

VIoT – A Step Towards Easing the Interoperability of IoT-Based Applications

Dannylo J. B. Egídio(✉) and Gibeon S. de Aquino Jr.

Department of Informatics and Applied Mathematic,
Federal University of Rio Grande do Norte, Natal, Brazil
dannylojohnathan@gmail.com, gibeon@dimap.ufrn.br

Abstract. The growing improvement in embedded computing, sensing technologies, and connected devices has enabled the advancement of innovative paradigms, such as the Internet of Things (IoT). However, the considerable diversity of devices and, consequently, of protocols that have emerged have made this process difficult. Numerous challenges emerged, such as heterogeneity, interoperability, multiplicity of data representation and services, etc. These challenges have made the development of IoT applications somewhat complex and costly since the capabilities of these devices, protocols, and standards have become device-specific, forcing the developer to come up with complex integration strategies to address these specifications. In this sense, this work proposes a framework, called VIoT, that seeks to facilitate the process of developing IoT applications for heterogeneous environments through device virtualization. The strategy will attempt to abstract the heterogeneous aspects related to the physical devices, which will be represented by a virtualized model, encapsulating the operations common to its protocols and providing a common interface between the real device and the virtualized one, thus reducing, from the developer's point of view, the effort in implementing applications for these environments.

Keywords: Internet of Things · Framework · Virtualization · Easing · Application

1 Introduction

The recent years have been marked by a growing advance in embedded computing, sensing technologies, and smart devices. Some predictions state that about 29 billion devices will be connected by 2022 [7]. This strong emergence of intelligent and connected devices has led to the development of a series of services that are both useful and important to people. These services seek to base themselves on innovative computational paradigms such as the Internet of Things (IoT).

The Internet of Things is a concept based on smart objects, considered as "thing", that cooperate with each other through the internet to reach a common

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 482–496, 2019.
https://doi.org/10.1007/978-3-030-24308-1_39

goal [10]. This concept has provoked the interest of both the industry and the market in basing their new services and applications on the IoT principles [3]. This way, smart ecosystems have been suggested for many contexts, such as smart homes, smart cities, smart farms, smart factories, among others [14].

However, many challenges still need to be overcome to achieve the IoT concepts. Among these challenges, it is possible to highlight the heterogeneity and interoperability of devices and standards, which are considered by Tayur [13] as one of the most significant limitations of IoT. Also, issues related to the devices' resource restrictions [2], the variety of forms of data representation, and multiple communication protocols [1,9,11] are also considered critical challenges of the IoT paradigm. Finally, all of these challenges consequently end up making the development of IoT applications complex and costly.

Considering these challenges, standards have been suggested by many suppliers consortia to deal with many of the limitations mentioned. These standards seek to provide mechanisms for the registration, discovery, and communication for the devices involved, as well as strategies for security, identification, and management of resources. However, they only reach a very specific and restricted set of devices. Besides, these patterns are usually incompatible with each other, requiring developers to formulate generally complex integration strategies [13].

In this sense, this work's goal is the development of the VIoT framework. Furthermore, VIoT aims to facilitate the development of IoT-based applications by abstracting the heterogeneous capabilities of the communication protocols. This approach will seek to treat the application devices uniformly, providing resources for their virtual representation. This way, it will be possible to reduce the effort needed to implement these applications, since the devices involved will be interoperable, even if they are heterogeneous.

This paper is organized into seven sections. Section 2 will delimit, in a detailed way, the problem addressed by this work. Section 3 will describe the works related to this research. Section 4, in its turn, will provide details on the implementation of the proposed framework's architecture, its components and the details related to instantiation and use. Section 5 will describe a proof of concept using VIoT and the evaluation of the proposal considering how easy it was to use. The Sect. 6 deals with conclusions and future works and the Sect. 7 will describe the acknowledgements.

2 The Problem

To illustrate VIoT's utility, we will use a typical scenario of a room with temperature control. The room would consist of a temperature sensor that would measure the temperature and a thermostat that would regulate the environment's temperature. Both would have separate communication protocols: the sensor would use UPnP, while the thermostat would work with Bluetooth.

In addition to the communication protocols, the devices also have different capabilities, which makes it difficult for the application to manipulate them. To handle the UPnP sensor, it is necessary to discover it on the network, know

its services and actions and send messages to its status variables. For the Bluetooth, however, it will be necessary to consult the network, pair with the chosen device and, after obtaining the connection, transmit the data required, finally terminating the connection.

To perform such an operation, two logic modules will be needed to deal with each of the protocols and their specificities. In this case, each module will contain the code (which must be written by the developer) needed to meet the characteristics and architecture of each protocol. The developer will need to handle each protocol individually for the application to be effective.

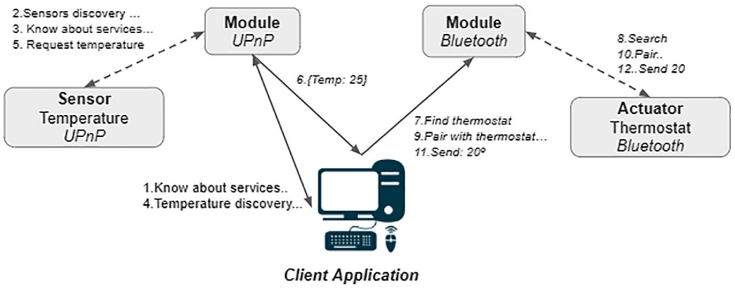


Fig. 1. Temperature control scenario with heterogeneous devices.

The Fig. 1 illustrates the scenario mentioned above: the application requests the available services from the UPnP module (1) which, in turn, sends its discovery commands to the UPnP stack (2) and receives information about the services available to inform the client application. Once it has the services in its hand, the application can discover the temperature (4) and receive, from the module, a JSON format containing its value (6).

After verifying the temperature's value, the application was able to identify that it is higher than desired, thus initiating its communication with the Bluetooth module (actuator protocol) to modify the temperature's state. For this, it is necessary to follow the steps required by the protocol: search for the thermostat (7) on the Bluetooth stack; when it is found on the network, pair with it (9) so that the protocol grants us a valid connection with the device for the communication; only then, send information about the appropriate temperature (11) which, in the example mentioned, is 20 °C.

The infrastructure considered in this scenario demonstrates the need to understand how each device operates and what are its service capabilities, how they are implemented and how they can be accessed and configured remotely. These questions have very different answers considering each device.

Our proposal, which aims to treat the heterogeneous devices in a unified and homogeneous way, will seek to promote the integration of these heterogeneous devices, thus reducing the effort needed to develop the application. The application's components will be abstracted as virtual devices in the application logic,

with properties and behaviors defined in a single virtualized model. Its capabilities will be abstracted from drivers that will be included in the VIoT. These drivers, when added, become proxies that know how to handle the individual capabilities of each protocol and consequently make VIoT capable of performing the operations needed on the devices.

3 Related Works

The proposal of [3] introduces a framework called DataTweet, which focuses on multiple domains and uses the Machine-to-Machine Measurement (M3) Framework to solve heterogeneity issues. The author's proposal is a two-part project division: the components of the application logic (AL) and the Common Service Entities (CSE). The first part abstracts all of the application's mechanisms in which the users are interested in, such as user interface, query mechanisms, and resources discovery, among others, while the second part abstracts the available services offered by the application in this module. The whole dynamics of registration, management, and interoperability is handled in the CSE, and the application-oriented functions belong to AL. These modules are intermediated by a common interface.

The proposal explained above is based on the work of [10] which, in turn, seeks to develop a conceptual framework based on the division of skills found in an IoT infrastructure. Their approach features descriptive languages called "Srijan Vocabulary" which will model existing devices and generate executable codes for the applications.

The work of [5] stands out for providing a framework that aims to treat the heterogeneity of protocols of an IoT application. The IoTDelegate abstracts the application in three layers: Plug-In, Collaboration, and Service Management. The first layer will remove any protocol-related dependency from the smart objects, while the others will create abstractions that allow the communication and collaboration of the constituent devices. The proposal of [6] reports a framework that is different from the ones previously mentioned, because it aims to simplify the development through a user-centered approach, reducing or eliminating the use of programming in levels closer to the hardware. The approach is based on models and uses Task Script for this purpose.

A proposal focused on abstraction that was considered in this work was the one made by [8], which proposes an API for accessing IoT-based services through REST and SOAP protocols. The approach seeks to provide means to facilitate the complexity inherent to the diversity of devices found in IoT applications. Basically, the proposal will present a framework that will provide uniform access to resources and operations through structured messages in SOAP and REST. The author used UPnP as the guiding standard for his experiment and evaluated aspects related to performance, discarding the use of SOAP due to results below expectations.

The work of [4] also raised the need to have a model that can uniformly represent the devices. The author proposed in his work a configuration platform

infrastructure (IoT-PIC) trying to consider the many layers found in an IoT infrastructure. Simplification occurs through a model-based approach for the development of new IoT-based applications and services. For this purpose, the author used XMPP in the entity communication process and Resource Application Interface (RAI) to handle the virtualization. However, the solution virtualizes according to the driver available in the RAI layer and returns a virtualized instance specific to each driver.

The approaches mentioned above relate to this work because they use ways of abstracting objects through model proposals. However, most of the proposals omit details of how the devices will relate to the models in the application. Although some evidence the need for network discovery mechanisms, such as [3], they do not report clearly when this operation occurs. Some proposals seek solutions through specialized virtualizations for specific protocols, such as [4], so that the specialized knowledge of each protocol is still necessary. Our proposal believes that globalized and unique virtualization would reduce the complexity inherent to the development of IoT applications, facilitating coding in the treatment of the several existing devices.

4 VIoT: Proposed Architecture

The proposed architecture is illustrated in Fig. 2 and considers common operations in IoT-based applications to enable a harmonic interaction between the VIoT and the physical devices present in the solution's infrastructure. The operations considered in the architecture were those related to registry, status update, and device discovery, which will be explained next.

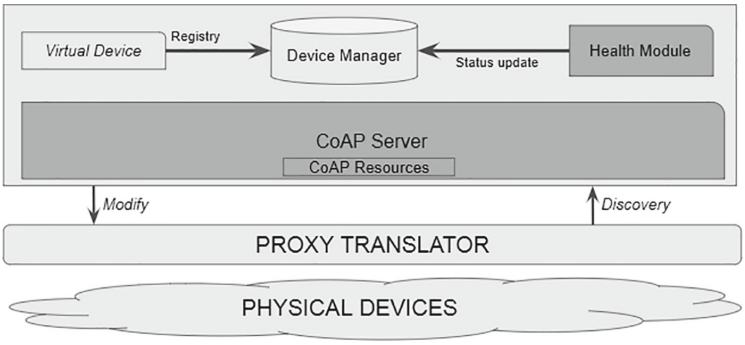


Fig. 2. Framework's architecture.

The registry operation is responsible for making the device known to VIoT by registering it in the Device Manager to considerer it, from the moment it is registered, a virtualized entity to be used in the application logic. The status update operation occurs periodically to prevent an inoperable device from being

available in VIoT. In the case of an inoperability happening, it will be removed from the Device Manager and disregarded by the device discovery operation.

The device discovery operation occurs in two different moments. The first moment is during the Proxy Translator discovery process, which will encapsulate each device discovered in a Virtual Device instance. The second moment occurs when the client application invokes it through the Device Manager's discovery methods. In this second moment, the devices returned in the discovery process will already be virtualized.

As for the architecture layers, as shown in Fig. 2, they are arranged as follows: the Physical Devices layer concentrates all the real devices found in the solution's infrastructure and their processing and communication capabilities. The Proxy Translator layer provides the communication protocol drivers inherent to the devices existing in the solution's infrastructure; each driver implements a common interface that will represent it as a valid Proxy for the framework. This layer is responsible for making the devices communicable and recognizable in VIoT.

Finally, the highest layer of the architecture concentrates VIoT's main idea. It is the layer responsible for the abstraction of the physical devices and the communication processes that form it. This layer is divided into four components: Virtual Devices, CoAP Server, Device Manager, and Health Module, which will be described in more details below.

4.1 Proxy Translator

The Proxy Translator's workflow is basically geared to the mechanisms that are particular to each proxy added, so that functions that are typical of the discovery of devices in the network and of sending and querying data for each device are necessarily implemented and available. This way, the process of discovering the devices of this layer implies in each proxy performing its own discovery processes and therefore encapsulating the devices found in Virtual Devices, thus registering in their own instances the proxy instance responsible for the operation, its resources and the actions it can perform.

Proxies are addable in this layer of the architecture. In this case, it is essential to consider that some communication protocols do not have high-level features such as automatic device discovery included in their structure. Therefore, for the proxy to be in accordance with VIoT's architecture, the developer will need to implement these additional features based on what the protocol can provide.

4.2 Virtual Device

A Virtual Device is a virtualized representation of the physical device present in the infrastructure of the IoT solution. It encapsulates the capabilities of the physical device so that they can be delivered to the client application through standardized methods. Virtualizing a device ensures that it will be handled uniformly during the process of developing the application's logic, since all of the

devices, regardless of their individual characteristics, will be represented through virtualization.

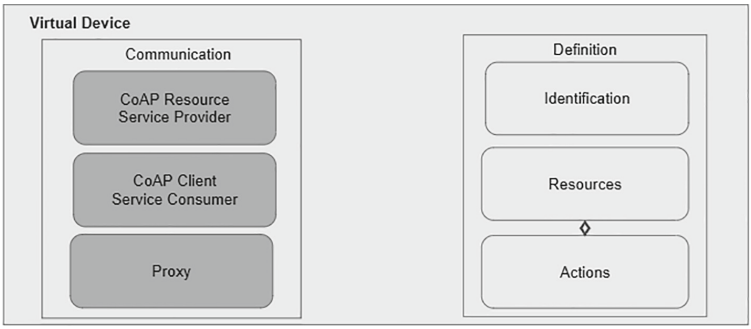


Fig. 3. Virtual Device's structure.

The internal structure of a Virtual Device is illustrated in Fig. 3. It is composed of two groups of components: the first group, called the communication group, represents the components that are essential for the communication between the Virtual Device and its real Device. The second group, called the definition group, represents the attributes that identify the Virtual Device in the VIoT. The definition group consists of the following components:

- **Identification:** stores, in the Virtual Device, the information that identifies it exclusively in the VIoT. The information that can be considered part of the identification of a device includes a unique ID, the type of device, the device's nominal description, its supplier, its version, etc.;
- **Resources:** represent a nominal description of a group of tasks that can be performed by a device, such as control the temperature, control the energy, etc. A Resource can be made up of one or more actions.
- **Actions:** represent a nominal description of each task that can be performed by the device, such as turning on or off the lights, checking the temperature status, etc. Each action belongs to a Resource.

The communication group, in its turn, consists of the following components:

- **CoAP Resource:** it is a representation of an Action through a URL in the standards required by the CoAP Protocol. For example: if we have an action named Check Temperature Status, we will have a CoAP Resource named coap://0.0.0.0/01/CheckTemperatureStatus, where 01 is the discovered device's identifier.
- **CoAP Client:** a CoAP Client represents an instance that can communicate with the valid CoAP Resources in the CoAP Server. This way, when the device wants to request an action, it will use the CoAP Client, informing it of the desired CoAP Resource.

– ***Proxy:*** represents an instance of the Proxy that discovered the physical device in the discovery process performed by the Proxy Translator.

The Coap Server managed by the VIoT will be responsible for mediating the communication between the proxies and their virtualized devices. This way, each call to its interaction methods will initiate the endpoints responsible for each Resource in the CoAP Server. An endpoint is a CoAP Resource represented through its URL. The interaction methods, in their turn, represent procedures capable of performing the communication between the virtual device and the physical device.

4.3 CoAP Server

The CoAP Server is an internal structure that will manage all of the resources and actions of the discovered devices, which have been converted into CoAP Resources. These resources have their own URL defined at the moment of virtualization and ensure that all communication with the device's resources happens through the CoAP Protocol, thus avoiding unexpected situations when dealing with devices and restricted networks.

The features of the CoAP Server, which represent the features and actions of the IoT solution's Devices, can also be accessed through the endpoints provided at the time of the VIoT's execution. These endpoints are the URLs mentioned above and can either return information from the devices or modify their state variables.

The use of the CoAP Protocol is justified by the need to deal with networks and devices with computational restrictions, as well as to enable other client applications to use the services of the IoT solution. These other applications will not necessarily need the VIoT if they know the endpoints it will make available.

4.4 Device Manager

The Device Manager is a component responsible for registering and storing all of the devices discovered and virtualized by the Proxy Translator layer. It is the access point of the client application to all of the devices that are virtualized and has dynamic discovery mechanisms that will enable the use of more comprehensive search semantics, such as search by the device's name or part of the name, by its identifier, by its type, etc. Any device that is not registered in the Manager is disregarded by the VIoT.

4.5 Health Module

The Health Module is a module that will periodically check the "health" status of the application's devices. It is essential because the physical devices involved in the IoT solution's infrastructure need to be always available and in full operation so that the application's logic will not fail due to the lack of availability of a device. This way, the module is responsible for monitoring all of the devices that

are registered, thus canceling the registration of those that, by chance, do not present viable conditions or fail to respond to a request made by the Health Module.

The monitoring starts from the first discovery of Devices performed by the Proxy Translator. This monitoring occurs through the status update operation, which will periodically request the discovery of the devices in the network, waiting to obtain a response from them. In cases of repeated denials, its registration is canceled in the Application Manager, and it consequently becomes unavailable in the discovery operations. For the proposed architecture, we defined three consecutive requests as the maximum tolerance. If the device exceeds this limit, it is considered unavailable by the Health Module. By default, the status update operation is repeated every 30 s after the first verification.

4.6 Device Communication

The communication between the devices is represented in Fig. 4 and illustrates how the VIoT will handle Virtual Devices in the process of executing the client application. The communication will only use methods of the Virtual Devices that were specifically defined for the interaction with the Physical Devices, which are *getDataEvent(action)* and *sendEvent(action, params)*.

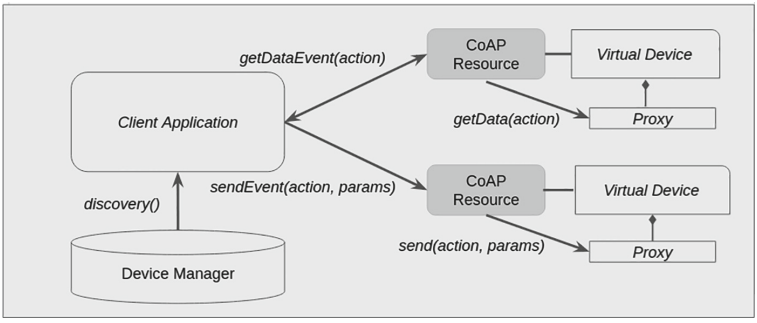


Fig. 4. Process of communication between virtual and physical devices.

The *getDataEvent(action)* method is responsible for requesting information from a particular action of the device that invoked it. The action represents what the device can perform. The *sendEvent(action, params)* method, in its turn, sends data to the device through well-defined parameters (*params*). The actions defined in this case modify the state of the devices that invoked them. In other words, invoking this method will not return any value, but will modify the state variables of the device that invoked it.

The Virtual Devices involved in the communication process need to be discovered by the Client Application through the *Device Manager*. If any requested device is not returned by the discovery method, it means that it is not operable.

In other words, it is not available for use by the Application or has not been identified by the *Proxy Translator* during the execution of that layer's discovery methods.

For the Virtual Devices to communicate with their physical devices, CoAP Resources are used as intermediaries. This way, when the application invokes any of the Devices' method of interaction (*getDataEvent* or *sendEvent*), the method is directed to the CoAP Resource that represents the desired action which, in turn, directs the method call to its Proxy, which also has a similar method of interaction (*getData* or *send*) for the action to be performed on the physical Device.

4.7 Framework Instanciation

The VIoT will be made available through a Java (.jar) compiled file that must be added to the IoT application's classpath. The whole instantiation process will occur through the Core class, which is responsible for methods that solve the dependencies necessary for VIoT's operation. The dependencies mentioned above are basically limited to searching for the existing proxies in the Proxy Translator layer and checking if there are any proxies defined by the client application. The base method for solving these dependencies is called *start()*.

A file called *config.properties* must be created in the root directory of the client application, and the VIoT will process it to define its execution guidelines. These guidelines are defined from two properties: *application.proxy.package* and *ignore.defaul*. The first property is responsible for indicating the package that will contain the additional proxies that were defined outside the VIoT. The second property can only assume two values: yes or no. This property will inform the framework if it should ignore its default proxies or should consider them during the client application's execution process. This routine has been defined for occasions when the device that will run the client application shows incompatibility with the proxies already natively defined by the VIoT.

```
1 Core.start();
2 VirtuaDevice sensor = DeviceManager.discovery("Sensor")
3 .get(0);
```

Fig. 5. Code responsible for the instantiation and discovery of devices in the VIoT.

After the proxies' dependency resolution process, which is triggered by the invocation of the *start()* method (Line 1), the devices can be searched using the DeviceManager and its discovery methods. Figure 5 shows the code snippet required for this process. Also, the example shown in the Figure demonstrates how to search for a device using its name.

It is possible to interact with virtualized devices through their interaction methods. Figure 6 demonstrates how to retrieve data captured by the virtualized

```
String rawData = sensor.getDataEvent(ResourceManager
.SensorService
.GET_DATA_SENSOR);
```

Fig. 6. Code required to obtain data from a sensor.

sensor. The *ResourceManager* is the class that maps all of the resources and actions defined by the proxies in the Proxy Translator's discovery process. In the case that was used as an example, the *getDataEvent* method shown in the Figure requests from the sensor the SensorService's *GET_DATA_SENSOR* action and stores its data in the *rawData* variable. The data is returned in the JSON format.

```
5  newValues.put(ResourceManager.ActuatorServices
6  .NEW_VALUE, "OFF");
7
8  actuator.sendEvent(ResourceManager
9  .ActuatorServices
10 .POWER_ACTION, newValues);
```

Fig. 7. Code required to send data to the actuator.

The use of the *sendEvent* method (see Fig. 7) is similar to the one shown in Fig. 6. However, among the mandatory parameters of this method, there is a mapping of the values that will represent the device's state variables and its new values. It is important to consider that these variables are also mapped by the *ResourceManager* in the Proxy Translator's discovery process. All of the code illustrated represents the basics of a client application that uses the VIoT as a facilitator resource for the devices' interoperability.

5 Evaluation

To evaluate the VIoT, a proof of concept was built based on an environment with temperature control. The scenario was chosen due to the simplicity of its structure; however, the main focus is the diversity of devices with heterogeneous capabilities. For this, a client application was implemented using the VIoT for the manipulation of the devices present in the IoT infrastructure. The client application can read the data captured by the temperature sensors and send a status change command to the thermostat, when necessary.

The implementation of the client application took place in an Android 4.1 device, the Samsung Galaxy Note, with Quad Core 1.9 GHz and 3 GB of RAM. A similar application with the same objective was developed in the same environment, without the aid of the VIoT. This was done to assist in the evaluation

of aspects related to the facility of the developers in dealing with the interoperability of heterogeneous devices.

It is important to consider that the projects had similar coding structures (same number of classes) and used a simulated sensor and a thermostat each (which we will call Implementation 1). The number of devices involved represents a minimally heterogeneous environment since each device has a different communication protocol. The metrics used for the analysis were Lines of Code (LOC) and Cyclomatic Complexity (CC). The first one aims at measuring an application by the number of lines of code implemented, while the second metric focuses on verifying the project's level of complexity through the generation of a flow graph [12].

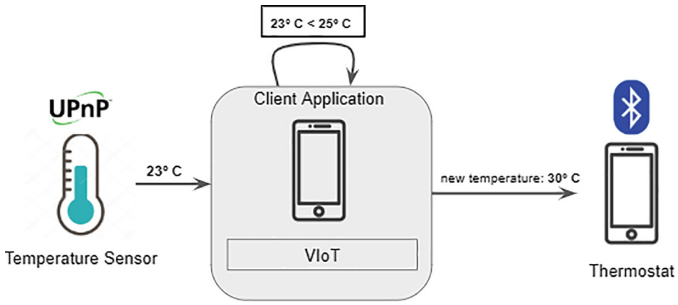


Fig. 8. Proof of concept of the proposed framework (VIoT).

The application developed used two distinct protocols, the Bluetooth and the UPnP. To make the development of the application viable, the Android platform was used, since it has a rich API focused on the manipulation of Bluetooth adapters. This way, an Android application was created to simulate a thermostat. The Bluetooth API was used for the definition of the application's routines, allowing the temperature to be set either from the application itself or through a remote command sent by a Bluetooth communication channel. The temperature sensors involved used the UPnP protocol and were programmatically simulated using Cling. The fact that the devices are simulated does not threaten the validity of the experiment since the protocols used in the simulations are identical to those used in real devices.

Thus, as shown in Fig. 8, the application recovered the virtualized sensors and thermostat and started its logic, verifying if the temperature captured by the sensors was equal to or greater than 25 °C. In cases of temperatures over 25 °C, the application would terminate its execution. In different cases, in which the temperatures were lower than 25 °C, the application would invoke the VirtualDevice's *sendEvent()* method, which represents the thermostat, sending parameters capable of modifying its state to 30 °C.

The lines of code from Implementation 1 were extracted and compared. The application that did not use the VIoT totaled 285 lines of code, while the one

that used it revealed a total of 118 lines of code, thus making it possible to attest an approximate reduction of 58,60% of lines of code in the application that used the VIoT.

Additionally, the analysis of the cyclomatic complexity also resulted in meaningful data. The application that used the VIoT had an average cyclomatic complexity of 12; for the application that did not use the VIoT, this value increased to 36. Thus, in terms of complexity and risk of development, the application that used the VIoT also demonstrated substantial evidence of a reduction in this aspect.

As another objective of the evaluation, it was also verified if the insertion of devices involved in the applications would impact the metrics that were found for the VIoT. For this, another pair of applications was developed, similar to the previous one, but containing ten simulated sensors and two thermostats each. This new pair of applications was called Implementation 2. In Implementation 2, 306 lines were counted in the application that did not use the VIoT, while 132 lines were found in the application that used it. Thus, it was possible to observe a total reduction of 56.90% in the number of lines, which represents a little more than 1% of difference compared to Implementation 1. The data referring to this metric was summarized in Table 1.

For the cyclomatic complexity, the numbers were more expressive: the general mean for the application that used the VIoT was 13, while the average for the application that did not use it was 43. In this case, the difference in LOC was considered very low and inexpressive, while the cyclomatic complexity revealed that there is a considerable increase in the risk for the application that does not use the VIoT when there are many devices involved. The other metrics considered and collected from implementations which also demonstrate advantages in using VIoT can be read in Table 2.

Table 1. LOCs of the proof of concept.

Projects	LOCs (with VIoT)	LOCs (without VIoT)
Implementation 1	118	285
Implementation 2	132	306

Table 2. Metrics collected from implementations.

	Implementation 1		Implementation 2	
Metrics	With VIoT	Without VIoT	With VIoT	Without VIoT
Number of methods	5	7	5	7
Lines of Code by Methods	11.41	15.86	7.34	10.52
Number of attributes	10	20	10	20
Lines of Code by Classes	59.1	142.5	66.0	153.5

This way, the numbers found showed that the use of the VIoT in the proposed proof of concept presents relevant indications for the reduction of coding effort and cyclomatic complexity in the process developing environments with heterogeneous devices.

6 Conclusions

IoT applications are permeated with critical challenges that need to be overcome. These challenges range from the variety of capabilities of communication devices and protocols to their computational and storage constraints. Consequently, these challenges make the development complex and costly.

In this context, our work presented the VIoT, a framework that can facilitate the development of IoT-based applications through a device virtualization approach. The approach aims to smooth out the heterogeneous aspects from the developer's point of view by reducing the coding effort needed to make the devices interoperable. It was possible to define and present an architecture that favors this approach, showing the components that form it and the means of instantiating it and extending it into IoT application projects.

Through a proof of concept that used a temperature control environment composed of devices with heterogeneous capacities, it was possible to test the proposed VIoT to obtain information necessary to validate it in aspects related to the reduction of coding, specifically focused on facilitating the development of IoT applications. This way, metrics (LOC and CC) were used to verify these aspects, thus obtaining, through this analysis, relevant results that attested a considerable decrease in the implementation effort.

As proposed future works we intend to create a Description Service Language (DSL) for the communication of IoT devices. The language will use the VIoT to accomplish its objectives. It will provide a simple syntax suitable for the definition of the devices' communication mechanisms, as well as for the execution of routines involving these devices and for the development of decision mechanisms, among other resources that are necessary for an IoT context. All of these resources must obey the architectural model defined in this work. We also believe that the evolution of the Health Module found in the VIoT's architecture would be essential for implementing more advanced fault tolerance strategies, guaranteeing greater reliability of the VIoT-based applications.

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

References

1. Ashraf, Q.M., Habaebi, M.H., Islam, M.R., Khan, S.: Device discovery and configuration scheme for Internet of Things. In: 2016 International Conference on Intelligent Systems Engineering (ICISE), pp. 38–43. IEEE (2016)
2. Carranza-García, F., Rodríguez-Domínguez, C., Garrido, J.L., Guerrero-Contreras, G.: BaaS-4US: a framework to develop standard backends as a service for ubiquitous applications. In: International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), pp. 23–30. IEEE (2016)
3. Datta, S.K., Bonnet, C.: Easing IoT application development through datatweet framework. In: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), pp. 430–435. IEEE (2016)
4. Ferrera, E., et al.: XMPP-based infrastructure for IoT network management and rapid services and applications development. Ann. Telecommun. **72**(7–8), 443–457 (2017)
5. Kum, S.W., Kang, M., Park, J.I.: Iot delegate: Smart home framework for heterogeneous IoT service collaboration. KSII Trans. Internet Inf. Syst. **10**(8) (2016)
6. Manione, R.: User centered integration of Internet of Things devices. In: Smart Sensors, Actuators, and MEMS VIII, vol. 10246, p. 102461K. International Society for Optics and Photonics (2017)
7. Mehta, A., Baddour, R., Svensson, F., Gustafsson, H., Elmroth, E.: Calvin constrained a framework for IoT applications in heterogeneous environments. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1063–1073. IEEE (2017)
8. de Melo Silva, C.C., Ferreira, H.G.C., de Sousa Júnior, R.T., Buiati, F., Villalba, L.J.G.: Design and evaluation of a services interface for the Internet of Things. Wirel. Pers. Commun. **91**(4), 1711–1748 (2016)
9. Pal, A., Mukherjee, A., Balamuralidhar, P.: Model-driven development for Internet of Things: towards easing the concerns of application developers. In: Giaffreda, R., et al. (eds.) IoT360 2014. LNICST, vol. 150, pp. 339–346. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19656-5_46
10. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. J. Syst. Softw. **103**, 62–84 (2015)
11. Sasirekha, S., Swamynathan, S.: Collaboration of IoT devices using semantically enabled resource oriented middleware. In: Proceedings of the Third International Symposium on Computer Vision and the Internet, pp. 98–105. ACM (2016)
12. Sharma, M., Singh, G., Arora, A., Kaur, P.: A comparative study of static object oriented metrics. Int. J. Advancements Technol. **3**(1), 25–34 (2012)
13. Tayur, V.M., Suchithra, R.: Review of interoperability approaches in application layer of Internet of Things. In: 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), pp. 322–326. IEEE (2017)
14. Yen, I.L., Zhang, S., Bastani, F., Zhang, Y.: A framework for IoT-based monitoring and diagnosis of manufacturing systems. In: 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 1–8. IEEE (2017)