Solving the IoT Cascading Failure Dilemma Using a Semantic Multi-agent System

Amal Guittoum[1,2(✉)], François Aïssaoui[1], Sébastien Bolle[1], Fabienne Boyer[2], and Noel De Palma[2]

[1] Orange Innovation, Meylan, France
{amal.guittoum,francois.aissaoui,sebastien.bolle}@orange.com
[2] University of Grenoble Alpes - LIG, Grenoble, France
{fabienne.boyer,noel.palma}@univ-grenoble-alpes.fr
https://www.orange.fr,https://www.liglab.fr/en

Abstract. Managing interdependent Internet of Things (IoT) devices can be challenging because different actors, e.g., operators and service providers, propose siloed Device Management (DM) solutions that are unable to handle cascading failures across multiple devices. To address this issue, we propose a novel approach based on a cooperative Multi-agent System (MAS) allowing siloed DM solutions to manage IoT cascading failures automatically and coordinately. The proposed MAS leverages Semantic Web standards to establish a common understanding of device dependencies and failures. It relies on the Digital Twin technology to represent dynamic device dependencies accurately for failure root cause identification. Our approach has been effective in handling cascading failures in Smart Home scenarios, reducing time to repair failure, saving Customer Care costs, and minimizing resource consumption in IoT infrastructure such as energy consumption.

Keywords: IoT Failure Management · Multi-agent system · Ontology · Semantic Digital Twin · Interoperability · Collaboration

1 Introduction

The Internet of Things (IoT) is gaining widespread popularity across multiple domains, such as healthcare and transportation. The pivotal factor for creating value in IoT is IoT devices that can perform tasks with minimal human intervention. To ensure smooth IoT operations, monitoring and managing these devices is essential. This is referred to as IoT Device Management (DM).

In today's IoT systems, DM is provided by siloed DM platforms, e.g., *Amazon Web Service (AWS)*[1], *Orange Live Objects*[2], governed by different actors, which can be operators, service providers, or device manufacturers [1,19,37]. These

[1] https://aws.amazon.com/fr/iot-device-management/.
[2] https://liveobjects.orange-business.com/.

T. R. Payne et al. (Eds.): ISWC 2023, LNCS 14266, pp. 325–344, 2023.
https://doi.org/10.1007/978-3-031-47243-5_18

DM platforms integrate devices built by different manufacturers to perform key DM functions such as firmware updates and Failure Management (FM). However, they are facing a strong limitation regarding the management of *Cascading Failure dilemmas* that arise when the failure of one device instigates the failure of dependent devices and applications managed by different DM actors [44].

Cascading failures are particularly problematic because they generate more customer calls to customer care applications of DM actors. Their mitigation usually requires human intervention, which increases the cost of Customer Care. For example, the Orange company reports a cost of *20*€ for one customer call and *100*€ for sending a technician, where customers perform *100 calls per week* to request IoT device recovery. Moreover, failures are one of the main causes of energy waste in connected environments. Studies show that 25–45% of HVAC energy consumption is wasted due to failures [25]. Despite cascading failures leading to business and environmental damages, there is no existing solution for managing them in multi-actor IoT systems, to the best of our knowledge.

In this paper, we address this open research challenge and propose a practical solution allowing siloed DM actors to manage cascading failures in an automatic and coordinated manner. This solution relies on a cooperative Multi-Agent System (MAS)[3]empowered by Semantic Web and Digital Twin technologies. More precisely, we rely on *cOllaborative caScading fAilure Management Agent (OSAMA)*, a semantic agent to be integrated into the legacy DM platforms in order to help them understand, collaborate and make effective decisions regarding Cascading Failure Management (CFM). OSAMA exploits a set of Semantic Web standards, such as ontologies, in order to simplify failure information exchange and enhance the interoperability among siloed DM platforms. It leverages the Semantic Digital Twin technology[4], modeling dynamic dependency relationships among IoT devices for failure root cause identification. Upon failure, OSAMA agents start a collaborative protocol that allows them to automatically identify the roots of the failures and recover the failed devices.

The contribution of this work includes (i) The IoT-F ontology describing IoT device failures and their recovery actions; (ii) The OSAMA agent leveraging the Belief-Desire-Intention model (BDI) [32] to enable collaborative CFM; (iii) A collaborative CFM protocol; (iv) A proof of concept for the proposed solution demonstrating its potential impact in reducing time to repair failure, saving Customer Care costs and minimizing resource consumption in IoT infrastructures. Moreover, we provide plans for large adoptions of our solution in the DM market.

This paper is organized as follows: We begin by providing the necessary background information to facilitate comprehension of our work. Next, we present a motivating use case. Then, we detail our proposed solution for CFM. Finally, we discuss the evaluation results, related work, and avenues for future research.

A table of acronyms (see Table 1) has been provided to enhance readability.

[3] MAS refers to a network of software agents that operate independently while being loosely connected to address complex problems that are beyond the individual capacities or knowledge of each agent.

[4] A Semantic Digital Twin is a virtual and synchronized representation of real-world entities and processes built using a semantic description.

Table 1. Acronyms used in this paper

Acronym	Meaning
BDI	Belief Desire Intention
CFM	Cascading Failure Management
DKG	Dependency Knowledge Graph
DM	IoT Device Management
DMP	Device Management platform Provider
FKB	Failure Knowledge Base
FM	Failure Management
FMEA	Failure Mode Effect Analysis
MAS	Multi-Agent System
MN	device MaNufacturer
OSAMA	cOllaborative caScading fAilure Management Agent
SP	Service Provider

2 Background

2.1 Overview of Legacy Solutions

Current IoT systems are managed by multiple actors, each having its legacy solution for managing failures on its IoT devices as part of its customer care services. We have conducted a market-based study[5] on the FM capabilities of these siloed DM actors. As a result, we identify the following profiles with distinct FM capabilities, including Device Manufacturers (MN), DM Platform Providers (DMP), and Service Providers (SP).

MNs build and deliver IoT devices to end users and IoT suppliers. Mostly, MNs do not have DM platforms to perform DM operations on their IoT devices. However, they may propose a mobile application-based solution in order to allow end users to perform basic DM operations on IoT devices such as *firmware update*. Moreover, they may acquire information about IoT device failures, their causes, effects, and recovery actions. This failure information is usually identified during the design and test stage for risk assessments using several approaches such as *Failure Mode Effect Analysis* (FMEA) [13]. It may be represented in tables used by customer care services to recover failures manually or build support pages to help end users manage failures on their devices [40]. DMPs propose a DM platform that allows several DM operations such as *firmware update and device reboot*. These DM platforms propose DM as a service solution for end users and enterprises to help them manage their IoT devices. They integrate IoT devices built by different MNs. Some of these DM platforms propose failure detection features using Machine Learning (ML)[6] or alarm-based system[7].

[5] This study is limited by the information available online.
[6] https://www.avsystem.com.
[7] https://aws.amazon.com/fr/iot-device-management/.

They may recover elementary failures on IoT devices remotely using DM operations. However, they do not acquire end-to-end solutions for automatic FM and are limited when the failure spreads across IoT devices managed by different DM platforms. SPs ensure IoT connectivity and provide IoT services via various devices such as Orange's *LiveBox* for connectivity and Samsung's *SmartThings hub* for home automation services. Each SP proposes its own proprietary DM platform for managing its devices. Proprietary DM platforms proposed by SP allow similar features as the DMP DM platform. Moreover, SPs own failure information on their provided devices.

2.2 Complex IoT Device Dependency

Dependency relationships between IoT devices cause failures to propagate across them. In our previous work [17], we provided a taxonomy for dependencies between IoT devices that exacerbate cascading failure propagation, including *(1)Direct dependencies* when IoT devices utilize each other's services (service dependency), and *(2)Indirect dependencies* that are generated due to interactions between sensors and actuators via the physical environment (environment-based dependency), or created by applications running on top of IoT devices, even when an application acts on a set of devices based on the state of other devices (state-based dependency) or when it forwards data flows between them (data-based dependency). These dependency relationships are abundant and dynamic.

2.3 IoT Failures

IoT integrates physical processes with digital connectivity, often using three components represented by devices, connectivity protocols, and cloud platforms [8]. While failures may occur in any of these components, IoT device failures are more common due to several factors, such as their minimal computational resources [28]. Failures on IoT devices are divided into two main classes: (1) *Fail-stop failure* when an IoT device becomes unresponsive to external requests; (2) *Non-fail stop failure* occurs when IoT device response diverges from the correct response, such failed responses are classified in the literature [9,27,28,36] into five categories: *High Variance*: when a device oscillates between states faster than the environment dictates; *Stuck-at*: when a device is expected to change its state, but it fails to do so; *Spike*: when the numeric state of a device increases or decreases at a faster rate than what is determined by the environment; *Outlier*: when a device reports incorrect state for a single poll; *Calibration*: when sensor data shows an offset, i.e., it has a different gain than the actual ground truth value. Cascading failures occur when one device's failure (fail or non-fail stop) spreads to other devices through dependencies.

3 Motivating Use Case

We consider a smart home managed by five DM actors. It illustrates the cascading failure scenarios that are difficult to overcome with market DM solutions.

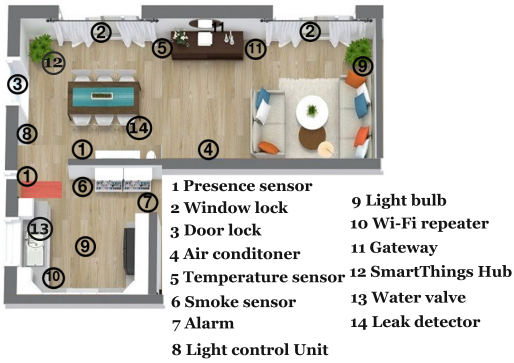


Fig. 1. Smart home architecture

3.1 Smart Home Scenario Architecture

The smart home scenario (see Fig. 1) consists of three intelligent systems deployed in a home consisting of a living room and a kitchen: **1) Light management system:** Relies on a *light sensor*, *presence detection sensors*, *light bulbs* installed in the living room and the kitchen, and a *light control unit*. The latter controls light using the light measurement service supplied by *the light sensor*, the presence detection services of *the presence sensors*, and the light bulbs' services. **2) Temperature management system:** Controls the home temperature using a *temperature sensor* and an *air conditioner*. It is mainly based on automation rules[8] 1–3 described in Table 3. **3) Security control system:** Launches *an alarm* when intruders, fires, or leaks are detected. It consists of *an alarm* that uses the presence detection services provided by *the presence sensors* to detect intruders. *The alarm* also uses temperature, smoke, and leak sensors' services for fire and leak detection. This system is reinforced by rules 4–7.

A gateway connects devices in the living room to the Internet, while a Wi-Fi repeater connects the kitchen devices. The SmartThings platform[9] [33] enables automation rules described in Table 3 using a *SmartThings Hub*. Devices in the smart home are managed by five DM actors with different profiles each proposing its own solution for managing devices integrated into its system (see Table 2).

3.2 Illustration of Cascading Failure Dilemma

Due to the dependencies among IoT devices, a failure of one of them can generate multiple cascading failures. Take Scenario 01 (see Table 4) as an example, in which a cascading failure occurs due to a high variance failure on the leak detector. The failure affects the alarm and the water valve, which are state-dependent

[8] Automation rules allow the automated composition of IoT services in a connected environment.

[9] SmartThings is Samsung's IoT platform that enables automation rules across IoT devices in Smart Homes.

Table 2. DM actors managing the Smart Home

DM actors	Profile	Managed devices
Orange	DMP	Leak detector, water valve, temperature sensor, windows door
	SP	Gateway, WI-FI repeater
Samsung	SP	SmartThings Hub
Amazon	DMP	Alarm, lights bulb airconditioner, smoke sensor, light control unit
Philips	MN	Motion sensor, light bulbs, light control unit windows, door, alarm
Kelvin	MN	Temperature sensor, airconditioner, leak detector, water valve

Table 3. The smart home automation rules

No	Type	Automation Rule
1	Comfort	Adjust the air conditioner regarding the temperature returned by the temperature sensor
2	Comfort	Open the two windows when the air conditioner is deactivated
3	Comfort	Close the two windows and turn on the air conditioner when the temperature exceeds a threshold
4	Security	Turn on the alarm and unlock the door and both windows upon detection of fire
5	Security	Turn on the alarm and close the water valve when leak is detected
6	Security	Notifies the User, closes the windows, closes the door, and turns on the alarm when detecting an intruder while the User is out of the home
7	Security	Set light bulbs to red when the alarm is activated

Table 4. Cascading Failure scenarios

Scenario	Root cause	Impacted devices	Detected at
1	High Variance on the leak detector	Alarm Water Valve Light bulbs	Light bulbs
2	High Variance on Smoke sensor	Alarm Light bulbs door windows	Door
3	Stuck at no motion on the motion sensor	Light control unit Light bulb	Light bulbs
4	Outlier on the temperature sensor	Window Airconditioner	Airconditioner
5	Stuck at smoke detected on the smoke sensor	Alarm Light bulbs Door Windows	Alarm
6	Spikes on the temperature sensor	Window Airconditioner	Airconditioner
7	Fail stop on the WIFI repeater	Alarm, light bulb smoke sensor, water valve	Alarm
8	Stuck at leak detected on the leak detector	Alarm Water Valve Light bulbs	Water valve

on the leak detector (see Rule 5 in Table 3). Additionally, the light bulbs are affected as they are state-dependent on the alarm, as per Rule 7. Such cascading failures pose a significant challenge for DM actors. In this case, the alarm and light bulbs are managed by the *Amazon* DM platform, whereas the water valve and leak detector are managed by the *Orange* DM platform (see Table 2). As a result, these siloed DM actors cannot identify the failure's root cause. Furthermore, the failure recovery information is distributed across different device manufacturers: *Kelvin* and *Philips*, which further complicates the situation.

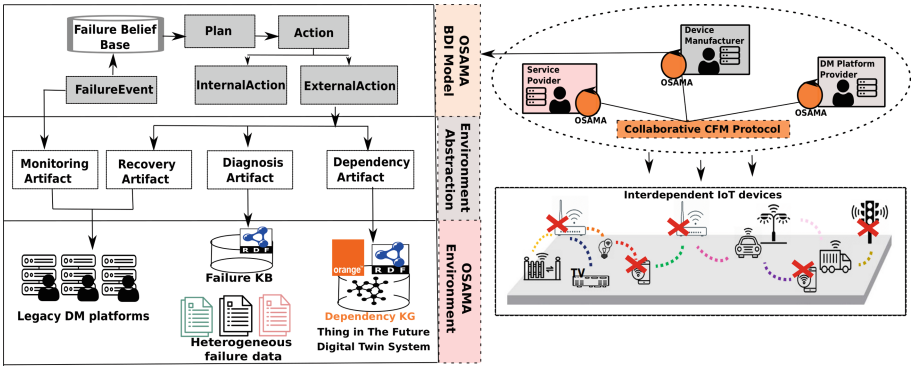


Fig. 2. Overview of Semantic Multi-OSAMA For Collaborative CFM

4 Semantic Multi-OSAMA for Collaborative CFM

MAS enables the integration of multiple legacy systems by developing an agent wrapper around them, enabling their participation in collaborative problem-solving and decision-making processes [42]. Relying on this advantage, we propose a MAS to help legacy DM solutions automatically manage cascading failure dilemmas. Our solution consists of a set of cooperative agents called *OSAMA* to be integrated by DM actors in their legacy solutions. These *OSAMAs* adopt a BDI model to handle cascading failures. They collaborate according to a *collaborative CFM protocol* to recover from cascading failures that spread across devices managed by different actors.

Within their shared environment, OSAMAs are provided by four (04) *Artifacts* encapsulating external services that they can explore at runtime to ease CFM (see Fig. 2): 1) *Monitoring Artifact*: Allows to monitor IoT devices and detect failures using legacy DM platforms; 2) *Diagnosis Artifact*: Allows to identify failure type and its compensatory actions using Failure Knowledge Base (FKB); 3) *Dependency Artifact*: Thanks to Semantic Digital Twins, this artifact allows to automatically access an accurate view of dynamic dependency relationships between IoT devices in order to ease cascading failure root cause identification; 4) *Recovery Artifact*: Allows to execute recovery actions on IoT devices using legacy DM platforms. In the following, we discuss the *OSAMA* BDI model, artifacts, and the *Collaborative CFM Protocol*.

4.1 OSAMA BDI Model

The *OSAMA* design follows a BDI model, which is helpful in developing autonomous agents in various domains thanks to its flexibility, robustness, and transparency [38]. The BDI agent model aims at programming rational agents based on human mental attitudes of beliefs, desires, and intentions [7,38]. Beliefs correspond to an agent's understanding of its surroundings, other agents, and itself. Desires refer to the conditions an agent wants to achieve, and intentions are the commitments to achieving those desires. In order to accomplish its desires, an agent utilizes a collection of plans executed in specific contextual circumstances. These plans consist of a series of actions that an agent must undertake, given the conditions implied by its belief base. The belief base is updated based on events that the agent perceives from its surrounding environment.

Table 5. OSAMA Internal and External Actions

Action	Type	Description
$sendCFMRequest(device_i, OSAMA_k)$	Internal	Send by the *OSAMA* that initiates the recovery of a detected cascading failure scenario requesting the $OSAMA_k$ to check and recover the $device_i$.
$responseCFMRequest(device, OSAMA_k)$	Internal	Send by the *OSAMA* that participates in the recovery of a cascading failure scenario involving $device_i$, to the $OSAMA_k$, the initiator of the cascading failure recovery.
$requestDiagnosis(OSAMA_k, device_i, S)$	Internal	Send by the *OSAMA* to request diagnosis information from the $OSAMA_k$ for $device_i$ having the symptoms[a] S, when it could not perform diagnosis by itself.
$getDiagnosisAgent(device_i)$	External	Allows an *OSAMA* to get candidate $OSAMA_d$ able to perform diagnosis on device $device_i$ when it could not perform diagnosis by itself.
$getDeviceState(device_i)$	External	Allows an *OSAMA* to check whether the $device_i$ is failed or not by accessing the monitoring artifact.
$getDeviceSymptoms(device_i)$	External	Allows an *OSAMA* to get symptoms of the $device_i$ by accessing the monitoring artifact.
$getDependency(device_i)$	External	Allows an *OSAMA* to get a list of devices to which the $device_i$ depends on by accessing the dependency artifact.
$recover(recoveryAction, device_i)$	External	Allows an *OSAMA* to perform the $recoveryAction$ on the $device_i$ by accessing the recovery artifact.
$diagnosis(device_i, symptoms)$	External	Allows an *OSAMA* to get diagnosis information such as proposed recovery action for the failed $device_i$ based on a set of $symptoms$ by accessing the diagnosis artifact.

[a] Symptoms refers to device characteristics describing device failed states such as memory usage.

Based on the explained BDI model terminology, we define the *OSAMA* as a tuple $<Evt, Blf, Pl, Act>$, where: $Evt = \{evt_1, evt_2, ..., evt_n\}$ represents a set of failure events perceived by the *OSAMA* through the monitoring artifact or reported by other *OSAMAs* during CFM. A failure event

$evt_i = (device_k, sourceType, source)$, where $sourceType$ indicates the failure is detected by monitoring artifact or other *OSAMA* specified by $source$. Failure events allow the *OSAMA* to update its *Belief Base* and take actions to handle failures; $Blf = \{blf_1, blf_2, ..., blf_n\}$ represents positive ground literals in a first-order logical language describing IoT devices state such as $blf_j^i = failed(device_i)$ if $device_i$ is failed, $recovered(device_i)$ otherwise. It is updated when receiving failure events or recovering a failed device; Act represents a set of internal and external actions that the *OSAMA* performs for CFM. Internal actions are executed by the *OSAMA*, while external actions access shared artifacts that abstract external services deployed in the *OSAMA* environment (See Table 5); $Pl = \{p_1, p_2, ..., p_n\}$ represents *OSAMA* plans. A plan $p_i \equiv evt \rightarrow Act_i$ has an event evt, including adding or deleting failure beliefs and receiving CFM requests. Such an event triggers a subset of *OSAMA* actions Act_i to handle failures and CFM requests.

4.2 Diagnosis Artifact

This artifact embeds a Failure Knowledge Base (FKB) that allows *OSAMAs* to get failure information such as possible compensatory actions, given a set of failure symptoms provided by the monitoring artifact, thanks to the SPARQL queries. We assume that each DM actor generates an FKB involving its governed failure information (see Sect. 2.1). These FKBs are built using an ontology called *IoT-F*[10] [14] (see Fig. 3) that we developed using the ontology engineering methodology NeOn [41]. The main purpose of the *IoT-F* ontology is to allow *OSAMAs* to share a global understanding of heterogeneous and distributed failure information. However, it has other intended usages such as assisting DM

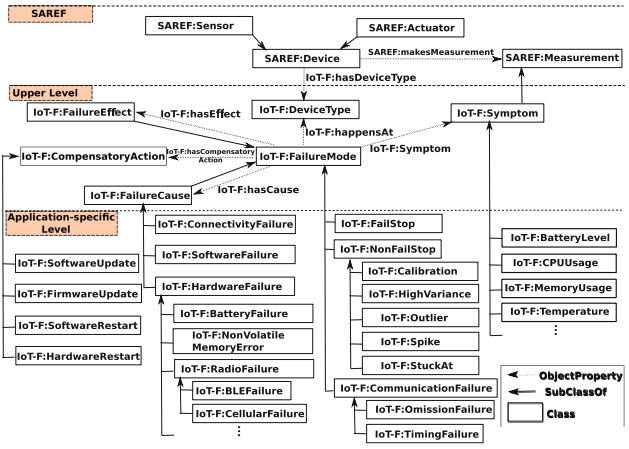


Fig. 3. IoT-F: IoT Failure Ontology

[10] https://iotfontology.github.io/.

actors in structuring their failure information, characterized by heterogeneity, incompleteness, and ambiguity [40]. The IoT-F ontology **reuses** the standardized ontology *SAREF*[11]. Its architecture is based on two levels inspired by the work [13]: 1) The top level is based on the FMEA concepts, which provide a generic model for failure description in any domain of interest. We reused FMEA concepts proposed in [13], which have been inspired by relevant standards such as IEC60812[12] and ISO13372[13]; 2) The application-specific level represents failures in IoT systems. To build the application-specific level, we reused a set of non-ontological resources that describe relevant information about IoT failure, such as literature taxonomies [9,28,29,36] for IoT failure, failure cause, and recovery actions taxonomies, and market DM models mainly *Matter*[14] and TR-181[15]. The main concept of the IoT-F ontology is *IoT-F:FailureMode* representing IoT failures associated with an IoT device type *IoT-F:DeviceType*, described by: a set of symptoms *IoT-F:Symptom* representing failure symptoms, causes *IoT-F:FailureCause*, effects *IoT-F:FailureEffect*, and possible compensatory actions *IoT-F:CompensatoryAction*. Each of these classes is specialized at the application-specific level to describe IoT failures.

4.3 Dependency Artifact

This artifact allows *OSAMAs* to identify failure root causes through the analysis of dependency relationships among IoT devices. It incorporates a framework, developed in our previous work [17], that enables automatic inference and analysis of dynamic dependency relationships among IoT devices using *Semantic Digital Twins*. Namely, the dependency relationships are represented as an IoT Dependency Knowledge Graph (DKG). The DKG serves as a Digital Twin view, representing the current devices and their dependencies.

In conceptual terms, the framework includes an ontology called IoT-D[16] that enables a shared representation of IoT dependencies across heterogenous DM solutions. The IoT-D ontology extends the standardized ontology *SAREF* to describe a set of contextual data that delineate direct and indirect dependencies among devices (refer to Sect. 2.2). By leveraging the IoT-D ontology, the framework automatically constructs the DKG through a three-step process: *Context extraction*, *Entity resolution*, and *Dependency inference*.

1. The context extraction step retrieves the context data, as described in the IoT-D ontology, from the siloed DM solutions and transforms it into KGs.
2. The entity resolution (ER) step aggregates duplicated entities found in the extracted context KGs, such as devices with different representations. This

[11] https://saref.etsi.org/core/v3.1.1/.
[12] https://webstore.iec.ch/publication/26359.
[13] https://www.iso.org/standard/52256.html.
[14] https://csa-iot.org/all-solutions/matter/.
[15] https://usp-data-models.broadband-forum.org/.
[16] https://iotdontology.github.io/.

step relies on the advanced features of the *SHACL standard*[17], namely SHACL rules and SHACL functions. The idea is to use a SHACL rule to infer the *owl:sameAs* relationships between duplicated entities in the extraction KGs. This SHACL rule uses SHACL functions to compute similarity metrics between attributes of two entities and then decide on the inference of the *owl:sameAs* relationships between them. SHACL functions may embed several similar functions. In our work, we used string similarity functions.

3. The dependency inference step builds the DKG by inferring dependency relationships in the aggregated KGs. It leverages a set of SHACL rules to infer dependency relationships among IoT device representations by reasoning on their contextual relationships provided in the aggregated context KGs.

In the DKG, IoT device representations are annotated with information about the OSAMAs that manage them, such as the *OSAMA communication modality*, using the *FOAF:Agent* class. This annotation allows OSAMAs to communicate with each other while exploring the DKG for failure root cause identification.

From a technical standpoint, the dependency artifact is integrated into the Orange Digital Twin platform *Thing in the future*(Thing'in)[18], which provides a set of APIs for OSAMAs to query the DKG when identifying failure root causes.

4.4 Monitoring and Recovery Artifacts

These artifacts embed monitoring and recovery functions of the legacy DM platform to allow *OSAMA* to monitor IoT devices, detect failures, and execute recovery actions. The monitoring artifact proactively sends failure events to its associated *OSAMA*. The recovery artifact allows *OSAMA* to execute recovery actions remotely, thanks to the remote management capabilities of legacy DM platforms. Note that we have chosen to reuse legacy DM platforms for monitoring and recovery as most of them provide such capabilities [39]. This could boost usability and save integration costs by avoiding the development of a solution from scratch, which consists in integrating heterogenous IoT devices through APIs to be accessed by *OSAMAs* for monitoring and recovery.

4.5 Collaborative CFM Protocol

Using the artifacts mentioned above, the *OSAMAs* collaborate with each other to solve cascading failure dilemmas according to a collaborative CFM protocol. We specialized the *OSAMA* into three different profiles, including *OSAMA-SP*, *OSAMA-DMP*, and *OSAMA-MN*, each having specified missions and artifacts according to their FM capabilities (see Sect. 2.1): *OSAMA-SP* and *OSAMA-DMP* are responsible for managing cascading failure requests. The first has full FM capabilities to manage failure on its devices. It collaborates with other *OSAMA-SPs* and *OSAMA-DMPs* for CFM. The latter manages failures collaboratively with *OSAMA-MNs* by requesting failure information owned by them.

[17] https://www.w3.org/TR/shacl-af/.
[18] https://tech2.thinginthefuture.com/.

Based on these profiles, the collaborative CFM protocol is described in Algorithm 1. The protocol is executed by *OSAMA-SP* and *OSAMA-DMP* when a failure event is reported on a device (line 2). *OSAMA* starts by updating the *belief base* in order to activate *failure plans* (line 3). Then, it requests device symptoms from the monitoring artifact (line 5). Next, depending on its profile, it either performs the diagnosis by itself (lines 6–8) or requests a diagnosis from other OSAMAs (lines 9–13) to get possible recovery actions. Next, it recovers the IoT device by executing the proposed compensatory action using the recovery artifact (line 14). If the device is still in a failed state (line 15), the *OSAMA* launches the plan for CFM: it queries the DKG of the failed device (line 17); for each device in the DKG, it requests cascading failure check from *OSAMA* managing it (line 18–21). Requested OSAMA deals with requests following the same algorithm by propagating on their turn the CFM request if they could not recover the failure. They respond when no more devices exist to explore (lines 27–29). After receiving all the responses, the *OSAMA* initiating the CFM request recovers the device (line 22) and notifies the customer care service if it is still in a failed state (lines 23–25). Then, it updates its *belief base* considering the device as recovered (line 26).

Algorithm 1. Collaborative CFM Protocol

```
 1: BEGIN
 2: if failure event evt = (device_i, sourceType, source) arrives then
 3:     Update the belief base Blf with predicate failed(device_i)
 4:     [Local Failure Plan]
 5:     S ← getDeviceSymptoms(device_i)
 6:     if Profile=SP then
 7:         recovery← diagnosis(device_i, S)
 8:     end if
 9:     if Profile=DMP then
10:         OSAMA_d← getDiagnosisAgent(device_i)
11:         requestDiagnosis(OSAMA_d, device_i, S)
12:         Wait for proposed recovery action recovery from OSAMA_d.
13:     end if
14:     recover(recovery, device_i)
15:     if getDeviceState(device_i) = failed then
16:         [Cascading Failure Plan]
17:         DKG ← getDependency(device_i)
18:         for (device_k, OSAMA_k) in DKG do
19:             sendCFMRequest(device_k, OSAMA_k)
20:             Wait for response OSAMA_k
21:         end for
22:         recover(recovery, device_i)
23:         if getDeviceState(device_i) = failed then
24:             Notify customer care service
25:         end if
26:         Update the belief base Blf with predicate recovered(device_i)
27:         if sourceType = OSAMA then
28:             responseCFMRequest(device_i, source)
29:         end if
30:     end if
31: end if
32: END
```

Let us illustrate the CFM protocol in Scenario 01 (see Table 4). In this scenario, a high variance failure is detected in the light bulbs by Amazon's OSAMA. To address this issue, Amazon OSAMA requests assistance from Philips OSAMA, the MN of the light bulbs, to diagnose and obtain recovery actions. Subsequently, Amazon OSAMA executes the proposed recovery plan; however, the light bulbs still report a high variance failure. The Amazon OSAMA assumes that the failure is due to a cascading failure and initiates

the CFM, which involves retrieving the DKG describing devices that the light bulbs depend on. The DKG refers to the alarm device, which is managed by Amazon OSAMA. Collaboratively, Amazon OSAMA and Philips OSAMA diagnose the alarm device and execute the recovery plan. However, the alarm device still reports a high variance failure. Amazon OSAMA continues the CFM approach by retrieving the DKG of the alarm device, which refers to the leak detector managed by Orange OSAMA. Amazon OSAMA requests assistance from Orange OSAMA, which collaboratively diagnoses the leak detector with the assistance of Kelvin OSAMA, the MN of the leak detector. After diagnosing and recovering the leak detector, Orange OSAMA notifies Amazon OSAMA, which notices that the alarm and light bulbs have returned to normal, successfully concluding the CFM request.

5 Evaluation

We implemented the *OSAMAs* with the *JaCaMo* framework (version 1.1)[19], which allows adaptable and scalable MAS management and coordination in complex environments [5]. Within the JaCaMo framework, the CFM protocol described in Algorithm 1 is implemented using the *Jason*[20] BDI technology allowing OSAMA agent to handle failure events in a parallel and coherent manner. The OSAMAs artifacts are implemented with the *Cartago*[21] technology that allows agents to access resources and services within their shared environment.

Reasoning in the diagnosis artifact is implemented with *Apache Jena* (version 3.4.0)[22]. To better represent a multi-actor deployment, we deployed the *OSAMAs* associated with the motivating use case in an Orange cloud infrastructure with the following resource: 1000 MIPS as CPU and 2GB as requested memory.

Based on these experimentation settings, we performed qualitative and quantitative evaluations of the proposed solution: The qualitative evaluation has been performed on the smart home use case presented in Sect. 3, by checking how our *OSAMAs* perform regarding cascading failure scenarios presented in Table 4. As described above, the smart home includes *17 IoT devices* managed by five DM actors and interconnected through *46 dependencies* described by the DKG. Each DM actor was associated with an OSAMA agent. The experiment has consisted in injecting failures in OSAMA agents' belief bases and letting them collaboratively perform CFM. Regarding the quantitative evaluation, it consisted of 1) measuring the completion time of the collaborative CFM protocol and 2) comparing our solution with the Orange legacy one based on simulated IoT infrastructures using our extension of the simulator *iFogSim* [18] that we refer to as *FMSim*. The result is discussed in the following.

Performance Evaluation. We evaluate the performance of the collaborative CFM protocol performed by the use case OSAMAs (see Table 2) in a cloud-based

[19] https://github.com/jacamo-lang/jacamo.
[20] https://jason.sourceforge.net/wp/.
[21] https://cartago.sourceforge.net/.
[22] https://github.com/apache/jena.

deployment. We measured the completion time of the collaborative CFM protocol on cascading failure scenarios involving devices with different dependency depths since this latter is the parameter that impacts the number of message exchanges between the OSAMAs during the collaborative CFM. We found that it takes, on average 5 s (see Fig. 5a), which we consider acceptable compared to the Orange legacy solution taking *from 15 to 20 min*.

Moreover, this performance can be enhanced more by reducing message exchange between *OSAMAs* using ML capabilities such as predicting the root cause of a cascading failure or offloading them to the edge to reduce latency. The DKG could also be deployed at the edge, as *Thing'in platform* allows this feature.

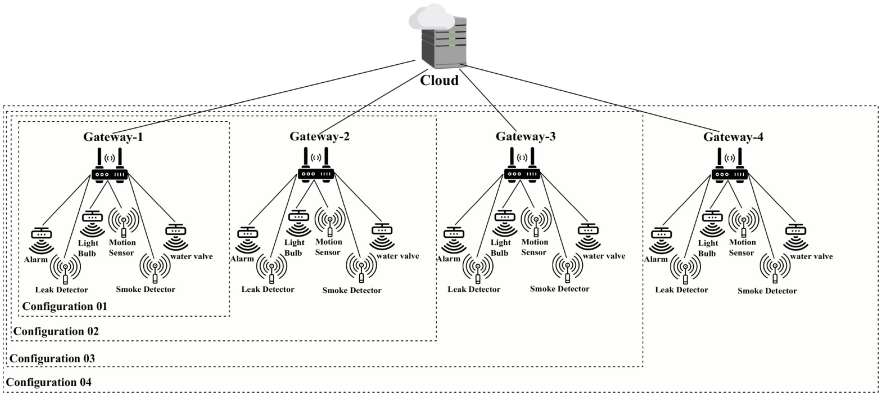
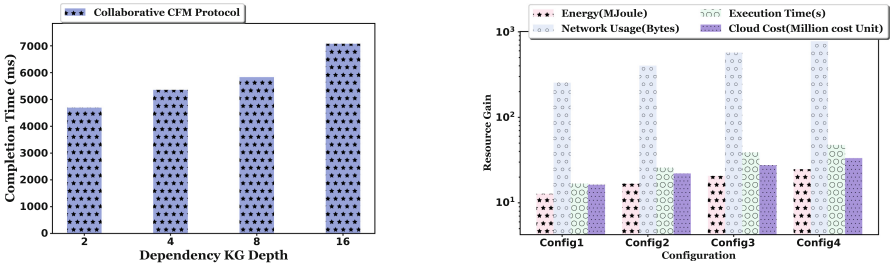


Fig. 4. Simulation Topology

Resource Consumption. Failures in IoT infrastructures can result in a significant loss of resources, as they make the infrastructure circulate useless and failed data and execute failed tasks. To show the impact of OSAMA in reducing such resource loss, we compared the resource consumption of IoT infrastructures managed by *OSAMAs* with those managed by the Orange legacy DM solutions using the simulator *FMSim*, our extension for the iFogSim simulator. The latter is a widely used Discrete Event Simulator for Fog and IoT because of its flexibility, scalability, and accessibility [30]. It uses a Sense-Process-Act model based on sensors, application modules, and actuators. Sensors send data to application modules deployed in Fog devices, which send actions as events to actuators according to a defined application logic. However, *iFogSim* does not support failure simulation on IoT devices. To this end, we developed *FMSim* [16], an extension for *iFogSim* allowing failure injection and recovery simulation on IoT devices. FMSim allows us to inject cascading failures (scenarios 1–6 described in Table 4) in simulated IoT infrastructures with different configurations (see Fig. 4), and measure resource consumption in the two cases: 1) The time to delete failure is set to OSAMA recovery time, and 2) The time to delete failure is set to legacy solution recovery time, represented by the average of failure recovery time of the Orange legacy solution taking *from 15 to 20 min*. Resource

consumption is represented by energy consumption, network usage, IoT application execution time, and the cost of executing IoT applications in the cloud. We report in Fig. 5b the relative resource gain achieved by our approach compared to legacy approaches deployed within the Orange organization. Specifically, in Configuration 4, we observed resource gains of *16 Mjoule*, *650 bytes* in terms of energy consumption and network usage respectively, which indicates that managing failures on IoT infrastructure using our solution instead of the legacy solution saves 16 Mjoule in energy consumption and 650 bytes in network usage. These gains can be attributed to the faster repair time achieved by *OSAMAs* compared to legacy solutions. As a result, *OSAMAs* enable swift recovery from resource-intensive failures, such as *High Variance*, thereby reducing resource loss in IoT infrastructure.



(a) CFM completion time as a function of the DKG depth (b) Resources gain of using *OSAMA* instead of legacy solution

Fig. 5. Experiment Results

6 Related Work

IoT failure management consists of three steps: failure detection, failure diagnosis for failure type and root cause identification, and failure recovery. IoT failure management is only partly treated in the literature, as most efforts focus on IoT failure detection [20,21,24], often notifying users to act themselves [28]. Few works have been proposed for IoT failure diagnosis. Early solutions propose a model-based approach, which consists of a mathematical model that describes the running behavior of devices [25]. Then, this model is used to estimate device output. The differences between the estimated and measured outputs are monitored to detect failures and determine their type [22]. Despite model-based approaches providing high accuracy models with low computation [3], they still need to be adapted for complex systems such as IoT [11]. To fill this gap, failure diagnosis techniques have evolved into data-driven approaches that analyze failure from signals or operational data using machine learning methods. These approaches leverage sensor device data to build models for fault identification and characterization [23]. Several algorithms were proposed for IoT fault diagnosis to detect and determine faulty devices and the type of fault [6,10,23,46].

However, the availability and quality of learning data of all possible fault types of large-scale systems such as IoT could be impossible [43].

The work [11] argues that knowledge-based failure diagnosis is especially well-suited for complex or multi-actor systems for which detailed mathematical models are unavailable, and diagnosis learning data are governed by different actors. These approaches rely on an FKB that contains failure diagnosis information provided by experts at the design or the operational level of devices. Experts-based fed of FKB ensures more accurate diagnosis results [12]. Several ontologies to generate FKB have been proposed for a range of assets such as Loaders [45], Cyber-physical system [2,34,35], Wireless Sensor Network [4], and Folio ontology for IoT systems [40]. To the best of our knowledge, the latter is the only proposed work to describe IoT failures. However, expressive classification for IoT failure behaviors, recovery actions, and failure symptoms is missing in this ontology. Moreover, the proposed approaches for IoT failure diagnosis do not address automatic failure root cause identification.

Regarding the failure recovery, some technical frameworks have been proposed in [26,28,29,31]. However, they rely mainly on device replication and replacement to recover from failure, which is costly and ineffective. Moreover, all existing solutions do not consider the practical reality: **IoT is managed by multiple actors using siloed and heterogenous DM platforms**, where devices and failure information are governed by different DM actors.

7 Conclusion and Future Work

In this paper, we presented our practical solution to help market DM actors address the dilemma of IoT cascading failures. It consists of a set of cooperative agents called OSAMAs, allowing siloed DM actors to manage cascading failures in an automatic and coordinated manner.

Our solution has shown how Semantic Web standards, e.g., ontologies and SHACL, unlock several challenges to solve the IoT cascading failure dilemma, such as efficiently storing, querying, and reasoning on abundant and heterogenous data related to IoT device dependencies and failures.

The evaluation results showed the significant impact and potential business savings of the proposed solution by reducing time to repair failures and minimizing resource waste in connected environments. We considered several design choices to ease and accelerate the adoption of the proposed solution by market DM actors, such as 1) the adoption of the BDI model that reflects human-like behavior, which eases the integration of the proposed solution by the DM actors, 2) the use of the FMEA model to design the IoT-F ontology, which has shown its usability in the literature based on a System Usability Scale (SUS) tests [13], and 3) the reuse of legacy platform features within the OSAMA agent for monitoring and recovery to save costs and accelerate integration efforts.

We have several plans for large-scale deployment and adoption of our solution in the device management market: The main plan consists in refining our solution based on further experimentation and user feedback within our device

management team in the Orange company, but also with Orange partners and relevant stakeholders to ensure the practical applicability and adoption of our approach, since it involves the collaborative effort of all device management actors. Another potential plan is to submit our solutions as a standard draft to standardization organizations in which Orange is involved, such as the European Telecommunications Standards Institute (ETSI) or the Connectivity Standards Alliance (CSA), to enable collaborative improvement and widest adoption of the proposed solution by several device management actors and experts.

Supplemental Material Availability. Source Code for IoT-F ontology [14], OSAMA agents [15], and FMSim simulator [16] is available from GitHub.

References

1. Aïssaoui, F., Berlemont, S., Douet, M., Mezghani, E.: A semantic model toward smart IoT device management. In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds.) Web, Artificial Intelligence and Network Applications, pp. 640–650. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44038-1_59
2. Ali, N., Hong, J.E.: Failure detection and prevention for cyber-physical systems using ontology-based knowledge base. Computers **7**(4) (2018). https://doi.org/10.3390/computers7040068. https://www.mdpi.com/2073-431X/7/4/68
3. Alsabilah, N., Rawat, D.B.: Anomaly detection in smart home networks using Kalman filter. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–6 (2021). https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484507
4. Benazzouz, Y., Keir Aktouf, O.E., Parissis, I.: A fault fuzzy-ontology for large scale fault-tolerant wireless sensor networks. Procedia Comput. Sci. **35**, 203–212 (2014). https://doi.org/10.1016/j.procs.2014.08.100. Knowledge-Based and Intelligent Information & Engineering Systems 18th Annual Conference, KES-2014 Gdynia, Poland, September 2014 Proceedings
5. Boissier, O., Bordini, R., Hubner, J., Ricci, A.: Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo. Intelligent Robotics and Autonomous Agents Series. MIT Press (2020)
6. Borhani, A., Zarandi, H.R.: Thingsdnd: IoT device failure detection and diagnosis for multi-user smart homes. In: 2022 18th European Dependable Computing Conference (EDCC), pp. 113–116 (2022). https://doi.org/10.1109/EDCC57035.2022.00028. Fault detection on sensors in smart home settings
7. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)
8. Celik, Z.B., Tan, G., Mcdaniel, P.: Iotguard: dynamic enforcement of security and safety policy in commodity IoT. In: Proceedings 2019 Network and Distributed System Security Symposium (2019)
9. Chakraborty, T., et al.: Fall-curve: a novel primitive for IoT fault detection and isolation. In: Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, SenSys 2018, pp. 95–107. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3274783.3274853
10. 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 **17**(1) (2017). https://doi.org/10.3390/s17010153

11. Chi, Y., Dong, Y., Wang, Z.J., Yu, F.R., Leung, V.C.M.: Knowledge-based fault diagnosis in industrial internet of things: a survey. IEEE Internet Things J. **9**(15), 12886–12900 (2022). https://doi.org/10.1109/JIOT.2022.3163606
12. Chi, Y., Wang, Z.J., Leung, V.C.M.: Distributed knowledge inference framework for intelligent fault diagnosis in IIoT systems. IEEE Trans. Netw. Sci. Eng. **9**(5), 3152–3165 (2022)
13. Emmanouilidis, C., Gregori, M., Al-Shdifat, A.: Context ontology development for connected maintenance services. IFAC-PapersOnLine **53**(2), 10923–10928 (2020). https://doi.org/10.1016/j.ifacol.2020.12.2833. 21st IFAC World Congress
14. Guittoum, A.: IoT-F ontology documentation. https://github.com/Orange-OpenSource/collaborativeDM-IoTF-ontology-documentation
15. Guittoum, A.: OSAMA agents in the Smart home use case. https://github.com/Orange-OpenSource/collaborativeDM-OSAMA-agent
16. Guittoum, A.: The FMSim simulator. https://github.com/Orange-OpenSource/collaborativeDM-FM-Simulator
17. Guittoum, A., et al.: Inferring threatening IoT dependencies using semantic digital twins toward collaborative IoT device management. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. SAC 2023. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3555776.3578573
18. Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments (2016)
19. Jia, Y., et al.: Who's in control? On security risks of disjointed IoT device management channels. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021, pp. 1289–1305. Association for Computing Machinery, New York (2021)
20. Kapitanova, K., Hoque, E., Stankovic, J.A., Whitehouse, K., Son, S.H.: Being smart about failures: assessing repairs in smart homes. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp 2012, pp. 51–60. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2370216.2370225
21. Kodeswaran, P., Kokku, R., Sen, S., Srivatsa, M.: Idea: a system for efficient failure management in smart IoT environments. In: MobiSys 2016 - Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services pp. 43–56 (2016). https://doi.org/10.1145/2906388.2906406
22. Lazarova-Molnar, S., Shaker, H.R., Mohamed, N., Jorgensen, B.N.: Fault detection and diagnosis for smart buildings: state of the art, trends and challenges. In: 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC), pp. 1–7 (2016). https://doi.org/10.1109/ICBDSC.2016.7460392
23. Li, J., Guo, Y., Wall, J., West, S.: Support vector machine based fault detection and diagnosis for HVAC systems. Int. J. Intell. Syst. Technol. Appl. **18**, 204 (2019)
24. Najari, N., Berlemont, S., Lefebvre, G., Duffner, S., Garcia, C.: Robust variational autoencoders and normalizing flows for unsupervised network anomaly detection. In: Barolli, L., Hussain, F., Enokido, T. (eds.) AINA 2022. LNNS, vol. 450, pp. 281–292. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99587-4_24
25. Najeh, H.: Diagnosis in building: new challenges. Theses, Université Grenoble Alpes; École nationale d'ingénieurs de Gabès (Tunisie) (2019)
26. Nishiguchi, Y., Yano, A., Ohtani, T., Matsukura, R., Kakuta, J.: IoT fault management platform with device virtualization. In: IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings 2018-January, pp. 257–262 (2018)

27. Norris, M., et al.: Iotrepair: systematically addressing device faults in commodity IoT. In: 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 142–148 (2020)
28. Norris, M., et al.: Iotrepair: flexible fault handling in diverse IoT deployments. ACM Trans. Internet Things **3**(3) (2022). https://doi.org/10.1145/3532194
29. Ozeer, U.I.Z.: Autonomic resilience of distributed IoT applications in the Fog. Theses, Université Grenoble Alpes (2019)
30. Perez Abreu, D., Velasquez, K., Curado, M., Monteiro, E.: A comparative analysis of simulators for the cloud to fog continuum. Simul. Model. Pract. Theory **101**, 102029 (2020)
31. Power, A.: A predictive fault-tolerance framework for IoT systems. Ph.D. thesis, Lancaster University (2020). https://doi.org/10.17635/lancaster/thesis/1063
32. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Van de Velde, W., Perram, J.W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0031845
33. Samsung: SmartThings rule. https://developer-preview.smartthings.com/docs/automations/rules/. Accessed 6 Avril 2022
34. Sanislav, T., Mois, G.: A dependability analysis model in the context of cyber-physical systems. In: 2017 18th International Carpathian Control Conference (ICCC), pp. 146–150 (2017)
35. Sanislav, T., Zeadally, S., Mois, G.D., Fouchal, H.: Reliability, failure detection and prevention in cyber-physical systems (CPSS) with agents. Concurr. Comput. Pract. Exp. **31**(24), e4481 (2019). https://doi.org/10.1002/cpe.4481
36. Sharma, A.B., Golubchik, L., Govindan, R.: Sensor faults: detection methods and prevalence in real-world datasets. ACM Trans. Sen. Netw. **6**(3) (2010)
37. Shibuya, M., Hasegawa, T., Yamaguchi, H.: A study on device management for IoT services with uncoordinated device operating history. In: ICN 2016, p. 84 (2016)
38. Silva, L.D., Meneguzzi, F., Logan, B.: BDI agent architectures: a survey. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pp. 4914–4921. International Joint Conferences on Artificial Intelligence Organization (2020)
39. Sinche, S., et al.: A survey of IoT management protocols and frameworks. IEEE Commun. Surv. Tutor. **22**(2), 1168–1190 (2020)
40. Steenwinckel, B., et al.: Towards adaptive anomaly detection and root cause analysis by automated extraction of knowledge from risk analyses. In: SSN@ISWC (2018)
41. Suárez-Figueroa, M.C., Gómez-Pérez, A., Fernández-López, M.: The NeOn methodology for ontology engineering. In: Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.) Ontology Engineering in a Networked World, pp. 9–34. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-24794-1_2
42. Carnegie Mellon University: Intelligent Software Agents. https://www.cs.cmu.edu/~softagents/multi.html
43. Wilhelm, Y., Reimann, P., Gauchel, W., Mitschang, B.: Overview on hybrid approaches to fault detection and diagnosis: combining data-driven, physics-based and knowledge-based models. Procedia CIRP **99**, 278–283 (2021). https://doi.org/10.1016/j.procir.2021.03.041. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15–17 July 2020
44. Xing, L.: Cascading failures in internet of things: review and perspectives on reliability and resilience. IEEE Internet Things J. **8**(1), 44–64 (2021)

45. Xu, F., Liu, X., Chen, W., Zhou, C., Cao, B.: Ontology-based method for fault diagnosis of loaders. Sensors **18**(3) (2018). https://doi.org/10.3390/s18030729
46. Zhang, H., Zhang, Q., Liu, J., Guo, H.: Fault detection and repairing for intelligent connected vehicles based on dynamic Bayesian network model. IEEE Internet Things J. **5**(4), 2431–2440 (2018)