A Management Architecture for IoT Smart Solutions: Design and Implementation

A Management Architecture for IoT Smart Solutions: Design and Implementation

Dener Silva[1] · Alexandre Heideker[1] · Ivan D. Zyrianoff[2] · João H. Kleinschmidt[1] · Luca Roffia[2] · Juha-Pekka Soininen[3] · Carlos A. Kamienski[1]

Abstract

The management of IoT solutions is a complex task due to their inherent distribution and heterogeneity. IoT management approaches focus on devices and connectivity, thus lacking a comprehensive understanding of the different software, hardware, and communication components that comprise an IoT-based solution. This paper proposes a novel four-layer IoT Management Architecture (IoTManA) that encompasses various aspects of a distributed infrastructure for managing, controlling, and monitoring software, hardware, and communication components, as well as dataflows and data quality. Our architecture provides a cross-layer graph-based view of the end-to-end path between devices and the cloud. IoTManA has been implemented in a set of software components named IoT management system (IoTManS) and tested in two scenarios—Smart Agriculture and Smart Cities—showing that it can significantly contribute to harnessing the complexity of managing IoT solutions. The cross-layer graph-based modeling of IoTManA facilitates the implemented management system (IoTManS) to detect and identify root causes of typically distributed failures occurring in IoT solutions. We conducted a performance analysis of IoTManS focusing on two aspects—failure detection time and scalability—to demonstrate application scenarios and capabilities. The results show that IoTManS can detect and identify the root cause of failures in 806ms to 90,036ms depending on its operation mode, adapting to different IoT needs. Also, the IoTManS scalability is directly proportional to the scalability of the underlying IoT Platform, managing up to 5,000 components simultaneously.

Keywords Internet of things · IoT management · Management architecture · IoT architecture

1 Introduction

The Internet of Things (IoT) has an increasing development, mainly because of the continuous efforts of a wide community of scientists, developers, vendors, and, particularly, end-users. Due to the significant number of solutions and needs, different players identified that either the available commercial IoT-enabled products are not suitable to their demands, or there is no product for a specific requirement, lacking a generic solution that can be reused in different scenarios [1].

IoT solutions generate vast amounts of data, creating new demands for computing systems such as distributed storage, processing capacity, and communication. Cloud computing facilitates the deployment of IoT solutions by providing flexibility in resource allocation and centralized management [2]. However, IoT solutions have specific requirements and constraints, such as low latency for real-time applications, which the Cloud cannot solve alone. A potential solution is moving the intelligence of devices to the edge of the network, allowing storage, processing, and low latency capabilities closer to the IoT devices. This concept is known as Edge computing and is defined as any computing and networking resource between the IoT devices and the cloud [3].

Fog and mist computing are introduced as edge computing approaches, acting as crucial components to IoT solutions. Fog computing brings the concept of Cloud computing solutions closer to things, providing computing and communications resources. Mist is the hardware located closest to the devices, is the extreme edge of a network. The introduction of edge computing to an already complex IoT solution brings new challenges for managing an inherently distributed and heterogeneous solution. However, most IoT management approaches in the literature explore two aspects of the field: devices (sensors/actuators) and wireless communications (LoRa, Wi-Fi, 4G) [4]. The management of real-world IoT solutions faces unexpected issues since the end-to-end communication between the things and the cloud passes through potentially many intermediate software, hardware, and communication components [5].

Practical IoT operational problems arise when the end-to-end dataflow stops being updated at the end-user monitoring application. When this happens, there is no comprehensive management solution to pinpoint the exact failure location and cause, which may be in any software, hardware, and communication component along the path between the sensors and the dashboard. The hands-on experience gained by managing an IoT-based Smart Irrigation platform for more than three years [5] provided us with precious understanding of the fundamental problems suffered by IoT systems and impelled us to provide innovative management solutions.

This paper presents a novel four-layer IoT management architecture (IoTManA) that can be applied to different smart applications, such as smart cities, smart agriculture, smart health care, and smart industry. The generic characteristic of IoTManA makes it able to operate in different environments regardless of the infrastructure or platform adopted by the IoT solution. Using a layer-based approach, IoT components are represented as Virtual Entities that can support different needs of an IoT environment, such as scalability, context awareness, and availability.

The IoT management system (IoTManS) implements IoTManA for smart water management, developed to demonstrate how the IoTManA can be applied to manage IoT solutions in distributed locations. IoTManS ensures the availability of the data collected by the sensors, the proper operation of the infrastructure—hardware and software components, communication between components, and analyzing data-flows and data quality by managing, monitoring, and evaluating each component in the end-to-end path. Here, we report a running version of IoTManS that manages a Smart Irrigation IoT solution [5] showing how it obtains context information to monitor and manage applications, ensuring the control needs for a decentralized solution. Also, we show results of testbed experiments with smart irrigation and smart city scenarios to demonstrate the practical use of our management architecture and system and evaluate performance and scalability.

This paper brings three main contributions to the IoT management:

- A management architecture that allows mapping how the failure of components in one layer impacts components from upper layers.
- A graph-based view of the end-to-end path between devices and the cloud, considering all the components in between, such as software, hardware, and communication components.
- An architecture and modeling approach for facilitating an integrated monitoring and failure detection system to automatically detect and inform root causes.

We highlight that the purpose of our architecture is the management of IoT solutions, which makes it different from existing IoT Architectures. Whereas there are different architectures for facilitating the understanding and development of IoT solutions [6–9], no similar IoT management architectures have been found in the literature. Existing IoT management approaches and architectures typically focus solely on device management [10].

This paper is organized as follow: Sect. 2 presents the background and related work. Section 3 details the proposed IoTManA and IoTManS. Section 4 presents the design, modeling and implementation of IoTManS and IoTManA based on an IoT smart agriculture solution. Section 5 presents a practical implementation of our architecture and system with examples and performance evaluations experiments. Section 6 presents and discusses the results of the performance evaluation experiments. Finally, Sect. 7 provides a discussion and a summary of future works.

2 Background and Related Work

IoT Management is a complex task due to the distributed nature of such applications that can cover a large geographical area with various software, hardware, and communication components. Therefore, a successful IoT Solution must provide management features that consider high levels of heterogeneity, interoperability, and scalability, thus needing a unifying architecture and middleware [11]. In this section, we present state-of-the-art IoT architecture and management efforts.

2.1 IoT Architectures

The IoT concept is based on devices transmitting data through a communication channel, and since there is no current leading solution, most standardization efforts aim at meeting this concept. Yearbook et al. [12] analyzed several proposals that try to meet IoT architecture needs, such as scalability and security. Numerous efforts on designing new IoT architectures can be found in the literature focusing on software components [6–9]. Washizaki et al [13] surveyed different architectures and design patterns for IoT systems. With the same direction, the International Organization for Standardization (ISO) proposes in ISO/IEC 30141 [14] a reference model and a reference architecture to generic IoT solutions that meet the characteristics necessary in a multi-layer approach based on reusable designs and industry best practices.

A multi-layer architecture facilitates understanding roles, locations, and abstraction levels of different networking, hardware, and software components and is nowadays adopted by most IoT Platforms. An IoT platform—also known as IoT software platform [1] or IoT middleware [15, 16]—implements an IoT architecture providing a variety of building blocks to facilitate the development of an IoT solution, such as device connectivity, device management, data management, data analytics, security, and visualization [5]. IoT platforms collect data from devices and enable the development of IoT solutions that control, monitor, and manage these devices. It is often composed of several middleware components, each of them focusing on a specific feature in a particular layer to provide an end-to-end platform involving data generation, transmission, storage, and processing.

The growing interest of different sectors of society in IoT technologies and the lack of standards opened up the opportunity for a breed of IoT platforms, amounting to 650 IoT companies on the open market in 2019 [17]. A drawback of this rapid evolution is the proliferation of heterogeneous sensor networks that lack interoperability [18]. One approach to this problem is the virtualization of fundamental components into programmable objects and communication between them regardless of their communication methods [19, 20]. The virtual representation of the object can then be treated as an entity within the managed environment, making the system itself also become an entity. The concept of virtualization of real components is explored in different areas. Girau et al. expand the concept of virtualization to the user, representing the user as a virtual entity that executes its tasks through functions [21].

Ray surveyed different architectures applied to specific domains such as device management, system management, heterogeneity management, data management, monitoring, and visualization [22]. The proposals target specific problems or scenarios and platforms, strengthened by recent IoT solutions that address specific domains. Frequent solutions manage devices [23, 24], applications [25, 26], security [27, 28], and many others specific aspects, such as interoperability [29].

Zyrianoff et al. [30] proposed a 5-layer IoT Architecture (IoTecture), that provides support for highly distributed data management functions and separates physical and data-driven models from application services. IoTecture objective is to help the deployment of IoT smart applications over different distributed locations in the mist, fog, and cloud implementing the concept of IoT computing continuum (IoTinuum)

proposed on the same work. IoTinuum considers the mapping of software components into physical locations, that provides a clear view of the different deployment locations for architectural components, divided up into five stages, namely Thing (sensor/actuator), Mist, Fog, Cloud, and Terminal (the place where the end-user interacts with a smart application). The mist can be considered the lowest hierarchical edge computing system, the closest to the devices. Mist components are usually applied to create communication facilities such as LoRaWAN gateways and Wi-Fi access points, providing storage and processing capabilities [30]. The five stages of IoTinuum define the end-to-end information path, starting with data collected by sensors up to commands executed by actuators. These five stages might not be necessarily present in all configuration scenarios. Instead, depending on application characteristics, requirements and constraints, Mist, Fog, or Cloud stages may not be present.

IoTManA is an extension of IoTecture. While IoTecture provides a reference architecture to IoT Platforms and guidelines to deploy a Smart Application focusing on data management, IoTManA is an architecture to manage IoT Platforms and Smart Applications that follow the IoTecture. In other words, IoTecture and IoTManA are both architectures, but the former aims at system development and deployment, whereas the latter aims at system management. IoTManA focuses on the data and managing components at different layers such as devices, networks, services, and applications executed in the different IoTinuum stages.

Pena et al. proposed a similar approach of an architectural reference model focusing on management, considering the different aspects of IoT Systems [10]. However, the proposal does not use a multi-layer approach, limiting the architecture to be used only by their own IoT Platform.

2.2 IoT Management

Besides inheriting management requirements from cloud and edge computing, IoT environments have specific challenges such as device, network, and application management that must yet consider problems of scalability and heterogeneity.

A typical research domain explored by IoT solutions is resource management in varied and distributed locations like the cloud, fog, or mist. Considering the edge components in their approaches, Mostafa et al. proposed an algorithm for the management and effective resource selection in fog computing [31]. Martinez et al. surveyed the design, resource, and management of fog computing systems focusing on service and resource allocation [3]. The same efforts are found in [32–35], and other studies extend the resource management in approaches that consider the mist as a component on the edge [36, 37] into the deployment. Bouakouk et al. [38] surveyed different paradigms towards Cloud-IoT, analyzing architectures and taxonomy focusing on the challenges in edge and cloud computing.

Elliot et al. [39] proposed an orchestrator to manage Docker containers between heterogeneous cloud service providers, considering the challenge of deploying different IoT Platform components . Jain et al. [40] propose a similar approach to design the whole distributed IoT application in one place, considering different application

components to be deployed at different edge nodes. Xu et al. [41] addressed the same concept applying a rule engine in a distributed hybrid cloud management platform considering the resource allocation between different cloud providers, treating the heterogeneity in the cloud level . Avasalcai et al. proposed a decentralized resource management technique and a technical framework to deploy latency-sensitive IoT solutions on edge devices [42]. Sinche et at. [43] surveyed IoT Management technologies, frameworks and protocols, focusing on devices and proposing a taxonomy for IoT devices management.

To the best of our knowledge, in the current literature, there is no architecture capable of providing manageability to an IoT solution in different layers simultaneously—software, hardware, connections, sensors—and considering a multi-location distributed infrastructure—cloud, fog, and mist. Most of the efforts in new architectures are directed to IoT Applications.

Compared with existing IoT management approaches, our proposal is technology- and platform-agnostic and acknowledges critical aspects needed by any IoT solution, such as deploying cloud and edge components, infrastructure abstraction, availability, and scalability. Our approach deals with data to implement solutions to different management problems. Here, we demonstrate how the architecture can be applied in a management system that allows failure detection while identifying the root cause, i.e., the process of identifying and isolating the main component responsible for the failure.

3 IoT Management Architecture (IoTManA)

The IoT Management Architecture (IoTManA) aims at managing different aspects of infrastructure (machines, devices, and communications), software components (brokers, databases, services), dataflow, and data quality, providing a manageable end-to-end environment. It can be applied to different areas due to the representation of components as virtual entities, being easy to model any IoT smart solution with heterogeneous devices and components. Also, it has been developed and extensively tested in the context of a smart irrigation solution [5].

IoTManA follows the IoT Computing Continuum (shortened to IoTinuum) [30] and uses a deductive to empirical approach, where the data was collected and modeled repeatedly during the development of the smart irrigation platform until the proposal reaches a sufficiently satisfying level of conciseness [44].

Figure 1 depicts the IoTManA 4-layered architecture:

– Layer 1 (Infrastructure) manages devices (sensors and actuators), physical and logical hardware (i.e., bare metal and virtual machines or containers), and network infrastructure. It includes the management of sensor, mist, fog, and cloud hardware components. The hardware components may be a simple Raspberry Pi, a computer, or a dedicated server. This layer also manages the links between the computing components, including LoRa or other wireless technology links between device and mist, a WLAN link (e.g., Wi-Fi) between mist and fog, and the Internet connection between fog and cloud.

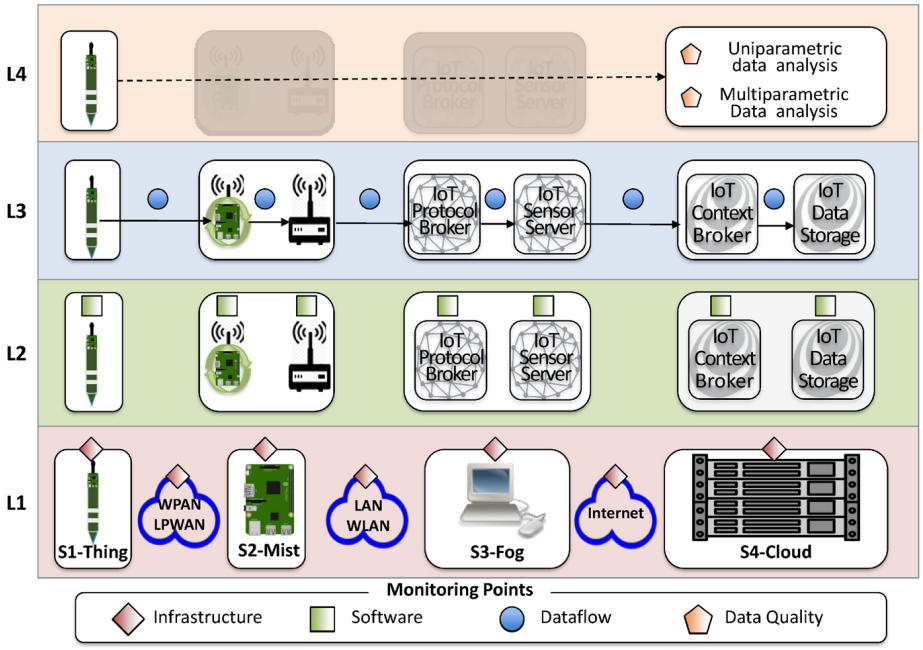

Fig. 1 IoT Management Architecture (IoTManA)

– Layer 2 (Software Components) manages software components (e.g., services or microservices) of the IoT Platform and Applications.
– Layer 3 (Dataflow) manages the availability of dataflows across hardware, software, and network elements. The steps of an end-to-end dataflow may be monitored indirectly rather than directly. For example, if data is received from a sensor in the cloud, it can be safely assumed that dataflow is actively working.
– Layer 4 (Data Quality) analyzes the data quality from two perspectives of veracity. A univariate data analysis performs simple checks of data ranges (e.g., maximum and minimum), periodicity (e.g., data received every x minutes), and the anomaly of one variable at a time (e.g., a substantial unexpected variation of a metric). A multivariate data analysis performs more complex analyses of multiple variables (e.g., comparing rainfall with soil moisture).

To provide a manageable end-to-end environment, the data availability (Layer 3) and data quality (Layer 4) are critical components since we consider the endpoint of the flow as the availability of the data for consumption. These Layers differ since Layer 3 monitors the hop-by-hop dataflow, whereas Layer 4 monitors the end-to-end data quality.

IoTManA represents a conceptual view of layers that increase abstraction, scope, and softwarization level from the bottom up, allowing one layer to be mapped into the layer below to link effects and facilitate root cause detection. The coherence between the four hierarchical layers is fundamental for guaranteeing that any adverse event (e.g., a failure) in a layer that affects components located in upper

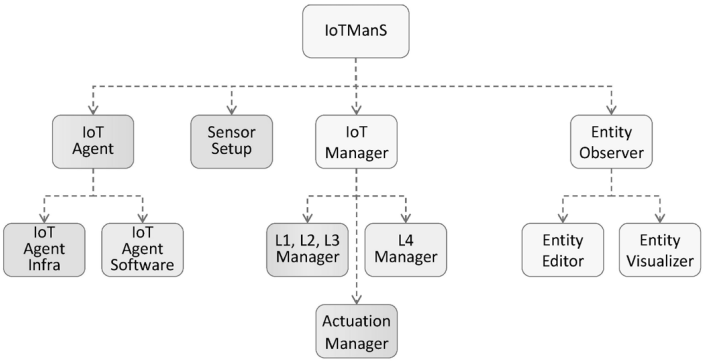

Fig. 2 IoT Management System (IoTManS)

layers will be immediately propagated to the management services in such a way for the cause-effect relationship to be understood so that appropriate corrective actions can be taken. Even though Layer 4 (Data Quality) is close to the analytical or data-intensive models of particular IoT Solutions, it must also be connected to Layer 3 (Dataflow) so that the problems in the latter can be mapped to the former. In other words, even though platforms may differ in terms of where and how components are placed, these layers must be interconnected. IoTManA can be implemented by different management platforms, such as our IoTManS presented in this paper.

4 IoT Management System (IoTManS)

This section presents the process of design, modeling, and implementation of the IoT Management System (IoTManS) using a use case based on a Smart Agriculture, detailing each software component of the IoTManS and its functions. The same process is applied to a simple Smart City solution with a different IoT Platform in order to show how IoTManS is not technology-dependent and can manage diverse solutions.

4.1 IoTManS Design

IoTManS implements IoTManA, as depicted by Fig. 2, where the color of the boxes relate to the colors of IoTManA layers in Fig. 1. We adopt the same color pattern throughout this paper. Components represented by white boxes implement more than one layer. IoTManS is composed of a set of software components that implement the abstract functions within IoTManA. The key software components comprising IoTManS are:

– IoT Agent: monitors and controls the mist, fog, and cloud machinery (physical/logical) and the software components running on them. Monitoring is a data collection of hardware/software components functioning, such as fault or perfor-

mance, which is sent to the IoT Manager. There are two types of IoT Agents: (1) IoTAgent Infra—for obtaining information from the infrastructure and system metrics; and IoTAgent Software—for obtaining data from software components such as availability and status. Control means any actuation on hardware, software, or communication components, such as turning on/off equipment or software components and changing configurations. IoT agents can monitor Docker containers, processes, connections between hosts, CPU and memory usage, and application availability. Availability is monitored by different approaches to validate the operating state of an application, such as sending a request to a health check endpoint in the software component (in case of a REST API available) or consuming the service (e.g., subscribing in a topic to the MQTT broker and then publish a message, receiving the same information in the form of a subscription).

- IoT Manager: is the core of IoTManS, responsible for monitoring, analyses, and actuation actions. The IoT Manager receives monitoring data from the IoT Agents and analyzes the end-to-end dataflows, handling the data and acting if necessary. It may also actuate remotely on IoT Agents, e.g., sending commands for changing the state of monitored components. The IoT Manager comprises two sub-components: (a) L1, L2, L3 Manager is responsible for the management of the three first layers on the IoTManA Layer; (b) L4 Manager that operates as a data quality analyzer; (c) Action Manager that allows changing the state of software, hardware, and communication components automatically or manually.
- IoT entity observer: composed of an IoT entity editor (IoTEE) and an IoT entity visualizer. The IoTEE allows CRUD (creating, reading, updating, and deleting) functions over virtual entities, which correspond to managed physical entities. The IoT entity visualizer is a custom dashboard that allows end-user and operators to check the status and the historical values (time series) of the managed entities.
- IoT sensor setup (IoTSS) is used in field installations (and updates) of physical sensors and their association with virtual entities. IoTSS works in places where 4G/Wi-Fi connectivity is not guaranteed due to limited coverage or shadow areas. The IoTSS binds the physical device and its digital representation in the system as a virtual entity, making it easy to install and manage new devices on a virtual level.

4.2 IoTManS Implementation and Operation Modes

IoTManS considers each virtual entity as a JSON representation of an entity to be managed, such as software, hardware, a communication component, a device, or a system. Virtual entities allow the IoT Manager to integrate and manage different aspects of the platform. For instance, sensors can be configured by the IoTSS using the virtual entity that contains attributes such as publish interval, operation method, format, and address that indicates where to publish the data.

The software components that compose IoTManS were initially deployed onto the FIWARE IoT Platform, using Orion Context Broker as the core component for receiving the management messages from the IoT Agents to the IoT Manager in a

Smart Irrigation solution, which was in turn replicated in a lab testbed to evaluate its performance, scalability, and effectiveness. Afterward, it was ported to store management messages in the ThingsBoard IoT platform, in an attempt to demonstrate that: (1) Both IoTManA and its implementation IoTManS are not dependent on particular platforms; (2) The performance of IoTManS depend exclusively on the underlying platform and not on the management system.

In the FIWARE implementation, IoT Agents update the status of each component to FIWARE Orion, which notifies the IoT Manager of each new data arrival. A similar approach is used in the ThingsBoard implementation, where IoT Agents update the status of each component as a device. Unlike FIWARE Orion that only receives the data and forwards it, ThingsBoard treats the data from the receiving to the time-series data storage, being necessary to use a plugin, also provided by the developer, to recover the data from the platform and forwards it to the IoT Manager. In other words, the IoT Manager operates as a data consumer of the IoT Platform. Therefore, the scalability of the IoT Manager is directly proportional to the scalability of the underlying IoT Platform it is implemented on.

The IoT Manager has two operation modes:

- Active mode: The IoT Manager periodically polls the graph-based structure, implemented by IoTManS to determine the state of each component, detecting and creating an event accordingly, updating the graph, and taking the corresponding actions. The active mode allows the IoT Manager to handle components that cannot be directed monitored (e.g., if the component has no interface or monitoring endpoint). For instance, when IoT Agents cannot collect metrics directly from a specific component, the IoT Manager can use the graph to analyze the state of each known device that composes the end-to-end path between sensors and the cloud. In solutions that apply time-driven sensors, IoT Manager can establish a threshold of data samples in order to determine the state of a component or Agent. For example, if the sampling time is 60 seconds and the threshold is three samples, after 180 seconds since the last message, the IoT Manager assumes that an IoT Agent is down.
- Passive mode: The IoT Manager does not actively check the graph and waits until a message from IoT Agents arrives with management data about the monitored components. In this method, the IoT Agent continuously monitors the components and notifies the IoT Manager when their state changes. This method can be applied in solutions that apply event-driven sensors, where the monitoring by sampling time is not relevant.

The active and passive modes can be combined for the real-time monitoring of certain components, allowing the IoT Manager to act and respond quickly upon changes at the monitored components, with no need to wait until the next graph polling. The combination of the passive and active modes allows the IoT Manager to operate in highly heterogeneous IoT Solutions, as it is able to determine the state of components, devices, connections, software, and hardware that are compatible or not with the IoT Agents in a stable or unstable environment, and demanding real-time monitoring or not.

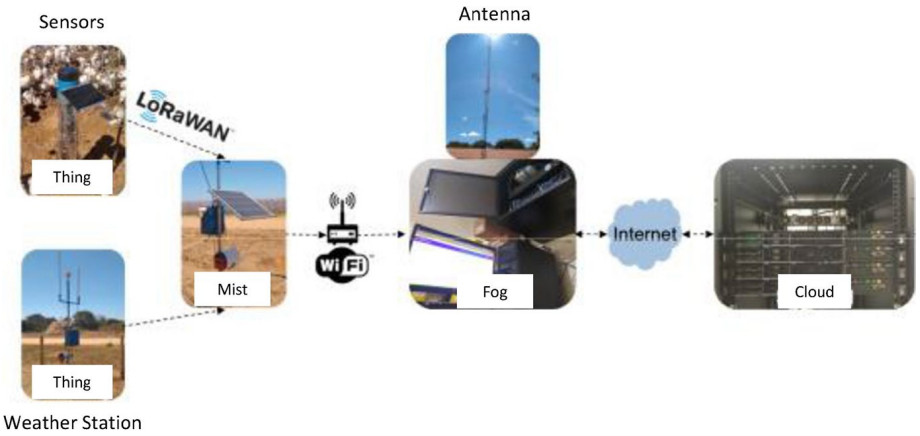

Fig. 3 Use Case on a Smart Irrigation Solution

Virtual entities are organized as a directed acyclic graph (DAG) where vertices are the monitored components and edges represent a cause-effect relationship between components. The use of a DAG allows IoT Manager to organize and represent the architecture and allow an easy understanding of its components and their relationships. The graph approach also gives the IoT Manager the structure necessary to detect and isolate failures.

4.3 Use Case

This section demonstrates the IoTManS operation in an IoT Smart Irrigation scenario. Figure 3 illustrates the deployment of a Smart Irrigation Solution based on the FIWARE IoT Platform[1] [5], applying the IoTinuum stages: Thing (soil moisture sensors and weather station), Mist (LoRaWAN gateway, Weather Station Collector), Fog (ChirpStack [45], MQTT), and Cloud (FIWARE components).

In this solution, the sensors transmit data to the LoRaPackageForwarder, a software component that receives and forwards the messages to the LoRaGateway-Bridge, which converts Packet Forwarder protocols into a ChirpStack data format. The Mist and the Fog implement a ChirpStack LoRaWAN server, an open-source software composed by Chirpstack Network and Application Servers, and Redis and PostgreSQL databases. ChirpStack is in charge of the communication and management of LoRaWAN devices, using the Mosquitto MQTT broker to exchange information between modules. The FIWARE IoT Agent (not the management IoT Agent, but the protocol translator) consumes data received by Mosquitto from ChirpStack and transforms it into the JSON NGSI [46] format used by FIWARE Orion and FIWARE Quantum Leap. Orion is a publish/subscribe broker responsible for creating, registering, updating, notifying, and subscribing to the FIWARE IoT Platform.

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

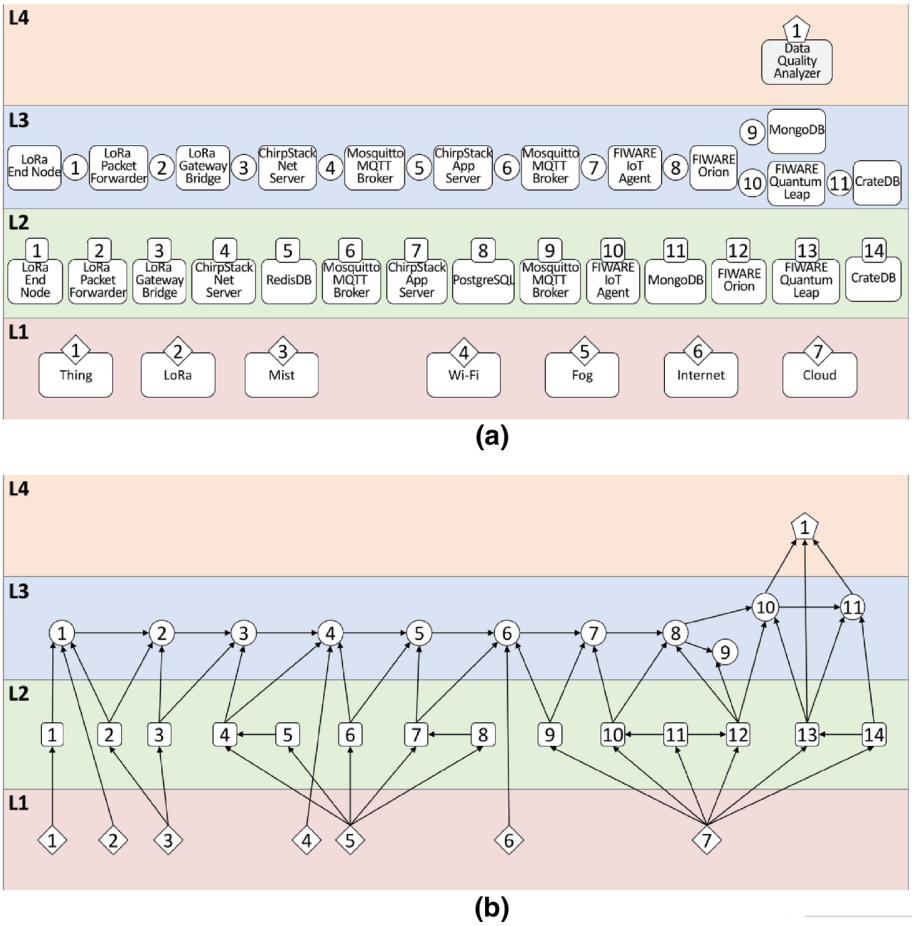

Fig. 4 **a** Modeling a smart irrigation solution—**b** IoTManS components and DAG representation

Orion is the core of FIWARE, storing only the last state of virtual entities in a MongoDB[2] database. To preserve historical data, Orion must work with other software components that store data, i.e., databases, such as FIWARE Quantum Leap. The latter subscribes to an entity and is notified by FIWARE Orion each time this entity is updated, converting the data from JSON NGSI to the native CrateDB[3] format.

4.4 Modeling

Figure 4a illustrates the representation of each component of the managed solution as a virtual entity. On Layers 1, 2, and 4, the numbering is on top, representing the

[2] https://www.mongodb.com.
[3] https://crate.io.

component itself. On Layer 3, the numbering is between components, since it represents the data flowing between them.

The components on Layer 1 represent the hardware elements and the state of the communication links between them: (a) Thing is a generic representation of any connected device; (b) LoRa represents the connection between Thing and the Mist; (c) Wi-Fi is the state of the local wireless connection between the Mist and Fog; (d) Internet is the component that represents the connectivity status between Fog and Cloud; (e) Cloud represents the hardware used to host the Application executed in a Cloud service provider.

On Layer 2, the numbers represent each software component necessary for the IoT environment to become operational, creating the end-to-end communication as described in Sect. 4.3.

On Layer 3, the numbers represent abstractions of the connectivity between pairwise components on Layer 2, creating an end-to-end dataflow. The network entities (such as Wi-Fi and Internet) represent the connectivity between two entities, storing data about latency and availability. In the Smart Irrigation scenario, a LoRa EndNode transmitting one packet every minute in a controlled environment checks the LoRa connection availability. Suppose the LoRa EndNode is transmitting data successfully. In that case, we can assume that the LoRa Radio connection is operational, so a failure in retrieving the data is located on the LoRa device or the Fog and its components.

The IoT Manager can check if the LoRa Connection is up by comparing the interval between the current and last transmission time from the LoRa EndNode entity. Entities on Layer 3 can be monitored using sniffers or other techniques for monitoring the connection between two components. For example, Orion and Quantum Leap have counters that store the number of messages sent and received, which allow us to verify the correct operation of the end-to-end dataflow. Whenever the measurement of a specific connection is not straightforward, the state of Layer 3 Components can be determined by the state of the components of each end of the connection.

On Layer 4, the Data Quality checks the arrival of each data to verify its integrity and usability.

Figure 4b follows the same numbering scheme to represent entities organized as a DAG, handling each layer and how each component affects other components on upper layers. Each entity, represented as a node, is self-contained with its attributes, such as hardware resources, process, and availability. Also, a cross-layer design allows relationships between entities from different layers in a bottom-up approach. For example, the failure of the mist node (e.g., a Raspberry Pi playing the role of a LoRaWAN gateway) causes the LoRa Packet Forwarder and the LoRa Gateway Bridge services to go down, which in turn interrupts the communication from the sensors with the following components.

In our approach, upper layers depend on lower layers to be operational. Layer 4 analyses the database and messages received to determine the data quality, depending on the software and database components. Layer 3 represents the dataflow and depends on each software component (Layer 2) and communication infrastructure (Layer 1) to be available, and any failure will affect the whole dataflow since it is

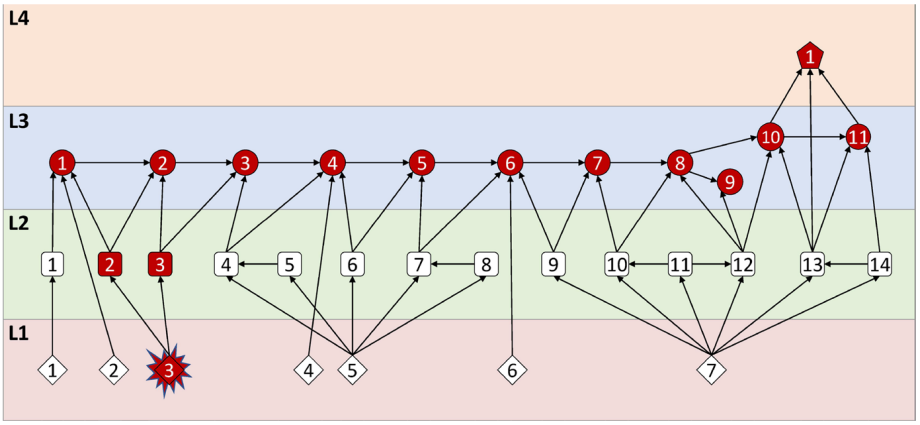

Fig. 5 Component Failure Representation

continuous. Layer 2 components depend on entities that represent the infrastructure where they execute.

Layer 2 can also represent software dependencies, indicated by the edge connecting components 5 to 4 at L2 in Fig. 4b. Component 4 (ChirpStack Net Server) depends on component 5 that represents its database (RedisDB). If the database (5) fails, the ChirpStack Net Server (4) will not be operational, and the failure that started at component 5 propagates to component 4 that also fails.

In other words, our graph modeling approach represents dependencies between components within the same IoTManA layer or from layer N to layer N+1. By making explicit the dependencies of each entity, the IoT Manager can analyze the relationship between components and correlate their status to determine the health of the platform. This approach also allows the IoT Manager to determine the impact of an individual software component in the platform, facilitating failure isolation, detection, impact analysis, and prevention.

Figure 5 represents the DAG illustrated on Fig. 4b in a failure scenario. Component 3 on Layer 1 (Mist) presents a failed hardware status. Since it is a Layer 1 failure, the IoT Manager can determine its impact on the whole platform even before other components detect the failure. The Layer 1 components are not directly impacted since there are no co-dependencies. However, the Layer 2 software components (2 and 3) executed on the Mist will fail since they depend on the infrastructure. This behavior will expand the failure to components 1, 2, and 3 on Layer 3 and then the whole dataflow. In this scenario, the IoT Manager can create the reverse path to locate and isolate the failure and then pinpoint component 3 on Layer 3 as the root cause of the problem.

The failure of a component that affects the operational status of other components executing on upper layers—as the fog hardware causing the failure of all software components running on it—is known in the literature as a common mode failure. Current approaches do not consider edge components or IoT environments. For example, Cerveira et al. deal with common mode failures on cloud environments [47] and Mauro et al. use containers to migrate services in case of failure [48]. The

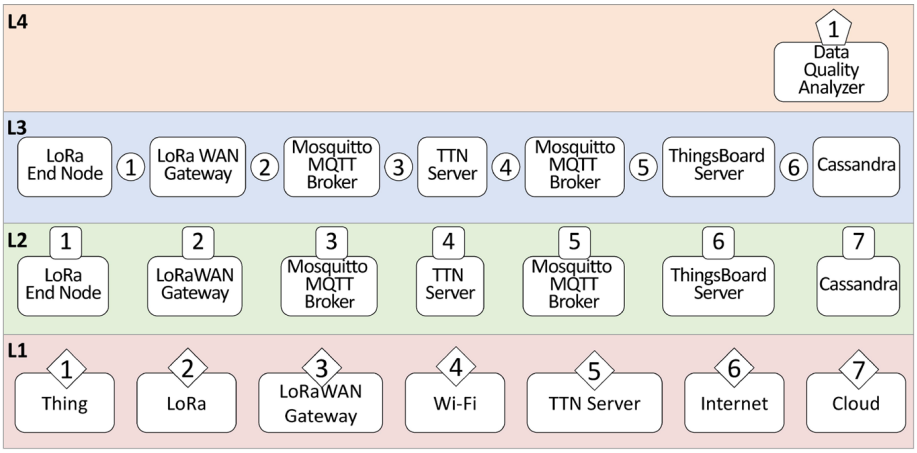

Fig. 6 Modeling an LoRa-TTN-Thingsboard based solution- entity representation

IoTManS graph approach considers the end-to-end impact described on Layer 3 dataflow, not only on the cloud but also on all IoTinuum stages.

The fourth Layer – Data Quality, is currently performed by the IoT Manager using univariate data analysis, periodically checking data ranges and anomalies on the sensor entity. Togneri et al. present a preliminary study using multivariate data, considering the end-to-end dataflow methods to identify relevant situations, such as sensor failures and the mismatch of contextual sensor information due to different spatial granularities during the data collection process [49].

IoTManA is generic enough to be adapted to different solutions, not being technology-dependent on the managed IoT solution. Porting it to other IoT solutions requires knowledge of its components, their dependencies, and the dataflows. Figure 6 illustrates the use of IoTManA in a generic solution that implements a LoRa Device sending data to a local LoRa Gateway connected to the TTN[4] (The Things Network) LoRaWAN server that in turn forwards the data to a ThingsBoard IoT Platform[5] hosted on an AWS Cloud Service Provider[6], a simple IoT scenario where a device is transmitting data to an IoT Platform on the cloud. The Things Network is a collaborative IoT ecosystem that creates networks, devices, and solutions using LoRaWAN. It provides tools for developers to publish and consume data from various sources and apply them to different IoT Solutions. Therefore, Fig. 6 does not represent any IoT solution in specific. However, this scenario aims at demonstrating that the modeling can be applied to different IoT Solutions, as long as there is prior knowledge of how the platform operates, its components, and the dependence between them. Using this knowledge, Fig. 7 represents how the components numbered on Fig. 6 can be organized in order to describe their dependencies and

[4] https://www.thethingsnetwork.org/.
[5] https://thingsboard.io/.
[6] https://aws.amazon.com/.

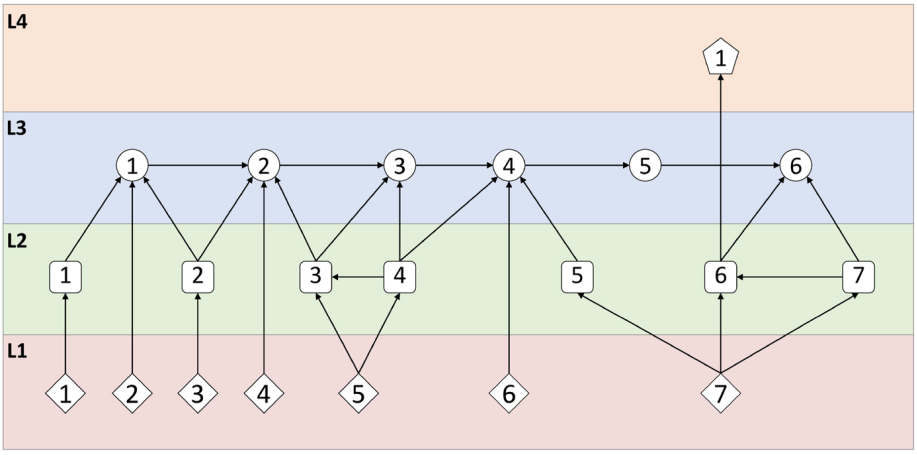

Fig. 7 Modeling an LoRa-TTN-Thingsboard based solution- DAG representation

dataflow. Each component is dependent on the related components on layers below, as software is dependent on hardware where its executed. On L2, component "MQTT Broker" is dependent on the availability of the "ThingsNetworkServer" and Component 6 (ThingsBoard Server) is dependent on its database on component 7 (Cassandra).

5 IoTManS Performance: Methodology

In order to evaluate the performance and scalability of the IoT Manager, we envisioned two sets of experiments. The first set of experiments—failure detection—aims at determining the IoT Manager performance regarding the failure detection time using the active and passive modes, also analyzing the time needed to isolate the failed component and determine the components affected by the failure in single and multiple failures scenarios, totaling four scenarios studied. The second set of experiments—Scalability—aims to evaluate the scalability of the IoT Manager in a Smart City scenario with a varying number of monitored components in two different IoT Platforms.

5.1 Methodology

5.1.1 Failure Detection

Figure 8 illustrates the testbed used in the first set of experiments, failure detection on smart irrigation scenario, and shows the placement of each software component. The red arrows indicate the IoTMA Software and IoTMA Infra collecting the data from software and hardware, respectively, and publishing it to a FIWARE Orion that notify the IoT Manager of each data arrival. This scenario was deployed in a testbed implementation using four virtual machines representing IoTinuum stages (Thing,

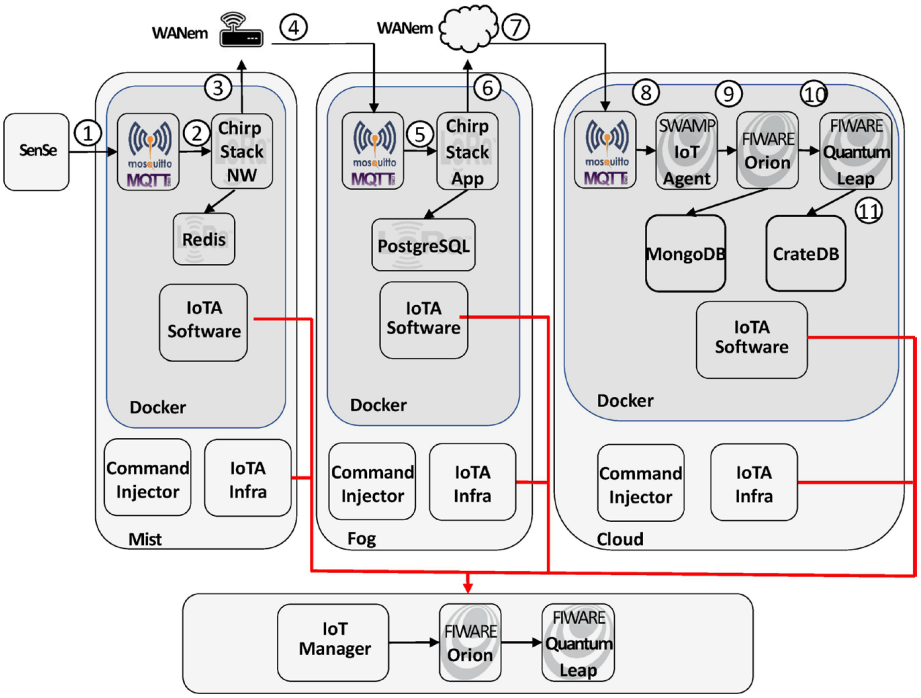

Fig. 8 IoT agent and IoT manager deployment

Mist, Fog, and Cloud), running a complete smart irrigation solution [5], but in a controlled testbed environment.

SenSE (sensor simulation environment) [50] is a simulator capable of generating synthetic traffic on a large scale, simulating thousands of devices transmitting LoRaWAN messages to the Mist, simulating an IoT Platform in operation. All software components are deployed as Docker containers, namely Mosquitto, ChirpStack Network Server, Redis, PostgreSQL, ChirpStack Application Server, IoT Agent, FIWARE Orion, MongoDB, FIWARE Quantum Leap, and CrateDB. The testbed executes an IoT Platform in execution with simulated data from SenSE, and its dataflow is numbered on Fig. 8 originating on SenSE through the Mist and Fog until the storage on the Cloud.

The command injector is a software component used in the experiments to send a command to a specific software component, starting or stopping the Docker container and storing the timestamp when the command was sent. Thus, the IoT Manager runs in an individual virtual machine, not being affected by changes in the monitored environment.

The connections between the Mist, Fog, and Cloud stages are emulated using a WANem [51] (Wide Area Network emulator) installation to simulate latency in the Wi-Fi and Internet connections with delays of 45ms each. Only the latency parameter is considered. In addition to the platform components, an instance of the IoT Agent is executed in each virtual machine, monitoring the four stages of the IoT Computing Continuum: Thing, Mist, Fog, and Cloud. The IoT Agents monitor the

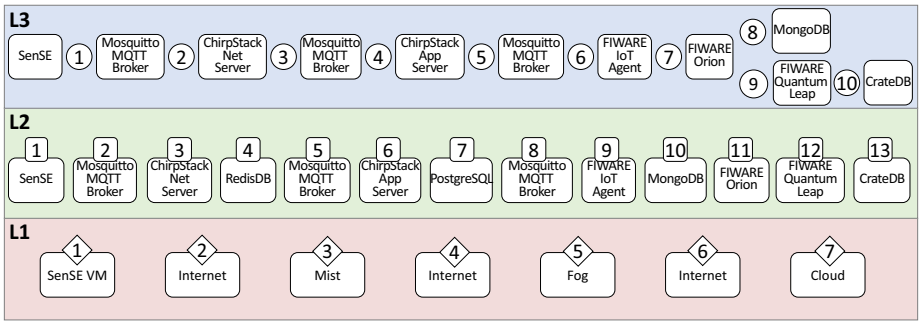

Fig. 9 Failure detection testbed—components representation

entities and send their status to the IoT Manager every 60 seconds. In this scenario, the manager operates the active method every 60 seconds. Since we are dealing with multiple virtual machines, we synchronized the machine clocks using an NTP (Network Time Protocol)[7], thus avoiding affecting the results. The components are represented in Fig. 9, and the DAG used by the IoT Manager to manage the components is illustrated in Fig. 10. Layer 4 is not considered in these scenarios.

Based on real data collected by the IoT Agents during five months of operation from the SWAMP Platform [5] deployed with FIWARE, we use a Weibull distribution to determine the lifetime of each component. Using this knowledge, we create a timeline with the lifetime of each component, determining the time of failure and its order. When the time is reached, the command injector sends a command to stop the Docker container of the correspondent software component storing the timestamp. Using the presented structure and components, the first set of experiments evaluates the behavior of the IoT Manager in four scenarios considering the main factor the detection time:

- Failure detection on active mode: when the IoT Manager detects a failure, first it creates an alarm, checking, in turn, the DAG to determine the root cause and evaluate which components will be affected by the failure. The operation time is determined by comparing the timestamp generated by the IoT Manager on the failure detection alarm and the timestamp generated by the command injector.
- Failure detection on passive mode: the IoT Agents check the monitored component every second, notifying the IoT Manager only on state changes.
- Operation recovery active mode: the command injector sends a command to start a component, whose timestamp is stored and compared to the timestamp of the IoT Manager when it detects for the first time after the fail that the state of the component was changed to normal.
- Operation recovery passive mode: the IoT Agents check the monitored component every second, notifying the IoT Manager only on state changes

[7] http://www.ntp.org/.

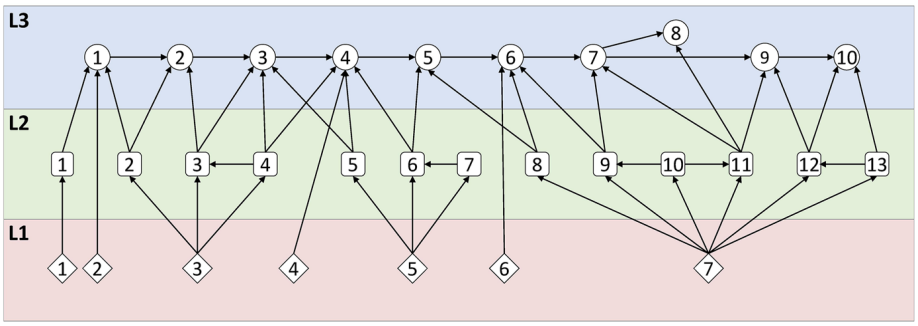

Fig. 10 Failure detection testbed—DAG representation

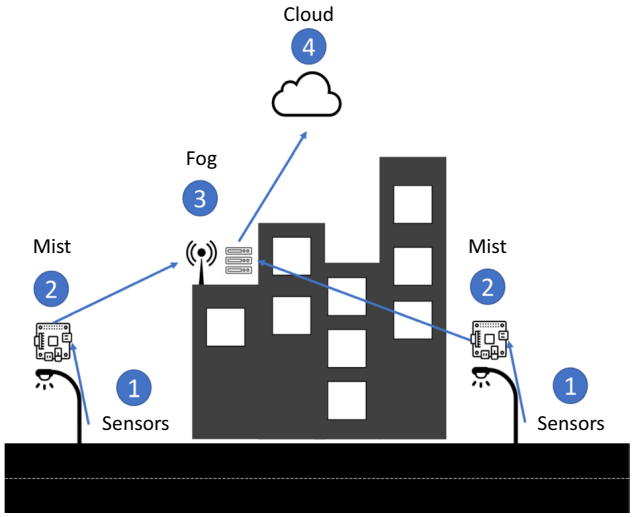

Fig. 11 IoT smart city solution

5.1.2 Scalability

IoTManS, like any IoT management system, must be scalable to keep up with the massive number of sensors and software, hardware, and communication components involved in any effective IoT solution. Figure 11 illustrates a Smart City solution that uses sensors (1) connected to a mist (2) located on a light street pole collecting temperature and air quality data. The data is transferred by a software component running on the mist that collects it from the sensors and forwards it to a Fog node (3) located nearby through Wi-Fi to be stored and processed. The fog node sends the data to the cloud (4) using an Internet connection, so the data is available to be consumed by different applications. The number of connected devices, sensors, and even Fog nodes scattered around the city may vary, and IoTManS needs to be prepared to meet these needs.

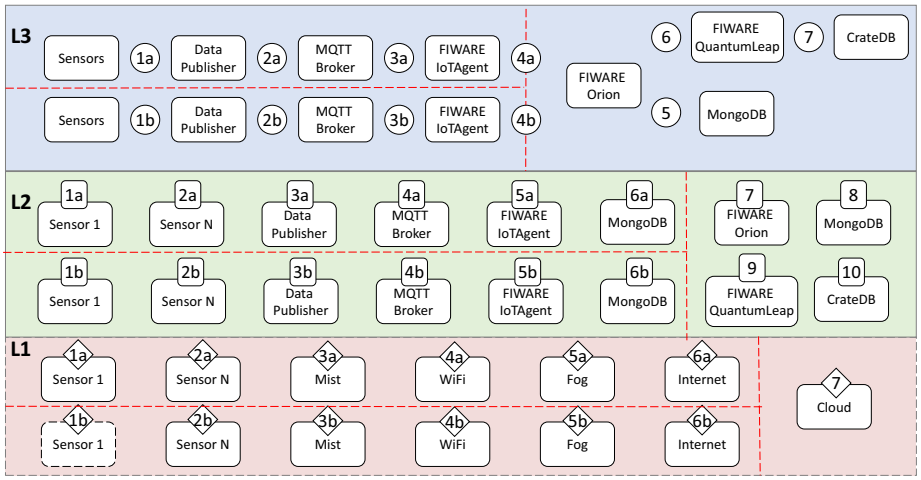

Fig. 12 Scalability test—components

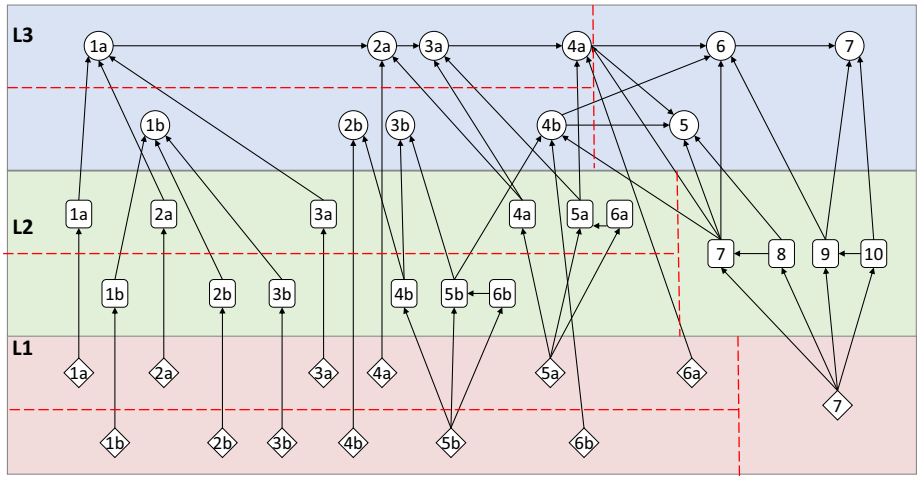

Fig. 13 Scalability test—DAG

The scenario described can be represented in IoTManS as illustrated in Fig. 12, where each end-to-end path is described. Figure 13 illustrates the DAG used to manage the smart city solution. Each new component—sensor, device, connection, fog—can be readily added as a virtual entity and attached to the DAG. Similarly, if one component needs to be removed, it can be achieved by removing the node, the branch related to it, and the edges connecting it to other independent nodes that can be reused by different components, like the fog in this example.

Scalability is evaluated for the passive mode since the active mode consists of polling the graph at predetermined time intervals, and the IoT Manager does not react to them in between polling events. In other words, the IoT Manager only

Table 1 Failure detection results

	Processing time	Confidence interval
Failure detection on active mode	90036 ms	5.64 ms
Failure detection on passive mode	806 ms	6.6 ms
Operation recovery active mode	94056 ms	2.98 ms
Operation recovery passive mode	820 ms	7.3 ms

considers the graph state when it starts the polling, ignoring changes even if they occur during the processing time. This experiment evaluates the scalability of the IoT Manager operating in the passive mode, where it receives multiple messages from different monitored components, detects failures, updates the DAG, and then creates alerts. The purpose of the experiment is to determine the number of components that can be managed simultaneously. The scenario for the Smart City solution illustrated in Fig. 13 was simulated varying the number of components—sensors, Mist nodes, Fog nodes—where IoT Agents send monitoring data simultaneously.

The performance of the underlying platform hosting IoTManS interferes with the results since the IoT Manager is a consumer of the IoT Platform and treats each component as an entity. The experiments evaluate the performance of the IoT Manager using both FIWARE and ThingsBoard platforms managing 1000, 1500, and 5000 components for the scenario depicted by Fig. 13. Due to the number of devices, we use simulated components sending data every 60 seconds using the Poisson distribution with 8, 10, and 40 messages per second. These numbers represent, respectively, low, medium, and high traffic and were chosen based on our experience on performance evaluations of IoT environments explored in [52, 53] and [30].

Each component has a 5% chance to fail in every message. When a component fails, a notification is generated with a failure timestamp and sent to the IoT Platform that forwards it to the IoT Manager. The IoT Manager handles the message, updates the graph, and creates an alert storing the current timestamp. The experiment evaluates the maximum number of managed components that the IoT Manager can handle before losing data using two IoT Platforms. The processing time is the main factor analyzed, comparing the timestamp of the failure to the timestamp when IoT Manager detects and handles the failure.

6 Results

We present here the results of the two sets of experiments. Each experiment was executed 50 times with an asymptotic confidence interval of 95%.

6.1 Failure Detection

Table 1 presents the results of the first set of experiments, Failure Detection.

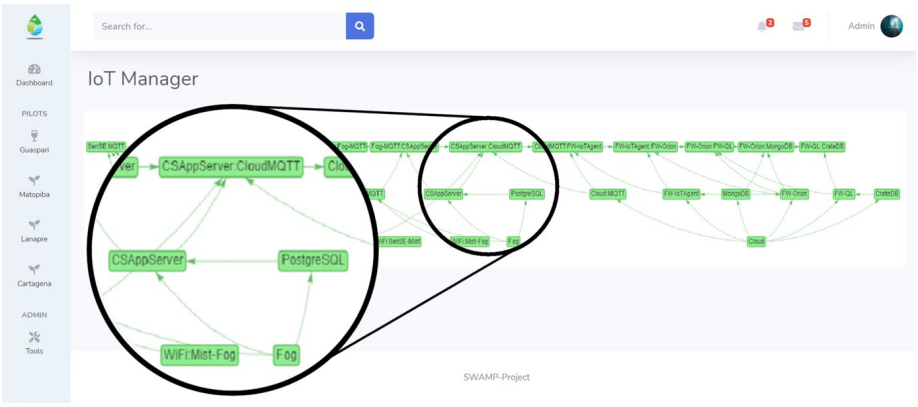

Fig. 14 Management graph on the dashboard before failure

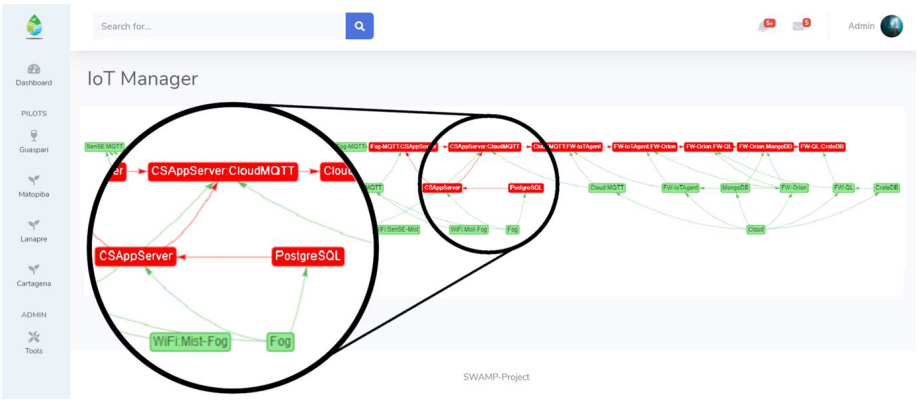

Fig. 15 Management graph on dashboard after failure

– Failure detection on active mode: after the Docker stop command executed by the command injector, the IoT Manager detects the malfunction of a component, creates the alarms, determines the root cause, and determine which components will be affected by the failure in 90.36 s with a 5.64 confidence interval. Figure 14 illustrates the DAG shown in 10 failure in the Layer 2 component PostgreSQL located on the Fog before the failure, changing to the representation on Fig. 15 after the detection time. Whenever an interruption on the dataflow is detected in the active mode, the IoT Manager searches the graph in the reverse path to isolate and locate the root cause of the failure. After detecting the root cause, in the next iteration (60 s), the IoT Manager identifies that the ChirpStack Application Server also fails since it depends on PostgreSQL. In other words, the IoT Manager can detect the failure in other components before the effective detection by the IoT Agents, since the Chirp-

Stack Application Server will try to reconnect to PostgreSQL for some time, not necessarily going down at the same time of PostgreSQL.
- Failure detection on passive mode: After the failure is injected, the IoT Manager detects and processes the notification in 806 ms with a confidence interval of 6.6ms.

- Operation recovery active mode: after the system recovered from a failure state, the IoT Manager needs 94.56 s with a confidence interval of 2.98 s to detect that the system is fully operative again. The results are similar to the Failure detection on active mode since both scenarios identify state changes in the system.
- Operation recovery passive mode: the IoT Manager detects and processes the notification in 820 ms with a confidence interval of 7.3 ms.

The difference between the significantly lower processing time needed by passive mode and active mode can be explained by the fact that the passive mode operates as event-driven while the active mode operates as time-driven. In event-driven operations, the notification occurs immediately after the fail, reducing the processing time. In the time-driven operation, the IoT Agent operates on a fixed time interval, being necessary to wait that time to publish the data. On the other hand, the passive mode IoT Agent needs to interact directly with the managed device to determine their state, which, since IoT Solutions are highly heterogeneous, is not always a possibility. The active mode can operate on devices that do not allow direct interaction by considering the states of other components connected to these devices. The combination of active and passive modes demonstrates how the system can fulfill the needs to operate on complex and heterogeneous IoT solutions.

Regarding the processing time in both passive and active modes, IoTManS demands a time slightly lower than the sampling time, making it adaptable to different IoT solutions and needs. In comparison with fault detection techniques such as fall-curve [54], which analyzes the behavior of the devices to detect a failure, the IoTManS performs better since it does not demand specific hardware and several samples to detect the fault. Fault tree analysis techniques adopted in [55] depend on historical data to create a minimally valid fault detection mechanism. According to Power and Kotonya, a Complex Event Processor technique operating in a distributed manner takes an average of 100 ms to process 100 events [56]. Besides having a lower failure detection time, IoTManS processing time also includes the diagnosis since it can identify the root cause and how the fault impacts the system.

Fault diagnosis includes three tasks: (i) fault detection—detect a fault or malfunction in the systems determining when it occurs; (ii) fault isolation—to determine the location of the faulty component; (iii) fault identification—determine the type, shape, and size of the fault [57]. In this experiment, IoTManS successfully implemented the three tasks of fault diagnosis with success using a signal-based fault diagnosis. The information gathered by IoTMan Agents can be used in future work to implement more robust fault diagnosis techniques in different components such

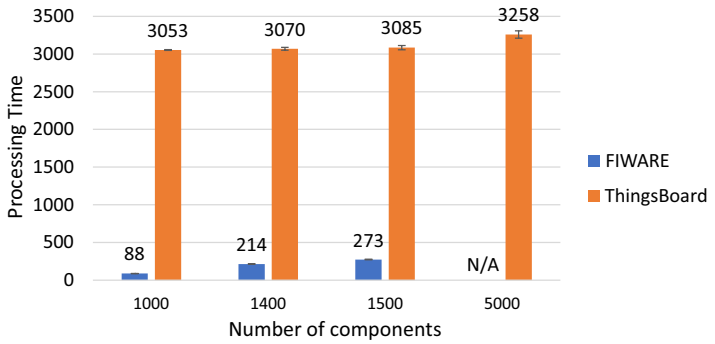

Fig. 16 Processing time on scalability scenario

as surveyed in [57] and [58], and specific techniques focused on software repair as surveyed in [59].

6.2 Scalability

Figure 16 shows the results obtained on the scalability experiments. Even though the FIWARE implementation of the IoT Manager had a lower processing time, the platform is more affected by the increasing number of components. For example, the processing time is significantly affected from the 1000 scenario to the 1500 scenario (88.13–273,79ms), compared to the variation on the exact scenarios for the ThingsBoard (3053.44–3085.32ms).

With 1500 components, FIWARE starts to refuse 10% of the messages received. This behavior occurs due to the dependence of FIWARE Orion on the MongoDB database, which demands a lot of computational resources for the continuous handling of messages creating a rapid shortage of resources. This behavior and limitation of the FIWARE platform were also identified in [5] and [30] that analyze the performance of the FIWARE platform in different Smart Irrigation scenarios, identifying the MongoDB as the bottleneck of the FIWARE platform. Furthermore, FIWARE Orion also lacks efficient message management and message queuing, simply rejecting messages that cannot be dealt with immediately. As a result, FIWARE was not able to execute the scenario with 5000 components.

The ThingsBoard Platform implementation of the IoT Manager requires considerably more time to handle the messages than FIWARE since the platform operation is different, not just forwarding the message but internally processing, storing, and forwarding the message using a plugin. On the other hand, it reveals higher stability in supporting a large number of components. The Things Boards platform was able to handle 5000 components sending data every 60 seconds without losing data. In both implementations, the IoT Manager received and handled all messages without any loss. In other words, the IoT Platform drops data when its limits are reached.

7 Conclusion

Our IoTManA and IoTManS approach for the management, deployment, configuration, monitoring, and maintenance of IoT solutions is a generic approach applied to different applications, such as Smart Agriculture, Smart Cities, Smart Health Care, and Smart Industry. The abstraction obtained with the virtualization of the components organized in a multi-layer architecture can provide management at levels such as system, application, device configuration, deployment, monitoring, and failures, automatically informing root causes of problems. Furthermore, the graph representation and visualization allow the understanding and mapping of the complexity and components and how components relate in an IoT solution.

Direct acyclic graphs are a generic method to correlate different components in an inherently distributed, complex, and heterogeneous IoT environment. This approach could be modeled in a way that allows entities to be dependent on their characteristics. For example, an application requiring a minimum free disk space to operate correctly can depend on its virtual entity and monitor the available disk space needed to be available, creating alarms or taking actions to guarantee its execution. In addition, the graph-based approach can increase context awareness to IoT Manager. The IoT Management Architecture and IoT Management System are successfully implemented in a real IoT Solution using different IoT Platforms as its core, managing with success at multiple IoT solutions levels and showing that the proposal is generic, scalable, and no technology dependent on the managed environment or in a specific IoT Platform to manage the messages.

Future work intends to explore the data collected by IoT Agents and managed by IoT Manager to develop mist, fog, and cloud awareness and placement, deployment, and resource management techniques. Finally, we intend to explore techniques applied in the data quality analysis necessary to Layer 4. Finally, applying the concepts proposed in this paper, we intend to develop a generic self-managed system.

Funding Funding was provided by ministério da ciência, tecnologia e inovações and h2020 industrial leadership.

References

1. Singh, K.J., Kapoor, D.S.: Create your own internet of things: a survey of IoT platforms. IEEE Consum. Electron. Mag. **6**(2), 57–68 (2017). https://doi.org/10.1109/MCE.2016.2640718
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010). https://doi.org/10.1016/j.comnet.2010.05.010
3. Martinez, I., Hafid, A.S., Jarray, A.: Design, resource management and evaluation of fog computing systems: a survey. IEEE Internet of Things J. (2020). https://doi.org/10.1109/MCE.2016.2640718
4. Silva, J..D..C.., Rodrigues, J.J., Al-Muhtadi, J., Rabêlo, R.A., Furtado, V.: Management platforms and protocols for Internet of Things: a survey. Sensors **19**(3), 676 (2019). https://doi.org/10.3390/s19030676

5. Kamienski, C., Soininen, J.P., Taumberger, M., Dantas, R., Toscano, A., Salmon Cinotti, T., Filev Maia, R., Torre Neto, A.: Smart water management platform: IoT-based precision irrigation for agriculture. Sensors **19**(2), 276 (2019). https://doi.org/10.3390/s19020276
6. Yun, M., Yuxin, B.: Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. In: 2010 International conference on advances in energy engineering, pp. 69–72. IEEE (2010). https://doi.org/10.1109/ICAEE.2010.5557611
7. Omoniwa, B., Hussain, R., Javed, M.A., Bouk, S.H., Malik, S.A.: Fog/edge computing-based IoT (FECIoT): architecture, applications, and research issuesa. IEEE Internet of Things J. **6**(3), 4118–4149 (2018). https://doi.org/10.1109/JIOT.2018.2875544
8. Khan, R., Khan, S.U., Zaheer, R., Khan, S.: Future Internet: the Internet of Things architecture, possible applications and key challenges. In: 2012 10th international conference on frontiers of information technology, pp. 257–260. IEEE (2012). https://doi.org/10.1109/FIT.2012.53
9. Mashal, I., Alsaryrah, O., Chung, T.Y., Yang, C.Z., Kuo, W.H., Agrawal, D.P.: Choices for interaction with things on Internet and underlying issues. Ad Hoc Netw. **28**, 68–90 (2015). https://doi.org/10.1016/j.adhoc.2014.12.006
10. Peña, M.A.L., Fernández, I.M.: SAT-IoT: An architectural model for a high-performance fog/edge/cloud IoT platform. In: 2019 IEEE 5th world forum on internet of things (WF-IoT), pp. 633–638. IEEE (2019). https://doi.org/10.1109/WF-IoT.2019.8767282
11. C., S.C.: A survey on architecture, protocols and challenges in IoT. Wireless Pers Communications (2020). https://doi.org/10.1007/s11277-020-07108-5
12. Yaqoob, I., Ahmed, E., Hashem, I.A.T., Ahmed, A.I.A., Gani, A., Imran, M., Guizani, M.: Internet of things architecture: recent advances, taxonomy, requirements, and open challenges. IEEE Wirel. Commun. **24**(3), 10–16 (2017). https://doi.org/10.1109/MWC.2017.1600421
13. Washizaki, H., Ogata, S., Hazeyama, A., Okubo, T., Fernandez, E.B., Yoshioka, N.: Landscape of architecture and design patterns for IoT systems. IEEE Internet of Things J. **7**(10), 10091–10101 (2020). https://doi.org/10.1109/JIOT.2020.3003528
14. 41, I.J.S.: Internet of Things and related: "ISO/IEC 30141. Internet of Things (IoT)—Reference Architecture (2018)
15. da Cruz, M.A., Rodrigues, J.J., Sangaiah, A.K., Al-Muhtadi, J., Korotaev, V.: Performance evaluation of IoT middleware. J. Netw. Comput. Appl. **109**, 53–65 (2018). https://doi.org/10.1109/GLOCOM.2018.8647381
16. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S.: Middleware for Internet of Things: a survey. IEEE Internet of Things J. **3**(1), 70–95 (2015). https://doi.org/10.1109/JIOT.2015.2498900
17. Lueth, K.: IoT Platform Companies Landscape 2019/2020: 620 IoT Platforms globally. IoT Analytics, Dec (2019)
18. Calderoni, L., Magnani, A., Maio, D.: IoT Manager: a case study of the design and implementation of an Open Source IoT Platform. In: 2019 IEEE 5th world forum on internet of things (WF-IoT), pp. 749–754. IEEE (2019). https://doi.org/10.1109/WF-IoT.2019.8767304
19. Yang, J., Park, H., Kim, Y., Choi, J.K.: Programmable objectification and Instance Hosting for IoT nodes. In: 2013 19th Asia-Pacific conference on communications (APCC), pp. 603–608. IEEE (2013). https://doi.org/10.1109/APCC.2013.6766019
20. Hejazi, H., Rajab, H., Cinkler, T., Lengyel, L.: Survey of platforms for massive IoT. In: 2018 IEEE international conference on future IoT technologies (future IoT), pp. 1–8. IEEE (2018). https://doi.org/10.1109/FIOT.2018.8325598
21. Girau, R., Pilloni, V., Atzori, L.: The virtual user: The holistic manager of our IoT applications. In: 2018 IEEE 4th world forum on internet of things (WF-IoT), pp. 149–154. IEEE (2018). https://doi.org/10.1109/WF-IoT.2018.8355115
22. Ray, P.P.: A survey on Internet of Things architectures. J. King Saud Univ.-Comput. Inf. Sci. **30**(3), 291–319 (2018). https://doi.org/10.1016/j.jksuci.2016.10.003
23. Kyuyeong, J., Hyojin, P., Jinhong, Y., Yongrok, K., Kyun, C.J.: A study of Instance Manager for programmable IoT nodes. In: 2014 IEEE 3rd global conference on consumer electronics (GCCE), pp. 350–351. IEEE (2014). https://doi.org/10.1109/GCCE.2014.7031289
24. Harrand, N., Fleurey, F., Morin, B., Husa, K.E.: ThingML: a language and code generation framework for heterogeneous targets. In: Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems, pp. 125–135 (2016). https://doi.org/10.1145/2976767.2976812

25. Kim, J., Yu, S., Lee, J.: Short paper: Wireless sensor network management for sustainable Internet of Things. In: 2014 IEEE world forum on internet of things (WF-IoT), pp. 177–178. IEEE (2014). https://doi.org/10.1109/WF-IoT.2014.6803147
26. Pahl, M.O.: Multi-tenant IoT service management towards an IoT app economy. In: 2019 IFIP/IEEE symposium on integrated network and service management (IM), pp. 1–4. IEEE (2019)
27. Karmakar, K.K., Varadharajan, V., Nepal, S., Tupakula, U.: SDN enabled secure IoT architecture. IEEE Internet of Things J. **8**, 6549–6564 (2020)
28. Ray, S., Bhunia, S., Jin, Y., Tehranipoor, M.: Security validation in IoT space. In: 2016 IEEE 34th VLSI test symposium (VTS), pp. 1–1. IEEE (2016). https://doi.org/10.1109/VTS.2016.7477288
29. Brut, M., Gatellier, P., Excoffier, D., Salhi, I., Cherrier, S., Ghamri, Y., Dumont, N., Lopez-Ramos, M.: When devices become collaborative: supporting device interoperability and behaviour reconfiguration across emergency management scenario. In: 2014 IEEE world forum on internet of things (WF-IoT), pp. 259–264. IEEE (2014). https://doi.org/10.1109/WF-IoT.2014.6803169
30. Zyrianoff, I., Heideker, A., Silva, D., Kleinschmidt, J., Soininen, J.P., Salmon Cinotti, T., Kamienski, C.: Architecting and deploying IoT smart applications: a performance-oriented approach. Sensors **20**(1), 84 (2020). https://doi.org/10.3390/s20010084
31. Mostafa, N., Al Ridhawi, I., Aloqaily, M.: Fog resource selection using historical executions. In: 2018 third international conference on fog and mobile edge computing (FMEC), pp. 272–276. IEEE (2018). https://doi.org/10.1109/FMEC.2018.8364078
32. Shaik, S., Baskiyar, S.: Resource and service management for Fog Infrastructure as a Service. In: 2018 IEEE international conference on smart cloud (SmartCloud), pp. 64–69. IEEE (2018). https://doi.org/10.1109/SmartCloud.2018.00019
33. Saha, A., Jindal, S.: EMARS: efficient management and allocation of resources in serverless. In: 2018 IEEE 11th international conference on cloud computing (CLOUD), pp. 827–830. IEEE (2018). https://doi.org/10.1109/CLOUD.2018.00113
34. Xu, J., Palanisamy, B., Ludwig, H., Wang, Q.: Zenith: Utility-aware resource allocation for edge computing. In: 2017 IEEE international conference on edge computing (EDGE), pp. 47–54. IEEE (2017). https://doi.org/10.1109/IEEE.EDGE.2017.15
35. Li, Y., Xu, L.: The service computational resource management strategy based on edge-cloud collaboration. In: 2019 IEEE 10th international conference on software engineering and service science (ICSESS), pp. 400–404. IEEE (2019). https://doi.org/10.1109/ICSESS47205.2019.9040830
36. Brogi, A., Carrasco, J., Durán, F., Pimentel, E., Soldani, J.: Robust management of trans-cloud applications. In: 2019 IEEE 12th international conference on cloud computing (CLOUD), pp. 219–223. IEEE (2019). https://doi.org/10.1109/CLOUD.2019.00046
37. Linaje, M., Berrocal, J., Galan-Benitez, A.: Mist and edge storage: Fair storage distribution in sensor networks. IEEE Access **7**, 123860–123876 (2019). https://doi.org/10.1109/ACCESS.2019.2938443
38. redha BOUAKOUK, M., ABDELLI, A., MOKDAD, L.: Survey on the Cloud-IoT paradigms: Taxonomy and architectures. In: 2020 IEEE symposium on computers and communications (ISCC), pp. 1–6. IEEE (2020). https://doi.org/10.1109/ISCC50000.2020.9219638
39. Elliott, D., Otero, C., Ridley, M., Merino, X.: A cloud-agnostic container orchestrator for improving interoperability. In: 2018 IEEE 11th international conference on cloud computing (CLOUD), pp. 958–961. IEEE (2018). https://doi.org/10.1109/CLOUD.2018.00145
40. Jain, R., Tata, S.: Cloud to edge: distributed deployment of process-aware IoT applications. In: 2017 IEEE international conference on edge computing (EDGE), pp. 182–189. IEEE (2017). https://doi.org/10.1109/IEEE.EDGE.2017.32
41. Xu, P., Su, J., Zhang, Z.: Distributed hybrid cloud management platform based on rule engine. In: 2018 IEEE 11th international conference on cloud computing (CLOUD), pp. 836–839. IEEE (2018). https://doi.org/10.1109/CLOUD.2018.00116
42. Avasalcai, C., Tsigkanos, C., Dustdar, S.: Decentralized resource auctioning for latency-sensitive edge computing. In: 2019 IEEE international conference on edge computing (EDGE), pp. 72–76. IEEE (2019). https://doi.org/10.1109/EDGE.2019.00027
43. Sinche, S., Raposo, D., Armando, N., Rodrigues, A., Boavida, F., Pereira, V., Silva, J.S.: A survey of IoT management protocols and frameworks. IEEE Commun. Surv. Tutor. **22**(2), 1168–1190 (2019). https://doi.org/10.1109/COMST.2019.2943087
44. Nickerson, R., Muntermann, J., Varshney, U., Isaac, H.: Taxonomy development in information systems: Developing a taxonomy of mobile applications. HAL, Working Papers (2009)
45. Chirpstack. "https://www.chirpstack.io". Accessed: 2021-05-10
46. Mobile Alliance, O.: Ngsi requirements - ngsi oma-rd-ngsi-v1.0. 2012 (2012)

47. Cerveira, F., Barbosa, R., Madeira, H.: Mitigating virtualization failures through migration to a co-located hypervisor. IEEE Access (2021). https://doi.org/10.1109/ACCESS.2021.3098644
48. Mauro, M.D., Galatro, G., Longo, M., Postiglione, F., Tambasco, M.: Comparative performability assessment of SFCs: the case of containerized IP multimedia subsystem. IEEE Trans. Netw. Serv. Manag. **18**(1), 258–272 (2021). https://doi.org/10.1109/TNSM.2020.3044232
49. Togneri, R., Camponogara, G., Soininen, J.P., Kamienski, C.: Foundations of data quality assurance for IoT-based smart applications. In: 2019 IEEE Latin-American conference on communications (LATINCOM), pp. 1–6. IEEE (2019). https://doi.org/10.1109/LATINCOM48065.2019.8937930
50. Zyrianoff, I.: SenSE - sensor simulation environment. 2017, github repository. "github.com/ivanzy/SenSE-Sensor-Simulation-Environment". Accessed 10 May, 2021
51. Kalitay, H.K., Nambiarz, M.K.: Designing WANem: A wide area network emulator tool. In: 2011 Third international conference on communication systems and networks (COMSNETS 2011), pp. 1–4. IEEE (2011). https://doi.org/10.1109/COMSNETS.2011.5716495
52. Zyrianoff, I., Heideker, A., Silva, D., Kamienski, C.: Scalability of an Internet of Things platform for smart water management for agriculture. In: 2018 23rd conference of open innovations association (FRUCT), pp. 432–439. IEEE (2018). https://doi.org/10.23919/FRUCT.2018.8588086
53. Quete, B.e.a.: Understanding the tradeoffs of LoRaWAN for IoT-based smart irrigation. IEEE international workshop on metrology for agriculture and forestry (MetroAgriFor) (2020). https://doi.org/10.1109/MetroAgriFor50201.2020.9277566
54. Chakraborty, T., Nambi, A.U., Chandra, R., Sharma, R., Swaminathan, M., Kapetanovic, Z., Appavoo, J.: Fall-curve: A novel primitive for iot fault detection and isolation. In: Proceedings of the 16th ACM conference on embedded networked sensor systems, SenSys '18, p. 95–107. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3274783.3274853
55. Chen, Y., Zhen, Z., Yu, H., Xu, J.: Application of fault tree analysis and fuzzy neural networks to fault diagnosis in the internet of things (iot) for aquaculture. Sensors (2017). https://doi.org/10.3390/s17010153
56. Power, A., Kotonya, G.: Bobocep: Distributed complex event processing for resilient fault-tolerance support in iot. In: 2020 IEEE sixth international conference on big data computing service and applications (BigDataService), pp. 109–112 (2020). https://doi.org/10.1109/BigDataService49289.2020.00024
57. Gao, Z., Cecati, C., Ding, S.X.: A survey of fault diagnosis and fault-tolerant techniques–part i: fault diagnosis with model-based and signal-based approaches. IEEE Trans. Ind. Electron. **62**(6), 3757–3767 (2015). https://doi.org/10.1109/TIE.2015.2417501
58. Gao, Z., Cecati, C., Ding, S.X.: A survey of fault diagnosis and fault-tolerant techniques–part ii: fault diagnosis with knowledge-based and hybrid/active approaches. IEEE Trans. Ind. Electron. **62**(6), 3768–3774 (2015). https://doi.org/10.1109/TIE.2015.2419013
59. Gazzola, L., Mariani, L., Micucci, D.: Automatic software repair: A survey. In: 2018 IEEE/ACM 40th international conference on software engineering (ICSE), pp. 1219–1219 (2018). https://doi.org/10.1145/3180155.3182526

Dener Silva is a Technologist in Informatics for Business Management from the Faculty of Technology of Mauá (2008). Master in Computer Science (2013) from Federal University of ABC, developing research in the area of management of optical networks. He is a public employee at the Federal University of ABC where he works in the area of scientific computing, also responsible for computer labs, cloud environments and super computers. Currently a PhD student in the Information Engineering program at Federal University of ABC conducting research in the area of management of distributed computing systems, Internet of Things, smart cities, multi-agent, cloud computing and fog computing.

Alexandre Heideker received a B.S. degree in computer science from the Federal University of ABC (UFABC), Santo André Brazil, in 2011, an M.S. degree from the UFABC in 2016, and a Ph.D. in Information Engineering from UFABC in 2021, where currently he also holds the position of researcher collaborator. His current research interests include network softwarization, Infrastructure as a Service (IaaS), cloud/fog computing, and IoT.

Ivan D. Zyrianoff is a PhD student from the University of Bologna and a member of the IoT-Prism lab. He holds a B.S. degree in Computer Science and an M.S. degree in Information Engineering from the Federal University of ABC, Brazil. His current research topics encompass interoperability for the Internet of Things and Edge Computing.

João H. Kleinschmidt received his B.S degree in Computer Engineering and Master in Computer Science at the Pontifical Catholic University of Paraná, Brazil, in 2001 and 2004, respectively. He earned a Ph.D. in Electrical Engineering from the State University of Campinas in 2008. He is currently associate professor at the Federal University of ABC in Santo André, SP, Brazil. His research interests are Internet of Things, computer networks, distributed systems, and information security.

Luca Roffia Assistant Professor at the University of Bologna and co-founder of VAIMEE srl spinoff, his research interests are mainly focused on investigating solutions based on Semantic Web and Linked Data technologies to enable Interoperable Data Spaces. He is the principal investigator of SEPA (SPARQL Event Processing Architecture) a publish-subscribe software architecture based on Linked Data technologies, like SPARQL, RDF and OWL. SEPA can support the development of open, distributed and context aware applications in all scenarios characterized by dynamic, heterogenous and not structured data like for example the Web of Things.

Juha-Pekka Soininen is a principal scientist at VTT Technical Research Centre of Finland. He received his MSc, LicTech and DSc (Technology) degrees from University of Oulu 1987, 1997 and 2004 respectively. He has been a research and senior research scientist before 2007, a research professor between 2007 and 2014, and a principal scientist since 2014. His current research deals with distributed computing systems, IoT and cyber-physical systems, and data spaces. He has over 80 peer-reviewed scientific publications.

Carlos A. Kamienski is a Full Professor of Computer Science at the Federal University of ABC (UFABC, Brazil). For eight years, he led the NUVEM Strategic Research Unit comprising faculty members and students working in five broad research lines, namely smart societies, virtual sensations, connected mobility, extreme computing, and integrated universes. He was the Brazilian coordinator of SWAMP from 2017 to 2021 (swamp-project.org), an EU-Brazil collaborative research project that developed IoT-based methods and approaches for smart water management in precision irrigation. His current research interests include Internet of Things, smart agriculture, smart cities, fog computing, network softwarization, and Future Internet.

Authors and Affiliations

Dener Silva[1] · Alexandre Heideker[1] · Ivan D. Zyrianoff[2] · João H. Kleinschmidt[1] · Luca Roffia[2] · Juha-Pekka Soininen[3] · Carlos A. Kamienski[1]

Alexandre Heideker
alexandre.heideker@ufabc.edu.br

Ivan D. Zyrianoff
ivandimitry.ribeiro@unibo.it

João H. Kleinschmidt
joao.kleinschmidt@ufabc.edu.br

Luca Roffia
luca.roffia@unibo.it

Juha-Pekka Soininen
Juha-Pekka.soininen@vtt.fi

Carlos A. Kamienski
carlos.kamienski@ufabc.edu.br

[1] Federal University of ABC, Santo André, Brazil

[2] University of Bologna, Bologna, Italy

[3] VTT Technical Research Centre, Otaniemi, Finland