HAL open science

Taming internet of things application development with the IoTvar middleware

Pedro Victor Borges Caldas da Silva, Chantal Taconet, Sophie Chabridon, Denis Conan, Everton Cavalcante, Thais Batista

▶ **To cite this version:**

Pedro Victor Borges Caldas da Silva, Chantal Taconet, Sophie Chabridon, Denis Conan, Everton Cavalcante, et al.. Taming internet of things application development with the IoTvar middleware. ACM Transactions on Internet Technology, 2023, 2, pp.1533-5399. 10.1145/3586010 . hal-04033054

HAL Id: hal-04033054
https://hal.science/hal-04033054v1

Submitted on 16 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Taming Internet of Things Application Development with the IoTvar Middleware

PEDRO VICTOR BORGES, CHANTAL TACONET, SOPHIE CHABRIDON, and DENIS CONAN, SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France
EVERTON CAVALCANTE and THAIS BATISTA, Federal University of Rio Grande do Norte, Brazil

In the last years, Internet of Things (IoT) platforms have been designed to provide IoT applications with various services such as device discovery, context management, and data filtering. The lack of standardization has led each IoT platform to propose its own abstractions, APIs, and data models. As a consequence, programming interactions between an IoT consuming application and an IoT platform is time-consuming, error prone, and depends on the developers' level of knowledge about the IoT platform. To address these issues, this paper introduces *IoTvar*, a middleware library deployed on the IoT consumer application that manages all its interactions with IoT platforms. IoTvar relies on declaring variables automatically mapped to sensors whose values are transparently updated with sensor observations through proxies on the client side. This paper presents the IoTvar architecture and shows how it has been integrated into the FIWARE, OM2M, and muDEBS platforms. We also report the results of experiments performed to evaluate IoTvar, showing that it reduces the effort required to declare and manage IoT variables and has no considerable impact on CPU, memory, and energy consumption.

CCS Concepts: • **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → *Distributed computing methodologies*; • **Software and its engineering** → *Application specific development environments.*

Additional Key Words and Phrases: Middleware, Internet of Things, Software abstractions, IoT platforms

1 INTRODUCTION

The Internet of Things (IoT) envisions a network of pervasive smart objects able to interact with each other through the Internet and perform several tasks, such as processing, capturing environmental variables, and reacting to external stimuli through the Internet. Despite several research and development advances in the last years, the IoT paradigm still presents many issues to address, in particular those related to application development and the high heterogeneity resulting from the inherent diversity of hardware and software technologies.

Many research efforts have been invested in facilitating the integration of IoT resources and services to provide software-defined distributed services [46]. One of these efforts culminated in developing IoT middleware to improve the support for developing IoT applications. These proposals aim to abstract the specificities of physical devices from end-users/applications, promote interoperability, and ease application development [7, 24, 27, 36].

Designing and implementing IoT applications is complex as it addresses different concerns, such as adequately identifying various stakeholders' roles at the different application development phases, heterogeneity in IoT systems, and handling a massive amount of heterogeneous data from disparate devices [33]. Furthermore, the immaturity, lack of convergence, and significant fragmentation of the IoT scenario hinder the development

Authors' addresses: Pedro Victor Borges, pedro.borges@telecom-sudparis.eu; Chantal Taconet, chantal.taconet@telecom-sudparis.eu; Sophie Chabridon, sophie.chabridon@telecom-sudparis.eu; Denis Conan, denis.conan@telecom-sudparis.eu, SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Évry and Palaiseau, France; Everton Cavalcante, everton.cavalcante@ufrn.br; Thais Batista, thaisbatista@gmail.com, Federal University of Rio Grande do Norte, Natal, Brazil.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1533-5399/2023/2-ART $15.00
https://doi.org/10.1145/3586010

Just Accepted

of applications that work across different physical devices and platforms, thus leading them to largely be vendor/platform- and hardware-specific [43].

Developers can rely on IoT platforms that facilitate the communication and data flow between IoT applications and devices. These platforms provide interfaces, interaction patterns, communication protocols, and computational capabilities to support application development. They also include services that provide functionalities such as device discovery, context management, and data analysis [35]. However, developing an IoT application remains challenging, even with the support of IoT platforms and middleware. For example, to display the current temperature at a given location, application developers have to program the interactions with an IoT platform that shows up several virtual entities providing updated temperature data around the area. Even if most of the platforms provide similar features, developers need to learn, for each platform, specific APIs, data models, and communication protocols. Other development tasks include selecting and handling the appropriate interaction pattern (e.g., request/reply or publish/subscribe), (un)marshalling data, and manipulating sensor identifiers and metadata, such as quality attributes of sensor data. Furthermore, they may have to tune the frequency of interactions to limit the usage of computational resources.

The Web of Things (WoT)[1] approach has been used since the dawn of the IoT paradigm as means of (re)using standardized Web technologies (such as the HTTP protocol and the REST architectural style), thing descriptions describing metadata and interfaces of IoT objects, and binding components for interacting through different protocols [34]. Nonetheless, the concepts of WoT need to be further complemented to address software engineering issues when developing IoT applications out of the Web.

Our previous work [4] introduced the IoTvar middleware, which abstracts the interaction with IoT platforms and lower the development cost of IoT applications in terms of lines of code. This enables for an easier integration of IoT applications and IoT devices. To provide this integration, the middleware introduces the concept of *IoT variables*. An IoT variable is a variable that has an observation attribute representing a value measured by a sensor in the environment. IoT variables are managed by IoTvar[2], an IoT middleware library that transparently updates the observation attribute. For this purpose, IoTvar provides proxies that manage all the interactions with IoT platforms.

Our previous work [4] also reported a first evaluation of IoTvar, with only one platform. This paper extends such work with the following contributions. First, we analyze the features of three IoT platforms that allow virtualizing sensors, discovering them, and providing context data to the applications. We analyzed FIWARE [13], OM2M [1], and muDEBS [23], and identified the common features of these IoT platforms to provide an architecture that can support multiple platforms and be easier to use from the perspective of application developers. Second, we enhanced the architecture of IoTvar towards the easy integration of new IoT platforms and instantiated it for those three platforms. In this architecture, we show how to integrate connectors to the three IoT platforms previously mentioned. Finally, we report computational experiments to assess the impact of IoTvar on energy, memory, and CPU consumption when integrating it with the analyzed platforms.

The remainder of this paper is organized as follows. In Section 2, we motivate our study by presenting works related to IoTvar. Section 3 presents an analysis of the three IoT platforms currently connected with IoTvar. Next, Sections 4 and 5 detail the IoTvar and evaluate it in terms of CPU, memory, and energy consumption. In Section 6, we conclude and propose future works.

Just Accepted

[1]https://www.w3.org/WoT/
[2]https://gitlabev.imtbs-tsp.eu/m4iot/iotvar

2 RELATED WORK

The IoT research area comprises many protocols, standards, and platform specifications, and several works propose generic abstractions to ease IoT application development. Existing proposals mainly focus on domain-specific languages (DSLs), application mashups, IoT middleware, and proxies.

Many domain-specific dedicated programming languages are used to provide application abstractions for inexperienced developers. More specifically, some middleware solutions provide their own DSL to provide abstractions for building IoT applications. Boujbel et al. [5] present MuScADeL (MultiScale Autonomic Deployment Language), a DSL that enables designers to declare multiscale deployment properties without precise knowledge of the deployment domain. Delicato et al. [8] propose a Web mashup DSL specifically tailored for wireless sensor networks (WSNs). Web mashups allow ad-hoc Web applications to be built upon the combination of real-time information (data, presentation, and functionality) through the composition of available services, such as publishing and discovering the capacities of available WSNs. Instead of proposing a new DSL, IoTvar comes up as a middleware library to be used by developers.

Web mashups foster easy, fast integration of data sources to produce augmented results that were not considered when producing raw data [3, 32]. Wirecloud [45] is an end-user Web-centered application mashup platform aimed at enabling end-users with no programming skills to create Web applications and dashboards/cockpits, e.g., to visualize their data of interest or to control their smart home or environment. Node-RED [28] allows wiring together hardware devices, APIs, and online services, and application elements can be saved or shared for reuse. Application mashups have advantages over a user-friendly graphical development with a fast composition, but they are limited to the graphical components and their configurations through a Web browser. On the other hand, IoTvar is a library that does not limit its usage and can be inserted in multiple application parts.

Eclipse Thingweb node-wot[3] is an implementation of the Web of Things that provides a browser bundle to visualize Things Descriptions (TDs), which are abstractions of physical or virtual entities, and to enable the interaction with IoT objects through the Web browser. Protocol bindings exist there to enable a TD to be adapted to a specific protocol, data payload formats, or platforms that combine both in specific ways. A scripting API represents an interface for scripts to discover and operate IoT objects, and to expose locally defined things. While Eclipse Thingweb is proposed to be a mediator between the application and the things, IoTvar is intended to be deployed at the application side as a library providing abstractions to interactions with IoT platforms. IoTvar also comes with the notion of IoT variables, in which sensors are be represented in the source code as a variable that updates sensor information automatically through a proxy. On the other hand, the interface of Eclipse Thingweb with IoT objects happens through protocol bindings such as HTTP and CoAP.

Most IoT middleware solutions work on the IoT device network side. We present here two of these middleware solutions that aim to master application energy consumption. Jeon and Jung propose Mint [17], a middleware that provides a high-level abstraction for IoT devices in the form of an API. Mint further provides integration with three layers (sensor abstract layer, system layer, and interaction layer), and aims to enhance the energy efficiency and performance of IoT devices through performance improvement offered by leveraging resource management and request processing. The main difference between Mint and IoTvar is that the former abstracts communication directly with IoT devices, while IoTvar proposes to abstract communication with IoT platforms. Kalbarczyk and Julien [18] propose Omni, a middleware offering an abstraction for discovering neighboring devices with their services, sharing profiles with nearby users, and interacting with beacons in a smart city. Omni uses a mix of Bluetooth and Wi-Fi technologies to discover neighboring devices and transfer data. Omni is similar to IoTvar as it is deployed on end-user appliances, but IoTvar is not limited to short-range networks and allow communication with other devices outside of the neighborhood.

[3]https://www.thingweb.io/

Just Accepted

IoT applications require a significant number of protocols to meet users' requirements and provide the required features. One way to reduce the complexity of using these protocols on the client side is to use the Proxy pattern [15]. Soundararajan and Robert [41] show how the Proxy pattern can help to implement a benchmarking application. They provide a client-side Proxy pattern that hides the complexity of protocols and encapsulates the knowledge of how to contact the servers. Similarly, Sutra et al. [42] build the CRESON system that creates a proxy on the client side to contact databases over the Internet. It abstracts the communication protocols involved while preparing to transfer and process data required by the application. The role played by the Proxy pattern in application development inspired the approach followed by IoTvar to manage the interactions with several IoT platforms deal with the choice of the interaction pattern,and the frequency of the updates, and abstract protocols for (un)marshalling data.

3 IOT PLATFORMS: THEIR ROLE IN THE IOT ECOSYSTEM AND THEIR FEATURES

A new generation of applications such as smart buildings, smart cities, agriculture, logistics, and supply chain manufacturing shares common requirements such as interacting with IoT devices according to various IoT protocols [2]. In addition, they should all offer extra-functional requirements such as fast response time, low energy consumption, availability, reliability, security, and high throughput. In this context, IoT platforms supporting their deployment is a recent trend: they provide services to deploy and run applications on the top of a hardware or software suite [25].

IoT platforms have been defined as the "middleware and the infrastructure that enable the end-users to interact with smart objects" [24], besides providing services such as programming frameworks, machine-to-machine (M2M) integration, data and device management, security and storage [40]. For this work, we consider context management IoT platforms, which represent a subset of IoT platforms. Context management platforms provide virtualization and abstraction of the sensors (a virtual, software-based representation of a sensor), sensor discovery, and provision of context data to applications through APIs and different interaction patterns.

We have selected three IoT platforms available as open-source projects. FIWARE/Orion and OM2M are widespread platforms used by the community and comply with IoT standards. muDEBS is a research platform with some interesting characteristics: it is a distributed event-based system that combines a semi-structured data model with content-based filtering. Although these platforms are different in many aspects, they share common features. This section presents an analysis of those features aiming at identifying the components that should be provided by an IoT middleware to interact with IoT platforms and which abstractions should be provided to IoT client applications.

3.1 Overview of FIWARE, OM2M, and muDEBS

FIWARE[4] is a generic, open-source platform supported by the European Community. It provides many extensible, reusable, interoperable components to allow for easy system development in different application domains, the so-called Generic Enablers (GEs). FIWARE encompasses GEs for context entity management, device management, historical data storage, event processing, security, and the creation of dashboards. The main FIWARE GE is the Orion Context Broker (or simply Orion), which is the context management IoT platform. As we focus on context management IoT platforms, this paper will refer to it as FIWARE/Orion from now on.

OM2M is an open-source implementation of the oneM2M standard [31] adopted by the Eclipse IoT Working Group. It provides an M2M service that can develop services independently of the underlying network. OM2M provides a horizontal Service Common Entity (CSE) that can be deployed in an M2M server, a gateway, or a device. Each CSE provides application enablement, security, triggering, notification, persistence, device inter-working,

[4]https://www.fiware.org

Just Accepted

and device management. The platform exposes an API providing many other services, such as resource discovery, application registration, and context data management.

muDEBS (Multiscale Distributed Event-Based System) is a research IoT platform designed to disseminate data in large-scale and heterogeneous systems involving clouds, cloudlets, desktops, laptops, mobile phones, and IoT smart objects. muDEBS introduces multi-scoping for limiting the broadcasting of subscription filters and enabling to forward notifications only to relevant scopes of the overlay network of brokers. The platform brings into play producer and consumer contracts. To implement localized scalability, broker nodes form an overlay that supports cycles and delimits scopes. Therefore, filters and data are tagged with scoping metadata that are specified in the producer and consumer contracts, and they are then broadcast only to relevant scopes, not to the entire system.

3.2 Context data model

IoT is characterized by a high degree of heterogeneity because of the many available technologies [21]. Devices from different manufacturers expose their data in various data models and use different protocols. A challenge for an IoT platform is to provide a unified data model. The model must include location data and quality attributes (e.g., freshness, resolution). Furthermore, a well-defined data model facilitates the development of wrappers and adapters to parse the data in IoT applications. We present below the context data models used by the three studied IoT platforms.

3.2.1 FIWARE/Orion data model. FIWARE/Orion follows the Next Generation Service Interface (NGSI) data model [26] to standardize information exchange and allow for component interoperability. In NGSI, information is structured in a generic way through entities that can represent physical and virtual elements such as a building, a car, sensors, or actuators. An NGSI *entity* has an identifier, a type, and a list of attributes. *Attributes* have a name, a type, a value, and a list of *metadata*, with the same data structure as the attributes. Figure 1 depicts all these elements.

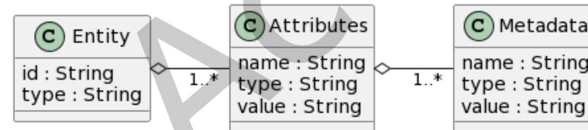


Fig. 1. Next Generation Service Interface (NGSI) entity model used by FIWARE/Orion.

3.2.2 OM2M data model. OM2M conforms to the data model of the oneM2M specification. Figure 2 shows the generic model used in the OM2M platform. The model contains a *resource type* that may include children resource types and *resource specific attributes*, each one with its own value. There is no limit to what data can be included in the data model. For instance, quality attributes and location data can also be included as named attributes.

3.2.3 muDEBS data model. The muDEBS platform uses a semi-structured data model "à la" XML and allows XPATH expressions in scripts written in JavaScript. Devices and clients can exchange any data. Clients publish contracts to brokers connected through the IoT platform, and any device with data interesting for the IoT application is forwarded to applications. Figure 3 shows the data model used by muDEBS to represent entities. A context *report* aggregates a set of context *observations*, with a *context observable* defining what is observed, e.g., a temperature sensor. The observable is linked to a *context entity* that can have a *relation* with other entities. The model offered by the muDEBS platform also makes it possible to provide localized data and quality attributes. When the contract sent by the client contains the need for a device to send the location and a given quality of the attributes, brokers can filter out context reports that do not contain these data.

Just Accepted

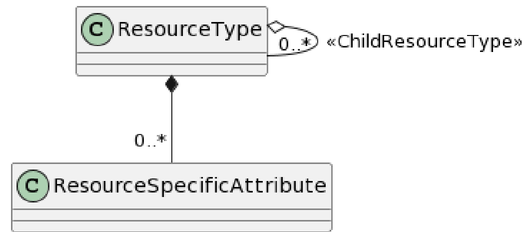


Fig. 2. OM2M generic data model.

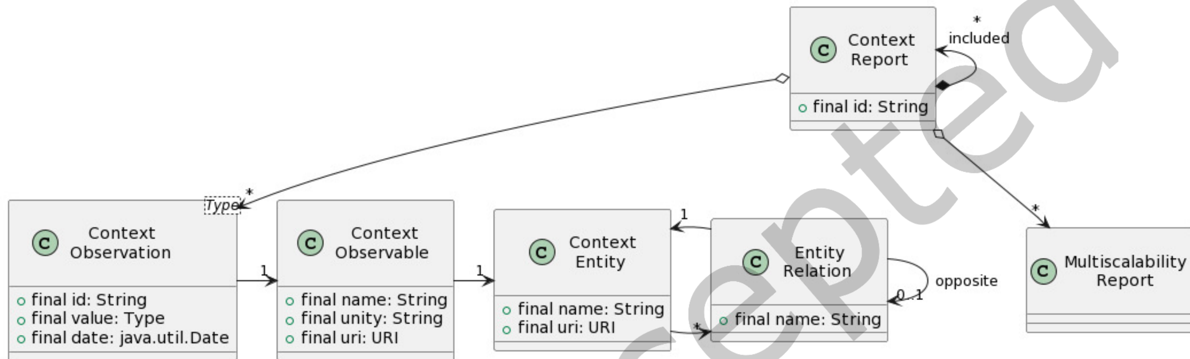


Fig. 3. muDEBS generic data model.

Just Accepted

3.3 Interaction patterns, APIs, and protocols

The communication among the many components of an IoT system is crucial. IoT platforms commonly provide two interaction patterns: synchronous requests or publish/subscribe. In the synchronous pattern, a client application consumes IoT data by sending a request to the IoT platform, which replies to the request. In the publish/subscribe pattern, a consumer application registers to the IoT platform and asynchronously receives publications when they are available [37]. In addition, all the IoT platforms propose APIs with a set of functions used by client applications and protocols for the exchanges. This section presents the interaction patterns, APIs, and protocols provided by the three studied IoT platforms.

FIWARE/Orion provides both synchronous and publish/subscribe interaction patterns. The platform provides RESTful APIs for interactions with context producers and IoT application consumers [12]. HTTP is used both for synchronous and publish/subscribe interactions. When using the synchronous pattern, application developers make HTTP requests to the API following a request/response behavior. On the other hand, for the publish/subscribe interaction pattern, the consumer application uses the API to subscribe to content and waits for HTTP requests from the broker to receive data.

OM2M also provides the two interaction patterns. The synchronous pattern uses a RESTful API so that the client application makes HTTP requests and receives data [30]. The publish/subscribe pattern uses the MQTT protocol [29]. A client application subscribes to topics using the MQTT protocol, and the platform sends publications to the subscribed client applications. OM2M benefits from MQTT characteristics such as different qualities of services (from "at least once" to "exactly once"), clear sessions, and retain flags for managing disconnections.

muDEBS provides only the publish/subscribe interaction pattern through an ad hoc API. The synchronous mode is emulated by providing a client application with the data that producers have last produced. These data are returned to the consumer when the subscription filter of the consumer is installed onto access brokers. Such brokers manage direct connections to producers.

3.4 Discovery facilities and filtering capabilities

IoT platforms include services to find context producers and filter the data these producers send. Filtering data ensures better control over the quality and the amount of data received by applications, avoiding unnecessary interactions with the IoT platform. This section presents the facilities provided by the three IoT platforms.

Using FIWARE/Orion, application developers make HTTP requests to the API by providing parameters such as identifier, type, and attributes to select entities. The filtering capability can use both a Simple Query Language or geographical queries. An easy-to-use query to filter out based on values of attributes would be like `q=temperature>40;humidity>40`. A geographical query to retrieve all the entities located closer than five kilometers from a given point can be expressed by `georel=near;minDistance:5000&geometry=point&coords=-40.4,-3.5`.

The discovery functionality in OM2M is implemented using an HTTP GET request passing parameters such as `fu`, which stands for "filter usage", and `lbl`, which stands for "label". For instance, to get a temperature sensor, the application would use `fu=1&lbl=Type/sensor`, where `fu=1` indicates that it is a discovery request. The discovery functionality also retrieves the set of resource URIs matching specific filter criteria. OM2M cannot filter data sent by sensors, thus leaving a gap for unneeded data to be sent to the application. Nonetheless, when using the publish/subscribe pattern, the application can specify the maximum frequency of notifications.

muDEBS does not need any discovery functionality because context reports describing observable entities and filters are content-based, i.e., subscription filters can select the entities from which data are to be received (e.g., temperature greater than 20 degrees Celsius). In addition, the muDEBS filtering functionality is extended into producer and consumer contracts to include quality of data [23] and privacy concerns [9], e.g., good precision of air quality data only, or only authorized end-users. Brokers ensure that only the relevant data are forwarded to consumers.

3.5 Synthesis of the IoT platforms analysis

Table 1 summarizes important features that the FIWARE/Orion, OM2M and muDEBS platforms bring into play. These features are: (i) *interaction pattern*, either synchronous or publish/subscribe; (ii) *application protocols*, showing how many and what are the protocols supported by the platform for applications to communicate with; (iii) *data models*, the standards used by the platforms to send/receive data from/to applications; (iv) *discovery service*, the process of automatically finding appropriate services and their providers by taking into consideration the context and Quality of Service (QoS) of requests; and (v) *filtering capabilities*, indicating how the platform can refine data sent/received from sensors, devices, and applications.

In summary, the FIWARE/Orion, OM2M, and muDEBS platforms provide different data models even though they share a typical structure in some parts, i.e., attributes and their metadata. The IoTvar architecture hence provides specific data unmarshallers for each IoT platform. Furthermore, the IoT platforms each have their own way of communicating. FIWARE/Orion relies on HTTP to do synchronous and publish/subscribe communication, OM2M relies on HTTP for synchronous communication and MQTT for publish/subscribe communication, and muDEBS uses its own protocol based on the AMQP specification. With each IoT platform having its communication protocol, the effort to develop IoT applications increases as it is more time-consuming to understand and adapt to each protocol. An abstraction on the top of these different communication protocols like IoTvar could be provided to reduce the effort from developers to implement the interaction with IoT platforms in their applications.

Just Accepted

Table 1. Comparison between FIWARE/Orion, OM2M, and MUDEBS

Characteristics	Platforms		
	FIWARE/Orion	OM2M	MUDEBS
Interaction patterns	Synchronous, publish-subscribe	Synchronous, publish/subscribe	Publish/subscribe
Application protocols	HTTP	HTTP, MQTT	Ad-hoc
Data models	NGSI	oneM2M	Ad-hoc, semi-structured
Discovery service	Yes	Yes	Yes
Filtering capabilities	Yes	No	Yes

4 IOTVAR

IoTvar is an IoT middleware library designed to abstract the interactions of client IoT applications with context management IoT platforms. The main concern is offering an abstraction level that enables developers to reduce the number of lines of code required to discover and interact with virtualized sensors.

With IoTvar, application developers define IoT variables in their source code, i.e., a variable with an observation attribute representing sensor data [4]. To transparently update data from IoT platforms, IoTvar provides proxies that handle the interactions between IoT variables and IoT platforms. Figure 4 illustrates the integration of IoTvar with IoT platforms.

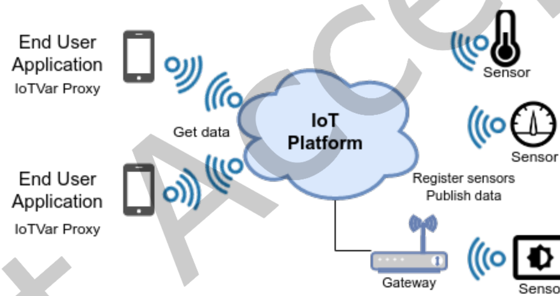


Fig. 4. Integration of IoTvar in an IoT system distributed architecture.

This section is structured as follows. In Section 4.1, we describe how IoTvar is used by IoT consumer application developers. In Sections 4.2 and 4.3, we detail the architecture of IoTvar and how to integrate a new IoT platform to it.

4.1 Using IoTvar

IoTvar relies on IoT variables to enable the interaction with IoT platforms, so that specific IoT variable is declared in the source code for each supported platform. Figure 5 shows a class diagram of IoTvar with its main classes and interfaces. The `IoTVariableCommon` class contains common attributes to be shared by all the IoT variables corresponding to the supported platforms. `IoTVariableFiware`, `IoTVariableMuDEBS`, and `IoTVariableOM2M` are classes that specialize the `IoTVariableCommon` class for each platform, thus inheriting the common attributes and implementing those that are specific for each platform. Moreover, data exchanged between the components and coming from the platforms are generically represented by the `Observation` class.

Just Accepted

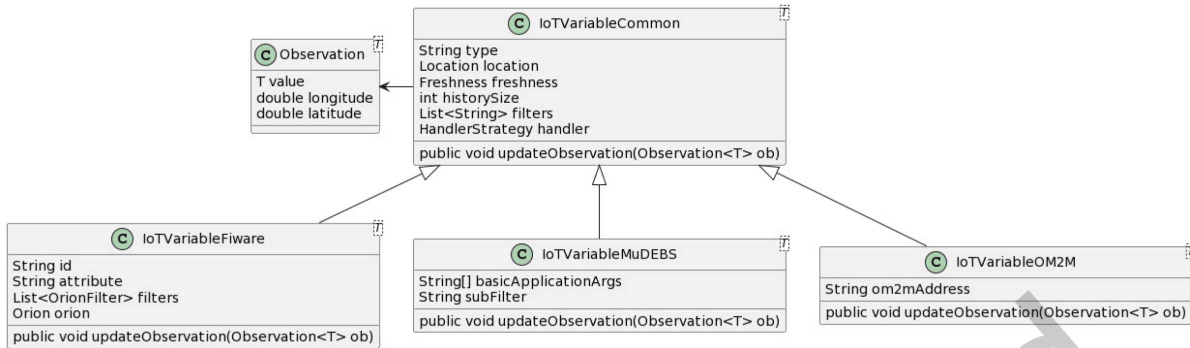


Fig. 5. IoTvar implementation elements

Listing 1 shows an excerpt of code in Java programming language using IoTvar. This code declares an IoT variable to display the up-to-date temperature gathered from a sensor in the vicinity of the Eiffel Tower in Paris, France. To declare an IoT variable, the developer provides IoTvar with: the identifier (line 2) and type (line 3) of the searched-for sensor (`Temperature` in line 4); the location of the Eiffel Tower with latitude, longitude, and radius in meters (line 5); the required refresh time as a quality parameter (line 6); the size of the local history of values (line 7); additional filters to be provided to the platform (e.g., `temperature>10;humidity>10` in line 7); the configuration parameter of the IoT platform (line 7); the interaction strategy with the IoT platform (line 8), and the class to be used for unmarshalling purposes (line 9).

Listing 1. Declaring a variable using IoTvar.

```
1  IoTVariableFiware<Integer> temperatureEiffelTower = new IoTVariableFiware<>(
2      "temperature_eiffel_tower_310",     // ID
3      "LM35",                             // Type
4      "Temperature",                      // Attribute
5      new Location("location", 48.8582602, 2.29449905431968, 100.0),
6      new RefreshTime(10, TimeUnit.SECONDS),
7      10, null, orionConfiguration,       // History size, filters, and
           platform config.
8      HandlerStrategy.SYNC                // or PubSub
9      Integer.class);
10 temperatureEiffelTower.registerIoTListenerP(display);
```

Just Accepted

The use of an IoT variable requires registering a listener to it (see line 10 in Listing 1), which is automatically activated when the observation is updated, i.e., either when a refresh has been requested (in the synchronous interaction pattern) or when a notification has been received (in the publish/subscribe interaction pattern). In IoTvar, an `IoTVarObserver` object represents a generic listener to be registered to an IoT variable. Therefore, a specific listener is codified by implementing this interface and the methods it defines. Listing 2 displays the basic interface for an observer with the `onUpdate}` and `updateIssue` methods: the former is called each time a new observation is provided by IoTvar and the latter is called when the update cannot comply with the specified refresh time constraint. The `updateIssue` method enables the developer to handle an error when no update is received, e.g., if the IoT platform is unreachable.

Listing 2. Interface of a listener.

```
1 public interface IoTVarObserver {
2   void onUpdate(Observation newObservation);
3   void updateIssue(String issue);
4 }
```

Listing 3 presents the code of `TextDisplay`, a simple observer class which implements the `IoTVarObserver` interface and displays the temperature around the Eiffel Tower. In this example, the observer logs the observation with the received temperature (line 4) and then logs any error, e.g., networking error, data error, etc. (line 7).

Listing 3. Declaration of a listener.

```
1 public class TemperatureDisplay<Meteo> implements IoTVarObserver {
2    private static final Logger logger = LogManager.getLogger(TextDisplay.class);
3    public void onUpdate(Observation newObservation) {
4       logger.info("Current temperature around the Eiffel Tower: " +
            newObservation);
5    }
6    public void updateIssue(String issue) {
7       logger.info("There was an error updating the sensor: " + issue);
8    }
9 }
```

Table 2 shows the differences regarding the number of lines of code that application developers need to write to directly use the APIs of FIWARE/Orion, OM2M, or muDEBS in comparison to using IoTvar. These amounts of code refer to implementing a behavior similar to the one presented in Listing 1, i.e., to gather information from a sensor close in a given location.

Table 2. Number of lines of code when developing with and without IoTvar

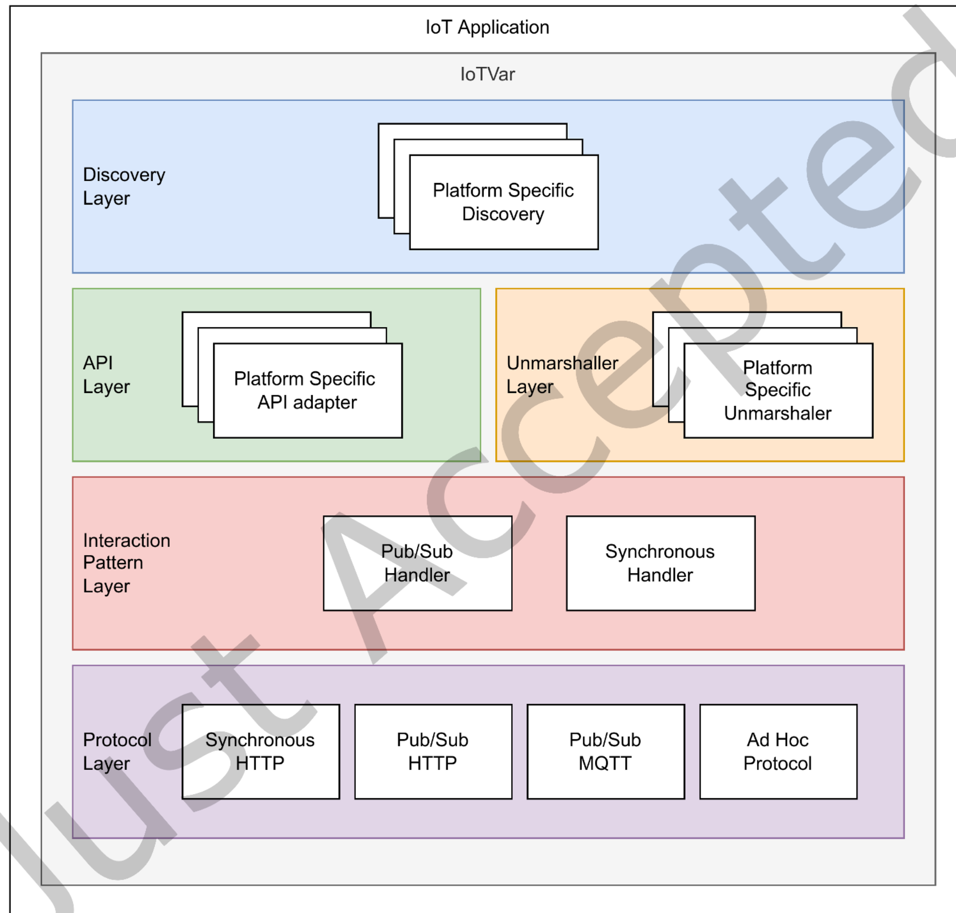
Platform	Interaction pattern	Lines of code	
		with IoTvar	without IoTvar
FIWARE/Orion	Synchronous	15	450
	Publish/subscribe	15	600
OM2M	Synchronous	15	400
	Publish/subscribe	15	200
muDEBS	Publish/subscribe	15	450

4.2 IoTvar Architecture

IoTvar is architected to ease the integration of new IoT platforms by exposing an interface implemented for the three currently supported platforms. It also proposes IoT variables that are a proxy [38] representing an entity of an IoT platform. The proxy activates the handler in charge of communicating with the IoT platform. When the proxy receives an updated observation, it unmarshalls the received data and updates the observation attribute of the corresponding IoT variable.

Just Accepted

Figure 6 shows the main components of IoTvar, with several components organized in layers to manage the interactions between client applications and IoT platforms. The *Protocol Layer* is shared by all the platforms and defines components implementing the respective platform protocols to interact with the client applications, such as HTTP and MQTT. The *Interaction Pattern Layer* provides components to manage connections with IoT platforms and enables IoTvar to handle the interactions depending on the patterns supported by the platform. The upper layers are platform-specific. The *API Layer* and the *Unmarshaller Layer* work as a glue that maps the API calls and data structures to the IoT platforms. The *Discovery Layer* provides functionalities for IoT applications to discover devices and manage filtering mechanisms, such as device discovery and data gathering.



Just Accepted

Fig. 6. IoTvar generic architecture.

4.3 Extending IoTvar

The current version of IoTvar supports only the HTTP, MQTT, and muDEBS protocols, which are the ones used by the IoT platforms integrated with it so far. To support a new platform with a different protocol, the *Protocol Layer* of IoTvar can be extended to provide such support. Figure 7 depicts the steps to integrate a new platform

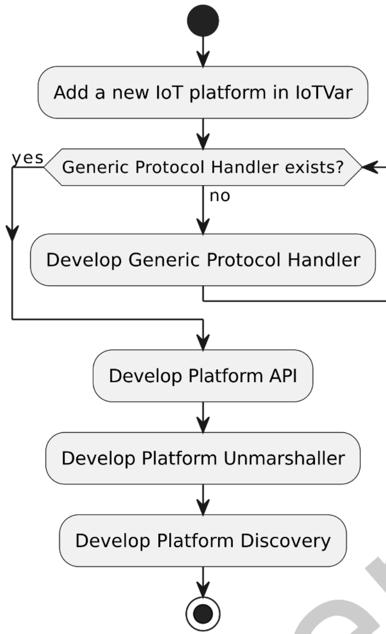


Fig. 7. Steps to extend IoTvar towards supporting new IoT platforms.

Just Accepted

with IoTvar. If there is no support for a protocol in use by the platform, the developer must provide a generic implementation for it. As each IoT platform may have its own data model and API, a specific data unmarshaller and an API adapter are required to translate the protocol and communication of the IoT platform to Java objects within IoTvar. This Java object is then used by a specific platform discovery component to provide information for the application via the `IoTVarObserver` common interface.

Table 3 details the number of lines of code per component of IoTvar. The number of lines needed to add an extension varies from one platform to another. IoTvar's muDEBS components were implemented with 199 lines of code while the ones for OM2M and FIWARE/Orion were implemented using 346 and 744 lines of code, respectively. The implementation of the muDEBS' data model along with its processing mainly comes from a library external to IoTvar, leaving most of the implementation in IoTvar mainly for communicating with the platform and hence reducing the number of lines of code required to integrate it with IoTvar. On the other hand, FIWARE/Orion and OM2M needed to implement the data model and interactions, thereby being more costly in terms of lines of code. Furthermore, the generic parts of IoTvar needed to provide the foundation for the platform-specific components, such as the *Synchronous HTTP* and the *Synchronous Handler*, with 154 and 139 lines of code. This is similar to the PubSub components of the *Interaction Pattern Layer* and the *Protocol Layer*, which respectively required 238 and 202 lines of code.

The generic solution IoTvar provides, across its architectural layers, is mainly possible thanks to the proxy pattern. Security in IoTvar relies on the availability of security mechanisms in the supported IoT platforms. For example, FIWARE/Orion provides a suite of GEs for this purpose, oneM2M defines security Common Service Functions, and muDEBS uses privacy contracts to control the dissemination of information according to application requirements. A new vertical layer encompassing the layers already present in IoTvar needs to be implemented to integrate these security mechanisms.

Table 3. Number of lines of code to implement each IoTvar component.

Layer	Component	Lines of code			
		Generic	muDEBS	OM2M	Orion
Discovery Layer	Platform-specific discovery	-	97	98	406
API Layer	Platform-specific API adapter	-	102	198	185
Unmarshaller Layer	Platform-specific unmarshaler	-	-	50	153
Interaction Pattern Layer	Synchronous Handler	235	-	-	-
	Pub/Sub Handler	154	-	-	-
Protocol Layer	Synchronous HTTP	139	-	-	-
	Pub/Sub HTTP	238	-	-	-
	Pub/Sub MQTT	202	-	-	-
	Ad Hoc Protocol	-	50	-	-
Total		917	249	346	744

Just Accepted

5 EVALUATION

This section reports a quantitative evaluation of IoTvar and its integration within the FIWARE/Orion, OM2M, and muDEBS platforms. This evaluation comprises a performance assessment that considers the same application written with and without IoTvar (i.e., directly accessing the IoT platform), aiming at measuring the overhead in terms of CPU, memory, and energy consumption. The application goal is to receive and display in the standard output the data returned from a temperature sensor in the vicinity of the Eiffel Tower. The tests done with this application varied the number of sensors (one sensor is one variable declared in the code) from 25 to 200 in a 25 sensors step, with a refresh time of one second for each sensor, and having a small local history of the ten latest values for each sensor.

Section 5.1 describes data collection and analysis procedures. Section 5.2 describes the computational infrastructure used for the experiments. Section 5.3 presents the results for the synchronous and publish/subscribe interaction patterns. Section 5.4 provides an overview of the results and conclusions upon the statistical analysis. Section 5.5 discusses a validity analysis of the performed evaluation.

5.1 Data Collection and Analysis Procedures

In the experiments, we collected performance measurements (CPU, memory, and energy consumption) by wrapping both the method called by the synchronous handler and the method to handle notifications sent by FIWARE/Orion, OM2M, and muDEBS. This has been implemented using AspectJ [20], which provides an encapsulation around the main methods of the IoTvar structure and is woven into the IoTvar code.

Data for each metric (CPU, memory, and energy) were collected during 30 executions of a five-minute test. We did not record the first minute of the test for warm-up purposes, ensuring that class loading is complete in the Java Virtual Machine (JVM) to avoid any interference in the results [14]. The last four minutes corresponded to the effective run phase. Afterwards, data were gathered into a single file composed of (i) the sum of the CPU used by the methods that process data in microseconds, (ii) the average memory used by the application in megabytes, and (iii) the energy consumed in Joules.

We used hypothesis testing to assess if the overhead caused by IoTvar is statistically significant. First, we used the Shapiro-Wilk test [39] to verify normality in data. Next, we used the Student's t and Mann-Whitney tests as alternatives for hypothesis testing depending on whether data follow a normal distribution or not [11]. A p-value

smaller than a significance level α indicates that IoTvar indeed causes a statistically significant overhead. We adopted $\alpha = 0.05$ as significance level for all these tests.

A p-value returned by a hypothesis test can indicate statistically significant differences, but it does not say how much, albeit minimal [10, 19]. Therefore, we calculated the effect size as means of strengthening the claims from the previous statistical tests and easing interpretation of results. The effect size was measured by the Vargha-Delaney's A measure [44] and interpreted according to the following levels of magnitude defined by Hess and Kromrey [16]: *negligible*, *small*, *medium*, and *large*.

5.2 Experimental Setup

Figure 8 illustrates the setup of the tests. The computer executing the client application consuming IoT data had an Intel® Core™ i7-8665U processor, 32 GB of RAM, and Debian 9 as the operating system. The server computer executing the IoT platform and the data producers, which simulated IoT sensors sending data to the platform, had an Intel® Core™ i7-4770K processor, 16 GB of RAM, and Linux Ubuntu 16.04 as the operating system. YoctoPuce Yocto-Watt wattmeter was plugged into the client computer to collect energy-related measures. The client application communicated with the server through a locally isolated Wi-Fi network.

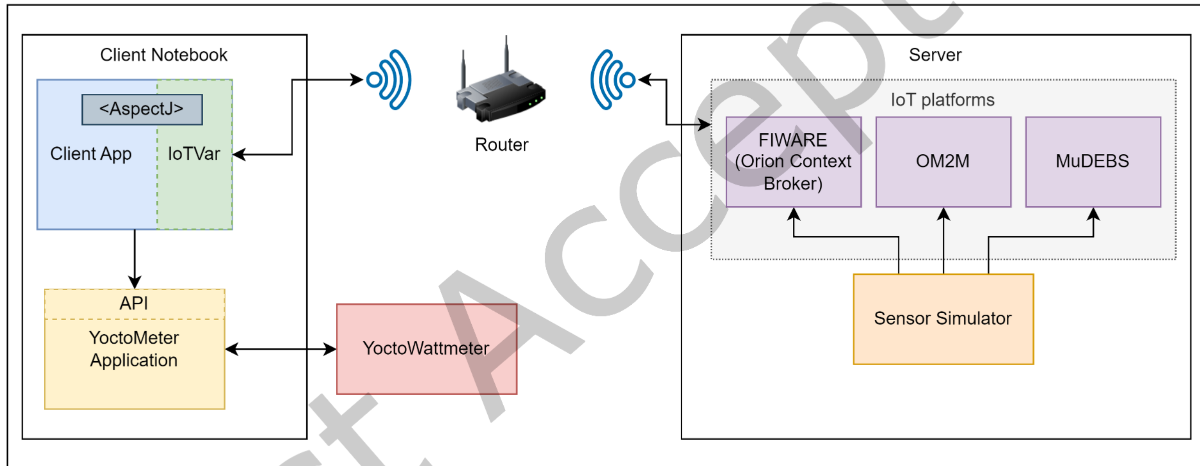


Fig. 8. IoTvar experimental setup.

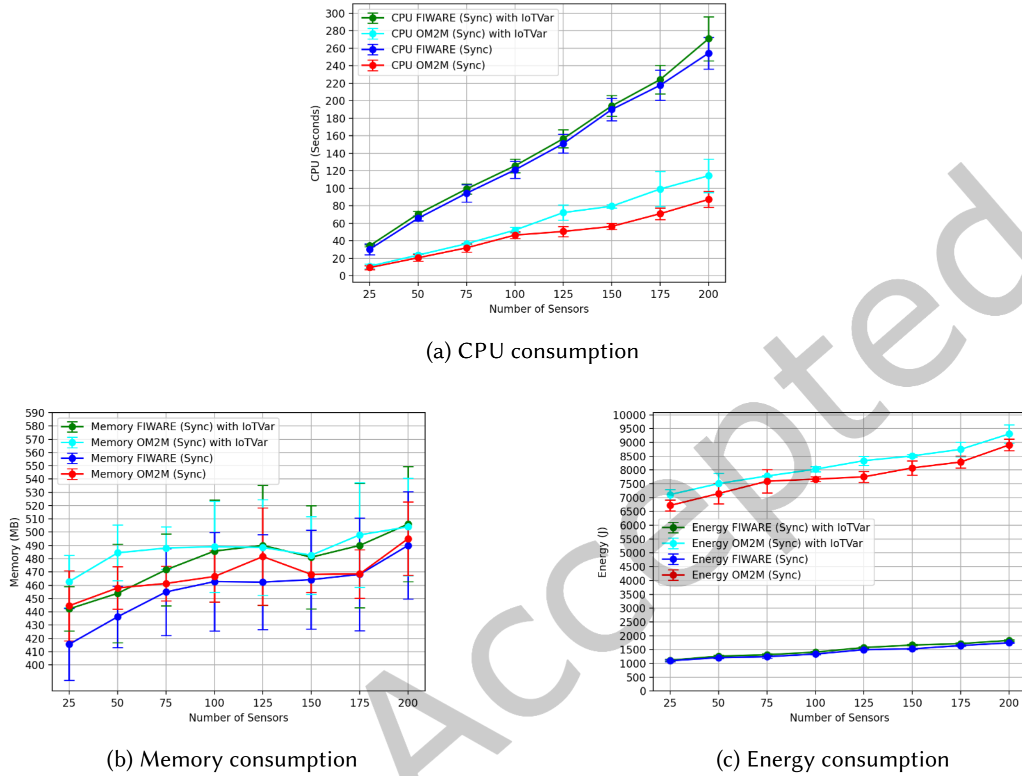
Just Accepted

The different versions of the IoT application (with or without IoTvar) get sensor data through synchronous calls or publish/subscribe notifications. When using IoTvar and synchronous calls, the application creates an IoT variable for each sensor, and IoTvar sends one request per second to get data for each declared sensor. When using publish/subscribe notifications, the application creates an IoT variable for each sensor, IoTvar registers to the corresponding entity, and the IoT platform notifies IoTvar about all the updates.

5.3 Results

5.3.1 Synchronous interaction pattern. Figure 9 shows the CPU, memory, and energy performance of synchronous calls for the FIWARE/Orion and OM2M platforms, which support the synchronous interaction pattern. IoTvar increases the demand for CPU and memory because more processing is necessary to handle the threads pool and maintain the collection of IoT variables (search, update, history). Consequently, the energy consumption with

IoTvar is greater than without IoTvar due to the overall CPU and memory usage to provide a useful abstraction to application developers. Table 4 shows the difference (in percentages) between using or not IoTvar.



(a) CPU consumption

(b) Memory consumption

(c) Energy consumption

Fig. 9. CPU, memory, and energy consumption of IoT platforms with and without IoTvar under synchronous interactions.

Table 4. Differences between using and not using IoTvar for the synchronous interaction pattern.

Platform	Metric	Number of IoTvar variables							
		25	50	75	100	125	150	175	200
FIWARE/Orion	CPU	12.38%	7.29%	5.07%	3.79%	3.68%	2.15%	2.94%	6.29%
	Memory	6.22%	3.97%	3.6%	4.85%	5.81%	3.57%	4.55%	3.23%
	Energy	1.79%	4.08%	5.53%	4.98%	4.92%	8.67%	4.13%	4.9%
OM2M	CPU	16.21%	13.92%	13.8%	11.59%	35.14%	34.02%	33.16%	26.86%
	Memory	4.01%	5.61%	5.62%	4.71%	1.41%	3.05%	6.09%	1.78%
	Energy	5.7%	4.91%	2.45%	4.59%	7.27%	5.23%	5.39%	4.39%

The results of the Shapiro-Wilk test pointed out that data were found not to follow a normal distribution (p-value $< \alpha = 0.05$), leading us to adopt the Mann-Whitney test in the subsequent analysis step [22]. The

Just Accepted

differences between using or not IoTvar were significant (p-value < $\alpha = 0.05$) for most cases regarding CPU, memory, and energy for FIWARE/Orion and OM2M.

Table 5 shows the evaluation of Vargha and Delaney's A for the magnitude of the overhead caused by IoTvar when using the synchronous interaction pattern with the FIWARE/Orion and OM2M platforms. The impact was medium for ten cases (41.67%) and large for nine cases (37.5%) when using FIWARE/Orion. The impact was also medium or large for most cases when using OM2M, specifically medium in three cases (12.5%) and large in 18 cases (75%).

Table 5. Magnitude of the overhead caused by IoTvar in the synchronous interaction pattern.

Platform	Metric	Number of IoTvar variables							
		25	50	75	100	125	150	175	200
FIWARE/Orion	CPU	M	L	S	S	M	S	S	M
	Memory	L	M	M	M	M	M	M	S
	Energy	M	L	L	L	L	L	L	L
OM2M	CPU	L	M	L	L	L	L	L	L
	Memory	M	L	L	M	N	S	L	N
	Energy	L	L	L	L	L	L	L	L

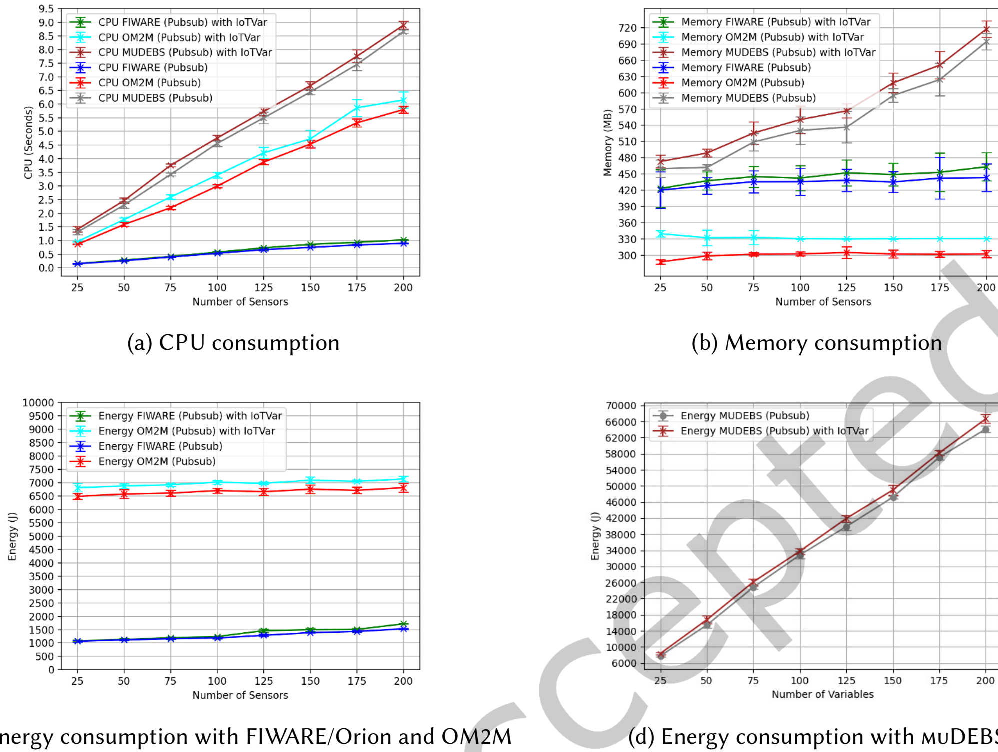
Levels of magnitude of the effect size: N = negligible, S = small, M = medium, L = large

5.3.2 Publish/subscribe interaction pattern. Figure 10 shows the CPU, memory, and energy performance of publish/subscribe notifications for all the three IoT platforms. It is possible to notice that that the CPU usage with IoTvar is higher than without using it. The overhead induced by IoTvar comes from the validations done inside the source code to ensure the correct update of variables, as well as handling multiple potential error cases. In addition, due to the number of proxy and error objects created when using IoTvar, memory consumption also increases. As a consequence, there is also an overhead in energy consumption for all the three IoT platforms. Table 6 shows the differences (in percentages) between using and not using IoTvar.

Table 6. Differences between using and not using IoTvar for the publish/subscribe interaction pattern.

Platform	Metric	Number of IoTvar variables							
		25	50	75	100	125	150	175	200
FIWARE/Orion	CPU	4.39%	8.19%	4.94%	7.23%	9.55%	13.99%	10.93%	13.45%
	Memory	0.68%	2.13%	2.1%	1.52%	3.12%	3.08%	2.43%	4.41%
	Energy	1.15%	1.36%	2.82%	4.5%	12.81%	7.65%	5.23%	11.51%
OM2M	CPU	10.96%	10.72%	16.4%	12.73%	8.37%	4.11%	9.87%	5.92%
	Memory	16.54%	10.56%	9.76%	8.82%	7.92%	8.86%	9.18%	8.91%
	Energy	4.89%	4.46%	4.64%	4.53%	4.61%	4.86%	4.89%	4.65%
muDEBS	CPU	7.42%	6.99%	9.28%	4.29%	4.16%	3.56%	3.95%	2.66%
	Memory	2.86%	5.61%	3.22%	3.7%	5.44%	3.85%	4.18%	3.31%
	Energy	7.46%	8.21%	5.18%	2.88%	4.93%	3.74%	1.87%	3.86%

Just Accepted



(a) CPU consumption

(b) Memory consumption

(c) Energy consumption with FIWARE/Orion and OM2M

(d) Energy consumption with MUDEBS

Fig. 10. CPU, memory, and energy consumption of IoT platforms with and without IoTvar under publish/subscribe interactions.

Just Accepted

The results of the Shapiro-Wilk test pointed out that data were found not to follow a normal distribution (p-value < $\alpha = 0.05$), leading us to adopt the Mann-Whitney test in the subsequent analysis step [22]. As for the synchronous interaction pattern, the differences between using or not IoTvar were significant (p-value < $\alpha = 0.05$) for most cases regarding CPU, memory, and energy for all the IoT platforms.

Table 5 shows the evaluation of Vargha and Delaney's A for the magnitude of the overhead caused by IoTvar when using the publish/subscribe interaction pattern with the IoT platforms. The impact was small for five cases (20.8%), medium for five cases (20.8%), and large for 12 cases (50%) when using FIWARE/Orion. The impact was large in almost all cases when using OM2M (23/24 ≈ 95.8%) and MUDEBS (21/24 = 87.5%).

5.4 Remarks on the results

The performed experiments and the analysis of their results showed how IoTvar behaves in some scenarios over different interaction patterns. The results presented in Section 5.3 point out that IoTvar brings a significant overhead when abstracting the IoT platform regarding CPU, memory, and energy consumption.

For applications that are required to use as low energy as possible, the slight increase in any resource consumption could be a problem. On the other hand, applications with the leeway to use more resources can benefit from using IoTvar to abstract interactions with IoT platforms and thus lowering development effort.

Table 7. Magnitude of the overhead caused by IoTvar in the publish/subscribe interaction pattern.

Platform	Metric	Number of IoTvar variables							
		25	50	75	100	125	150	175	200
FIWARE/Orion	CPU	M	L	L	M	L	L	L	L
	Memory	N	M	S	N	S	M	S	M
	Energy	S	S	L	L	L	L	L	L
OM2M	CPU	L	L	L	L	L	S	L	L
	Memory	L	L	L	L	L	L	L	L
	Energy	L	L	L	L	L	L	L	L
muDEBS	CPU	M	L	L	L	L	L	L	L
	Memory	L	L	M	M	L	L	L	L
	Energy	L	L	L	L	L	L	L	L

5.5 Threats to validity

Any empirical study has limitations that may constitute threats to the validity of observed results and outlined conclusions. This section reports the study's validity by describing the main identified threats and the adopted strategies to mitigate them.

External validity. External validity concerns the ability to generalize the obtained results to other contexts. The most significant threat to the external validity of this work are if the results would be similar for different IoT platforms. We presented here an evaluation of IoTvar with three platforms adopting synchronous and publish/subscribe interaction patterns, which are commonly observed in IoT. Nevertheless, we still need to carryout further experiments to assess the behavior of IoTvar considering different request payloads.

Internal validity. Internal validity concerns the conduction of the study, more specifically focusing on data collection procedures, which may affect the cause-effect relationships to reach the obtained conclusions. The main factors that could negatively affect the internal validity of this study are possible inaccuracies in data collection. The activity of the computational infrastructure used in the experiments can not be totally controlled, but we have striven to reduce discrepancies in data by shutting down or disabling operating system processes and device interactions that could interfere in the execution. As we mentioned in Section 5.1, considering the JVM warm-up and disregarding the first run minute were actions taken to ensure no interferences on the collected metrics.

Conclusion validity. Conclusion validity mainly concerns the analysis of obtained results, being related to the extent to which they can be regarded as correct and how rigorous the process to establish conclusions from those results was. To minimize possible threats to the conclusion validity of this study, the obtained results were analyzed by using robust statistic techniques. Each execution scenario was performed 30 times (see Section 5.2), thus ensuring an affordable statistical significance for the results when performing the statistical tests.

6 CONCLUSION

The heterogeneity in the overabundance of IoT platforms presents a significant challenge to finding, selecting, and using IoT resources, e.g., devices, sensors, services, and context data. Therefore, it is important to provide

Just Accepted

techniques that enable clients to discover, retrieve and use data produced by them. Due to the different types of data provided by IoT platforms and the various ways to interact with them, it is valuable to gather this data at a low development cost.

This paper has presented IoTvar, a middleware library that provides application developers with a way of interacting with an IoT platform using a few lines of code. For this purpose, IoTvar encompasses proxies representing IoT platform virtual entities. These proxies handle the complexity of interacting with the IoT platform in synchronous and publish/subscribe interaction patterns. Additionally, IoTvar offers a bypass for understanding the IoT platform-specific API and data model. We have described the integration of IoTvar with the FIWARE, OM2M and muDEBS IoT platforms and detailed its architecture and how it can be extended for other platforms. Extending IoTvar to support a new IoT platform relies on generic components and new ones for new APIs, unmarshalers, and, if necessary, interaction patterns and protocols.

We have also performed a quantitative evaluation of IoTvar with all the three currently supported platforms addressing the balance between the relative cost of IoTvar for both synchronous and publish/subscribe handling of information and the benefits for developers. The statistical analysis results revealed an impact in most cases regarding CPU, memory, and energy consumption. Despite being significant from the statistical point of view, such an impact could be acceptable depending on the application characteristics and objectives.

The current analysis of the IoTvar also led us to two open issues. The first one is a work in progress regarding the security and privacy of IoT data, which are important issues to be considered for the successful deployment of IoT applications. In the case of IoTvar, these features depend on what the supported IoT platforms provide through their APIs, which would then be implemented in the middleware and provided to the IoT application. Another open issue is on-demand resource utilization. IoT applications may be unpredictable concerning resource (e.g., memory and CPU) utilization. For example, the number of requests made by an IoT application to an IoT platform may increase suddenly, or sensor devices could get disconnected from the network. IoTvar currently provides an API that returns errors when searching for unavailable sensor data. Developers can add error handling to react to these errors and better control what the application does, thus controlling resource utilization. Nonetheless, developing IoT middleware design patterns for better control of resource usage is a challenging task that requires further studies.

In future work, we intend to integrate IoTvar with other platforms to bring more options for application developers. This also includes increasing the discoverability proposed by IoTvar, extending it to provide support for direct communication with IoT devices. We also plan to go further on the tests by having different types of payloads to ensure the behavior of IoTvar over different scenarios that require low or high amounts of data to be transferred from the IoT platform to the client. Furthermore, concerning security, we plan to integrate the security features from the IoT platforms currently supported in IoTvar and provide additional features to complement what other platforms offer. Finally, we will implement energy-efficient strategies at the middleware level, such as message grouping, QoS management (when supported by the IoT platform), and favoring the publish-subscribe pattern over the synchronous pattern when available [6]. This will also possibly improve the performance of IoTvar and further validate its use by IoT applications.

Just Accepted

REFERENCES

[1] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. 2014. OM2M: Extensible ETSI-compliant M2M Service Platform with Self-configuration Capability. *Procedia Computer Science* 32 (2014), 1079–1086. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).

[2] Parvaneh Asghari, Amir Rahmani, and Hamid Haj Seyyed Javadi. 2018. Internet of Things applications: A Systematic Review. *Computer Networks* 148 (12 2018), 241–261.

[3] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. 2008. Services Mashups: The New Generation of Web Applications. *Internet Computing, IEEE* 12 (10 2008), 13–15. https://doi.org/10.1109/MIC.2008.110

[4] Pedro Victor Borges, Chantal Taconet, Sophie Chabridon, Denis Conan, Thais Batista, Everton Cavalcante, and Cesar Batista. 2019. Mastering Interactions with Internet of Things Platforms through the IoTVar Middleware. *Proceedings of the 13th International Conference on Ubiquitous Computing and Ambient Intelligence(UCAmI)* 31, 1 (Nov 2019), 78.
[5] Raja Boujbel, Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Jean-Paul Arcangeli, and Claire Lecocq. 2014. MuScADeL: A Deployment DSL Based on a Multiscale Characterization Framework. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*. 708–715. https://doi.org/10.1109/COMPSACW.2014.120
[6] Rodrigo Canek, Pedro Borges, and Chantal Taconet. 2022. Analysis of the Impact of Interaction Patterns and IoT Protocols on Energy Consumption of IoT Consumer Applications. In *Distributed Applications and Interoperable Systems*, David Eyers and Spyros Voulgaris (Eds.). Springer International Publishing, Cham, 131–147.
[7] M. A. Chaqfeh and N. Mohamed. 2012. Challenges in middleware solutions for the internet of things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*. 21–26.
[8] Flávia C. Delicato, Paulo F. Pires, and Thais Batista. 2013. *The Programming and Execution Module (PEM)*. Springer London, London, 45–55. https://doi.org/10.1007/978-1-4471-5481-5_5
[9] N. Denis, P. Chaffardon, D. Conan, M. Laurent, S. Chabridon, and J. Leneutre. 2020. Privacy-preserving Content-based Publish/Subscribe with Encrypted Matching and Data Splitting. In *Proc. of the 17th International Joint Conference on e-Business and Telecommunications*. INSTICC, SciTePress, Paris, France, 405–414.
[10] Paul D. Ellis. 2010. *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results*. Cambridge University Press, United Kingdom. https://doi.org/10.1017/CBO9780511761676
[11] Michael P. Fay and Michael A. Proschan. 2010. Wilcoxon-Mann-Whitney or *t*-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys* 4 (2010), 1–39. https://doi.org/10.1214/09-SS051
[12] FIWARE. 2021. FIWARE-NGSI v2 Specification. http://telefonicaid.github.io/fiware-orion/api/v2/stable/
[13] FIWARE consortium. 2022. *FIWARE Platform*. https://www.fiware.org
[14] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. *SIGPLAN Not.* 42, 10 (oct 2007), 57–76. https://doi.org/10.1145/1297105.1297033
[15] Ross Harmes and Dustin Diaz. 2008. The Proxy Pattern. In *Pro JavaScript Design Patterns*. Apress, Berkeley, CA, 197–214.
[16] Melinda Hess and Jeffrey Kromrey. 2004. Robust Confidence Intervals for Effect Sizes: A Comparative Study of Cohen's d and Cliff's Delta Under Non-normality and Heterogeneous Variances. *Paper Presented at the Annual Meeting of the American Educational Research Association* (01 2004).
[17] Soobin Jeon and Inbum Jung. 2017. MinT: Middleware for Cooperative Interaction of Things. *Sensors* 17, 6 (2017). https://doi.org/10.3390/s17061452
[18] Tomasz Kalbarczyk and Christine Julien. 2018. Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild. In *Proceedings of the 19th International Middleware Conference* (Rennes, France) *(Middleware '18)*. Association for Computing Machinery, New York, NY, USA, 161–173. https://doi.org/10.1145/3274808.3274821
[19] Ken Kelley and Kristopher J. Preacher. 2012. On effect size. *Psychological Methods* 17, 2 (2012), 137–152. https://doi.org/10.1037/a0028086
[20] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. 2001. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*. Springer-Verlag, Berlin, Heidelberg, 327–353.
[21] Shancang Li, Li Da Xu, and Shanshan Zhao. 2015. The internet of things: a survey. *Information Systems Frontiers* 17, 2 (April 2015), 243–259.
[22] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50 – 60. https://doi.org/10.1214/aoms/1177730491
[23] Pierrick Marie, Léon Lim, Atif Manzoor, Sophie Chabridon, Denis Conan, and Thierry Desprats. 2014. QoC-Aware Context Data Distribution in the Internet of Things *(M4IOT '14)*. Association for Computing Machinery, New York, NY, USA, 13–18.
[24] Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. 2016. A gap analysis of Internet-of-Things platforms. *Computer Communications* 89-90 (Sept. 2016), 5–16.
[25] Bhumi Nakhuva and Tushar Champaneria. 2015. Study of various internet of things platforms. *International Journal of Computer Science & Engineering Survey* 6, 6 (2015), 61–74.
[26] NGSI 2021. FIWARE-NGSI v2 Specification. https://fiware.github.io/specifications/ngsiv2/stable/
[27] Anne H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. 2017. IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal* 4, 1 (2017), 1–20. https://doi.org/10.1109/JIOT.2016.2615180
[28] NODE-RED 2021. Node-Red. https://nodered.org/
[29] OASIS. 2015. MQTT Version 3.1.1 Plus Errata 01. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf. Accessed on 21-05-2021.
[30] OM2M API 2021. the oneM2M REST APIs. https://www.onem2m.org/getting-started/onem2m-overview/application-program-interfaces-api
[31] oneM2M Partners 2019. *oneM2M Services Platform*. oneM2M Partners. Release 3.
[32] Open mashup alliance [n.d.]. Enterprise Mashup Markup Language. http://www.openmashup.org/.

Just Accepted

[33] Pankesh Patel and Damien Cassou. 2015. Enabling high-level application development for the Internet of Things. *Journal of Systems and Software* 103 (2015), 62–84.

[34] Dave Raggett. 2015. The web of things: Challenges and opportunities. *Computer* 48, 5 (2015), 26–32.

[35] Partha Pratim Ray. 2016. A survey of IoT cloud platforms. *Future Computing and Informatics Journal* 1, 1 (2016), 35–46.

[36] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. 2016. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal* 3, 1 (2016), 70–95.

[37] April Reeve. 2013. Chapter 12 - Data Integration Patterns. In *Managing Data in Motion*, April Reeve (Ed.). Morgan Kaufmann, Boston, 79–85.

[38] Marc Shapiro. 1986. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *Int. Conf. on Distr. Comp. Sys. (ICDCS) (Int. Conf. on Distr. Comp. Sys. (ICDCS))*. IEEE, Cambridge, MA, USA, United States, 198–204. https://hal.inria.fr/inria-00444651

[39] Samuel Sanford Shapiro and Martin Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3-4 (1965), 591–611.

[40] K. J. Singh and D. S. Kapoor. 2017. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine* 6, 2 (2017), 57–68.

[41] Karthik Soundararajan and Robert Brennan. 2008. Design patterns for real-time distributed control system benchmarking. *Robotics and Computer-integrated Manufacturing - ROBOT COMPUT-INTEGR MANUF* 24 (10 2008), 606–615.

[42] Pierre Sutra, Etienne Rivière, Cristian Cotes, Marc Sánchez Artigas, Pedro Garcia Lopez, Emmanuel Bernard, William Burns, and Galder Zamarreño. 2017. CRESON: Callable and Replicated Shared Objects over NoSQL. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 115–128.

[43] Antero Taivalsaari and Tommi Mikkonen. 2017. A roadmap to the Programmable World: Software challenges in the IoT Era. *IEEE Software* 34, 1 (Jan.-Feb. 2017), 72–80. https://doi.org/10.1109/MS.2017.26

[44] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[45] WIRECLOUD 2021. Wirecloud. https://wirecloud.readthedocs.io/en/stable/

[46] Franco Zambonelli. 2017. Key abstractions for IoT-oriented Software Engineering. *IEEE Software* 34, 1 (Jan.-Feb. 2017), 38–45. https://doi.org/10.1109/MS.2017.3

Just Accepted