IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 27, 2020, accepted November 15, 2020, date of publication November 30, 2020,
date of current version December 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3041321

A RESTful Rule Management Framework for Internet of Things Applications

FEDERICA PAGANELLI[1,2], (Member, IEEE), GEORGIOS MYLONAS[3,4], AND GIOVANNI CUFFARO[5,6]

[1]Department of Computer Science, University of Pisa, 56126 Pisa, Italy
[2]CNIT, 56127 Pisa, Italy
[3]Computer Technology Institute and Press "Diophantus", 265 04 Patras, Greece
[4]Industrial Systems Institute, "Athena" Research Center, 151 25 Patras, Greece
[5]CNIT, University of Florence Research Unit, 50139 Florence, Italy
[6]Firlab SRL, 50131 Florence, Italy

Corresponding author: Federica Paganelli (federica.paganelli@unipi.it)

This work was supported in part by the Green Awareness in Action (GAIA) Research Project funded by the European Union (Horizon 2020 Research and Innovation Program) under Grant 696029.

ABSTRACT Web technologies are currently regarded as key enabling factors for the Internet of Things (IoT), and substantial effort is being dedicated to bringing sensors and data from the real world to the Web. In addition, rule-based automation mechanisms are expected to play a significant role in the effective integration of the physical world with the virtual world by leveraging a trigger-action paradigm. Although several rule engines are already available, limited effort has been devoted to rule-based solutions that are tailored to the IoT and consider rule configurability and extensibility according to application requirements. In this work, we propose a RESTful rule management framework for IoT applications that satisfies these requirements. The framework is centered around a resource-based graph, which enables the uniform representation of things (e.g., sensors and domain entities) and rules as URI-addressable resources. We describe the design and implementation choices of the main rule management features (rule scheduling, activation and RESTful operations for managing rules at various levels of configurability and extensibility). Finally, we present a case study and performance evaluation results regarding the use of this rule management framework in a set of school buildings that were part of a real-world IoT deployment that was realized within the Horizon 2020 GAIA research project, with the objective of promoting energy-saving behaviors in school communities.

INDEX TERMS Behavior-based energy saving, education, Internet of Things, REST, rule engine, web of things.

I. INTRODUCTION

The Internet of Things (IoT) is currently producing a very large amount of raw data via continuous collection by sensors. In this context, it is widely recognized that effective and efficient techniques are needed for converting this growing amount of data into usable knowledge [1]. However, while substantial research and development efforts have been devoted to device management and IP-based connectivity technologies, less attention has been given to challenges that are related to the development of applications for the emerging ecosystem of IP-enabled devices and objects [2]. In this respect, one of the main challenges is the smooth and seamless integration of the virtual and physical worlds to enable users to manage and access things of interest [3].

Web technologies are considered the main enabling factor for unleashing *"the potential of the Internet of Things by making it accessible and programmable by developers that are not necessarily experts in ubiquitous computing"* [2]. This also leads to the concept of the Web of Things (WoT), which is an area of the IoT that focuses on the challenge of making smart things accessible and manageable through open Web technologies [4], [5]. Indeed, due to the adoption of such standards, the WoT promotes interoperability among systems and the integration of smart things with the impressive amount of information resources and services that are already on the Web [6]. In this context, the Representational State Transfer (REST) architectural style [7] is widely adopted for

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Huei Cheng.

incorporating sensors and, more generally, smart things into the Web [8] since it defines a set of principles for distributed hypermedia application design with scalability, simplicity and loose-coupling requirements in mind.

Although RESTful Web exposure of physical objects satisfies the basic requirements for making representations of things accessible to clients through Web standards, in many cases, mechanisms are needed for automating reactive behaviors upon the occurrence of specified conditions (e.g., one could be interested in being notified if the average temperature at a location exceeds a specified threshold). In this context, rule-based approaches are expected to play a significant role in the IoT, since, as stated by Perera *et al.* in [9] *"they are the easiest and simplest way to model human thinking and reasoning in machines"*. More recently, Ghiani *et al.* [10] argued that *"adopting a trigger-action paradigm represents a promising approach because of its compact and intuitive structure. . ."* and *". . .various IoT-based application domains can benefit from such trigger-action paradigm"*. Typically, state-of-the-art approaches for implementing rule-based mechanisms in IoT include: *i)* ad hoc IoT applications that are customized for specified application objectives (such as energy efficiency [11]), *ii)* rule-based reasoning techniques that are integrated with an IoT middleware (e.g., [3], [12]), and *iii)* general-purpose rule engines, such as Jess [13].

To the best of our knowledge, related studies do not address the design of rule-based solutions that are tailored to the IoT and consider the requirements of rule configurability and extensibility to satisfy application requirements. We identified such requirements based on our practical experiences in European research projects in the IoT domain, namely, SmartSantander [14] and, more recently, OrganiCity [15] and Green Awareness in Action (GAIA) [16]. In these projects, a similar approach is pursued for the provision by IoT middleware of uniform web exposure of sensor data that are collected from multiple sites. These data are made available to endless applications. Nonetheless, we also observed that in many application scenarios similar rules should be replicated and slightly customized depending on the physical site from where the data originate. For instance, a school building manager may want to activate a rule to automatically turn off artificial lights when the natural light is sufficient for guaranteeing visual comfort in a classroom. It is likely that this rule must be replicated in various areas of a building. Replication may require users to manage low level technological details, such as identifiers of sensors and actuators that cover the area of interest, and to set suitable threshold values (the visual comfort requirements differ amomg rooms, such as classrooms, halls, and libraries, depending on the usage). In some cases, similar rules must be slightly customized to comply with national regulations (e.g., penalties for low power factor values). Moreover, in practical scenarios, end users with basic or no technological skills may be required to create, modify and customize rules during system operation, for example for handling new regulations, new processes or the availability of newly added sensors.

Recently, end user development techniques have been investigated in the IoT and context-aware computing domain, that leverage a rule-based approach to pursue similar objectives ([10], [17], [18]). Most related studies focus on end user interaction and provide complete solutions that range from data acquisition to end user authoring tools; however, in many cases, it is not clear whether inner components (e.g., rule-based components) are provided as modular solutions that can be integrated with third-party IoT middleware and end user applications or whether rule management features are exposed through programmatic interfaces.

In this work, we propose a RESTful rule management framework that enables the creation and modification of rules and the management of their activation (firing) for IoT applications. Key requirements of rule configurability and extensibility are mainly addressed through *i)* a web resource graph model, which enables the uniform representation of things and rules as web resources, and *ii)* the design and development of a rule engine that provides rule management features (scheduling, activation, persistence, and web-based exposure). Here, we present a resource-based graph model, which is purposely kept minimal to facilitate interoperation with third-party middleware, and a rule management framework implementation. We also present a detailed case study on energy efficiency in school buildings, which was developed within the Horizon 2020 GAIA research project [16], [19].

This paper extends a prior publication [20] by *i)* providing a consolidated resource model and a general-purpose IoT rule engine implementation, with enhanced features for rule creation and management, *ii)* validating the proposed approach through a case study on IoT for behavior-based energy efficiency in schools and discussing obstacles and lessons learnt within the case study, and *iii)* evaluating the performance of the rule engine that is used in the case study with encouraging results in terms of response latency for most significant operations.

The remainder of this article is organized as follows. In Section II, we discuss related studies that motivate our contribution. In Section III, we describe the resource-based model, and in Section IV, we provide design and implementation details of the proposed rule management framework. In Section V, we report on the use of the framework within the GAIA project for promoting energy savings initiatives in school buildings, we present performance evaluation results, and we discuss our work and provide insights into future work. Finally, Section VI presents the conclusions of the paper.

II. RELATED WORK

The IoT Reference Model that was proposed by Cisco in [21] consists of a seven-layered architecture with the objective of providing guidelines for IoT deployments and promoting replicability and collaboration. This model highlights the transition from *data in motion* (real-time and stream-based

data) to *data at rest* (data that are stored on devices and available through query-based APIs). In this work, we focus on *data at rest* since the proposed rule management framework leverages data abstraction functions that are typically provided by IoT middleware to expose IoT data as web resources and, thus, expose REST APIs for managing rules and related notifications. Hereafter, we comprehensively discuss the state of the art in this area, which includes stream and event processing solutions that are typically conceived for *data in motion*, middleware and rule-based reasoning for IoT applications.

A. EVENT PROCESSING

Event processing (EP) techniques are considered key technologies in the IoT domain since they enable the analysis of streams of events for the extraction of useful insights into real-world scenarios [22]. Complex event processing (CEP) is a subset of EP techniques whose objective is to detect complex events. CEP techniques are used to correlate or aggregate events over a time window and to conduct pattern detection in scenarios that involve multiple events or when it is necessary to be aware of the state of the working memory to take an action [23]. Modern EP techniques operate on streams of events that are processed as soon as they arrive; such systems are usually called Stream Processing Engines (SPEs). SPEs are designed to process incoming messages as they arrive, without necessarily storing them. Some operations still require the storage of an internal state; this can be realized using a conventional or embedded database [24]. SPEs use specialized primitives and constructs (e.g., time-windows, aggregation, and filtering) to express stream-oriented processing logic and are built for operating with potentially unbounded streams of data. Open-source stream processing engines, such as Apache Storm, Apache Stream and Apache Flink, have been implemented for use in the cloud and are currently also adopted in IoT environments [25]. Various studies propose novel stream processing frameworks that are designed for the IoT environment (e.g., DART [26] and PESP [27]), while [28] proposes a CEP mechanism that leverages edge computing for IoT data real-time processing.

B. MIDDLEWARE AND SERVICES FOR IoT APPLICATIONS

IoT middleware and service platforms provide connectivity to physical things and sensor data through high-level APIs that hide the heterogeneity of sensor and actuator technologies from applications. An extensive survey on IoT middleware is conducted in [29]. A review of both proprietary and open-source IoT platforms is conducted by Mineraud *et al.* in [30], which focuses on the ability to meet the expectations of IoT users, while a more comprehensive and recent survey is available in [31]. Some of the identified weaknesses of the state-of-the-art solutions are i) the scarce support that is offered to application developers beyond basic REST APIs for accessing sensor data [30] and ii) the need to abstract the heterogeneity of the resources of IoT systems [31].

A pioneering contribution in the specification and implementation of REST APIs for the IoT is the study by Guinard *et al.* [32]. A REST-based approach is applied by some of these authors in the AutoWoT project [33] to provide a toolkit that facilitates the rapid integration of smart devices into the Web by offering a general strategy for modeling and automatically generating web resources to represent a hierarchical structure of devices. WebPlug [34] is a framework for the WoT that enables users to deploy personal services by composing web resources that represent physical objects. A framework for supporting the development of IoT applications is proposed by the authors in a previous work [6]. The framework consists of a Web resource information model that is based on aggregation and reference relations, and middleware and tools for developing, publishing and composing web resources. Paraimpu [12] is a Web-based platform that enables the definition of mash-ups of smart things and includes a JavaScript-based rule engine for conducting data filtering operations. Yao *et al.* [3] present a layered web-based framework for managing and sharing data that are produced by physical things. The system also includes a rule engine for the automatic management and control of devices and also offers a GUI for defining rules and actions. A script-based framework, namely, ScriptIoT, has been proposed in [35] to enable users with little or no programming expertise to develop IoT applications with minimal effort. In [17], the need for higher-level representation of trigger-action rules for IoT end user development is advocated and supported by a user study; an ontology-based model that enables users to define abstract and technology-independent trigger-action rules in the IoT is also provided. Our work has a similar objective; however, while [17] exploits semantic reasoning capabilities to provide users with differentiated abstraction levels of interaction (high-level and medium-level), our system focuses on rule management and on enabling the composition and REST-based exposure of rules. Similar to [17], our rule management framework provides features that facilitate rule creation by users (e.g., by hiding and resolving on behalf of users low-level technological details, such as customizing rules with sensors identifiers), but it does not use a semantic model. Semantic models might complement our approach as additional sources of information and mediation between vocabularies.

"If This Then That" (IFTTT) [36] is a service that enables users to write simple applications that connect web services, social media sites and physical devices by leveraging the trigger-action paradigm (e.g., "If I arrive home then turn lights on"), but it is limited in terms of expressiveness since it does not allow users to create more structured rules (e.g., rules that combine multiple events and actions) [37]. In [10], a set of tools are presented that enable end users to specify trigger-action rules for the customization of Web applications via a context-driven approach. In [18], these tools are complemented by end user debugging support. MOZART [38] provides a rule-based composition language and a graphical

user interface for specifying an IoT application. It has been implemented as a set of components on top of Mozilla WebThings. While such works provide complete end user solutions that are comprised of a rule authoring editor and a rule execution engine, our contribution consists of a rule management framework that offers rule configuration and composition features and APIs for programmatic access by third-party applications.

IoT platforms are comprehensively evaluated in [39]. Among industry-driven IoT platforms, AWS IoT [40] includes a rule engine. The AWS IoT rules define one or more actions to be performed based on the data in a MQTT message. Typically, such actions enable the extraction of data from a message and the routing of them to another AWS service. Therefore, while the AWS IoT rules are fully integrated into the AWS IoT platform, our contribution aims at providing an IoT rule engine that operates at the level of data at REST and is not tied to an IoT platform.

Finally, hereafter, we briefly discuss application-specific IoT middleware that apply rule-based reasoning approaches, and we focus especially on those solutions that aim at realizing energy savings (similarly to our case study). Stavropoulos *et al.* [11] developed an energy saving system that was tested in a Greek public university building. Energy saving policies are enforced by two rule-based mechanisms that are implemented on top of an IoT middleware: one that is based on production rules and another that uses defeasible logics. Mainetti *et al.* [41] propose a rule-based semantic architecture for enabling users to define and enforce automation policies in an IoT context. SESAME-S [42] is a smart home prototype that leverages ontology-based context management techniques and rule-based reasoning that is implemented with the JESS rule engine [13] for enforcing energy-aware environmental control policies (such as automating the turning on/off of devices). A smart heating, ventilation and air conditioning (HVAC) system, which aims at minimizing energy costs while guaranteeing the thermal comfort of users, is proposed in [43]. Papaioannou *et al.* [44] implemented a gamification approach for improving behavior-based energy savings in public buildings, which was based on a rule-based game engine. More recently, an energy efficiency framework for domestic applications has been proposed in [45]. This framework uses a supervised machine learning classifier to process data from household sensors to identify abnormalities and formulate energy-saving recommendations. In comparison, while solutions in [44] and [45] have been developed for supporting energy-savings challenges, our rule-engine aims at supporting general-purpose IoT applications; furthermore, a case study evaluation has been conducted to evaluate our approach within an energy-saving case study in school buildings.

In conclusion, our contribution differs from the literature as follows. First, while stream and event processing techniques focus on data in motion, our approach focuses on data at rest. Thus, our rule management framework can be regarded as a pluggable service of an IoT middleware. Additionally, although our contribution is similar to [3] in that our solution also operates at the service layer abstraction level and enables users to create new rules via a web interface, it also provides an original contribution in that it offers rule configurability and extensibility features and does not depend on any IoT middleware since it has been conceived as a pluggable module that can be used with any IoT middleware that exposes REST APIs.

III. RULE MANAGEMENT FOR THE IoT

As discussed in the related work section, rule-based mechanisms are commonly applied in IoT scenarios to evaluate conditions that are expressed in terms of resource status. However, practical IoT deployments also demand the following:

- *Extensibility*: A new type of rules might be created from scratch (programmatically) or by editing the configuration of already implemented rules (e.g., by customizing an expression to be evaluated) to address unforeseen application needs and/or context changes (e.g., new sensors or new recommendation scenarios).
- *Configurability*: The same type of rule might be applied to create distinct instances for different areas of a target IoT environment (e.g., a school building) and related subareas (e.g., classrooms) with suitable customization of thresholds, sensors' identifiers (URIs) and the content of notification messages.

In addition, the system should offer rule management capabilities to various types of users (e.g., developers, system administrators, and end users who vary in terms of technical skills and access roles) and, thus, allow different degrees of freedom in rule behavior specification and activation. Thus, various levels of rule customization and specifications must be supported, from programmatical rule behavior definition, configuration of threshold values and changes in the condition expression at runtime, to assistance to end users in rule creation.

Hereafter we introduce the rule management framework that we developed to satisfy these requirements. The framework consists of the following main components:

- a *resource-based graph model*, which enables the uniform representation of things (including sensors and domain entities) and rules as URI-addressable resources. The adoption of this resource model and its exposure through RESTful APIs in the proposed framework provides an easy way of conducting Create, Read, Update and Delete (CRUD) operations on rules at runtime since conditions are expressed in terms of facts that are related to REST-based resources;
- a "*rule engine*", which is a software implementation of rule management features. This rule engine implements rule scheduling, activation, persistence, and web-based exposure features and offers a set of RESTful operations for creating and modifying rules at various levels of configurability and extensibility.

Finally, the framework has been conceived as a stand-alone module that offers REST-based APIs and interacts with third-party IoT/WoT middleware through REST APIs.

A. RESOURCE-BASED GRAPH MODEL

We define a resource-based model as a graph $G = (R, E)$, where R is a set of resources i, with $i \in \{1, .., |R|\}$ and E is a set of directed edges (i, j) from resource i to resource j with $i, j \in \{1, .., |R|\}$.

A *resource* is any entity that can be identified through a URI and can be accessed and manipulated through CRUD operations (in accordance with the main principles and best practices of the REST architectural style). A resource can be assigned an unqualified name and other descriptive attributes.

An *edge* is a directed link between two *resources* to which a meaningful label may be assigned to specify the type of relationship.

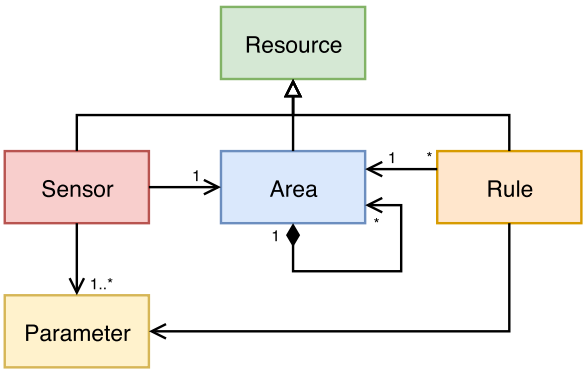

FIGURE 1. UML class diagram [46] of the resource-based graph model.

These are highly general definitions. We define the following types of resources, which characterize the IoT domain: *area*, *sensor*, *parameter* and *rule*. Fig. 1 presents a UML class diagram of this resource model [46]. For readability, the edges in Fig. 1 are not labeled, while the relations between resources that are represented by edges are highlighted in italics in the textual description of the model that is reported below.

An *area* is a resource that represents a physical or symbolic location/area. It can be further characterized by information that pertains to the domain of interest (e.g., an area that represents a school building can be enhanced with information regarding the number of students, surface, energy consumption, etc.). An area may *contain* other areas.

A *sensor* represents a physical or virtual sensor that gathers measurements that are related to some physical phenomena and *is located in* a specified area (the location of the sensor/area in which the measurement is conducted).

A *parameter* represents a physical phenomenon (e.g., temperature) that is directly *measured by* a sensor or derived from sensor measurements (e.g., relative humidity).

A *rule is* a resource whose internal behavior consists of verifying a condition and, if the condition holds, triggering an action. A rule condition *is a function of* a set of parameters. A rule *is associated with* an area, namely, the verified condition pertains to the associated area. Although a rule will typically evaluate parameters in the area to which it is associated, external parameters might also be considered if necessary (e.g., external temperature).

The model has been kept as simple as possible (a "minimal" resource model) to facilitate interoperation with third-party middleware. Although minimal, this resource model enables the representation of significant entities of the IoT domain and the definition of define rule resources on top of them. Moreover, a mapping of our model to the Web of Things (WoT) Thing Description model [47] is being developed.

Fig. 2 presents a portion of a resource-based graph that represents symbolic locations, sensors and measurements in a specified IoT application scenario, namely, a school that is involved in the experimentation activities of the GAIA research project [16]). As shown in the figure, a building may be modeled as a graph, whose nodes represent physical areas inside the building (e.g., a hall) and deployed sensors and related rules can be attached to the nodes to detect conditions of interest (e.g., anomalous energy consumption or power factor values). More precisely, the root area represents a School, which contains a Sport Block (e.g., the gymnasium hall), a Hall and a Teaching Block (not shown in the figure for clarity), which contains Classrooms and Laboratories. A Power Factor rule, which checks whether the power factor is below a specified threshold, is assigned to the Hall area. A rule (called Natural Light) evaluates the level of luminosity and the absorbed active power due to the use of artificial light in the Hall to suggest energy saving actions (e.g., switching off the light and exploiting the natural light). A Comfort Index rule is assigned to the Sport Block to evaluate the heat index via the joint evaluation of temperature and relative humidity measurements.

B. RULE STRUCTURE AND BEHAVIOUR

Our framework offers features for managing condition action (CA) rules in the form of:

$$IF\ some\ condition\ THEN\ some\ action \tag{1}$$

In a CA rule pattern, if the condition that is described in the IF clause is true, then one or more actions are executed. Reactive rules in the form of the event condition action (ECA) pattern are not supported by our system since our focus is on IoT data at rest.

By leveraging an object-oriented model, we define an abstract *Rule* class that provides the basic structure and behavior of a rule. It offers common attributes and the default implementation of basic operations together with services that are needed for I/O operations (e.g., invocation of external web services, log of the events, and notification mechanisms). Table 1 lists the main attributes of the *Rule* class. A rule is assigned a local name, an identifier and a URI since it is a web-addressable resource, while *fireInterval* and *fireCron* are optional attributes that are used to configure the frequency of rule firing occurrences. If the condition is satisfied, the action

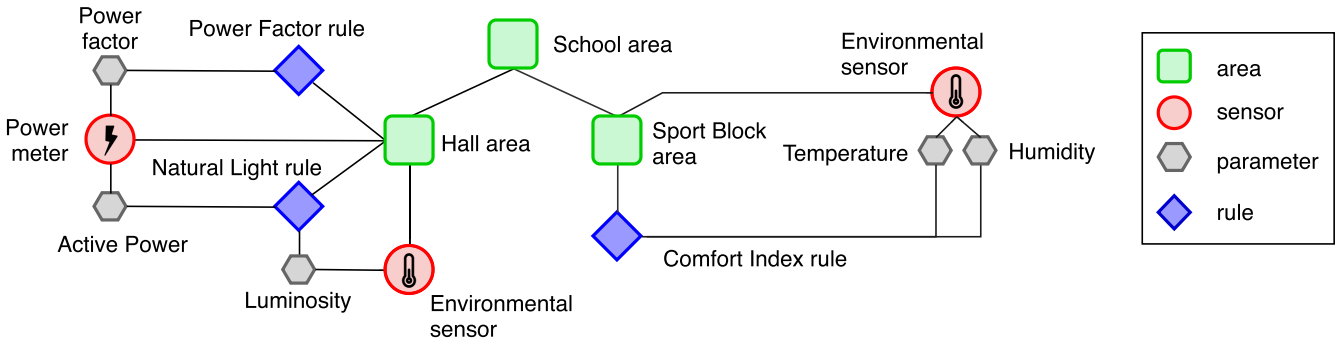

FIGURE 2. Simplified version of a resource graph that represents a school building area with two subareas (a Hall and a Sport Block). For example, the *Natural Light* rule uses the Active Power measurement that is provided by the power meter and the luminosity that is measured by the environmental sensor in the Hall to determine whether a light can be turned off without compromising user comfort.

TABLE 1. Attributes of the rule class.

Attribute	Description
name	name of the rule
id	internal rule identifier
URI	web-accessible rule identifier
description	a brief description of the rule behavior
fireInterval	interval in seconds between two consecutive firings of the rule (optional, default = 0)
fireCron	A cron expression for limiting the firing of the rule to a specified time instant or period (optional)

is performed if either *i*) the interval between the latest fire and the current time exceeds *fireInterval* or *ii*) the current time matches the cron expression in *fireCron*.

The behavior of a rule is defined through the following list of methods:

- *init()*: It executes initialization tasks, which includes rule validation (it checks if all the required fields are loaded). This method is invoked when the rule is loaded, and the rule is discarded if it returns false.
- *condition()*: Its objective is to determine whether a condition expression is true. This is left as an abstract method that must be implemented in a subclass.
- *fire()*: It invokes the *action()* method if *condition()* has returned true.
- *action()*: It implements an action. A default action can be specified or left as an abstract method.

Every rule subclass must implement the condition() method, which defines the condition to be checked, and can customize its behavior by overriding the default action() method.

Rule subclasses define more specific behaviors, especially in terms of condition evaluation. Rule subclasses are concrete classes that can be instantiated (rule instances). A rule subclass definition captures a behavior that is common to all its rule instances, but it provides users with the desired degree of freedom to further specify the rule behavior and instantiation (e.g., the specification of the condition, the type of parameters to be evaluated, and the area to which the instance should be attached). Rule subclasses can be specified to support various levels of customizability at the instantiation time (see Section IV-C for further details).

Rule classes and instances differ in how they refer to the resource model that is described above. In a rule subclass definition, conditions can be specified without necessarily referring to any resource in the model. If necessary, the rule subclass definition can be enhanced with additional attributes or annotations that enable reference to resources in the model when a rule instance is created. For instance, the minimal attribute list in Table 1 can be enhanced with additional attributes that act as placeholders for condition fields (e.g., specifying parameters to be evaluated, such as temperature and humidity). This example will be further clarified by referring to our current rule engine implementation in Section IV-C. Instead, an instance of a rule subclass is a resource in the graph-based resource model (it is associated with an area, and its condition is a function that evaluates measurements of specified parameters). For instance, in the example in Fig. 2, a Comfort Index rule instance evaluates temperature and humidity measurements in the Sport Block of a school.

Finally, a rule is further specialized as an atomic rule or a composite rule. An atomic rule is a rule for which the condition is specified in a self-contained way, while a composite rule's condition is defined as a composition of conditions of other rules (child rules), which can be atomic or composite rules. We have defined the following main composition operators:

- *AnyCompositeRule*: It implements a logical OR operation, namely, if at least one child condition is true, the composite rule is triggered.
- *AllCompositeRule*: It implements a logical AND operation, namely, the composite rule's action is triggered if all child conditions are true.
- *RepeatingRule*: It is triggered if the condition of a child rule has been consecutively verified at least *n* times (with *n* configurable) within a specified time interval (e.g., active power consumption measurements exceed a specified threshold at least 5 consecutive times during a one-hour interval).

IV. RULE ENGINE DESIGN AND IMPLEMENTATION

The rule engine logic leverages the resource information model that is described in Section III and handles corresponding rule instances. It is responsible for periodically checking

rule conditions and performing related actions. It also exposes REST CRUD operations to provide users with various levels of configurable and extensible rule management.

Since rules are associated with areas and other resources (e.g., parent areas, sensors, and measurements) that are maintained and made accessible by a third-party IoT middleware, several implementation options for managing such resources are possible. For instance, copies of these resources may be maintained by a persistence service local to the rule engine or they can be directly retrieved from the external IoT middleware (called hereafter the *IoT provider*). In our implementation, almost static data such as areas and related metadata are stored locally and refreshed if necessary, while sensor measurements are retrieved from the IoT provider at each iteration of the execution flow, as detailed below.

In the remainder of this section, we describe the software architecture of the rule engine, explain the implemented execution flow and, finally, explain how the proposed system support users in creating, handling and customizing rules.

A. ARCHITECTURE

The rule engine is implemented as a Java application by leveraging the Spring Framework [48] and it is composed of the following main modules (see Fig. 3):

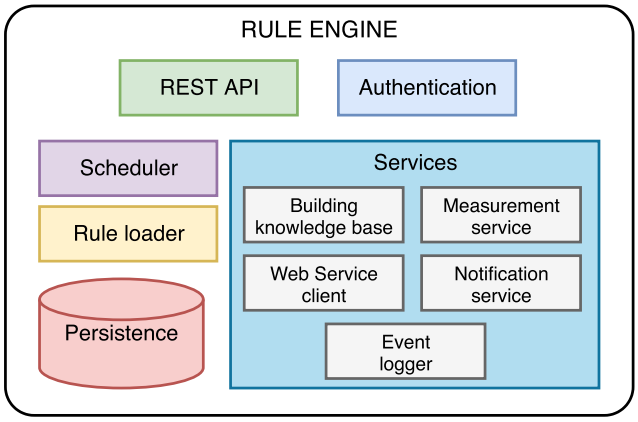

FIGURE 3. Architecture of the rule engine.

- *Persistence*: It stores rules and the parts of the graph that represent rules and related areas in a database (DB).
- *Rule loader*: It interacts with the DB and its main tasks are traversing the rule tree that is loaded from the DB and instantiating the corresponding Java classes.
- *REST APIs*: This module exposes HTTP-based REST APIs for managing rules and the engine.
- *Scheduler*: Its main responsibility is to fire rules at the specified frequency.
- *Authentication*: It keeps the rule engine authenticated to external services (e.g., IoT providers) in compliance with OAuth2 [49].
- *Services*: They implement common utility features that can be reused by rule instances.

At runtime, the Rule Loader component loads and processes the tree structure of the rules that have been stored in the DB and it instantiates the corresponding rule class for each encountered rule. By leveraging the Java Reflection API, it fills the object attributes with corresponding values that are retrieved from the DB. The rule structure representation follows the Composite pattern that allows the mixing of individual rules and their container objects (areas). Both rule and area objects implement the ''fire'' method: areas forward the method invocation to their children (rules), while rules implement the condition action (CA) behaviour. A scheduler, which is implemented by leveraging the Spring Framework Scheduler, periodically retrieves updated resource information and sensor measurements by querying the corresponding URIs that are handled by the IoT provider and executes the fire method on the rule tree.

As discussed above, the rule engine also implements services that can be reused for the management of rule instances:

- *Measurement service*: It updates the measurements that are required by the rules by querying the corresponding URIs exposed by the IoT provider. It fetches only the resources that are needed by the instantiated rules; the measurements are updated at every iteration before firing the rules. If the IoT provider supports this option, this service batches the requests to minimize the communication overhead. All retrieved measurements are cached for possible reuse by other rule instances.
- *Event logger*: It logs significant events (e.g., rule firing events) for reporting and analysis. For instance, when a rule is fired, it logs the most significant properties of the rule firing event.
- *Notification service*: It is used to send real-time notifications to third-party applications, such as messages via WebSocket and/or email delivery.
- *Building knowledge base*: It provides descriptive metadata for each area (e.g., type and square meters) and dynamic information, such as occupancy schedules and school calendars.
- *Web service client*: External web services may be invoked for the retrieval of additional information to be evaluated in a rule condition (e.g., weather information or air pollution).

Each rule has a reference to all implemented services; hence, it uses them to implement the condition() and action() methods.

The persistence layer has been implemented using OrientDB [50], since it is a hybrid Graph/Document NoSQL database that offers a persistence model that accommodates well our resource graph-based model. The DB handles every record as a set of key-value entries, which is called a *document*, with direct linking between documents leading to fast retrieval of related data compared to joins in an RDBMS. Moreover, it supports both schema-less and schema-full modes, thereby enabling us to follow a schema-hybrid approach in which classes have a predefined set of properties, which can be extended with new fields at

TABLE 2. Example of a rule instance that is stored in the database.

Field name	Description	Value (example)
@rid	ID of the vertex in the DB	25:241
@class	class name in the DB	Comfort Index
name	name of rule instance	Comfort Index in classroom Y
description	brief textual description	checking the comfort index value in classroom Y of school Z
message	message to be delivered to users through a notification action	switch off the lights and open the curtains
temperature_uri	URI of the temperature resource	gw1/roomX/temp
humidity_uri	URI of the humidity resource	gw1/roomX/humidity
threshold	threshold for the comfort index value	32

instantiation time. This feature is necessary for supporting the specification of additional fields in *User-defined Rules*, as described in Section IV-C. Finally, it offers a set of REST APIs that facilitate access to resources and it comes with a web GUI that can be used to author the resource model, including the rule structure and fields.

In our implementation, every Java Rule subclass is mapped onto a class in the OrientDB schema, which also includes inheritance relations (every rule inherits from the upper Rule class in the DB schema, thereby mimicking the application model). Once the rule classes have been defined, rule instances can be added to the database, by defining suitable relations among rules and areas in a tree structure and specifying the required fields. Table 2 presents an example of information that is related to a Rule instance that is maintained in the DB.

B. EXECUTION FLOW

The execution flow is comprised of the following main phases: initialization, scheduled execution, and rule firing.

1) INITIALIZATION

The engine loads the resource model from the database and locally replicates the tree-based rule model by following the Composite pattern [51]. More specifically, for each rule resource in the DB, a Java object is instantiated. In practice, for each vertex that is retrieved from the DB, an instance of the Java class whose name matches the value of the @class property is created, and its attributes take the values of the corresponding fields in the DB. If the *init()* method successfully terminates (all required fields are available and valid), the object is added to the tree of Rule instances. For simplicity, we refer to the flow of actions that are necessary for managing one rule tree (e.g., the set of rules that are associated with one school building), but in practice additional rule trees can be managed. The flow of actions is presented in detail in Sequence 1.

2) SCHEDULED EXECUTION

This step consists of the periodical execution of the following tasks. First, measurements that are required by the rules are acquired through suitable requests to specified URIs and shared among all rule instance. Then, the *fire()* method is recursively invoked on the rule tree. The flow of actions is reported in detail in Sequence 2.

1. Load the rule tree from the database.
2. Traverse the tree and replicate the structure by instantiating the correct Java classes using the @class property of each vertex that is stored in the database.
3. For each rule:
   a) Fill the fields with the values that are stored in the DB.
   b) Initialize the object that is conducting basic validation.
   c) Add the rule to the parent node if the initialization method returns true.

Sequence 1: Initialization

1. Check if the rule tree needs an update (the update can be forced by an external request).
2. Update the measurements that are required by the rules.
3. For each rule tree, fire the rules:
   a) Obtain the root of the tree.
   b) Call recursively the fire method on the composite structure.

Sequence 2: Scheduled Execution

3) RULE FIRING

Rule firing consists of the execution of the specified action upon verification of a condition. Each rule activation can be scheduled at the desired frequency. However, a suitable bounding must be considered with respect to the overall IoT system operation (e.g., the firing period should not be shorter than the sensor's sampling period). Rule firing is conducted through the steps that are reported in Sequence 3, which also include the execution of a default action method.

1) Check whether the parameters of the rule (fireCron and fireInterval) allow the rule to be fired. If not, stop here.
2) Evaluate the condition, and, if true, invoke the action method.
3) Create an entity that is filled with a subset of the rule's fields (e.g., measurements and textual suggestion).
4) Send the entity as a notification by email and through the WebSocket channel.
5) Log the event in the database.

Sequence 3: Rule Firing

C. RULE CREATION AND MANAGEMENT

The rule engine offers a set of REST APIs that enable users to create and modify rules with various levels of configurability and extensibility. With the term user, we refer here to: *i)* a developer, namely, either a rule engine developer who accesses the source code and DB APIs or an application developer who accesses the engine APIs, and *ii)* a suitably trained end user who accesses rule management capabilities through a GUI.

The engine supports various ways of specifying and modifying rule subclasses in the system, namely, various extents to which the rule behavior is programmatically specified or can be further customized by the end user at runtime through APIs. In the current implementation, rules can be grouped into two main categories:

- *Built-in rules*: The rule behavior is defined programmatically and, thus, is completely customizable by developers. Built-in rules are direct subclasses of the GaiaRule and their behavior in terms of condition() and action() method implementation is defined programmatically. Nonetheless, the values of various configurable parameters (e.g., threshold and parameters URI values) can be customized through the APIs when the rule is instantiated.
- *User-defined Rules*: The behavior is only partially programmatically defined, and is finalized by end users through the specification of a set of configurable parameters and fields. Two main types of user-defined rules are currently supported:
  a) ThresholdRule: Its condition compares a measurement value with a threshold. The measurement, the threshold and the comparison operator are configurable and can be provided at runtime by users.
  b) ExpressionRule: Its condition evaluates an expression using a customized version of the expr4j[1] library notation, which is defined by the end user and contains references to measurements and other parameters (URIs in the graph-based resource model).

D. ASSISTED RULE CREATION

When a user creates a rule instance, information is needed to suitably characterize the instance (e.g., the associated area, sensor identifiers and threshold values). Providing this information can be error-prone and tedious. To address this issue, the engine offers an assisted rule creation feature, which, given the rule class and the area that is associated with the instance to be created, produces a rule instance proposal into which suitable values have been filled. The implementation of this feature requires the storage of default or placeholder values of parameters for each Rule class, together with metadata (a rule ''template''). More specifically, for each field of the class, a default value, an optional brief description of the field and a flag that qualifies the field as either mandatory or optional can be specified. For instance default values can be provided for threshold values, while they cannot be provided for fields that are related to URIs since these values depend on the area to which the rule instance will be attached. To handle these cases, we use placeholder values that contain hints for deriving the suitable value according to the rule context. For instance, the *temperature_uri* field can be set to a default value (e.g., a ''Temperature'' string). When the rule instance is created, the system will try to find, inside the area to which the rule must be attached, a URI for gathering temperature measurements for that area. As an additional mechanism, if the placeholder value has not been set, the system will attempt to infer it by parsing the name of the field using simple rules (e.g., *humidity_uri* is mapped to a relative humidity parameter).

Although we have considered examples in which rules use measurements of parameters that are associated with the same area, more complex scenarios may arise, for instance, rule conditions that use measurements from sensors that are located in areas that differ from the rule's area, such as a parent area, the root of the building tree-based representation, or an external location. We defined three prefixes for field names to address these cases:

- *ext_*: the sensor should measure an outdoor parameter (e.g., external temperature);
- *root_* : the system will look for a sensor that is attached to the root area (the building itself);
- *parent_*: the required measured parameter is related to the parent of the area to which the sensor is attached.

This mechanism also assumes that the same taxonomy is used by the IoT provider to define measured parameter types, since we rely on a simple syntactical matching. The adoption of ontologies and annotations in the Rule subclass definition could improve interoperability with third-party naming conventions and will be investigated in future work.

The assisted rule creation service is exposed as a REST API so that it can be invoked by external end user applications (e.g., rule authoring tools). Fig. 4 presents an example of a flow of REST invocations that implement the assisted rule creation procedure. The end user interacts with a GUI and starts creating a rule instance by specifying a site (an area identifier) and the type of rule subclass to be instantiated. A REST request message for rule creation that contains this information is sent to the rule engine. The rule engine creates a rule instance by leveraging user-provided information, the rule's default values and information that can be inferred through the model (URIs for retrieving measurements). The user may inspect the specifications of the rule instance, modify it if necessary, and, finally, request the creation of the rule instance. We chose to implement a partially automated procedure that requires the user's confirmation instead of a completely automated procedure to enable users to check the results and make modifications if necessary.

[1] http://expr4j.sourceforge.net/

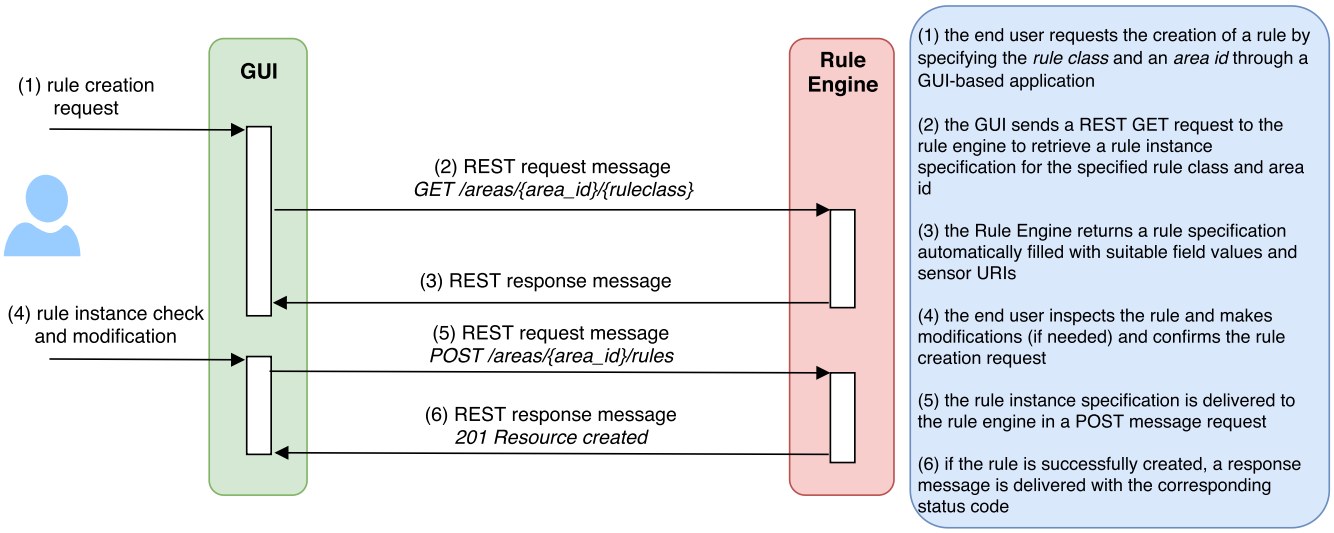

FIGURE 4. Example of assisted rule creation.

V. CASE STUDY EVALUATION

In this section, we evaluate the effectiveness of the proposed rule engine via a case study that is implemented in the context of the GAIA Horizon 2020 European project [19], [52]. First, we describe the objectives of the project and its real-world IoT deployment in several school buildings. Then, we report on how the rule management framework, which is currently operational and accessible to authorized users, has been fully integrated into GAIA and we present performance evaluation results. This evaluation aims at validating the proposed rule engine framework as a service that can be provided on top of third-party IoT middleware and accessed through external end user applications. A performance evaluation is also conducted with this objective, by measuring the latency of a set of significant operations (operations that require access to IoT middleware and operations that are suitable for invocation by end user applications).

A. IoT FOR BEHAVIOUR-BASED ENERGY EFFICIENCY

The GAIA project aimed at promoting energy efficiency and sustainability awareness in education environments by focusing on behavioral change to kick-start energy savings. GAIA implemented a real-world multisite IoT infrastructure in which sensor measurements are used to enhance educational experiences to help engage students in every-day energy-friendly practices. The GAIA monitored sites consist of 25 school buildings in Greece, Italy and Sweden, which cover a range of local climatic conditions and educational levels (primary, secondary, high school and university). At each site, a subset of classrooms, shared spaces and teacher/staff rooms have been selected for GAIA's environmental monitoring and the following types of measurements are periodically sampled and acquired: *(a)* the power consumption of the whole building and selected rooms/areas, *(b)* environmental data within individual classes (temperature, humidity, luminosity, noise, and other data such as particulate matter and CO_2 depending on the school deployment), and *(c)* weather conditions and air pollution levels. The GAIA Platform, which is described in [16], is comprised of over 1200 IoT monitoring endpoints, which were created using heterogeneous hardware and software technologies and supports various commercial hardware/sensor vendors and open-source solutions [53]. This real-world IoT deployment, which encompasses hundreds of monitoring end points, enables a case study to be conducted at a larger scale compared to similar experimental activities in related studies [11], [41], [42]. In the GAIA platform, at the end of the first year (after 6 months of fully operational infrastructure), approximately 250 million tuples had been collected, which correspond to approximately 115 GB of data. The data size of a message is 100 bytes on average. The acquisition time interval ranges from 1 second to 5 minutes and varies from site to site.

The GAIA IoT middleware provides data acquisition, processing and storage features. It is composed of modules that are accessible through REST APIs:

- *Measurement repository:* It stores all the measurements and provides aggregates at various granularities; it also contains the structure of the buildings and metadata of the sensors (e.g., measuring unit and parameters).
- *AAA:* It is the Authentication, Authorization and Accounting service for applications and services that are used within GAIA.
- *Building knowledge base:* It provides information about the school buildings, such as floor plans, room areas, elevation, and year of construction, along with dynamic information such as activity calendars of the schools and the schedules of the classrooms, laboratories, and conference hall.
- *Analytics:* It provides on-demand analytics and anomaly detection analysis of a building's power consumption profile.

TABLE 3. Built-in rule subclasses that were implemented in the GAIA case study.

Rule	Description	Sensors (measurements)
Exploit Natural Light	If the lights are ON and the room is OCCUPIED the rule checks the internal and external luminosities to determine whether the artificial lighting can be switched off without compromising the user comfort.	Ext. Luminosity, Power, Occupancy/Schedule
Don't waste energy	If the room is empty and the active power measurement exceeds a specified threshold, a notification is delivered to suggest switching off lights and/or devices.	Power, Occupancy/Schedule
Holiday shutdown	A message is sent to the building manager by email before a holiday period to remind him/her to check and switch off devices before the vacation period.	Schedule
Power Factor	If the power factor falls below a threshold the system logs the occurrence. The events are recorded to be later retrieved and analyzed by the building manager.	Power Factor
Temperature forecast	The objective of this scenario is to warn people of the likely sudden decrease of the temperature in the next days, to suggest that they wear warmer clothes and to suggest that the building manager decreases the comfort temperature of the heating system by 1 degree.	Weather forecast, Schedule
CO_2 **Level control**	The objective of the scenario is to maintain a satisfactory level of CO_2 inside an area. This is realized by triggering a notification delivery when a comfort level threshold is exceeded (e.g., 1000 ppm).	External and Internal Temperatures, CO_2
Comfort index	Under high temperature and humidity conditions, the comfort factor can be controlled by evaluating the heat index parameter. The system sends an alert to the building manager to make him aware of the problem.	Temperature, Relative Humidity

The rule management framework that is presented in this work has been used in GAIA to produce energy-saving recommendations for the pursuit of energy efficiency goals. The recommendations primarily provide suggestions for energy-saving behavior changes (targeting building managers, students and teachers) but can also provide building managers with suggestions for technical maintenance interventions or building renewal actions. Given the range of schools that are involved in the project experimentation activities, the recommendations have been customized according to school building characteristics (e.g., geographical location, internal space allocation, and user habits).

B. RULE DESIGN FOR THE GAIA CASE STUDY

To use the rule engine to produce energy-saving recommendations according to the environmental and energy consumption behaviors of GAIA schools, we defined a set of rule subclasses, which are called GAIA rules.

GAIA rules are built-in rules that are designed to support GAIA behavior-based energy saving objectives. We specified rules' objectives, related parameters and values (e.g., thresholds) in collaboration with school building managers. First, we selected a set of recommendation examples from previous experiences in energy savings in school buildings [54]. Moving from these previously elaborated energy saving tips we specified a set of scenarios and rules in consideration of the account number, type and locations of the sensors that are installed in each school [16], [52].

Then, we prepared a document that describes the rules to non-technical users (building managers or teachers/principals of GAIA schools). For each rule we provided a textual description, an example of usage in a realistic scenario and a set of questions that are aimed at: i) determining whether the rule could be applied in areas of the school; ii) checking the availability of sensors or other sources of information for evaluating the rule condition; and iii) establishing meaningful values for thresholds according to the type of area, its usage, weekly schedule and, if necessary, a methodology for defining such values.

The submission of questionnaires to school personnel was mediated by members of the project team, technical assistance and translation into the native language were provided. At the end of this step, we had an initial picture of rule-based recommendations that could be instantiated. Two rule subclasses were deleted from the list since they were considered either noninteresting or unfeasible. We created an initial set of rule instances (34 rule instances) to bootstrap the usage of the rule engine, and additional rule instances were added in the second half of the project (94 rule instances in total). Table 3 lists the main rule subclasses that were instantiated in GAIA schools.

End users (students, teachers, and building managers) may access the latest notifications and query previously triggered notifications through a building management web application that was developed within GAIA [55]. Leveraging the REST APIs that are exposed by the rule engine, this web application also enables authorized users to create new simple threshold rules and to instantiate the built-in rules, which are listed in Table 3. A system administrator can also use a *dashboard*, namely, a web application that was developed within GAIA for navigating the resource model of each school and visualizing associated rules, and the OrientDB web-based Graphical User Interface to access, navigate and modify the graph of instantiated rules.

C. EXAMPLE OF A GAIA-ENABLED ENERGY SAVING THROUGH BEHAVIOURAL CHANGE

We briefly describe an activity that was conducted in a high school in Italy to provide an example of the possible impact of the rule engine usage in a school community. Leveraging data and notifications that were provided by the building management web application, students planned and realized a simple

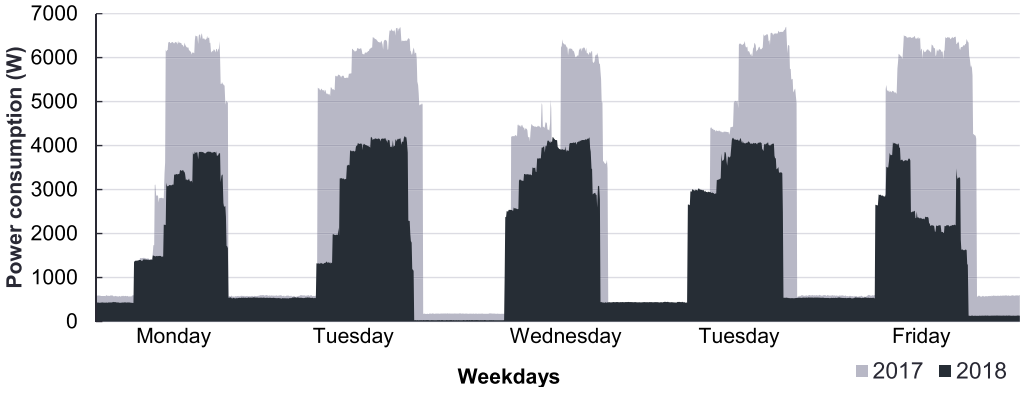

FIGURE 5. Power consumption reduction in a school in Prato: comparison of power consumption profiles in 2017 (no GAIA activities) and 2018 (with GAIA activities).

but effective energy saving action plan, which consisted of actively monitoring the usage of lighting in classrooms and in common areas and performing simple actions after receiving notifications from the rule engine (e.g., switching off lights and engaging janitors in checking and reducing energy waste situations). As a result of a 2 week-activity, the power consumption in the central hall was determined to be approximately 4.9 kW during a typical work day. Examining indoor luminosity data, sensors indicated that this consumption was unnecessary, since the illuminance from natural light was close to 400 lux, which corresponds to satisfactory indoor lighting conditions. Students determined that this was a recurring event in this building area and an ''Exploit Natural Light'' rule instance was created to monitor the situation and create alerts that were paired with energy saving recommendations. The behavioral change that was triggered by these notifications led to significant reductions in the energy consumption (40%) and maximum power value (37%) in that area during a 1-week period in 2018 compared to the same period in the previous year under similar light conditions (see Fig. 5).

D. PERFORMANCE EVALUATION

Hereafter, we report on testing activities that were conducted to evaluate the performance of the rule engine that was configured and run for the case study within the GAIA project (the rule engine managed 94 rule instances and registered 200,000 ca. events by the end of the project). The rule engine is deployed in a Ubuntu 16.04 Virtual Machine (VM) that is configured with 4 GB RAM and 2 vCPUs on an Intel(R) Core(TM) i5-3340 CPU @ 3.10GHz.

The tests aim at measuring the latency in performing a set of significant operations, although, as discussed above, the system does not aim at implementing real-time service behavior. Some of these operations are triggered by an external client; the client is located in the same machine as that on which the rule engine ran so that response time is not influenced by network latency.

Hereafter, we describe the adopted metrics and test methodology.

- *Scheduled iteration time*. This is the time that is needed to accomplish a scheduled execution flow, as described in Section IV. The iteration period is configurable and is set to 5 minutes in the current deployment. We provide an average value of this metric that has been calculated over an interval of approximately 24 hours (271 iterations). Measured values also account for delays that are caused by possible errors (e.g., possible temporary communication failures with the GAIA platform for sensor data acquisition).
- *Query time for events*. This is the response time to a query regarding registered events. Since this value may depend on the type of issued query, we defined a set of representative queries by varying query parameter values. We specified various values for the maximum number of events to be returned in the result, possible filters (e.g., school id and rule class id), and history intervals (e.g., 1, 5, and 10 days) (see Table 4). For each type of event query, we created a set of 20 requests. At each iteration, the rule engine is queried, and the elapsed time is recorded. The parameters of the query differ among the iterations: the start and end time ranges are updated at every iteration so that values need not to be cached.
- *Rule instance proposal retrieval time*. We focused on the first part of the assisted rule creation workflow (Fig. 4), namely, the retrieval of a rule instance proposal with values that are filled in by the engine. We considered 26 requests for areas that were mixed between the Power Factor and Comfort Index rule classes. The availability of default values for mandatory rule fields had been prechecked to prevent misleading results.
- *Rule creation time*. The client sends 30 requests for the creation of rule instances of the SimpleThresholdRule class for a demo school building. The client waits 30 seconds after each request.

The results are reported in Table 4. Regarding the scheduled iteration, the maximum time per iteration during the monitored period is 21.8 seconds, while the minimum is 5.4 seconds. The average iteration time is 8.4 seconds with

TABLE 4. Performance evaluation of the rule engine in the GAIA case study.

Metric	Average (ms)	STD (ms)
Iteration time	8808	3372
Query time for events (limit 10, filtered by school id, 5 days)	379	78
Query time for events (limit 1000, filtered by school id, 1 day)	371	44
Query time for events (limit 1000, filtered by school id, 10 days)	584	88
Query time for events (latest 100, filtered by rule class id)	1149	456
Rule instance proposal retrieval time	690	223
Rule instance creation time	632	111

a standard deviation of 3.2 seconds, because some rules use cached values, while others must retrieve required data from the platform or external web services. Therefore, the measured delay depends on what and how many rules are fired at each iteration (for instance, rule instance *a* could be configured to be fired every 2 hours, and rule instance *b* every 5 minutes). Regarding event queries, latency measurements strongly depend on the time range and filters that are applied; indeed, retrieving the latest 100 events that are filtered by a specified rule class may take more than 1 second, whereas retrieving 10 events for a specified school in a 5-day temporal window takes less than 400 ms on average. Regarding the assisted rule creation operation, both operations (rule instance proposal retrieval and rule instance creation) require 600 ms ca. Therefore, the measured response times are within the range of response time limits that is suggested in [56].

We also conducted tests to evaluate the times that are needed for loading the resource model from the DB, and for rule instantiation against the total number of rules (from 100 to 12800 rules). The results, which show a sublinear relationship with the rule cardinality, have been reported in a previous study [20].

E. DISCUSSION AND FUTURE WORK

The evaluation results that were gathered from the case study within the GAIA project support the feasibility of the proposed approach. Our rule engine enabled users to configure and extend rules in the specified IoT application domain; in addition, various ways of specifying and instantiating rules were made available to them (built-in and user-defined rule instances were accessible through a set of web-based applications). The expressivity of the rule model and rule creation and configurability levels that were supported by APIs were sufficiently rich for satisfying the requirements that were identified in the GAIA trial in schools. However, some of the rule creation and configuration tasks were performed by the research project staff (technical and nontechnical personnel), although this occurred with the collaboration of end users. The main obstacles that were encountered in the adoption of the rule engine in this case study are as follows: (i) in most schools the role of building manager is unfilled and school principals are not required to have expertise on energy management or knowledge of critical energy-consuming situations in school buildings; (ii) only a subset of teachers have ICT skills (as end users) and technical/scientific background for guiding students in rule-supported energy saving activities; and (iii) teachers tend to have little time available to dedicate to extracurricular activities.

From this experience, the following lessons were learnt in relation to the use of the rule engine in IoT-enabled educational environments:

- It is necessary to provide direct and informal support to users (especially teachers, since they are the gateway to students) in codesigning tailored activities, including the configuration of rule instances that support the analysis of IoT sensor data;
- Engaging material and step-by-step guidelines and examples should be provided to minimize the additional time that must be devoted by teachers and students to the design, configuration and use of rules in their energy-saving experiments.
- More generally, detailed material should be provided that explain the objective of each rule class and examples of how a user can specify a rule instance in a specified area (e.g., how suitable threshold values could be estimated through experiments).

To consolidate these lessons, further energy-saving and educational activities are planned in additional schools that are joining GAIA follow-up activities. A set of educational materials [57] has been produced within the project for guiding teachers in implementing student activities that target energy saving by leveraging GAIA tools, which include energy saving recommendations that are provided by the rule engine. Additional examples and documentation will be added to this material as additional activities are completed to serve as guidelines and inspiring examples for new teachers and schools. In the future, investigation of the use of the rule engine for a similar objective (energy saving) in various contexts (e.g., private homes or organizations) in which users might directly benefit from energy savings that are realized through rule-based notifications (e.g., economic benefits) would also be of interest.

Since the rule management framework APIs have been designed to be accessed by third-party applications, a subset of rule management features have been exposed to end users through the GAIA Building Management System application. Future research directions will include the development of a rule authoring tool that will enable end users to easily browse rules, to create and configure new rule instances and to attach them to areas that represent sensor-monitored sites through a GUI. This will enable us to conduct a careful evaluation in terms of end user acceptance of the services that are offered by the rule engine and of the usability and perceived usefulness of the companion graphical tool.

The case study that is presented in Section V demonstrates the correct interaction of the rule engine with a third party IoT middleware. However, interoperability with external IoT middleware should be further evaluated with respect

to alternative IoT middleware implementations, as we plan to do in the future. This could be easily realized if resources are URI-addressable and the model can be recursively navigated (subresources can be dynamically retrieved from parent resources). Finally, the generalization of the approach should be validated by adopting the framework in other IoT application domains (e.g., home monitoring and smart cities).

VI. CONCLUSION

Bridging the physical and the digital world through the use of standard Web technologies is one of the main directions that have been taken by the IoT community. Rule-based approaches can substantially simplify the ways in which we define the interaction between these two domains and there is often substantial overlap of such definitions in real-life applications; thus, web-based rule management frameworks have a role to play in this domain.

In this work, we proposed a framework and a reference rule engine implementation that extend a minimal web-of-things graph representation with rule-based data processing capabilities. By leveraging the graph abstraction and REST guidelines, such processing capabilities are uniformly treated as URI-addressable resources, while their positions in the graph and the relations with other resources (e.g., sensors and locations) are essential for the meaningful specification of rules and their relations with the physical environment. We described the rule management capabilities that are offered and their provision to users of facilities for configuring and extending rules and managing rule instances.

We presented a case study in which the proposed rule engine was used to support a set of energy saving and educational activities in a few European school buildings that were involved in the GAIA project. In the case study, we experimentally implemented the rule engine in a practical scenario with nontechnical users (students, teachers and school staff). We also evaluated the performance of the rule engine instance that was used in the case study, with encouraging results in terms of the required latency for performing significant internal operations (the required latency for the periodic execution of measurement updates and checking of rule conditions) and operations that were triggered from an external client (e.g., rule creation and query response time). Then, we discussed the obtained results and highlighted the beneficial effects in energy saving activities and the encountered obstacles. Finally, we presented the lessons that were learnt and discussed directions for future investigation.

REFERENCES

[1] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.
[2] *Services in the Future Internet*, FP8 Expert Group, Eur. Commission, Brussels, Belgium, Feb. 2011.
[3] L. Yao, Q. Z. Sheng, and S. Dustdar, "Web-based management of the Internet of Things," *IEEE Internet Comput.*, vol. 19, no. 4, pp. 60–67, Jul. 2015.
[4] H. Elazhary, "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *J. Netw. Comput. Appl.*, vol. 128, pp. 105–140, Feb. 2019.
[5] M. Tang, Y. Xia, B. Tang, Y. Zhou, B. Cao, and R. Hu, "Mining collaboration patterns between APIs for mashup creation in Web of Things," *IEEE Access*, vol. 7, pp. 14206–14215, 2019.
[6] F. Paganelli, S. Turchi, and D. Giuli, "A Web of things framework for RESTful applications and its experimentation in a smart city," *IEEE Syst. J.*, vol. 10, no. 4, pp. 1412–1423, Dec. 2016.
[7] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.
[8] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the Web of things," in *Proc. Internet Things (IOT)*, Nov. 2010, pp. 1–8.
[9] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.
[10] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, "Personalization of context-dependent applications through trigger-action rules," *ACM Trans. Comput.-Hum. Interact.*, vol. 24, no. 2, pp. 14:1–14:33, Apr. 2017.
[11] T. G. Stavropoulos, E. Kontopoulos, N. Bassiliades, J. Argyriou, A. Bikakis, D. Vrakas, and I. Vlahavas, "Rule-based approaches for energy savings in an ambient intelligence environment," *Pervas. Mobile Comput.*, vol. 19, pp. 1–23, May 2015.
[12] A. Pintus, D. Carboni, and A. Piras, "Paraimpu: A platform for a social Web of things," in *Proc. 21st Int. Conf. World Wide Web (WWW)*, New York, NY, USA, 2012, pp. 401–404.
[13] E. Friedman-Hill, *Jess in Action: Java Rule-Based Systems*. Shelter Island, NY, USA: J, Manning Publications Company, 2003.
[14] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "SmartSantander: IoT experimentation over a smart city testbed," *Comput. Netw.*, vol. 61, pp. 217–238, Mar. 2014.
[15] V. Gutierrez, D. Amaxilatis, G. Mylonas, and L. Munoz, "Empowering citizens toward the co-creation of sustainable cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 668–676, Apr. 2018.
[16] D. Amaxilatis, O. Akrivopoulos, G. Mylonas, and I. Chatzigiannakis, "An IoT-based solution for monitoring a fleet of educational buildings focusing on energy efficiency," *Sensors*, vol. 17, no. 10, p. 2296, Oct. 2017.
[17] F. Corno, L. De Russis, and A. Monge Roffarello, "A high-level semantic approach to end-user development in the Internet of Things," *Int. J. Hum.-Comput. Stud.*, vol. 125, pp. 41–54, May 2019.
[18] M. Manca, F. Paternò, C. Santoro, and L. Corcella, "Supporting end-user debugging of trigger-action rules for IoT applications," *Int. J. Hum.-Comput. Stud.*, vol. 123, pp. 56–69, 2019.
[19] GAIA Project Consortium. *GAIA H2020 Project Website*. Accessed: Mar. 5, 2020. [Online]. Available: http://gaia-project.eu
[20] G. Cuffaro, F. Paganelli, and G. Mylonas, "A resource-based rule engine for energy savings recommendations in educational buildings," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.
[21] Cisco. (2014). *The Internet of Things Reference Model*. Accessed: Apr. 16, 2020. [Online]. Available: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf
[22] M. Dayarathna and S. Perera, "Recent advancements in event processing," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 33:1–33:36, Feb. 2018.
[23] D. Luckham, *The Power of Events*, vol. 204. Reading, MA, USA: Addison-Wesley, 2002.
[24] T. Bass, "Mythbusters: Event stream processing versus complex event processing," in *Proc. Inaugural Int. Conf. Distrib. Event-Based Syst.*, 2007, p. 1.
[25] S. Yang, "IoT stream processing and analytics in the fog," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 21–27, Aug. 2017.
[26] J.-H. Choi, J. Park, H. D. Park, and O.-G. Min, "DART: Fast and efficient distributed stream processing framework for Internet of Things," *ETRI J.*, vol. 39, no. 2, pp. 202–212, 2017.
[27] C. Hochreiner, M. Vogler, S. Schulte, and S. Dustdar, "Elastic stream processing for the Internet of Things," in *Proc. IEEE 9th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2016, pp. 100–107.
[28] L. Lan, R. Shi, B. Wang, L. Zhang, and N. Jiang, "A universal complex event processing mechanism based on edge computing for Internet of Things real-time monitoring," *IEEE Access*, vol. 7, pp. 101865–101878, 2019.
[29] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[30] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Commun.*, vols. 89–90, pp. 5–16, Sep. 2016.
[31] W. Kassab and K. A. Darabkh, "A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations," *J. Netw. Comput. Appl.*, vol. 163, Aug. 2020, Art. no. 102663.
[32] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the Web of things," in *Proc. 6th Int. Conf. Netw. Sens. Syst. (INSS)*, Jun. 2009, pp. 1–4.
[33] S. Mayer, D. Guinard, and V. Trifa, "Facilitating the integration and interaction of real-world services for the Web of things," *Proc. Urban Internet Things–Towards Program. Real-Time Cities (UrbanIOT)*, 2010.
[34] B. Ostermaier, F. Schlup, and K. Römer, "WebPlug: A framework for the Web of things," in *Proc. 8th IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PERCOM Workshops)*, Mar. 2010, pp. 690–695.
[35] H.-C. Hsieh, K.-D. Chang, L.-F. Wang, J.-L. Chen, and H.-C. Chao, "ScriptIoT: A script framework for and Internet-of-Things applications," *IEEE Internet Things J.*, vol. 3, no. 4, pp. 628–636, Aug. 2016.
[36] S. Ovadia, "Automate the Internet with 'if this then that' (IFTTT)," *Behav. Social Sci. Librarian*, vol. 33, no. 4, pp. 208–211, Oct. 2014.
[37] B. Ur, M. P. Y. Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, "Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2016, pp. 3227–3231.
[38] A. Krishna, M. L. Pallec, A. Martinez, R. Mateescu, and G. Salaün, "MOZART: Design and deployment of advanced IoT applications," in *Proc. Companion Proc. Web Conf.*, Apr. 2020, pp. 163–166.
[39] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt, "A detailed analysis of IoT platform architectures: Concepts, similarities, and differences," in *Internet of Everything*. Singapore: Springer, 2018, pp. 81–101, doi: 10.1007/978-981-10-5861-5_4.
[40] *AWS IoT*. Accessed: Sep. 25, 2020. [Online]. Available: https://docs.aws.amazon.com/iot/
[41] L. Mainetti, V. Mighali, L. Patrono, and P. Rametta, "A novel rule-based semantic architecture for IoT building automation systems," in *Proc. 23rd Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2015, pp. 124–131.
[42] A. Fensel, S. Tomic, V. Kumar, M. Stefanovic, S. V. Aleshin, and D. O. Novikov, "SESAME-S: Semantic smart home system for energy efficiency," *Informatik-Spektrum*, vol. 36, no. 1, pp. 46–57, Feb. 2013, doi: 10.1007/s00287-012-0665-9.
[43] C. Marche, M. Nitti, and V. Pilloni, "Energy efficiency in smart building: A comfort aware approach based on social Internet of Things," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.
[44] T. G. Papaioannou, K. Vasilakis, N. Dimitriou, A. Garbi, and A. Schoofs, "A sensor-enabled rule engine for changing energy-wasting behaviours in public buildings," in *Proc. IEEE Int. Energy Conf. (ENERGYCON)*, Jun. 2018, pp. 1–6.
[45] A. Alsalemi, Y. Himeur, F. Bensaali, A. Amira, C. Sardianos, I. Varlamis, and G. Dimitrakopoulos, "Achieving domestic energy efficiency using micro-moments and intelligent recommendations," *IEEE Access*, vol. 8, pp. 15047–15055, 2020.
[46] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Reading, MA, USA: Addison-Wesley, 1999.
[47] M. Kovatsch, T. Kamiya, M. McCool, V. Charpenay, and S. Käbisch. Web of things (WoT) thing description. W3C Recommendation, Apr. 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/.
[48] R. Johnson *et al. The Spring Framework—Reference Documentation*. Accessed: Dec. 4, 2020. [Online]. Available: https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/index.html
[49] D. Hardt, *The Oauth 2.0 Authorization Framework*, document RFC 6749, IETF, Internet Requests for Comments, Oct. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt.
[50] *Orientdb Official Web Site*. Accessed: Mar. 1, 2020. [Online]. Available: https://orientdb.com/
[51] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*, vol. 49, no. 120. Reading, MA, USA: Addison-Wesley, 1995, p. 11.
[52] G. Mylonas, D. Amaxilatis, I. Chatzigiannakis, A. Anagnostopoulos, and F. Paganelli, "Enabling sustainability and energy awareness in schools based on IoT and real-world data," *IEEE Pervas. Comput.*, vol. 17, no. 4, pp. 53–63, Oct. 2018, doi: 10.1109/MPRV.2018.2873855.
[53] L. Pocero, D. Amaxilatis, G. Mylonas, and I. Chatzigiannakis, "Open source IoT meter devices for smart and energy-efficient school buildings," *HardwareX*, vol. 1, pp. 54–67, Apr. 2017.
[54] A. Galata, F. Di Gennaro, G. Pedone, Y. Roderick, M. Brogan, and A. Sretenovic, "A catalogue of optimization scenarios-to enhance decision-making in establishing an efficient energy management programme," in *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM*. Vienna, Austria: CRC Press, 2014, p. 383.
[55] G. Mylonas, D. Amaxilatis, H. Leligou, T. Zahariadis, E. Zacharioudakis, J. Hofstaetter, A. Friedl, F. Paganelli, G. Cuffaro, and J. Lerch, "Addressing behavioral change towards energy efficiency in European educational buildings," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.
[56] J. Nielsen. *Website Response Times*. Accessed: Mar. 28, 2020. [Online]. Available: https://www.nngroup.com/articles/website-response-times/
[57] GAIA Project Consortium. *GAIA Educational Material*. Accessed: Sep. 5, 2020. [Online]. Available: http://gaia-project.eu/index.php/en/educational-material/

FEDERICA PAGANELLI (Member, IEEE) received the Ph.D. degree in telematics and information society from the University of Florence, Italy, in 2004.

From 2006 to 2018, she was a Researcher with CNIT, Italy. She is currently an Assistant Professor with the Department of Computer Science, University of Pisa. Her research interests include resource management in software-defined network infrastructures, network virtualization, and protocols and services for the Internet of Things. She has cochaired several editions of the Workshop on Orchestration for Software-Defined Infrastructures (O4SDI) and serves as an Associate Editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and *Future Internet*. She contributed to the standardization activities of the IEEE NGSON Working Group.

GEORGIOS MYLONAS received the diploma, M.Sc., and Ph.D. degrees from the Department of Computer Engineering and Informatics, University of Patras, Greece. He is currently a Senior Researcher with the Industrial Systems Institute, Patras, and the Computer Technology Institute and Press "Diophantus", Patras. His research interests include the IoT, wireless sensor networks, distributed systems, and pervasive games. He has been involved in the AEOLUS, WISEBED, SmartSantander, and AUDIS and OrganiCity projects, and has focused on algorithmic and software issues of wireless sensor networks. He coordinated the Green Awareness in Action (GAIA) H2020 Project.

GIOVANNI CUFFARO received the M.S. degree in computer engineering from the University of Florence, Italy, in 2018.

He was with the National Interuniversity Consortium for Telecommunications, Italy, from 2016 to 2019, where he was involved in the GAIA project and in experiments within the 5GINFIRE and Fed4FIRE+ projects. He is currently working as a Full Stack Developer with FirLab s.r.l., Firenze. His current research interests include distributed and cloud computing, network and service management, and Web and mobile applications.