IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 11, 2022, accepted May 30, 2022, date of publication June 8, 2022, date of current version June 15, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3181166

A Domain-Specific Language for Modeling IoT System Architectures That Support Monitoring

LENIN ERAZO-GARZÓN[1], PRISCILA CEDILLO[2], (Member, IEEE), GUSTAVO ROSSI[3], AND JOSÉ MOYANO[2]

[1]LIDI, Universidad del Azuay, Cuenca 010204, Ecuador
[2]Department of Computer Science, Universidad de Cuenca, Cuenca 010203, Ecuador
[3]LIFIA, Fac. Informática, UNLP y CONICET, La Plata 1900, Argentina

Corresponding author: Priscila Cedillo (priscila.cedillo@ucuenca.edu.ec)

This work was supported in part by the Vice-Chancellor for Research at the Universidad del Azuay under Grant 2020-0059, Grant 2021-0040, and Grant 2022-0030; and in part by the Vice-Chancellor for Research at the Universidad de Cuenca through the Project "Fog computing applied to monitor devices used in assisted living environments; study case: A platform for the elderly," under Grant XVIII-2018.

ABSTRACT The Internet of Things (IoT) is a technological paradigm involved in a diversity of domains with favorable impacts on people's daily lives and the development of industry and cities. Nowadays, one of the most critical challenges is developing software for IoT systems since the traditional Software Engineering methodologies and tools are unproductive in the face of the complex requirements resulting from the highly distributed, heterogeneous, and dynamic scenarios in which these systems operate. Model-Driven Engineering (MDE) emerges as an appropriate approach to abstract the complexity of IoT systems. However, there are no domain-specific languages (DSLs) aligned to standardized reference architectures for IoT. Furthermore, existing DSLs have an incomplete language to represent the IoT entities that may be needed at the edge, fog, and cloud layers to monitor IoT environments. Therefore, this paper proposes a domain-specific language named Monitor-IoT, which supports developers in designing multi-layer monitoring architectures for IoT systems with high abstraction, expressiveness, and flexibility. Monitor-IoT consists of a high-level visual modeling language and a metamodel aligned with the ISO/IEC 30141:2018 reference architecture. In addition, it provides a language capable of modeling architectures with a wide variety of digital entities and dataflows (synchronous and asynchronous) between them across the edge, fog, and cloud layers to support the monitoring of a diversity of IoT scenarios. The empirical evaluation of Monitor-IoT through the application of an experiment, which contemplates the use of the Technology Acceptance Model (TAM), demonstrates the intention of the participants to use this tool in the future since they consider it easy to use and useful.

INDEX TERMS Architecture, domain-specific language (DSL), Internet of Things (IoT), metamodel, model-driven engineering (MDE), monitoring.

I. INTRODUCTION

The Internet of Things (IoT) is a technological paradigm that has evolved dramatically in recent years. Its growth has been stimulated by the great variety of intelligent objects (things) that are being interconnected through the Internet to provide services to users across a broad spectrum of application domains (e.g., home, health, education, industry, transportation) [1], [2]. According to Gartner [3], the total number of IoT devices installed in 2017 (8.4 billion) already exceeded the total population of human beings; furthermore, a forecast by the International Data Corporation (IDC) [4] estimates that there will be 41.6 billion IoT devices in 2025.

Currently, one of the most critical challenges for the scientific community and the industry is the development of software for IoT systems because traditional Software Engineering methodologies and tools are mainly focused on the implementation domain (programming to low level) instead of the problem domain [5]. In addition, those solutions have proven to be not very productive in the face of the intrinsic characteristics of IoT systems [6], such as i) *heterogeneity*, made up of a set of physical devices, each one with different programming interfaces (APIs), operating systems, computing capabilities, communication protocols (e.g., CoAP, MQTT, HTTP), and data exchange standards (e.g., XML,

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

JSON, RDF); ii) *distributed computing*, comprises a set of computing nodes, located on the edge, fog, or cloud to achieve a balance in the use of resources; iii) *scalability*, requiring the transparent expansion of its infrastructure, discovering and incorporating new devices or services in real-time; and iv) *uncertainty*, operate in highly changeable scenarios, making it impossible to identify all the requirements in the development stages of an IoT system [7].

Therefore, there is a need to create formal Software Engineering methods and tools adapted to the specificities of IoT systems, which simplify and speed up their development with a high level of abstraction, flexibility, and extensibility [6]. Model-Driven Engineering (MDE) emerges as a promising software development approach to bridge the problem-implementation gap and consequently abstract the complex and dynamic aspects of the new generation of systems (e.g., IoT) and their environment [8], [9]. Specifically, to address the complexity of these systems, MDE focuses on developing technologies that combine domain-specific languages (DSLs), transformation engines, and generators [9].

There are several proposals on DSLs oriented to the IoT domain [10]–[19]. However, they are not based on a globally accepted metamodel or aligned with a standardized reference architecture for IoT (e.g., ISO/IEC 30141:2018 [20]). In turn, these DSLs have a language made up of a reduced set of digital entities and dataflows (interactions) between them at the edge, fog, and cloud layers; this limits the representation of the possible variants of monitoring processes that may be required, which depend on the hardware capabilities of the devices, microcontrollers, and servers, as well as the network bandwidth and data volume, among the most important. According to Mineraud *et al.* [21], one of the research gaps in IoT is the absence of DSLs that offer functional primitives to describe the problem and the solution space with a high level of abstraction (e.g., primitives to manage dataflows catalogs or data fusion and aggregation operations), in order to simplify the development of IoT applications.

This study is a first step towards developing a methodological approach and a set of tools based on MDE [8] and the MAPE-K feedback loop [22] to build self-aware and self-adaptive IoT systems. That is, systems capable of obtaining metrics to autonomously evaluate the current state and possible future evolution of themselves and their environment and, based on this knowledge, act if necessary (e.g., explain, report, suggest, self-adapt). The MAPE-K loop consists of four stages: i) monitor or collect data on the state of the system and its environment; ii) analyze the data collected to assess the situation and detect any anomaly or problem; iii) plan how to adapt the system to solve a detected problem; and iv) execute the adaptation plan based on knowledge of the system and its environment [22].

In this context, the scope of this paper is the monitoring stage of the MAPE-K. Hence, a domain-specific language named Monitor-IoT is proposed to design IoT system architectures that support monitoring through the edge, fog, and cloud layers. Monitor-IoT DSL is mainly oriented to IoT applications that are characterized by requiring an infrastructure that efficiently integrates various resources from the edge, fog, and cloud layers in order to collect, transport, process, and store data at different levels of aggregation for later analysis, unlike other IoT applications that are limited only to reacting from the data collected at that time.

Monitor-IoT provides a graphical designer (high-level visual language) built in the Obeo Designer Community and Eclipse Sirius tools [23] to support developers in modeling IoT multi-layer monitoring architectures with a high level of abstraction, expressiveness, and flexibility. Thus, developers are freed from implementation details, facilitating and speeding up the development tasks. Furthermore, Monitor-IoT is based on a metamodel specified in Ecore [24] and aligned with the reference architecture for IoT ISO/IEC 30141:2018 [20] that provides a language to mainly support: i) the definition of entities (physical or digital) to be monitored; ii) the definition of digital entities (computing nodes and their resources) that support the monitoring processes (data collection, transport, processing, and storage) at the edge, fog, and cloud layers; iii) the specification of the properties to be monitored for each entity; iv) the definition of dataflows between digital entities, based on synchronous or asynchronous communication; and v) the establishment of aggregation operations for the collected data.

It should be emphasized that although Monitor-IoT is oriented in the first instance to the design of architectures that support the monitoring processes of IoT systems, the proposed metamodel is highly flexible and extensible. The expansion and validation of Monitor-IoT to support other types of processes (e.g., analysis, generation of reports for decision-making, actuation) will be addressed in future work since the final purpose is to build a DSL that covers all the stages of the MAPE-K.

The structure of this paper is as follows: Section 2 presents a brief overview of the main concepts addressed in this study and a review and comparative analysis of related work. Section 3 describes the Monitor-IoT DSL. Section 4 explains the proposed process to design monitoring architectures for IoT systems with Monitor-IoT. Section 5 provides an illustrative example to demonstrate the usefulness of Monitor-IoT. Section 6 describes the empirical evaluation of Monitor-IoT. Finally, Section 7 presents the conclusions and lines of future work.

II. BACKGROUND AND RELATED WORK

The term IoT refers to a global infrastructure made up of a set of ubiquitous intelligent objects interconnected by information and communication technologies to collect, store, process, and analyze data from the physical and virtual world and react with the minimum human intervention [7], [25], [26].

An IoT infrastructure includes: i) *devices* to collect data (monitoring) or perform an action on the environment (actuation), which can be standard or specialized objects. These objects can be carried by people, be located in homes,

form part of the infrastructure of a city, or be integrated into the equipment of a factory; ii) *gateways* to securely interconnect the devices with the cloud, controlling the exchange of data between them; and iii) c*loud* to store, process, and analyze data in real-time or in batches in order to obtain knowledge and respond to changes in the environment [27], [28].

Additionally, due to the exponential growth of IoT devices and data generated [4], a solution based solely on the cloud may be impractical for some IoT scenarios. Therefore, an IoT infrastructure may require solutions that decentralize applications, management, and data analysis to solve performance problems, network congestion, security, reliability, among others. In this sense, as a complement to cloud computing, two solutions or technological paradigms have been proposed: edge computing and fog computing. Edge computing is the peripheral network layer, also known as proximity network or IoT network, encompassing the IoT devices (sensors, actuators) and other devices (IoT gateways) to which the IoT devices connect to access local or wide area networks. Its purpose is to provide local storage and processing capabilities to these devices to react immediately without transferring the data to another location. In comparison, fog computing comprises a set of decentralized computing nodes located in local or wide area networks (before the cloud) that act as mediating instances between the cloud and the devices located at the network edge (sensors, actuators, IoT gateways). Fog computing decouples the hardware and software functions from the cloud, allowing dynamic reconfigurations of what data are stored and processed in the fog nodes and what data are prepared and sent to the cloud for storage and further analysis. Therefore, fog computing is hierarchical, running different IoT applications that process data from a large number and diversity of devices, unlike edge computing, which runs specific IoT applications for a small number of devices [29].

Consequently, IoT takes advantage of a large part of existing and emerging technologies, combining them to create new products and services, and ultimately improve the user experience. Among the IoT-enabled technologies are: communication networking technologies, sensing/control technologies, device/hardware technologies, and software technologies [1], [2].

Model-Driven Engineering (MDE) is a software development approach in which domain models are created and systematically transformed into concrete implementations [8]. Therefore, MDE increases the importance of models since they are used for documentation and communication, and as central artifacts of the software development process. Furthermore, models raise the abstraction level and automation of the development process, which favors the management of software complexity and change, and improves several of the quality attributes of software (for example, productivity, maintainability, flexibility) [30].

The fundamental elements of MDE are: i) *model*, abstract representation of an aspect of a software system; ii) *domain-specific language (DSL)*, a language with a concrete syntax (notation), abstract syntax (metamodel), and specialized semantics to create models in a specific domain; iii) *metamodel*, a conceptual model of a DSL that defines the concepts of language and the relationships between them, as well as the rules that establish when a model is well-formed; and iv) *model transformations*, to achieve the automation of the models, through their translation into code [31].

The following subsections present a bibliographic review of the main proposals of reference architecture models, metamodels, and DSLs existing in the literature for the IoT domain. In addition, a comparative analysis between different DSL proposals and this work (Monitor-IoT) is included.

A. REFERENCE ARCHITECTURE MODELS FOR IoT

ISO/IEC 30141:2018 [20] provides a standardized IoT reference architecture, which includes a generic conceptual model that describes the concepts of the common entities of an IoT system and their relationships. The conceptual model is derived into a high-level reference model, broken down into five architectural views: functional, system, information, communication, and usage. In turn, Bauer *et al.* [32] propose an IoT Architectural Reference Model (IoT ARM) based on five submodels: i) domain [33], includes the main IoT concepts (e.g., physical entities, virtual entities, devices, resources, services) and the relationships between these; ii) information, defines the structure of information related to IoT; iii) functional, identifies groups of functionalities of an IoT system; iv) communication for heterogeneous IoT environments; and v) trust, security, and privacy for IoT. In the same way, Patel *et al.* [34], [35] present a methodology that separates the development of IoT applications into four concerns: domain, functional, implementation, and platform, which are represented in a conceptual model.

B. METAMODELS AND DSLs FOR IoT

Concerning metamodels and DSLs for the IoT domain, the most relevant proposals are described and analyzed below:

1) Eterovic *et al.* [10] focus on creating an IoT systems development language powerful enough for professionals and understandable enough for end-users that provide the requirements. Hence, they propose a Visual Domain-Specific Modeling Language (VDSML) for IoT using a stereotype-based UML profile. The VDSML allows the modeling of real and virtual things, which can contain a virtual collection of items, such as inputs (sensors), outputs (actuators), and software components. One limitation is the low specialization of the language, requiring more specific stereotypes to represent edge, fog, or cloud nodes, as well as software components, such as middleware, services, applications, or databases. Additionally, a usability test was used to validate the VDSML by calculating the SUS (System Usability Scale) score, task success rate, time on task, and user error rate.

2) Salihbegovic *et al.* [11] propose DSL-4-IoT to tackle the complexity and heterogeneity of IoT systems. This is a VDSML to design the structure of an IoT system using hierarchical blocks (system, subsystem, devices, device channels) mainly focused on the edge layer. The blocks can be saved in libraries to support reusability and scalability. The configuration files generated on the DSL-4-IoT run on the OpenHAB runtime engine. In turn, to demonstrate the viability and usability of the solution, an IoT testbed was implemented, spanning a variety of static and mobile sensors in two application domains (smart home and remote patient monitoring).
3) ThingML [12] includes a modeling language, a methodology, and tools to support multiplatform code generation for distributed reactive systems. The ThingML language comprises two structures: i) Things that represent software components; and ii) Configurations that describe their interconnection. Software components can include properties, functions, messages, ports, and state machines. However, ThingML does not allow modeling the nodes on which the software components run at the edge, fog, and cloud layers. Several case studies, combining various implementation platforms, were carried out to evaluate ThingML.
4) Pramudianto *et al.* [13] present an architecture for developing IoT prototypes, separating domain modeling from technological implementations. Using a model-driven tool called IoTLink, domain experts can create domain models, composing virtual objects linked to implementation technologies (sensors, actuators) to abstract their complexities and specificities. IoTLink generates artifacts in Java from the defined model. As with ThingML, a disadvantage of IoTLink is that it does not cover modeling of the compute nodes on which Java artifacts reside and run. Finally, IoTLink was evaluated against classical Java development using a controlled experiment in terms of development time and user satisfaction.
5) Negash *et al.* [14] introduce a flexible and scalable approach that enhances programmability and modifiability in the perception layer of an IoT system, especially in resource-constrained devices. The approach uses a domain-specific language (DoS-IL) with a textual notation to write lightweight scripts and store them in a gateway. In turn, the scripts are requested and executed by the perception layer devices using an embedded resource browser that integrates a DoS-IL interpreter and manipulates a Device Object Model (DOM). However, this proposal does not present an empirical evaluation.
6) Costa *et al.* [15] propose a DSL (SoaML4IoT) to design SOA-based IoT systems. The DSL extends the Service Oriented Architecture Modeling Language (SoaML) with specific concepts of the IoT domain. The edge nodes are represented in a general way, requiring specialization in sensors, actuators, tags, or IoT gateways (controllers). Instead of an empirical evaluation, a comparative analysis with other DSL proposals was performed to determine the level of expressivity of SoaML4IoT.
7) Alulema *et al.* [16] present a model-driven approach that includes a DSL, a graphical editor, and a Model to Text (M2T) transformation to generate the code for IoT nodes containing controllers, sensors, and actuators. Consequently, this solution only targets the edge layer of an IoT platform. In turn, the authors designed a smart home scenario to demonstrate the functionality and feasibility of the proposed approach.
8) C. G. Garcia *et al.* [17] propose a graphical DSL (MOCSL) aimed at people without programming knowledge in order to facilitate the creation of native applications for smart objects interconnected through the IoT Midgar platform. With MOCSL, users can create the necessary logic for their objects by only selecting the sensors and actuators they want to use on their smartphone or Arduino. Therefore, MOCSL also only focuses on the edge layer of an IoT platform. In addition, to validate the usefulness and efficiency of the solution, the authors carried out an experiment in which two profiles of participants with different levels of knowledge about smart objects had to create a basic Arduino application using MOCSL and another graphical editor.
9) Barriga *et al.* [18] have built a DSL (SimulateIoT) to design, code, and deploy IoT system simulations. The solution comprises a domain metamodel, a graphical concrete syntax, and model-to-text transformation algorithms. The IoT simulation model created in the graphical editor can include sensors, actuators, fog nodes, cloud nodes, databases, and complex event processing engines. However, these elements can only be connected using asynchronous communication protocols (publish-subscribe). The solution was implemented in two case studies of IoT environments (smart building and agriculture) to demonstrate the expressiveness of the solution through the opinion and impression of the authors, suggesting a need for the use of more rigorous techniques (based on statistics) to test the end-user perception.
10) Alfonso *et al.* [19] propose a DSL to model the static and dynamic aspects of an IoT deployment. The DSL includes an MPS projectional editor with textual, tabular, and tree view notation. Furthermore, the solution comprises a prototype Kubernetes manifest generator to deploy the modeled IoT systems. This work only provides a proof of concept of the code generator prototype. Therefore, as with the previous proposal, a more rigorous empirical evaluation is required to validate the usability and usefulness of the DSL concerning the end-user.

C. COMPARATIVE ANALYSIS

Based on this bibliographic review, there are important advances in the field of DSLs for IoT, whose common denominator has been abstracting the complexity and heterogeneity of IoT systems to speed up their development.

However, several limitations have been identified in the related work. Tables 1 and 2 present a comparative analysis between the DSLs studied and Monitor-IoT. The main limitations and challenges encountered and that are addressed in this paper through Monitor-IoT are discussed below:

1) Although some proposals on DSLs [13], [14] are based on the IoT Architectural Reference Model (IoT ARM) [32], there is no globally accepted metamodel. Ambiguities have even been observed between the solutions studied regarding the meaning and use of various IoT terms and concepts, the lack of consensus being an open problem. Furthermore, the proposed DSLs are not based on a metamodel aligned with the standardized

TABLE 1. Comparative analysis between the DSLs studied and Monitor-IoT. (Part 1: Capacity of specification and modeling of entities, resources, properties, and dataflows.)

Related Work	Physical entity	Edge Node		Fog Node	Cloud Node	Communication Networks and Protocols (synchronous, asynchronous)	Resource		Entity Property	Entity category and property template to promote reuse	Dataflow	Database structure (tables, columns)
		IoT Device (Sensor, Actuator, Tag)	IoT Gateway				Hardware (CPU, RAM, Network Interface, Battery, etc.)	Software				
Eterovic *et al.* [10]	Yes	Yes	No	No	No	No	No	Software Component.	No	No	Yes	No
DSL-4-IoT [11]	No	Yes	Yes	No	Yes	Yes	No	Service.	No	No	No	No
ThingML [12]	No	No	No	No	No	Yes	No	Software Component, API.	Yes	No	No	No
IoTLink [13]	Yes	Yes	No	No	No	Yes	No	Software Object (class instance), API, Service, Database.	Yes	No	Yes	No
DoS-IL [14]	No	Yes	No	No	No	Yes	Resources available in the constrained device (DOM).	Resources available in the constrained device (DOM).	Yes	No	No	No
SoaML4IoT [15]	Yes	Yes	Yes	Yes	Yes	Partial (synchronous)	No	Application, Software Component, Service.	Yes	No	No	No
Alulema *et al.* [16]	No	Yes	Yes	No	No	Partial (synchronous)	No	Service.	No	No	Yes	No
MOCSL [17]	No	Yes	Yes	No	No	Partial (synchronous)	No	Application.	No	No	No	No
SimulateIoT [18]	No	Yes	No	Yes	Yes	Partial (asynchronous)	No	Broker, Topic, Database, CEP (Complex Event Processing) Engine, ESP (Event Stream Processing) Engine.	No	No	Yes	No
Alfonso *et al.* [19]	Partial (Region)	Yes	Yes	Yes	Yes	Partial (asynchronous)	Resource representation non-extensible. The resources (CPU, RAM, storage) are defined as attributes of the Node, Container, and Application metaclasses.	Container, Application, Topic.	No	No	No	No
Monitor-IoT	Yes	Yes	Yes	Yes	Yes	Yes	Network Interface through a specific metaclass; and any other resource through the generic Resource metaclass.	API, Application, Middleware, Broker, Topic, Service, and Database through specific metaclasses; and any other resource through the generic Resource metaclass.	Yes (any property for physical and digital entities through the Property metaclass)	Yes	Yes (any combination of dataflows between digital entities at the edge, fog, and cloud layers)	Yes

TABLE 2. Comparative analysis between the DSLs studied and Monitor-IoT. (Part 2: Standardization, visual modeling language, methodology, and empirical evaluation.)

Related Work	Alignment with ISO/IEC 30141:2018	Visual Modeling Language (Graphical Editor)	Methodological process that guides the use of the DSL	Empirical evaluation	
				Type	Purpose
Eterovic *et al.* [10]	No	Yes	No	Experiment (Usability test)	Prove that the language is suited for non-technical and technical users in terms of simplicity.
DSL-4-IoT [11]	No	Yes	No	Proof of concept	Demonstrate the viability and usability of the solution.
ThingML [12]	No	No	Yes	Case study	Demonstrate the functionality and feasibility of the solution.
IoTLink [13]	No	Yes	Yes	Experiment (Questionnaire, descriptive statistics, and hypothesis testing)	Evaluate the effectiveness, efficiency, and usability of the tool.
DoS-IL [14]	No	No	No	None	
SoaML4IoT [15]	No	Yes	No	None	
Alulema *et al.* [16]	No	Yes	No	Case study (Based only on the opinion and impression of the authors. Statistical techniques have not been used)	Demonstrate the functionality and feasibility of the solution.
MOCSL [17]	No	Yes	No	Experiment (Questionnaire, descriptive statistics, box plots, and hypothesis testing)	Validate the Usefulness of the DSL for users without programming knowledge and the level of efficiency concerning other alternatives.
SimulateIoT [18]	No	Yes	Yes (general)	Case study (Based only on the opinion and impression of the authors. Statistical techniques have not been used)	Demonstrate the expressiveness of the DSL.
Alfonso *et al.* [19]	No	No	No	Proof of concept	Demonstrate the functionality and feasibility of the solution.
Monitor-IoT	Yes	Yes	Yes (detailed)	Quasi-experiment (Technology Acceptance Model - TAM [36], questionnaire, descriptive statistics, box plots, hypothesis testing, and simple linear regression analysis)	User perception about the ease of use, usefulness, and intention of future use of the DSL.

reference architecture for IoT ISO/IEC 30141:2018 (see Table 2).

2) The proposals studied are based on metamodels with a limited language to represent the key concepts that may be required for monitoring an IoT system, such as physical entities, digital entities (computing nodes and their resources), communication networks and protocols, entity properties, and dataflows between entities. Table 1 shows in detail that, unlike Monitor-IoT, the analyzed DSLs cannot specify or model all the key concepts mentioned.
3) Regarding physical entities, three proposals [10], [13], [15] include the possibility of representing any physical entity in their metamodels. In turn, the proposal by Alfonso *et al.* [19] allows representing only the regions or places to monitor in an IoT scenario.
4) Most of the DSLs studied focus on the edge layer of an IoT infrastructure, allowing the specification of sensors, actuators, tags, and to a lesser extent, IoT gateways located in the proximity network. On the contrary, there are only three studies [15], [18], [19] that support the modeling of fog and cloud nodes in their metamodels.
5) Some proposals represent the software and hardware resources used by edge, fog, and cloud nodes through generic concepts in their metamodels. However, this limits the possibility of specifying in detail the attributes and configurations of the essential resources to monitor and control IoT scenarios (APIs, applications, services, databases, middlewares, brokers, network interface), requiring a specialization of these resources in the metamodels. In turn, although other proposals focus on specialization, they do not represent all the essential resources mentioned. Therefore, metamodels that combine these two approaches are required, including a specialization/generalization relationship between subclasses to define in detail the essential resources of an IoT system and a concrete superclass to represent any other hardware or software resource (e.g., CPU, RAM, Battery).
6) In particular, the studied solutions and their metamodels have limitations in representing the software or data resources used by a typical IoT monitoring scenario. Thus, only two proposals [13], [18] consider the database concept in their metamodels; however, these proposals do not allow modeling the logical storage structure of the database (tables and columns). In turn, only one study [13] allows specifying aggregation and merger operations on the monitored data. Also, several metamodels do not allow specifying the compute nodes on which the resources reside or run. For example, although two proposals [18], [19] contemplate using topics to support asynchronous communication, they do not incorporate the broker concept to represent the resource in which those topics are managed. Another limitation is that the solutions assume that a sensor can only collect data for a single property, unlike reality, where there are sensors that can collect data for multiple properties (e.g., the DHT11 sensor collects temperature and humidity) [18].
7) A group of DSLs [12]–[15] explicitly include the property concept in their metamodels to characterize certain types of IoT entities. However, it is necessary to delve into this aspect in order to propose metamodels with an extensible structure capable of supporting the specification of various types of properties (telemetry, state, and metadata) for any IoT entity, whether physical or digital (computing nodes and resources), depending on the needs of each IoT scenario. In this sense, telemetry properties have information about a physical entity collected from sensors (e.g., temperature in a room, blood pressure of a patient); state properties contain information that describes the current status of a digital entity (e.g., battery power level of a sensor, amount of memory or processing used by an edge node, bandwidth used by a network); and metadata, which have information that rarely changes and that is known at design time (e.g., serial number, brand, model).
8) Since an IoT system can have many entities of the same type, metamodels must also support generic structures (e.g., property templates per entity category) to allow common properties between entities to be specified only once. However, this is an aspect that is not considered by the related work. Another aspect not covered is mapping the properties to be monitored both with the APIs responsible for collecting the data and with the storage structures (database, tables, columns) where said data are stored.
9) The proposals [10], [13], [16], [18] that support the modeling of dataflows do not cover the diversity of possibilities of communication links and data exchange between digital entities (APIs, applications, services, databases, middlewares, brokers) that the IoT scenarios may require across the edge, fog, and cloud layers. Depending on the hardware capabilities of the IoT devices, the network bandwidth, and the data volume, different types of digital entities and communication configurations between them may be needed to support the motorization of an IoT environment. From direct communications between the IoT devices (sensors) and the cloud nodes to including intermediate nodes located on edge (IoT gateway) or fog layers, responsible for carrying out the routing, processing (merging, aggregation), and data storage operations. Also, synchronous or asynchronous communications between digital entities may be required. Therefore, one of the challenges is to propose sufficiently flexible and extensible DSLs to allow graphical modeling (with a high degree of abstraction) of dataflows as a sequential set of various types of communication links between digital entities. In turn, it must be possible that the types of links between digital entities can be combined and ordered in different ways

to form dataflows according to the needs of each IoT scenario.

10) On the other hand, only three proposals [12], [13], [18] (see Table 2) have included as part of their solution the definition of a methodological process that guides the tasks of specification, modeling, and implementation of IoT solutions with the support of the proposed DSL tool. In this sense, it is desirable to have a methodological process associated with a DSL tool to ensure that end-users can use it in a disciplined, efficient, and effective manner.
11) Finally, although most of the DSLs studied have a visual modeling language (graphical concrete syntax), few proposals [10], [13], [17] have carried out a rigorous empirical evaluation based on the end-user perception to validate the ease of use and usefulness of the DSL tool (see Table 2). However, these proposals focus only on the edge layer or have a limited modeling language. On the contrary, those proposals that focus on the edge, fog, and cloud layers [15], [18], [19] do not include an empirical evaluation or have only validated their solutions based on the opinion and impression of their authors.

III. MONITOR-IoT DSL

First, the Monitor-IoT metamodel (abstract syntax) is described, including a review of the main metaclasses and their relationships (subsection A). Then, the Monitor-IoT graphical designer (concrete syntax) is presented (subsection B).

A. MONITOR-IoT METAMODEL

The Monitor-IoT metamodel was built from the reference architecture for IoT ISO/IEC 30141:2018. This architecture includes a generic and abstract conceptual model that describes the concepts of the key entities (physical and digital) and their relationships within a typical IoT system [20]. Hence, the key entities and relationships of this conceptual model were reused to create the metamodel. This reuse (alignment) provides the metamodel with a common vocabulary that can be used unambiguously in any IoT subdomain or implementation. Specifically, 18 of the 21 concepts contained in the conceptual model were reused. Most of the reused concepts (entity, physical entity, digital entity, network, IoT user, human user, non-human user, IoT device, sensor, tag, actuator, IoT gateway, application, service, database) have been represented as metaclasses in the metamodel, except three concepts (domain, endpoint, identifier) that have been included as attributes of metaclasses (see Fig. 9 and 10). Then, based on the analysis of the related work, the most common entities among the proposals, which also are relevant for data collection, transport, processing, and storage of an IoT system, were included in the metamodel. In turn, additional entities necessary to represent important monitoring aspects not covered (e.g., dataflows, property templates, database structure, middlewares, brokers, property mapping with APIs and database, data aggregation operations) were incorporated into the metamodel. Finally, it was essential to understand how entities collaborate during the monitoring processes of an IoT environment in order to define the relationships between said entities.

This construction approach has made it possible to ensure the semantic coherence of the metamodel (alignment with ISO/IEC 30141:2018) and provide it with a more complete and flexible language capable of modeling a variety of alternatives for monitoring processes and dataflows in IoT systems.

Furthermore, Monitor-IoT allows the construction of IoT monitoring architectures with asynchronous and synchronous communication between the computing nodes. In the first case, the sending and receiving of data are temporarily separated, which is crucial for improving the performance of large-scale IoT systems. Whereas in the second case, the data exchange (sending - receiving) is carried out in real-time to prioritize those requests that require an immediate response.

The Monitor-IoT metamodel has been specified in Ecore, using the tools included in the Eclipse Modeling Framework (EMF) [24]. Fig. 1 presents an extract of the metamodel with the main metaclasses and relationships. In contrast, Fig. 9, 10, 11, and 12 (shown in Appendix A) contain the complete metamodel, including all metaclasses, relationships, attributes, and enumerations. The metaclasses included in the figures are identified with a number enclosed in a circle of different colors. The red identifies the metaclasses aligned with ISO/IEC 30141:2018, the blue identifies the metaclasses selected from the proposals studied (related work), and the green identifies the additional metaclasses incorporated by the researchers of this work.

The most important Monitor-IoT metaclasses are described below, while the remaining metaclasses are detailed in Appendix.

MonitoringArchitectureModel. Main metaclass that describes and contains the multi-layer monitoring architecture model of an IoT system (see Fig. 1(1), 9(1), and 11(1)).

IoTSystem. Represents a system composed of a set of physical (things) and digital entities that interact and cooperate in the collection, transmission, processing, and storage of data from the physical and virtual world, as well as to act on physical entities from digital instructions, in order to provide services to end-users in a variety of domains (e.g., home, health, transportation, industry). The metamodel enables the representation of IoT ecosystems by reusing the physical and computational capabilities of individual IoT systems (subsystems) existing in different domains. This solution contributes to the provision of smarter emerging services with efficient use of resources. For example, an emerging IoT disaster recovery ecosystem can be made up of a transportation system, a health system, and an emergency system (see Fig. 1(2) and 9(2)).

Entity. A global concept; it represents anything (physical or digital) with distinctive and independent characteristics (e.g., places, people, home appliances, electronic devices,

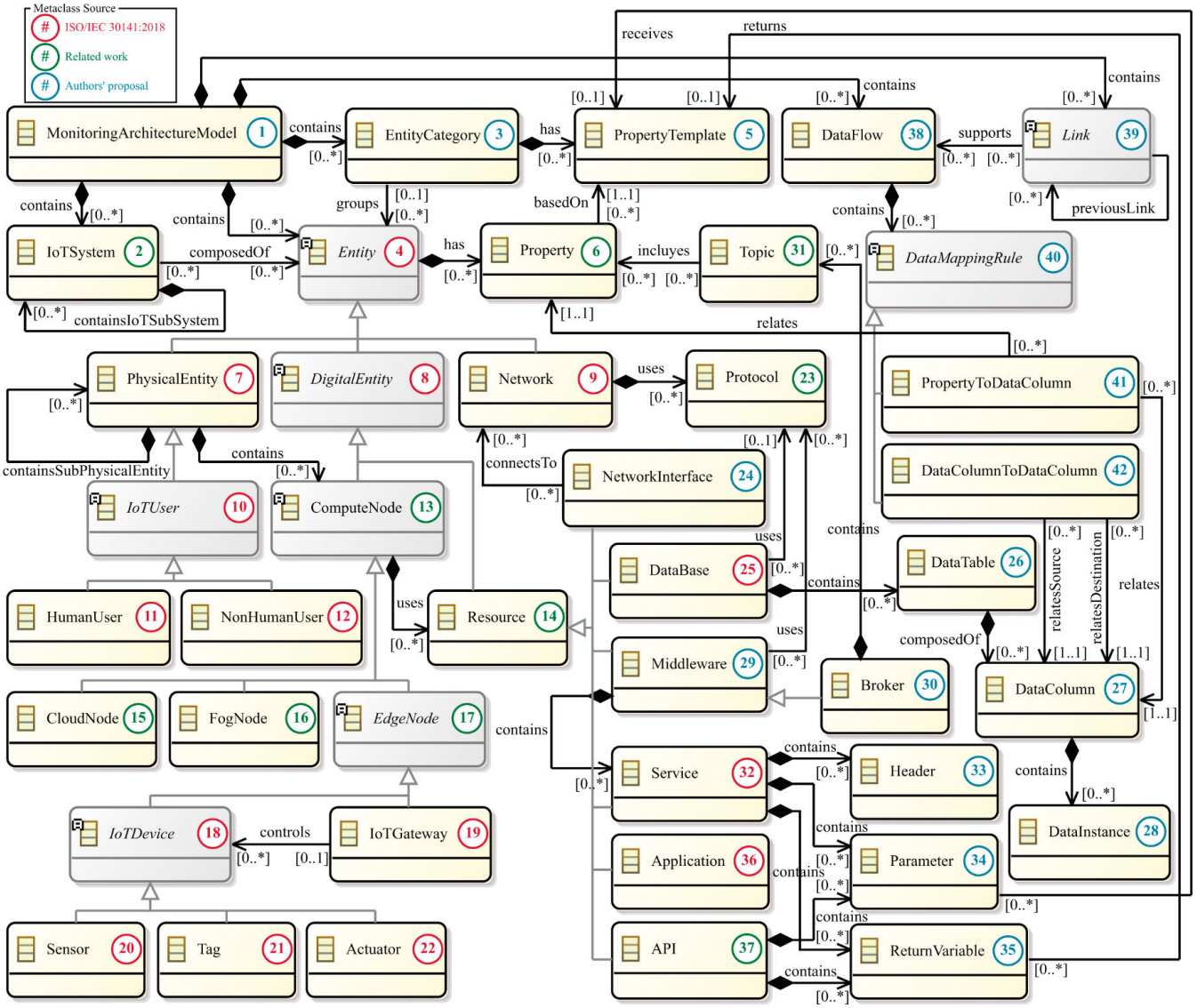

FIGURE 1. Extract of the Monitor-IoT metamodel with the main metaclasses and relationships.

servers, communication networks, software components). In some of the proposals studied, it is often called "entity of interest" to highlight that it is something of user interest to achieve their objectives (see Fig. 1(4) and 9(4)).

PhysicalEntity. Represents a real-world thing that is monitored by a sensor and/or controlled by an actuator. Physical entities can be: living organisms, objects, or the environment (e.g., buildings, people, animals, vehicles, home appliances, articles, clothing, electronic devices). A physical entity extends to the digital world by incorporating, containing, or carrying one or more information and communication technology devices that provide an interface to obtain information or act on the physical entity. In addition, a physical entity can contain other physical entities. For example, a house can contain rooms, which in turn contain home appliances (see Fig. 1(7) and 9(8)).

DigitalEntity. Represents a computational element (at the hardware or software level) of an IoT system. These elements can be: cloud nodes, fog nodes, IoT gateways, IoT devices (sensors, tags, actuators), middlewares, databases, services, APIs, applications, network interfaces, among others. Therefore, digital entities can be specialized in: computing nodes and hardware or software resources (see Fig. 1(8) and 9(9)).

Property. Defines an attribute that characterizes a particular physical or digital entity. Properties make it possible to identify or describe an entity, determine its state at a given moment, as well as its evolution over time (see Fig. 1(6), 9(7), 10(10), and 11(7)). In turn, the general specifications of a property are contained in a template using the *PropertyTemplate metaclass* (see Appendix A).

IoTUser. Represents an actor or beneficiary of an IoT system that can specialize in: i) *HumanUser*, a person who

interacts with an IoT system through one or more applications that run on user devices (e.g., personal computer, tablet, smartphone, other specialized devices); and ii) *NonHumanUser*, machine, artifact, or device (e.g., robot, vehicle) that acts on behalf of human users and can interact with one or more services offered by an IoT system through a communication network (see Fig. 1(10-12) and 9(11-13)).

ComputingNode. Represents a digital entity with capabilities to collect, exchange, process, and store data in an IoT system. Computing nodes can be located on the cloud, fog, or edge layers. In addition, these nodes can contain hardware and software resources, such as network interfaces, middleware, brokers, databases, services, APIs, or applications (see Fig. 1(13), 9(14), and 10(2)).

CloudNode. A centralized computing node with high capacities to process, analyze, and store the data obtained directly from the IoT devices or fog and edge intermediate computing nodes (see Fig. 1(15) and 9(16)).

FogNode. A decentralized computing node that acts as a mediating instance between the cloud and edge devices (sensors, actuators, IoT gateways). Fog nodes play the role of the intermediate layer; they decide what data are stored and processed in this layer and what data are prepared and sent to the cloud for storage and subsequent analysis. The objective of fog nodes is to reduce costs, latency, and data traffic on the network by implementing data storage and processing nodes closer to the source, that is, to IoT devices (see Fig. 1(16) and 9(17)).

EdgeNode. A computing node or device located at the network edge of an IoT system. Edge nodes can be: IoT devices (sensors, tags, actuators) or IoT gateways. Therefore, they can have varied hardware capabilities, being capable of storing and processing data on the same device, that is, at the network edge. Furthermore, these nodes can be located close to or embedded in physical entities to provide them with detection, processing, storage, or actuation capabilities (see Fig. 1(17) and 9(18)).

Resource. Represents a software or hardware component used by the cloud, fog, or edge nodes to detect, collect, process, and store data on physical entities and act on these entities based on digital instructions. The resources can be specialized in: network interfaces, APIs, applications, middlewares, services, databases, or configuration files (see Fig. 1(14), 9(15), and 10(1)).

Network. Represents an infrastructure that supports the communication and exchange of data between a set of digital entities. The metamodel supports various communication technologies (e.g., Ethernet, Wi-Fi, Bluetooth, Zigbee, Ultra-Wideband, RFID) and communication protocols (*metaclass Protocol*) to interconnect heterogeneous IoT nodes with diverse hardware capabilities. Communication protocols, according to their functions, are organized in layers. For example, at the application layer level, there are several useful protocols for different IoT scenarios, such as CoAP (Constrained Application Protocol), MQTT (Message Queue Telemetry Transport), and HTTP/REST (Hypertext Transfer Protocol / Representational State Transfer) (see Fig. 1(9, 23), 9(10, 24), and 10(5, 15)).

DataFlow. Represents with a high level of abstraction the trajectory of the data and how digital entities (nodes or resources) interact within a process of data collection, transfer, processing, and storage. In this sense, a dataflow describes how the raw data collected by IoT devices are transported through the different computing nodes and resources until stored (at different levels of aggregation) in the corresponding columns and tables of a database. The *executionTimeInterval* and *flowExecutionTime* properties define the frequency and time of dataflow execution, respectively. The communication type supported by a dataflow can be: i) *synchronous*, uses a request/response model, where the sending - receiving of data is carried out in real-time between a source digital entity (client) that makes the request and a destination digital entity (server) that receives the request; and ii) *asynchronous*, uses an event-based data publication/subscription model, that is, the data sent by the issuing digital entities (publishers) are published through topics in an intermediate entity (Broker) so that the receiving digital entities (subscribers) can read them later. A dataflow comprises a set of ordered and interconnected links (*Link metaclass*). Each link allows communication and data exchange between two digital entities. The link types supported by the metamodel are: *API-IoTDevice*, *App-API*, *Service-API*, *App-Service*, *App-Broker*, *Service-Broker*, *Service-Service*, and *Service-Database*. The *previousLink* relationship defines the order of the links that make up a dataflow. Furthermore, a dataflow contains one or more data mapping rules (*DataMappingRule metaclass*) to map data sources (source) to storage locations (destination) of a dataflow; whereas links define how the data will be transported between the source and the destination (traceability). Data mapping rules can specialize in metaclasses: i) *PropertyToDataColumn*, associates an entity property with a table column; and ii) *DataColumnToDataColumn*, associates two data columns, a source column to which aggregation operations are applied, and its result is stored in a destination column. The diversity of configurations and combinations of links between digital entities that a dataflow can support provides flexibility and extensibility to the metamodel. Thus, in addition to data collection and aggregation flows, the metamodel could support report generation flows for decision-making and actuation flows. However, in this paper, only the first two types of dataflows are evaluated, leaving the others to be addressed in future work (see Fig. 1(38-42), 10(22-30), and 11(2-8)).

B. MONITOR-IoT GRAPHICAL DESIGNER

Monitor-IoT provides a graphical design tool (high-level visual modeling language) that simplifies and facilitates the creation and maintenance of multi-layer monitoring architecture models for IoT systems conforming to the domain vocabulary (metamodel) presented in the previous subsection. The graphical designer has been built in Obeo Designer Community Edition and Eclipse Sirius [23] to take advantage

of the modeling technologies included in EMF (Eclipse Modeling Framework) and GMF (Graphical Modeling Framework). In this way, Monitor-IoT supports creating a modeling workbench comprising a set of editors (diagrams, tables, trees) whose structure and behavior are determined by the metamodel. Fig. 2 shows the graphical notation (concrete syntax) for each Monitor-IoT metaclass.

Through Monitor-IoT's visual modeling interface, developers can create architecture models, providing only high-level specifications about the digital entities (e.g., sensors, APIs, applications, services, brokers, databases) and interrelationships between these (dataflows) required at the edge, fog, and cloud layers to support the monitoring processes of an IoT system; without worrying about how these entities and dataflows should be implemented or coded at a low level.

In turn, Monitor-IoT creates a serialized monitoring architecture model in XMI so that a computer can interpret it. Hence, the specifications included in these models can be processed by transformation algorithms to automatically generate the software resources of an IoT system in charge of supporting the monitoring operations (e.g., APIs, applications, services, brokers, topics, databases).

In particular, the interrelationships (links) between the digital entities that are part of the dataflows can be used to automatically generate the code that supports the communication interfaces and data exchange between the software resources. For example, a monitoring architecture model may contain a dataflow that relates to the following digital entities: i) an API that obtains data from a CO sensor; ii) a software application on a gateway that receives the data from the API and sends them through a web service to a fog node; and iii) a web service that stores the monitored data in a database of the fog node (see Fig. 5). The specifications of this dataflow can be used to automatically include, as part of the programming logic of the gateway application, the code to call the API of the CO sensor and obtain the monitored data. Also, the code to consume the web service and transfer the monitored data to the fog node (synchronous communication) can be added below. Likewise, in the body of the web service, the code to store the monitored data and its metadata in a specific table of the fog node database can be encapsulated.

This development approach on which Monitor-IoT is based allows developers to focus their efforts on defining the problem while freeing them from tasks such as coding APIs, applications, or web services, creating topics in a broker, or constructing the logical storage structure of a database (tables and columns).

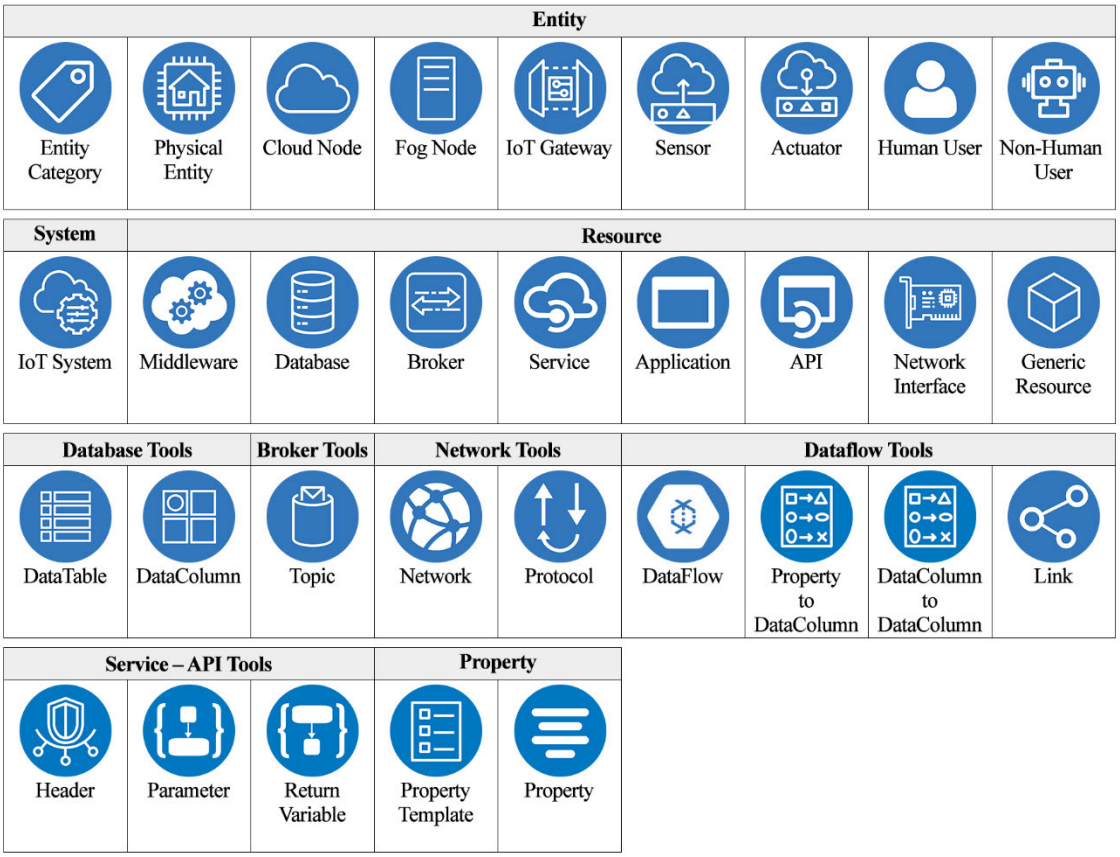

FIGURE 2. Monitor-IoT DSL graphical notation.

Consequently, the monitoring architecture models obtained from Monitor-IoT constitute an abstraction layer that acts as a bridge between the monitoring requirements of an IoT system and its implementation details. Furthermore, this abstraction layer represented by models and its alignment with the ISO/IEC 30141:2018 standard allows Monitor-IoT to be used in any subdomain and technology platform of IoT. As the models generated by Monitor-IoT are high-level domain specifications, the target technology platform can be quickly changed by implementing new transformation algorithms; however, the specifications of the IoT monitoring architecture models remain fixed.

Currently, a transformation engine (synchronizer) is being implemented in Node.js to automatically create the software components of the IoT system (e.g., APIs, applications, services, brokers, topics, databases) from the monitoring architecture model. These components allow collecting the data from sensors, performing aggregation operations on these data, and storing them in the respective databases. Although medium-term, the focus will be on an engine capable of maintaining a causal relationship between an IoT system and its architecture model at runtime so that any change in the IoT system is reflected in the model and vice-versa, without having to halt its operation.

Finally, the source files of the Monitor-IoT tool and a video that explains the steps for its configuration and execution can be obtained from the tool's website: https://sites.google.com/uazuay.edu.ec/dslmonitoriot.

IV. DESIGN PROCESS OF IoT MONITORING ARCHITECTURES USING MONITOR-IoT

The proposed process to model the multi-layer monitoring architecture of an IoT system using Monitor-IoT consists of three main activities: i) Identification of subsystems, entities, properties, and dataflows; ii) Modeling of subsystems, entities, and properties; and iii) Modeling of dataflows. In turn, each activity is divided into several specific tasks. In Fig. 3, the process diagram based on SPEM 2.0 (Software & Systems Process Engineering Meta-Model Specification) [37] is presented, which details the execution order of the activities and tasks defined to design IoT monitoring architectures with the support of Monitor-IoT, as well as the input and output artifacts that are used during the process. The activities and tasks included in the process diagram are described below.

1) IDENTIFICATION OF SUBSYSTEMS, ENTITIES, PROPERTIES, AND DATAFLOWS

This activity receives as input the specification of functional and non-functional requirements of an IoT system to analyze it and prepare the documentation required by the Monitor-IoT DSL to carry out the subsequent monitoring architecture modeling activities. This activity is divided into four tasks:

1.1) ***Decomposition of the IoT system.*** Defines the hierarchical structure of the IoT system, breaking it down into subsystems if they exist. This task aims to promote the creation of IoT monitoring architectures that guarantee high cohesion and low coupling between its parts, and in this way, promote the reuse of the computational and physical capacities of all or part of the IoT system in other applications or scenarios. The data to document for each system or subsystem are: acronym (identifier), name, description, and application domain to which it belongs.

1.2) ***Identification of entities.*** Defines the physical or digital entities to monitor. In addition, it determines the digital entities (computing nodes and IoT devices), the hardware/software resources (e.g., network interfaces, middlewares, databases, services, APIs, applications), and the communication networks necessary to support the monitoring processes at the cloud, fog, and edge layers. The entities are organized and grouped into subsystems if they exist. The main data documented by entity are: name, description, metaclass, category, and subsystem to which it belongs.

1.3) ***Identification of entity properties.*** Describes the properties to be monitored for each entity. The data to be specified are: name and definition of the property, type of property (telemetry, state, metadata), data type and measurement unit of the property, if the property allows the unique identification of the entity, and whether the property value is assigned at design or run time.

1.4) ***Identification of dataflows.*** In this task, the dataflows are specified, describing how the digital entities interact with each other to support data collection, transport, processing, and storage. The data to be documented are: description and type of dataflow (collection or aggregation), type of communication that the dataflow uses (synchronous or asynchronous), and frequency and time of dataflow execution.

2) MODELING OF SUBSYSTEMS, ENTITIES, AND PROPERTIES

First modeling activity that uses the Monitor-IoT DSL. In general terms, this activity focuses on modeling the hierarchical structure between the system and its subsystems, designing the structure between physical and digital entities (computing nodes and their resources), and configuring the properties to monitor for each entity. This activity is divided into seven tasks, several of which run in parallel:

2.1) ***Modeling of subsystems.*** Designs the hierarchical structure between the IoT system and its subsystems if it exists. In addition, this task groups the entities into subsystems, being able a physical or digital entity to be reused by several subsystems.

2.2) ***Modeling of physical entities.*** Designs the hierarchical structure between the physical entities identified in the first activity.

2.3) ***Modeling of computing nodes at the edge, fog, and cloud layers.*** Adds to the model the cloud nodes, fog nodes, gateways, and/or IoT devices (sensors, tags, actuators) specified in the first activity and necessary to support the monitoring processes of the IoT system. The computing nodes can be contained within the physical

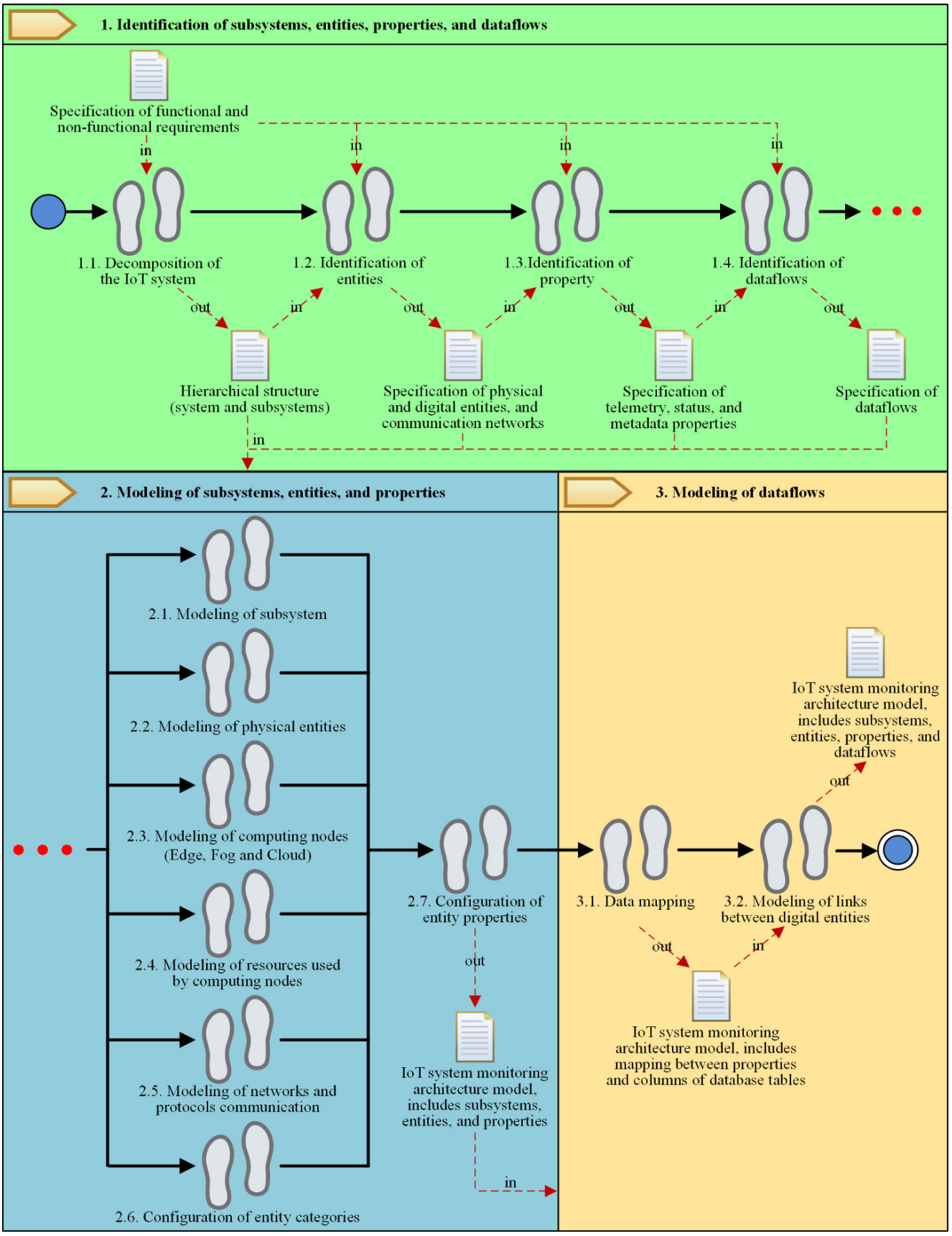

FIGURE 3. Process diagram to design monitoring architectures for IoT systems with Monitor-IoT.

entities to provide them with detection, processing, storage, or actuation capabilities.

2.4) ***Modeling of resources used by computing nodes.*** Adds to the model the hardware and software resources required by the computing nodes specified in the first activity.

2.5) ***Modeling of communication networks and protocols.*** Adds to the model the networks and protocols that support synchronous and asynchronous communication between digital entities. In addition, it establishes the networks and protocols used by each of the software resources (e.g., middlewares, databases).

2.6) ***Configuration of entity categories.*** This task groups entities of the same type (they share common properties to be monitored) into categories.

2.7) ***Configuration of entity properties.*** In this task, the properties to be monitored are created, together with their templates containing the properties' general specifications. The information contained in the templates can be reused later in the configuration of common properties for other entities.

3) MODELING OF DATAFLOWS

Once the physical and digital entities have been incorporated into the model, this last activity designs how digital entities interact to support dataflows, for which it is divided into two tasks:

3.1) ***Data mapping.*** In this task, the data sources (source) are mapped to the logical storage locations (destination) for each dataflow.

3.2) ***Modeling of links between digital entities.*** For each dataflow, an ordered sequence of links is designed between the digital entities. This sequence determines the trajectory of data and how computing nodes and resources interact in a dataflow to transport the data from the source to its storage location.

V. ILLUSTRATIVE IoT SCENARIO

An IoT scenario within the Ambient Assisted Living subdomain (see Fig. 4) is presented in this section to demonstrate the usefulness of the Monitor-IoT DSL. The proposed scenario aims to design a monitoring architecture for an emergency management system aimed at older people who live alone in their homes. The system comprises two subsystems: i) an environmental control subsystem responsible for monitoring temperature, carbon monoxide (CO), and smoke in the house environment; and ii) a healthcare subsystem to monitor the heart rate of the older adult. For this, the subsystems require various computing devices and nodes at the edge, fog, and cloud layers, which are described in the following subsections.

A. EDGE LAYER

The IoT platform includes different types of sensors to support monitoring requirements. A temperature sensor has been installed in the older adult's bedroom (see Fig. 4(a)), while a CO and smoke sensor has been installed in the kitchen (see Fig. 4(b)). These sensors collect data from the environment every 10 minutes. In addition, the user has a smartwatch (Samsung Galaxy Watch3) that includes a heart rate sensor, which collects data every 30 minutes (see Fig. 4(c)).

The IoT platform also includes the use of two gateways. On the one hand, a microcontroller (Raspberry Pi 3) that executes a software application (environmentController) that calls two APIs (getTemperature, getCO&Smoke) to collect the data from the temperature, CO, and smoke sensors via Bluetooth (see Fig. 4(d)). In turn, the software application synchronously sends the collected data to a local server located in the user's home (fog node), consuming a RESTful service provided by the fog node. On the other hand, the smartwatch (Samsung Galaxy Watch3) acts as a gateway that runs an App (healthController) that calls an API (getHeartRate) to retrieve data from the heart rate sensor (see Fig. 4(c)). Additionally, the App asynchronously sends the collected data to the fog node, using a broker as an intermediary.

The gateways and the fog node communicate over a Wi-Fi local area network using the application protocols: HTTP (synchronous communication) and MQTT (asynchronous communication).

B. FOG LAYER

As mentioned, this layer contains a server (HP Proliant Microserver) located in the user's home with local storage and processing capabilities (see Fig. 4(e)). In addition, this fog node includes the following resources: i) a PostgreSQL database management system whose objective is to store the low-level raw data from all the sensors of the IoT platform; ii) an Eclipse Mosquitto broker that implements an asynchronous data communication model (publication/subscription model) between the smartwatch and the fog node; and iii) an application server created using the Node.js Express framework that contains RESTful services for data exchange, aggregation, and storage.

The fog node provides the following services: i) a RESTful service (saveEnvironmentData) that is consumed directly by the Raspberry Pi 3 software application in order to send the data obtained from the temperature, CO, and smoke sensors for storage in the PostgreSQL database (supports synchronous data collection flows); ii) a RESTful service (saveHealthData) in charge of obtaining the user's heart rate data from the broker and then storing them in the PostgreSQL database (supports asynchronous data collection flows); and iii) a RESTful service (saveHealthSummaryDaily) that calculates the daily average, maximum, and minimum values of the user's heart rate and stores them in the cloud node (supports data aggregation flows).

C. CLOUD LAYER

The proposed IoT scenario requires a cloud node implemented on the Google Cloud Platform (see Fig. 4(f)). This node has a PostgreSQL database management system to store

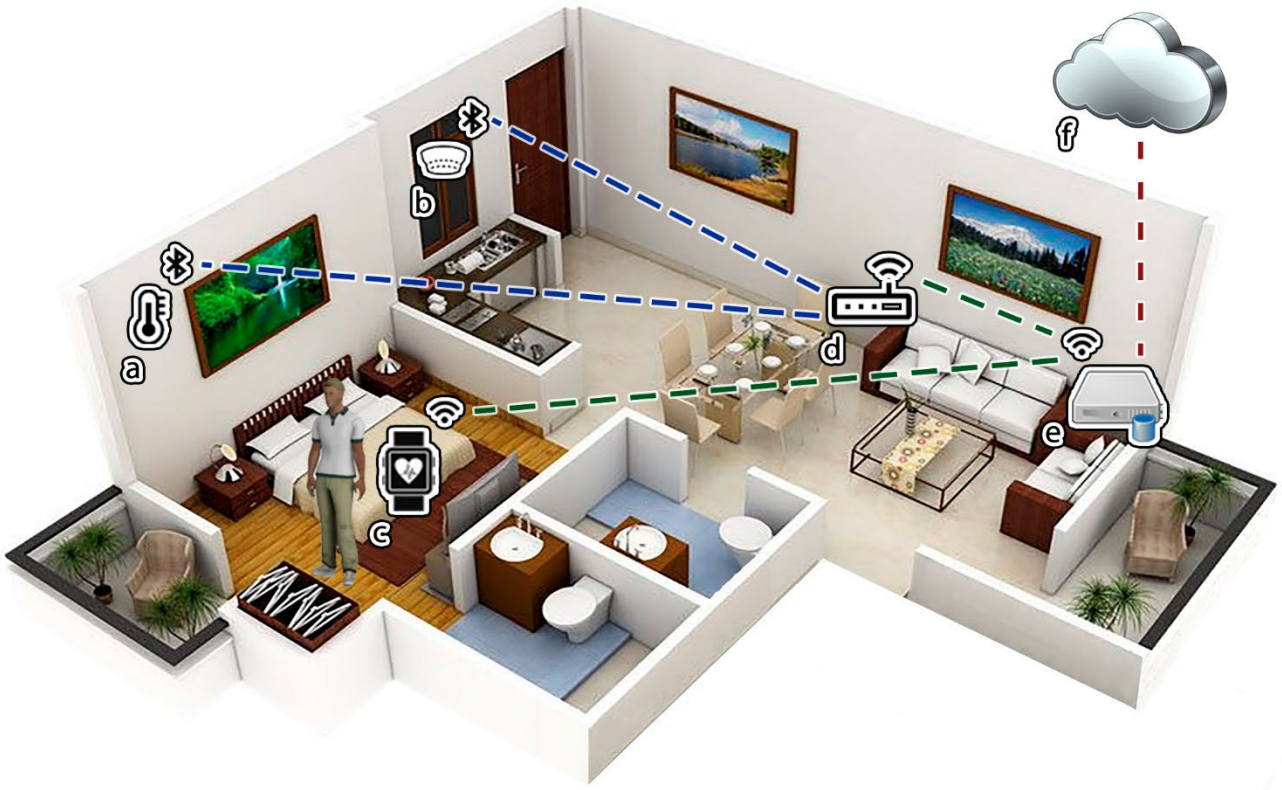

FIGURE 4. Illustrative IoT scenario.

the data resulting from the aggregation operations executed in the fog node. The fog and cloud nodes communicate over the Internet using the HTTP application protocol.

D. RESULTING ARCHITECTURE MODEL

Once the IoT scenario has been described, Fig. 5 presents the multi-layer monitoring architecture designed in the Monitor-IoT DSL to solve the monitoring requirements proposed in this scenario.

In Fig. 5(1), the physical and digital entities are modeled, and the dataflows between them. Physical entities are organized hierarchically; for example, the house contains the bedroom and the kitchen, while the older adult contains (wears) a smartwatch. The digital entities are distributed and contained in the physical entities. The temperature, CO, and smoke sensors are distributed in the rooms of the house. The gateway (Raspberry Pi 3) and the fog node (HP Proliant Microserver) are contained locally in the house. The heart rate sensor and a gateway are included in the smartwatch (Samsung Galaxy Watch3). The cloud node (Google Cloud Platform), being an external entity, is modeled outside the house. Concerning software resources, these are modeled within the computing nodes following the requirements established in the IoT scenario. The controller applications and sensor APIs running on the gateways. The broker (Eclipse Mosquitto), the application server (Node.js Express), and their RESTful services running on the fog node. While the databases (PostgreSQL) are located in both the fog and cloud node.

In Fig. 5(2), the hierarchical structure of the system is modeled, decomposing it into subsystems. This structure enables entities to be grouped, reused, and shared between subsystems.

In Fig. 5(3), the networks and protocols that support communication between digital entities and their resources are modeled. For this case, three networks have been designed. The former is a proximity network that uses Bluetooth to communicate between the sensors (temperature, CO, and smoke) and the respective gateway (Raspberry Pi 3). Then, a Wi-Fi LAN network that uses the MQTT (asynchronous communication) protocol between the smartwatch and the fog node and the HTTP (synchronous communication) protocol between the Raspberry Pi 3 and the fog node. Finally, a WAN network that uses the HTTP protocol between the fog and cloud nodes.

The resolution of the illustrative IoT scenario has shown that the Monitor-IoT tool is expressive enough to model the monitoring architecture of an IoT system with a high degree of abstraction. Specifically, how physical and digital entities (nodes and resources) relate and interact through dataflows to support the data collection, transportation, aggregation, and storage processes.

VI. EMPIRICAL EVALUATION OF MONITOR-IoT

In this section, the empirical evaluation of the Monitor-IoT DSL is presented. It is aimed to know the user perception about the ease of use, usefulness, and intention of future

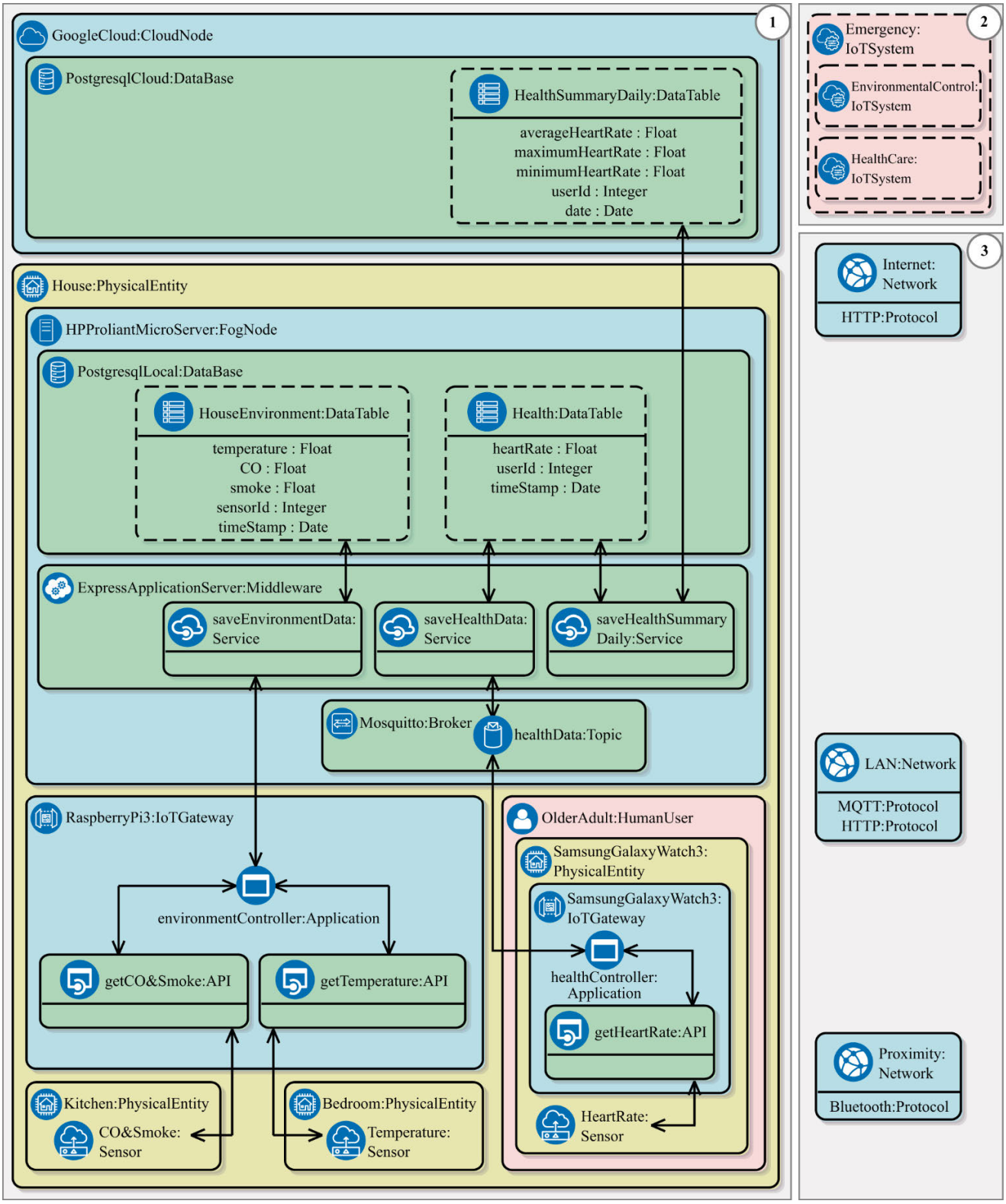

FIGURE 5. Multi-layer monitoring architecture for the IoT system specified in the illustrative scenario.

use of this tool to design monitoring architectures for IoT systems.

For the evaluation of Monitor-IoT, a quasi-experiment was used as an empirical strategy [38]. It includes the participation of undergraduate students from the last semesters of the Systems Engineering College at the University of Azuay and the University of Cuenca. Participants used Monitor-IoT to design a monitoring architecture for the healthcare subsystem

described in the previous section as part of the illustrative IoT scenario. In addition, the Technology Acceptance Model (TAM) proposed by Davis [36] was used. It integrates user perception variables as a mechanism to predict the adoption and use of Monitor-IoT in future practices.

The following subsections present the adaptation of the TAM for the evaluation of Monitor-IoT. Also, the design, execution, and analysis of the results of the quasi-experiment.

A. ADAPTATION OF THE TAM FOR THE EVALUATION OF MONITOR-IOT

The TAM is a widely used model to predict the acceptance and use of new technology. This model is based on the following primary constructors: i) *Perceived Ease of Use* (*E*), the degree to which a user believes that using a target system will require less effort to perform their tasks; ii) *Perceived Usefulness* (*U*), the subjective probability of the user that using a target system will improve their performance at work; iii) *Attitude Towards Using* (*A*), user's desire to use the target system (both E and U predict A); iv) *Behavioral Intention to Use (BI)*, a measure of resistance to perform a specific behavior, (U and A influence BI); and v) *Use*, represents the current use of the system, which is predicted by BI [36].

It is necessary to operationalize the TAM for its application in evaluating the Monitor-IoT DSL. The operationalization consists of defining the TAM constructors based on the relevant perception variables for Monitor-IoT and adapting a measurement instrument (questionnaire) to evaluate these variables in the Monitor-IoT context.

Table 3 presents the questionnaire adapted to measure the perception variables of the TAM, made up of twelve questions, distributed as follows: i) five questions to measure perceived ease of use (E); ii) four questions to evaluate the perceived usefulness (U); iii) two questions focused on the intention of use (BI); and iv) an open-ended question to collect suggestions about the Monitor-IoT DSL.

In turn, the questions were formulated using the Likert scale (5 points), assuming three as a neutral value, from which the Monitor-IoT tool can be accepted or rejected according to the perception variables.

According to the adaptation of the TAM, the probability that the Monitor-IoT DSL will be accepted in practice can be predicted by testing the following hypotheses:

- $H1_0$: The Monitor-IoT DSL is perceived as difficult to use, $H1_0 = \neg H1_1$.
- $H2_0$: The Monitor-IoT DSL is not perceived as useful, $H2_0 = \neg H2_1$.
- $H3_0$: There is no intention to use the Monitor-IoT DSL in the future, $H3_0 = \neg H3_1$.
- $H4_0$: The perceived usefulness is not determined by the perceived ease of use, $H4_0 = \neg H4_1$.
- $H5_0$: The intention to use is not determined by the perceived ease of use, $H5_0 = \neg H5_1$.
- $H6_0$: The intention to use is not determined by the perceived usefulness, $H6_0 = \neg H6_1$.

TABLE 3. Questionnaire to measure perception.

No.	Question
E1	Monitor-IoT graphical designer is intuitive and easy to understand?
E2	The graphical notation (representation of the components) used by Monitor-IoT is intuitive and easy to understand?
E3	Monitor-IoT graphical designer is easy to use?
E4	The activities and tasks to follow to design a monitoring architecture for IoT systems with the support of Monitor-IoT are easy to understand?
E5	The activities and tasks to follow to design a monitoring architecture for IoT systems with the support of Monitor-IoT are easy to implement?
U1	The use of Monitor-IoT could reduce the time and effort required to develop, deploy, and maintain IoT systems?
U2	Monitor-IoT is useful to support the development, deployment, and maintenance activities of IoT systems?
U3	The use of Monitor-IoT could improve your performance in developing, deploying, and maintaining IoT systems?
U4	The monitoring architecture model obtained from Monitor-IoT is useful to represent how the monitoring requirements of an IoT system should be implemented?
BI1	If you need to design a monitoring architecture for IoT systems in the future, would you use Monitor-IoT?
BI2	Would you recommend using Monitor-IoT to design monitoring architectures for IoT systems?
OQ1	Do you have any suggestions regarding the Monitor-IoT tool? (open-ended question)

These hypotheses implicitly contain the TAM variables (E, U, and BI). Finally, In Fig. 6, the proposed model to evaluate the user perception about the adoption and use of Monitor-IoT is presented.

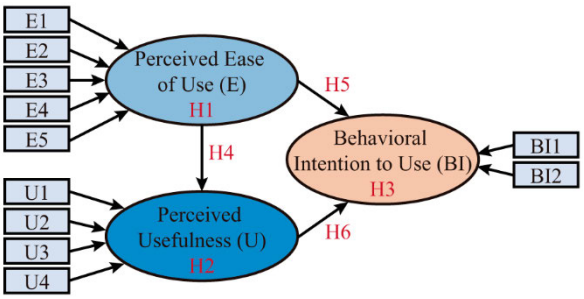

FIGURE 6. Model adapted from the TAM to evaluate the user perception of the use of Monitor-IoT.

B. DESIGN OF THE QUASI-EXPERIMENT

The quasi-experiment was designed based on the experimental process proposed by Wohlin *et al.* [39]. As a first step, the *quasi-experiment goal* was formulated according to the Goal-Question Metric (GQM) paradigm proposed by Basili *et al.* [40]. In addition, the *research questions* were defined. Tables 4 and 5 present the goal and the research questions, respectively.

Research questions were evaluated through hypothesis testing. In particular, the first question was studied through the hypotheses: $H1_0$, $H2_0$, and $H4_0$. Moreover, the second

TABLE 4. Goal-question-metric for the quasi-experiment.

Evaluate:	The Monitor-IoT DSL in the design of monitoring architectures for IoT systems.
With the purpose of:	Know the user perception regarding the ease of use, usefulness, and intention of future use of the Monitor-IoT DSL.
From the viewpoint of:	Researcher.
In the context of:	A group of undergraduate students from the last semesters of the Systems Engineering College.

TABLE 5. Research questions.

No.	Question
RQ1	Is the Monitor-IoT DSL perceived as easy to use and useful?
RQ2	Is there the intention to use the Monitor-IoT DSL in the future? If so, is the intention to use the result of the user perception?

question was analyzed through the hypotheses: $H3_0$, $H5_0$, and $H6_0$.

Regarding the *quasi-experiment context*, it was determined by: i) the Monitor-IoT DSL tool; ii) the IoT scenario used in the Monitor-IoT evaluation, which is within the Ambient Assisted Living subdomain and consists of the design of the monitoring architecture for the healthcare subsystem, described in Section 5; and iii) the participation of 33 undergraduate students from the last semesters of the Systems Engineering College at the University of Azuay and the University of Cuenca, who have solid knowledge about Software Architectures, Model-Driven Development, and Internet of Things.

The *quasi-experiment tasks* were defined systematically and orderly, following the proposed process to design monitoring architectures for IoT systems with the support of the Monitor-IoT DSL presented in section 4.

The *quasi-experiment dependent variables* to evaluate Monitor-IoT were the perception variables defined in the TAM (E, U, and BI). These variables allowed testing the hypotheses and were measured using the questionnaire presented in Table 3.

The *quasi-experimental material* consisted of the documentation and tools necessary to carry out the experimental tasks and a questionnaire (Google Forms) to measure the user perception once the experiment was carried out. The supporting documentation included: i) a presentation used by the researchers to train the participants, which describes the main concepts related to the Monitor-IoT DSL, the base software (Obeo Designer Community Edition) used by Monitor-IoT, the Monitor-IoT graphical designer's work environment, and a training exercise that contemplates the design of the monitoring architecture of the environmental control subsystem, described in the illustrative IoT scenario (section 5); ii) a video tutorial with the description of the steps for the installation and configuration of the Monitor-IoT tool; iii) a video tutorial on the training exercise; iii) a guide with the description of the process to design monitoring architectures for IoT systems using Monitor-IoT; and iv) a worksheet with the description of the experimental exercise (design of the monitoring architecture for the healthcare subsystem), which includes several fields to collect metrics on the activities carried out by the participants and the products obtained. The experimental material was distributed to the participants through a website.

C. EXECUTION OF THE QUASI-EXPERIMENT

The quasi-experiment was carried out virtually on the Zoom platform with 33 undergraduate students of the Systems Engineering College, 21 belonging to the University of Cuenca and 12 to the University of Azuay. In addition, it lasted four hours and was divided into two sessions.

The first session focused on the training of the participants. This session lasted an hour and a half. It included the following activities: i) presentation of the main concepts related to Monitor-IoT (e.g., Internet of Things, systems architecture, domain-specific languages); ii) explanation and support in the installation and configuration process of Monitor-IoT, iii) review of the work environment (design area, tool palette, properties window) of Monitor-IoT, how to instantiate the elements of the DSL, define their properties, and interconnect them with each other; and iv) resolution of the training exercise in order to demonstrate how Monitor-IoT should be used to design a monitoring architecture according to the process presented in section 4.

The second session was oriented to the development of the experimental exercise by the participants. This session lasted two and a half hours, distributed as follows: i) 20 minutes to present the practical case (healthcare subsystem) to be solved with the Monitor-IoT tool; ii) 120 minutes to design the monitoring architecture of the experimental exercise; however, it should be noted that the participants were allowed to finish the experiment, even if time was running out in order to mitigate the ceiling effect; and iii) 10 minutes to answer a questionnaire in Google Forms with the questions presented in Table 3, as well as to send the resolved worksheet of the experimental exercise and the file with the architecture model obtained in XMI format. The researchers responsible for experimenting clarified any doubts or questions throughout the development of the experimental exercise.

D. RESULTS OF THE QUASI-EXPERIMENT

Descriptive statistics, box plots, and tests with the support of the SPSS V20 tool (with $\alpha = 0.05$) were used to analyze the data collected from the experimental exercise. The data were analyzed to contrast the established hypotheses that include the TAM perception variables (E, U, and BI). For the acceptance or rejection of the hypotheses, the significance levels suggested by Moody [41] were used.

Before analyzing user perception, the monitoring architecture models for the health subsystem designed by the participants were evaluated concerning the requirements established in the experimental scenario. The evaluation

results determined that all 33 participants satisfactorily fulfilled the experimental requirements and tasks. Therefore, all the cases were qualified as valid for further analysis.

1) ANALYSIS OF THE USER PERCEPTION

Fig. 7 presents the box plots for each perception variable. As a common denominator, it is evident that the mean of each variable is greater than the neutral value (3) of the Likert scale. In turn, the box plot of the intention to use variable (BI) shows a single outlier belonging to the participant with identifier 1. This participant has been excluded to avoid distortions in the subsequent statistical analysis.

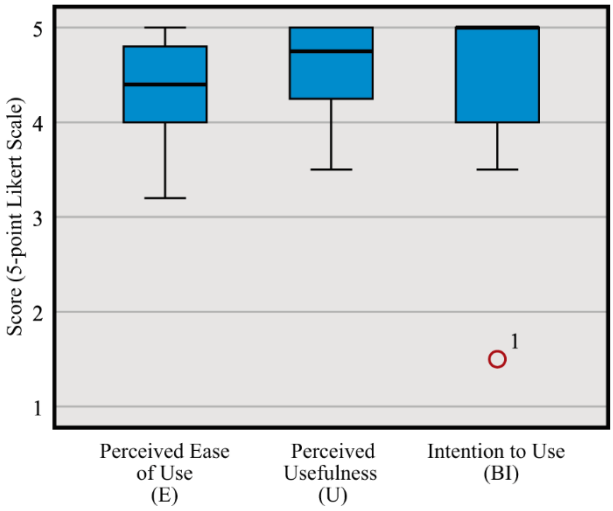

FIGURE 7. Box plots of the perception variables. (E, U, and BI.)

As part of the analysis, the Shapiro-Wilk test was applied to check if the perception variables have a normal distribution and determine the type of test to be used to contrast the null hypotheses $H1_0$, $H2_0$, and $H3_0$. The penultimate column of Table 6 shows the results of the Shapiro-Wilk test, where it can be observed that all the variables studied (E, U, and BI) do not have a normal distribution ($p < 0.05$). Therefore, the Wilcoxon one-tailed one-sample test (nonparametric test) with a test value equal to three (neutral value of the Likert scale) has been selected to contrast the hypotheses.

The results of the Wilcoxon one-tailed one-sample test for each variable are presented in the last column of Table 6, which shows that all the perception variables (E, U, and BI) have a very high level of significance ($p < 0.001$). Thus, these results allow rejecting the null hypotheses $H1_0$, $H2_0$, and $H3_0$, which means that, the participants perceive the Monitor-IoT DSL as an easy-to-use and useful tool, and they intend to use this tool to design monitoring architectures for IoT systems in future projects.

2) ANALYSIS OF CAUSAL RELATIONSHIPS

At this point, the structural part of the TAM is validated in terms of the causal relationships between the variables perceived ease of use (E), perceived usefulness (U), and intention to use (BI). The simple linear regression statistical method has been used to test the hypotheses $H4_0$, $H5_0$, and $H6_0$. These hypotheses are defined as causal relationships between the variables indicated above. The results obtained for each of the causal relationships are described below:

- ***Perceived Ease of Use vs. Perceived Usefulness.*** A linear regression model was constructed with E as the independent variable and U as the dependent variable, as presented in (1). The resulting model shows a high significance ($p < 0.01$), and the coefficient of determination (R^2) indicates that the variable E explains 33.1% of the variance of U (see Table 7). Consequently, the results allow rejecting the null hypothesis $H4_0$ and accepting its alternative hypothesis, which means that the perceived usefulness (U) is partly determined by the perceived ease of use (E).

$$U = 2.124 + 0.547 * E \tag{1}$$

- ***Perceived Ease of Use vs. Intention to Use.*** The linear regression model is presented in (2), where E is the independent variable and BI is the dependent variable. The resulting model has a very high significance ($p < 0.001$), and the coefficient of determination (R^2) indicates that the variable E explains 50.1% of the variance of BI (see Table 8). Hence, the results allow rejecting the null hypothesis $H5_0$ and accepting its alternative hypothesis, which corroborates that the intention of use (BI) is partly determined by the perceived ease of use (E).

$$BI = 1.186 + 0.763 * E \tag{2}$$

- ***Perceived Usefulness vs. Intention to Use.*** The linear regression model is presented in (3), where U is the independent variable and BI is the dependent variable. The model obtained has a very high significance ($p < 0.001$), and the coefficient of determination (R^2) indicates that the variable U explains 48.8% of the variance of BI (see Table 9). Therefore, the results allow rejecting the null hypothesis $H6_0$ and accepting its alternative hypothesis, which means that the intention of use (BI) is partly determined by the perceived Usefulness (U).

$$BI = 0.964 + 0.791 * U \tag{3}$$

3) DISCUSSION

The quantitative evaluation showed that the experiment participants found the Monitor-IoT tool easy to use and useful. It even evidenced that they would be willing to use Monitor-IoT to design IoT monitoring architectures in future projects. Likewise, this evaluation demonstrated that the participants' perception of ease of use and usefulness partly determines their intentions to use Monitor-IoT in the future. However, there may be other variables that influence the participants' intention to use. Fig. 8 presents a synthesis of the results obtained in the evaluation of Monitor-IoT with the support of the TAM.

TABLE 6. Results for perception variables.

Variable	Min.	Max.	Mean	Std. Dev.	Shapiro-Wilk test p-value	Wilcoxon one-tailed one-sample test p-value
E	3.2000	5.0000	4.4250	0.5099	**0.013 < 0.05**	**0.000 < 0.001**
U	3.5000	5.0000	4.5469	0.4854	**0.000 < 0.05**	**0.000 < 0.001**
BI	3.5000	5.0000	4.5625	0.5499	**0.000 < 0.05**	**0.000 < 0.001**

TABLE 7. Simple linear regression model between perceived ease of use and perceived usefulness.

	Coefficient	Err. Std.	t.	Sig. (p)	R	R^2
Constant	2.124	0.633	3.355	0.002		
E	0.547	0.142	3.850	**0.001 < 0.01**	**0.575**	**0.331**

TABLE 8. Simple linear regression model between perceived ease of use and intension of use.

	Coefficient	Err. Std.	t.	Sig. (p)	R	R^2
Constant	1.186	0.620	1.914	0.336		
E	0.763	0.139	5.483	**0.000 < 0.001**	**0.707**	**0.501**

TABLE 9. Simple linear regression model between perceived usefulness of use and intension of use.

	Coefficient	Err. Std.	t.	Sig. (p)	R	R^2
Constant	0.964	0.677	1.425	0.795		
U	0.791	0.148	5.348	**0.000 < 0.001**	**0.699**	**0.488**

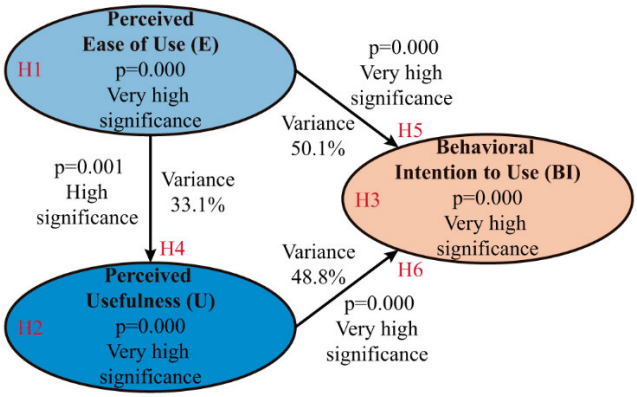

FIGURE 8. Synthesis of the results of the application of the TAM to Monitor-IoT.

Regarding the qualitative evaluation (open-ended question), a remarkable aspect is that several participants consider it essential to have a methodological process associated with Monitor-IoT since it was very helpful in fulfilling the tasks and requirements of the experiment. In turn, the participants' suggestions have allowed the refinement and continuous improvement of the Monitor-IoT tool and its methodological process.

The IoT scenarios used during the pilot test and training and experimental exercises demonstrated that Monitor-IoT has a sufficiently comprehensive domain language to represent the key concepts that the diverse IoT monitoring environments may require. In this sense, the evaluation of Monitor-IoT contemplated: i) different types of physical entities and properties to be monitored; ii) diversity of computing nodes, together with their essential resources (APIs, applications, services, databases and their structure, brokers, topics, network interface) at the edge, fog, and cloud layers; and iii) communication flows, both synchronous and asynchronous. Likewise, the evaluation evidenced a saving of time and effort when modeling the IoT monitoring scenarios architectures due to using a language that allows the definition of property templates for IoT entities of the same type. In general, the empirical evaluation has confirmed that Monitor-IoT satisfactorily covers the limitations of the analyzed DSLs and the challenges exposed in the related work.

Finally, unlike related work on DSL solutions that focus on the edge, fog, and cloud layers, this research work designed and executed a rigorous empirical evaluation to validate the usability of Monitor-IoT based on the end-user perception.

E. THREATS TO THE VALIDITY OF THE QUASI-EXPERIMENT

This subsection presents the main threats to the validity of the Monitor-IoT evaluation results that were mitigated during the development of the quasi-experiment. These threats are organized according to the classification proposed by Cook and Campbell [42].

1) INTERNAL VALIDITY

In this study, internal validity is relevant because causal relationships are examined. The main threats to internal validity were the participants' experience and the biases produced by the researchers, the experimental material, and the Monitor-IoT tool. Regarding participants' experience, a training phase

was prepared, focused on developing a practical case that provides participants with a clear understanding of the design process of monitoring architectures for IoT systems with the support of the Monitor-IoT DSL. In turn, the biases produced by the researchers, the experimental material, and the Monitor-IoT tool were reduced through a pilot experiment. Here, expert researchers in the area participated during the development of the experiment in detecting and correcting errors related to ambiguities in the experimental material and usability problems of the Monitor-IoT tool.

2) EXTERNAL VALIDITY

The representativeness and generalizability of the evaluation results may be affected by the evaluation design and the context of the selected participants. On the one hand, to mitigate the threats related to the evaluation design, a sufficiently complete IoT scenario was proposed, including different types of entities (physical and digital) and types of communication (synchronous, asynchronous). In addition, the IoT scenario included multiple subdomains (e.g., smart home, health) throughout the pilot test, training case study, and experimental exercise. On the other hand, to reduce the threats related to the context of the participants, the experiment had the participation of undergraduate students from the last semesters of the Systems Engineering College of several universities. These students could be considered the next generation of professionals [43] as it has been shown that, under certain conditions, there is not a significant difference between this type of students and professionals [44], [45]. Therefore, the ability of these students to design architectures for IoT systems is comparable to junior professionals, even more so when only students who had passed courses on Software Architectures, Model-Driven Development, and Internet of Things have been selected.

3) CONSTRUCT VALIDITY

The main threat was the reliability of the questionnaire, for which the Cronbach's alpha reliability test [46] was applied to each group of questions related to the perception variables. The results of the tests were greater than the minimum accepted threshold ($\alpha = 0.70$), obtaining a Cronbach's alpha for perceived ease of use (E) of 0.808, perceived usefulness (U) of 0.778, and intention to use (BI) of 0.855, which shows high internal consistency.

4) CONCLUSION VALIDITY

The sample size of 33 participants was one of the main problems since it can affect the causality between the different variables. However, the results were encouraging since all the participants successfully carried out the requirements and tasks proposed in the experiment. In turn, as mentioned, a homogeneous group of participants has been selected to control the risk of variation due to individual differences that may grow due to treatment.

VII. CONCLUSION AND FUTURE WORK

The bibliographic review has shown that despite the existence of several proposals on DSLs oriented to the domain of IoT, some problems persist in this field, mainly related to the functional limitations that DSLs present. This situation is due to metamodels that provide a limited language to represent the possible digital entities and variants of dataflows (synchronous and asynchronous) that an IoT system may require during the monitoring processes at the edge, fog, and cloud layers. Furthermore, these metamodels are not aligned with a standardized reference architecture for IoT; even ambiguities have been observed between the solutions regarding the meaning and use of various concepts.

In this sense, the main contribution of this work consists of providing a domain-specific language (Monitor-IoT) to facilitate and streamline the design of multi-layer monitoring architectures for IoT systems with a high level of abstraction, expressiveness, and flexibility. For this, the Monitor-IoT DSL comprises a graphical designer and a metamodel aligned with the reference architecture for IoT ISO/IEC 30141:2018, which also incorporates a set of common and relevant entities among the related work studied. Furthermore, it ensures the semantic coherence of the metamodel and provides a language capable of modeling a wide variety of digital entities and dataflows between them to support the data collection, transportation, processing, and storage processes.

The empirical evaluation of Monitor-IoT, through the application of a quasi-experiment and the Technology Acceptance Model (TAM), demonstrated the favorable intention of the experiment participants to design monitoring architectures for IoT systems in future projects with the support of Monitor-IoT; since they also consider Monitor-IoT as an easy-to-use and useful tool for the development of IoT systems.

This work is a first step towards developing a methodology and support infrastructure based on MDE and MAPE-K to build self-aware and self-adaptive IoT systems. In future work, it is proposed to build a synchronization engine to support at runtime the causal relationship between the IoT system and its architecture model designed in Monitor-IoT. In addition, it is necessary to continue with the evaluation of Monitor-IoT through the execution of experiments or case studies in other IoT subdomains with the participation of industry professionals, as well as expanding the scope of Monitor-IoT to support the report generation dataflows for decision-making and the actuation dataflows of an IoT system.

APPENDIX
ADDITIONAL METACLASSES OF THE MONITOR-IoT METAMODEL

This appendix presents the description of the additional metaclasses of the Monitor-IoT metamodel. In turn, it includes Fig. 9, 10, 11, and 12 with all the metaclasses, relationships, attributes, and enumerations of the metamodel.

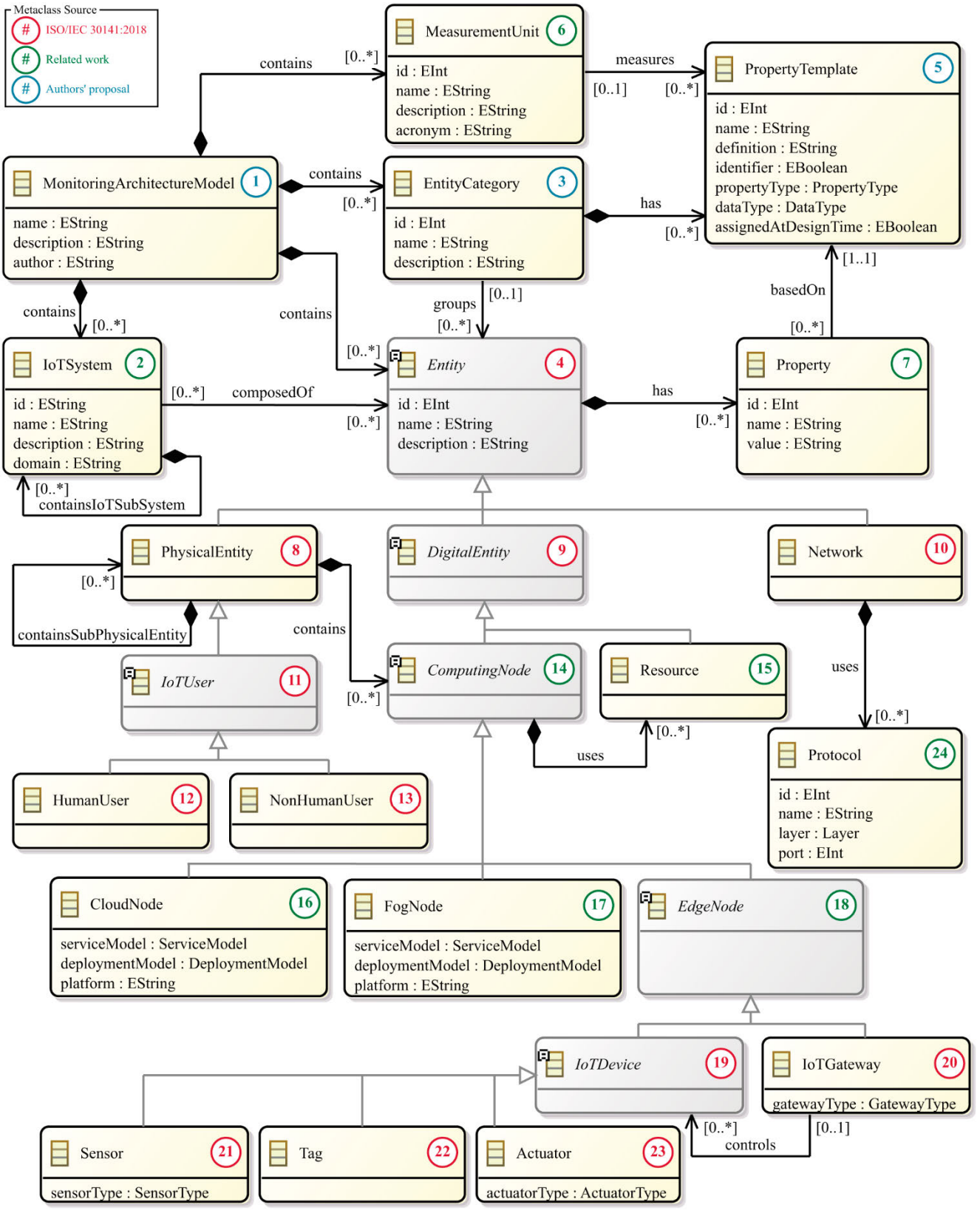

FIGURE 9. Monitor-IoT metamodel (Part 1), includes metaclasses and relationships to represent subsystems, physical entities, digital entities (computing nodes, IoT devices), and communication networks.

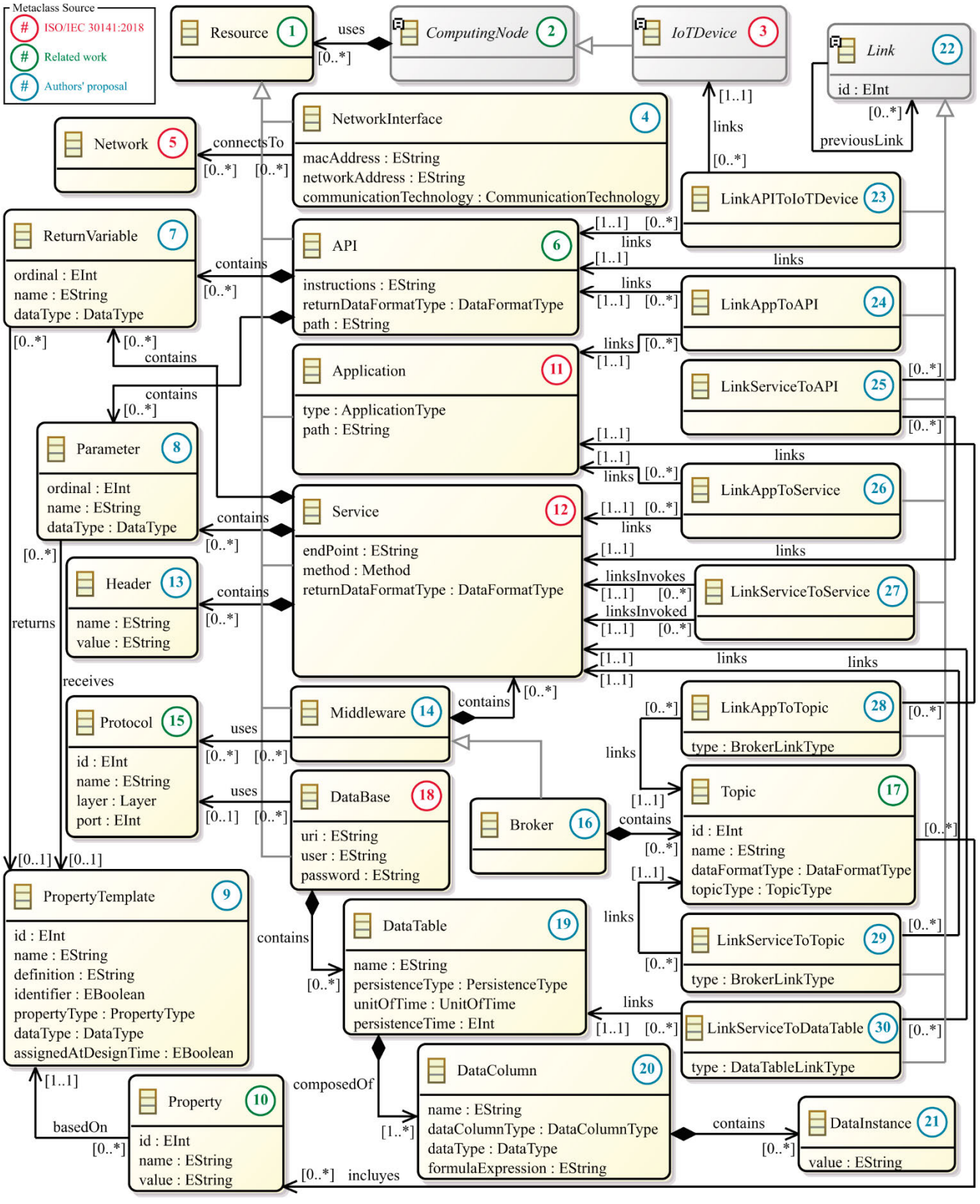

FIGURE 10. Monitor-IoT metamodel (Part 2), includes metaclasses and relationships to represent computing resources and the interactions (links) between them.

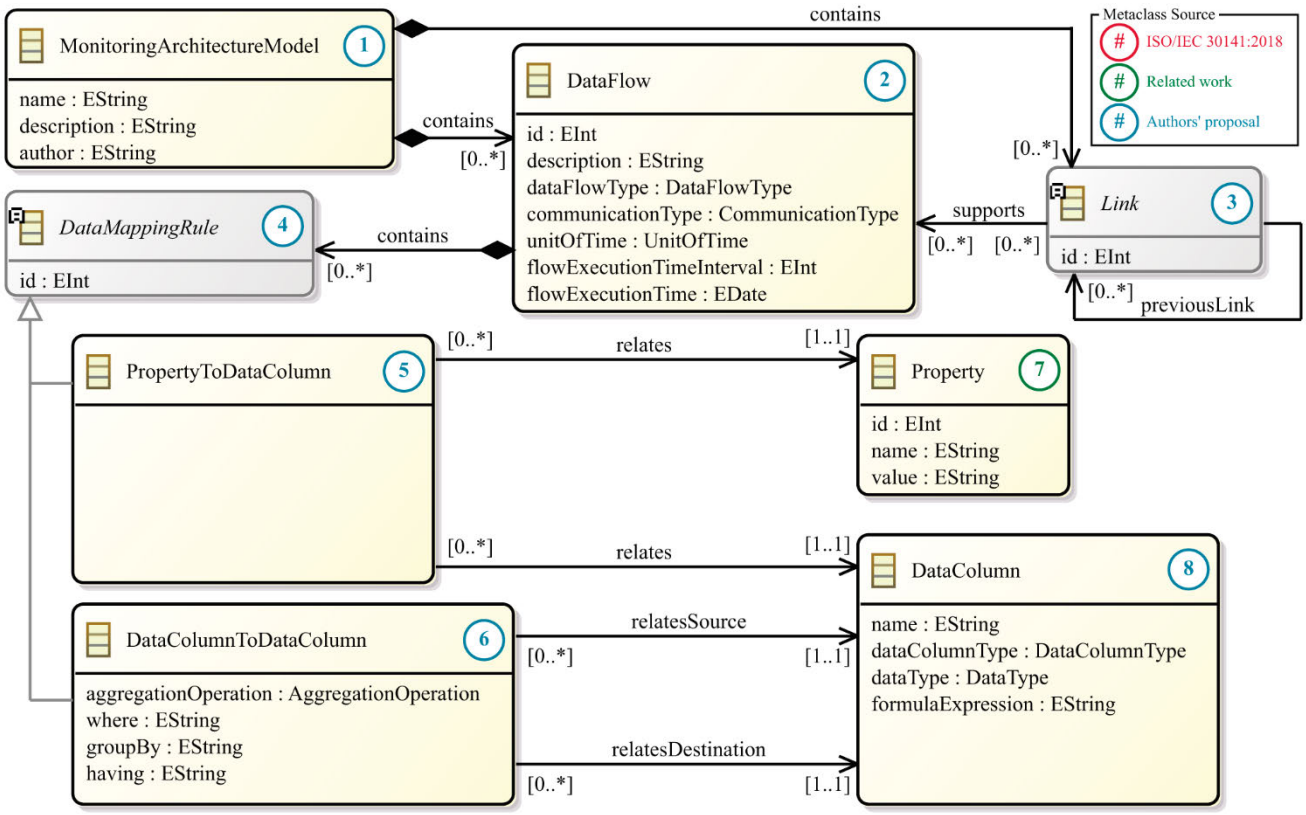

FIGURE 11. Monitor-IoT metamodel (Part 3), includes metaclasses and relationships to represent dataflows, links between digital entities, and mapping rules.

EntityCategory. Defines a grouping of entities of the same type to specify a single time (using a template) general information about the common properties and APIs of these entities. This abstraction reduces the workload when modeling the IoT monitoring architecture. For example, it can create a category that represents all the rooms to be monitored in a house (see Fig. 1(3) and 9(3)).

PropertyTemplate. Defines a template with the general specifications of a common property for all entities in a category. The metamodel supports three types of properties: i) *metadata*, which contains information that rarely changes about an entity (e.g., serial number, brand, model, manufacture date) and can be determined at design time; ii) *telemetry*, data about a physical entity or the environment collected through sensors (e.g., temperature or humidity in a room, blood pressure of a patient); and iii) *status*, information that describes the current status of a digital entity (e.g., battery power level of a sensor, amount of memory or processing used by an edge node, bandwidth used by a network). A property can uniquely identify an entity within an IoT system (*identifier attribute).* In addition, it has a data type (*dataType attribute*) and a measurement unit (*MeasurementUnit metaclass)* (see Fig. 1(5), 9(5, 6), and 10(9)).

IoTDevice. A device located at the network edge that associates physical entities in the real world with other digital entities in an IoT system. IoT devices interact through a network, and depending on their hardware and power capabilities, they can communicate directly or through an IoT gateway with digital entities (cloud or fog nodes) located in local or wide area networks. Sensors, tags, and actuators are a type of IoT device (see Fig. 1(18), 9(19), and 10(3)).

Sensor. An IoT device that perceives or monitors specific properties of a physical entity and transforms them into digital data to be transmitted over a network. For example, a temperature sensor that sends measurements to a smartphone application to monitor the temperature of a room (see Fig. 1(20) and 9(21)).

Tag. An object that can be applied to or incorporated into a physical entity to identify or trace it. The identification process is carried out by sensors (readers) that read the tags (see Fig. 1(21) and 9(22)).

Actuator. An IoT device that acts on (changes) the properties of a physical entity based on digital instructions; that is, it allows modifying the state of a physical entity. For example, a mobile application that connects via Bluetooth with an air conditioner (actuator) to control the temperature of a room (see Fig. 1(22) and 9(23)).

IoTGateway. A computing node located at the network edge. It acts as an intermediary to interconnect IoT devices located in proximity networks with other digital entities (cloud or fog nodes) located in local or wide area networks. In turn, a gateway can offer a wide range of functionalities,

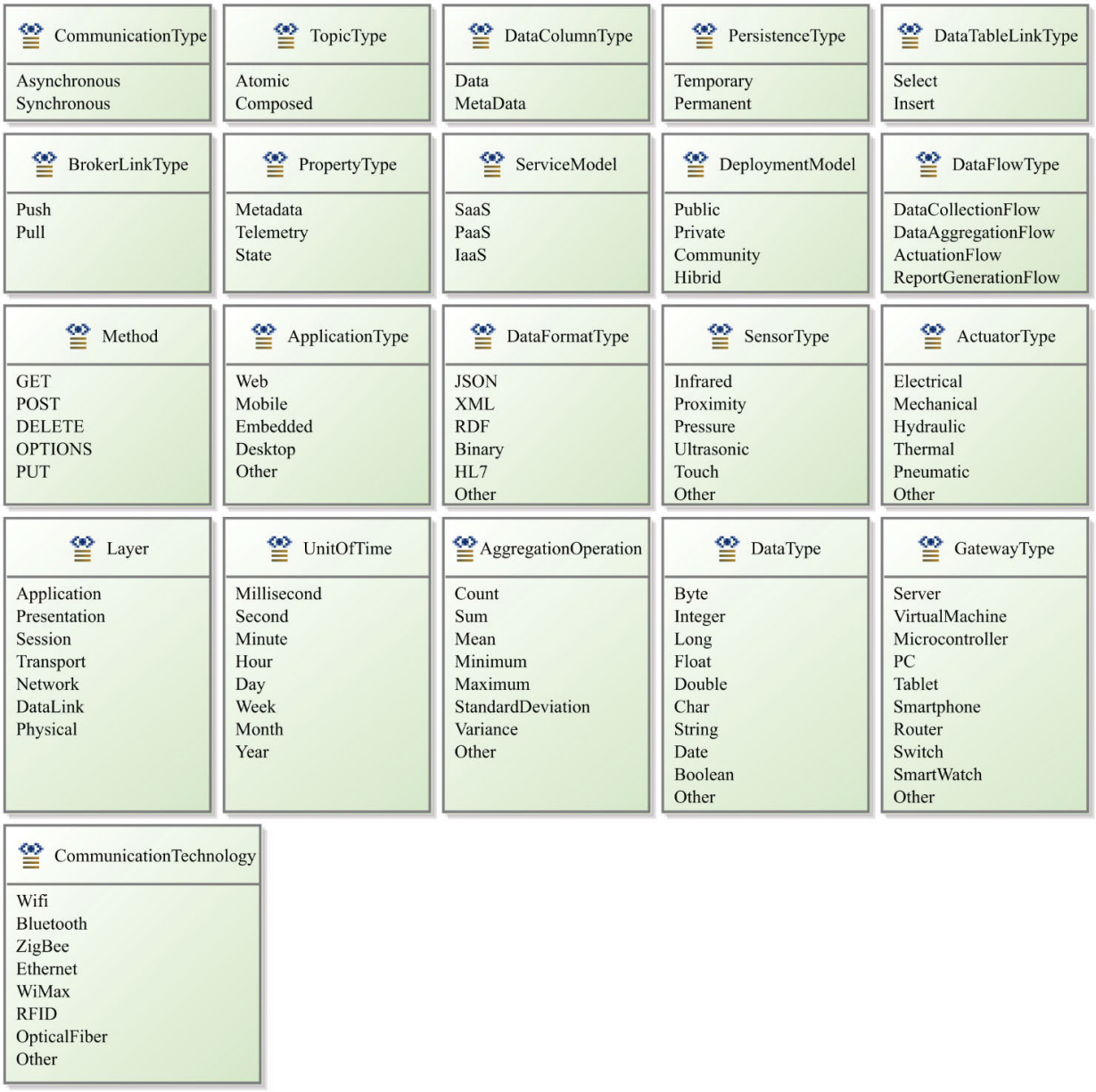

FIGURE 12. Monitor-IoT metamodel (Part 4), includes the enumerations used by the attributes of the metaclasses.

such as i) agent for configuring, managing, and controlling IoT devices; ii) data cleaning, filtering, and routing; that is, it determines what data should be sent to the fog or cloud nodes and what data should be processed locally; iii) storage of data collected by associated IoT devices, as a solution to problems related to intermittent or saturated communication networks; iv) local data processing for fast and effective control of actuators based on input data from sensors; v) protocol conversion and address mapping; and vi) implementation of security at the edge device level. Furthermore, the gateways can be standalone equipment or be integrated with other IoT detection and control devices (sensors, actuators). Therefore, they can be implemented in: servers, personal computers, microcontrollers, smartphones, routers, virtual machines, among others (see Fig. 1(19) and 9(20)).

NetworkInterface. Specifies an interface that connects a computing node or IoT device to a communication network to share resources and data. A network interface uses a communication technology (e.g., Wi-Fi, Ethernet, Bluetooth) and has some kind of network address that consists of a unique node identifier in its own right (see Fig. 1(24) and 10(4)).

API (Application Programming Interface). A software component that encapsulates functionalities to manage IoT devices (sensors, tags, actuators) with a high degree of abstraction. Specifically, an API can retrieve raw data of the physical entities properties or act on (change) them. The metamodel allows specifying the input data (*Parameter metaclass*) and the output information (*ReturnVariable metaclass*) of an API, as well as encapsulating its instruction

sequence (*instructions attribute*) (see Fig. 1(34, 35, 37) and 10(6-8)).

Application. A software component that offers a set of functionalities to perform a specific task. An application can be: desktop, web, mobile, or embedded, and it can run on the cloud, fog, or edge nodes. Furthermore, an application can call APIs to access the data observed by the sensors or control the actuators, consume services to exchange data with other digital entities (synchronous communication), and publish or receive data through a broker (asynchronous communication). Finally, an application can have a visual interface to interact with human users. For example, in an IoT system for a smart home, a user using an application installed on a smartphone can monitor or adjust room temperature (see Fig. 1(36) and 10(11)).

Service. A software component that, through an open and standardized interface, provides a set of functional capabilities to exchange, store, and process data, hiding the heterogeneity of the entities involved. The services are contained in middlewares and can be accessed through endpoints. A service can interact with other services to provide high-level functionality. Also, services can receive input data (*Parameter metaclass*) and produce output information (*ReturnVariable metaclass*) represented in different formats (e.g., JSON, XML, RDF). Additionally, services can contain headers to send metadata associated with the service request or response (*Header metaclass*) (see Fig. 1(32-35) and 10(7, 8, 12, 13)).

Middleware. A software component that supports and provides services for synchronous and asynchronous data management (exchange, processing, storage). This resource facilitates interoperability between heterogeneous digital entities located in a distributed manner in the edge, fog, and cloud layers of an IoT platform, through the implementation of services that hide the particular implementation details of each digital entity. In addition, the metamodel allows specifying one or more protocols and communication ports through which the middleware offers its services (see Fig. 1(29) and 10(14)).

Broker. It is an explicit specialization of a middleware that supports messaging or asynchronous data transfer between the digital entities of an IoT system. A broker is responsible for receiving, organizing, and publishing the data (payload) of the IoT entities (publishers) in topics and distributing them to all the IoT entities (subscribers) that have subscribed to said topics. This data transmission model is recommended for the Internet of Things since IoT devices often have power, processing, and bandwidth limitations. The properties data of an entity are organized and identified by the *Topic metaclass* in a Broker. Topics can be atomic if they include data for a single property or composite if they include data for multiple properties. Furthermore, they can use different formats for data exchange (e.g., JSON, XML, RDF, HL7) (see Fig. 1(30, 31) and 10(16, 17)).

DataBase. A software component that stores the data of the monitored properties in an IoT system. The data to be stored can be raw (obtained directly from IoT devices) or result from aggregation operations on the collected data. The metamodel allows specifying the connection string of the databases used. A database contains tables (*DataTable metaclass*) that can be temporary or permanent. For the first case, a time interval must be defined to manage the persistence of the data. In turn, a table is made up of columns (*DataColumn metaclass*) that can be of two types: i) *data*, which contains the data collected or processed for a property; and ii) *metadata*, which contains information that describes the data collected or processed; for example, the IoT device that collected the data, when the data were collected (timestamp), where the data were collected, or the quality of the data collected (precision, completeness). Finally, the *DataInstance metaclass* represents the column values, that is, the values of the monitored properties and their associated metadata (see Fig. 1(25-28), 10(18-21), and 11(8)).

Finally, Fig. 12 presents the enumerations of the Monitor-IoT metamodel. Note that some of the enumerations used by the attributes of the metaclasses are not exhaustive (e.g., SensorType, ActuatorType, AggregationOperation, CommunicationTechnology); that is, they do not reflect all the possible values. Hence, new values may be required for the enumerations depending on the needs of each IoT scenario.

ACKNOWLEDGMENT

The authors would like to thank the Universidad del Azuay and the Universidad de Cuenca for their continued support, and David C. Siddons for revising the English translation.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Jun. 2010, doi: 10.1016/j.comnet.2010.05.010.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015, doi: 10.1109/COMST.2015.2444095.

[3] R. van der Meulen. *Gartner Says 8.4 Billion Connected 'Things' Will be in Use in 2017, up 31 Percent From 2016*. Gartner, Egham, U.K. Accessed: Feb. 7, 2017. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016

[4] E. Estopace. *IDC Forecasts Connected IoT Devices to Generate 79.4 ZB of Data in 2025*. FutureIoT. Accessed: Jun. 22, 2019. [Online]. Available: https://futureiot.tech/idc-forecasts-connected-iot-devices-to-generate-79-4zb-of-data-in-2025/

[5] X. Chen, A. Li, X. Zeng, W. Guo, and G. Huang, "Runtime model based approach to IoT application development," *Frontiers Comput. Sci.*, vol. 9, no. 4, pp. 540–553, 2015, doi: 10.1007/s11704-015-4362-0.

[6] C. M. Sosa-Reyna, E. Tello-Leal, and D. Lara-Alabazares, "Methodology for the model-driven development of service oriented IoT applications," *J. Syst. Archit.*, vol. 90, pp. 15–22, Oct. 2018, doi: 10.1016/j.sysarc.2018.08.008.

[7] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: Vision, applications and research challenges," *Ad Hoc Netw.*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012, doi: 10.1016/j.adhoc.2012.02.016.

[8] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proc. Future Softw. Eng. (FOSE)*, Minneapolis, MN, USA, May 2007, pp. 37–54, doi: 10.1109/FOSE.2007.14.

[9] D. C. Schmidt, "Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006, doi: 10.1109/MC.2006.58.

[10] T. Eterovic, E. Kaljic, D. Donko, A. Salihbegovic, and S. Ribic, "An Internet of Things visual domain specific modeling language based on UML," in *Proc. XXV Int. Conf. Inf., Commun. Autom. Technol. (ICAT)*, Sarajevo, Bosnia Herzegovina, Oct. 2015, pp. 1–5, doi: 10.1109/ICAT.2015.7340537.
[11] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic, "Design of a domain specific language and IDE for Internet of Things applications," in *Proc. 38th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, Opatija, Croatia, May 2015, pp. 996–1001, doi: 10.1109/MIPRO.2015.7160420.
[12] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa, "ThingML: A language and code generation framework for heterogeneous targets," in *Proc. ACM/IEEE 19th Int. Conf. Model Driven Eng. Lang. Syst.*, New York, NY, USA, Oct. 2016, pp. 125–135, doi: 10.1145/2976767.2976812.
[13] F. Pramudianto, M. Eisenhauer, C. A. Kamienski, D. Sadok, and E. J. Souto, "Connecting the Internet of Things rapidly through a model driven approach," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, Reston, VA, USA, Dec. 2016, pp. 135–140, doi: 10.1109/WF-IoT.2016.7845416.
[14] B. Negash, T. Westerlund, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "DoS-IL: A domain specific Internet of Things language for resource constrained devices," *Proc. Comput. Sci.*, vol. 109, pp. 416–423, May 2017, doi: 10.1016/j.procs.2017.05.411.
[15] B. Costa, P. F. Pires, and F. C. Delicato, "Modeling SOA-based IoT applications with SoaML4IoT," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, Limerick, Ireland, Apr. 2019, pp. 496–501, doi: 10.1109/WF-IoT.2019.8767218.
[16] D. Alulema, J. Criado, and L. Iribarne, "A model-driven approach for the integration of hardware nodes in the IoT," in *New Knowledge in Information Systems and Technologies (WorldCIST)* (Advances in Intelligent Systems and Computing), vol. 930, Á. Rocha, H. Adeli, L. Reis, and S. Costanzo, Eds. Cham, Switzerland: Springer, 2019, pp. 801–811, doi: 10.1007/978-3-030-16181-1_75.
[17] C. G. García, D. Meana-Llorián, V. García-Díaz, A. C. Jiménez, and J. P. Anzola, "Midgar: Creation of a graphic domain-specific language to generate smart objects for Internet of Things scenarios using model-driven engineering," *IEEE Access*, vol. 8, pp. 141872–141894, 2020, doi: 10.1109/ACCESS.2020.3012503.
[18] J. A. Barriga, P. J. Clemente, E. Sosa-Sánchez, and Á. E. Prieto, "SimulateIoT: Domain specific language to design, code generation and execute IoT simulation environments," *IEEE Access*, vol. 9, pp. 92531–92552, 2021, doi: 10.1109/ACCESS.2021.3092528.
[19] I. Alfonso, K. Garcés, H. Castro, and J. Cabot, "Modeling self-adaptative IoT architectures," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Fukuoka, Japan, Oct. 2021, pp. 761–766, doi: 10.1109/MODELS-C53483.2021.00122.
[20] *Internet of Things (IoT)—Reference Architecture*, Standard ISO/IEC 30141:2018, ISO, Aug. 2018. [Online]. Available: https://www.iso.org/standard/65695.html
[21] J. Mineraud, O. Mazhelisb, X. Suc, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Commun.*, vols. 89–90, pp. 5–16, Sep. 2016, doi: 10.1016/j.comcom.2016.03.015.
[22] (Jun. 2005). *An Architectural Blueprint for Autonomic Computing*. IBM, White Paper. [Online]. Available: https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
[23] Obeo Designer Community. (11.4). *OBEO—Eclipse Sirius*. Accessed: Feb. 7, 2022. [Online]. Available: https://www.obeodesigner.com/en/product/sirius
[24] Eclipse Foundation. *Eclipse Modeling Framework (EMF) Documentation*. Eclipse. Accessed: Feb. 7, 2022. [Online]. Available: https://www.eclipse.org/modeling/emf/docs/
[25] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," *J. Comput. Commun.*, vol. 3, pp. 164–173, Jan. 2015, doi: 10.4236/jcc.2015.35021.
[26] *Internet of Things (IoT) Preliminary Report 2014*, ISO, Geneva, Switzerland, Standard ISO/IEC JTC 1, 2014. [Online]. Available: https://www.iso.org/files/live/sites/isoorg/files/developing_standards/docs/en/internet_of_things_report-jtc1.pdf
[27] P. Fremantle, "A reference architecture for the Internet of Things," WSO2, Mountain View, CA, USA, White Paper, Version 0.9.0, Oct. 2015. [Online]. Available: https://docs.huihoo.com/wso2/wso2-whitepaper-a-reference-architecture-for-the-internet-of-things.pdf
[28] (2014). *The Internet of Things Reference Model*. Cisco, White Paper. [Online]. Available: https://es.scribd.com/document/440866359/IoT-Reference-Model-White-Paper-June-4-2014-pdf
[29] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST SP 500-325, Mar. 2018. [Online]. Available: https://doi.org/10.6028/NIST.SP.500-325
[30] B. Selic, "MDA manifestations," *Eur. J. Inform. Prof.*, vol. 9, no. 2, pp. 12–16, Apr. 2008.
[31] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Boston, MA, USA: Pearson, 2008.
[32] M. Bauer, N. Bui, J. D. Loof, and C. Magerkurth, "IoT reference model," in *Enabling Things to Talk*, A. Bassi, Eds. Berlin, Germany: Springer, 2013, pp. 113–162, doi: 10.1007/978-3-642-40403-0_7.
[33] S. Haller, A. Serbanati, M. Bauer, and F. Carrez, "A domain model for the Internet of Things," in *Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber, Phys. Social Comput.*, Beijing, China, Aug. 2013, pp. 411–417, doi: 10.1109/GreenCom-iThings-CPSCom.2013.87.
[34] P. Patel, A. Pathak, T. Teixeira, and V. Issarny, "Towards application development for the Internet of Things," in *Proc. 8th Middleware Doctoral Symp. (MDS)*, Lisbon, Portugal, 2011, pp. 1–6, doi: 10.1145/2093190.2093195.
[35] P. Patel and D. Cassou, "Enabling high-level application development for the Internet of Things," *J. Syst. Softw.*, vol. 103, pp. 62–84, May 2015, doi: 10.1016/j.jss.2015.01.027.
[36] F. D. Davis, "A technology acceptance model for empirically testing new end-user information systems: Theory and results," Ph.D. dissertation, Dept. Sloan School Manage., MIT, Boston, MA, USA, 1985. [Online]. Available: https://dspace.mit.edu/handle/1721.1/15192
[37] (Apr. 2008). *Software & Systems Process Engineering Meta Model Specification Version 2.0, Object Management Group—OMG*. [Online]. Available: https://www.omg.org/spec/SPEM/2.0/PDF
[38] P. Cedillo, E. Insfran, S. Abrahao, and J. Vanderdonckt, "Empirical evaluation of a method for monitoring cloud services based on models at runtime," *IEEE Access*, vol. 9, pp. 55898–55919, 2021, doi: 10.1109/ACCESS.2021.3071417.
[39] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
[40] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed. New York, NY, USA: Wiley 1994, pp. 528–532.
[41] D. L. Moody, "Dealing with complexity: A practical method for representing large entity relationship models," Ph.D. dissertation, Dept. Inf. Syst., Univ. Melbourne, Melbourne, VIC, Australia, 2001.
[42] T. D. Cook and D. T. Campbell, *Quasi-experimentation: Design & Analysis Issues in Field Settings*. Boston, MA, USA: Houghton Mifflin, 1979.
[43] B. A. Kitchenham, S. L. Pfleeger, and L. M. Pickard, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002, doi: 10.1109/TSE.2002.1027796.
[44] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—A comparative study of students and professionals in lead-time impact assessment," *Empirical Softw. Eng.*, vol. 5, no. 3, pp. 201–214, Nov. 2000, doi: 10.1023/A:1026586415054.
[45] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 456–473, Jul. 1999, doi: 10.1109/32.799939.
[46] L. J. Cronbach, "Coefficient alpha and the internal structure of tests," *Psychometrika*, vol. 16, no. 3, pp. 297–334, Sep. 1951, doi: 10.1007/BF02310555.

LENIN ERAZO-GARZÓN received the degree in systems engineering from the Universidad de Cuenca, Ecuador, in 1999, and two master's degrees in business administration and in software engineering and computer systems from Universidad del Azuay and Universidad Internacional de La Rioja, Spain, in 2012 and 2018, respectively. He is currently pursuing the Ph.D. degree in computer sciences with the Universidad Nacional de La Plata, Argentina. He has been an Associate Professor with the Universidad del Azuay, Ecuador, since 2001. His current research interests include model-driven engineering, the Internet of Things (IoT), self-aware systems, and software quality.

PRISCILA CEDILLO (Member, IEEE) received two master's degrees in telematics and in software engineering, information systems, and formal methods from the Universidad de Cuenca and the Universitat Politècnica de València (UPV), respectively, and the Ph.D. degree in computer science from UPV, in 2017. She obtained a scholarship from the Senescyt for her doctoral studies and from the Fundación Carolina for two postdoctoral research stays. She has been an Associate Professor with the Universidad de Cuenca, Ecuador, since 2009. Her main research interests include model-driven engineering, cloud computing, software quality, and the Internet of Things (IoT).

GUSTAVO ROSSI received the Ph.D. degree from PUC-Rio, Brazil. He is a Full Professor with Universidad Nacional de La Plata, Argentina, and a Researcher with CONICET. His current research interests include web engineering, requirements engineering, and human–computer interaction. He is a member of the Editorial Board of IEEE Internet Computing, IEEE IT Professional, and the *World Wide Web* journal.

JOSÉ MOYANO received degree in the systems engineering from the Universidad de Cuenca, Ecuador, in 2021. He worked for two years as a Research Assistant on the project "Fog computing applied to monitor devices used in assisted living environments; study case: A platform for the elderly" at the Universidad de Cuenca. His interests include developing software for cloud computing and the Internet of Things.