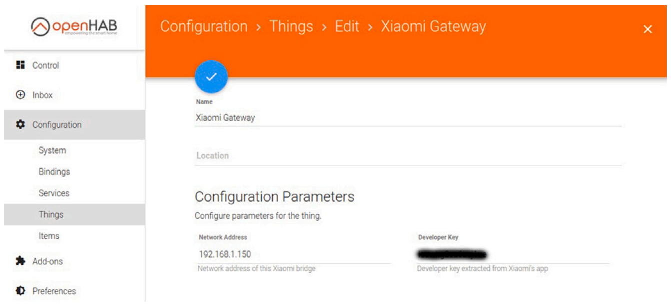
All the required logic is written using Python and executed at boot time by means of a *crontab task*.

OpenHAB, presented in Section 3 and hosted on the home NAS, is used to bridge the connection among the proprietary Xiaomi smart home sensors. The adopted modules are: (i) Xiaomi Smart Home Binding,[15] which is used to communicate with the sensor gateway; (ii) MQTT Binding,[16] which is used to connect to an MQTT broker. Configuration for the Xiaomi binding is done through the OpenHAB web GUI, by providing the gateway IP and private API key, as shown in Fig. 16. By using the web GUI, it is possible to automatically search and add

[14] https://www.raspberrypi.org/products/raspberry-pi-zero-w/.

[15] https://www.openhab.org/addons/bindings/mihome/.

[16] https://www.openhab.org/addons/bindings/mqtt/.



Name
Xiaomi Gateway

Location

Configuration Parameters
Configure parameters for the thing.

Network Address	Developer Key
192.168.1.150	[redacted]
Network address of this Xiaomi bridge	Developer key extracted from Xiaomi's app

Fig. 16. Domotics - OpenHAB Xiaomi binding configuration.

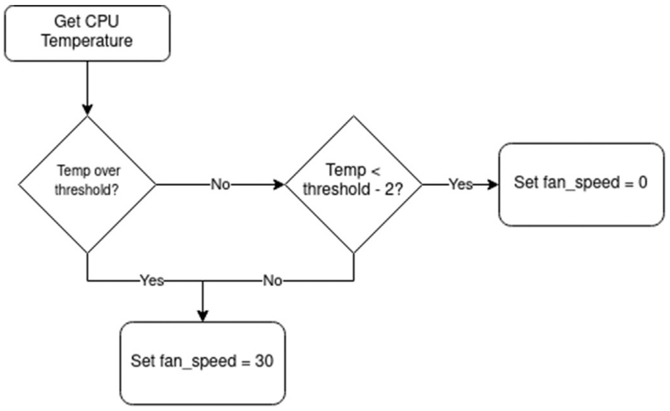


Fig. 17. Domotics - fan's profile flowchart.

all detected/connected sensors creating for each dedicated *item*, which can be referenced for *rule*'s definition. The MQTT binding is configured by adding a proper *things* file in the OpenHAB configuration folder, thus allowing it to be referenced for MQTT communication. The next step is defining the logical rules to follow for publishing sensor data over MQTT. These are defined by adding *rules*' file into the OpenHAB configuration folder. As defined before, the rule triggers every detected state change on the observed sensor by publishing its new value to the related topic.

The system's core is the Ignition SCADA, the software in charge of reading and managing all attached devices, connecting via MQTT protocol. Ignition leans on a Microsoft SQL Server database, which is used as a local buffer to periodically store and change the status of sensors' information in case the connection towards the historical server is lost, but also to manage data migration towards the historical database, which is hosted on an external VPS server. The database structure is straightforward, since it includes only logs about room sensors and light status. Instead, the topics' structure includes information about: the Raspberry Pi CPU temperature (i.e., *rpi/cpu/temperature*); the fan speed (i.e., *rpi/fan/rpm*); the room's temperature and humidity sensors (i.e., *home/room_n/temperature* and *home/room_n/humidity*); the light state (i.e., *home/light_n/state*). A memory tag is added to hold the current desired fan speed, in the range [0:100]. This tag implements a python script, which triggers a value update, checks if the new value is valid and publishes it to the dedicated topic *rpi/fan/speed*. To automatically manage the fan, two support memory tags are needed: (i) a Boolean, to toggle on or off the automatic fan profile; (ii) a float, to set the desired target CPU temperature. Also, a script is applied to the CPU temperature tag, which triggers each new read value, comparing it to the desired temperature and deciding if it is necessary to turn the fan on or off. To avoid continuously switching the fan state, due to the temperature fluctuations around the threshold value, the fan is turned off once a temperature 2 °C lower than the target one is reached. A scheme of the just described behavior is sketched in Fig. 17.

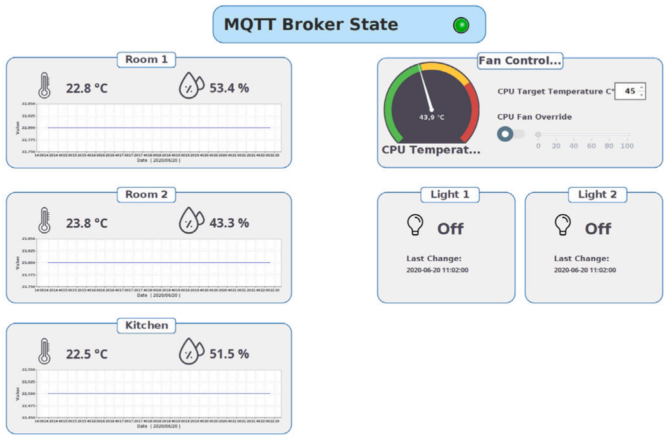


Fig. 18. Domotics - Vision HMI. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Furthermore, a web GUI visualizes and controls the connected devices; it is developed using Vision, which is a UI building tool, provided by Ignition, and it is able to build a local HMI, as shown in Fig. 18. Vision is composed of four parts:

- MQTT broker status, where the right LED indicates if the broker is currently available (green) or not (red).
- Room temperature and humidity, which indicate, for each room, the last detected information; while also displaying a chart with the trend of the last hours. The chart toggles between temperature and humidity by pressing the appropriate current value.
- Light status, which shows the current light state and the last timestamp it toggled.
- RaspberryPi fan control, which displays a gauge with the current CPU temperature. This is divided into three color-coded areas: (i) $green$ as desired temperature; (ii) $yellow$ means "above target"; (iii) red as over temperature. The user can interact with the spinner being able to change the desired temperature. Also, it is possible to override the automatic fan controller with the appropriate switch and, then, use the slider to set the fan speed manually.

As just anticipated, the historical database is hosted on an external VPS server, and it is paired with a data visualization tool, named Grafana (see Section 3), that allows you to visualize the gathered (and stored) data in various user's defined charts. The database used for this task is InfluxDB (see Section 3) for two main reasons: (i) it is specifically designed to manage time-based information; (ii) it is officially supported by the Grafana suite. For security reasons, two separate accounts have been defined on the historical database: (i) read/write, for the migration task; (ii) read-only, for the Grafana connection. As stated in Section 3, InfluxDB defines two types of data: tags and fields. For this specific use case, the $roomID$ and $lightID$ were added as tags, since they are mainly used in the $WHERE$ statement to filter data; while the other information is stored into fields. Two measurements are used: one for rooms and one for lights.

An overview of the envisioned domotics system is provided in Fig. 19, which resembles all the components described for such a use case. A demo video is available at https://youtu.be/-5Gg5I0B3Ak.

5. Discussion

Early design and prototyping are fundamental to speed up the development process, and a framework, as Node-RED, represents a viable solution in this direction, due to the features that emerged in the discussion of the case studies in Section 4. Hence, to achieve such

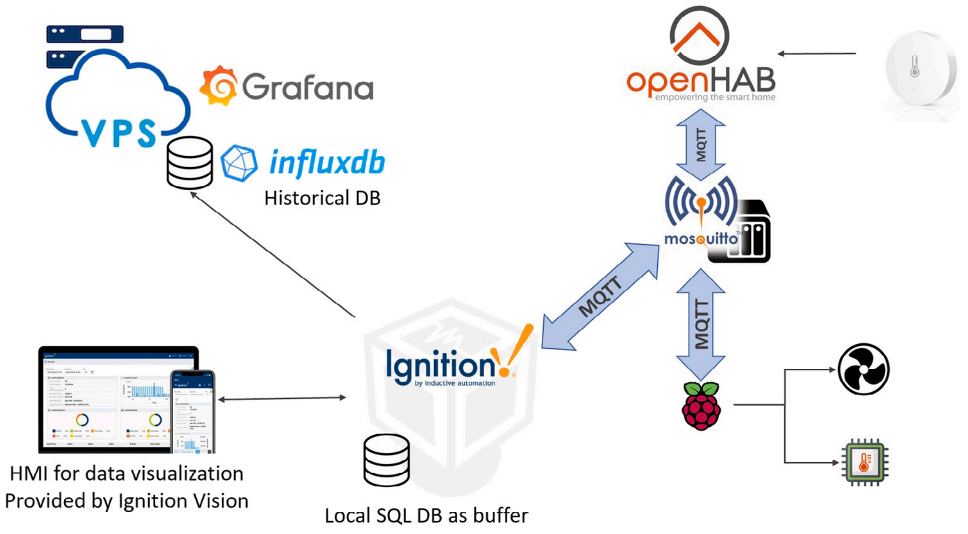


Fig. 19. Domotics - system architecture.

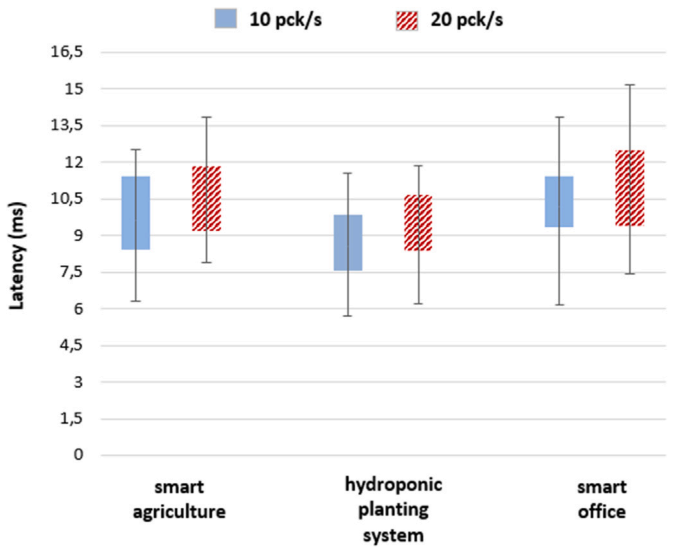


Fig. 20. Latency: whiskers-box diagram for smart agriculture, hydroponic planting systems, and smart office.

a goal, proper tools must be adopted. With this respect, promising solutions are pointed out in this paper, ranging from high-level modeling, by means of the Node-RED tool, to a low one, adopting Ignition SCADA software and OpenHAB. Nevertheless, all the presented case studies could have been implemented with one or the other, since they are both valid candidates for modeling IoT applications and infrastructures. Surely, Node-RED is more intuitive and easier to set up with respect to Ignition SCADA software and the related tools (i.e., OpenHAB, Grafana). Such a comparison reveals to be useful to conclude that the adoption of Node-RED facilitates the prototyping of IoT systems by providing an integrated set of technologies and by abstracting low-level details and communication protocols. Instead, the integration among Ignition SCADA, OpenHAB, Grafana, and Xiaomi sensors has been more complex. InfluxDB is natively designed to manage time-based information; while MongoDB is more document-oriented. Table 2 sums up the technologies, communication protocols, and database adopted in the investigated case studies.

Looking at the involved elements shown in Table 2, we can conclude that the use of such technologies requires programming knowledge and experience in IoT, network protocols and databases by the end-users. Regarding such an aspect, we asked a group of 52 master's students, who attended a lab at our university, to code and prototype simple IoT scenarios through Node-RED. After this experience, we conducted an analysis about the usability perceived by the students about the adopted tool. They noted that the mechanism based on flows is straightforward to adopt; in this way, the information flow is clear. Then, inside the flows, users are free to add the desired code to regulate the system's behavior. A further important feature emerged about Node-RED is that you can easily find online documentation with examples, in case you need assistance to solve an issue. Table 3 shows the percentages obtained from the consultation with the students concerning the just discussed aspects. Certainly, being able to manage more complex functionalities requires more in-depth study of the tool, which, in this paper, has been adopted for the first three case studies (i.e., smart agriculture, hydroponic planting systems, smart office). A virtual machine containing the code and the structure of such scenarios is available at the Bitbucket repository available here: https://bitbucket.org/alessandrarizzardi/.

Except in the case of the domotics scenario, the others, in fact, have been preliminarily assessed by installing the corresponding Node-RED application on a Raspberry Pi 3 B equipped with 1 GB of RAM. An ASUS laptop, equipped with 32 GB of RAM, 11th Generation Intel(R) Core(TM) i7 processor and Windows 11 Pro as operating system, is used to emulate the behavior of the sensors (i.e., by means of the Java programs, mentioned in Sections 4.1–4.2–4.3). Such simulated devices basically generate random (but close to reality) data, as presented in the corresponding Sections 4.1–4.2–4.3, and send them to the Node-RED application as if they came from different nodes. Note that also real data sets could be integrated and used as sources for testing the analyzed scenarios, as well as real-time data incoming from connected physical devices (as the domotics case study) or obtained from web services. The laptop and Raspberry Pi communicate via a WiFi network. The simulation duration is set to 24 h and the packet rate is set to 10 and 20 packets per second. Such rates have been chosen to stress the network load and to have an idea of how the system could scale, hence one rate doubles the other. Figs. 20 and 21 show, respectively: (i) the end-to-end latency of data from their generation to their availability on the visualization dashboard; (ii) the packet delivery ratio (PDR), which measures the percentage of sent packets correctly received by Node-RED. Note that such metrics are indices of the reliability of the network. In fact, we can note that, in all scenarios, latency remains in an optimal range, which is between 6 ms and 15 ms on average. Also, PDR is above 94% on average, thus ensuring the correct reception

Table 2
Technologies, communication protocols and database adopted for the case studies.

Case study	Involved technologies	Communication protocols	Database
Smart agriculture	Node-RED, simulated sensors	HTTP, MQTT	MongoDB
Hydroponic planting system	Node-RED, simulated sensors	MQTT	MongoDB
Smart office	Node-RED, simulated sensors	HTTP, MQTT	MongoDB
Domotics	Xiaomi room temperature/humidity sensors Ignition SCADA, RaspberryPi Zero W20, Xiaomi smart plugs, OpenHAB, Grafana	MQTT	InfluxDB

Table 3
Outcomes of the students' consultation.

Question	Do not agree	Partially agree	Totally agree
Node-RED is easy to use?	0%	14%	86%
Is the information flow clear?	0%	0%	100%
Code customization is possible?	0%	14%	86%
Is online documentation available?	0%	0%	100%

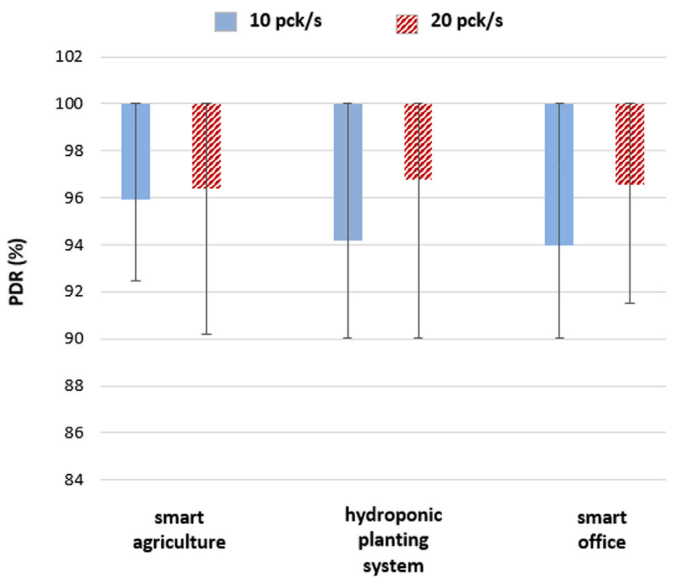


Fig. 21. PDR: whiskers-box diagram for smart agriculture, hydroponic planting systems, and smart office.

of packets inside the application. Instead, Fig. 22 represents the mean CPU load on the Raspberry Pi, which does not exceed the 30% in all case studies. More specifically, in the scenarios with a rate of 10 packets per second, the CPU load remains in the range 19%–25,5% on average, while in the scenarios with a rate of 20 packets per second, the CPU load is, on average, in the range 22,5%–28,5%. Concerning energy consumption (Fig. 23), we installed a dedicated module, named *Power Monitor*,[17] which regularly feeds the node that simulates the incoming data from sources with a real number as payload representing the average consumption in Watts since the last message. Hence, we calculate the consumption each time Node-RED receives a packet. We approximately consider the transmission consumption equal to 1,4 Watt and initial energy of 100 J [46]. The obtained results demonstrate the potentiality of the envisaged solutions, which should be further evaluated, in the next future, in large-scale scenarios in order to infer their scalability. Note that Raspberry Pi has been selected to run the IoT platform, leveraging a lightweight approach for performing the network tasks. In a wider environment, a distributed approach must be pursued, where the IoT platform is split into more 'smart' devices, in order to better manage the resources, for example following the fog computing principles [47].

[17] Node-RED Power Monitor, https://flows.nodered.org/node/node-red-contrib-power-monitor.

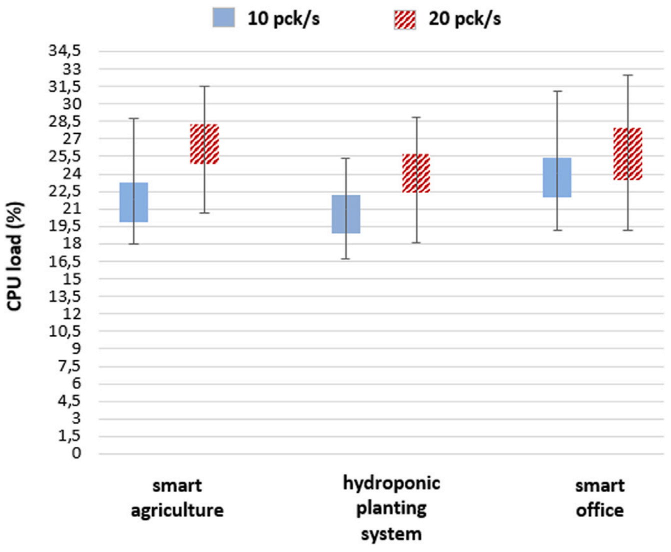


Fig. 22. CPU load: whiskers-box diagram for smart agriculture, hydroponic planting systems, and smart office.

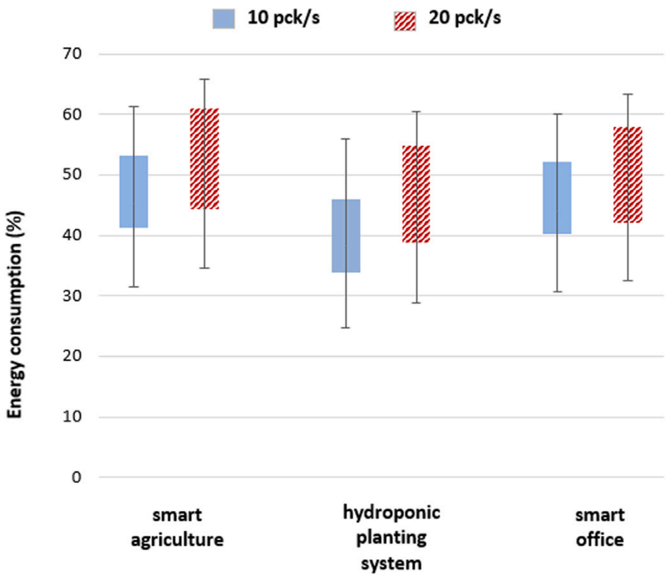


Fig. 23. Energy consumption: whiskers-box diagram for smart agriculture, hydroponic planting systems, and smart office.

Concerning such a point, it is interesting to discuss about the kind of network architecture to be adopted in smart scenarios. More traditional ones are those based on cloud-IoT systems, where information from the IoT devices is directly transmitted to the cloud data center; such a solution inevitably creates a bottleneck in environments, which need to process a huge amount of data, possibly also in real-time. To cope with such an issue, other approaches were raised, as follows [48]:

- *Fog computing:* essentially, it is the extension of the cloud to the edge of the network; more in detail, fog computing allows decentralized computing by processing data at the fog nodes (i.e., gateways, routers, and cloud services), instead of centralizing the processing task in the cloud data center. Such an approach can be of great utility to smart cities and, in general, where many devices use real-time data to perform various tasks. Hence, fog computing can be used for multiple business applications that require instant data processing [49]. However, business leaders must analyze the pros and cons of fog computing in their specific situations before adopting this network infrastructure. In particular, the pros concern reduced costs of storage and computing, low latency, and reduced bandwidth requirement compared to cloud computing; additionally, confidential data can be secured as it can be stored at the fog node. Moreover, fog computing can process larger volumes of data than edge computing, since it can process real-time requests. Cons include that fog computing depends on multiple links for transferring data from the physical asset chain to the digital layer, representing potential network failure points [50].
- *Edge computing:* it can be used for processing data directly on IoT devices, without relying on the cloud or fog, guaranteeing to process data in near real-time, and reducing the overhead at the centralized cloud. Edge computing can be used in connected buildings to perform simple tasks, like turning on the heater or lights near real-time [51]. Also, it can simplify predictive maintenance in organizations by sending instant alerts about possible failures. Pros include the possibility, for IoT devices, to determine which data needs to be stored locally and which data needs to be sent to the cloud for analysis; hence, sensitive data can be discretely stored at its source. Instead, the cons include that: (i) edge computing is less scalable compared to fog computing; (ii) edge computing supports little interoperability, which might make IoT devices incompatible with certain cloud services and operating systems; (iii) edge computing does not support resource pooling.
- *Mist computing:* it is adopted at the extreme edge of a network, which consists of micro-controllers and sensors. Hence, mist computing can harvest resources with the help of computation and communication capabilities available on the sensors, thus enabling local decision-making processes; as a consequence, microcontrollers and microcomputers are used to transfer data to fog computing nodes and, eventually, to the cloud. Mist computing can be effectively applied when devices only serve a singular purpose. Pros include: (i) conserving bandwidth and battery power, since only essential data is transferred to the gateway, server, or router; (ii) adopting access control mechanisms that can ensure data privacy at a local level. Cons include that microcomputers and sensors used in the infrastructure of mist computing can only be used for lightweight data processing and a narrow range of tasks, such as sending a location or performing a continuous interpretation of data collected from the environment [52]. Hence, such devices can be used for limited applications.

In general, the best architecture to be adopted depends on the application requirements, as emerged from the above discussion. To this end, some experiments could be conducted, as presented in [53], where three alternative deployment models of IoT applications (i.e., edge, IoT+cloud, fog) are compared in the realization of a bonsai greenhouse company. In this work, we could envision adopting mist computing for the hydroponic planting system, the smart office, and the domotics scenario; while an edge computing-based architecture could be the best choice for the smart agriculture environment.

Table 4
Analysis of case studies' features.

Case study	Sustainability	Heterogeneous communication technologies	Security&Privacy
Smart agriculture	yes	yes	partly
Hydroponic planting system	no	yes	partly
Smart office	yes	yes	partly
Domotics	no	yes	no

Concerning pivotal aspects, such as sustainability, adoption and integration of heterogeneous communication technologies, security and privacy, Table 4 points out if they are considered or not, or if they are only partially addressed in the investigated case studies. We note that smart agriculture and smart office scenarios embrace the sustainability requirement more naturally due to the need to save energy and reduce costs, given the environment's situation at various periods. Heterogeneous communication technologies cooperate in all scenarios, since data are gathered from sources and shared with final users, who obviously adopt different devices. Finally, security and privacy requirements are only partly considered, since information is transmitted in an encrypted way, in almost any scenario, but no access control policies or authorization mechanisms have been integrated. The following threats could affect the presented indoor and outdoor scenarios:

- **Integrity violation and eavesdropping**: the content of transmitted data could be accessed or modified by malicious entities; to cope with such an issue, encryption mechanisms along with end-to-end verification systems must be integrated into the system; moreover, key management schemes must be envisioned to proper handle encryption and decryption tasks.
- **Unauthorized access**: access control refers to the permissions in the usage of resources, assigned to different actors within the IoT network; in the presented scenarios access control policies should be defined in order to regulate the permissions to process and share the data transmitted within the network. Furthermore, mechanisms for the identification of involved entities should also be put into action, by means of credentials of certificates provided by certified and trusted authorities.
- **Privacy violation**: information transmitted in such scenarios could be sensitive (e.g., personal information related to people's movements, habits, and interactions with other people) and, as a consequence, should be protected and possibly anonymized against malicious parties. Note that sensitive data should be separated from identification ones, in order to strongly reduce the inference process.

Possible integration concerning security will be proposed in Section 7. To summarize, the following advantages, emerged from this work, can be pointed out:

- Application's logic and underlying simulated or emulated hardware are separated, by distinguishing data producers' functionalities from the IoT system's core.
- Interactions between the application's front-end and back-end can be effectively glued together in a prototype, enabling a faster evaluation of their correct/incorrect behavior.
- Different tools, software or web-based platform, database engines, and programming languages can be combined and tested in a single simulation environment. Also, real devices (e.g., Raspberry Pi, Arduino) can be integrated.
- Tools natively provide or can integrate options for generating well-structured dashboards, charts, gauges, and so on, which help in monitoring and visualizing the output of the analyzed scenario.

A last fundamental aspect should be considered: the maintenance and continuous improvement of the presented IoT systems, while ensuring their availability, is complex, but ever more essential, as also emerged from the limitations pointed out in Table 1. The issues arise from the need to adapt frequently and rapidly to new requests. More in detail, increasing competition and rapidly changing market needs require, from nowadays companies', flexibility and fast time to market of their products [54]. To cope with the emerges challenge, *DevOps* practice [55] could be adopted. It is an approach that facilitates the collaboration among the IT teams and accelerates/improves the cycles of maintenance of running systems, in order to detect and solve issues before end-users are impacted. *DevOps* culture, to be effective, must be coupled with: (i) the *agile* concept, which consists in removing process barriers empowering individuals, producing working software rapidly, collaborating closely with customers, and promptly responding to changes; (ii) *CI/CD (Continuous Integration/Continuous Delivery)*, which is a software engineering practice where members of a team integrate and deploy (after build and test) their work with increasing frequency. However, *CI/CD (Continuous Integration/Continuous Delivery)* approach is not an easy task and is even more complex in the IoT context, due to the fact that IoT systems are an integration of: (i) a hardware infrastructure, usually including a set of connected physical devices, such as servers, physical objects, sensors, actuators, etc., and communication networks; (ii) a software infrastructure (i.e., the IoT platform or framework) that supports the development, deployment, and execution of end-to-end IoT applications; (iii) a set of end-user applications running over the infrastructure. In order to efficiently adopt such an approach, the only solution is to improve the automation tools capabilities and verifiation processes. In such a direction, proper procedures should be added, for example, to the Node-RED execution flows, in order to periodically provide feedbacks about the system's behavior. Such a goal must be pursued before starting the prototyping phase. In this way, the applications under development can be constantly monitored and updates should happen in a safe mode.

6. Conclusions

The analysis carried out in this paper was motivated by the lack, in literature, of case studies and operative examples of indoor and outdoor systems, able to embrace both the logic application (intended as an IoT management middle layer), the involved protocols and devices, along with considerations on performance. In such a direction, the main contribution of the proposed work consists of designing and prototyping four different scenarios related to smart agriculture, hydroponic planting system, smart office and domotics. Both the approaches (i.e., Node-RED and Ignition SCADA with OpenHAB), employed for their realization, have provided a complete view of the IoT application case studies, allowing to analyze and simulate them, considering all of them the involved entities and messages' passing. Different technologies, protocols and languages, such as Java, JSON, Node.js, MQTT, HTTP, MongoDB, InfluxDB have been integrated without worrying about interoperability issues, thanks to the capabilities of the adopted tools. In that sense, such approaches represent viable solutions for performing preliminary tests on heterogeneous IoT scenarios. In fact, errors or malfunctions in the application's logic or in the devices' interactions can be promptly revealed and corrected before deploying. Hence, the support provided by the proposed tools could help avoid wasting time in deployment attempts.

7. Future research directions

Three critical aspects still deserve attention: scalability, security&privacy, power consumption evaluation. In fact, the case studies, presented in this paper, do not give a clear idea about the possible scalability of the envisioned environments. Such a kind of analysis could be carried out, as an open research activity, by defining re-usable modules and components to be integrated (and replicated) in a more complex system, following edge and mist computing approach, as pointed out in Section 5. Instead, security&privacy requirements can be achieved at various levels:

- MQTT protocol's level: proper credentials are needed to authenticate nodes/users to the broker; moreover, the whole MQTT communication exchange can be kept secure by means of the framework proposed in [56] .
- At database's level: proper credentials are needed to perform the authentication towards the database [57]; a set of roles and associated permissions can be defined on the basis of the specific case study.
- At devices' level: the devices directly manages encrypted messages, which are sent to the broker and, then, only decrypted by legitimate users; in this direction, many security protocols already exists targeted to IoT devices [58].
- At the IoT core platform's level: complex security and privacy policy enforcement mechanisms could be integrated within the core application itself [59].

In such a direction, a notable recent work [60] proposes a declarative framework, consisting of a *Prolog* prototype, named *Solomon*. It allows specifying policies for mediating contrasting (user and/or global) goals and actuator settings in smart environments. In this sense, the framework is totally cross-domain and it works as a service through *Logic Programming-as-a-Service (LPaaS)*. We will consider two aspects of such an approach in our work: (i) the customized management of policies to perform clever access control to resources/facilities; (ii) the possibility to deploy the platform either to cloud or edge servers, thus improving scalability. Furthermore, blockchain technology will be also considered to manage access control among the involved IoT entities [61].

Finally, concerning the assessment of power consumption, further experiments will be conducted by means of real sensors, connected to the Node-RED framework, instead of the current virtually simulated ones.

CRediT authorship contribution statement

Alessandra Rizzardi: Conceptualization, Funding acquisition, Methodology, Writing – original draft, Writing – review & editing. **Sabrina Sicari:** Funding acquisition, Supervision, Writing – review & editing. **Alberto Coen-Porisini:** Funding acquisition, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the SERENA-IIoT project, which has been funded by MUR (Ministero dell'Università e della Ricerca) under the PRIN 2022 program (project code 2022CN4EBH), and in part by project SERICS (project code PE00000014), under the NRRP MUR program funded by the EU - NGEU.

References

[1] Yang Geng, Wenjie Ji, Zhe Wang, Borong Lin, Yingxin Zhu, A review of operating performance in green buildings: Energy use, indoor environmental quality and occupant satisfaction.

[2] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, Mahmood Alzubaidi, Internet of things (IoT) communication protocols, in: 2017 8th International Conference on Information Technology, (ICIT), IEEE, 2017, pp. 685–690.

[3] Ali Nauman, Yazdan Ahmad Qadri, Muhammad Amjad, Yousaf Bin Zikria, Muhammad Khalil Afzal, Sung Won Kim, Multimedia internet of things: A comprehensive survey, IEEE Access 8 (2020) 8202–8250.

[4] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, Omer Rana, Fog computing for the internet of things: A survey, ACM Trans. Internet Technol. (TOIT) 19 (2) (2019) 1–41.

[5] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, Alberto Coen-Porisini, Security, privacy and trust in internet of things: The road ahead, Comput. Netw. 76 (2015) 146–164.

[6] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, Moussa Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376.

[7] S.R. Jino Ramson, D. Jackuline Moni, Applications of wireless sensor networks - A survey, in: 2017 International Conference on Innovations in Electrical, Electronics, Instrumentation and Media Technology, (ICEEIMT), IEEE, 2017, pp. 325–329.

[8] O.S. Albahri, A.S. Albahri, K.I. Mohammed, A.A. Zaidan, B.B. Zaidan, M. Hashim, Omar H. Salman, Systematic review of real-time remote health monitoring system in triage and priority-based sensor technology: Taxonomy, open challenges, motivation and recommendations, J. Med. Syst. 42 (5) (2018) 80.

[9] Rachael C. Walker, Allison Tong, Kirsten Howard, Suetonia C. Palmer, Patient expectations and experiences of remote monitoring for chronic diseases: systematic review and thematic synthesis of qualitative studies, Int. J. Med. Inform. 124 (2019) 78–85.

[10] Yazdan Ahmad Qadri, Ali Nauman, Yousaf Bin Zikria, Athanasios V. Vasilakos, Sung Won Kim, The future of healthcare internet of things: A survey of emerging technologies, IEEE Commun. Surv. Tutor. 22 (2) (2020) 1121–1167.

[11] Yousaf Bin Zikria, Heejung Yu, Muhammad Khalil Afzal, Mubashir Husain Rehmani, Oliver Hahm, Internet of things (iot): operating system, applications and protocols design, and validation techniques, 2018.

[12] Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, Sabrina Sicari, A comparative study of recent wireless sensor network simulators, ACM Trans. Sensor Netw. 12 (3) (2016) 20.

[13] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Tariq Umer, Yousaf Bin Zikria, Role of iot technology in agriculture: A systematic literature review, Electronics 9 (2) (2020) 319.

[14] Yingli Zhu, Jingjiang Song, Fuzhou Dong, Applications of wireless sensor network in the agriculture environment monitoring, Procedia Eng. 16 (2011) 608–614.

[15] S.R. Prathibha, Anupama Hongal, M.P. Jyothi, IoT based monitoring system in smart agriculture, in: 2017 International Conference on Recent Advances in Electronics and Communication Technology, (ICRAECT), IEEE, 2017, pp. 81–84.

[16] Partha Pratim Ray, Internet of things for smart agriculture: Technologies, practices and future direction, J. Ambient Intell. Smart Environ. 9 (4) (2017) 395–420.

[17] Elias Junior Biondo, José Lima, Thadeu Brito, Alberto Yoshihiro, Real-time monitoring and controlling of internal parameters for smart buildings, in: 1st Symposium of Applied Science for, 2021, p. 28.

[18] Rizwan Majeed, Nurul Azma Abdullah, Imran Ashraf, Yousaf Bin Zikria, Muhammad Faheem Mushtaq, Muhammad Umer, An intelligent, secure, and smart home automation system, Sci. Program. 2020 (2020).

[19] J. Hadabas, Miklos Hovari, Imre Vass, Attila Kertész, Iolt smart pot: an iot-cloud solution for monitoring plant growth in greenhouses, 2019.

[20] Olivier Debauche, Saïd Mahmoudi, Mohammed Amin Belarbi, Mohammed El Adoui, Sidi Ahmed Mahmoudi, Internet of things: Learning and practices. application to smart home, in: 2018 International Conference on Advanced Communication Technologies and Networking, (CommNet), IEEE, 2018, pp. 1–6.

[21] D. Costantino, G. Malagnini, F. Carrera, A. Rizzardi, P. Boccadoro, S. Sicari, L.A. Grieco, Solving interoperability within the smart building: A real test-bed, in: 2018 IEEE International Conference on Communications Workshops, (ICC Workshops), 2018, pp. 1–6.

[22] Antonio La Marra, Fabio Martinelli, Paolo Mori, Andrea Saracino, Implementing usage control in internet of things: A smart home use case, in: Trustcom/BigDataSE/ICESS, 2017 IEEE, IEEE, 2017, pp. 1056–1063.

[23] Zhenfeng Xu, Junjie Chen, Yahui Wang, Zhen Fan, A remote monitoring system for greenhouse based on the internet of things, in: MATEC Web of Conferences, Vol. 77, EDP Sciences, 2016, 04001.

[24] Minwoo Ryu, Jaeho Kim, Jaeseok Yun, Integrated semantics service platform for the internet of things: A case study of a smart office.

[25] Moataz Soliman, Tobi Abiodun, Tarek Hamouda, Jiehan Zhou, Chung-Horng Lung, Smart home: Integrating internet of things with web services and cloud computing, in: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, (CloudCom), IEEE, 2013, pp. 317–320.

[26] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, Alberto Coen-Porisini, Securing the smart home: A real case study, Internet Technol. Lett. 1 (3) (2018) e22.

[27] Rachida Ait Abdelouahid, Olivier Debauche, Abdelaziz Marzak, Internet of things: A new interoperable iot platform, application to a smart building, Procedia Comput. Sci. 191 (2021) 511–517.

[28] Cor N. Verdouw, J. Wolfert, A.J.M. Beulens, A. Rialland, Virtualization of food supply chains with the internet of things, J. Food Eng. 176 (2016) 128–136.

[29] Murad Khan, Bhagya Nathali Silva, Kijun Han, Internet of things based energy aware smart home control system, IEEE Access 4 (2016) 7556–7566.

[30] Yuzhe Jiang, Xingcheng Liu, Shixing Lian, Design and implementation of smart-home monitoring system with the internet of things technology, in: Wireless Communications, Networking and Applications, Springer, 2016, pp. 473–484.

[31] Mohamed Tabaa, Brahim Chouri, Safa Saadaoui, Karim Alami, Industrial communication based on modbus and node-red, Procedia Comput. Sci. 130 (2018) 583–588.

[32] Piyush Pariyumbud, Rohit Samkaria, Nitin Karnatak, Rajesh Singh, Anita Gehlot, Sushabhan Choudhary, Smart factory automation using internet of things to ensure quality manufacturing, J. Embedded Syst. Appl. 5 (3) (2018) 1–6.

[33] Paolo Brizzi, Davide Conzon, Hussein Khaleel, Riccardo Tomasi, Claudio Pastrone, A.M. Spirito, M. Knechtel, Ferry Pramudianto, P. Cultrona, Bringing the internet of things along the manufacturing line: A case study in controlling the industrial robot and monitoring energy consumption remotely, in: Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on, IEEE, 2013, pp. 1–8.

[34] P. Xiang, Design of smart home system based on the technology of internet of things, Res. J. Appl. Sci. Eng. Technol. 4 (14) (2012) 2236–2240.

[35] Isaac Machorro-Cano, José Oscar Olmedo-Aguirre, Giner Alor-Hernández, Lisbeth Rodríguez-Mazahua, José Luis Sánchez-Cervantes, Asdrúbal López-Chau, SCM-IoT: an aproach for internet of things services integration and coordination, Appl. Sci. 12 (6) (2022) 3133.

[36] Josimar Reyes-Campos, Giner Alor-Hernández, Isaac Machorro-Cano, José Oscar Olmedo-Aguirre, José Luis Sánchez-Cervantes, Lisbeth Rodríguez-Mazahua, Discovery of resident behavior patterns using machine learning techniques and IoT paradigm, Mathematics 9 (3) (2021) 219.

[37] Josimar Reyes-Campos, Giner Alor-Hernández, Isaac Machorro-Cano, José Luis Sánchez-Cervantes, Hilarión Muñoz Contreras, José Oscar Olmedo-Aguirre, IntelihOgarT: A smart platform to contribute comfort in intelligent home environments by using internet of things paradigm and machine learning, in: Technologies and Innovation: 6th International Conference, CITI 2020, Guayaquil, Ecuador, Vol. 6, 2020, pp. 139–150.

[38] Mario A. Paredes-Valverde, Giner Alor-Hernández, Jorge L. García-Alcaráz, María del Pilar Salas-Zárate, Luis O Colombo-Mendoza, José L. Sánchez-Cervantes, IntelliHome: An internet of things-based system for electrical energy saving in smart home environment, Comput. Intell. 36 (1) (2020) 203–224.

[39] Isaac Machorro-Cano, Giner Alor-Hernández, Mario Andrés Paredes-Valverde, Lisbeth Rodríguez-Mazahua, José Luis Sánchez-Cervantes, José Oscar Olmedo-Aguirre, HEMS-IoT: A big data and machine learning-based smart home system for energy saving, Energies 13 (5) (2020) 1097.

[40] Paolo Arcaini, Raffaela Mirandola, Elvinia Riccobene, Patrizia Scandurra, Alberto Arrigoni, Daniele Bosc, Federico Modica, Rita Pedercini, Smart home platform supporting decentralized adaptive automation control, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020, pp. 1893–1900.

[41] Francesco Gianni, Simone Mora, Monica Divitini, Rapiot toolkit: Rapid prototyping of collaborative internet of things applications, Future Gener. Comput. Syst. 95 (2019) 867–879.

[42] Francesco Gianni, Simone Mora, Monica Divitini, Rapid prototyping internet of things applications for augmented objects: The tiles toolkit approach, in: European Conference on Ambient Intelligence, Springer, 2018, pp. 204–220.

[43] Angelo P. Castellani, Nicola Bui, Paolo Casari, Michele Rossi, Zach Shelby, Michele Zorzi, Architecture and protocols for the internet of things: A case study, in: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, IEEE, 2010, pp. 678–683.

[44] Michael Blackstock, Rodger Lea, Toward a distributed data flow platform for the web of things (distributed node-red), in: Proceedings of the 5th International Workshop on Web of Things, ACM, 2014, pp. 34–39.

[45] Baihaqi Siregar, Syahril Efendi, Heru Pranoto, Roy Ginting, Ulfi Andayani, Fahmi Fahmi, Remote monitoring system for hydroponic planting media, in: 2017 International Conference on ICT for Smart Society, (ICISS), IEEE, 2017, pp. 1–6.

[46] Karthick Ramasamy, Mohammad Hossein Anisi, Anish Jindal, E2da: Energy efficient data aggregation and end-to-end security in 3d reconfigurable wsn, IEEE Trans. Green Commun. Netw. (2021).

[47] Sabrina Sicari, Alessandra Rizzardi, Alberto Coen-Porisini, Increasing the pervasiveness of the iot: fog computing coupled with pub & sub and security, in: 2020 IEEE International Conference on Smart Internet of Things, (SmartIoT), IEEE, 2020, pp. 64–71.

[48] Shwet Ketu, Pramod Kumar Mishra, Cloud, fog and mist computing in iot: An indication of emerging opportunities, IETE Tech. Rev. (2021) 1–12.

[49] Goudarzi Mohammad, Wu Huaming, Palaniswami Marimuthu, Buyya Rajkumar, An application placement technique for concurrent IoT applications in edge and fog computing environments, IEEE Trans. Mobile Comput. 20 (4) (2020) 1298–1311.

[50] Abdali Taj-Aldeen Naser, Hassan Rosilah, Aman Azana Hafizah Mohd, Nguyen Quang Ngoc, Fog computing advancement: Concept, architecture, applications, advantages, and open issues, IEEE Access 9 (2021) 75961–75980.

[51] Zhao Yongli, Wang Wei, Li Yajie, Meixner Carlos Colman, Tornatore Massimo, Zhang Jie, Edge computing and networking: A survey on infrastructures and applications, IEEE Access 7 (2019) 101213–101230.

[52] Liyanage Mohan, Chang Chii, Srirama Satish Narayana, mePaaS: mobile-embedded platform as a service for distributing fog computing to edge nodes, in: 17th International Conference on Parallel and Distributed Computing, Applications and Technologies, (PDCAT), 2016, pp. 73–80.

[53] Antonio Brogi, Stefano Forti, Ahmad Ibrahim, Luca Rinaldi, Bonsai in the fog: An active learning lab with fog computing, in: 2018 Third International Conference on Fog and Mobile Edge Computing, (FMEC), IEEE, 2018, pp. 79–86.

[54] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, Sorin-Aurel Moraru, Continuous testing in the development of iot applications, in: 2019 International Conference on Sensing and Instrumentation in IoT Era, (ISSI), IEEE, 2019, pp. 1–6.

[55] Miguel A. López-Peña, Jessica Díaz, Jorge E. Pérez, Héctor Humanes, Devops for iot systems: Fast and continuous monitoring feedback of system availability, IEEE Internet Things J. 7 (10) (2020) 10695–10707.

[56] A. Rizzardi, S. Sicari, D. Miorandi, A. Coen-Porisini, AUPS: An open source AUthenticated publish/subscribe system for the internet of things, Inf. Syst. 62 (2016) 29–41.

[57] Guo Yubin, Zhang Liankuan, Lin Fengren, Li Ximing, A solution for privacy-preserving data manipulation and query on nosql database, J. Comput. 8 (6) (2013) 1427–1432.

[58] R. Yugha, S. Chithra, A survey on technologies and security protocols: Reference for future generation IoT, J. Netw. Comput. Appl. 169 (2020) 102763, IEEE.

[59] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, A. Coen-Porisini, Security policy enforcement for networked smart objects, Comput. Netw. 108 (2016) 133–147.

[60] Giuseppe Bisicchia, Stefano Forti, Antonio Brogi, A declarative goal-oriented framework for smart environments with lpaas, 2021, arXiv preprint arXiv:2106.13083.

[61] Shantanu Pal, Ali Dorri, Raja Jurdak, Blockchain for IoT access control: Recent trends and future research directions, J. Netw. Comput. Appl. 203 (2022) 103371.

Alessandra Rizzardi is Assistant Professor at University of Insubria (Varese), where she received BS/MS degree in Computer Science 110/110 cum laude in 2011 and 2013, respectively. In 2016 she got Ph.D. in Computer Science and Computational Mathematics at the same university, under the guidance of Prof. Sabrina Sicari. Her research activity is on WSN and IoT security issues. She is member of ETT, ITL, and Sensors editorial board. She is IEEE member.

Sabrina Sicari is Associate Professor at University of Insubria (Varese). She received degree in Electronical Engineering, 110/110 cum laude, from University of Catania, in 2002, where in 2006 she got Ph.D. in Computer and Telecommunications Engineering, followed by Prof. Aurelio La Corte. She is member of COMNET, IEEE IoT, ETT, ITL editorial board. Her research activity security, privacy and trust in WSN, WMSN, IoT, and distributed systems. She is IEEE senior member.

Alberto Coen Porisini received Dr. Eng. degree and Ph.D. in Computer Engineering from Politecnico di Milano in 1987 and 1992. He is Full Professor of Software Engineering at Universitá degli Studi dell'Insubria since 2001, Dean of the School of Science from 2006 and Dean from 2012 to 2018. His research regards specification/design of realtime systems, privacy models and WSN.