IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 22, 2021, accepted April 4, 2021, date of publication April 14, 2021, date of current version April 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3073240

Enabling Internet of Media Things With Edge-Based Virtual Multimedia Sensors

ANSELMO LUIZ ÉDEN BATTISTI, DÉBORA CHRISTINA MUCHALUAT-SAADE, AND FLÁVIA C. DELICATO

MídiaCom Laboratory, Institute of Computing, Universidade Federal Fluminense (UFF), Niterói 24210-240, Brazil

Corresponding author: Anselmo Luiz Éden Battisti (anselmo@midiacom.uff.br)

This work was supported in part by the São Paulo Research Foundation (FAPESP) under Grant 2015/24144-7, in part by the Coordination for the Improvement of Higher Education Personnel (CAPES), in part by the CAPES-Print, in part by the National Council for Scientific and Technological Development (CNPq), and in part by the Rio de Janeiro State Research Foundation (FAPERJ).

ABSTRACT Multimedia sensors have recently become a significant data source in the Internet of Things (IoT), giving rise to the Internet of Media Things (IoMT). Since multimedia applications are usually latency-sensitive, data processing in the cloud is not always suitable. A strategy to minimize delay is processing multimedia streams closer to data sources, exploiting resources at the network edge. Moreover, virtualization is widely used to reduce complexity arising from heterogeneity in IoT environments. In this paper, we propose an edge-based architecture and platform to manage virtual multimedia sensors (VMS), enabling IoMT applications to be easily deployed. Our proposal encompasses V-PRISM, a software architecture tailored for IoMT, and ALFA, a distributed implementation of the architecture. V-PRISM components were designed to be deployed and executed in multiple edge nodes. VMSs are in charge of processing multimedia streams and provide an abstraction layer between IoMT applications and physical devices that produce those streams. This paper describes the proposal and the results of experiments showing that the proposed approach can successfully perform multimedia stream processing for applications that requires low-latency.

INDEX TERMS Edge computing, Internet of Media Things, Internet of Things, virtual multimedia sensors.

I. INTRODUCTION

With the widespread of the Internet of Things (IoT) and its integration with cloud computing, a new paradigm named Cloud of Things (CoT), or Cloud-assisted IoT, has recently emerged [1] exploiting the synergy between IoT and the cloud. In this scenario, places, people, and objects are continuous data generation sources that are consumed by applications that receive processed data streams through the cloud. The cloud provides a vast amount of storage and processing capabilities for the data generated by IoT devices while abstracting their heterogeneity. By offering sensing and actuation as a service, cloud providers broaden their portfolio of services for users and applications.

One of the enabling technologies of the CoT paradigm is virtualization. It refers to the process of building a logical abstraction of hardware and/or software features. Virtualization is at the core of cloud computing. It allows hiding from clients the variety of types of infrastructures, platforms and data available at the back-end, promoting the decoupling between entities that produce and consume resources and facilitating application delivery. It has also been used in other contexts, such as communication networks [2], and more recently in sensors and sensor networks [3]. In wireless sensor network systems, virtualization allows the creation of virtual sensor nodes, which abstract physical sensor nodes and are responsible for providing services to applications and end-users. The mapping of physical sensors to their virtual counterparts is done through a virtualization model. Several virtualization models are described in the literature specifically designed for wireless sensors and sensor networks [4]. Such models are tailored for sensors with reduced processing, memory and battery capacities, equipped with wireless communication interfaces. Wireless sensors are examples of devices that make up an IoT system.

Among the various types of sensors that compose the IoT infrastructure, and therefore, the CoT, multimedia devices have increasingly stood out. In a report produced by Cisco [5], they estimate that by 2022 about 80% of the Internet bandwidth will consist of multimedia streams. The relevance of this type of device has given rise to the concept of Internet of Media Things (IoMT)[1] [6], or even Multimedia Internet

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

[1] IoMT is also called Internet of Multimedia Things

of Things (M-IoT) [7]. In this work, we will adopt the term IoMT also adopted in the ISO/IEC 23093-1:2020 standard.[2]

In the IoMT, sensors are cameras and microphones with limited processing and storage capacity, which connect to heterogeneous devices and diverse applications. Traditional sensors, such as temperature, pressure, and light detection, typically generate discrete data. On the other hand, multimedia sensors produce continuous, massive data streams with nontrivial structure and temporal significance. Such features make the processing of multimedia streams more complex than traditional sensor data. Moreover, there is high heterogeneity in the communication protocols and data formats utilized by multimedia devices, adding an extra level of complexity in the acquisition, processing, and data consumption.

The particular features of multimedia streams in the context of IoMT make the use of existing sensor virtualization models unsuitable. Therefore, it is necessary to design virtualization models tailored for multimedia sensors. Considering the high heterogeneity of multimedia devices and the specific requirements of multimedia stream processing, the design of models and mechanisms to abstract multimedia sensors will enable the development of innovative services in several relevant areas such as health, security [8], education and entertainment.

In a typical CoT system, sensor-generated multimedia streams are virtualized in the cloud and delivered on-demand to applications. Such an approach implies that the massive amount of generated data needs to be transferred from the devices to the cloud, thus demanding a considerable amount of bandwidth. Moreover, cloud computing incurs high latency for data exchange between cloud servers and devices. Therefore, the cloud-based IoT model cannot meet the strict temporal requirements of multimedia applications. A promising strategy that has recently gained momentum to decrease the latency for applications is to migrate (part of) processing from the cloud to the network edge, placing it closer to the sensing devices (data sources). Such strategy is exploited by recent paradigms of edge [9] and fog computing [10].

In this work, we use the terms edge and fog interchangeably. We adopt the definition of Edge Computing (EC) provided by [11], which is a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum. The use of EC brings benefits as decreasing delay and bandwidth consumption at the network core, better use of available resources, and increasing data security and privacy. As multimedia applications benefit from running in low-latency environments, designing an architecture for the virtualization of multimedia sensors at the network edge is a promising approach.

This work proposes a distributed environment for the virtualization of multimedia sensors in edge computing environments. The proposed approach encompasses a software architecture, named V-PRISM, and its concrete distributed implementation, called ALFA. V-PRISM components are responsible for processing multimedia streams produced by physical multimedia devices. Stream processing is performed by entities named Virtual Multimedia Sensors (VMS). The architecture considers IoMT systems following a three-tier approach encompassing the cloud, the edge, and the things tiers. All components of the architecture are deployed in the edge tier. Besides presenting the logic components of V-PRISM and their operation, this paper also describes its concrete implementation in a platform named ALFA. We developed and validated V-PRISM logical components and adopted several technologies for ALFA deployment. Furthermore, we developed a set of virtual multimedia sensors and virtual devices to evaluate our approach in real scenarios.

This work is an extension of our initial proposal [12]. In this new version, V-PRISM has evolved to a fully distributed edge architecture with several new components like VMS allocation, Environment Monitoring and Stream Sharer. We also conduct new experiments showing that the proposed approach can perform multimedia stream processing for applications that require low-latency. Besides, to the best of our knowledge, V-PRISM is the only sensor virtualization architecture coined specifically to support multimedia streams.

The remainder of this paper is organized as follows. Section II provides a brief background on sensor virtualization architectures and compares our proposal to related work. We present our proposed distributed V-PRISM architecture in Section III. In Section IV, we discuss the ALFA platform. In Section V, we discuss virtual devices and virtual multimedia sensors available in our implementation. In Section VI, we present an evaluation of our work considering QoS aspects of video processing in VMS deployed in the edge and cloud nodes. Section VII brings main conclusions and future work.

II. RELATED WORK

A. INTERNET OF MEDIA THINGS (IoMT)

Cameras and microphones have become a significant source of data in the IoT. The use of multimedia sensors in IoT environments has given rise to a new subset of the IoT called Internet of Media Things (IoMT). Beyond the traditional IoT challenges, IoMT has specific issues that must be addressed to enable its adoption on a large scale. Table 1 lists some fundamental differences between IoT and IoMT [13].

TABLE 1. Comparison between IoT and IoMT [13].

IoT Scenario	IoMT Scenario
Linear Data	Bulky Data
Low Processing	High Processing
Low Storage	Massive Storage
Low Bandwidth	High Bandwidth
Delay Tolerant	Delay Sensitive
Low Power Consumption	High Power Consumption
Simple Data Encoding	Complex Media Encoding

In the IoMT scenario, data collected, processed, transported, and consumed are multimedia streams. Traditional

[2]https://www.iso.org/news/ref2449.html

sensors, as temperature and humidity sensors, produce discrete data organized in simple data structures. In contrast, multimedia sensors produce continuous data with a complex data structure. Moreover, the bulky nature of multimedia streams, in contrast to the IoT limited network bandwidth, increases the challenges to satisfy application QoS (Quality of Service) requirements [7]. Therefore, new techniques, standards, and frameworks must be developed to deal with these new challenges.

The advance of IoMT brings new light to a dynamic scenario where multimedia sensors are part of our daily life and whose multimedia data can be used unpredictably by IoT software developers. The deployment of multimedia sensors is not an easy task. This difficulty has risen a new type of service called multimedia sensing as a service (MSaaS) [14]. In this scenario, infrastructure providers offer MSaaS to IoMT application, enabling a fast growth of this type of applications. Because of that, the proposal of architectures that virtualize multimedia devices is a promising trend.

The significant heterogeneity of devices and multimedia vendors brings an extra effort to develop, deploy, and manage IoMT applications. On the other hand, the adoption of an underlying architecture or framework may reduce that effort significantly. Nowadays, IoT architectures are mainly focused on discrete data sharing and processing.

B. SENSOR VIRTUALIZATION ARCHITECTURES

To select related work about Sensor Virtualization Architectures (SVA), we adopted the software architecture definition provided by [15], where an architecture consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system into discrete, non-overlapping parts or components. The coordination strategy enables defining interfaces between those parts. Extending the software architecture definition provided by [15], we define an SVA as a set of software components and patterns that enable the development and management of virtual sensors (VS), that are the source of data to applications. In this section, we discuss some previously proposed VSAs to manage virtual sensor environments. We did not restrict our search for architectures that manage multimedia virtual sensor because they are in a limited number.

There are some approaches to deal with VS. In [16], the authors propose a scalable virtual sensor framework that supports building a logical dataflow (LDF) by visualizing physical sensors or custom virtual sensors. A web-based virtual sensor editor was implemented using the framework to simplify the creation and configuration of the LDF. Each virtual sensor is composed of a set of linked mathematical functions provided by the MathJS library.[3] Each VS is a process running inside a cluster server. There is only one cluster, and it is elastic to adapt to the unexpected change in terms of the number of concurrent users. This adaptation is done by adding or removing a node to/from the cluster. The authors do not discuss the strategy adopted for VS allocation. Besides that, as the VS is composed only of purely predefined mathematical functions, the capability of providing new VS types is limited to scalar data.

In [17], the authors propose FITOR, an orchestration system for IoT applications in the fog environment. This approach provides an optimized fog service provisioning strategy for minimizing the provisioning cost of IoT applications and meeting their computational and latency requirements. The architectural model assumes that the edge nodes must be a Docker host machine, and each running container must have a set of services to collect data from the container. The strategy adopted to collect statistical data was not optimized, increasing the resources used in an already limited resource environment. In addition, the approach was not specifically tailored to deal with multimedia edge node environments. Besides that, they did not specify which resource allocation method was used.

A new sensor cloud architecture for IoT based on fog computing was proposed in [18]. The architecture preprocesses raw sensor data on fog nodes and provides temporary storage of the results. It controls and manages diverse types of sensors in IoT devices through the virtualization of physical sensors providing dynamic, on-demand, elastic, and standardized Sensing-as-a-Service to the users. However, the proposed architecture does not allow the inclusion of multimedia sensors or VS remote management.

Two similar approaches were proposed in [19] and [20]. Both studies propose a web-based authoring tool for creating virtual sensors. In these tools, a VS is created by aggregating the data provided by physical sensors, and, the data-flow is chained to generate VS for complex event detection. VS types are defined using JSON and Javascript. When the execution of a VS starts, the VS type definitions (JSON and Javascript files) are loaded from a database. After that, the loaded script is evaluated by a Javascript Engine. The Javascript Engine runs in the server where the architecture is executed. It implies that there is only one node for running VSs. Besides that, the architecture defines that the devices must generate scalar data.

Another relevant work was discussed in [21]. The authors proposed a framework to virtualize multiple types of sensors. In addition, they developed a communication protocol called OMCP (OSIRIS Module Communication Protocol), which allows external applications to manage virtual sensors. It is important to highlight that, as far as we are concerned, that study was the only framework that formally discussed and implemented such a type of interaction. Unfortunately, the authors did not discuss aspects of virtual sensor deployment, an essential issue for sensor virtualization.

Besides academic work, many industry initiatives use the virtual sensor approach to deal with IoT devices. In [22], the authors compare the Amazon AWS Greengrass and Azure IoT Edge. Both are commercial platforms that can be used to virtualize sensors. The platforms enable the manipulation of scalar data and some types of multimedia streams like

[3]https://mathjs.org/

audio and static images. Both platforms distribute virtual sensors in the edge and the cloud. The authors concludes that Greengrass and Azure Edge performances are similar in many test cases, but they found that Azure Edge exhibits higher end-to-end latency due to the batch-based processing adopted in the multimedia scenario. That higher latency scenario represents a problem for latency-sensitive applications. Besides that, those platforms have only commercial licenses, and they are proprietary software.

Another organization that is heavily working on the standardization of edge computing architectures is the European Telecommunications Standards Institute (ETSI). ETSI defines standards to deal with telecommunications, broadcasting and other electronic communication networks and services. In [23], they provided a framework and a reference architecture for Multi-access Edge Computing (MEC). The document describes a MEC system that enables MEC applications to run efficiently and seamlessly in a multi-access network. This is a general guide to develop architectures to a more specific domain. Our architecture follows some of the standards described in [23]. In particular, the idea of an orchestrator, as we can see in Fig. 1, that shows a complete view about physical devices and software components of the environment and defines in which edge node the application must be started or when the application must to be relocated.

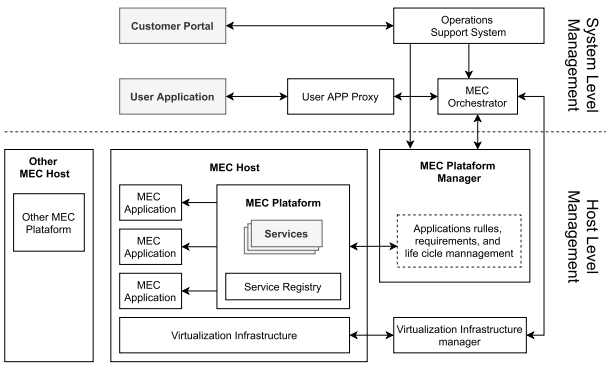

FIGURE 1. MEC System reference architecture [23].

A relevant feature depicted in Fig. 1 is that the infrastructure can handle many MEC Hosts (edge nodes) and, each host can execute different MEC application (virtual multimedia sensors). In this way, different application providers could deliver their applications to be executed inside the MEC infrastructure. Some factors that enable this level of interoperability are the application virtualization and the existence of a Service Registry component.

A comparison among VSAs is depicted in Table 2. The last row of the table represents the features of our distributed V-PRISM proposal. Each column of the table means:

- **Node Management**: We analyze if the SVA allows node management. A node is a computational device where one or more VSs will be executed.

TABLE 2. Comparison between sensor virtualization architectures.

Papers	Node Management	VS Type Setup	API for VS Creation	Resource Allocation	Management Interface	Lightweight Virtualization	Component Deployment	SVA implementation	Multimedia Stream
Kim-Hung [16]	-	✓	-	-	✓	-	E	-	-
Donassolo [17]	✓	✓	-	✓	✓	✓	E	-	-
Wei and Wu [18]	✓	✓	-	✓	✓	✓	E	-	-
Zhang et al. [19]	-	✓	-	-	✓	-	C	-	-
Jeong et al. [20]	-	✓	-	-	✓	-	C	-	-
Santos et al. [21]	-	✓	✓	-	-	-	E	✓	-
ETSI [23]	✓	✓	✓	✓	✓	✓	E	-	-
V-PRISM	✓	✓	✓	✓	✓	✓	E	✓	✓

- **VS Type Setup**: We analyze if the SVA is capable of defining which kind of VS type can be executed in each node of the infrastructure (either edge or cloud), meaning that a single node can run different VS types.
- **API for VS Management**: We analyze if the SVA allows the use of mechanisms that enable remote applications to manage VSs via an API (Application Programming Interface).
- **Resource Allocation**: We analyze if the SVA provides mechanisms for dynamic resource allocation.
- **Management Interface**: We analyze if the SVA has any interface (graphical or via command line) for managing its components.
- **Lightweight Virtualization**: We analyze if the SVA uses a light virtualization method for virtualizing its components.
- **Component Deployment**: We analyze where a VS is deployed. The deployment can be done in the cloud (C) or in the edge (E).
- **SVA implementation**: We analyze if a public implementation is provided.
- **Multimedia Stream**: We analyze if the SVA supports multimedia streams.

In contrast with most of the other architectures, our proposal, V-PRISM, provides multiple improvements, which are: i) lightweight virtualization approach, allowing the execution of multiple virtual multimedia sensors in multiple heterogeneous edge nodes; ii) external applications can manage VMSs through an API; iii) the VMS allocation can be automated by multiple resource allocation algorithms, and these algorithms are selected by the IoMT infrastructure owner; iv) we provide a web interface to manage V-PRISM components; v) programmers can develop and deploy new VS types following the guidelines provided by our architecture; vi) all the features proposed are implemented and available under the MIT License at our GitHub[4]; vii) to the best

[4] https://github.com/midiacom/alfa

of our knowledge, V-PRISM is the only sensor virtualization architecture that supports multimedia streams.

III. DISTRIBUTED V-PRISM ARCHITECTURE

In this section we present the distributed V-PRISM architecture as an extension of our initial work [12]. Our current proposal supports virtualization and management of VMSs in distributed edge nodes.

A. THREE-TIER ARCHITECTURE FOR IoMT

For the design of V-PRISM, we consider that IoMT systems follow a three-tier architecture encompassing: i) the cloud, ii) the edge, and iii) the things tiers, as proposed in [24]. Fig. 2 shows an overview depicting how each entity considered in our proposal is deployed in each tier. The main goals and composition of each tier are:

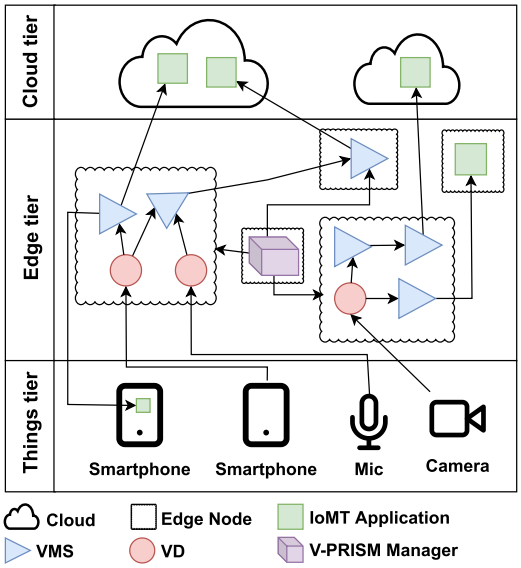

FIGURE 2. V-PRISM three-tier architecture overview.

The **Cloud** tier is the top level of the architecture. The elements that integrate this tier are machines hosted in big data centers, with powerful CPU, memory, storage, and network capabilities. Usually, IoMT applications are hosted in this tier. The cloud can provide almost unlimited resources for applications, but the processing nodes in this tier are far from IoMT devices. This distance implies an increase of the communication latency.

The **Edge** tier is the intermediate layer between the things and the cloud. The elements that make up this tier are resource-constrained machines hosted in small data centers, or single-board computers like Raspberry deployed directly inside the monitored environment. The resources at the edge are limited, in contrast to the cloud, although they are closer to the physical devices. Thus this tier can provide services with lower latency in comparison to the cloud.

The **Things** tier is where multimedia devices are hosted. Media things are typically heterogeneous and produce a massive amount of data. They usually have limited resources available to process and store the generated data. The goal of this tier is to produce data streams. Moreover, devices in this tier may contain, in addition to sensors, some processing capacity, although limited. Therefore, simple processing tasks can be performed in this tier.

V-PRISM architecture encompasses the following types of elements: Virtual Multimedia Sensors (VMS), Virtual Devices (VD), and V-PRISM Manager components. All these elements are deployed in edge nodes, as presented in Fig. 2. A VD receives/collects multimedia streams from a physical device. One or more VMSs can request the multimedia stream from the VD to perform some operation over the data. Finally, the VMS output will be sent to other VMSs or IoMT applications. We consider IoMT applications as any type of external V-PRISM entity that consumes the output of a VMS. Typically IoMT applications are hosted in the cloud, but they can also be hosted in the edge and the things tiers. V-PRISM can handle all these deployment configurations for IoMT applications (at the cloud, edge or things tiers).

The three-tier architecture is an approach designed to explore the integration of the things, the edge and the cloud tiers. In V-PRISM, we adopt this strategy once each tier has its own responsibilities. In the next section, we detail the VMS Categories proposed in this work.

B. VMS CATEGORIES

To facilitate understanding and organizing the different types of VMS, we propose a categorization according to the multimedia stream abstraction provided by the VMS type. Besides helping to organize the various possible types of VMS that can be built on V-PRISM, these categories also potentially serve for the following two purposes:

- The proposed categories group the VMS by the provided functionalities (from the simplest to the most complex ones), so they also help to guide the amount of resources that will be needed to instantiate each type. More complex nodes will require more infrastructure resources for their deployment and execution. Therefore, a minimum amount of resources required for each VMS category can be stipulated and a resource allocation algorithm can use this information for its decision making. Only edge nodes able to provide the minimum resources of a given category would be candidates for the implementation of nodes of the respective type;
- Provide templates for each VMS type, which facilitates developers work.

To the best of our knowledge, no previous work proposed a categorization tailored for VMS. In order to produce a comprehensive organization, we studied the already proposed categorizations for virtual sensors presented in [21], [25], and [16]. Fig. 3 depicts the proposed hierarchical categorization.

The categorization presented in Fig. 3 was inspired on the information classification proposed in [26]. The author coined three concepts to define information abstraction levels. The first one is that information is a **thing**

(information-as-thing) when the information is only data. The second one is **knowledge** (Information-as-knowledge) when an entity makes some reasoning over the information, and the third one is **process** (Information-as-process) when the information consumed by some entity promotes some change in the entity that receives the information. Therefore, inspired by this work, our proposed organization is also composed of three main categories defined as follows. VMS as a **Thing**, when the VMS only forwards the data collected by the device. VMS as a **Process**, when the VMS, besides forwarding the data, adds new features to the virtual device. VMS as a **Service**, when the VMS provides a high-level abstraction feature over the multimedia stream.

We also propose the following subcategories: Replicator, Improver, Converter, Selector, Aggregator, Detector, and Transformer. It is important to note that this is not exhaustive subcategorization. With the advancements in IoT environments, new devices can enable innovative VMS types, forcing the creation of new categories to cover them.

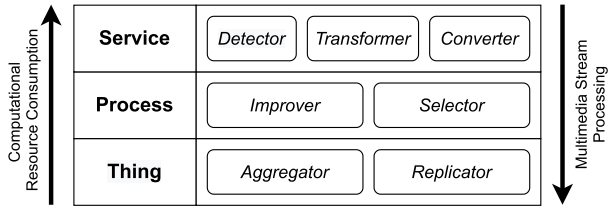

FIGURE 3. VMS hierarchical categorization.

As depicted in Fig. 3, the computational complexity and resource consumption tend to grow when the abstraction level of the VMS category rises. On the other hand, the level of multimedia stream processing reduces as the category level decreases. It indicates that VMSs defined in different categories have different requirements. Thus this hierarchical categorization can be used as a parameter to resource allocation algorithms once a VMS cannot run inside every edge node available.

1) SUBCATEGORIZATION DESCRIPTION

An *Improver* VMS can provide to IoMT applications features that do not exist in the physical multimedia device. A frequent situation in IoT environments is devices deployed without proper firmware updates, which allows security threats. It happens because, typically, an IoMT system can run for years, and during this time, vendors can stop making improvements in the device's firmware. In these situations, an *Improver* VMS can be used to increase the lifetime for legacy multimedia devices already deployed in the environment.

A *Converter* VMS can be used when the multimedia stream produced by the device needs to be changed to meet the IoMT application requirements. For example, if an old microphone can only produce a WAV stream, but the IoMT application requires an OGG audio stream.

The *Selector* VMS category is composed by VMS types that receive multiple multimedia streams as input and then choose only one stream as output. All the multimedia input streams must be of the same type. The multimedia stream is not modified, only forwarded to the IoMT application. Each *Selector* VMS has at least one selection function. The selection function can use a simple QoS device stream analytics or a complex multimedia pattern recognition. For pattern recognition, the VMS can implement artificial intelligent algorithms as proposed in [27].

An *Aggregator* VMS receives multiple multimedia streams and sends a single combination of them to the IoMT application. There are two classes of *Aggregator* VMS. The first one will aggregate multimedia streams of the same type (for example, two or more video streams), and the second one will aggregate multimedia streams of different types (for example, audio streams and video streams).

The *Detector* VMS category encompasses VMSs that perform event detection in a multimedia stream. Artificial intelligence techniques can be used in these VMS [27]. It is essential to mention that the VMS runs in an edge node, usually with low capabilities. Thus, the algorithm selected to execute the detection needs to be compatible with the resources available in the edge node. It is possible to develop a *Detector* VMS to perform complex event detection. In [28], the authors presented a complex real-time event detection framework for resource-limited multimedia sensor networks. The same type of strategy can be applied to a VMS running in an edge node.

A *Transformer* VMS changes the nature of the stream. It is used when the multimedia stream needs to be transformed into another data type. An example of transformation is when video streams must be converted into a file or an audio stream must be converted into text. A car license plate detection VMS is an example of a *Transformer* and also a *Detector*.

This subcategorization is a partial list, and it can be extended as new VMS types are developed. In the following sections, we present the V-PRISM logic components. The components and interactions between them will be described in detail.

C. V-PRISM COMPONENTS

In recent years, there has been a lot of effort to specify a general-purpose architecture to run and manage applications in the edge [8], [29], [30]. Our architecture follows the meta specifications presented by ETSI [23]. As it is a general specification, it leaves domain detail definitions to more specific architectures.

V-PRISM logic components are depicted in Fig. 4. They are responsible for processing multimedia streams produced by multimedia devices and consumed by IoMT applications. IoMT applications are typically deployed in the cloud, and physical multimedia devices are typically placed in the things tier. Stream processing is performed by the set of virtual multimedia sensors (VMS) provided and hosted by V-PRISM.

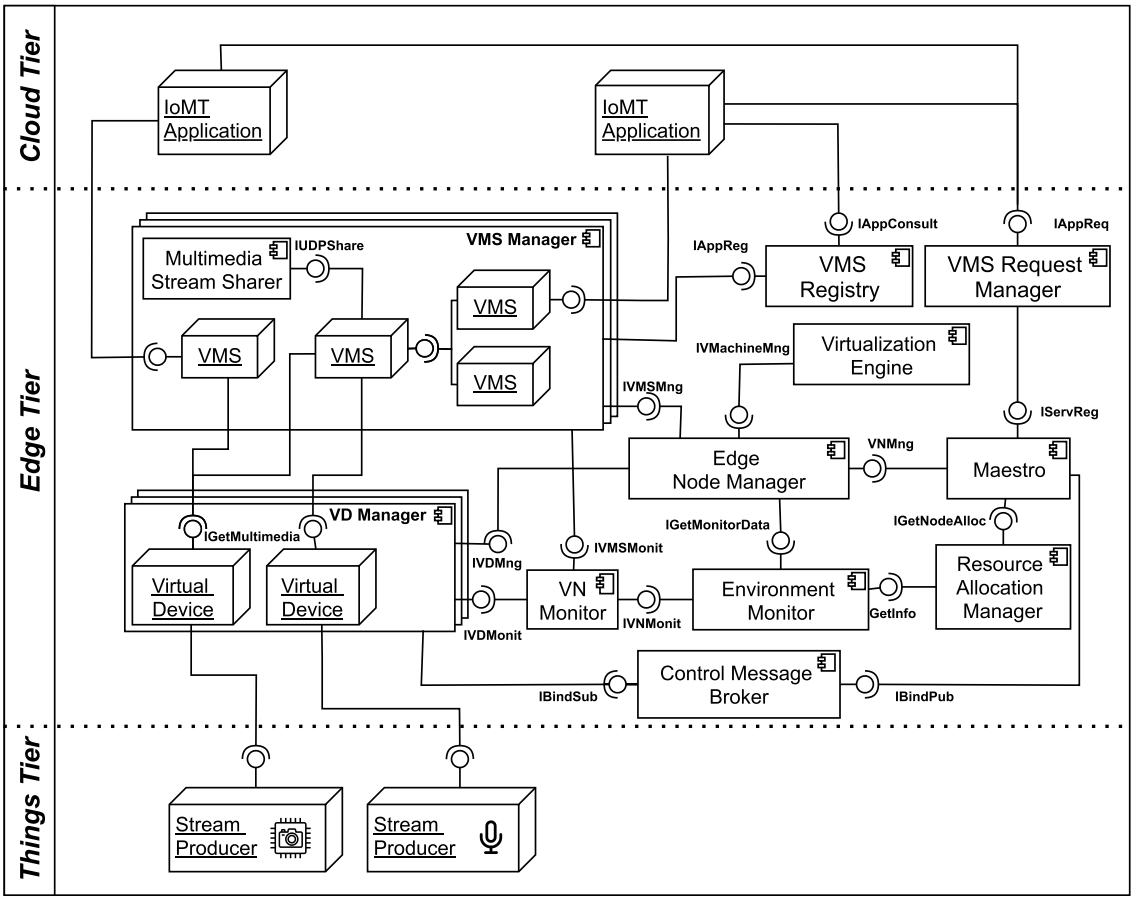

FIGURE 4. V-PRISM distributed architecture.

All V-PRISM components can be deployed in multiple edge nodes in a distributed fashion.

1) VIRTUAL MULTIMEDIA SENSOR

A Virtual Multimedia Sensor (VMS) is the architectural component responsible for providing multimedia stream processing. Therefore, a VMS component performs some process in a multimedia stream produced by physical devices (media things). Examples of processing are converting video formats, extracting multimedia features, aggregating new characteristics to already existing multimedia streams, etc. For each specific type of multimedia stream processing, there will be one VMS type.

The data processed by a VMS can be used by two types of entities. The first one is the IoMT application that requested the corresponding stream. The second one is another VMS. Therefore, VMSs can be chained, where the output of a given VMS is used as input for the next VMS. Chaining VMSs allows reusing the output of a stream processing while creating complex processing flows to accommodate application requirements.

Each VMS has one or more input ports, connected to a virtual device (VD) or to another VMS, and one output port. It can be noticed in Fig. 4 that the connection between a VMS and an application is unidirectional. This fact implies that the application does not have a direct communication channel with the VMS. If the application needs to send a control message to a VMS, this must be done using the VMS Registry component.

A VMS is defined as a tuple $VMS = \{P, I, S, D, T, SH, E\}$ where $P = \{par_1, \ldots, par_n\}$ is a set of configuration parameters $par = \{name : value\}$ that defines how that specific VMS will process the multimedia stream, such as saturation level, noise level, video quality, etc.; $I = \{en_1, \ldots, en_n\}$ is a set composed of VD or VMS entity providers of input streams. S is the software that performs the corresponding multimedia stream processing, i.e., this value can be the container ID if the software is running inside a Docker environment, the PID if its running on LXC (Linux Containers), or any other type of software identification; D is an $(IP, PORT)$ tuple that defines the first address used to send the output result (it can identify an IoMT application or another VMS); T refers to the VMS type, each implementation of V-PRISM can choose the best approach to identify the type, it can be the type name, a pointer reference, or an ID; SH is a boolean value that defines if the processed output stream can be shared with other VMSs or IoMT applications; E is an optional parameter that defines in some cases in which edge node the VMS must be executed.

2) VIRTUAL DEVICE

A Virtual Device (VD) is the architectural component responsible for forwarding a multimedia stream produced by a multimedia device to a given VMS. Each VD is connected to one physical device. Each VD is programmed to connect to a specific device type (e.g. USB camera) and to send a specific type of multimedia stream, e.g. H.264 video stream. This approach makes the architecture flexible for incorporating different VD types. The VD output type cannot be changed in execution time. The VD is data stream agnostic since it has no knowledge of the content being carried. This feature decreases the complexity of creating new VD types.

The association between the physical device and a VD must be defined using V-PRISM stream descriptors during the setup process and cannot be changed after the VD starts running. The VD type must be compatible with the type of data stream and the communication protocol available at the physical device. A VD operates as a driver, interpreting specific data formats and protocols. In contrast, it is possible to have many associations between a VD and VMSs. VD and VMS have a weak association since it can be established and changed during execution time.

A VD is defined as a tuple $VD = \{P, C, S, T, E\}$, where $P = \{par_1, \ldots, par_n\}$ is a set of parameters $par = \{name : value\}$ that identifies the device, such as a path for the device, the device IP address, etc.; $C = \{cnf_1, \ldots, cnf_n\}$ is a set of the device configurations $cnf = \{name : value\}$, each device has different configurations, for example, video resolution or audio quality; S is the software that performs multimedia stream collecting and forwarding, i.e., this value can be the container ID if the software is running inside a Docker environment, the PID if its running on LXC, or any other type of software identification. T is the VD data type, it defines the type of the VD output, which can be audio or video; E is an optional parameter that defines in which edge node the VD must be started. The edge node can be identified in many ways depending on the implementation infrastructure. For example, in a Docker environment, it can be the $(IP, PORT)$ where the Docker API is listening.

3) V-PRISM MANAGER COMPONENTS

The **VMS Registry** component is used to manage and provide a descriptor list about the availability of VMS types in each edge node. The IoMT application will query this list before making the VMS creation requests. The descriptor list can be fully retrieved or queried by specific parameters such as location, VMS categories, and VMS types. For example, an IoMT application can request a list of all VMSs that perform voice recognition in a specific location.

The **VMS Request Manager** (VRM) component is responsible for receiving requests made by the IoMT application to create or destroy a VMS. It is through the VRM that external demands created by IoMT applications reach other V-PRISM components. This component uses a message-driven communication. Such approach allows heterogeneous systems to communicate in a loosely coupled way, and it fits the dynamic nature of IoT and CoT systems. The communication protocol used between these entities is not in the scope of this work.

Maestro is the first component executed after a VMS creation request is admitted by VRM. It is a V-PRISM core component and has two main functions. The first one is to manage the VMS life cycle and the second one is to orchestrate the VMS stream pipeline. Maestro works together with other components to provide these functions. In the VMS life cycle management, Maestro uses the Edge Node Manager component to send commands to create, destroy, or move a VMS from one edge node to another. Maestro consumes functions provided by the Resource Allocation Manager component to determine in which edge node a VMS will be created.

The **Resource Allocation Manager** (RAM) is the component that defines in which edge node the requested VMS must be instantiated. Different types of resource allocation algorithms can be implemented in this component. Some algorithms proposed in other studies [30]–[33] could be used as well.

Each VMS and VD can be developed using the technology that better fits its requirements, for example, a VMS that detects movement can be developed in Python using OpenCV. In contrast, a VD that collects data from a microphone can be developed using C language. This means that, for V-PRISM, the VMS is agnostic about the technology adopted in the component development. To overcome the heterogeneity of hardware, software and network communication standards, each VMS and VD will run inside its private virtual environment based on light virtualization. The **Virtualization Engine** component uses the low-level API provided by the virtualization platform to virtualize the components of V-PRISM. Another responsibility taken by the virtualization engine component is the communication mechanism between components. It must abstract the heterogeneity of the network environment. In this manner, virtual entities can focus only on their main tasks.

The **Edge Node Manager** (ENM) is the component responsible for managing all edge nodes that run V-PRISM components. In the MEC Architecture [23], ENM has the same functionalities described in the MEC Platform Manager. ENM is responsible for the following functions: receiving virtualized resource fault reports; measuring the performance of the virtualization infrastructure; providing elements for managing V-PRISM core components, and preparing the virtualization infrastructure to run VMS and VD images.

The **Environment Monitor** component gathers data (CPU, memory used, battery level) from the edge nodes. The **Virtual Node Monitor** collects statistics data from the virtual entities (VMS and VD). The freshness of the data depends on the capabilities of the devices in the edge environment. Typically these data are not collected in real-time because the resource allocation algorithm is time-constrained.

The **VMS Manager** component is responsible for providing interfaces to manage VMSs. Moreover, it keeps track of VMS types available in each edge node. Each edge node has the capabilities to run a subset of all VMS types available. A VMS type is implemented by a software that performs some processing over a multimedia stream.

The **VD Manager** (VDM) component is responsible for providing interfaces to create, destroy, and manage virtual devices. Each VD stores physical sensor metadata, like the format of the generated stream, the physical location of the sensor, its data-sampling rate, etc. The attributes stored about each sensor can be distinct and depend on the physical nature of the multimedia sensor.

The **Stream Sharer** (SS) component provides a function that enables the VMS to send a multimedia stream to more than one destination. By default, the VMS has many input ports but only one output port. This component allows the VMS to focus solely on the main task that is multimedia processing and not in the process of sharing the result stream.

D. V-PRISM OPERATION

1) V-PRISM DEPLOYMENT

V-PRISM is composed of many distinct components. The components can be deployed in different edge nodes. Every edge node that compounds V-PRISM must have at least the Virtualization Engine component, and one or more VMS types or VD types.

Before entering the V-PRISM environment, an edge node must be registered in the Edge Node Manager component. The edge node must also have a unique identifier (ID) that will be used to address the control messages sent to it. Typically the virtual engine adopted in the development already provides some ID. In Docker, for example, the ID is composed by a string with 26 characters.

An edge node can host virtual entities like VMSs and VDs. The virtual entity software must be deployed and available at the edge node. Real-time software deployment can increase virtual entity startup time.

Network communication between all edge nodes that compound V-PRISM environment is mandatory. For improving V-PRISM security, we advocate that all edge nodes must be part of the same private network, which can be either a physical or a virtual network. The only components that must have a direct connection to the Internet are the VMS Registry and VMSs that send data to an IoMT application hosted outside the private network.

Each physical device that produces a multimedia stream is attached to only one virtual device (VD). Before a VMS uses the multimedia stream provided by a VD, the parameters for VD configurations must be informed, and the virtual device must be started.

2) INITIALIZATION OF A VMS

The initialization of a VMS has two phases. The first is the admission phase, where the VMS Request Manager (VRM) component defines if the IoMT application request is valid or not. After that, the second phase is started and it consists of a call to Maestro to start the VMS creation. Fig. 5 shows the sequence diagram of the second phase.

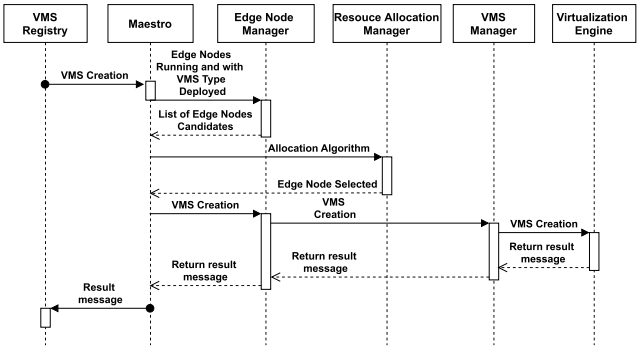

FIGURE 5. VMS creation sequence diagram.

First, the VMS Registry component sends to Maestro the request to create a new VMS. Maestro requests the Edge Node Manager component to retrieve the list of all edge nodes that are running and have already deployed the corresponding VMS type. Finally, the list of candidate edge nodes and the information provided by the IoMT application are sent to the Resource Allocation Manager. It will choose, based on a resource allocation algorithm, one edge node to run the VMS, and that information will be sent to VMS Manager that will call the Virtualization Engine to create the VMS.

IV. ALFA DISTRIBUTED PLATFORM

In this section, we present ALFA, a distributed platform based on V-PRISM. ALFA provides essential features to enable IoMT to use edge computing to perform multimedia processing. Fig. 6 depicts ALFA deployment diagram. For better comprehension, we also provide a video demonstration that can be seen at https://www.youtube.com/watch?v=-kC9T2KIp94.

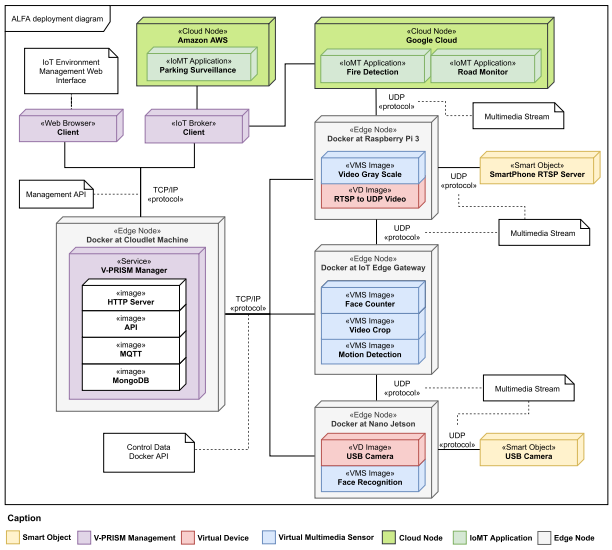

FIGURE 6. ALFA deployment diagram.

As mentioned, the edge tier comprises heterogeneous nodes with limited computational resources (compared to the cloud tier). Edge nodes can range from single-board computers [34], to IoT gateways or even micro datacenters [35]. ALFA components shown in Fig. 6 can be distributed among different edge nodes, provided that it has the minimum required capabilities.

ALFA was built using Docker. Docker enables the adoption of containers. A container is a standard unit of software that contains the code and all dependencies to execute the system. Thus the application runs quickly and reliably when moved from one computing environment to another. The adoption of containers in a heterogeneous environment, like the edge, can improve network efficiency, computation, and data storage [36].

Another useful technology that Docker offers is the network overlay driver. Since each VMS or VD can be deployed in a different edge node attached to a different network and geographically distributed, we need to implement a way to enable container-to-container communication. If the edge nodes are attached to different IP subnetworks, the Docker overlay network enables direct communication between the containers as if they were in the same network.

To implement ALFA components, an edge node should be a device endowed with storage, memory, and CPU, capable of running Docker version 19.0 or higher. On the other hand, not every VMS or VD needs to have Internet access. Only VMSs that send data to IoMT applications running in the cloud must have Internet connectivity.

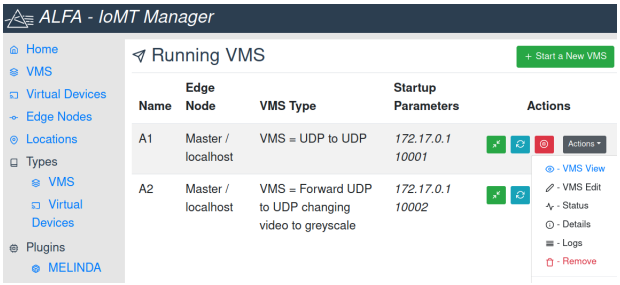

FIGURE 7. Web interface for listing VMS instantiated.

To facilitate managing ALFA infrastructure, we developed a web application, depicted in Fig. 7 and 8. The IoMT infrastructure owner uses the interface depicted in Fig. 7 to manage the running VMS instances. In Fig. 6, the HTTP Server inside V-PRISM Manager is running the web application service. It is important to note that this service can be deployed in any of the edge nodes of the environment. The main modules of the web application are **VMS**: it lists all created VMSs inside any edge node and allows VMS management; **Virtual Devices**: it lists all the virtual devices created that can be used as a source of multimedia data; **Edge Nodes**: it manages all edge nodes that will be used to run a VMS or a VD; **Locations**: it manages all the places where a physical device can be deployed. A location can be a meeting room, a hall, etc.

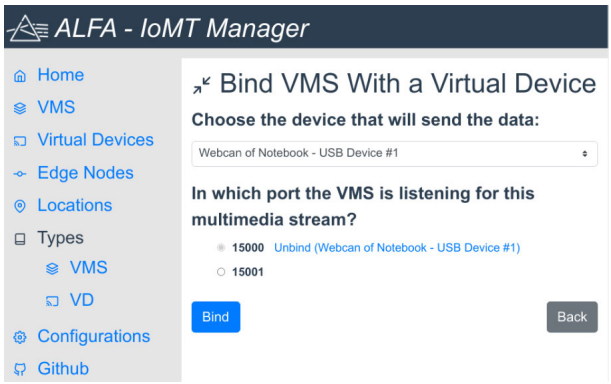

FIGURE 8. Web interface for bind VMS and VD.

Many Virtual Devices can be placed in the same location. **VMS types**: it manages all the possible VMS types that can be initiated in ALFA deployment; **Device Types**: it manages all the possible VD types that can be initiated in ALFA deployment.

As mentioned previously, a VD can be a source of a multimedia stream to many VMSs. When we associate a VMS with a VD, we bind them. In Fig. 8, we show the web application interface used to bind a VMS to a VD. The bind will be between a *Video Merge* and some VD. The *Video Merge* VMS has two input ports, we can see that the port 15000 is already attached with the VD USB Webcam, and the port 15001 can be attached with another VD. Each VMS developer will define each port the VMS will listen to.

V. ALFA VMS, VD AND USE CASE SCENARIO

V-PRISM architecture was developed to be extended by the creation of new VMS and VD types. In this section, we present a set of VDs and VMSs currently available in ALFA platform. By presenting those VDs and VMSs, we explore some possibilities that V-PRISM brings to IoMT and edge in the use case section.

A. AVAILABLE VIRTUAL DEVICES

In this section, many VD types developed in ALFA are presented. Each VD is encapsulated in a Docker image that contains the source code of software that collects or receives data from an IoMT device. New VD types can be developed and installed as needed.

The *RTSP to UDP Video* is a VD created to establish a connection with devices that use Real-Time Streaming Protocol (RTSP). RTSP [37] is an application-level protocol used over audio and video streams to provide controlled, on-demand delivery of real-time data. As a widely adopted protocol, data from many IoMT devices can be accessed over RTSP. Because of that, we developed a VD capable of connecting to devices that use this protocol. This VD type collects data from a device that is not physically attached to the edge node where the VD is running. The physical device uses WiFi or another wireless infrastructure to transfer the data to the edge node.

The *USB Camera* and *Mic* are VDs that collect data from devices attached to the edge node where the VD container is running. These VD are typically used when the edge node is a single board-computer like Raspberry Pi. By default, a Docker container is not allowed to access data and the attached devices at the host machine. To allow it and bind a host device (camera, microphone) to a container, when the container is started, it is necessary to explicitly give access and map the external device to an inside path at the Docker container.

For example, in Fig. 6, the *Smartphone RTSP Server* and *USB Camera* devices are multimedia devices whose streams are sent via UDP to a VD. The first one uses a wireless connection, and the second one is attached to the edge node. Therefore, ALFA can handle different types of devices.

B. AVAILABLE VIRTUAL MULTIMEDIA SENSORS

In this section, some of the VMSs developed in ALFA will be presented. Each VMS, as we mentioned previously, is encapsulated in a Docker image that contains the source code to process a multimedia stream and deliver the result to an IoMT application or another VMS. As ALFA is flexible, new VMS types can be developed and installed as needed.

The *Video Grayscale* VMS converts a colored video stream into grayscale. This VMS is useful in situations where the IoMT application runs some transform algorithms, like the HoughCircle present in OpenCV. The first step of a Hough transform is to convert the video stream to grayscale, because of that, transferring the colored video will be a resource waste. In our proposed categorization, presented in Section III-B, this VMS is a *Transformer*.

The *Video Crop* VMS cuts part of a video stream. This VMS is useful in situations where the application will need only a fraction of the complete video. For example, a surveillance application can be interested only in the video of the door instead of the video of all the environment. In our categorization, this VMS is a *Transformer*.

The *Video QR Code Detection* VMS can be used to detect and extract data from a QR Code inside a video stream. This VMS is useful when a QR Code is used to interact with a remote application using QR Code and camera to execute some task. The QR Code data extracted can be used by the application as a signal to start the execution of another task. In our categorization, this VMS is a *Detector*.

The *Noise Detector* VMS can post in an MQTT topic if the sound volume of the environment was higher than a configured threshold. This VMS is useful in the Industrial Internet of Thing (IIoT) to monitor facilities and machines where the noise level can be used to predict a dangerous incident. In our categorization, this VMS is a *Detector*.

The *Face Counter* VMS can post in an MQTT topic the number of faces identified in a video stream. This VMS can be combined with another VMS to count the number of people in an ambient. This VMS was created to exemplify the adoption of deep learning for multimedia processing. We used library *Face Recognition* available at GitHub.[5] In our categorization, this VMS is a *Detector*.

One of the premises of V-PRISM is the flexibility to incorporate new VD types. it allows that VMSs and VDs can be built using different techniques. In ALFA, we test these two features, by developing multiple VMSs and VDs for different goals and using different techniques, to validate the approaches proposed in V-PRISM.

C. USE CASE SCENARIO

Here we describe the adoption of V-PRISM, IoMT, VMSs, edge, and cloud computing providing complex service for users. The use case is deployed in a *car parking* and works as a surveillance system. Fig. 9 depicts a car being stolen (1) and, when the car's alarm starts to make noise, the sound is captured by the *smart pole* microphone (3) and, processed by a *VMS Alarm Detection* hosted in the edge node (5).

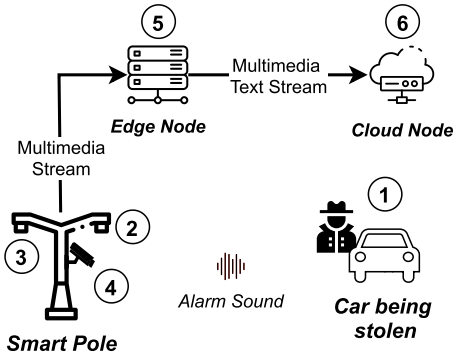

FIGURE 9. Surveillance system in a car parking.

The event detected by *VMS Alarm Detection* dispatches a signal to turn on the *smart pole* light (2). The video stream provided by *smart pole* camera (4) is delivered to a *VMS Video Record* that stores the video in the edge node (5). The same video stream is processed by a *VMS Person Detection* and, if a person is detected in the video stream, the image frame is stored in a security system hosted in the cloud (6). As presented, V-PRISM can be used to deploy services composed of multiple independent VMS types.

VI. EVALUATION

As reported in our initial work [12], we analyzed that the adoption of V-PRISM and virtual multimedia sensors in IoMT environments improves the resource consumption in the IoMT device and in the IoMT network. In the current work, we present novel experiments to identify if the placement of V-PRISM components affects the QoS aspects of IoMT applications.

The latency and total time of video stream processing are critical QoS parameters [7], [38] [39]. The experiments described in this section assesses QoS aspects of video processing in VMS deployed in the edge and cloud nodes.

[5]https://github.com/ageitgey/face_recognition

In order to plan the experiments, we adopted the Goal Question Metric (GQM) [40] approach. GQM model is a hierarchical (three levels) structure that, in each level, refines the granularity of what is relevant in order to provide reliable insights on a phenomenon. A **goal** represents which phenomenon should be analyzed, where each goal can be represented as one or more **questions**, and for each question, a set of metrics will be used to answer it. The goal of this evaluation is *G1: Analyze if the adoption of V-PRISM to process multimedia streams in the edge improves QoS aspects of IoMT application in comparison with the cloud approach*. Table 3 describes questions Q1 and Q2 that the experiments aim at answering. Table 4 depicts the metrics collected in the experiments to answer these questions.

TABLE 3. Questions for G1-VMS placement experiment.

Question	Description
Q1	Does the deployment of the proposed architecture in the edge reduce the total data loss during the multimedia stream processing, compared with the cloud deployment?
Q2	Does the deployment of the proposed architecture in the edge reduce the total delay in the IoMT applications, compared with the cloud deployment?

TABLE 4. Summary of the metrics used to answer the questions of G1 goal.

Metric	Metric description	How is it calculated?	Question
FRL	Frame Loss (FRL) is the percentage of lost frames with the image that should trigger the event detection alert.	It is the division of QR Code frames detected by the total QR Code frames sent by the device.	Q1
QFD	Quantity of Frames Detected (QFD) is the percentage of frames with the image that will trigger the event detection.	QFD = 1 - FRL	Q1
QDE	Quantity of Data Extracted (QDE) is the percentage of frames whose data could be extracted by the VMS.	It is the division of the number of QR Code frames detected whose data could be extracted by the number of QR Code frames detected.	Q1
FFD	First Frame Detection (FFD) is the time when the first video frame was available to be processed.	It is the difference between the instant when the virtual device starts sending the video stream and the instant when the first video frame with QR Code was available in the VMS.	Q2
FDD	Frame Detection Difference (FDD) is the difference between the time instants of successive detection events.	In the video used as input for the experiment, each QR Code is displayed for 1 second, the FDM represents the time difference between two consecutive detection values.	Q2
DTN	Detection Time of Noise (DTN) is the time difference between the detection of successive noise.	In the input audio, each noise was played with 5 seconds interval, DTN is the time difference of two consecutive noise detection events.	Q2

For the deployment of the experiments, we used the following environment configurations. The edge node was emulated using a virtual machine with 1 GB RAM and 1 vCPU, 1.8 GHz, Intel i7 8565U. The edge node is running inside a local WiFi network at Niterói, Brazil, and from now on this node will be only called edge node. The cloud node was a virtual machine with 1 vCPUs, 2.5 GHz, Intel Xeon Family, and 1 GB of RAM. It is located physically at the United States at North of Virginia in Amazon Cloud, and from now on this node will be only called cloud node. The one-way-delay was calculated using the ping command round-trip-time (RTT). The following subsections detail our experiments and discussion of the obtained results.

A. QR CODE DETECTION PROCESSING

For this experiment we use the *Video QR Code Detection* VMS, described in Section V-B. It was used as an event detection over a video stream. This VMS detects and extracts information of a QR Code inside a video frame. The video used as data source for this experiment has 70 seconds and was encoded in H.264 with 25 frames per second (1750 frames in total), and with 200 $\times$ 200 spatial resolution. We inserted a QR Code image in 450 of these frames.

We run the experiment for the edge and cloud nodes ten times. Table 5 shows the mean values of the data collected. The column named *QR Code Detected* represents the number of video frames with a QR Code that were detected. The *Data Extracted* column represents the number of video frames with a QR Code detected whose stored data could be retrieved. The *FRL* (Frame Loss) column represents the percentage of frames with QR Code not detected by the VMS. The column named *QFD* (Quantity of Frames Detected) represents the percentage of frames with QR Code detected in the VMS. The column named *QDE* (Quantity of Data Extracted) represents the percentage of frames with QR Code whose data were extracted by the VMS.

TABLE 5. Mean FRL, QFD and QDE metrics over 450 QR Code frames (ms).

Tier	QR Code Detected	Data Extracted	FRL	QFD	QDE
Edge	318	219	29%	71%	69%
Cloud	357	303	20%	80%	85%

Extracting features from a video stream, like QR Code detection, is an intensive CPU and memory task. Table 5 shows that the cloud's computational power was decisive to obtain better processing results in contrast with the edge. Some QR Codes were detected, but the data was not extracted due to network packet loss or even CPU shortage.

To obtain metrics *First Frame Detection* (FFD) and *Frame Detection Difference* (FDD), we run another experiment depicted in Fig. 10. To execute the components of this experiment, we used the Containernet [41] network emulator. Containernet is a Mininet extension [42], and it allows the execution of Docker container as host machines inside the emulated network. As each VMS is running inside Docker

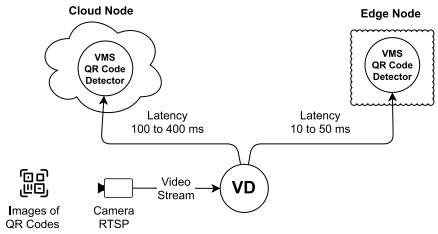

FIGURE 10. Experiment design to obtain metrics for total latency.

containers, we can connect V-PRISM components to Containernet network to run the experiment. The script that defines the topology was developed in Python. The network links between the elements have parameters that can be configured depending on the experiment goals. We used the delay parameter to differentiate the traffic for cloud and edge node artificially. In this emulation, we defined the same limits of CPU and memory for the edge and cloud nodes, since we are interested in the delay metric only. We also ran this experiment ten times for the edge and for the cloud nodes.

In Fig. 11, each bar represents a node where the VMS *Video QR Code Detection* is running. The nodes with latency between 10ms and 50ms represent the edge nodes. The nodes with a latency greater than 50ms represent the cloud nodes. This figure shows the average FFD metric with 95% confidence interval. We can observe that the time for the detection of the first QR Code (y-axis of the graph) grows as the latency grows. It means that even in a situation where there is sufficient bandwidth between the IoMT device and the node, the latency affects the total time detection significantly.

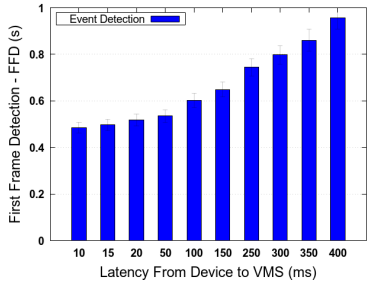

FIGURE 11. *First Frame Detection* (FFD).

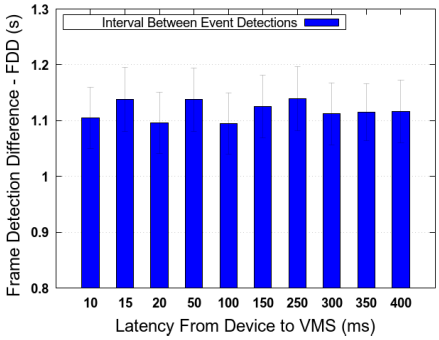

FIGURE 12. *Frame Detection Difference* (FDD).

In Fig. 12, each bar represents a node where the VMS *Video QR Code Detection* is running. As before, the nodes with latency between 10ms and 50ms represent the edge nodes, and the nodes with a latency greater than 50ms represent the cloud nodes. This figure shows the average FDD metric with 95% confidence interval. The interval between each successive QR Code frame in the original video is 1 second. We can observe that the average FDD metric is approximately 1.1 seconds independently of the latency. It means that the latency does not affect the time between successive event detection in the same video stream.

B. NOISE DETECTION

For this experiment, we deployed two *Noise Detector* VMS, one in the edge node and another in the cloud node. The microphone was placed at the same room of the edge node where the virtual device (VD) was deployed. A noise source was configured to make a beep sound, and it was played 80 times with 5s interval between each beep.

The delay between the VD and the cloud was 67.5ms, whereas the delay between the VD and the edge node where the VMS was deployed was negligible. The available bandwidth between the edge environment and cloud was 100Mbps. We performed multiple tests in different moments to minimize the network congestion effect.

The audio stream produced has a 1Mb/s bitrate. The stream, as we can see in Fig. 13, is sent in parallel to both VMS A (edge) and B (cloud). The time interval between successive noise detection events in the VMS in the cloud and edge was calculated, named DTN (Detection Time of Noise) metric. Table 6 depicts the results of the experiment. The average noise detection time in the edge was 918ms with a standard deviation of 88ms. In the cloud, it was 967ms with a standard deviation of 137ms. The noise detection average time in the edge was 49ms lower in comparison to the cloud.

The standard deviation of detection in the edge was lower than the cloud, thus pointing to higher predictability of delay in the edge than in the cloud. Another important point to notice is that the average difference between detection time

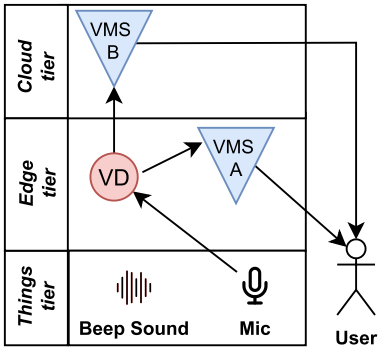

FIGURE 13. Audio processing experiment in edge and cloud nodes.

TABLE 6. DTN metric (ms).

	AVG Noise Detection	STD Deviation
Edge	918	88
Cloud	967	137

in the edge and in the cloud was less than the delay between the VD and cloud, meaning that cloud processing time was shorter than edge processing time. This is expected, since the cloud has a higher computing power than edge devices. Therefore, we can conclude that the choice between edge or cloud processing should consider not only the delay between the physical device and the cloud but also the processing time of the multimedia stream.

C. DISCUSSION

Based on FRL, QFD, and QDE metrics, we can answer *Q1: Does the deployment of the proposed architecture in the edge reduce the total data loss during multimedia stream processing, compared with the cloud deployment?* as follows. The adoption of V-PRISM in the edge could not reduce the overall data loss based on our experiment. We conclude that the leading cause of this behavior is that the cloud has more computational power and can process the multimedia stream at a higher rate than the edge nodes.

Based on metrics FFD, FDD, and DTN, we can answer*Q2: Does the deployment of the proposed architecture in the edge reduce the total delay in the IoMT applications, compared with the cloud deployment?* as following. The adoption of V-PRISM in the edge can reduce the overall delay during the processing of multimedia stream, but the time difference between successive event detection is still the same, independently of VMS placement.

Based on the results presented, we conclude that the placement of VMS in edge nodes (in contrast with the cloud) to run CPU and memory-intensive tasks cannot reduce the total data loss. However, a VMS placed in an edge node, in comparison with the cloud node, can reduce the total time for event detection in a video stream. Thus, goal G1 was achieved, considering the delay QoS metric.

VII. CONCLUSION

In this work, we presented V-PRISM, an innovative architecture to virtualize and manage virtual multimedia sensors. Our proposal relies on a three-tier architecture where devices, placed in the things tier, generate multimedia streams that are processed by entities called virtual multimedia sensors (VMS). VMSs are deployed in multiple edge nodes, and their output data is delivered to IoMT applications.

In contrast with other architectures, our proposal improvements are i) adoption of lightweight virtualization; ii) flexible management of VMS through standardized API; iii) multiple allocations policy can be executed. To the best of our knowledge, V-PRISM is the first sensor virtualization architecture that supports multimedia things.

We validated the V-PRISM architecture by implementing a distributed IoMT platform named ALFA. It instantiates the components of our three-tier architecture. ALFA was released under an open-source license available at GitHub[6] under the MIT License.

[6] https://github.com/midiacom/alfa

Another novel aspect of our implementation is the creation and provision of several types of VMSs and VDs. These VMSs and VDs will help the developers to create their own components. ALFA provides an extensible mechanism to run different types of algorithms to resource allocation that are used to the deployment of VMS in multiple edge nodes.

We also performed experiments about the VMS placement in edge and cloud nodes. We concluded that the functionalities provided by VMSs are a crucial factor in determining which kind of node is better for VMS placement.

As future work, we are going to develop more resource-intensive VMSs for facial recognition and speech translation in order to further investigate QoS aspects. We also plan to specify a domain-specific configuration language to instantiate and run ALFA components. We are currently working on an integration with the FIWARE platform [43].

Some VMSs receive a multimedia stream from many virtual devices whose data may be provided by different tools. In this work, we did not discuss this feature in detail, but it will be important to develop temporal synchronization mechanisms in the future. We will also introduce a virtual entity for dealing with multimedia actuators in V-PRISM.

ACKNOWLEDGMENT

Débora Muchaluat-Saade and Flávia C. Delicato are CNPq Fellows.

REFERENCES

[1] A. Botta, W. Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Generat. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.

[2] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A survey on sensor-cloud: Architecture, applications, and approaches," *Int. J. Distrib. Sensor Netw.*, vol. 9, no. 2, Feb. 2013, Art. no. 917923.

[3] M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, "A survey on virtualization of wireless sensor networks," *Sensors*, vol. 12, no. 2, pp. 2175–2207, Feb. 2012.

[4] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 553–576, 1st Quart., 2016.

[5] T. Barnett, J. Jain, A. Usha, and T. Khurana, "Cisco visual networking index (VNI) complete forecast, 2017–2022," Cisco, Tech. Rep. C11-741490-00 11/18, 2018.

[6] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, "Internet of multimedia things: Vision and challenges," *Ad Hoc Netw.*, vol. 33, pp. 87–111, Oct. 2015.

[7] A. Nauman, Y. A. Qadri, M. Amjad, Y. B. Zikria, M. K. Afzal, and S. W. Kim, "Multimedia Internet of Things: A comprehensive survey," *IEEE Access*, vol. 8, pp. 8202–8250, 2020.

[8] A. J. V. Neto, Z. Zhao, J. J. P. C. Rodrigues, H. B. Camboim, and T. Braun, "Fog-based crime-assistance in smart IoT transportation system," *IEEE Access*, vol. 6, pp. 11101–11111, 2018.

[9] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[10] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1826–1857, 3rd Quart., 2018.

[11] *IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*, IEEE Standard 1934-2018, IEEE Communications Society, 2018.

[12] A. L. E. Battisti, D. C. Muchaluat-Saade, and F. C. Delicato, "V-PRISM: An edge-based IoT architecture to virtualize multimedia sensors," in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, Jun. 2020, pp. 1–6.
[13] Y. B. Zikria, M. K. Afzal, and S. W. Kim, "Internet of multimedia things (IoMT): Opportunities, challenges and solutions," *Sensors*, vol. 20, no. 8, pp. 1–8, 2020.
[14] Q. Wang, Y. Zhao, W. Wang, D. Minoli, K. Sohraby, H. Zhu, and B. Occhiogrosso, "Multimedia IoT systems and applications," in *Proc. Global Internet Things Summit (GIoTS)*, no. 2, Jun. 2017, pp. 1–6.
[15] R. G. Crispen and L. D. Stuckey, "Structural model: Architecture for software designers," in *Proc. TRI-Ada*, 1994, pp. 272–281.
[16] L. Kim-Hung, S. K. Datta, C. Bonnet, F. Hamon, and A. Boudonne, "A scalable IoT framework to design logical data flow using virtual sensor," in *Proc. WiMob*, no. 2, Oct. 2017, pp. 1–7.
[17] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos, "Demo: Fog based framework for IoT service orchestration," in *Proc. CCNC*, 2019, pp. 1–6.
[18] X. Wei and L. Wu, "A new proposed sensor cloud architecture based on fog computing for Internet of Things," in *Proc. IEEE Int. Congr. Cybermatics*, Jul. 2019, pp. 615–620.
[19] J. Zhang, Z. Li, O. Sandoval, N. Xin, Y. Ren, R. A. Martin, B. Iannucci, M. Griss, S. Rosenberg, J. Cao, and A. Rowe, "Supporting personizable virtual Internet of Things," in *Proc. UIC/ATC*, 2013, pp. 329–336.
[20] Y. Jeong, H. Joo, G. Hong, D. Shin, and S. Lee, "AVIoT: Web-based interactive authoring and visualization of indoor Internet of Things," *IEEE Trans. Consum. Electron.*, vol. 61, no. 3, pp. 295–301, Aug. 2015.
[21] F. Santos and R. Guerra, "OSIRIS Framework: Construindo sistemas de monitoramento com redes de sensores sem fio para compartilhar dados," in *Proc. SBRC*, 2015, pp. 1–8.
[22] A. Das, S. Patterson, and M. Wittie, "EdgeBench: Benchmarking edge computing platforms," in *Proc. UCC Companion*, 2019, pp. 175–180.
[23] *Multi-Access Edge Computing (MEC); Framework and Reference Architecture*, Standard MEC 003-V2.1.1, ETSI, Valbonne, France, 2019.
[24] W. Li, I. Santos, F. C. Delicato, P. F. Pires, L. Pirmez, W. Wei, H. Song, A. Zomaya, and S. Khan, "System modelling and performance evaluation of a three-tier cloud of things," *Future Gener. Comput. Syst.*, vol. 70, pp. 104–125, May 2017.
[25] S. Madria, V. Kumar, and R. Dalvi, "Sensor cloud: A cloud of virtual sensors," *IEEE Softw.*, vol. 31, no. 2, pp. 70–77, Mar. 2014.
[26] M. K. Buckland, "Information as thing," *J. Amer. Soc. Inf. Sci.*, vol. 42, no. 5, pp. 351–360, Jun. 1991.
[27] X. Lai, T. Yang, Z. Wang, and P. Chen, "IoT implementation of Kalman filter to improve accuracy of air quality monitoring and prediction," *Appl. Sci.*, vol. 9, no. 9, p. 1831, May 2019.
[28] F. Al Machot, K. Kyamakya, B. Dieber, and B. Rinner, "Real time complex event detection for resource-limited multimedia sensor networks," in *Proc. 8th AVSS*, 2011, pp. 468–473.
[29] C. Mouradian, F. Ebrahimnezhad, Y. Jebbar, J. K. Ahluwalia, S. N. Afrasiabi, R. H. Glitho, and A. Moghe, "An IoT platform-as-a-service for NFV-based hybrid cloud/fog systems," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6102–6115, Jul. 2020.
[30] M. P. Alves, F. C. Delicato, I. L. Santos, and P. F. Pires, "LW-CoEdge: A lightweight virtualization model and collaboration process for edge computing," *World Wide Web*, vol. 23, no. 2, pp. 1127–1175, Mar. 2020.
[31] B. Yang, W. K. Chai, G. Pavlou, and K. V. Katsaros, "Seamless support of low latency mobile applications with NFV-enabled mobile edge-cloud," in *Proc. CloudNet*, 2016, pp. 136–141.
[32] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 47–54.
[33] A. Aral, I. Brandic, R. B. Uriarte, R. De Nicola, and V. Scoca, "Addressing application latency requirements through edge scheduling," *J. Grid Comput.*, vol. 17, no. 4, pp. 677–698, Dec. 2019.
[34] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.
[35] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT," in *Proc. AINA*, Mar. 2015, pp. 687–694.
[36] K. Aruna and G. Pradeep, "Performance and scalability improvement using IoT-based edge computing container technologies," *Social Netw. Comput. Sci.*, vol. 1, no. 2, pp. 1–7, Mar. 2020.
[37] H. Schulzrinne, A. Rao, and R. Lanphier, *Real Time Streaming Protocol (RTSP)*, document RFC 2326, 1998.
[38] L. Chen, D. Jiang, H. Song, P. Wang, R. Bao, K. Zhang, and Y. Li, "A lightweight end-side user experience data collection system for quality evaluation of multimedia communications," *IEEE Access*, vol. 6, pp. 15408–15419, 2018.
[39] P. Bellavista, C. Giannelli, T. Lagkas, and P. Sarigiannidis, "Quality management of surveillance multimedia streams via federated SDN controllers in fiwi-iot integrated deployment environments," *IEEE Access*, vol. 6, pp. 21324–21341, 2018.
[40] V. R. Basili, "Software modeling and measurement: The goal/question/metric paradigm," Univ. Maryland, College Park, MD, USA, Tech. Rep. CS-TR-2956, 1992.
[41] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *Proc. NFV-SDN*, Nov. 2016, pp. 148–153.
[42] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw. (Hotnets)*, vol. 3. New York, NY, USA: ACM Press, Jun. 2010, pp. 1–6.
[43] (2019). *FIWARE*. Accessed: Jul. 1, 2020. [Online]. Available: https://www.fiware.org/about-us/

ANSELMO LUIZ ÉDEN BATTISTI is currently pursuing the Ph.D. degree with Fluminense Federal University (UFF), Brazil. His main research interests include edge computing, Internet of Things, and multimedia sensor virtualization.

DÉBORA CHRISTINA MUCHALUAT-SAADE received the Ph.D. degree in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), in 2003. She is currently a Full Professor with the Department of Computer Science, Fluminense Federal University (UFF), Brazil. She is also one of the founders and heads of the MídiaCom Research Laboratory. She has contributed to the design and development of the Ginga-NCL middleware, used in the Brazilian Digital TV Standard and IPTV services. Her main research interests include multimedia systems, computer networks, IoT, smart grids, and e-health.

FLÁVIA C. DELICATO is currently an Associate Professor with Fluminense Federal University, Brazil. She is also a CNPq Level 1D Researcher. She has coordinated and participated in several research and development projects funded by industry and funding agencies. She has written two books and more than 180 published articles. Her main research interests include middleware for edge/fog computing and Internet of Things, sensor virtualization, and software engineering techniques applied to ubiquitous computing. She serves as a member of the Editorial Board of several international journals, such as *Ad hoc Networks*, *ACM Computer Surveys*, IEEE Transactions on Service Computing, and IEEE Open Journal of the Communications Society. Since 2015, she has been a Collaborating Researcher with the Centre for Distributed and High-Performance Computing, University of Sydney, Australia. Since 2017, she has been a member of the IEEE Technical Committee on Smart World.