Collaborative Cascading Failure Management on Internet of Thing Devices

Amal Guittoum

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : *MSTII - Mathématiques, Sciences et technologies de l'information, Informatique Spécialité : Informatique*
Unité de recherche : *Laboratoire d'Informatique de Grenoble et Orange Innovation*

Gestion Collaborative des Pannes en Cascade sur les Objets Connectés

Collaborative Cascading Failure Management on Internet of Things Devices

Présentée par :

Amal GUITTOUM

Direction de thèse :

Noel DE PALMA PROFESSEUR DES UNIVERSITES, Université Grenoble Alpes	Directeur de thèse
Sébastien BOLLE DIRECTEUR DE PROGRAMME DE RECHERCHE, Orange Innovation	Co-encadrant de thèse
Fabienne BOYER PROFESSEURE ASSOCIEE, Université Grenoble Alpes	Co-directrice de thèse
François AISSAOUI INGENIEUR DOCTEUR, Orange Innovation	Co-encadrant de thèse

Rapporteurs :

Fabien GANDON
DIRECTEUR DE RECHERCHE, Centre INRIA d' Université Côte d'Azur
Olivier BOISSIER
PROFESSEUR, École Nationale Supérieure des Mines de Saint-Étienne

Thèse soutenue publiquement le **23 février 2024**, devant le jury composé de :

Sihem AMER-YAHIA DIRECTRICE DE RECHERCHE, CNRS Délégation Alpes	Présidente
Noel DE PALMA PROFESSEUR DES UNIVERSITES, Université Grenoble Alpes	Directeur de thèse
Fabien GANDON DIRECTEUR DE RECHERCHE, Centre INRIA d' Université Côte d'Azur	Rapporteur
Olivier BOISSIER PROFESSEUR, École Nationale Supérieure des Mines de Saint-Étienne	Rapporteur
Thierry MONTEIL PROFESSEUR, INSA Toulouse	Examinateur
Nathalie HERNANDEZ PROFESSEURE DES UNIVERSITES, Université de Toulouse - Jean Jaurès	Examinatrice
Didier DONSEZ PROFESSEUR DES UNIVERSITES, Université Grenoble Alpes	Examinateur

إلى زوجي وحبيبي أحمد حلقوم

إلى أمي وأبي الغاليين

Acknowledgements

For accepting to review my thesis, I would like to thank *Fabien Gandon* and *Olivier Boissier*. I would also like to thank *Sihem Amer-Yahia*, *Nathalie Hernandez*, *Didier Donsez*, and *Thierry Monteil* who accepted to be on my jury.

I would like to express my deepest gratitude to my advisors, Fabienne, François, Noel, and Sébastien for their unwavering support, guidance, and invaluable insights throughout the entire journey of my doctoral research. Their mentorship has been instrumental in shaping the direction and quality of this thesis. More precisely, I would like to thank Fabienne for making me learn how to write a good research paper and for the time we spent together at her desk discussing our papers, François for transferring to me his deep thinking, ideas, and technical skills, Noel's availability and advice, despite his significant responsibilities, have been truly appreciated, Sébastien who was the "hard work model" for me, his large and transversal research vision has contributed a lot to my doctoral research.

I am also thankful for the encouragement and constructive feedback provided by my team at Orange Innovation whose expertise has enriched the content and rigor of my work. I extend my appreciation to the Orange company for providing a stimulating research and development environment and resources essential for the completion of this research.

I am grateful to my colleagues and peers who have contributed to this work. Many thanks to *Matthieu*, an Orange expert, who made me learn about the USP protocol, and to *Julien*, a researcher at Orange, with whom I worked on the *Collaborative LAN troubleshooting demonstration*. I would like to thank also *Sybille* and *Melissa*, responsible for the DOMUS testbed, for their precious contributions.

My heartfelt thanks go to my best friend *Yasmine*, and all my family, my mother *Fadila*, my sisters *Amina*, *Aya*, *Khadidja*, my brother *Hassan* for their unwavering support, love, and understanding during the challenging phases of this doctoral journey. Their encouragement has been my pillar of strength. Special thanks to my father *Moussa*, who is my main source of courage, and to **my husband *Sir Ahmed*** who has been a constant source of love and support.

Lastly, I extend my gratitude to all those who, directly or indirectly, have been a part of this academic endeavor. This thesis stands as a collective achievement, and I am deeply appreciative of the collaborative spirit that has fueled its completion.

Contents

List of Figures

List of Tables

List of Terms and Abbreviation

API *Application Programming Interfaces*

BDI *Belief Desire Intention*

CFM *Cascading Failure Management*

DKG *IoT Dependency Knowledge Graph*

DM *Device Management*

DMP *DM Platform Provider*

DT *Digital Twin*

ER *Entity Resolution*

FKB *Failure Knowledge Base*

FMEA *Failure Mode Effect Analysis*

IoT *Internet of Things*

IoT-D *Internet of Things Dependency*

IoT-F *Internet of Things Failure*

KG *Knowledge Graph*

LwM2M *Lightweight M2M*

MAS *Multi-Agent System*

MN *Device Manufacturers*

OSAMA *cOllaborative caScading fAilure Management Agent*

OWL *Web Ontology Language*

QoE *Quality of Experience*

RDF *Resource Description Framework*

SHACL *Shapes Constraint Language*

SP *Service Provider*

SPARQL *SPARQL Protocol and RDF Query Language*

SW *Semantic Web*

TD *Thing Description*

Thing in *Thing In The Future*

USP *User Services Platform*

W3C *World Wide Web Consortium*

Chapter 1

Introduction

Summary

This chapter introduces the Thesis by first defining its context and scope. Then, it highlights the addressed challenges through a Smart Home use case. After that, it provides a global overview of the proposed solution and the thesis contributions.

Contents

1 The management of the Internet of Things devices

The *Internet of Things* (IoT) is increasingly becoming a transformative technological force by reshaping the very fabric of our modern world, enabling various industries and sectors, each with its unique set of applications and benefits. For instance, in smart homes, IoT is revolutionizing how we interact with and manage our living spaces. IoT devices, from thermostats to smart speakers, communicate with each other to create an environment that responds to our needs and preferences. For instance, smart thermostats can learn our temperature preferences and adjust accordingly, not only enhancing comfort but also saving energy. For the same purpose,

IoT is enabling smart grid technologies to foster energy saving and management. These grids incorporate sensors, communication networks, and advanced analytics to monitor and manage the flow of electricity more efficiently. Smart grids can detect and respond to power outages, balance supply and demand in real-time, and integrate renewable energy sources.

One of the pivotal elements for creating substantial value within IoT ecosystem resides in the capability of IoT devices to autonomously perform tasks with minimal human intervention. This ability to collect, process, and transmit data, make decisions, and interact with other devices and systems is at the heart of what makes IoT transformative. However, to ensure the correct and efficient operation of IoT systems, it is imperative to have robust systems in place for the monitoring and management of these devices. This is referred to as *Device Management* (DM).

DM constitutes a set of operations and processes essential to ensure the well-functioning of IoT devices. These operations are executed remotely on IoT devices through DM solutions and platforms, which allow for efficient maintenance of IoT devices' health, security, and longevity. Among DM operations, we find device firmware [1] updates, device reboot and reset, and device monitoring and failure management.

Indeed, Regular firmware updates are essential for keeping IoT devices up-to-date with the latest features, security patches, and bug fixes. These updates ensure that devices can adapt to evolving standards and protocols. Moreover, devices must be equipped with the latest security measures to protect against vulnerabilities and cyber threats. Regarding device reboot and reset, IoT devices may encounter issues or performance degradation that can be resolved through a simple reboot, where DM solutions can remotely initiate reboots to restore normal operation. A device may require a factory reset to return to its default configuration in more complex cases. This can be especially important when devices need to be decommissioned i.e., removed from the IoT system.

While firmware updates, device reboots, and resets are important DM operations, the scope of this discipline extends far beyond these elementary actions. Continuous monitoring of device health is an essential DM feature. This includes tracking factors like temperature, battery life, network connectivity, and sensor accuracy. It may be enhanced with an alert system or sophisticated data analytics features to detect and address IoT device failures, ensuring they remain in their correct state. After the detection of failures through device monitoring, DM solutions may perform failure diagnosis and recovery remotely through DM operations. We refer to this process starting from monitoring and failure detection to failure recovery as *IoT failure management*.

In current IoT systems, the DM features are often provided through separate and isolated DM platforms and solutions, each of which is administered by different stakeholders. These stakeholders can include operators, service providers, and device manufacturers [Aïssaoui, 2020; Jia, 2021; Shibuya, 2016].

DM platforms, such as *Amazon Web Service*[2], *LiveObjects*[3], are provided by DM platform providers to offer DM as a service, providing users and IoT suppliers with essential DM features

[1] Firmware is a microcode or software that is embedded into the memory of IoT devices to help them operate.
[2] https://aws.amazon.com/fr/iot-device-management/
[3] https://liveobjects.orange-business.com/

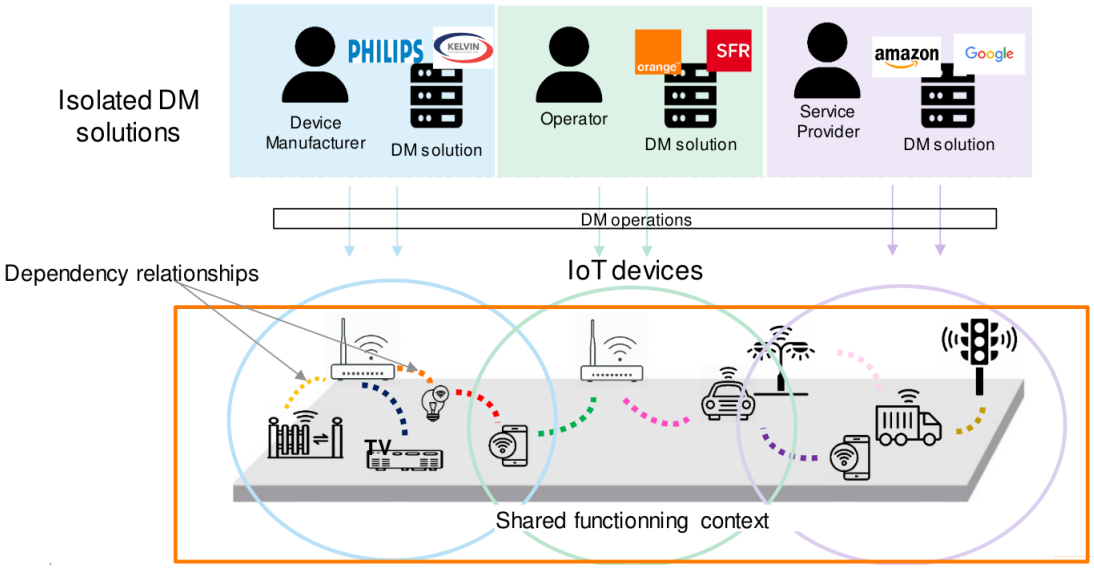


Figure 1.1: Siloed Management of Interdependent IoT devices.

to maintain and optimize their interdependent IoT devices. To achieve this, these platforms offer integration capabilities, allowing them to work with IoT devices from various device manufacturers. This integration is made possible through dedicated *Application Programming Interfaces* (API) that ease communication between the DM platform and heterogeneous IoT devices.

Moreover, Device Manufacturers often contribute to the DM ecosystem by developing mobile applications that serve as user interfaces, enabling consumers to configure, monitor, and control their IoT devices, such as *Hue app* [4] proposed by *Philips*. These interactions can take place through local connections or cloud-based services, offering users flexibility and convenience in managing their devices. On the other hand service providers 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 service provider proposes its own proprietary DM platform for managing its devices.

This results in a complex landscape where IoT devices, each potentially manufactured by a different manufacturer, are managed and controlled by separate DM solutions and platforms. This reveals the following practical reality (see Figure 1.1): *Interdependent IoT devices are managed by siloed DM solutions and platforms governed by different actors.*

2 Problem Statement

These siloed DM solutions exhibit several challenges when managing interdependent IoT devices in an uncoordinated and chaotic manner [Jia, 2021]. To address these challenges, we believe that there are needs for the cultivation of collaboration and interoperability functionalities within the confines of these isolated DM solutions. These capabilities allow siloed DM solutions to work together to solve complex problems concerning their multi-actor governance. We refer to this

[4] https://www.philips-hue.com/en-us/explore-hue/apps

paradigm of collaborating DM solutions as *Collaborative DM*.

In this section, we illustrate challenges facing siloed DM solutions through a Smart Home use case including a set of IoT devices managed by multiple DM solutions governed by different actors, in order to demonstrate the need for the collaborative DM. Then, we focus on one of these challenges, namely *Cascading Failure Management* (CFM), which is the challenge addressed by this Thesis due its valuable business impact in practice.

2.1 Smart Home Use Case

We consider a smart home managed by five DM actors (see Figure 1.2). It will be used to demonstrate the need for collaborative DM. Moreover, it will be used in the rest of the thesis for illustration purposes. It consists of three intelligent systems deployed in a home consisting of a living room and a kitchen:

- **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.
- **Temperature management system** Controls the home temperature using a *temperature sensor* and an *airconditioner*. It is mainly based on automation rules[5] 1–3 described in Table 1.2.
- **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 provided by Orange connects devices in the living room to the Internet, while an Orange Wi-Fi repeater connects the kitchen devices. The SmartThings platform[6] enables automation rules described in Table 1.2 using a *SmartThings Hub* [7]. 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 1.1).

2.2 On the need of Collaborative DM

The siloed DM actors described in the use case are exposed to new threats due to interdependencies among IoT devices. These interdependencies are related to data and service exchange among IoT devices, which may be generated through for example automation rules. For instance, rule 2 (see Table 1.2) makes the windows dependent on the air-conditioner as it acts on the firsts based on the state of the second. These dependencies are complex, abundant, and generate various threats that current DM solutions can not face with their siloed capabilities.

[5] Automation rules allow the automated composition of IoT services in a connected environment.
[6] SmartThings is Samsung's IoT platform that enables automation rules across IoT devices in Smart Homes.
[7] `https://www.samsung.com/us/smart-home/smartthings/hubs/samsung-smartthings-hub-f-hub-us-2/`

Figure 1.2: Smart home architecture

Table 1.1: DM actors managing the Smart Home

DM actors	Profile	Managed devices
Orange	DM Provider Service Provider	Leak detector, airconditioner, water valve, temperature sensor, windows, door, Gateway, Wi-Fi repeater
Samsung	Service Provider	SmartThings Hub
Amazon	DM Provider	Alarm, lights bulb, smoke sensor, light control unit, vocal assistant
Philips	Device Manufacturer	Motion sensor, light bulbs, light control unit windows, door, alarm
Kelvin	Device Manufacturer	Temperature sensor, airconditioner, leak detector, water valve

One such threat is the phenomenon of *cascading failures*, where the failure of one device triggers a cascade of undesired state changes in devices that depend on it [Xing, 2021]. Root cause identification and failure recovery in such a scenario are complex since devices are managed by siloed DM solutions unaware of interdependencies between devices. Let us take the scenario where the airconditioner in the living room fails. This failure propagates to its dependent devices and services: 1) the vocal assistant cannot respond to the prompt "Turn off the airconditioner", and 2) the SmartThings hub can no longer close the windows when the airconditioner is deactivated and open it when the temperature exceeds a threshold (see Rules 2 and 3 in Table 1.2). These failed devices are managed by different DM actors: Orange manages

Table 1.2: 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 a 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.

the airconditioner, which is built by Kelvin, Amazon manages the vocal assistant, and Samsung manages the SmartThings hub. These siloed DM actors cannot identify the failure's root cause since they do not have knowledge of dependency relationships among IoT devices.

Another threat is failures during the execution of DM operations, e.g., *reboot* or *firmware update*, on interdependent devices. Indeed, DM operations can cause devices to be temporarily unable to provide their services, leading to failures on dependent devices and breaking updates [Mezghani, 2020; Zdankin, 2021]. These failures are exacerbated by the uncoordinated and parallel execution of DM operations using multiple DM solutions [Jia, 2021] and are usually hard to revert [Zdankin, 2021]. Let's suppose Orange reboots the gateway while Amazon updates the vocal assistant. Due to the connectivity dependency, the gateway reboot interrupts the vocal assistant's Internet connection, which results in the firmware image not being downloaded correctly and the vocal assistant failing [Mezghani, 2020].

In addition, siloed DM solutions suffer from service reconfiguration issues. Namely, they find it difficult to automatically integrate new devices within multi-actor IoT services. Let's assume the user wants to install a new light bulb in the smart home to be controlled through the vocal assistant. Considering current DM capabilities, the light bulb control can no longer be installed automatically on the vocal assistant, since they are managed by the different DM actors: Orange and Amazon.

The Figure 1.3 summarizes the motivations behind the collaborative DM in the form of three (03) use cases, namely, CFM, DM failure management, and service reconfiguration. In our work, we leverage the collaborative DM paradigm to conceive a collaborative CFM solution on IoT devices managed by siloed DM actors. However, our proposed solution is generic enough to be extended to cover other collaborative DM use cases. In the following, we detail the CFM use case.

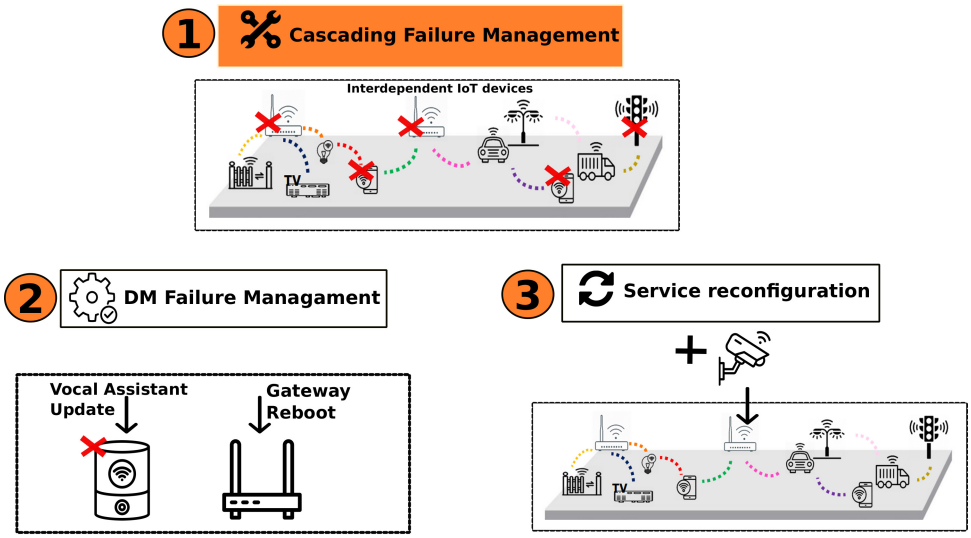


Figure 1.3: Collaborative DM use cases.

2.3 Cascading Failure Management

As mentioned above, *Cascading Failures* is a serious dilemma facing market DM actors. They arise when the failure of one device instigates the failure of dependent devices and applications [Xing, 2021]. They 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 [Najeh, 2019]. 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.

Managing cascading failure on interdependent devices managed by siloed DM actors needs to consider the following challenges:

- *CH01: DM solution heterogeneity*
  Legacy DM solutions are heterogeneous, each using its own protocol and data model. Thus, building a collaborative solution across these heterogeneous DM solutions must integrate an interoperability layer allowing them to understand each other and work together toward CFM.

- *CH02: Distributed governance of data and control*
  IoT devices are managed by siloed DM solutions each having partial control and data of IoT devices. A global governance strategy must be considered to allow information sharing and collaborative control across IoT devices.

- *CH03: Recognition of the global dependency topology*
  The first step towards managing cascading failures is the recognition of dependency re-

lationships among IoT devices. This allows the identification of the failure's root cause. However, this task is challenging since IoT dependency is abundant, dynamic, and governed by different actors, i.e., the data describing IoT dependencies is distributed across siloed DM solutions managed by different actors. For instance, in the described smart home use case, automation rules are governed by Samsung while connectivity dependencies are governed by Orange.

- *CH04: Complexity of the IoT system*
  IoT devices are abundant and more and more interdependent in exchanging data and services. Moreover, they are heterogeneous, leading to heterogeneity in their failure behavior and information. Thus, the CFM solution needs to be scalable, and able to handle heterogeneous IoT failure information.

Considering these challenges, we propose automatic and collaborative solutions for managing cascading failure on interdependent IoT devices managed by siloed DM solutions. In the following, we discuss the proposed solutions and highlight its contributions.

3 Global Overview of the proposed solutions

This research work proposes an automatic and collaborative CFM approach for IoT devices managed by siloed DM solutions. It relies on two main systems, namely *A Semantic Digital Twin for IoT dependency inference* and *A Semantic Multi-agent System for automatic and collaborative CFM* (see Figure 1.4).

3.1 A Semantic Digital Twin for IoT dependency Inference

We propose a framework that allows DM actors to collaboratively infer dependency relationships among IoT devices that we refer to as *IoT dependency Topology*. The framework accesses heterogeneous dependencies data from siloed DM solutions and aggregates them using the Semantic *Digital Twin* (DT) technology, which refers to a virtual and synchronized representation of real-world entities and processes built using a standardized vocabulary called *Ontology*. More precisely, the dependency relationships are represented as a global *IoT Dependency Knowledge Graph* (DKG) served as a DT view, representing the current devices and their dependencies.

Conceptually, the proposed framework relies on an ontology called *Internet of Things Dependency* (IoT-D) that enables a shared representation of IoT dependencies across heterogeneous DM solutions. The IoT-D ontology describes a set of contextual data that delineates dependencies among devices. By leveraging the IoT-D ontology, the framework automatically constructs the global DKG through a three-step process namely *Context extraction*, *Entity resolution*, and *Dependency inference*. The first step extracts the context data from legacy DM solutions and transforms it into KGs, the second aggregates the extracted context KGs, and the last infers the DKG from the aggregated context KGs.

The proposed framework relies on the Orange DT platform *Thing In The Future* (Thing

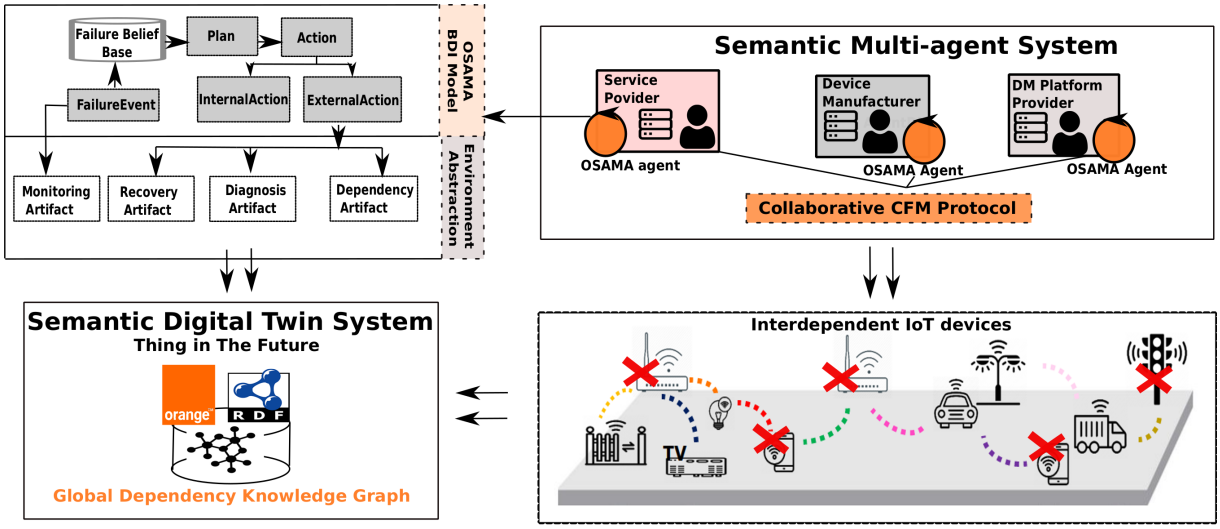


Figure 1.4: A global overview of the proposed solution.

in) [8] and is designed to be integrated into customer care applications of DM actors as a human-based decision support tool to help efficient management of dependencies-related threats. More specifically, it can be used in various business scenarios, such as identifying the root cause of a cascading failure, and dependency-aware planning of DM operations, e.g., *firmware update* to prevent DM failures.

3.2 A Semantic Multi-agent System for automatic and collaborative CFM

To go beyond the human-based analysis of the IoT dependency topology, we propose a practical solution allowing siloed DM actors to manage cascading failures in an automatic and coordinated manner, by relying on the developed framework for IoT dependency inference. This solution consists of a cooperative *Multi-Agent System* (MAS), which 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. 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 CFM.

OSAMA exploits a set of *Semantic Web* (SW) standards, such as ontologies, in order to simplify failure information exchange and enhance the interoperability among siloed DM platforms. It leverages the Semantic DT technology, 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. They adopt a *Belief Desire Intention* (BDI) model to handle effectively and smartly cascading failure events 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: 1) *Monitoring Artifact*: Allows

[8] https://www.thinginthefuture.com/

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 structured according to an ontology called *Internet of Things Failure* (IoT-F); 3) *Dependency Artifact*: Thanks to Semantic DT, 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.

4 Contributions

This work includes research and experimental contributions. The latter is guided by the industrial context of our work related to the Orange company.

4.1 Research Contributions

- The IoT-D ontology that provides context-based modeling for IoT dependencies in the form of a KG. This ontology allows DM actors to have a common and shared understanding of dependency relationships between IoT devices.
- A KG entity resolution approach using rules and functions from the advanced features of *Shapes Constraint Language* (SHACL)[9] for aggregating dependency information extracted from siloed DM solutions.
- A rule-based approach to infer the topology of IoT dependency relationships, relying on the SHACL standard.
- The IoT-F ontology, which describes IoT failure information such as device failures and their recovery actions. This ontology allows DM actors to have a common and shared understanding of IoT failure information.
- The OSAMA agent leveraging the BDI model to enable collaborative CFM across legacy DM solutions.
- A collaborative CFM protocol allowing the OSAMA agents to collaborate toward CFM.

4.2 Experimental Contributions

- A proof of concept for the IoT dependency inference framework, integrated into the Orange DT platform Thing in, which was validated on simulated Smart Home scenarios and the realistic testbed *DOMUS* [10].
- A proof of concept for the semantic MAS solution with Cloud-based deployment, demonstrating its potential impact in reducing time to repair failure, and minimizing resource consumption in IoT infrastructures, such as energy consumption.

[9] https://www.w3.org/TR/shacl-af/
[10] https://www.liglab.fr/fr/recherche/plateformes/domus

- The *FMSim* simulator, a simulator we developed in the context of this work, extending the state of the art simulators for IoT failures injection and recovery simulation. This simulator allowed us to evaluate our proposed solution for CFM.

- The *Collaborative LAN Troubleshooting Demonstration*, an innovative demonstration developed in collaboration with other teams at Orange Innovation highlighting the benefit of our work in enhancing the customer care services of the Orange company and potentially all market DM actors.

The present research work was carried out in the *FLAM* team of Orange, Meylan in collaboration with the *ERODS* research team of the LIG lab. It led to the following research publications:

- Amal Guittoum, François Aïssaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "A Semantic Framework for IoT dependency Inference using Semantic Digital Twins". COMPAS 2022: Conférence francophone d'informatique en Parallélisme Architecture et Système (Compas), MIS - Laboratoire Modélisation, Informatique et Système - de l'Université de Picardie Jules Verne., Jul 2022, Amiens, France.

- Amal Guittoum, Francois Assaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Inferring Threatening IoT Dependencies Using Semantic Digital Twins Toward Collaborative IoT Device Management". Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. SAC 23. Tallinn, Estonia: Association for Computing Machinery, 2023, pp. 1732–1741

- Amal Guittoum, François Assaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Leveraging Semantic Technologies for Collaborative Inference of Threatening IoT Dependencies". SIGAPP Appl. Comput. Rev. 23.3 (2023), pp. 32–48

- Amal Guittoum, François Assaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Solving the IoT Cascading Failure Dilemma Using a Semantic Multi-agent System". The Semantic Web – ISWC 2023. Lecture Notes in Computer Science, vol 14266.

Moreover, our work has been presented in the Chair *MIAI Edge Intelligence* [11], *Eclipse IoT Days* [12], *Orange Silicon Valley*, and awarded in the contest *Ma thèse en trois Minutes* (see the final pitch[13]). Furthermore, it has been published in a general public article within *The Conversation France* journal [14].

5 Thesis Structure

The rest of this Thesis is organized into seven (07) chapters, with two (02) Appendix:

[11] https://edge-intelligence.imag.fr/
[12] https://wiki.eclipse.org/Eclipse_IoT_Day_Grenoble_2023
[13] https://mastermedia.orange.com/publicMedia?t=pmqNUEy1Hy
[14] https://theconversation.com/objets-connectes-quand-les-pannes-en-cascade-se-propagent-dun-objet-a-lautre-203713

Chapter 02: Background

The chapter 2 introduces the industrial and scientific context of our work. It highlights the motivation of Orange behind the present research. It introduces the different concepts used in our method namely SW standards, DT, and MAS systems. It presents applications of those concepts and the benefits of their combination, to justify our technical choices. Moreover, this chapter presents the different technologies involved in realizing our proposed methods and proof of concepts such as Thing in and *JaCaMo*.

Chapter 03: State of The Art

The chapter 3 presents the state of the art of the different research axes to which our work has contributed. Namely, *IoT dependency extraction and modeling*, *Failure management*, *Ontologies for IoT*. We discuss research efforts on these axes and identify research gaps.

Chapter 04: Inferring Threatening IoT dependencies using Semantic Digital Twins

The chapter 4 presents our contributions proposed for IoT dependency inference. We shed light on our modeling for IoT dependencies and highlight essential technical steps allowing for the extraction and inference of IoT dependencies collaboratively and automatically.

Chapter 05: Solving the IoT cascading failure dilemma using Semantic Multi-agent systems

The chapter 5 shows our contributions to CFM across siloed DM platforms and solutions. We present how the framework IoT dependency inference, presented in the previous chapter, is extended using a MAS allowing for automatic and collaborative cascading failure diagnosis and recovery.

Chapter 06: Collaborative LAN Troubleshooting Demonstration

The chapter 6 sheds light on the concrete demonstration of our work, the *Collaborative LAN Troubleshooting Demonstration*. We present its technical architecture and practical user stories highlighting the outcomes of our current research within an industrial context at Orange.

Chapter 07: Conclusion

The chapter 7 concludes this work by summarizing our main contributions and discussing several perspectives for enhancing and preparing the road for the large adoption of our work in the DM market.

Appendix A: FMSim, IoT Failure Simulator

The Appendix A describes an experimental contribution of the thesis, which consists of the FMSim simulator for IoT failure injection and recovery simulation.

Appendix B: Correctness verification of the Collaborative CFM protocol

The Appendix B presents a correctness verification study of the proposed CFM protocol through an experimental approach allowing the verification of its behavior regarding a set of challenging scenarios such as *request deadlock*.

Chapter 2

Background

Summary

This chapter introduces the industrial and scientific context of our work. It introduces the different concepts used in our method namely *Semantic Web* (SW) standards, *Digital Twin* (DT), and *Multi-Agent System* (MAS). We highlight applications of those technologies and the benefits of their combination, to justify our technical choices. Moreover, this chapter presents the different technologies involved in realizing our proposed methods and proof of concepts such as *Thing In The Future* (Thing in) and *JaCaMo*.

Contents

As mentioned in the introduction, the main goal of this thesis is to manage cascading failure on interdependent *Internet of Things* (IoT) devices managed by different actors and siloed *Device Management* (DM) platforms. The achievement of such a goal has started by building knowledge on IoT and DM considering industry and research insights, which will be presented in the first part of this chapter. The second part describes our research's scientific context, including concepts and technologies that we rely on in our methods: *Semantic Web (SW) technologies, Multi-Agent System (MAS), and Digital Twin (DT)*. Moreover, we discuss their current application and combination to justify our technical choices.

1 Industrial Context

1.1 Internet of Things

For over two decades, the IoT has transformed industries and led digital transformation in various domains. The term Internet of Things was introduced in 1999 by the British technology pioneer *Kevin Ashton* to describe the connection of the physical world objects to the Internet through sensors [Keith D Foote, 2022]. There is, however, no single, universal definition of the IoT. The ITU-T defines the IoT as: "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [ITU-T Y2060, 2012]. Authors in [Rose, 2015] argue that IoT encompasses situations in which objects, sensors, and ordinary items that aren't typically seen as computers are empowered with network connectivity and computing capabilities. This enables these devices to autonomously produce, share, and utilize data without human intervention.

IoT has become increasingly widespread in everyday life, demonstrating potential impact and large adoption, thanks to continuous technological developments and considerable investments. The widely adopted application of IoT is Smart Homes, where IoT devices control and automate various aspects of a home, such as lighting, thermostats, security cameras, appliances, and entertainment systems, providing increased convenience and energy efficiency. However, it is also adopted in other domains including but not limited to: manufacturing and industrial settings, enabling the monitoring and optimizing of equipment and processes, leading to improved efficiency, predictive maintenance, and better resource management; healthcare, where IoT devices play a crucial role in remote patient monitoring, wearable health trackers, and connected medical devices, allowing for real-time health data collection and personalized care; and transportation, enabling better traffic management, autonomous vehicles, and improved public transportation services.

Several companies and research organizations argue on the potential impact and large adoption of IoT: Huawei forecasts 100 billion IoT connections by 2025 [HuaweiTechnologies, 2018]; McKinsey Global Institute predicts the financial impact of IoT may be as much as 3.9$ to 11.1$ trillion by 2025 [Manyika, 2015]; The Orange company deploys and manages today 21.1 million connected objects for risk prevention, comfort, and resource optimization [OrangeBuisness, 2023].

In the following, we present a global overview of the IoT architecture and technologies and highlight IoT challenges.

1.1.1 Architecture

Currently, there is a lack of a unified reference architecture for IoT, and creating one is proving extremely complex, despite numerous standardization efforts being made [Lombardi, 2021]. The most common architecture (see Figure 2.1) includes three components represented by IoT devices, connectivity, and IoT platforms [Celik, 2019]. A centralized gateway often links devices within a physical environment. This gateway utilizes cloud services to synchronize device states,

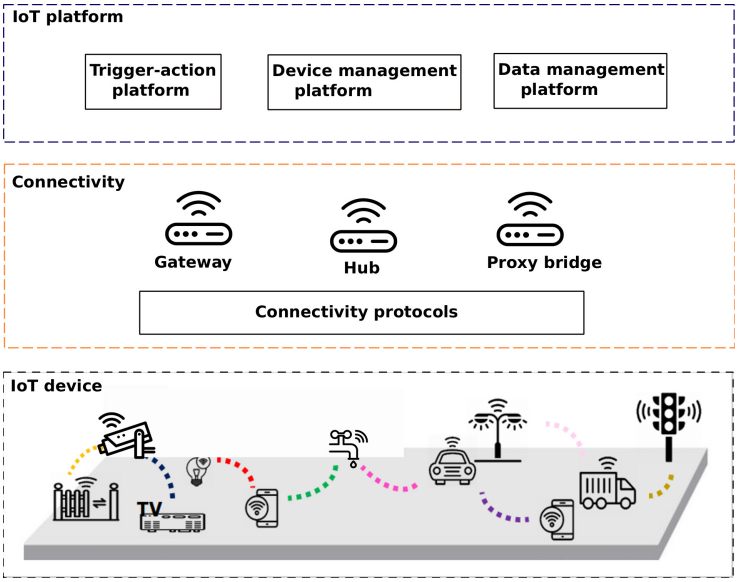


Figure 2.1: IoT architecture

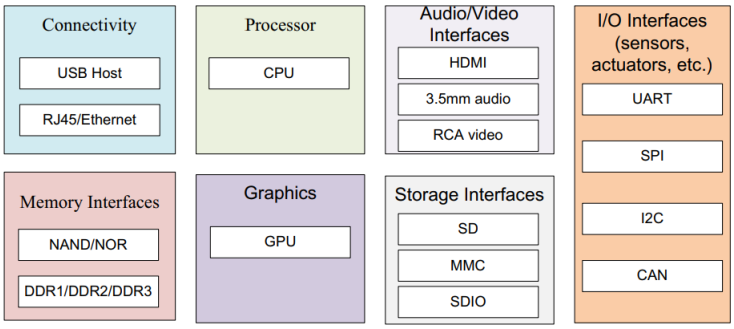


Figure 2.2: Generic architecture of an IoT device [Ray, 2018]

process IoT data, and offer remote control and monitoring interfaces.

- IoT devices: consist of sensors and actuators that enable interaction with a physical environment. Sensors detect physical events and collect and send data to other IoT devices, the gateway, or IoT platforms to be processed and used to actuate the devices. For example, a presence sensor detects a presence event and interacts with the connected light switch (actuator) that turns on the lights [Celik, 2019]. From an architectural point of view (see Figure 2.2), an IoT device may consist of several interfaces to ensure the storage, data processing, network connectivity, and communication to other devices [Ray, 2018].

- Connectivity: represented by a set of protocols and technologies that allow IoT devices to communicate with each other and to network devices. Several low-power connectivity technologies have been proposed in the IoT market (see Figure 2.3) such as *Zigbee* [1],

[1] https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

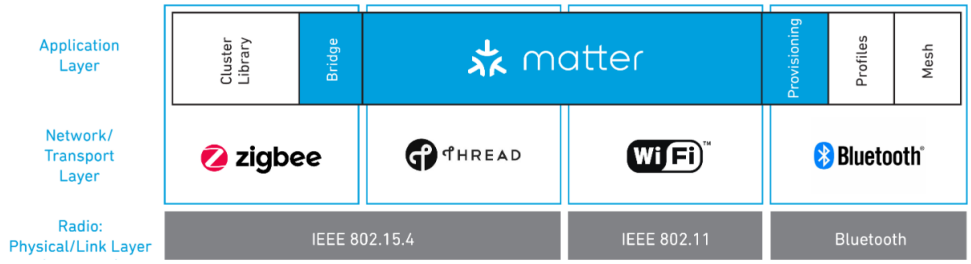


Figure 2.3: IoT Connectivity protocols From [Qorvo, 2021]

Zwave [2] and *Thread* [3] to connect IoT devices in smart Home, and *LoRaWAN* that enable IoT devices to communicate over significant distances such as in smart cities. Recently, the Connectivity Standards Alliance and other major IT companies, including Apple, Google, and Amazon, launched the connectivity standard *Matter*[4], which enhances the previously mentioned protocols to allow communication between IoT devices proposed by different vendors. Matter is based on existing IP protocols such as TCP/IP, HTTP, and TLS, and uses a common data model to ensure interoperability between devices. The structure of this data model (see Figure 2.4) describes an IoT device by a set of *Nodes*, each node describes the global functionality of the device and is composed of a set of *Endpoints* that represent a specified functionality of a device such as dimmable light. Each endpoint is described by a set of *Clusters* that describe an elementary functionality of the endpoint such as the on/off function for the dimmable light. We find in each cluster a set of *Attributes* that represent proprieties of the cluster and a set of *Commands* and *Events* to control the function described by the cluster. The specifications of Matter include several clusters such as *Network Cluster* defines how devices discover and join a Matter network, and the cluster dedicated to IoT device diagnosis is named *Diagnostics Cluster*. It describes a set of IoT device failure causes and monitoring indicators, to ease IoT failure management.

- IoT platforms are pivotal in delivering application-specific services to end users by effectively managing IoT devices and facilitating their interactions. These platforms have essential functions, including IoT data collection and analytics, ensuring interoperability among devices, and device management. They extend across diverse domains such as healthcare, transportation, and agriculture, empowering organizations and individuals to access innovative services tailored to their needs. One widely adopted type of IoT platform is the *trigger-action platform*, empowered by the *end user programming paradigm*, enabling end users to build intelligent services by connecting their devices through automation rules and scenes. These rules follow a straightforward format: *"if trigger, then action,"* where a predefined set of actions is executed when a specific event or trigger occurs. For instance, envision a user setting up an automation rule: "If the presence sensor detects movement, then turn on the light bulb". Several trigger-action platforms were proposed in the IoT

[2] https://www.z-wave.com/
[3] https://www.threadgroup.org/What-is-Thread/Thread-Benefits
[4] https://csa-iot.org/all-solutions/matter/

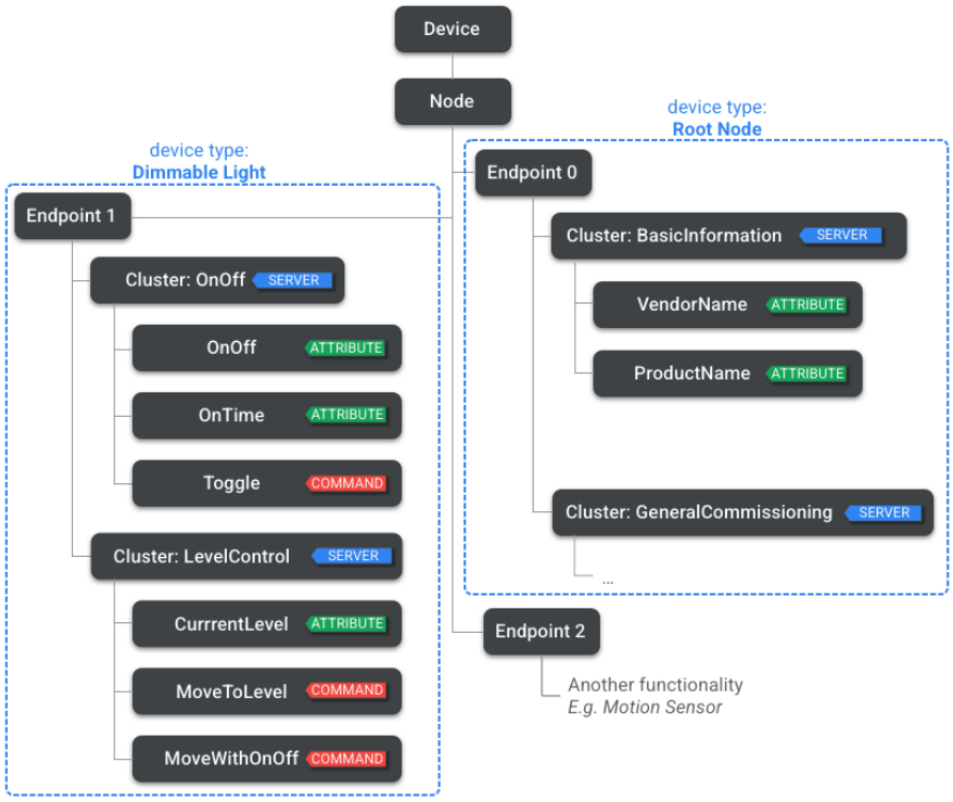


Figure 2.4: Matter data model [GoogleDev, 2023]

market, such as *IFTTT*[5], *SmartThings*[6] and *OpenHab*[7]. *IoTMashup* is the internal Orange trigger-action platform. These platforms provide REST API for users and IoT developers to allow them to register IoT devices and create automation rules using dedicated *Rule Model*. For instance, the *SmartThings* platform provides the *Rule API* for automation rule management. IoT platforms and more precisely Trigger-action platforms create abundant interactions and dependencies relationships among IoT devices to deliver complex and innovative IoT services.

1.1.2 IoT challenges

IoT technology faces several challenges that must be addressed to realize its potential and widespread adoption [Radoglou Grammatikis, 2019]. Some of the key challenges include:

- Security: IoT devices often collect and transmit sensitive data, making them susceptible to cyberattacks and unauthorized access. Moreover, most devices and IoT products do not get enough testing and updates to protect them from various security threats. To address these security challenges, IoT operators should consider robust security measures, e.g., regular updates, when deploying IoT solutions.

[5] https://ifttt.com/
[6] https://www.smartthings.com/
[7] https://www.openhab.org/

- Interoperability: is the capability of diverse systems, devices, or components to work together, enabling collaboration and efficient data exchange. Ensuring interoperability between various IoT devices and platforms provided by various manufacturers and service providers is challenging. To handle such a challenge, organizations and industry groups are working to establish standards and protocols to ensure interoperability between IoT devices and platforms. The *oneM2M standard* [8] was proposed in 2012 as a worldwide initiative to ensure the scalability and interoperability of IoT systems. In January 2016, the European Commission initiated funding for seven projects focused on addressing different aspects of interoperability within IoT. One of these projects, INTER-IoT [9], aims to design, implement, and experiment with an open cross-layer framework and methodology [Ganzha, 2017]. So far, the European Union has also funded several research projects under the H2020 program focusing on the federation of IoT platforms [Noura, 2019]. Despite these efforts, no universal interoperability standards have been adopted for the IoT and none can be expected to materialize in the near future. Thus, interoperability between IoT devices and platforms needs to be addressed when conceiving any IoT solution.

- Scalability: the number of IoT devices are increasing exponentially, with 70 billion IoT devices by 2025 [10]. Thus, managing and scaling IoT system become more complex. Indeed, IoT systems must be designed to handle large volumes of data and ensure the well-functioning of an ever-growing number of devices.

- Reliability: refers to the ability to provide a correct service continuously. In the context of IoT, devices are provided with limited computing resources and often deployed in harsh environments or remote locations where maintenance is difficult, making them widely exposed to failures that result in service disruptions and impact the overall reliability of the IoT system. Statistics have shown that an IoT device may fail four hours per day [Norris, 2022]. Thus, ensuring the reliability of these devices is crucial to avoid failures and disruptions in the IoT system.

1.2 IoT Device Management

An efficient solution to handle some of the above-mentioned challenges facing the IoT is to perform what we call *Device Management* (DM), which refers to the remote administration of IoT devices to ensure they are well-functioning. It is defined by the ITU-T in recommendation Y.4702 as "an essential set of management capabilities in the IoT, providing support for, but not be limited to, devices' remote activation and de-activation, diagnostics, firmware/software updating, and sensor node working status management" [ITU-T Y4702, 2016].

Technically, DM adopts a client-server architecture (see Figure 2.5) consisting of a *DM server* managing a software component called *DM client* installed on IoT devices to allow their remote administration. Communication between DM servers and DM clients is ensured using

[8] https://www.onem2m.org/
[9] https://inter-iot.eu/
[10] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

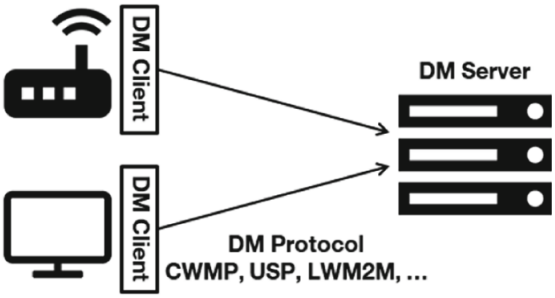


Figure 2.5: DM architecture from [Aïssaoui, 2020]

DM protocol. Relying on this architecture, common capabilities of DM include:

- Provisioning: includes pre-commissioning to pre-provision the IoT device with a set of credentials that allow it to securely connect to a given DM server or platform; commissioning or onboarding that refers to registering the device to allow its remote configuration; and configuration of new functionalities and services on IoT devices.
- Maintenance consists mainly of firmware over-the-air (FOTA) and software over-the-air (SOTA) updates of IoT devices to fix bugs, enhance functionalities, and prevent eventual security threats.
- Troubleshooting: including monitoring device states to detect device failure and security breaches and diagnosing device failure to define failure type and adequate recovery actions.

Due to the importance of DM, several efforts were conducted by researchers, industrial companies, and standard organizations to shape its architecture and functionalities. Several DM protocols were proposed and integrated into a dedicated industrial DM platform, while some research efforts have been proposed to unlock specific DM challenges. The following discusses DM protocols as well as research and market DM solutions. Finally, we highlight DM challenges.

1.2.1 DM Protocols and Standards

A DM protocol implements a set of DM operations e.g., a firmware update that the DM server can execute on the IoT device. Technically, it consists of a specific *DM data model* implemented in the *DM client* side, representing different resources such as device capabilities, device data, and supported DM operations exposed to manipulate these resources as CRUD operations. The most promising DM protocols are the *Lightweight M2M* (LwM2M) protocol[11] from the Open Mobile Alliance or the *User Services Platform* (USP) [12] from the Broadband Forum. The former is more adopted by market DM solutions and is more adapted to IoT devices with low computing resources.

- *Lightweight M2M* (LwM2M)
  A standardized DM protocol with its first version published in 2017. It proposes a

[11] https://www.openmobilealliance.org/release/LightweightM2M/
[12] https://usp.technology/

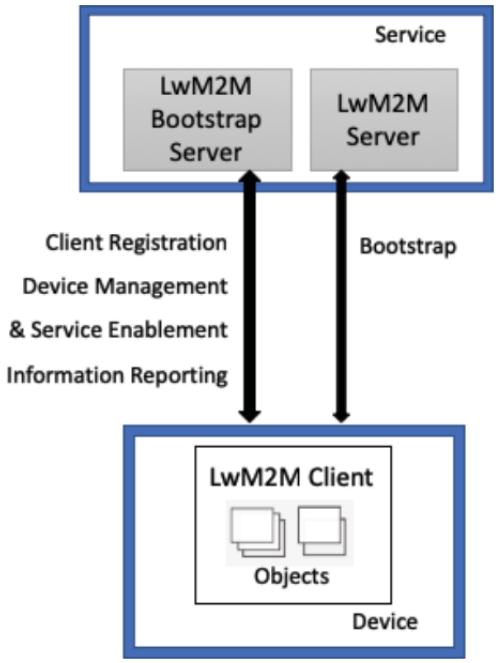


Figure 2.6: Architecture of the LwM2M protocol form [Sinche, 2020]

lightweight architecture and is quite effective over unstable connections and low-bandwidth networks. It implements a dedicated data model to describe devices' resources as objects, listing their attributes and access policies. The communication is based on User Datagram Protocol working over Constrained Application Protocol with the support of various data formats such as Type-Length-Value. LwM2M utilizes Datagram Transport Layer Security as a security layer to ensure authentication, confidentiality, and data integrity. LwM2M implements four (04) interfaces (see Figure 2.6) to enable dedicated DM operations between LwM2M server and LwM2M clients [Sinche, 2020]:

- Bootstrap: allows LwM2M servers to perform pre-commissioning on LwM2M clients.
- Client registration: allows LwM2M clients to register on LwM2M servers.
- DM and service enablement: allows LwM2M servers to perform CRUD operations on LwM2M client's resources represented as Objects.
- Information reporting: allows LwM2M server to track changes in LwM2M client objects through a notification system.

- *User Services Platform* (USP)
  A modern DM standard created by the Broadband Forum as a successor of the *TR-069* standard. For USP, DM server is called *Controller* and DM client is referred to as *Agent* (see Figure 2.7). USP proposes several data models such as TR-181 [13] that allow the representation of device resource called *Service Element*. These service elements are represented in the USP agent and can be accessed by one or many USP controllers. Data

[13] https://usp-data-models.broadband-forum.org/tr-181-2-15-1-usp.html

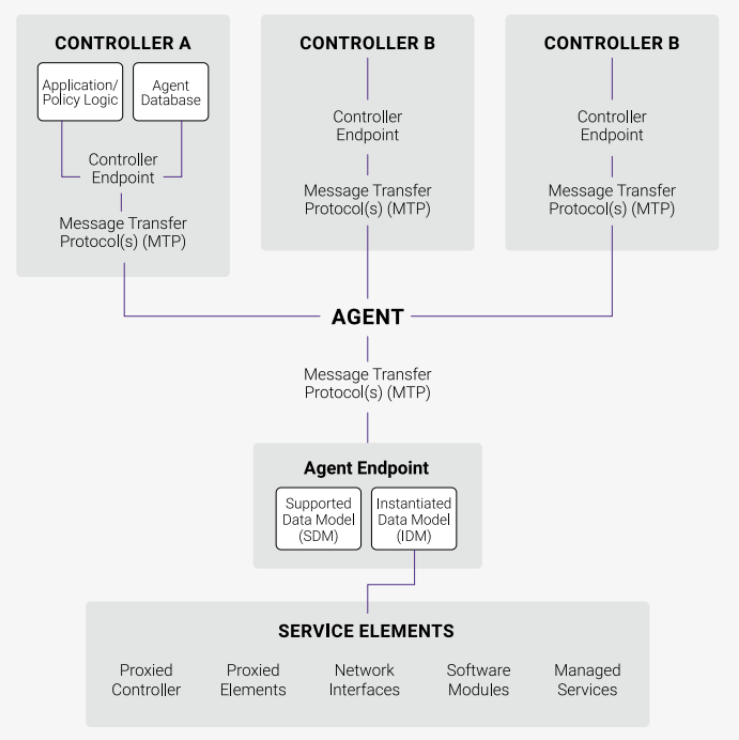


Figure 2.7: The architecture of the USP protocol from [Avsystem, 2023]

transfer in USP can be securely ensured by several Message Transfer Protocols such as STOMP and MQTT, thanks to data encryption and strict access control support. Among the advantages of USP are its modular and rich data models and its multi-controller architecture that allows customers to participate as well in the management of their devices, reducing the workload of customer care centers and improving customer satisfaction.

1.2.2 Market DM solution

As DM matters, big vendors like *Amazon*, *Orange*, and *Microsoft* have proliferated in the DM market by integrating DM services into their IoT platforms (see Table 2.1).

Amazon proposes a set of DM services integrated into its platform *Amazon Web Service* (AWS) mainly in AWS IoT core and AWS IoT Device Defender[14]. The main functionalities consist of registering and organizing a fleet of IoT devices, performing OTA updates, and monitoring device state with an alert system. *Orange* have deployed a DM as a service solution through its platform *LiveObjects*[15], by proposing several DM functionalities such as massive provisioning, devices inventory, configurations and firmware updates, and device monitoring. Liveobjects DM solution supports standardized protocols such as LwM2M, MQTT, and LoRa. The platform *AZURE*[16] of *Microsoft* proposes several DM features such as provisioning new devices, monitoring, and configuration using the LwM2M protocol. The latter was also used

[14] https://aws.amazon.com/fr/iot-device-management/
[15] https://liveobjects.orange-business.com/#/liveobjects
[16] https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-device-management-overview

in *Samsung*' platform *ARTIK Cloud*, *NOKIA*' platform *IMPACT* [17], *IBM*' platform *WATSON IoT* [18], and *Coiote* [19] platform of the *AVSYSTEM* company to enable multiple DM features.

Table 2.1: DM in market IoT platforms enriched version from the study [Sinche, 2020]

DM solution	Company	DM protocol	Features
Amazon Web Service	Amazon	MQTT	Registering and organizing a fleet of IoT devices, Performing OTA updates, Monitoring device state with an alert system.
LiveObjects	Orange	LwM2M, MQTT, websockets, REST, LoRa, SMS	Massive provisioning and campaign management, Devices inventory, Configurations and firmware updates, Device monitoring.
AZURE IoT	Microsoft	LwM2M	Provisioning of IoT device fleet, Configuring and updating IoT device fleet, Device monitoring using ML solution.
ARTIK Cloud	Samsung	LwM2M	Registering IoT devices, Device connectivity monitoring, OTA updates.
WATSON IoT	IBM	LwM2M MQTT	Registering IoT devices, OTA updates.
Coiote	AVSYSTEM	LwM2M CoAP SNMP MQTT	Device discovery, OTA updates.
IMPACT	NOKIA	LwM2M	Device monitoring, Device discovery, OTA updates.

Each DM solution is built upon a specific IoT infrastructure, encompassing the underlying hardware, software, and communication protocols that enable device connectivity. Some platforms might rely on cloud-based infrastructures, while others might opt for edge-computing solutions to minimize latency and enhance real-time processing capabilities. They even use proprietary DM Protocols, designed to offer optimized performance, security, and efficiency for their specific ecosystem of devices. These protocols are carefully crafted to suit the unique requirements of the platform, ensuring data exchange and device control. However, most rely on standard-Compliant DM protocols to ensure interoperability and compatibility between devices

[17] https://www.nokia.com/networks/internet-of-things/impact-iot-platform/
[18] https://internetofthings.ibmcloud.com/
[19] https://www.avsystem.com/coiote-iot-device-management-platform/

from different manufacturers. To ensure efficient data handling, storage, and analysis, market DM platforms often employ their data models. These models define how data is structured, processed, and interpreted within the platform. While some platforms may follow common data models, others might create proprietary ones optimized for specific use cases. As a result, market DM solutions are heterogeneous, each promotes its own IoT infrastructure, proprietary and standard– compliant protocols, and data models.

1.2.3 Research trends on DM

There has been several academia efforts addressing DM problems and proposing technical frameworks to enhance legacy DM features:

The work [Datta, 2015] proposes a framework for managing IoT devices. The device configurations are described using the CoRE Link format [20]. The framework has three layers: the Proxy layer that manages devices unable to integrate the system due to their limited DM-supported capabilities; the Configuration layer responsible for extracting and managing device configurations; and the Service layer that allows access control and configuration management via RESTful APIs. This framework can be deployed in a smart device, a gateway, or the cloud depending on the size of the managed IoT devices. In terms of implementation, the proposed solution has been implemented via the oneM2M standard. This work was improved in [Perumal, 2016] which enriched the Configuration layer with device discovery functionalities and processing services, the latter being used to collect device data for reasoning and self-management.

In [Pham, 2016], the authors propose an architecture that combines the two management approaches used to manage devices in smart homes: direct, where devices are managed directly by a cloud platform, and indirect, where DM is done through a gateway that interacts with the remote management platform. The solution is based on installing smart applications for data management, which are placed according to the device's capacity: in the IoT device or the smart gateway.

In [Ferreira, 2017], a framework named DEVMEN is proposed, consisting of a set of web services for adding, automatically provisioning, and monitoring new IoT devices on the network. A semantic ontology (see Chapter 2 Section 2.1.1) was used to identify the devices and create their profile by matching the existing device's profiles in the knowledge base.

A DM system based on an interoperable lightweight agent is proposed in [Maloney, 2019]. The solution assists in executing updates and security configurations on a large fleet of IoT devices. The work [Mavromatis, 2020] proposes a distributed framework for managing devices belonging to different applications (multi-domain). The solution mainly addresses the scalability problem and uses software-defined networking (SDN) in an Edge environment for IoT device control and provisioning. Authors in [Armando, 2019] propose a unified management solution to address the problem of heterogeneity between sensor types by considering virtual and human sensors. The solution uses the concept of extended IoT, the ITU-T reference architecture, and the LwM2M protocol. Authors in [Moura, 2019] propose an interoperable DM solution in an Industrial IoT environment. The proposed solution allows for managing any devices involved

[20] https://tools.ietf.org/html/rfc6690

in industrial operations. It includes several modules for device configuration, maintenance, and security and a central module to coordinate between the other modules. However, the proposed solution has not been validated.

Recently, Jia Yan et al. propose in [Jia, 2021] the framework *CGuard* to address the phenomenon of *chaotic device Management* that describes the non-alignment of security policies on an IoT device managed by siloed and fragmented DM solutions, which may lead to serious security threats.

The industrial community conducted some research efforts to handle the fragmentation of the DM market. The operator KDDI argues that DM is performed in a horizontal specialization business model, where devices are managed by multiple DM platforms governed by different actors. In their work [Shibuya, 2016], they propose a federated approach to calculate the cumulative failure rate of devices in a horizontal business model where device history information is managed by multiple manufacturers and service providers. The cumulative failure rate is an index that measures the reliability of a fleet of devices. It is defined as the ratio of failed devices to the number of devices. The approach analyzes several distributed information systems to extract the device's operation history to calculate the cumulative failure rate.

In the same context, Orange claimed the necessity to federate DM solutions in their presentation [Bolle, 2019] for the European Telecommunications Standards Institute (ETSI). Next, they propose in [Aïssaoui, 2020] a semantic model for the DM domain to enable the unified management of IoT devices managed by multiple actors. The proposed model describes the DM domain to provide a unified understanding of DM for heterogeneous and distributed DM solutions managed by different actors. The proposed ontology is based on the SAREF ontology[21] and describes the supported DM functions for each device and the means to access them. The solution was enriched via a use case of selecting a proxy for integrating a new device in an existing IoT system. This ontology was used in the work [Mezghani, 2020], which proposes an autonomous system using MAPE-K loop for autonomous coordination between heterogeneous DM solutions. The solution orchestrates the execution of DM operations by considering conflicts related to connectivity dependencies between IoT devices. Moreover, Orange has conducted several efforts to address the scalability of legacy DM solutions by proposing solutions for distributing the DM processes through autonomous MAPE-K loop and constraint programming [Ayeb, 2020b; Moualla, 2022].

1.2.4 DM Challenges

Despite these efforts, current DM is facing several challenges due to the complexity and diversity of the IoT network:

- Multi-level heterogeneity: The current DM is facing significant challenges due to multi-level heterogeneity, which includes heterogeneity among IoT devices, DM protocols, and DM platforms. DM platforms must integrate heterogeneous IoT devices with their various supported connectivity protocols and consider their heterogeneous capabilities. Moreover,

[21] https://saref.etsi.org/

current market DM platforms are quite heterogeneous and governed by different vendors, leading to fragmented and siloed DM solutions managing interdependent IoT devices and applications. This architecture presents several challenges that prevent DM solutions from providing full business value. A main challenge is cascading failure, where a failure of one IoT device spreads across devices managed by siloed DM platforms. In such scenarios, it is difficult to identify the source of failure and recover the failed devices. Therefore, interoperability across siloed DM platforms is crucial to enable collaborative DM processes across siloed DM solutions to unlock these challenges.

- Scalability: managing the huge number of IoT devices and their data in a complex task. Indeed, billions of devices must be provisioned, registered, and continuously monitored and updated within the DM platforms. This challenge may be addressed through distributed computing and load-balancing within DM platforms.

- Dependency-related threats: complex and abundant interdependencies relationships between IoT devices generate several threats that the current DM solution can not handle. In addition to the cascading failure discussed above, failures during the execution of DM operations, e.g., *firmware update* or *reboot* on interdependent devices, are another dependency-related threat. Indeed, DM operations can cause devices to be temporarily unable to provide their services, leading to failures on dependent devices and breaking updates [Mezghani, 2020; Zdankin, 2021]. These failures are exacerbated by the uncoordinated execution of DM operations using multiple DM solutions [Jia, 2021] and are usually hard to revert [Zdankin, 2021].

In our work, we are particularly interested in endowing market DM solutions with efficient and interoperable capabilities to tackle dependency-related threats, mainly cascading failures on IoT devices.

1.3 Orange motivation behind the presented research

As a telecommunications operator, Orange currently manages over 21 million IoT devices, primarily in the Smart Home domain, deployed at its customers' premises. These devices require continuous maintenance and repairs to ensure a better Customer *Quality of Experience* (QoE), a key business element, particularly for Orange. For that, Orange disposes of a set of DM solutions within customer care services to manage and troubleshoot customer devices, involving interacting with customers through phone calls: The customer reports the failure on their device; Orange technician uses a set of tools to identify the source of the failure and relies on the DM solution to remotely perform management operations on the device; If the reported failure persists, the Orange technician guides the customer through steps to resolve the problem or schedules an on-site technical intervention. However, this process incurs a significant cost in terms of time and money, impacting the QoE of Orange customers. Indeed, Orange 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. These costs are exacerbated when failures

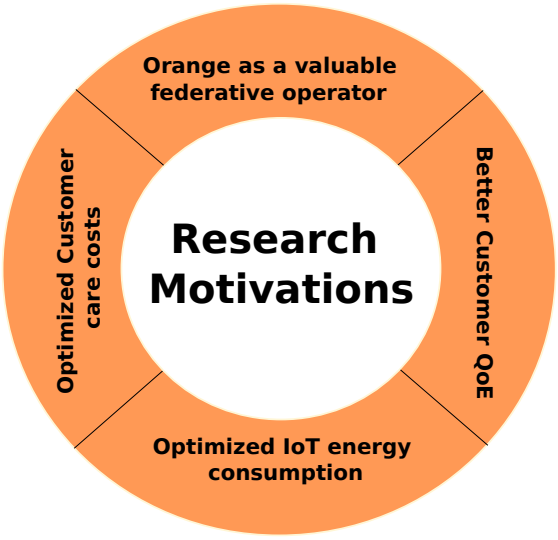


Figure 2.8: Orange Motivation behind the presented research

are caused by the failure of other devices managed by other operators and service providers. In such a situation of cascading failures, it is quite difficult for Orange and other operators to identify the source of failure and recover the failed devices since they provide isolated solutions for failure management. Moreover, these 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 [Najeh, 2019].

In this context, our work within *Orange*, more precisely *Orange Innovation* [22] is guided by the following motivations (see Figure 2.8): 1) Boosting *Orange* value in the DM market by proposing federative tools for other operators. This motivation is guided by the Orange vision of *Service facilitator* consisting in the development of future Home services with partners, such as helping them manage cascading failures on their devices; 2) Enhancing Orange customer QoE; 3) Reducing customer care costs by minimizing customer calls and technician intervention; 4) Optimizing IoT energy consumption, by minimizing energy loss related to IoT device failures.

2 Scientific Context

This section describes scientific concepts used in our methods. We present their architectures and technological implementations as well as their utility especially for tackling the cascading failure problem.

2.1 Semantic Web

When it appeared, the *Semantic Web* (SW) was envisioned as an enhancement of the current World Wide Web with *machine-understandable information*, in order to move from a web linking documents serving mostly human-to-human communication to one linking everything such as applications, things, and people, where humans efficiently exploit web information with the help

[22] Orange Innovation is the research and innovation Lab of the Orange company.

of machines [Gandon, 2018]. The main motivation is to enhance scalability and allow more sophisticated processing and exploitation of data provided by the Web involving both humans and machines.

The story of the SW started in September 1998 when a researcher called *Tim Berners-Lee* introduced the *SW Road map* giving the blueprints of the architecture of the SW [Berners-Lee, 1998]. Then, the concept was given significant visibility to a broad audience in 2001 through an article in the Scientific American [Berners-Lee, 2001]. However, there were other efforts, such as the DARPA Agent Markup Language (DAML) program started in 2000 to develop a SW language and corresponding tools.

After that, several SW tools, methods, and standards were developed, focusing on how to represent the data on the Web, and make it understandable by machines together with services a.k.a "intelligent agents" utilizing and reasoning on the Web data. Such intelligent agents make the SW overlap with the field of Artificial Intelligence [Hitzler, 2021].

These developments have reshaped the SW's initial vision, i.e., "extending the current Web" to a more valuable vision. Indeed, the SW community noticed that the methods and tools developed by the field can also provide added value in information integration and management fields, by easing data sharing, discovery, integration, and reuse.

In our work, we rely on such vision to respond to our research questions. We leveraged several SW standards to ease information management in the context of Cascading Failure management. In the following, we present the SW standards used by our work within the SW architecture and highlight recent SW applications mainly in the information management field.

2.1.1 Architecture

The architecture of the SW was proposed during the first half of the years 2000 by the *World Wide Web Consortium* (W3C) in the form of a stack known as "SW Layer Cake" structuring the SW technologies and standards (see Figure 2.9). It includes data identification standards through URI/IRI, data representation language and formats, meta-data representation language, which allows more efficient representation of data, data query languages, and tools for reasoning on the data. This stack was updated in [Gandon, 2018] to consider recent standards and technologies (see Figure 2.10). This work uses the term "stack overflow" to denote the growing number of proposed standards for the SW that confirms the interest it generates in various fields. In the following, we present SW standards used in our work.

- *Resource Description Framework* (RDF) [23]: is a standard model for data representation and interchange on the Web with its current version (1.1) published in 2014. RDF leverages URIs to name entities and the relationship between them. These entities and relationships are structured as triples represented by subject, predicate representing the relationship, and object. Thanks to this structure, unstructured and structured data with different schemas can be mixed, exposed, and shared across heterogeneous applications.

[23] https://www.w3.org/TR/rdf11-concepts/

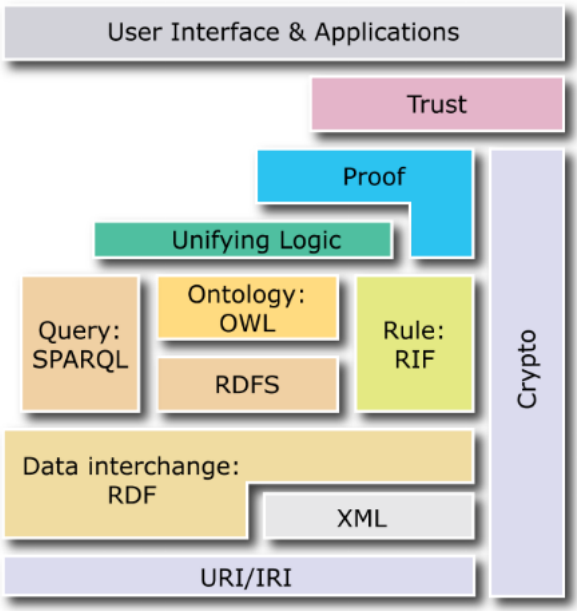


Figure 2.9: Semantic Web Stack [Berners-Lee, 1998]

- *Web Ontology Language* (OWL)[24]: is a standard schema language published in 2009. It is designed to represent a common vocabulary known as "Ontology" for published data. The most common definition of an ontology was given by [Studer, 1998] as "a formal, explicit specification of a shared conceptualization". An ontology describes three elements: 1) a set of concepts called *classes* described through 2) a set of *data properties*, and linked with each other through 3) relations called *object properties*. The goal of an ontology is to achieve *Semantic interoperability* i.e., providing a common understanding of a domain of knowledge across heterogeneous systems. OWL provides dedicated language to describe classes, data, and object properties. Moreover, it disposes of advanced features such as defining sophisticated semantics called *axioms* on the ontology element e.g., constraints and restriction, and linking multiple ontologies together by linking their concepts and relations.

- *SPARQL Protocol and RDF Query Language* (SPARQL)[25]: In order to query published data formalized in RDF, the SPARQL query language was proposed in 2008. It allows adding, removing, and retrieving data from RDF data, thanks to graph pattern-matching techniques. Moreover, it uses mathematical operations and utility functions to create filters and bindings. SPARQL also supports optional graph patterns along with their conjunctions and disjunctions.

- *Shapes Constraint Language* (SHACL) Advanced Features [26]: The SHACL[27] standard has been defined in the reasoning layer of the SW stack mainly for RDF data validation

[24] https://www.w3.org/TR/owl-features/
[25] https://www.w3.org/TR/rdf-sparql-query/
[26] https://www.w3.org/TR/shacl-af/
[27] https://www.w3.org/TR/shacl/

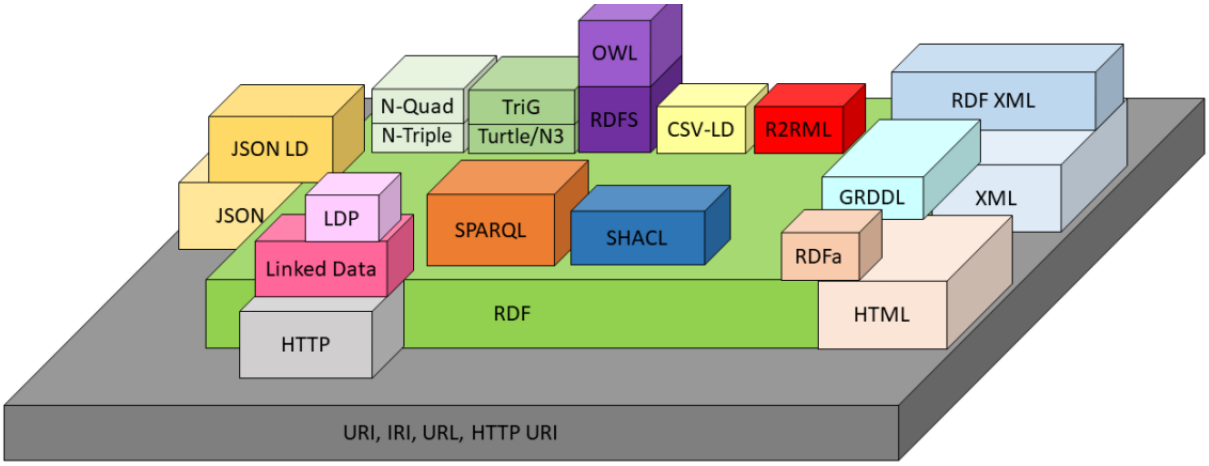


Figure 2.10: Updated Semantic Web Stack [Gandon, 2018]

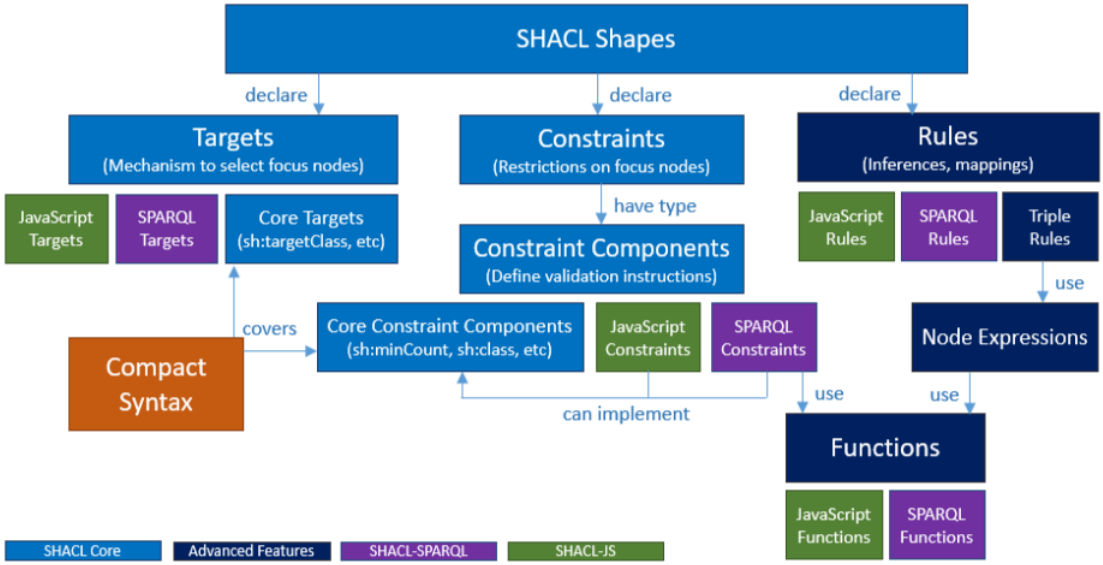


Figure 2.11: Features of the standard proposed by TopQuadrant [28]

(see Figure 2.11). The SHACL reasoner checks RDF data regarding a set of constraints called "shapes" and returns a validation report. A working group note was proposed to extend SHACL through a set of advanced features named SHACL rule and SHACL function. SHACL rule allows inferring new triples in RDF data by reasoning on existing ones. SHACL functions define operations that produce an RDF term, giving a set of parameters and a data graph. SHACL functions can be used within FILTER or BIND clauses within SPARQL queries or in SHACL rules in order to perform data transformations such as string concatenation or mathematical operations.

- *Thing Description* (TD)[29]: this standard describes, in the form of ontology, a set of the metadata and interfaces of thing interactions in the Web. A Thing is an abstraction of a

[28] https://www.topquadrant.com/
[29] https://www.w3.org/2019/wot/td

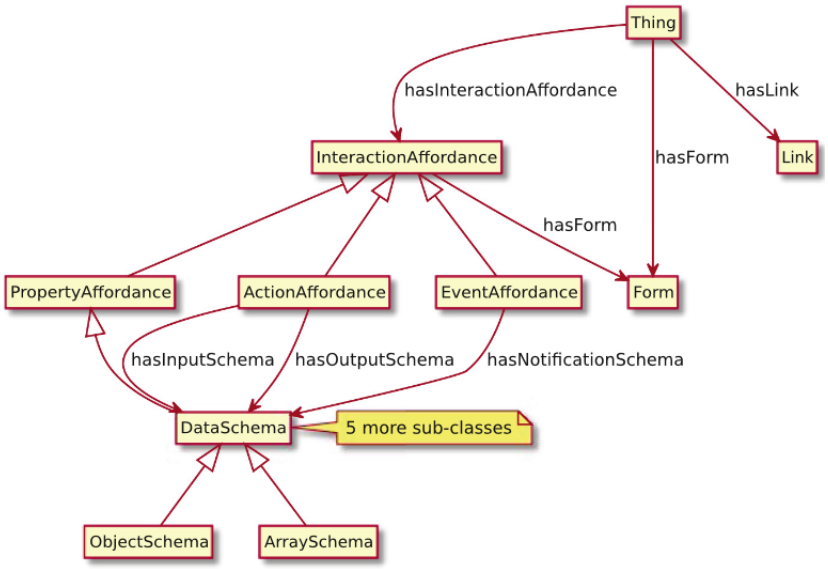


Figure 2.12: The Thing Description Ontology [Charpenay, 2020]

physical or virtual entity that exposes modalities that allow their interactions in the Web of Things. Three types of interactions named *affordances* are defined (see Figure 2.12): *Action Affordance* allows to invoke a function of the Thing; *Property Affordance* exposes information of the Thing. This information can be retrieved or updated; and *Event Affordance* describes events related to the Thing. These affordances are described using *links* and *forms* proposed in *Hypermedia Controls Ontology*[30], allowing a shared representation of links and access form in the Web.

2.1.2 Semantic Web Applications

Within more than 20 years of existence, the SW applications have evolved through three perspectives. The earlier was the one using the SW as an extension for the Web. The second perspective includes applying the SW standards and tools for information management and integration. The more recent perspective involves investigating the usage of ontologies, linked data, and more recently, *Knowledge Graph* (KG) together with the W3C SW standards [Hitzler, 2021]. Linked data refers to design principles for linking and sharing opened RDF data on the Web based on community contribution. While the KG technology, proposed firstly by Google in 2012, represents closed knowledge separately governed and managed by industrial actors. Relying on these perspectives, several applications of the SW were proposed in the literature in a wide range of domains. In the following, we discuss relevant applications of the SW standards extracted mainly from the in-use track of the International Conference of Sematic Web (ISWC), which is the premier forum of SW. We focused on applications showing practical adoptions of SW standards to break silos and unlock industrial challenges.

The work [Piro, 2016] presents a relevant application of Semantic Technologies in health-

[30] https://www.w3.org/2019/wot/hypermedia

care data analysis that has emerged from the collaboration between Oxford University and Kaiser Permanente a US healthcare provider (HMO). The idea is to use SW technologies mainly SPARQL and rule-based reasoning through RDFox [31] to compute quality measures of healthcare providers such as the proportion of diabetic patients having regular eye examinations, across multiple and heterogeneous data sources. The use of SW technologies led to highly encouraging results: only 174 rules were required, compared to the roughly 3,000 lines of complex and hard-to-maintain SQL code of their previously used HMO solution. Secondly, RDFox could easily handle 1.6 billion triples of patient data and compute the quality measures in approximately 30 minutes. In the same healthcare information management context, SW technologies were applied in the pharmaceutical domain, where heterogeneous and unstructured technical information was efficiently and automatically retrieved from PDF files to boost pharmaceutical documents and information searches [Gentile, 2019].

Accenture company has leveraged the SW technologies for project risk management and mitigation within their Intelligent Risk Management tool (IRM) [Wu, 2017]. The approach integrates projects and their context data into an enterprise knowledge graph and interprets risk mitigation actions based on quality manager profiles through semantic reasoning. User studies showed that quality managers could efficiently select actions for risk mitigation.

The Open University (OU), in collaboration with Springer Nature, created Smart Book Recommender (SBR), an ontology- based recommender system supports their Computer Science editorial team in selecting the products to market at specific venues. SBR proved its usability through a user study involving seven SN editors and seven OU researchers showed that SBR was able to suggest relevant materials [Thanapalasingam, 2018]. Another ontology-based recommendation system was proposed in [Obeid, 2018] to guide high school students in selecting a major and a university.

A relevant application of SW technologies was introduced in [Rojas, 2021b] to help the European Union Agency for Railways structure its 28 distributed and independent data sources describing technical information about the European railways, such as railway infrastructure aspects and rolling stock. The proposed approach was validated through the use case of route compatibility checks.

2.1.3 Semantic Web and IoT

Moving to the IoT domain, the SW technologies have played a pivotal role in achieving interoperability, a major challenge for the IoT (see Chapter 2 Section 1.1.2). Within the IoT, the SW standards aim to define consensus that facilitates the sharing, reuse, integration, and interrogation of data, extracted from different IoT devices and applications, ensuring cooperation between the IoT devices by facilitating communication between them to boost their intelligence aspects. The combination between IoT and SW standards led to a new paradigm, the SW of Things (SWoT) [Scioscia, 2009]. Several projects were involved in the development of the SWoT,

[31] https://www.w3.org/2001/sw/wiki/RDFox

like IoT-A [32] and the linked open vocabulary project (LOV4IoT [33]) that collects and regroups numerous relevant ontologies for the IoT domain.

Moreover, SWoT has been applied in several works, mainly for usages related to IoT data representation and IoT services description, discovery, and selection [Rhayem, 2020]. Regarding IoT data representation, the Semantic Sensor Network Incubator Group, belonging to the W3C, developed an ontology called Semantic Sensor Network (SSN) to represent IoT sensor data. The MELODY projects introduced a semantic actuator network (SAN)[34] to represent the semantics of actuator capabilities. Many other ontologies were proposed to semantically describe various IoT aspects [Seydoux, 2018].

SWoT has been leveraged as an efficient tool to ease IoT services discovery and selection. IoT service selection consists in selecting the most relevant service from as set of discovered IoT devices based on several criteria, such as Quality of Service (QoS). The idea is to semantically annotate IoT services to allow IoT devices and applications to efficiently discover and select IoT services [Pahl, 2019; Khadir, 2020; Khadir, 2022].

Recently, Orange has initiated the usage of SWoT to manage IoT devices provided by siloed DM platforms [Bolle, 2019], by modeling DM data of IoT devices and leveraging the TD standard (see Section 2.1.1) to perform DM operations on interdependent IoT devices managed by siloed DM platforms [Aïssaoui, 2020]. Our work explores more use cases of SWoT to provide interoperable DM.

2.2 Digital Twin

The rapid growth of several technologies such as cloud computing, big data, IoT, and sensor technologies has given birth to the *Digital Twin* (DT) technology, which refers to the virtual and synchronized representation of physical world objects. In 2017, 2018, and 2019, Gartner classified the DT as one of the top ten most promising technological trends in the next decade [Tao, 2018]. In 2020, Gartner also listed the DT as an emerging technology for the next 5-10 years [Tao, 2022]. The emergence of the concept of *Metaverse* in 2022 has revived the use of DT. Within the 3D digital spaces, physical world entities are represented as DT, powered by the technologies of virtual reality, augmented reality, and artificial intelligence [Far, 2022].

The concept of DT was first presented by Grieves in 2003 at the University of Michigan [Grieves, 2014]. Years later, the National Aeronautical Space Administration (NASA) established a crucial milestone in defining the DT, with their famous definition: "DT is a multi-physics, multiscale, probabilistic, ultra fidelity simulation that reflects, in a timely manner, the state of a corresponding twin based on the historical data, real-time sensor data, and physical model" [Glaessgen, 2012]. Up to now, several explanations and definitions of DT have been proposed. The DT Consortium [35] defines the DT as: "A DT is a virtual representation of real-world entities and processes, synchronized at a specified frequency and fidelity". The CIRP Encyclopedia of Production Engineering proposed the following definition: "A DT is a digital

[32] https://www.iot-a.eu/
[33] http://www.lov4iot.appspot.com/
[34] https://www.irit.fr/recherches/MELODI/ontologies/SAN.html
[35] https://www.digitaltwinconsortium.org/hot-topics/the-definition-of-a-digital-twin/

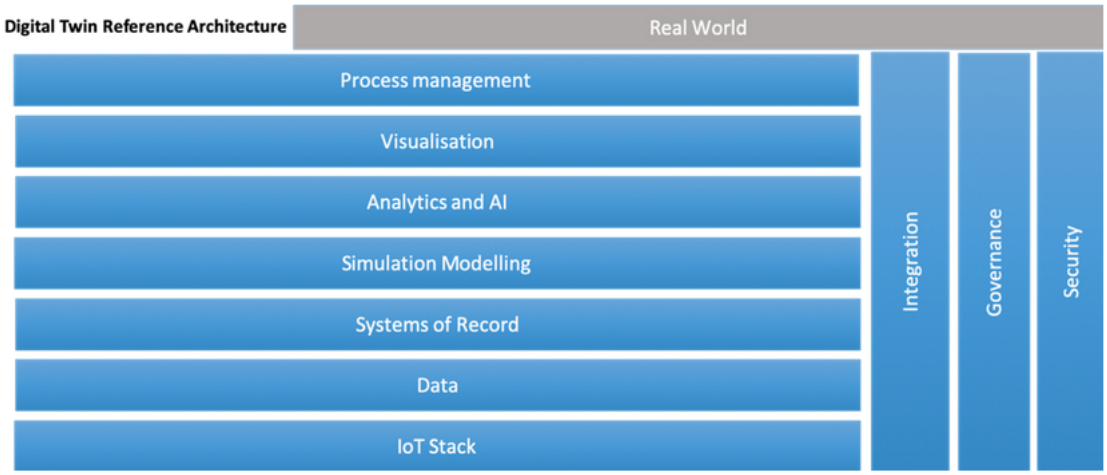


Figure 2.13: IBM reference architecture for DT [Andy Stanford-Clark, 2019]

representation of an active unique product (real device, object, machine, service, or intangible asset) or unique product-service system (a system consisting of a product and a related service) that comprises its selected characteristics, properties, conditions, and behaviors by means of models, information, and data within a single or even across multiple life cycle phases" [Stark, 2019].

From an architectural point of view, IBM provided a 7-layer reference architecture for the DT technology (see Figure 2.13). The first layer describes IoT technologies such as sensors allowing data collection from the physical world, and the second, third, and fourth layer allows collecting, managing, and structuring data according to a specific DT model. The fifth layer allows analyzing the DT data through artificial intelligence tools. And as an appropriate user interface is very important for almost every service of the DT [Steindl, 2020], the sixth layer provides utilities for DT visualization. And the last layer allows process management through the DT to boost decision-making, reduce time to market and improve quality. These seven layers are enhanced with integration, governance, and security capabilities to ensure the DT system is appropriately coupled, and governed to ensure the quality of data and secured [Andy Stanford-Clark, 2019].

In the following, we provide market and open source DT implementations through the presentation of market DT platforms, focusing on the Orange implementation of the DT namely *Thing in The future platform*. Finally, we discuss current applications of the DT in various domains mainly the IoT.

2.2.1 DT Platforms

To meet the need for large adoption and deployment of the DT technology, major software companies such as *Amazon* and *Microsoft* have begun providing support for creating and operating the DT, commonly referred to as DT platforms. The open-source community has also taken part in developing the DT through various platforms.

- *AWS IoT TwinMaker*[36] is the DT platform proposed by *Amazon*. Thanks to customized data connectors, this platform connects heterogeneous data such as IoT, video, and application data to build an accurate DT in a graph-based model, which can be visualized in 3D format and combined with existing 3D models to get a holistic view of the physical system. Moreover, it provides plugins allowing to building of various web applications to serve several use cases such as predictive maintenance, production optimization, and building monitoring.

- *Azure Digital Twin*[37] proposed by *Microsoft*. It collects data from IoT and enterprise applications to build a virtual representation of the physical world. It relies on an open-language model to create a DT of any connected environment. It proposes several connectors to other *Microsoft* platforms such as *Azure IoT Hub* for data collection and *Azure Data Explorer* and *Event Hub* to track changes in the DT. Moreover, it enables the DT historization features to keep track of different changes in the DT. This allows for an insightful analysis of the connected environment such as predicting system failures.

- *Ditto* [38] there are some open-source initiatives for the DT such as *iTwin.js*[39], *Kuzzle IoT* [40], and *Ditto*. The latter is among the valuable open-source efforts for the DT. It builds a DT for devices connected to the Internet. It proposes various features such as device monitoring, state, and access control management for the DT. DT data is collected through a connectivity layer supporting various technologies and protocols such as *Eclipse Hono*, *MQTT*, and *Apache Kafka*. It can be accessed through multiple IoT applications using HTTP and *WebSocket*. *Ditto* is currently used by several enterprise such as *Bosch* as part of their platform *Bosch IoT things*.

2.2.2 Thing in The future, more than a DT Platform

Thing In The Future (Thing in) [41] is the research DT platform proposed by *Orange* [Derrien, 2019]. It represents connected and non-connected objects and their interactions within the physical world. It allows real-world data collection from IoT platforms using connectors. IoT device's data are accessed using *data access modalities* relying on the SW standard TD (see Chapter 2 Section 2.1.1). The collected data are combined and mapped to a graph-based representation that can be syntactic or semantic using SW ontologies (see Chapter 2 Section 2.1.1). These graph-based representations are leveraged even for human-based analysis or to build other business services.

The architecture of Thing in (see Figure 2.14) is composed of three layers namely *Core layer*, *Enabler layer*, and *Service layer*.

[36] https://aws.amazon.com/fr/iot-twinmaker/
[37] https://azure.microsoft.com/en-us/products/digital-twins/
[38] https://www.eclipse.org/ditto/
[39] https://www.itwinjs.org/
[40] https://kuzzle.io/kuzzle-iot-open-source-platform/digital-twins/
[41] https://www.thinginthefuture.com/

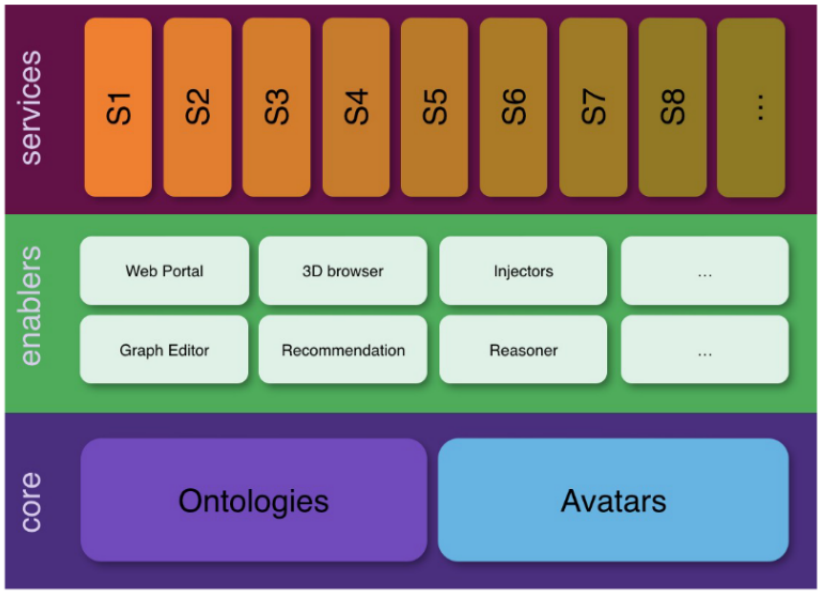


Figure 2.14: Thing in The Future Architecture [Derrien, 2019]

- *Core Layer*: encompasses descriptions of the physical world objects, a.k.a *Avatars*, in the form of a graph including their properties and relationships, these descriptions can be based on semantic ontologies. This layer exposes APIs for performing CRUD operations on these descriptions (e.g., create avatar, update avatar, delete avatar).
- *Enabler Layer*: Thing in enablers are services that are used to add functionalities to Thing in through the use of APIs offered by the Core layer such as *Injection Enabler* which allows injecting raw data in different formats such as *JSON* and *BIM* to generate and combine graph representations in Thing in.
- *Service Layer*: using Thing in enablers, this layer involves business use cases of Thing in for creating value-added services and applications such as product management in a factory [42], as well as security management in a smart building [43].

In addition to its DT features, Thing in is a multi-sided federation platform. Namely, it allows information sharing across various partners, customers, and suppliers collaborating to build valuable business services. In our work, we rely on both the DT and multi-sided capabilities of Thing in to enable cascading failure management on IoT devices managed by different actors.

2.2.3 Applications of Digital Twin

The DT is affecting all industries, leveraging the digital-twin-based simulation as key to performing effective decisions. For instance, manufacturing processes within Industry 4.0 leverage the DT to replicate production systems in real-time and analyze them, enabling various activities such as monitoring, maintenance, management, optimization, and safety [Cimino, 2019].

[42] https://www.thinginthefuture.com/spip.php?article117#main_nav_fermer
[43] https://www.thinginthefuture.com/spip.php?article118#main_nav_fermer

In Healthcare, DT are mainly used in two cases: patient DT and DT of medical devices. The patient's DT is formed by translating their physical attributes and physiological changes into a real-time digital representation, aiming to provide accurate diagnosis and the subsequent implementation of treatments tailored to each patient. The DT for medical devices improves device development, functionality, and maintenance [Erol, 2020]. Moving to the energy management domain, the concept of energy DT is under exploration to enhance on-site operations and reduce targeted energy consumption. This approach helps in the development of energy-efficient designs and the evolution of production processes and facilities. Moreover, it enables the transition to renewable fuels and improved integration of locally generated renewable energy sources [Yu, 2022]. Besides, the DT are used for disaster management in Smart cities, enhancing preparedness, response, and mitigation efforts by creating a virtual environment representing real-world disaster scenarios [Ford, 2020]. The DT enables other valuable use cases in the construction industry. Indeed, a DT for building involves creating a virtual representation of a physical building in a digital environment. This digital representation involves various aspects of the building's design, construction, operation, and maintenance [Opoku, 2021]. The *BIM2TWIN* [44] European collaborative project, in which Orange is a partner, is a practical example of using DT for construction.

2.2.4 Digital Twin and IoT

The relationship between IoT and the DT is characterized by a bidirectional exchange of benefits. Indeed, IoT feeds data to the DT through sensors that collect real-time data from physical assets, environments, and processes. This data is fed into the DT, enabling an accurate and synchronized virtual representation of the physical system. Thus, allowing the DT to monitor the health, performance, and operational status of assets, enabling predictive maintenance and timely interventions. Several works proposed to integrate IoT [Al-Ali, 2020] and industrial IoT [Souza, 2019] to the DT architecture for real-time data collection. IoT actuation could also be used as a way to reflect the status of the system in the digital back to its physical counterpart.

On the other side, the DT boosts IoT design and maintenance. Namely, the DT simulates IoT-enabled systems, allowing for predictive modeling and analysis. IoT data analyzed within the DT helps predict device failures, reducing downtime and minimizing IoT operational risks. For instance, the work [Gupta, 2021] used DT for anomaly detection on IoT-based healthcare systems, and authors in [Nguyen, 2022] propose *TaS*, a DT based tool for test and simulation of IoT environments, in order to detect and predict failures in evolving IoT systems.

In our work, we leverage the latter direction by using the DT to enhance IoT operations, mainly managing cascading failure on IoT devices.

2.3 Multi-agent System

Multi-Agent System (MAS) was introduced as a branch of distributed artificial intelligence [Dorri, 2018] during the Distributed Artificial Intelligence Workshop in June 1980 at Endicott House,

[44] https://bim2twin.eu/

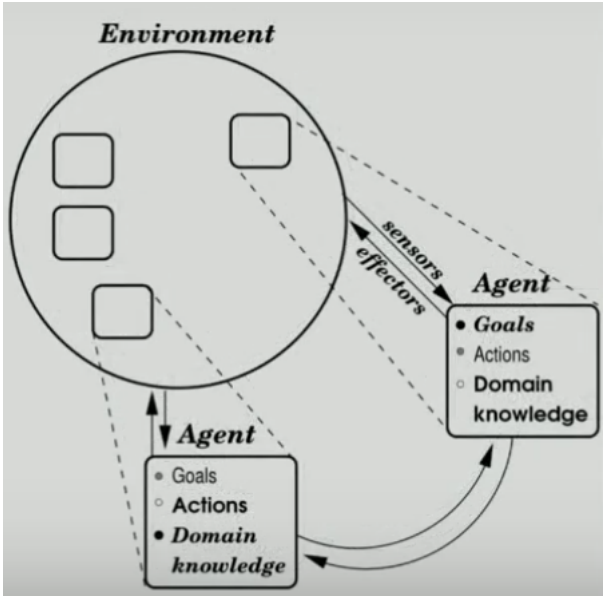


Figure 2.15: A representation of the MAS from the talk of *Stefano Albrecht* in The multi-agent research group in the Alain turning Institution [Stefano Albrecht, 2020]

Massachusetts Institute of Technology [45]. It refers to a network of loosely coupled entities known as *Agents* that interact within their shared environment to solve complex problems beyond each agent's individual capabilities or knowledge. The most valuable work conducted by the researcher *Michael Wooldridge* defines the MAS as multiple agents that interact with each other. These agents often act on behalf of distinct users with varied goals and motivations. To effectively interact, they must be able to collaborate, synchronize, and negotiate with each other as humans do [Wooldridge, 2009]. Another definition provided by [Boissier, 2020] is "an organized ensemble of autonomous goal-oriented entities called *agents*, communicating with each other and interacting within an environment." (see Figure 2.15). This work argues that agents may go beyond software to cover humans, hardware, or any other autonomous entity having a goal to achieve independently of other entities. Based on these definitions, a MAS has a set of properties that allows distinguishing it from other similar systems such as *object-oriented programming* and *expert system* [Rocha, 2017; Boissier, 2020; Dorri, 2018]:

- *Autonomy:* Each agent in the MAS must be autonomous, allowing it to perform actions independently based on its own goals, local knowledge, and its perception of the environment.
- *Decentralization:* Agents in the MAS have full decentralized control and knowledge, where each agent governs a portion of knowledge and can perform a subset of control actions to reach its own goal or to coordinate with other agents toward common goals.
- *Coordination:* Agents communicate with each other through a set of coordination algorithms, mechanisms, and processes, allowing them to interact and collaborate with each other to achieve common goals.

[45] https://dspace.mit.edu/handle/1721.1/41155

- *Heterogeneity:* Denotes the diversity and variety among individual agents. Each agent has its own characteristics, capabilities, knowledge, goals, or behaviors. This diversity is a key feature of MAS and contributes to their ability to handle complex and varied tasks.

- *Adaptation:* Refers to the ability of individual agents or the MAS to adjust their behaviors, strategies, or structures in response to environmental changes, agent characteristics, or goals.

Relying on these properties, the MAS provides several advantages for developing efficient, decentralized, and open software systems designed to function in an ever-changing environment, engaging with and executing tasks on behalf of human users and legacy software systems [Boissier, 2013]. More precisely, it provides the following advantages:

- *Increase performance:* Within the MAS, a complex task is divided into smaller tasks, each assigned to a distinct agent. This process naturally distributes the related costs, such as processing and energy usage, among multiple agents. This often leads to a cost-effective resolution, unlike an alternative where a centralized and potent entity attempts to tackle the complex problem.

- *Reliability:* Thanks to the decentralization property, the MAS provides high reliability. Namely, when an agent fails, its tasks can be readily reassigned to other agents, avoiding the "single point of failure" problem.

- *Human integration:* Integrating humans as "human agents" into the MAS allows to address several challenges that are difficult for purely automated approaches. Indeed, human agents may empower other computer-based agents with their expertise, intuition, and judgment, particularly in complex, uncertain, or ambiguous situations.

- *Legacy system integration and collaboration:* The MAS facilitates the integration and cooperation of legacy systems, which might be geographically dispersed. By enveloping these systems with agent interfaces, they can collaborate and share information to solve complex problems.

In our work, we leverage these advantages of MAS, mainly the last one to allow legacy DM solutions to collaborate towards solving cascading failure on interdependent IoT devices. In the following, we present methods and tools for multi-agent programming and highlight relevant applications of MAS mainly in IoT.

2.3.1 Multi-agent Programming Paradigms and Tools

Developing the MAS requires considering its key properties mentioned above and addressing complex challenges such as learning and security. Several programming paradigms and tools were proposed to fulfill these challenges. Regarding the multi-agent development paradigms, almost five (05) visions were proposed in the literature [Boissier, 2013]: *Agent-oriented programming*; *Interaction-oriented programming*; *Environment-oriented programming*, *Organization-oriented*

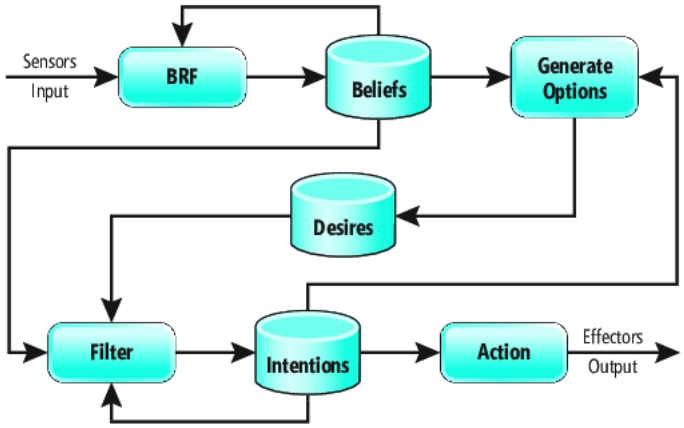


Figure 2.16: The BDI agent architecture [Arnaldo Perez, 2019]

programming, and *Multi-agent oriented programming*. Each of these paradigms has given rise to several tools and software implementations. In the following, we discuss the multi-agent development paradigms with their associated tools:

- *Agent-oriented programming:* this paradigm was introduced by Shoham [Shoham, 1993]. It mainly leverages the *Belief-Desire-Intention* model to program rational autonomous agents. The BDI agent model (see Figure 2.16) aims at programming rational agents based on human mental attitudes of beliefs, desires, and intentions [Bratman, 1987; Silva, 2020]. 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. Many programming languages and tools were developed as an implementation for the BDI model and more generally for the agent-programming model. Among these tools we found *JACK platform* and its language *Jal* [Howden, 2001], *3APL language* [Dastani, 2004], and *AgentSpeak language* and its extension *Jason* [Bordini, 2007].

- *Environment-oriented programming:* This paradigm considers the environment a first-class entity in the MAS. This is because exploiting the MAS environment results in better solutions for complex and dynamic system infrastructure and advanced problem domains such as ad-hoc networks or ubiquitous computing [Weyns, 2005]. More precisely, programming the environment of the MAS can be used to 1) allow agents to access the environment properties and adapt to their changes dynamically; 2) ease agents' access to the external services and resources in the environment; 3) introduce well-designed computational structures that aid agents in their tasks, even extending to moderating and governing agents' interactions to promote organizational and coordination objectives [Ricci, 2011]. Some agent programming tools such as *Jason* and *2APL* have proposed to implement the environment as a single computational object, with a single state. While more valuable efforts

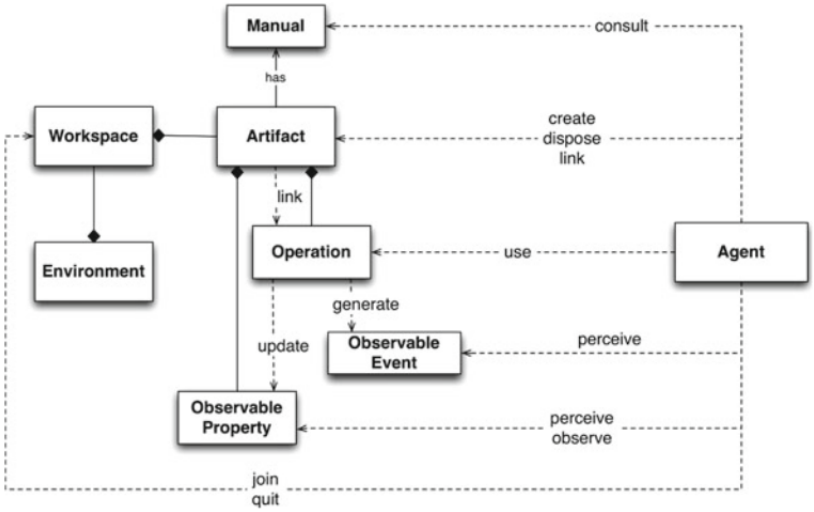


Figure 2.17: The Agent and Artifact Meta-model [Ricci, 2011]

were introduced through the *Agent and Artifact (A&A)* meta-model [Omicini, 2008] (see Figure 2.17), in which the environment is encoded as a dynamic collection of interactive entities referred to as *artifacts*. These artifacts essentially embed various resources and services that agents within the shared environment can collectively utilize and leverage. They expose a set of *Operations* allowing agents to perceive their environment events a.k.a *Observable Events* and act on its properties a.k.a *Observable properties*. These artifacts might be structured within singular or multiple *Workspaces* potentially spread across distinct network nodes. A workspace serves as a domain encompassing one or more tasks that engage a group of agents along with associated artifacts [Ricci, 2011]. The most valuable implementation of the agent environment meta-model *A&A* is the technology *CArtAgO* [Ricci, 2009].

- *Organization-oriented programming:* refers to a paradigm where agents' design, behavior, and interactions are structured around organizational structures and goals. This approach emphasizes establishing a predefined organizational structure such as agent groups and roles with their responsibilities, to achieve specific organizational objectives. Thus, agents are not treated as isolated entities but as organization entities part of an organized collective with well-defined roles and relationships [Hannoun, 2000]. Some effort on organization-oriented programming was presented as *teamwork-oriented programming* proposing several programming tools such as *Karma* [Pynadath, 2003]. Other efforts have adopted the term organization-oriented programming by developing many tools for organization structure description and execution, such as *OPERA* [Dignum, 2004], *MOISE* [Hannoun, 2000] and its extensions *S-MOISE+* [Hübner, 2005] and *MOISE+* [Hubner, 2007]. With these tools, the structure of the agent organization is represented by as set of agent *Groups*. Within a group, each agent member plays a *Role*, Based on its role, an agent performs a set of *Missions* and should respect a set of *Norms*. Agent missions are sub-goals of the global *Goal* the agent organization wants to achieve.

- *Interaction-oriented programming:* was introduced by [Huhns, 2001] to handle in a robust manner the interactions among agents, their environment, and their organizations. In other words, this programming paradigm aims to define how agents exchange information, coordinate actions, and collaborate to achieve collective goals within the MAS, to avoid erroneous behavior and increase reliability. This paradigm was implemented in several agent communication languages such as *Jason* through speech acts theory, where agents inform each other about changes and exchange novel information [Seidita, 2022].

- *Multi-agent oriented programming:* was proposed in [Boissier, 2013], aiming to provide an efficient programming paradigm for complex MAS by bringing together the above-mentioned paradigms considered as dimensions (see Figure 2.18), namely *agent*, *environment*, *organization*, and *interaction*, and keeping their integration alive from design to execution. The interaction dimension defines interactions between the other dimensions. The combination of these dimensions enables a more controllable and effective programming model that is easily extensible and reusable thanks to its modularity. One proposed implementation for this recent paradigm is the *JaCaMo framework* [46] that combines the abstraction of the following tools: *Jason* to implement the agent dimension, *CArtAgO* for the environment dimension and *MOISE* for the organization dimension. More precisely, the abstractions of the agent dimension are related to the Jason meta-model, inspired by the BDI architecture. The environment dimension is based on the CArtAgO meta-model, where the agent environment is described using the *A&A* meta-model. The organization dimension is based on *MOISE* meta-model, describing agent organization structure such as group and roles with their associated missions and norms [Boissier, 2020].

In our work, we have chosen to adopt the multi-agent-oriented programming paradigm, leveraging its implementation through JaCaMo to develop our multi-agent-based solution. This strategic decision enables us to create a MAS that excels in robustness, modularity, and overall coherence, setting the foundation for a highly effective multi-agent solution. More precisely, it allows us to effectively model different aspects of the cascading failure problem, such as modeling the IoT device's environment including their interactions and dependencies through the environment dimension, with the integration of legacy DM solutions using the agent dimension, and a potential usage of the organization dimension to elaborate roles and norms between DM actors when managing cascading failures.

2.3.2 Application of multi-agent system

As mentioned above, a MAS's advantages stem from its decentralized, adaptive, efficient, and collaborative nature. These properties enable the system to tackle complex problems, distribute tasks effectively, and adapt to changing environments, making it a valuable approach in various domains. The MAS may be applied for system modeling and simulation or as software development architecture to solve various problems at run time.

[46] `https://jacamo.sourceforge.net/`

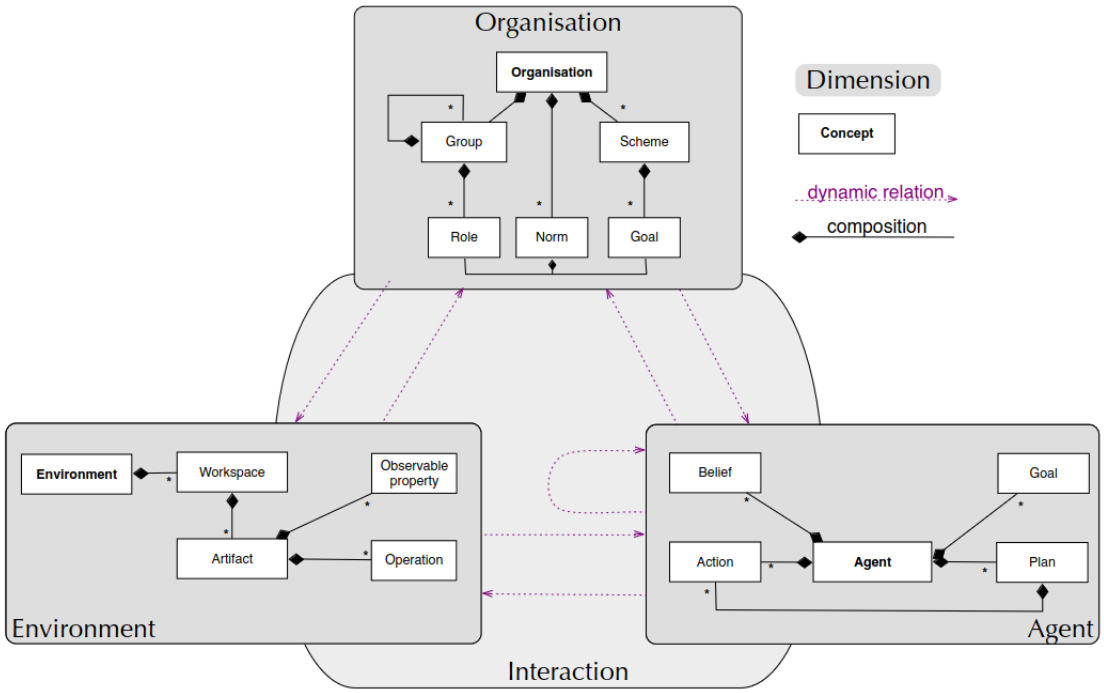


Figure 2.18: A global view of Multi-agent oriented programming dimensions [Boissier, 2020]

A wide range of multi-agent applications in robotics applied to industry, security, and military domains were studied for over two decades [Dorri, 2018]. More precisely, multi-robot systems use MAS for efficient cooperation and coordination between robots and their trajectory planning. Thus allowing them to perform complex dynamic tasks distributed in space, enhance coordination, adapt to feedback, and integrate machining perturbation and imperfection [Pouvreau, 2023; Dong, 2023]. In the context of autonomous vehicles, MAS can enhance traffic flow, manage intersections, and optimize routes for multiple vehicles, reducing congestion and improving overall traffic efficiency [Arel, 2010]. In these applications, the MAS is combined with *Reinforcement Learning* and *Distributed Optimization* to boost agent intelligence and learning capabilities and address specific problem domains.

Another relevant application of MAS is smart grid management, including balancing energy loads, fault detection and isolation, marketing energy, pricing, scheduling energy, reliability, and smart grid network security [Mahela, 2022]. Several works [Ikram, 2022; Babalola, 2016; Babalola, 2014] have leveraged domain-specific MAS to detect and mitigate cascading failures in smart grids using collaborative load regulation.

Moreover, MAS have been used in logistics and supply chains for modeling, optimizing, and managing distributed processes by coordinating the actions of multiple entities in a dynamic and complex environment. For instance, the work investigates using the MAS with game theory for parking management with multiple gates [Noviello, 2023]. In [Jaimez-González, 2021], authors propose a collaborative MAS that explores different strategies of the whole production process with different stakeholders and offers solutions for managing supply chains in distributed e-commerce environments.

2.3.3 Multi-agent system and IoT

MAS can boost the impact of IoT in three forms: First, as IoT seeks to automate operations and forge an intelligent ecosystem where diverse components engage in autonomous interactions, MAS can empower these features to add more autonomy, reliability, and efficiency to IoT architecture [Gheysari, 2022]. For instance, authors in [Singh, 2017] address using MAS to support distributed computing, taking the IoT as a use case. This work argues that building MAS-based IoT solutions allows the integration of multiple stakeholders within the distributed digital process to engage in complex interactions, sometimes over highly constrained resources, easing networking, data management, and analytics. Second, MAS can be used for IoT in the simulation of IoT devices and applications at the test and design stage to identify product defects and increase their quality [Jung, 2018]. Third, a MAS may be used to manage IoT failures, such as to conceive an intrusion detection system [Liang, 2020], as it allows handling distributed and large failure data for IoT failure detection. However, the last use case is slightly addressed in the literature. In our work, we further explore using MAS for IoT management, mainly for handling cascading failures on IoT devices managed by different actors.

2.4 The combination of MAS, DT, and SW standards

It is undeniable that DT, MAS, and SW standards have attracted the research community's interest in recent years. In the previous sections, we have discussed the key properties of each of these technologies independently. In this section, we investigate their amalgamation and how they can be combined to boost the value of each other.

Several efforts have studied the relationship between MAS and DT [Minerva, 2020; Pretel, 2022]. The most common vision is adopting the MAS as a software architecture for building the digital part of the DT. This is because DT and MAS have common properties, such as their goal to represent physical entities through software components. Thus, researchers tried to learn from approaches and tools proposed for MAS to fulfill DT's development requirements, especially for large DT [Minerva, 2020], adding some level of autonomy so that DT features could be exploited by autonomous agents. Moreover, there have been suggestions to use the MAS for enabling the coordination and interaction of multiple DT [Niati, 2020; Pretel, 2022].

On the other side, DT may feed agents with large and real-time data allowing them to make better decisions. In this case, DT are seen as a database artifact part of the multi-agent environment. In this context, the work [Croatti, 2020] introduces the vision of *agent-based DT* considering DT as an effective blueprint for MAS allowing them to conceive and design digital environments mirroring the physical world, providing models to reason about them, and support their decision-making, and cooperation with human users as well.

Within these various visions of combining the DT and MAS, SW standards are used to ease (Big) data integration and reasoning within DT and enable interoperability across heterogeneous agents. Indeed, several works leveraged semantic ontologies to organize the knowledge of the physical asset in their DT, which is fed by heterogeneous and abundant data [Moder, 2020; Boje, 2020; Yu, 2021]. On the other hand, the idea of SW-enabled intelligent agents and MAS has

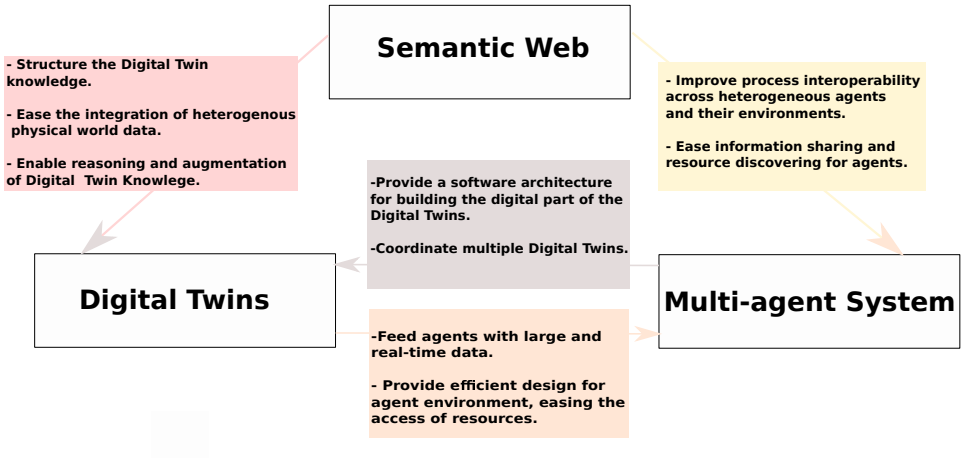


Figure 2.19: The combination of MAS, SW and DT

been around for almost as long as the idea of the SW itself [Ciortea, 2018]; this idea is becoming more active, especially with the emergence of new standards such as Linked Data or SWoT, which aim to improve the interoperability of heterogeneous environments and provide a solid foundation for building effective MAS. Several works were proposed to integrate SW standards into the MAS architecture. Authors in [Schraudner, 2021] introduce a semantic-enabled MAS comprising simple reflex agents tightly integrated into the application layer of the SW. In [Bella, 2022], authors propose the OASIS ontology, an Ontology for Agents, Systems, and Integration of Services, to deliver a higher-level, semantic representation system as well as a communication protocol for agents and their commitments. Moreover, several programming tools were proposed such as *Hypermedea*[47], an advanced version of the JaCaMo multi-agent programming framework, that has been extended to operate within Web and Web of Things settings, enabling it to effectively navigate Linked Data [Charpenay, 2022; Ciortea, 2019]. Recently, *AJAN*[48] a modular framework designed for engineering agents built upon SW standards, was proposed. AJAN offers a versatile execution and modeling environment through web services, complemented by an RDF-based modeling language tailored for deliberative agents [Antakli, 2023].

In our work, we adopt the vision of using the DT as an artifact to empower a MAS knowledge and adaptation. This vision is underpinned by a twofold usage of SW standards: firstly, as an efficient AI tool facilitating the organization and rationalization of knowledge inherent to the DT; secondly, as an interoperability enabler for our MAS approach.

3 Conclusion

In conclusion, this chapter has laid the foundation for the research presented in this Thesis, which aims to address the critical issue of managing cascading failures in interdependent IoT devices that are managed by various actors and siloed IoT device management platforms. The chapter began by providing an extensive background in IoT and IoT device management, drawing from

[47] https://hypermedea.github.io/
[48] https://github.com/aantakli/AJAN-service

industry and research insights. This knowledge serves as a crucial backdrop for the subsequent research and analysis.

Furthermore, we explored the scientific context of our research, delving into key concepts and technologies such as SW technologies, MAS, and DT. We not only introduced these concepts but also discussed their current applications and combinations. This discussion served to justify the technical choices made in the research, laying the groundwork for the subsequent chapters that will delve deeper into the methods, experiments, and findings aimed at achieving the stated research goal.

Chapter 3

State of The Art

Summary

This chapter presents the state of the art of the different research axes to which our work has contributed. Namely, *IoT dependency extraction and modeling*, *Failure management*, *Ontologies for IoT*. We discuss research efforts on these axes and identify research gaps.

Contents

We believe the first step towards automatic management of *Internet of Things* (IoT) cascading failures is identifying dependency relationships among IoT devices, in order to identify the root cause and elaborate a convenient plan for recovering the failed devices. That's why we start by studying related work on IoT dependency extraction and modeling. Secondly, literature on failure management in distributed systems in general and IoT in particular is discussed. We investigate the presented methods for failure detection, diagnosis, and recovery and its consideration of cascading failure. Last, as we relied on ontology-based modeling in our solutions, we studied proposed ontologies for IoT, their scopes, and how they can be reused and combined to

fulfill our solution's specifications, mainly related to the IoT *Cascading Failure Management* (CFM) domain.

1 IoT dependency extraction and modeling

Identifying IoT dependencies is crucial for designing, deploying, and managing IoT systems effectively [Huang, 2016]. An IoT dependency refers to the unidirectional relationship between two IoT devices resulting from their direct and indirect interactions in IoT environments. These interactions may be the exchange of data and services, e.g., connectivity service, between them directly, through IoT applications and platforms, or the physical environment (see Figure 3.1).

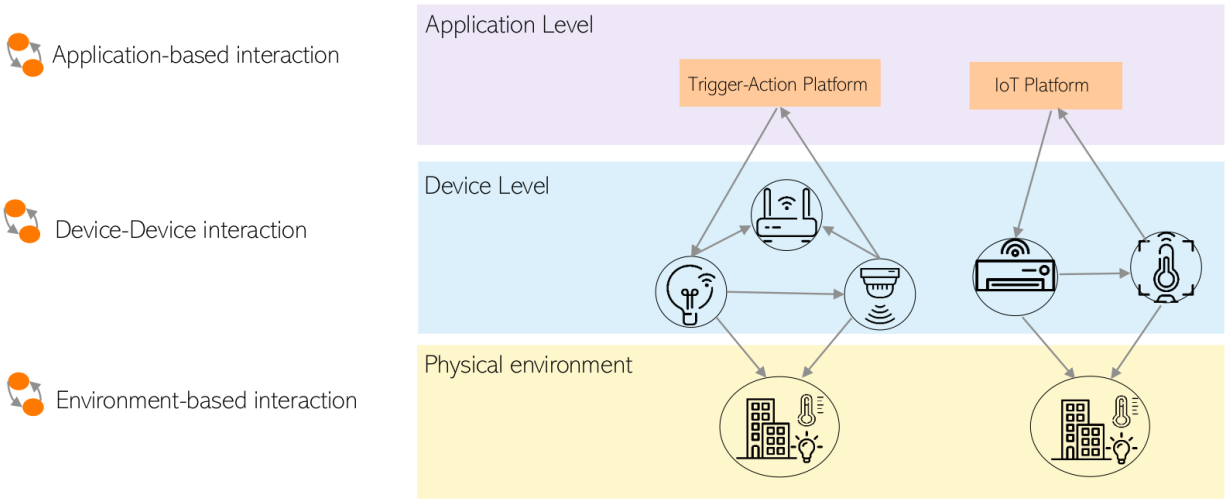


Figure 3.1: IoT interaction model.

Automatically identifying IoT dependencies is a challenging task since they are:

- *Abundant:* due to the large size of IoT systems. Moreover, the number of IoT dependencies is exacerbated by the large adoption of trigger-action platforms by smart home users (27 million users for IFTTT platform). As mentioned in Chapter 2 Section 1.1.1, trigger-action platforms generate many dependencies between IoT devices as they link IoT devices through automation rules, which act on IoT devices based on the state of the others and compose their services.
- *Dynamic:* due to the dynamicity of the IoT system, which is continually evolving in response to shifting conditions and the introduction of new devices.
- *Undocumented:* as IoT dependencies are described in high technical forms. For instance, the direct exchange of services among devices may be described using an API call or within a broker in the case of a publish-subscribe communication model.
- *Heterogeneous:* IoT dependencies are described using heterogeneous data models depending on the technological environment where they are implanted, such as a specific technology of a trigger-action platform or a connectivity broker.

- *Governed by different actors:* since IoT devices are managed by different actors (see Chapter 2 Section 1.2.2), IoT dependencies information are distributed across heterogeneous IoT platforms and connectivity devices governed by different actors that may be operators or service providers.

IoT dependencies extraction and modeling are only partly treated in the literature. Some works discuss models for IoT dependencies, while others are interested in IoT dependency identification and extraction.

To build a tool for security analysis of IoT systems, the work [Mohsin, 2016] proposes a formal model for IoT dependencies and interactions through *Satisfiability Modulo Theories (SMT)*, which is a logic-based model used for system verification. The proposed model for IoT dependencies is used with other SMT models for policy-level behavior and IoT-specific threats to unveil complex chains of hidden attack vectors, given a pre-defined attacker's objectives. This work models *functional dependencies*, where IoT devices use common sensor data and act on the same actuators or when they perform operations over common environmental features, and *network-based dependencies*, where IoT devices depend on network devices for communication. However, it does not consider IoT dependency generated through IoT applications.

Guided by the motivation to establish a benchmark for modeling, developing, and managing IoT devices, the authors of [Huang, 2016] propose a stochastic model to describe and analyze service dependencies among IoT devices. The model is based on *Markov chains*, where each state in the proposed *Markov model* represents an IoT device, and dependencies are represented by transitions between these states. Transition probabilities are extracted through the analysis and normalization of network traffic collected from the router connecting IoT devices. The Markov chain solution, which represents stationary probabilities, allows for modeling the usage time of an IoT device within the system. This information provides insight into critical devices, such as the most frequently used ones. However, this work considers only dependencies extracted from network traffic, which do not allow the identification of indirect dependencies, such as IoT dependencies generated through the physical environment and IoT applications.

The authors in [Mohsin, 2017] propose the *IoTChecker framework*, which is used to automatically detect security misconfigurations in an IoT system. The solution provides a set of semantic ontologies that describe the behavior of IoT devices, access network configurations, and security configurations. A set of Semantic Web Rule Language rules (SWRL) has been proposed to infer the knowledge required to detect security constraint violations. Among the proposed ontologies, we found the *IoT behavior ontology (IoTB)*, describing the functioning context IoT devices, including event-driven transformations, information communication and processing, and interaction between IoT devices and the physical environment. More precisely, *IoTB ontology* considers two types of IoT dependencies: *direct dependencies*, which means one IoT device uses another to perform its function, and *indirect dependencies*, where the actions of one device can have an indirect impact through the environment on the function of its dependent device. However, this ontology does not model IoT dependency generated through IoT applications.

The work [Laštovička, 2017] introduces a theoretical concept for a dynamic graph-based

model, which automatically identifies and extracts IoT dependencies by analyzing network traffic. Graph-based algorithms, such as *Clique detection*, are employed to analyze the graph that depicts IoT dependencies. The main goal of this research is to assess IoT device criticality. These critical devices are those on which a substantial number of dependencies rely. However, like the work [Huang, 2016], this work model only dependencies that can be extracted from network traffic, which do not include IoT dependencies generated through the physical environment and IoT applications.

Recently, authors in [Mariani, 2023] propose a multi-agent learning approach to discover environment-based IoT dependencies, referred to as *Causal Network*. IoT causal networks represent change causality among IoT devices within their shared environment. For instance, recognizing that the states of the air conditioner and the temperature sensor in a room are not simply correlated but that the first causes the second to change. The proposed approach assumes a *Multi-Agent System* (MAS) is deployed in an IoT environment, e.g., a Smart Home. Each agent has a partial view of the IoT causal network learned using Bayesian inference. To unveil the global IoT causal network, agents collaborate with each other according to a collaborative learning protocol. The main purpose of the proposed approach is to allow agents in agent-based IoT deployment to understand their operational environment better, properly decide action plans, and explain such decisions.

On the other hand, many works have proposed solutions for different IoT management and security problems assumed to have IoT dependency information without proposing methods for their extraction and modeling. For instance, the work [Mezghani, 2020] presents a solution for coordinating *Device Management* (DM) operations on IoT devices considering their dependencies to avoid failures during the execution of DM operations on IoT devices. This work assumes having prior knowledge of IoT dependencies among IoT devices. The same hypothesis was adopted in the works [Zdankin, 2021; Xing, 2018; Yu, 2015] proposing solutions for dependency preserving updates in smart home, cascading failure analysis, and cascading attacks management, respectively.

Research Gaps

The landscape of research on IoT dependency modeling and extraction reveals many limitations (see Table 3.1):

- Foremost among these limitations is the pervasive challenge of the accuracy and scalability of existing models designed for IoT dependency. For instance, using Bayesian network-based modeling, while conceptually robust, is encumbered by its reputation as a computationally intensive paradigm. The computational demands imposed by Bayesian network-based modeling often limit its practical utility, particularly within large-scale IoT environments.
- Moreover, the proposed models do not consider modeling direct and indirect interactions between IoT devices and they do not provide a documented model of IoT dependencies despite the work using a semantic ontology, which offers the promise of a richer and interoperable representation of dependency relationships.
- Furthermore, it bears noting that these models frequently fail to address the dynamic aspect of IoT dependencies, where IoT dependency information is extracted and maintained by human intervention. Some works consider periodic extraction of IoT dependencies by analyzing network traffic. However, this does not include indirect dependencies, especially environment-based ones.
- Last, the proposed works do not consider the practical reality: *IoT is managed by multiple actors*, although dependency information is governed by different actors.

Table 3.1: Related Work on IoT dependency extraction and modeling.

Work	Model	Dynamicity	Accuracy	Documentation	Scalability	Heterogeneity	Multi-actor	Dependency Type
[Mohsin, 2016]	SMT	✗	✓	✗	✗	✗	✗	Direct dependency Environment dependency
[Huang, 2016]	Markov chain	✓	✓	✗	✓	✗	✗	Direct dependency
[Mohsin, 2017]	Ontology with SWRL inference rule	✗	✓	✓	✓	✓	✗	Direct dependency Environment dependency Network dependency
[Laštovička, 2017]	Graph Theory	✓	✓	✗	-	✗	✗	Direct dependency
[Mariani, 2023]	Bayesian network	✗	✗	✗	✗	✗	✗	Direct dependency Environment dependency application-based dependency

2 Failure Management in Distributed Systems

Distributed systems are groups of interconnected nodes, which work towards a shared goal. They can be homogeneous e.g., cluster, or heterogeneous e.g., Grid, Cloud, P2P, and IoT, and are subjected to different kinds of challenges such as *Resource Allocation*, *Security*, and *Failure Management* a.k.a *Fault Tolerance* [Ledmi, 2018]. *Fault Tolerance* in distributed systems consists on two elements *Failure Detection* and *Fault handling*.

2.1 Failure Detection approaches

Failure detection in distributed systems consists of monitoring components by collecting relevant information allowing to deduce the failed state [Ozeer, 2019]. Detecting failure in a distributed system is no longer obvious. Indeed, distributed systems, especially large ones, are characterized by a high level of asynchrony, long message delay, and a high probability of message loss, and their topology might change as a result of reconfigurations [Hayashibara, 2002]. Two main models for failure detection are proposed: *Push Model* and *Pull Model* [Hayashibara, 2002; Ozeer, 2019].

- *Push Model:* in this model, the monitored components take an active role, while the monitoring entity (the failure detector) is passive. Each monitored component regularly dispatches messages, typically in the form of heartbeat messages, to the failure detector responsible for overseeing it. The role of the failure detector is to detect a suspected component failure, specifically when it ceases to receive a heartbeat message from the component within a predefined time interval. This absence of a heartbeat indicates a potential crash or failure of the monitored component.
- *Pull Model:* in this model, the monitored components assume a passive role, whereas the monitor or failure detector takes an active stance. The monitor proactively sends inquiries at regular intervals to the monitored components. In response to these aliveness requests, the monitored component acknowledges by sending a reply back to the monitor. However, if the monitor does not receive a reply from a monitored component within a specified time window (timeout), it assumes a potential failure.

2.2 Fault-handling approaches

Once the failure is detected, fault-handling approaches are employed to recover the failed parts. Several fault-handling approaches were proposed in the literature, which are divided into reactive and proactive approaches.

2.2.1 Reactive Fault handling

Reactive fault tolerance methods are employed to mitigate the effects of system failures once they have unexpectedly occurred. These techniques are designed to respond to failures that were not anticipated in advance. The most well-known techniques for reactive fault handling

in distributed systems are *Replication*, *State Saving*, and *Reconfiguration* [Ozeer, 2019]. These techniques may be used independently or combined to ensure fault handling in distributed systems.

- *Replication:* refers to duplicating critical hardware, software, or data across multiple nodes or servers within a distributed system. If a failure occurs on one node, the system can easily switch to a redundant copy, maintaining service availability and data integrity. There are two main classes of replication techniques: *primary-backup replication* and *active replication* [Guerraoui, 1996]. Primary backup replication, also known as active-passive replication, is a fault tolerance technique used in distributed computing systems to enhance system reliability and availability. In primary backup replication, there are two or more identical copies of a service or component, with one designated as the primary and the others as backups. The primary component actively handles requests and operations while the backups remain in a standby state, ready to take over in case the primary component fails. This replication method ensures that the system can continue functioning even when the primary component encounters a failure [Budhiraja, 1993]. Active replication is a fault tolerance technique used in distributed computing systems to ensure the high availability and reliability of services or components. In active replication, multiple copies of a service or component run concurrently, and all of them actively process client requests simultaneously. The Delta-4 is one of the concrete implementations of active replication, representing an open dependable distributed computing systems architecture using a dedicated protocol for coordinating active replicas [Chereque, 1992].

- *State Restoration* consists of saving the correct state of distributed system components and restoring them in case of failure. There are two main approaches for saving information from which the correct state of a failed component in a distributed system can be restored: *Checkpoint* and *Message logging*. Checkpoint refers to a procedure that preserves the current state of a computation by storing it in stable storage. These checkpoints are created at regular intervals during the routine operation of a program. This stored information, including the process's state, its operating environment, register values, and more, is retained in stable storage for potential use in the event of node failures. In case a failure is detected, the process can be reverted to the most recent saved state [Bansal, 2011]. Message logging consists of recording emitted and received messages on a stable storage. Once an entity fails within a distributed system, its messages are replaying in a carefully defined sequence, to reach the pre-failure state [Ozeer, 2019]. There are many proposed techniques for message logging such as *pessimistic message logging*, which operates on the assumption that a failure could happen at any moment. As a result, it takes proactive measures by logging an event onto stable storage before the event is actually processed. This proactive step ensures that all processed messages are reliably recorded on stable storage, eliminating the risk of losing critical data in the event of a failure [Alvisi, 1998].

- *Reconfiguration* refers to the process of dynamically adjusting the configuration of a distributed system in response to changes such as node failures, network issues, or changes

in system requirements. Reconfiguration can involve adding or removing replicas, redistributing workload, or changing the communication patterns within the distributed system. Several approaches were proposed for dynamic reconfiguration for distributed system and applications. The work [Ozeer, 2019] proposes a set of reconfiguration techniques proposed for distributed IoT applications deployed on the Fog nodes. It relies on wrappers that encapsulate software elements and IoT devices involved in IoT applications allowing their reconfiguration. For software elements, the reconfiguration process begins with an initial attempt to restart on the local fog node. In case of failure, alternative involves a placement reconfiguration, which entails the re-initialization of the software element on an alternate fog node. For IoT device failures, the reconfiguration initially entails a reboot of the failed device. If this reboot attempt is unsuccessful, a more extensive architectural reconfiguration is necessary to replace the failed device. In cases where a replacement device is not available, a functional reconfiguration, referred to as operating in a degraded mode, is executed as a fallback measure. Authors in [Castaldi, 2003] introduce *Lira*, a framework for dynamic reconfiguration of distributed system. This framework is designed to bolster the fault tolerance of component-based applications by identifying failures and effecting recovery through on-the-fly reconfiguration, both at the component and application levels. During run-time, the system determines an appropriate new configuration by adhering to a predefined set of reconfiguration policies. The architecture of Lira consists of a collection of agents that interact with managed application components. These agents are responsible for implementing the reconfiguration logic and engage in asynchronous communication with other agents via the *SNMP* management protocol. In [Bouchenak, 2005], the middleware *JADE* is proposed for repair management in distributed system, mainly *J2EE* application server clusters. *JADE* includes several fault tolerance techniques such as replication and dynamic reconfiguration. It relies on the *FRACTAL* component model allowing the reconfiguration of the managed application, by adding and removing replicated components, and updating connections.

2.2.2 Proactive Fault handling

Proactive fault tolerance techniques in distributed systems are strategies and mechanisms implemented to prevent or mitigate potential faults and failures before they occur. More precisely, proactive techniques are forward-looking and aim to anticipate and prevent issues, whereas reactive mechanisms respond to issues after they have occurred. Among these techniques, we find *Software Rejuvenation*, *Load balancing*, and *Preemptive Migration* [Ledmi, 2018].

- *Software Rejuvenation:* is a proactive fault tolerance technique used in distributed systems and other computing environments to enhance system reliability and prevent potential failures. The main idea behind software rejuvenation is to intentionally restart or refresh software components or the entire system before they become susceptible to aging-related failures or issues. This proactive approach helps mitigate the gradual degradation of system performance and stability over time, reducing the risk of unplanned outages or

service disruptions.

- Load Balancing: is a technique used to distribute incoming network traffic or computational workload across multiple nodes, servers, or resources to ensure efficient resource utilization, maximize system performance, and prevent any single component from becoming a bottleneck. The main function of load balancing is to achieve optimal resource allocation and maintain high availability of services while minimizing response time and preventing overloading of individual components, which prevent failures in distributed systems.

- Preemptive Migration: is a proactive fault tolerance technique used in distributed systems to enhance system reliability and availability by preemptively relocating resources or services before a failure occurs. This technique aims to mitigate the impact of potential failures by moving resources away from problematic conditions or unstable components. Preemptive migration can be particularly valuable in scenarios where the cost of downtime or system unavailability is high, such as in cloud computing, data centers, and critical infrastructure systems.

The existing approaches for fault detection and handling in distributed systems may be adapted to manage IoT failures, mainly managing IoT application failures [Ozeer, 2019]. However, failure management in IoT systems raises a set of unique challenges regarding traditional distributed systems. These challenges have been discussed in [Norris, 2022] and include: 1) the diversity of IoT failures, which is related to the IoT heterogeneity; 2) IoT failures are highly contextual depending on the physical environment; 3) IoT failures are characterized by their cascading behavior; 4) IoT failure management must be performed with minimum human intervention. To address these challenges, many failure management approaches were proposed for the IoT, which will be discussed in the following.

3 Failure Management in IoT

As described in Chapter 2 Section 1.1.1, IoT integrates physical processes with digital connectivity, often using three components represented by devices, connectivity, and IoT platforms. While failures may occur in any of these components, IoT platforms and connectivity networks are less likely to fail. This is because they are carefully designed, deployed, and monitored thanks to the reliability of advanced technologies such as *Cloud computing* and *5G/6G networks*. Conversely, IoT devices, are characterized by constrained computational resources, environmental rigors, software intricacies, and power limitations, making them more likely exposed to failure [Norris, 2022]. Furthermore, these frequent failures on IoT devices are exacerbated by the *cascading failure phenomenon*. Namely, owing to the interdependencies among IoT devices, the failure of a single device can potentially trigger a hazardous and cascading sequence of failures in others. Thus, managing IoT device failures is key to ensuring the reliability and efficiency of IoT systems to provide better *Quality of Experience* (QoE). In our work, we are interested by managing failures on the IoT device level with particular interest on cascading failures.

IoT failure management on IoT devices consists of three steps: *Failure detection*, *Failure diagnosis* for failure type and root cause identification, and *Failure recovery* [Nishiguchi, 2018]. These steps must be carefully considered and integrated together in order to build an end-to-end IoT failure management solution able to automatically handle failures on IoT devices [Norris, 2022].

Before discussing research work on IoT device failure detection, diagnosis, and recovery, we present taxonomies for IoT device failures. After that, we discuss failure management capabilities in market DM platforms and customer care services. Finally, we highlight research gaps in IoT failure management in both research and industry.

3.1 IoT device failures

Several taxonomies were proposed in the literature for IoT device failures [Norris, 2022; Chakraborty, 2018; Sharma, 2010; Ozeer, 2019]. To the best of our knowledge, there is no single standard reference taxonomy. This is due to the heterogeneity and diversity of IoT devices, including software and hardware diversity, which result in various failure types. In the following, we discuss failure taxonomies proposed in the literature. Then, we present our taxonomy for IoT device failures that aggregates and enriches the state-of-the-art taxonomies.

The work [Nishiguchi, 2018] proposes a taxonomy of failures in smart home environments. Failures are classified into two categories: *Equipment failures*, represented by hardware and software failures, and *Network failure* such as interference between connectivity devices. While authors in [Power, 2020] adopt a taxonomy proposed for distributed systems [Cristian, 1991] to describe IoT device failures, represented by: *Omission* when an IoT device stops sending and receiving messages. *Crash* when an IoT device stops totally. *Timing* when an IoT device's response arrives too early or too late. *Response*, when the value of an IoT device response or the state transition occurs, is incorrect. *Arbitrary* when an IoT device sends an arbitrary response at an arbitrary time.

Some works focus on failures related to data generated by IoT devices. This class of failure is widely discussed in the literature due to the importance of data accuracy, especially in critical environments (e.g., nuclear power plants). Additionally, these failures occur relatively frequently. For instance, the work [Shih, 2016] mentions that a temperature sensor has more than 15% false measurements. This failure class has the following sub-classes [Choi, 2018; Chakraborty, 2018] (see Figure 3.2): *High Variance*: when a device oscillates between states faster than the environment dictates; *Stuck-at*: when a device fails in reflecting changes in the environment; *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 an 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. The work [Norris, 2020] proposes a taxonomy for IoT device failure based on the failure impact: *Fail-Stop failures* occur when a device stops functioning and no longer responds to external requests; *Non-fail stop failures* occur when a device exhibits an incorrect state, either by producing incorrect sensor readings or not responding correctly to a request; *Cascading failures* occur when a failed IoT device triggers failures on other devices.

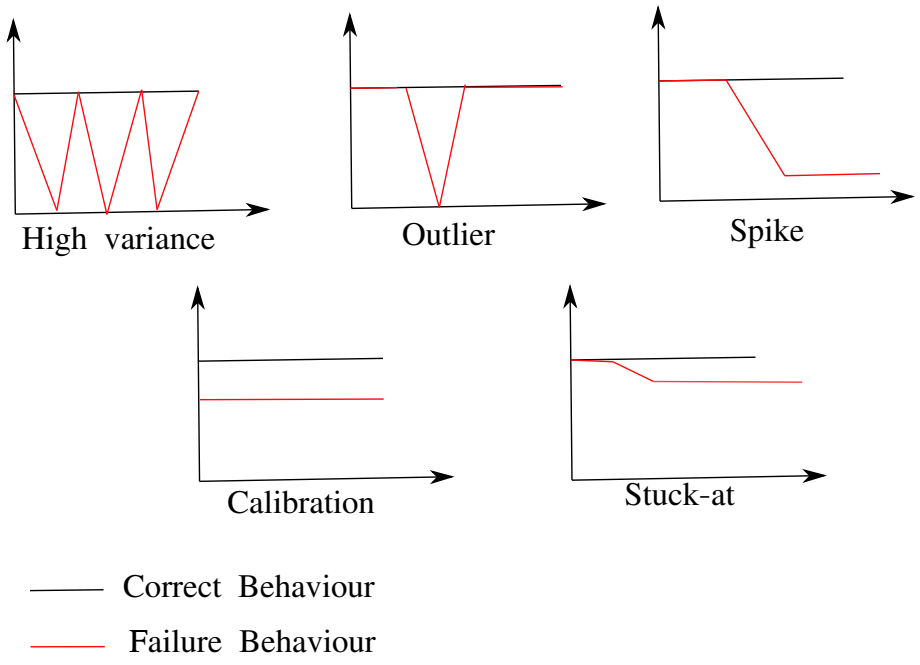


Figure 3.2: Sensor Failures

The authors [Ayeb, 2020b; Mezghani, 2020] discussed failures occurring during the execution of DM operations on IoT devices. These failures referred to as *DM failures*, may be related to connectivity issues so that DM operations such as updates can not be executed properly. They are exacerbated by the chaotic and uncoordinated execution of DM operations by multiple DM platforms on interdependent IoT devices [Mezghani, 2020].

Based on these taxonomies, we propose a global taxonomy for IoT failures presented in Figure 3.3, covering failures on IoT devices: IoT device failures are classified into three categories: *single failure*, *cascading failure*, and *DM failures*. Single failures are classified into *fail-stop* and *non-fail stop failure*. The former includes the typical failure of distributed systems, namely *omission*, and *crash*. The latter includes the different failure classes related to sensor data explained above, namely *high variance*, *stuck-at*, *outlier*, *calibration*, and *spike*. Our work considers the management of single and cascading failures on IoT devices managed by different DM platforms. Managing DM failures is out of the scope of our work. We refer the reader to the works [Ayeb, 2020b; Ayeb, 2020a] on DM failure management.

3.2 IoT failure management from research perspective

3.2.1 Failure Detection

Most academia efforts on IoT failure management focus on IoT failure detection, mostly known as *Anomaly Detection*, which detects failures and often notifies users to act themselves [Norris, 2022]. Several failure detection frameworks, such as SMART [Kapitanova, 2012] and IDEA [Kodeswaran, 2016], rely on the analysis of *Activities of Daily Living* as a way to detect problems or failures in IoT devices. For example, if a smart thermostat fails to adjust the temperature as expected during the daily routine, the system might detect this anomaly as a potential

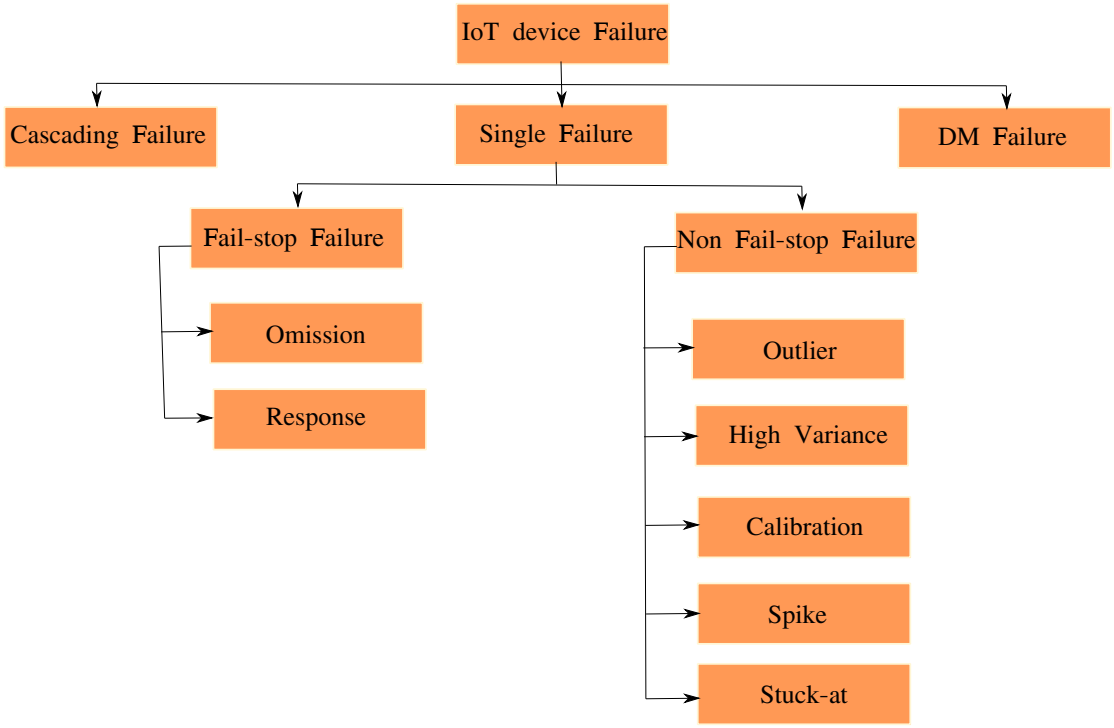


Figure 3.3: Taxonomy of IoT device failures.

IoT device failure. Other approaches rely on network traffic analysis to detect failure on IoT devices [Najari, 2021; Najari, 2022; Portela, 2023], which mostly build the normal behavior of IoT devices through their network traffic. Then, the current behavior of the IoT device is compared to the normal one at run time to detect anomalies.

Within these approaches, anomalies are detected using even geometrical, statistical, or machine learning-based methods. Geometrical methods, known also as proximity-based methods, rely on density-based and distance-based techniques to separate anomalous data points, which appear in sparse regions compared to the normal data. Statistical methods consist of building a statistical model of what is considered "normal" behavior within the dataset. This often involves calculating statistical measures such as mean, standard deviation, or percentiles. This statistical model is used to establish filters or thresholds that define the range of values within which most data points should fall. Data points that fall outside these predefined ranges are considered anomalies. Machine learning-based, unlike statistical-based approaches that rely on predefined statistical measures, learn patterns directly from the data, making them more adaptable and suitable for more complex scenarios [Giannoni, 2018].

The adopted machine learning model for anomaly detection depends on several characteristics of the managed IoT system, such as the type of training data such as audio and time series, and the availability of training labels. For instance, long short-term memory (LSTM) and transformer models are the best choice for audio data [Chatterjee, 2022].

Machine learning-based anomaly detection is an active research area, where *441 works* have been published on this topic in 2022 according to the *Web of Science* [1]. The proposed works

[1] http://webofscience.com/

address several problems such as the privacy and distribution of learning data [Wang, 2023] and learning the normal behavior from data including anomalies [Najari, 2022].

3.2.2 Failure Diagnosis

IoT failure diagnosis aims to not only detect but also identify failure types and perform failure root cause identification. 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. Then, this model is used to estimate device output. The differences between the estimated outputs and the measured outputs are monitored to detect failures and determine their type [Lazarova-Molnar, 2016; Chi, 2022a]. Observers are one of the output estimators that are effective for failure diagnosis. A nonlinear observer-based failure diagnosis method is introduced in [Subramaniam A, 2018] for diagnosing a wide variety of actuator failures in HVAC systems. An adaptive observer is used in [Teng, 2021] to determine the location of failed sensor and actuator failures in Wind Turbines. However, observers are limited in handling random noises or uncertainties in model inputs [Chi, 2022a]. To fill this gap, stochastic models mainly filter-based ones were proposed for output estimation combined with statistic testing. For instance, authors in [Alsabilah, 2021] combine Kalman filter and Shapiro-Wilk test to diagnose cyber-attacks in smart home networks. Despite filters provide high accuracy models with low computation [Chi, 2022a; Alsabilah, 2021]. They are still not adapted for complex systems such as large-scale IoT systems [Chi, 2022a].

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. Signal-based methods utilize measured signals rather than explicit input-output models for failure diagnosis [Gao, 2015a]. They consist of real-time analysis of a set of measured signals such as vibration, noise, or pressure using signal processing algorithms e.g., Fourier transforms [Najeh, 2019; Chi, 2022a]. In [Giantomassi, 2014a; Giantomassi, 2014b], authors proposed a signal-based diagnosis approach for smart home by monitoring the trend of ambient temperature using Kernel Canonical Variate Analysis and Multi-Scale Principal Component Analysis, respectively. The main limitation of the signal-based method is the negligence of system dynamic inputs, whose diagnosis performance may be degraded under unknown input disturbances or unbalanced conditions [Gao, 2015b]. Machine learning-based approaches leverage sensor device data (named also historical data) to build models for failure identification and characterization [Chi, 2022a]. Several algorithms were proposed for IoT failure diagnosis to detect and determine faulty devices and the type of failure: Support Vector Machine [Li, 2019], Artificial Neural Network [Borhani, 2022], Fuzzy Neural Network [Chen, 2017], Bayesian Network [Zhang, 2018]. However, the availability and quality of learning data of all possible failure types of large-scale systems such as IoT systems could be impossible [Wilhelm, 2021].

The work [Chi, 2022a] argues that knowledge-based failure diagnosis is especially well-suited for complex or multi-actor systems for which detailed mathematical models are not available and diagnosis learning data are governed by different actors. These approaches consist of a knowledge base (KB) that contains failure diagnosis information provided by experts at the

design or the operational level of devices. Failure diagnosis information describes, according to failure symptoms, failure types with their possible recovery actions. Experts-based fed of failure diagnosis KB ensures more accurate diagnosis results [Chi, 2022b].

The common limitation of existing approaches for failure diagnosis is that they only consider failure type identifications, neglecting failure root source identification, which is an important process especially when dealing with cascading failures. Failure root cause identification has been addressed in the context of IoT reliability modeling and analysis, which refers to the process of building a mathematical model to represent the system failure criteria in a logical manner and then evaluating the model to obtain the system reliability metric [Xing, 2020; Fu, 2021]. However, these approaches are proposed to enhance the design of IoT systems to prevent cascading failure not to enable failure root cause identification at run time for cascading failure mitigation. The work [Nishiguchi, 2018] proposes a theoretical framework for network failure diagnosis on IoT devices by developing plugins to discover the network topology in IoT systems. However, the authors have not proposed any concrete implementation of these plugins. To the best of our knowledge, there are no failure management solutions for IoT that include concrete failure root cause identification features.

3.2.3 Failure Recovery

Failure recovery consists of performing a set of operations on failed IoT devices to bring them to their correct state. To this end, some technical frameworks have been proposed in the literature: Authors in [Nishiguchi, 2018] propose a framework using recovery actions according to the failure cause such as *device replacement* or *recharging* when the failure cause is device battery failure. The proposed recovery actions include *device replacement*, which consists of replacing the failed device with other devices even by replicating devices or discovering devices providing similar services as the failed device; *device reconfiguration* to deal with DM failures; and typical DM operations such as *update*, and *reboot*.

In [Norris, 2020; Norris, 2022], authors propose the IoTRepair framework, a solution that automates failure recovery on IoT devices. The solution provides a set of failure recovery functions based on the type of failure and device, as well as a failure manager that combines recovery functions to fix complex failures. Among the proposed failure functions, we find *reboot*, *update*, and other failure tolerance techniques such as *device replacement*. The author proposes an adapted *checkpointing with rollback* for IoT devices, which consist of saving the correct states of IoT devices and restore these states in case of failures. The correct state of an IoT device is defined by its outputs. For instance, an IoT system equipped with a presence sensor, motion sensor, and light functionality can feature a Checkpoint represented as ["Presence": 1, "Motion": 0, "Light": 1]. This configuration signifies that Presence has been detected, Motion is not detected, and the Light is turned on. Sensor states, which encompass environmental data, are immutable and collectively define the system's current state. Conversely, actuator states are modifiable through actuation and constitute the elements that can be rolled back.

Authors in [Ozeer, 2019] propose a framework to recover failures in IoT applications deployed in Fog devices. The proposed framework relies on *device replacement* as the main function to

recover IoT applications using failed devices. Device candidates for device replacement are discovered from the Orange digital twin system *Thing'in* (see Chapter 2 Section 2.2.2).

The existing frameworks rely mainly on device replication and replacement to recover from failure, which is costly and not effective [Norris, 2022]. Moreover, they assume having central control of all devices that they integrate through APIs, neglecting the practical reality: *IoT is managed by multiple actors using siloed DM platforms*, where device and failure information are governed by different DM actors.

DM operations are key recovery actions for IoT devices enabling automatic and remote failure recovery. Nonetheless, there are instances where these automated procedures fall short, necessitating human intervention to restore normal device functionality, such as when remote recovery actions fail or are not supported by the devices.

3.3 IoT Failure management from industrial perspective

In practice, 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[2] on the FM capabilities of these siloed solutions. As a result, we identify the following profiles with distinct failure management capabilities, including *Device Manufacturers* (MN), *DM Platform Provider* (DMP), and *Service Provider* (SP) (see Figure 3.4).

- MNs build and deliver IoT devices to end users and IoT suppliers. Mostly, MNs do not have DM platforms to perform recovery 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) [Emmanouilidis, 2020]. 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 [Steenwinckel, 2018].

- 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 [3] or alarm-based system [4]. 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.

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

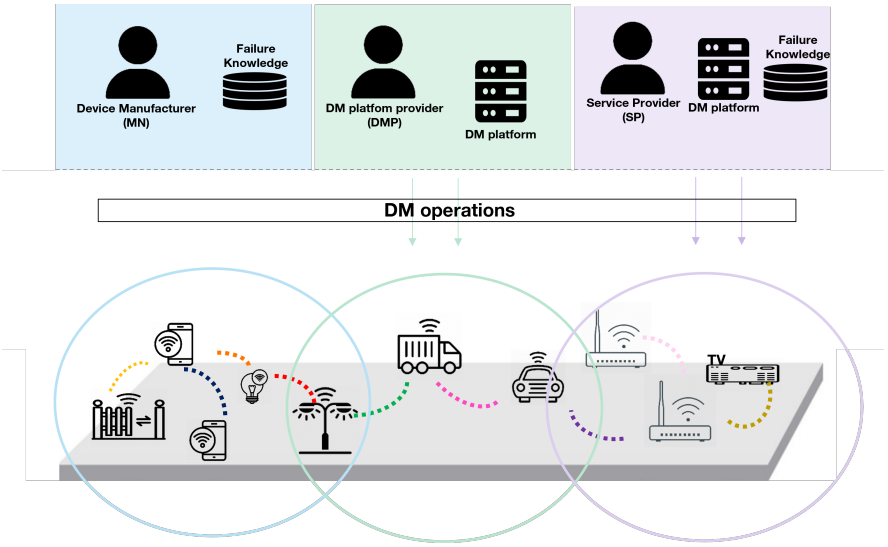


Figure 3.4: Failure management profiles.

- 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.

As a result, in Market failure management solutions mainly DM platforms, there is a pressing requirement for an end-to-end failure management solution tailored for IoT devices. This imperative arises from the fact that actors within this ecosystem possess only partial knowledge concerning failures, and they exert a degree of control over interdependent IoT devices. To address this complexity effectively, there is an acute need to facilitate collaborative efforts among these actors, with the main objective of resolving failures, particularly those of a cascading nature, in an automatic and effective manner.

Research Gaps

Numerous limitations can be discerned within the existing research approaches of IoT failure management, as delineated in Table 3.2.

- In the area of failure detection, most relevant approaches rely on the application of Machine Learning models. However, these data-driven strategies confront many constraints due to their dependence on the availability and veracity of learning data, describing the behavior of IoT, a large, complex, and highly heterogeneous system.
- In terms of failure diagnosis, knowledge-based methods have emerged as a promising avenue for the accurate identification and description of IoT device failures. However, these approaches neglect to incorporate failure root cause identification features.
- Turning to the realm of failure recovery, a myriad of frameworks has been advanced. Nonetheless, they tend to neglect the IoT device heterogeneity and the multiplicity of connectivity protocols governing their operation, particularly when orchestrating remote recovery actions. Furthermore, these frameworks disproportionately rely on device replacement as the principal action for recovering failures, an approach that is not only cost-prohibitive but also fraught with inefficacy and raises a problem of sustainability.
- It is worth noting that although several approaches have been proffered for failure detection, diagnosis, or recovery, none have tried to holistically integrate these facets into a comprehensive and end-to-end framework for IoT failure management.
- An oversight prevalent in existing approaches is the absence of a concerted strategy for managing cascading failures afflicting IoT devices. Notably, there exists a work that tentatively addresses cascading failures through the deployment of checkpointing and rollback recovery techniques, necessitating the preservation of the precise state of all IoT devices.
- Moreover, it is imperative to underscore that all the existing approaches operate under the implicit assumption of possessing total control over IoT devices, thereby overlooking the complex governance structures wherein these devices, in practice, are managed by distinct actors with varying degrees of authority and influence.
- From the industrial perspective, market failure management solutions are missing holistic and end-to-end approaches for failure management on IoT devices, which underscores the need of collaborative failure management efforts.

Table 3.2: Related Work on IoT Failure Management

Work	Detection	Diagnosis	Recovery	Cascading Failure	Multi-actor
[Kapitanova, 2012]	✓	✗	✗	✗	✗
[Giantomassi, 2014a; Giantomassi, 2014b]	✗	✓	✗	✗	✗
[Kodeswaran, 2016]	✓	✗	✗	✗	✗
[Gao, 2015a]	✗	✓	✗	✗	✗
[Lazarova-Molnar, 2016]	✗	✓	✗	✗	✗
[Gao, 2015b]	✗	✓	✗	✗	✗
[Chen, 2017]	✗	✓	✗	✗	✗
[Subramaniam A, 2018]	✗	✓	✗	✗	✗
[Nishiguchi, 2018]	✓	✗	✓	✗	✗
[Najeh, 2019]	✗	✓	✗	✗	✗
[Li, 2019]	✗	✓	✗	✗	✗
[Zhang, 2018]	✗	✓	✗	✗	✗
[Ozeer, 2019]	✓	✗	✓	✗	✗
[Alsabilah, 2021]	✗	✓	✗	✗	✗
[Norris, 2022; Norris, 2020]	✗	✗	✓	✓	✗
[Najari, 2021]	✓	✗	✗	✗	✗
[Borhani, 2022]	✗	✓	✗	✗	✗
[Teng, 2021]	✗	✓	✗	✗	✗
[Portela, 2023]	✓	✗	✗	✗	✗

4 Ontologies for IoT

In order to allow legacy DM platforms to automatically and collaboratively manage cascading failures on IoT devices, heterogeneous DM platforms need to be able to understand each other through a common language. Semantic ontologies are key to allowing these features (see Chapter 2 Section 2.1.1). In the following, we first present reference ontologies in IoT. Then, we discuss existing ontologies and investigate their capabilities to model IoT failure and cascading failure management domains, represented by modeling dependency relationships between IoT devices and their failure information.

4.1 Ontologies of reference in IoT

- SAREF [5] [García-Castro, 2023]: firstly developed, in close interaction with the industry, to represent smart appliances with the aim to facilitate the management of energy and services on smart appliances managed by different stakeholders [Daniele, 2015]. It is supported by the European Commission and then adopted by the European Telecommunications Standards Institute as a reference ontology for IoT. It was built in a bottom-up fashion based on market data models for smart appliances [Seydoux, 2018]. It has several extensions [6] each cover a specific domain such as *SAREF4SYST* for systems, connections, and connection points and *SAREF4ENER* for the energy domain.

- SSN [7] [Hitzler, 2019]: proposed by W3C Semantic Sensor Network Incubator group to model sensors and observations, it allows representing the properties and measurement capabilities of sensors to provide a generic description of these devices. It also serves to describe the data collected by the sensors and the context of their acquisition. It provides a standardized and semantically rich framework for describing and modeling sensor data and sensor-related information framework that different sensor networks and IoT platforms can adopt [Compton, 2012].

- oneM2M Base [8]: documented in the Technical Specification TS-0012[9]. It defines the core concepts involved in the oneM2M standard architecture such as Device, Service, and Variable. The oneM2M standard represents a global standard for IoT that provides a common platform and framework for developing and deploying IoT solutions.

- *IoT-Lite* [10]: a lightweight ontology that instantiates the old version of SSN ontology named SSNX to allow the representation of IoT elements and resources to enable interoperability and efficient discovery of sensor data in heterogeneous platforms. The IoT-Lite core's straightforwardness and adaptability contributed to its integration into two European

[5] https://saref.etsi.org/
[6] https://saref.etsi.org/extensions.html
[7] https://www.w3.org/TR/vocab-ssn/
[8] https://git.onem2m.org/MAS/BaseOntology/
[9] https://www.onem2m.org/images/pdf/TS-0012-Base_Ontology-V3_7_3.pdf
[10] https://www.w3.org/submissions/iot-lite/

projects, FIWARE [11] and FIESTA-IoT [12] [Seydoux, 2018].

- *IoT-O* [13]: serves as a foundational IoT ontology, focusing on capturing fundamental concepts related to IoT systems and applications. Its primary purpose is to represent broad, cross-cutting knowledge about IoT, with the flexibility to incorporate specialized, application-specific information as needed. To design IoT-O, a set of well-defined ontologies are reused, such as SSN, SAN [14], and MSM [15].

4.2 Ontologies for IoT dependency Modeling

As discussed in the previous section, several ontologies were proposed to model multiple aspects of IoT systems. The standardized ontology SSN[16] is considered the more valuable effort in this area. Other ontologies have extended this ontology to model other aspects of IoT systems, IoT resources in *IoT-Lite ontology*,[17] and IoT system evolution with power constraints in *IoT-O ontology*. The *SAREF ontology* is another widely acknowledged ontology enabling semantic interoperability for smart appliances. However, these efforts do not consider modeling interactions and dependencies within IoT systems. The work [Mohsin, 2017] proposed a network of ontologies to assess security risks in IoT systems. Among them, the *IoTB ontology* describes dependencies between IoT devices. Despite the IoTB ontology covering a set of IoT dependencies, it doesn't consider indirect dependencies such as application-based dependencies. This last have been partially modeled in the *EuPont ontology* [18] [Corno, 2017], which allows the representation of interactions between IoT devices within IoT application and automation rules.

4.3 Ontologies for IoT Failure Modeling

Previous works have demonstrated that the use of semantic modeling is of interest for failure information, reflecting the need for dynamic failure knowledge composition and utilization. Indeed, failure information within DM platforms and customer care services is generally described in the form of tables generated using risk assessment methods such as FMEA, which provide prior information gathering expert-driven insights into the system. However, many people are involved in this process, resulting in disambiguations and incompleteness. This motivates the research community to leverage this failure information in order to define a domain-specific ontology based on the FMEA concepts. Thus, allowing such knowledge to be more complete, accurate and efficiently exploitable at operating time. Several ontologies have been proposed for a range of assets such as Wind Turbines [Zhou, 2015], Loaders [Xu, 2018], Rotating Machinery [Chen, 2015], Production Printers [Emmanouilidis, 2020], and NORIA-O ontology [19] for IT

[11] https://www.fiware.org/
[12] https://cordis.europa.eu/project/id/643943
[13] https://www.irit.fr/recherches/MELODI/ontologies/IoT-O.html
[14] https://www.irit.fr/recherches/MELODI/ontologies/SAN.html
[15] https://lov.linkeddata.es/dataset/lov/vocabs/msm
[16] https://www.w3.org/TR/vocab-ssn/
[17] https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/
[18] https://elite.polito.it/ontologies/eupont.owl
[19] https://orange-opensource.github.io/noria-ontology/NORIA-O/doc

Table 3.3: Related Work on Ontologies for IoT

Ontology	Dependency Modeling			Failure Modeling
	Direct Dependency	Application-based Dependency	Environment-based Dependency	
SAREF	● ○ ○	○ ○ ○	● ● ○	○ ○ ○
SSN	○ ○ ○	○ ○ ○	● ● ○	○ ○ ○
SOSA	○ ○ ○	○ ○ ○	● ● ○	○ ○ ○
IoT-Lite	● ○ ○	○ ○ ○	○ ○ ○	○ ○ ○
IoT-O	● ○ ○	○ ○ ○	● ● ○	○ ○ ○
EuPont	● ○ ○	● ● ○	○ ○ ○	○ ○ ○
oneM2M Base	○ ○ ○	○ ○ ○	○ ○ ○	○ ○ ○
IoTB	● ● ●	○ ○ ○	● ● ●	○ ○ ○
Folio	○ ○ ○	○ ○ ○	○ ○ ○	● ● ○

network [Lionel Tailhardat, 2022], Cyber-physical system [Sanislav, 2017; Sanislav, 2019; Ali, 2018], Wireless Sensor Network [Benazzouz, 2014], and Folio ontology[20] for IoT systems [Steenwinckel, 2018]. The latter is the only proposed work to describe IoT failures, to the best of our knowledge. However, expressive classification for IoT failure behaviors, recovery actions, and failure symptoms is missing in this ontology.

Research Gaps

Numerous ontologies have been proposed to model various aspects of the IoT. However, they are limited regarding modeling the IoT cascading failure domains (see Table 3.3). One significant limitation of existing IoT ontologies is their failure to adequately capture direct and indirect dependencies and interactions among IoT devices, which is crucial in identifying root causes of failure. Regarding direct dependencies modeling, some ontologies such as *SAREF* allow the description of IoT devices and their exposed services which allows partial representation of direct dependencies. However, they do not provide semantics related to the usage of these exposed services by other devices. Semantics describing IoT device's interaction within IoT applications have been partially considered only in the *EuPont ontology*. While environment-based dependencies have been modeled in several ontologies through the representations of the environment elements and their properties. Furthermore, the existing reference IoT ontologies do not consider modeling failure information related to IoT devices, which were partially handled by the *Folio ontology*. Thus, existing ontologies need to be semantically enriched and combined to address the modeling of both direct and indirect dependencies and failure information.

[20] https://github.com/IBCNServices/Folio-Ontology

5 Conclusion

In conclusion, this chapter has set the stage for our research journey into the automatic management of cascading failures in IoT systems managed by different DM actors. We began by emphasizing the critical importance of identifying dependency relationships among IoT devices as the initial step in resolving these complex issues and developing effective recovery plans. To do so, we embarked on a thorough exploration of the existing body of research in this area.

Our study encompassed two main dimensions: first, we delved into the realm of IoT dependency extraction and modeling, aiming to gain insights into the methodologies and techniques available for understanding the complex dependencies among IoT devices. This foundational knowledge is pivotal for root cause identification and recovery planning.

Secondly, we ventured into the broader domain of failure management in distributed systems, with a particular focus on IoT. We examined the existing literature to uncover the various methods and approaches proposed for failure detection, diagnosis, and recovery, paying special attention to their applicability in the context of cascading failures.

Finally, as our research heavily relies on ontology-based modeling, we conducted a literature review of the ontologies proposed for IoT. This exploration encompassed the scopes and purposes of these ontologies and how they can be effectively reused and combined to align with the specific requirements of our solution, particularly in the realm of IoT cascading failure management.

By addressing these essential aspects in this chapter, we have identified key research gaps to support our contributions, which will be presented in the next chapters.

Chapter 4

Inferring Threatening IoT Dependencies using Semantic Digital Twins

Summary

This chapter presents our proposed solution for IoT dependency inference. First, we show our modeling for IoT dependencies. Then, we shed light on the proposed framework for automatic and collaborative inference of IoT dependencies from DM solutions managed by different DM actors. We illustrate our approach in a smart home use case based on the *Digital Twin* (DT) platform *Thing In The Future* (Thing in).

Contents

Inferring IoT dependency relationships among *Internet of Things* (IoT) devices is crucial for *Cascading Failure Management* (CFM) in order to identify the failure root cause and recover the system effectively [Xing, 2020]. It is also a key element in unlocking several IoT issues such as *Device Management* (DM) failures [Mezghani, 2020] and dependency-related attacks [Yu, 2015].

We refer to these issues as *Dependency-related threats*, and to dependencies generating these threats as *threatening dependencies*.

As elaborated upon in the previous section (see Chapter 3 Section 1), the task of inferring threatening IoT dependencies is highly intricate. It necessitates the identification of a dynamic and extensive array of dependencies, drawing from heterogeneous data sources maintained by diverse DM actors. The review of existing literature, as detailed in Chapter 3 Section 1, exposes several research gaps in the domain of IoT dependency modeling and extraction. Existing models and approaches, in terms of both accuracy and scalability, leave much to be desired. Additionally, these models often fail to adequately address the dynamic nature of IoT dependencies, which are frequently managed through manual interventions. Moreover, they neglect that dependency information is distributed across siloed and heterogeneous DM solution governed by different DM actors.

In this chapter, we propose a collaborative framework that infers and characterizes the topology of threatening dependencies by accessing and aggregating data from legacy DM solutions. It combines the assets of *Semantic Web* (SW) (see Chapter 2 Section 2.1) and *Digital Twin* (DT) (see Chapter 2 Section 2.2) technologies to capture on-demand the topology of dependencies, and it is designed to be integrated into legacy DM solutions within *customer care services* of DM actors to boost decision making and enhance customer *Quality of Experience* (QoE). Our framework relies on Orange's DT platform Thing in (see Chapter 2 Section 2.2.2) for federation and information sharing across DM actors, as well as to implement DT features.

The present chapter is organized as follows. We present motivating examples to illustrate dependency-related threats in the smart home scenario presented in Chapter 1 Section 2.1. Then, we give our model for threatening dependencies and the proposed framework. Finally, we present extensive qualitative and quantitative evaluations of the proposed contributions.

We highlight that the contributions of this chapter have been published in [Guittoum, 2023b; Guittoum, 2023c] and presented in *Eclipse IoT Days 2023* [1].

1 Motivating Examples

We rely on the smart home use cases presented in Chapter 1 Section 2.1 to illustrate the dependencies-related threats that are difficult to overcome with siloed DM solutions.

Example 1. (Cascading Failure) *As a reminder, cascading failure corresponds to the case where a failure of a device triggers the failure of some of its dependent devices. Let us take the scenario where the temperature sensor fails and starts to return high-temperature values. This failure referred to as* Spike failure *(see Chapter 3 Section 3.1). Due to dependencies between devices, this failure propagates to the temperature dependent devices and services (see Figure 4.1): 1) the air conditioner becomes inoperable (see Rule 1 and 3), 2) the two windows reflect failed behavior swapping from closed (see Rule 3) and open (see Rule 4) states, 3) the alarm turns on announcing fire is detected (see Rule 4 in Table 1.2), 4) the light bulb is inexplicably set to red due to the alarm failure (see Rule 7 in Table 1.2), 5) the door is unexpectedly unlocked upon the*

[1] https://wiki.eclipse.org/Eclipse_IoT_Day_Grenoble_2023

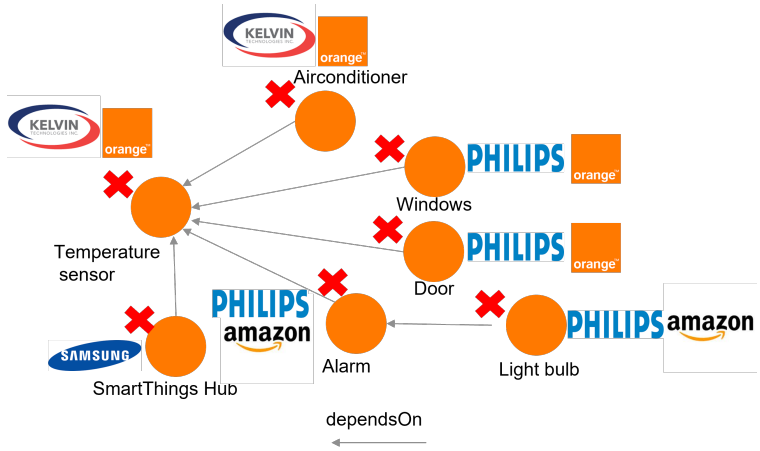


Figure 4.1: The presented cascading failure scenario

false detection of fire (see Rule 4 in Table 1.2), and 6) SmartThings hub can no longer control the chaotic event generated due to automation rules. These failed devices are managed by different DM actors (see Table 1.1): Orange manages the temperature sensor, the windows, the door, and the airconditioner. Amazon manages the alarm and the light bulb, while Samsung manages the SmartThings hub. Moreover, these devices are built by different manufacturers Kelvin for the temperature sensor and the airconditioner and Philips for the windows, the door, the alarm, and the light bulb. These siloed actors cannot identify the failure root cause since they do not have knowledge of dependency relationships among IoT devices.

Example 2. (DM failures) *DM failures correspond to the case where processing a DM operation on a device triggers the failure of its dependent's devices. Let's suppose Orange reboots the gateway while Amazon updates the vocal assistant. Due to the connectivity dependency, the gateway reboot interrupts the vocal assistant's Internet connection, which results in the firmware image not being downloaded correctly and the vocal assistant failing [Mezghani, 2020].*

Example 3. (Dependency-related Attacks) *Attacks related to device dependencies correspond to the case where an attacker exploits device dependencies to compromise IoT systems. For instance, an attacker can exploit the state dependency between the airconditioner and the windows (see Rule 2 in Table 1.2) to gain access to the home by disabling the airconditioner to open the windows [Yu, 2015]. Siloed DM actors need to be aware of these threatening dependencies in order to create dependency-aware security policies to defend against dependency-related attacks.*

As a first step toward solving and preventing these threats, DM actors need a decision-support framework that allows extracting the dependencies between IoT devices, considering their abundant character and their dynamic aspects. We can see that dependencies can be deduced from the functional contexts of IoT devices, such as service exchange between devices, connectivity topology on the gateway[2], and automation rules described in the SmartThings Hub. This information is heterogeneous and distributed across siloed DM actors. In the following, we

[2]Connectivity topology of a connectivity device such as gateway or Wi-Fi repeater represents the list of devices connected to it.

will show how this information can be automatically extracted, unified according to a shared model, and leveraged to infer the global threatening dependency topology.

2 Context-Based Modeling for Threatening Dependencies

This section illustrates our interoperable approach for modeling threatening dependencies. We explain how we evolved from a characterization of IoT dependencies by analyzing dependencies-related threats to the *Internet of Things Dependency* (IoT-D) ontology, which provides a comprehensive and interoperable representation of threatening dependencies and their context.

2.1 Threatening Dependencies Characterization

We conducted an analysis study of dependencies generating the different threats illustrated in Chapter 4 Section 1. The result of this study is a taxonomy of threatening dependencies (See Figure 4.2). We distinguish two types of threatening dependencies: direct and indirect.

Dependencies are direct when IoT devices use direct services of each other. We call this type of dependencies *Service dependencies*. For example, the alarm using the smoke sensor's detection service has a service dependency on the smoke sensor. A special kind of service dependencies is *Connectivity dependencies* when IoT devices use connectivity services of connectivity devices such as a gateway or Wi-Fi repeater. Interactions between sensors and actuators through the

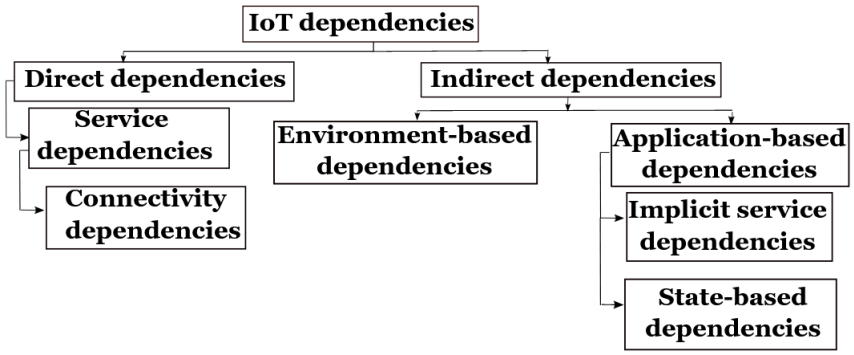


Figure 4.2: Threatening dependencies taxonomy.

physical environment create indirect dependencies between them called *Environment-based dependencies*. For example, the temperature sensor has an environment-based dependency on the airconditioner because it measures the room temperature modified by the airconditioner. Indirect dependencies can also arise from applications running on top of IoT devices, thus forming *Application-based dependencies*. Indeed, an automation rule applies actions to a set of devices depending on how the state of other devices changes, creating *State-based dependencies*. For example, an automation rule may open the two windows depending on the state of the air conditioner whether it is active or not (see Rule 2 in Table 1.2). In addition, IoT applications generate an implicit exchange of services between IoT devices. For example, Rule 1 in Table 1.2 uses the temperature value returned by the temperature sensor to adjust the airconditioner. Here

the airconditioner implicitly uses the temperature sensor service. We call these dependencies *Implicit service dependencies*.

State-based dependencies, explicit and implicit service dependencies exacerbate cascading failure propagation, as illustrated in *Example 1*. Meanwhile, connectivity dependencies generate failures during the execution of DM operations when not respected, as demonstrated in *Example 2*. State-based and environment-based dependencies are used to develop attacks on user security, as shown in *Example 3*.

2.2 Threatening Dependencies Data Sources

Threatening dependencies described in Chapter 4 Section 2.1 can be derived from a set of information that we call *context data*. Context data refer to, but are not limited to, connectivity topology, exchange of services between devices, or applicative automation rules. It is distributed across siloed DM data sources governed by different DM actors and following heterogeneous data models.

Example 4. (Dependency context data) *Let us take the use case as an example: Assuming Orange and Amazon DM platform are using the USP DM protocol (see Chapter 2 Section 1.2.1). In this case, device services and their interactions within the physical environment are distributed across Orange and Amazon DM platforms and represented in the USP controller according to the data model* TR-181,[3] *which enables the representation of the* IoT capabilities[4] *of a given device. Interactions at the IoT application level reside in the SmartThings platform in the form of automation rules represented using the dedicated data model.*[5] *The topology of connectivity resides in the gateway managed by Orange, where it is described in the USP controller according to the data model* TR-181*, which enables the representation of the connectivity topology discovered using the standard* IEEE 1905.1 *[standard, 2013].*[6]

To obtain the global topology of threatening dependencies, this heterogeneous context data must be represented according to a unified data model. To build such a model, we rely on an *Ontology*, which is a key enabler of data sharing and interoperability across siloed organizations and systems (see Chapter 2 Section 2.1.1).

2.3 Threatening Dependencies Modeling

We propose an ontology called *Internet of Things Dependency* (IoT-D) (shown in Figure 4.3) that allows an interoperable representation of threatening dependencies context data in the form of *Knowledge Graph* (KG), as well as the modalities allowing their extraction. The context data KG is used to infer the *IoT Dependency Knowledge Graph* (DKG). The dependency relationships are represented by inferred object properties [7] in the form *iotd:has*DependencyTo*, where (*)

[3] https://usp-data-models.broadband-forum.org/

[4] https://usp.technology/specification/14-index-iot-data-model-theory-of-operation.html

[5] https://developer-preview.smartthings.com/docs/automations/rules

[6] https://cwmp-data-models.broadband-forum.org/tr-181-2-11-0.html#D.Device:2.Device.IEEE1905.

[7] Inferred object property results from classical inferences that happen as soon as we use ontology-oriented knowledge graphs.

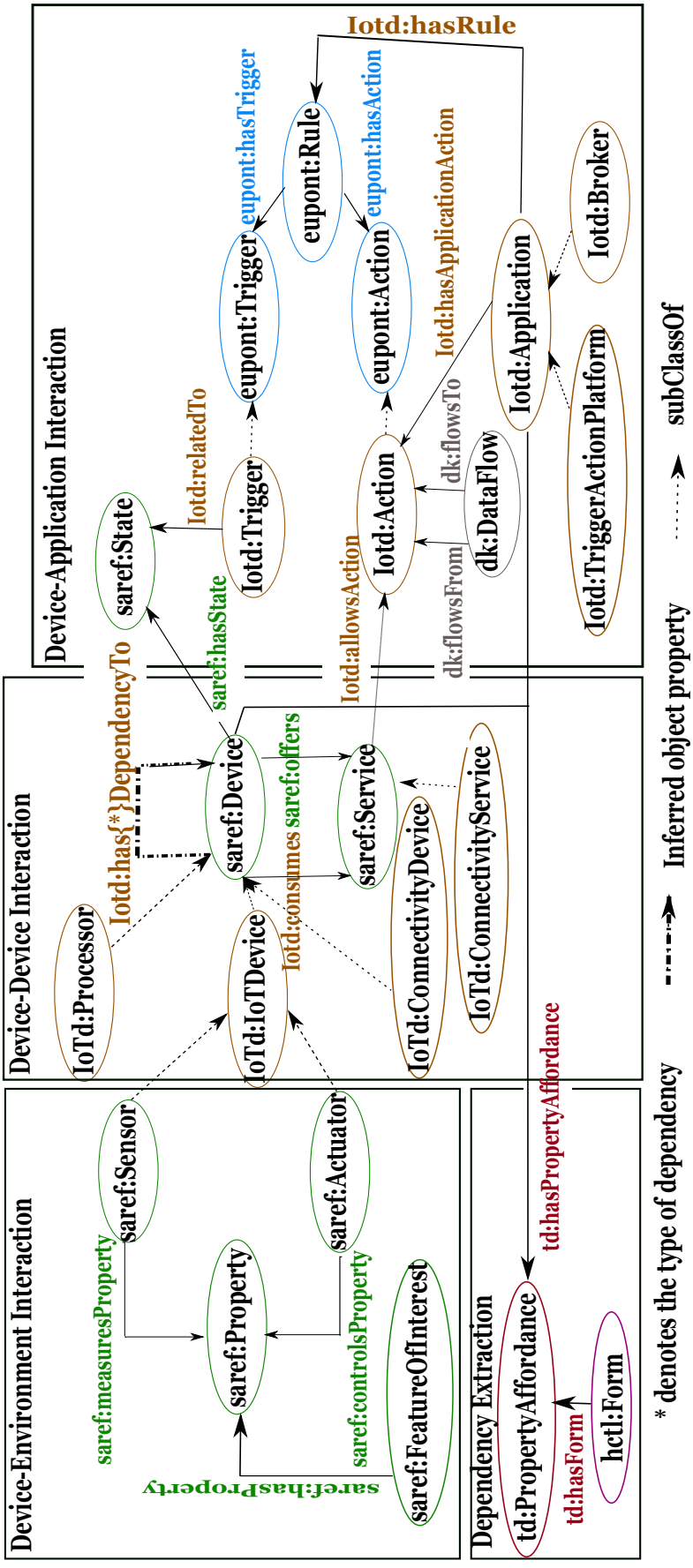


Figure 4.3: IoT-D ontology.

refers to the type of dependency relationship. This context-based representation allows for rich documentation of threatening dependencies to support decision-making when addressing dependencies-related threats. For instance, when a cascading failure occurs, in addition to the topology of dependencies helping to identify the source of the failure, this representation allows the identification of the services that caused the failure and even similar devices that could replace the failed device. Moreover, using the KG-based model for IoT dependencies allows for efficient analysis of large, heterogeneous, structured, and unstructured dependency context data [Fensel, 2020].

The IoT-D ontology is designed using the ontology engineering methodology NeOn [Suárez-Figueroa, 2012] that considers ontology engineering best practices, such as reusing existing ontologies and modularization. It **extends** the standardized ontology SAREF[8] (see Chapter 3 Section 4.1), with the **reuse** of *Data Knowledge ontology (DK)*[9], *End user Programming ontology (EuPont)*[10], and the TD ontology (see Chapter 2 Section 2.1.1). It includes four modules:

1. *Device-Device Interaction module:* describes the capabilities of IoT devices in terms of service provisioning and usage to model the context of direct dependencies. It is based on the enrichment of the SAREF ontology by describing the direct exchange of services between devices through the relation *iotd:consumes* and specializing the *saref:Service* and *saref:Device* to account for connectivity devices and services, allowing the inference of service and connectivity dependencies between devices.

2. *Device-Environment interaction module:* represents, using the SAREF ontology, the interaction of devices within the physical environment through sensing and actuation, enabling the inference of environment-based dependencies. More precisely, the variables of the environment such as *living room* are represented through the class *saref:FeatureOfInterest*. Their properties such as *living room temperature* are represented through the class *saref:Property*. Interactions of IoT devices with these properties are modeled using the object properties *saref:measuresProperty* and *saref:controlsProperty* for sensing and actuating interactions respectively.

3. *Device-Application interaction module:* describes device interactions in IoT applications. Based on the EuPont ontology, an IoT application is represented using rules and actions. A rule is in the form **if** *iotd:Trigger* **then** *iotd:Action*. Triggers are related to device state changes (*saref:State*). Actions are executed by IoT services (*saref:Service*). This trigger-action-based model allows representing state dependencies between devices, i.e., when an IoT application acts on one device based on the state of another. Also, it allows the representation of implicit service dependencies context in the form of data flows e.g., *temperature measurements* between *iotd:Action*, using the class *dk:DataFlow* of the DK Ontology.

4. *Dependency Extraction module:* describes modalities allowing the proposed framework to

[8] https://saref.etsi.org/core/v3.1.1/
[9] http://www.data-knowledge.org/dk/1.2/index-en.html
[10] https://elite.polito.it/ontologies/eupont.owl

extract the dependency information, described in the other modules, from DM solutions. It leverages the TD standard with the class *td:Property Affordance* to represent dependency data endpoints in an interoperable manner. Dependency data endpoints are *REST APIs* allowing to access context data from DM solutions. Dependency extraction modalities will be explained more with relevant examples in Chapter 4 Section 3.1.

Note that the IoT-D ontology conceptualization is representative of IoT systems with no regard to the application domain (e.g., smart home, smart city), which makes it reusable in a wide scope of domains [Seydoux, 2018]. Moreover, this ontology can be easily extended to consider other types of IoT dependencies thanks to its modular design. We make available online the documentation of the IoT-D ontology. [11] [12]

3 Proposed Framework

Relying on the IoT-D ontology, we propose a framework (see Figure 4.4) that allows on-demand inference of threatening dependencies owned by siloed DM solutions. The framework relies on the DT platform Thing in to expose the inferred DKG and to communicate with DM actors. It involves three main steps that rely on SW standards, namely *Context extraction*, *Entity resolution*, and *Dependency inference*. The first step extracts the context data from legacy DM solutions and transforms it into KGs, the second aggregates the extracted context KGs, and the last infers threatening dependencies topology described by the DKG from the aggregated context KGs. These steps are detailed in the following.

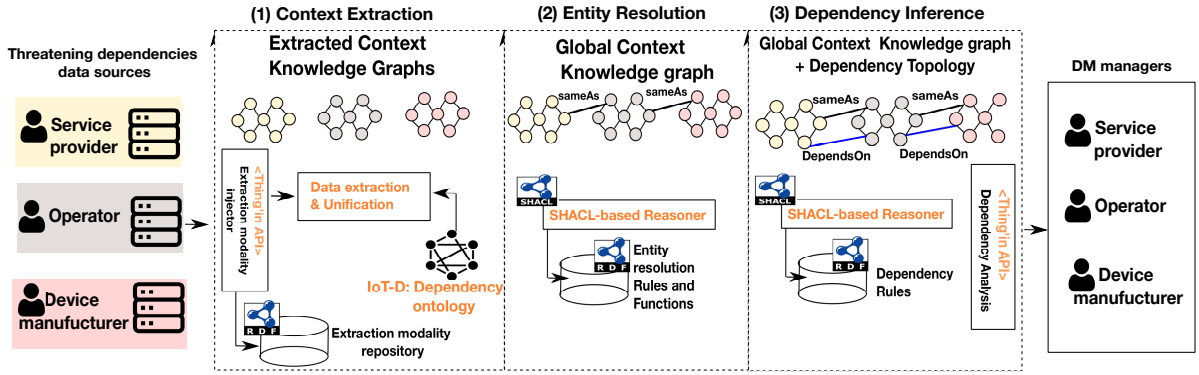


Figure 4.4: Framework overview - On-demand inference of threatening IoT dependencies.

3.1 Step 1: Context Extraction

This step aims to extract the context data from the siloed DM solutions and transform it into KGs according to the IoT-D ontology. We rely on the TD standard to describe the extraction modalities that allow context data extraction from DM solutions. An extraction modality includes information about the data to be extracted, such as the URL of the extraction and the

[11] https://w3id.org/iotd
[12] https://github.com/Orange-OpenSource/ISWC-IoT-D-ontology-Documentation

format e.g., *json*. It is provided to the framework by DM actors through a federative data injection enabler of the Thing in platform (see Figure 4.5). It is stored in Thing in to be used by the context extraction step. Note that: 1) using the TD standard enables technology-agnostic data extraction, which eases the integration of heterogeneous DM solutions, 2) the extracted context data is operational data extracted from reliable DM solutions so it does not contain outliers.

```
1  /*Declaration of the extraction data source here is the gateway*/
2  :Gateway rdf:type
3  iotd:ConnectivityDevice;
4  td:hasPropertyAffordance[
5  td:hasForm [
6  /*Definition of information about the extraction data*/
7  hctl:forContentType
8       "application/json" ;
9  hctl:hasOperationType
10      td:readProperty ;
11 /*Definition of the extraction link*/
12 hctl:hasTarget
13              "{$USPLink}/dataModel=Device.IEEE1905.NetTopology."
14 ]].
```

Listing 4.1: Connectivity topology extraction modality.

Example 5. (Context extraction) *Let us consider the extraction of the connectivity topology from the gateway managed by Orange: the operator Orange injects into Thing in the extraction modality described in Listing 4.1. The context extraction step uses this modality to extract the connectivity topology from the gateway by accessing the operator's USP controller, using the link presented with the property* hasTarget *(lines 11-13). The extracted data described in the TR-181 model (see Listing 4.2) is then transformed into KG and stored in Thing in.*

```
1  [ {"requested_path":
2  "Device.IEEE1905.NetTopology.",
3  "resolved_path_results": [
4  { "resolved_path":
5    "Device.IEEE1905.NetTopology.",
6   "result_params": [
7    {
8    "param_name": "IEEE1905Device.1.FriendlyName",
9    "value": "TempSensor"
10    },
11    { "param_name":        "IEEE1905Device.2.FriendlyName",
12    "value": "bulb1"
13    },
14    { "param_name":   "IEEE1905Device.3.FriendlyName",
15        "value": "LightSensor"
16     } ...
```

Listing 4.2: Part of the extracted connectivity topology.

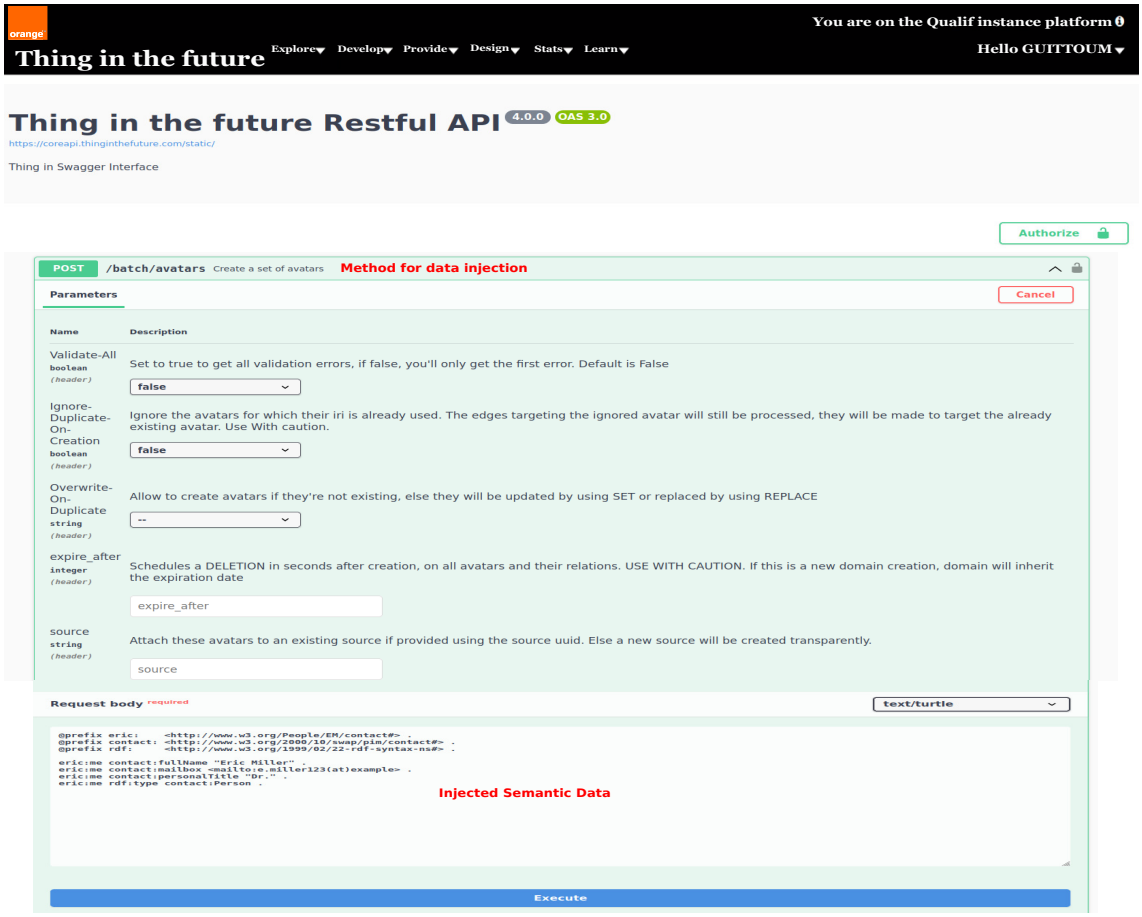


Figure 4.5: Thing in Data Injection Enabler

3.2 Step 2: Entity Resolution

3.2.1 Problem Statement

As an IoT device may be managed by multiple DM solutions [Jia, 2021], the extracted context KGs may contain duplicate entities, such as devices with different representations (see Figure 4.6). For example, the temperature sensor may be named *tempSensorModelX* in Orange gateway, while being registered as *TempSensor* in Samsung's SmartThings Hub. These duplicated representations must be identified and resolved to allow consistent reasoning across the extracted KGs. This problem is referred to in the literature as the *Entity Resolution* (ER) problem, which consists of aggregating similar entities in data extracted from different sources to increase data quality [Saeedi, 2020].

The ER problem has been widely studied for over 70 years in different domains such as knowledge fusion and social network reconciliation [Saeedi, 2020; Li, 2020a; Leitão, 2013]. It is also treated in the KG domain for KG completion, where several heuristics are mainly based on ML [Mugeni, 2023]. However, for cases where training data is unavailable, which is our case, non-learning approaches are suitable [Christophides, 2020]. The authors propose in [Benbernou, 2021] a non-learning ER approach based on SWRL rules that leverages a set of functional keys, which represents a set of attributes allowing the identification of similar entities. However,

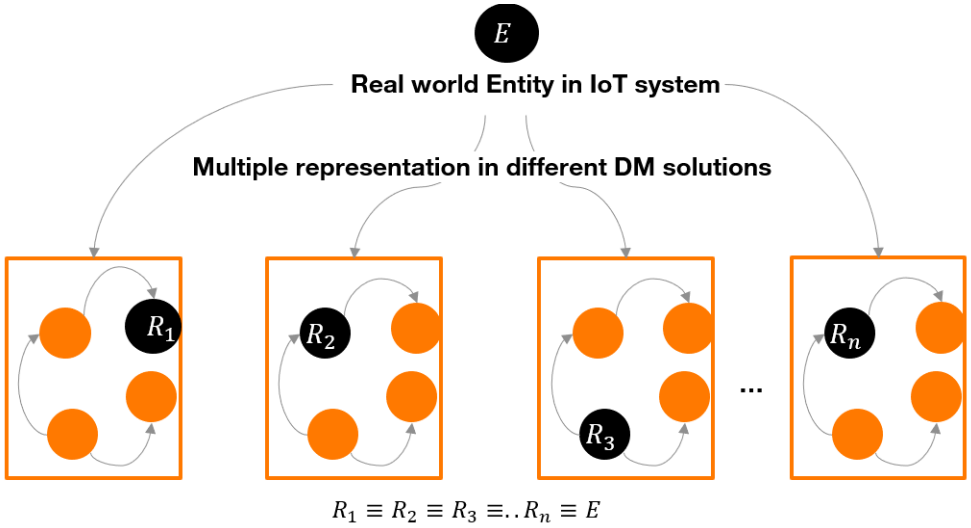


Figure 4.6: Illustration of the ER problem.

this approach does not consider complex similarity computation between the functional keys. This work has inspired us to propose a novel non-learning ER approach based on the advanced features of the *Shapes Constraint Language* (SHACL) standard (see Chapter 2 Section 2.1.1), which allow embedding complex similarity computations in ER inference rules. In the following, we formally describe the ER problem for our case. Then, we present our proposed SHACL-based ER approach.

Entities to resolve in our case are instances of *iotd:IoTDevice* and *SAREF:Service*, since they are shared entities among the siloed DM solutions. More formally, consider n DM solutions, a context KG extracted from the i^{th} DM solution is a tuple:
$\mathcal{KG}_i = (\mathcal{E}_i, \mathcal{R}_i, \mathcal{A}_i, \mathcal{L}_i, \mathcal{T}_i)$, where $\mathcal{E}_i$ is the set of entities, $\mathcal{A}_i$ the set of data properties, $\mathcal{R}_i$ the set of object properties, $\mathcal{L}_i$ the set of literals and $\mathcal{T}_i$ is the set of triples. We distinguish data triples $\mathcal{T}_A$ and object triples $\mathcal{T}_R$, where $\mathcal{T}_A : \mathcal{E} \times \mathcal{A} \times \mathcal{L}$ are triples linking entities and literals, and $\mathcal{T}_R : \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ link entities. Our goal is to build the global context KG i.e., $\mathcal{KG}_g$, by identifying and linking similar entities in the extracted context KGs. The $\mathcal{KG}_g$ consists on the union of the extracted context KGs: $\mathcal{KG}_1, \mathcal{KG}_2, ...\mathcal{KG}_n$ enriched by similarity object triples $\mathcal{T}_{RS} : \mathcal{E} \times \{owl : sameAs\} \times \mathcal{E}$ that links similar entities, and the object property $owl : sameAs$, i.e.,

$$\mathcal{KG}_g = (\bigcup_{i=1}^{i=n} \mathcal{E}_i, \bigcup_{i=1}^{i=n} \mathcal{R}_i \cup \{\text{owl:sameAs}\}, \bigcup_{i=1}^{i=n} \mathcal{A}_i, \bigcup_{i=1}^{i=n} \mathcal{L}_i, \bigcup_{i=1}^{i=n} \mathcal{T}_i \cup \mathcal{T}_{RS})$$

3.2.2 Method

To determine the $\mathcal{KG}_g$, we rely on an inference rule-based ER approach using the advanced features of the *SHACL standard*: SHACL rule and SHACL function (see Figure 4.7).

First, each entity to be resolved in the extracted context KGs is automatically annotated by a set of resolution attributes.

Definition 1 (Resolution attribute) a resolution attribute RA describes a DM metadata that allows entity identification across the siloed DM solutions. For instance, *device serial*

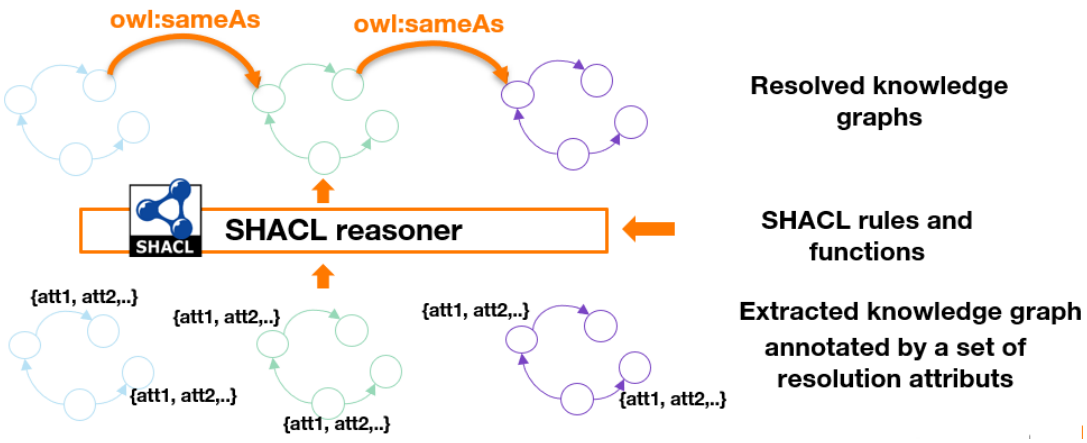


Figure 4.7: Illustration of the proposed SHACL-based ER approach.

number is a resolution attribute for IoT devices. These resolution attributes are extracted from DM solutions during the context extraction step and represented in the context KGs as literals with associated data properties such as *iotd:hasSerialNumber*.

An annotated context KG is described by $\mathcal{KG}'_i$:

$$\mathcal{KG}'_i = (\mathcal{E}_i, \mathcal{R}_i, \mathcal{A}_i \cup \{\text{ iotd:hasRA}_k\}, \mathcal{L}_i \cup \{\text{RA}_k\}, \mathcal{T}_i \cup \mathcal{T}_{RA})$$

$\forall e \in \mathcal{E}_i$, e instance of iotd:IoTDevice or e instance of SAREF:Service, $\forall RA_k$ a resolution attribute extracted form the i^{th} DM solution and $\mathcal{T}_{RA} = (e,\ \text{iotd:hasRA}_k, RA_k)$, represents data triples that link entities e with their resolution attributes RA_k. After the annotation process, a SHACL Rule is used to build the $\mathcal{KG}_g$ by performing the ER on the annotated context KG i.e., $\mathcal{KG}'_i$. This rule automatically infers the *owl:sameAs* relationship between similar entities. SHACL functions allow the ER SHACL rule to perform similarity computations among entities in the extracted KGs using the resolution attributes. The similarity between two entities is a weighted sum of similarities between their resolution attributes. The similarity between two resolution attributes is calculated using similarity functions.

Definition 2 (Similarity function) a similarity function sim: $\mathcal{L}^2 \rightarrow \mathbb{R}^+$ is a string similarity function associated with a given resolution attribute e.g., *strict string similarity* for device serial number and *Jaro similarity* for device manufacturer name.

Definition 3 (Resolution attribute weight) a resolution attribute weight $w \in \mathbb{R}^+$represents the impact of the resolution attribute in the resolution, such as 0.9 for device serial number and 0.5 for device name.

To optimize the number of created *owl:sameAs* object properties, we perform the ER only between the KGs extracted from the DM providers and the KGs extracted from other actors i.e., service providers and device manufacturers, since DM providers acquire information about all IoT devices and their services. The attribute property *orgIoT:source* [13] is used to define from which actor an entity is extracted.

[13] An attribute property related to the ontology Orange IoT used in Thing in to manage Digital Twins.

More formally, consider the entity e_i extracted from the service providers or the device manufacturer DM solution. It is linked to the set of resolution attributes $\mathcal{RA}^i$ using a set of data properties $\mathcal{A}^i$. Resolving this entity consists of finding its most similar entity e_p extracted from the DM provider's DM solution that satisfies: $\max \sum_{A_k \in \mathcal{A}^i \cap \mathcal{A}^p} w_k \cdot \text{sim}^k \left(RA_k^i, RA_k^p\right)$, such as $\mathcal{RA}^p$ is the set of resolution attributes linked to e_p using the data properties $\mathcal{A}^p$, sim^k and w_k are the similarity function, and the weight associated with the resolution attribute RA_k respectively. We consider the intersection between the resolution attribute data properties sets since DM solutions may contain different resolution attribute types.

```
iotd:EntityResolution
  rdf:type sh:NodeShape ;
  sh:targetClass iotd:IoTDevice ;
  sh:rule [
  rdf:type sh:SPARQLRule ;
  sh:construct """
/*Construct the sameAs relationship between the similar device's representations*/
  CONSTRUCT {
   $this owl:sameAs ?device .
            }
/*Constraints to check before building the sameAs relationship*/
/*Constraints are similarity evaluation between device's representations based on the
    resolution attributes*/
    WHERE {
    /*For each device representation ($this), find its most similar representation
       (?devices) */
        {
         SELECT $this ?device WHERE {
         {
/*Calculate the similarity by calling SHACL functions*/
            SELECT $this
         (MAX(0.5*iotd:similarityFunction(?n2, ?n1)+0.1*iotd:
            similarityFunction(?mn2, ?mn1)
          +0.9*iotd:similarityFunction(?sn2, ?sn1)) AS ?val) WHERE
           {
/*Select the resolution attributes used in the similarity calculation */
             $this orgIoT:source ?s .
             OPTIONAL {$this iotd:hasDeviceName ?n2 .}
             OPTIONAL {$this iotd:hasManufacturerName ?mn2 .}
             OPTIONAL {$this iotd:hasSerialNumber ?sn2.}
             ?device a iotd:IoTDevice .
             ?device orgIoT:source ?src .
             OPTIONAL {?device iotd:hasDeviceName ?n1.}
             OPTIONAL {?device iotd:hasManufacturerName ?mn1.}
             OPTIONAL {?device iotd:hasSerialNumber ?sn1.}
             FILTER (?device!=$this && ?s="other" && ?src="DMProvider")
             }
             group by $this
```

```
36          }
37              OPTIONAL {$this iotd:hasDeviceName ?n2 .}
38              $this orgIoT:source ?s .
39              OPTIONAL {$this iotd:hasManufacturerName ?mn2 .}
40              OPTIONAL {$this iotd:hasSerialNumber ?sn2.}
41              OPTIONAL {?device iotd:hasDeviceName ?n1 .}
42              OPTIONAL {?device iotd:hasManufacturerName ?mn1 .}
43              OPTIONAL {?device iotd:hasSerialNumber ?sn1.}
44              ?device orgIoT:source ?src .
45        Filter(?device!=$this && ?s="other" && ?src="DMProvider" &&
46      (0.5*iotd:similarityFunction(?n2, ?n1) +0.1*iotd:similarityFunction
           (?mn2, ?mn1)
47      +0.9*iotd:similarityFunction(?sn2, ?sn1))=?val)} } }""";] ; .
```

Listing 4.3: SHACL rule for Entity Resolution.

Concretely, consider the SHACL rule presented in Listing 4.3. It performs ER between IoT device representations using the resolution attributes: *Device Name, Manufacturer Name, and Serial number*. Their weights are 0.5, 0.1, and 0.9, respectively, representing their impact on the resolution. The ER is performed as follows: for each IoT device in the KG extracted from service providers (defined by *orgIoT:source="other"*), its most similar representation in the KGs of DM providers (defined by *orgIoT:source="DMProvider"*) is retrieved (line 16) and the *owl:sameAs* relationship is created among them (line 9). Similarity functions (lines 20-21, 46-47) are implemented using SHACL functions (see Listing 4.4), which are developed using SPARQL extension via registered URI (line 19).[14] To the best of our knowledge, this is the first ER approach based on the SHACL standard.

```
1  iotd:similarityFunction
2          a sh:SPARQLFunction ;
3  /*Define the operators of the function */
4          sh:parameter [
5                  sh:path iotd:op1 ;
6                  sh:datatype xsd:string ;
7                  sh:description "The first operand" ;
8          ] ;
9          sh:parameter [
10                 sh:path iotd:op2 ;
11                 sh:datatype xsd:string ;
12                 sh:description "The second operand" ;
13         ] ;
14 /*Define the output type*/
15         sh:returnType xsd:double ;
16 /*Define the function call */
17 sh:select """
18 SELECT
19 (<http://www.example.org/StrictStringFunction>($op1, $op2)
```

[14] https://www.w3.org/TR/rdf-sparql-query/#extensionFunctions

```
20 AS ?result) WHERE { }   """ .
```

Listing 4.4: String similarity function as SHACL function.

3.3 Step 3: Dependency Inference

This step infers the threatening dependencies topology from the global context KG obtained after the ER is performed, using the SHACL standard again: We design a SHACL rule for each dependency type described in Chapter 4 Section 2.1. These SHACL rules infer dependency relationships between devices by reasoning around contextual relationships in the global context KG. The developed SHACL rules for dependency inference will be presented in the following.

Service dependency inference

The rule presented in Listing 4.5 infers service dependency between two IoT devices by creating *hasServiceDependencyTo* relationship (line 10) when a device consumes a service offered by another device (lines 17-19).

```
1  iotd:ServiceDependency
2    rdf:type sh:NodeShape ;
3    sh:targetClass dp:IoTDevice ;
4
5    sh:rule [
6        rdf:type sh:SPARQLRule ;
7        sh:construct """
8  /*Construct the dependency relationship (here is the service dependency) */
9           CONSTRUCT {
10      $this iotd:hasServiceDependencyTo ?device .
11            }
12 /*Constraints to check before constructing the dependency relationship */
13
14 /*Constraints are contextual relationships */
15        WHERE {
16 /*Contextual relationships required to infer the service dependency*/
17                  ?device a  iotd:IoTDevice .
18                    ?device core:offers ?service .
19                    $this iotd:consumes  ?service .
20 }""" ;] ;.
```

Listing 4.5: SHACL rule for Service dependency inference.

Connectivity dependency inference

The rule presented in Listing 4.6 infers connectivity dependency between IoT devices and connectivity devices by creating *hasConnectivityDependencyTo* relationship (line 10) when an IoT device consumes a connectivity service offered by a connectivity device (lines 16-18).

```
iotd:ConnectivtyDependency
  rdf:type sh:NodeShape ;
  sh:targetClass dp:IoTDevice ;

sh:rule [
      rdf:type sh:SPARQLRule ;
      sh:construct """
/*Construct the dependency relationship (here is the connectivity dependency) */
         CONSTRUCT {
        $this iotd:hasConnectivityDependencyTo ?device .
          }
/*Constraints to check before constructing the dependency relationship */
/*Constraints are contextual relationships */
    WHERE {
/*Contextual relationships required to infer the connectivity dependency*/
             ?device a  iotd:ConnectivityDevice .
             ?device core:offers ?service .
             $this iotd:consumes  ?service .
}
""" ; ] ;.
```

Listing 4.6: SHACL rule for connectivity dependency inference.

Environment-based dependency inference

The rule illustrated in Listing 4.7 deduces environment-based dependencies. It accomplishes this by establishing a *hasEnvironmentDependencyTo* relationship between two IoT devices (line 11). This linkage is established when the initial IoT device observes a property of the physical environment, such as the *Living Room temperature*, which is altered by the actions of the other IoT device (lines 15-17).

```
iotd:EnvDependency
  rdf:type sh:NodeShape ;
  sh:targetClass dp:IoTDevice ;
  sh:rule [
      rdf:type sh:SPARQLRule ;
      sh:construct """
/*Construct the dependency relationship (here is the environment dependency) */
          CONSTRUCT {
               $this iotd:hasEnvironmentDependencyTo ?device .
                 }
/*Constraints to check before constructing the dependency relationship */
/*Constraints are contextual relationships */
            WHERE {
/*Contextual relationships required to infer the environment dependency*/
                          ?device a  iotd:IoTDevice .
```

```
16                          ?device iotd:changesProperty ?property.
17                          $this core:measures ?property.
18 .}""";] ;.
```

Listing 4.7: SHACL rule for environment dependency inference.

State-based dependency inference

The rule showcased in Listing 4.8 infers state-based dependencies. It accomplishes this by establishing a *hasStateDependencyTo* relationship between two devices (line 9). This relationship is established when the rule detects an automation rule that acts on one device based on the state of another (lines 15-24).

```
1  iotd:StateBasedDependency
2    rdf:type sh:NodeShape;
3    sh:targetClass iotd:IoTDevice;
4    sh:rule [
5    rdf:type sh:SPARQLRule ;
6    sh:construct """
7  /*Construct the dependency relationship (here is the state-based dependency) */
8    CONSTRUCT {
9       $this iotd:hasStateDependencyTo ?device .
10                 }
11 /*Constraints to check before constructing the dependency relationship */
12 /*Constraints are contextual relationships */
13   WHERE {
14 /*Contextual relationships required to infer the state-based dependency*/
15      ?device a  iotd:IoTDevice ;
16               saref:hasState ?state .
17      ?trigger a iotd:Trigger
18               iotd:relatedTo ?state .
19      $this core:offers  ?service .
20      ?service a saref:Service ;
21               iotd:allowsAction ?action .
22      ?rule a  eupont:Rule ;
23               eupont:hasAction ?action ;
24               eupont:hasTrigger ?trigger .
25       } """ ;] ;.
```

Listing 4.8: SHACL rule for State-based dependencies inference.

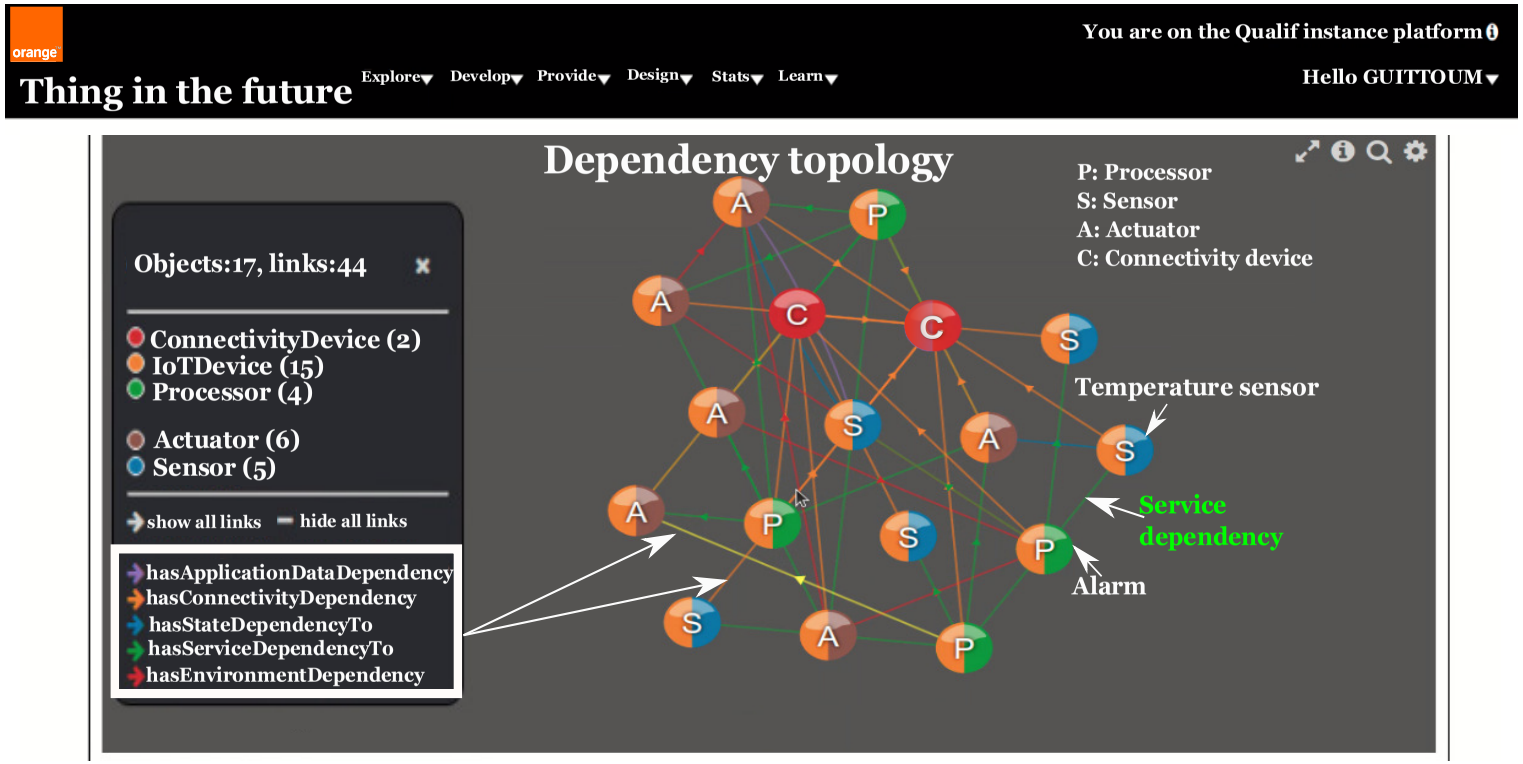


Figure 4.8: The inferred dependencies topology from the simulated Smart Home Scenario.

Implicit service dependency inference

The rule depicted in Listing 4.9 deduces implicit service dependencies by establishing a *hasImplicitServiceDependencyTo* relationship between two IoT devices (line 11). This relationship is established when, within a specific IoT application, the first IoT device receives data flows from other IoT devices through IoT application actions (lines 15-21).

```
1  iotd:ImplicitServiceDependency
2    rdf:type sh:NodeShape ;
3    sh:targetClass dp:IoTDevice ;
4    sh:rule [
5        rdf:type sh:SPARQLRule ;
6        sh:construct """
7  /*Construct the dependency relationship (here is the implicit service dependency) */
8                    CONSTRUCT {
9                $this iotd:hasImplicitServiceDependencyTo ?device .
10                    }
11 /*Constraints to check before constructing the dependency relationship */
12 /*Constraints are contextual relationships */
13       WHERE {
14 /*Contextual relationships required to infer the implicit service dependency*/
15                   ?device a  iotd:IoTDevice .
16                   ?device core:offers ?service1 .
17                   $this core:offers ?service2 .
18                ?service1 iotd:allowsAction ?action1.
19                   ?service2 iotd:allowsAction ?action2 .
20                   ?dataFlow dk:flowsFrom ?action1.
21                   ?dataFlow dk:flowsTo ?action2.}""";] ;.
```

Listing 4.9: SHACL rule for implicit service dependency inference.

Note that current version of the SHACL Advanced Features performs inference only in a single iteration of rules and the proposed rules for dependency inference are not dependant. This ensures a reliable stopping criteria for dependency inference and avoid infinite loop complications.

Once the global context KG is enriched by the inferred DKG describing threatening dependencies, it is exposed as a DT feature describing devices and their dependencies. That representation can be easily analyzed and queried by multiple DM actors thanks to Thing in APIs [15]. The inferred dependency topology from the smart home use case is shown in Figure 4.8.

4 Evaluation

We comprehensively evaluated our framework, encompassing both qualitative and quantitative aspects. In the qualitative evaluation, we: i) Inferred the threatening IoT dependency topology within a simulated smart home environment. ii) Extended our analysis to a real-world smart

[15] https://wiki.thinginthefuture.com/

home environment, specifically the DOMUS testbed [16]. iii) Thoroughly assessed the proposed IoT-D ontology, focusing on its qualitative attributes such as completeness. Additionally, in the quantitative evaluation, we: i) Assessed the performance of our framework, providing insights into its efficiency and effectiveness. ii) Conducted a quantitative evaluation of the IoT-D ontology, showing valuable quantitative metrics to measure its semantic richness.

4.1 Qualitative Evaluation

4.1.1 Simulated Smart Home Scenario

We evaluated the proposed framework on the simulated smart home scenario in Chapter 1 Section 2.1. IoT devices were simulated using the Open Source USP agents[17]. Regarding the involved DM solutions, we deployed the Orange implementation of the USP controller locally, and leveraged the cloud developer API of the Smarthings platform [18] to create and query the automation rules. Using the proposed framework, we were able to identify *44* threatening dependency relationships among the simulated Smart Home IoT devices (see Figure 4.8). We depended on human-based verification to validate the accuracy of these results, and the findings affirmed their correctness. This approach enables us to perform a qualitative validation of the proposed methodology within the specified use case.

4.1.2 Realistic Smart Home: DOMUS Testbed

We evaluated the proposed framework on *DOMUS* testbed[19] (see Figure 4.9), which represents a connected apartment of $62m^2$ including more than *90* IoT devices installed in four rooms: a living room, a bedroom, a kitchen, a bathroom, controlled from an independent control room. The latter includes software and hardware allowing the monitoring of the whole apartment. IoT devices of the DOMUS testbed are connected to each other through the *OpenHab Platform*,[20] (see Chapter 2 Section 1.1.1) which enables a set of automation rules and allows the devices to access the Internet thanks to *brigdes* devices [21].

To infer the DOMUS' dependency topology, we followed a four-step process, which can be adopted to infer the dependency topology in any other realistic IoT system: First, we identified the context data source that describes IoT dependencies, represented by the automation rules and connectivity links between the OpenHab bridges and the IoT devices. This information is described in the configuration files of the OpenHab platform. Second, we defined REST endpoints that allow the extraction of these context data. We formalized the extraction modalities related to these REST endpoints using the TD standards to be used by the proposed framework for on-demand context data extraction (see Listing 4.10). Third, we defined a mapping that allows the transformation of the extracted context data to KG according to the IoT-D ontology.

[16] https://www.liglab.fr/fr/recherche/plateformes/domus
[17] https://github.com/BroadbandForum/obuspa
[18] https://developer.smartthings.com/docs/api/public#tag/Rules
[19] https://www.liglab.fr/fr/recherche/plateformes/domus
[20] https://www.openhab.org/
[21] https://www.openhab.org/docs/concepts/things.html#bridges

Figure 4.9: DOMUS Testbed [DOMUS, 2023].

Last, we ran our framework to infer the threatening dependency topology. The result is depicted in Figure 4.10 representing the inferred dependency topology. We have automatically identified *106* threatening dependency relationships among DOMUS IoT devices. We validated the accuracy of these results through human-based verification in collaboration with the engineer responsible for *DOMUS*, affirming their correctness. This method enables us to qualitatively validate the proposed approach within the DOMUS use case.

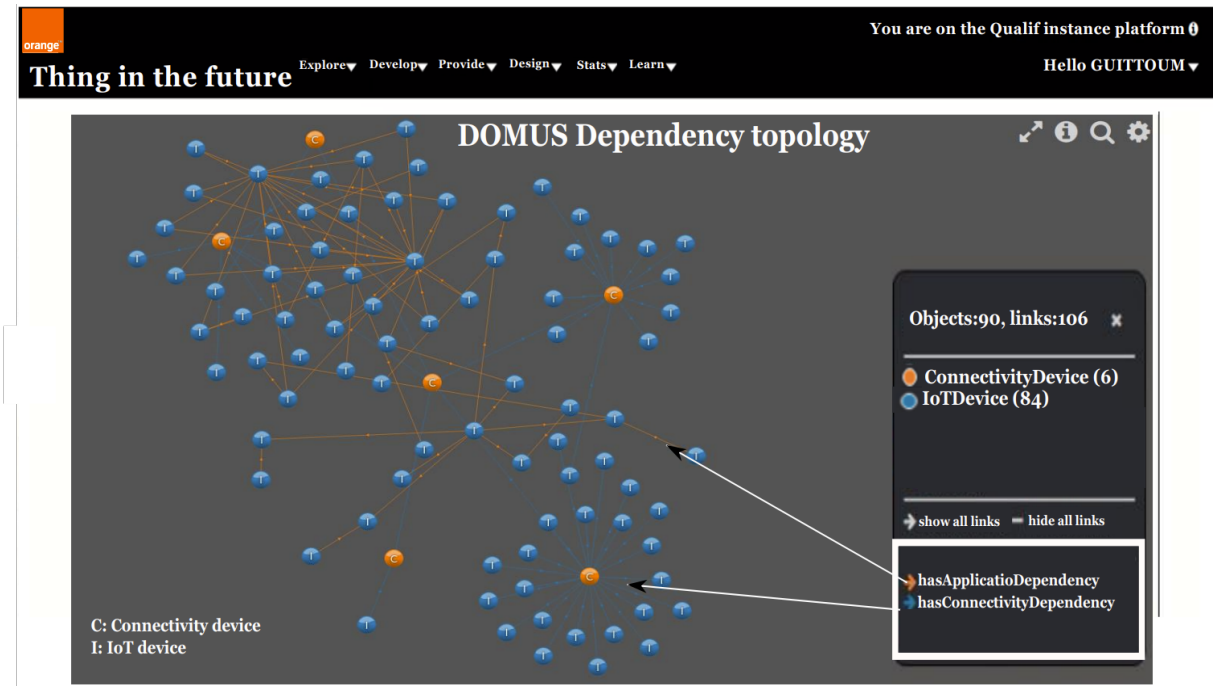


Figure 4.10: The inferred dependencies topology from DOMUS Testbed.

```
iotd:openHab rdf:type  iotd:Application;
    td:hasPropertyAffordance   iotd:propappOpenHabRules , iotd:
       propappOpenConnectivityTopology .

iotd:propappOpenHabRules  rdf:type       td:PropertyAffordance ;
                           td:hasForm      demo:capabilitiesOpenHabRules
                              .
iotd:capabilitiesOpenHabRules rdf:type   hctl:Form ;
                           hctl:forContentType    "application/json" ;
                          hctl:hasOperationType  td:readProperty ;
                          hctl:hasTarget         "http://openhab:8080/
                             api/v1/rules/" .
iotd:propappOpenConnectivityTopology  rdf:type       td:
   PropertyAffordance ;
                           td:hasForm      iotd:
                              capabilitiesOpenHabConnectivity.
iotd:capabilitiesOpenHabConnectivity rdf:type   hctl:Form ;
                           hctl:forContentType    "application/json" ;
                          hctl:hasOperationType  td:readProperty ;
                          hctl:hasTarget         "http://openhab:8080/
                             api/v1/connectivityFile/" .
```

Listing 4.10: DOMUS data extraction modalities.

4.1.3 IoT-D ontology Qualitative Evaluation

The evaluation of a semantic ontology can be conducted through qualitative methods, utilizing a set of Competency Questions (CQ) and executing SPARQL queries against the instances of the ontology to determine if the defined ontology can effectively address these CQs [Uschold, 1996; Ihsan, 2023]. In this study, we identified a total of 22 CQs that were categorized into three classes: Topology Recognition (TR), Dependency Information (DI), and Dependency Information Access (DIA). A sample of these CQs can be found in Table 4.1. Subsequently, we formulated a series of SPARQL queries for each individual CQ. These queries were designed to assess the capability of the ontology to provide relevant answers. The Listing 4.11 presents an example of a SPARQL query allowing to answer the competency question CQ22. We make available online all the competency questions with their associated SPARQL queries.[22]

Finally, we executed the specified SPARQL queries using the *Protégé* SPARQL endpoint on two distinct datasets: one derived from the simulated smart home scenario and another from the DOMUS testbed. Notably, we found that the context KG built upon the IoT-D ontology successfully provided answers to all the CQs. This result proves the IoT-D ontology's completeness and its ability to encompass diverse IoT scenarios effectively.

[22] https://github.com/Orange-OpenSource/ISWC-IoT-D-ontology-Documentation/blob/master/CQs/CompentencyQuentions.md

Table 4.1: Part of the identified CQs.

No.	Competency Question	Class
CQ1	What are IoT devices present in the managed IoT system?	TR
CQ2	What are IoT applications present in the managed IoT system?	TR
CQ3	What are the services consumed by a given device?	DI
CQ4	What are device actions associated with a given automation rule?	DI
CQ5	How to access the connectivity topology of a given connectivity device?	DIA
..	..	..
CQ22	How to access the connectivity topology of a given connectivity device?	DIA

```
SELECT ?uri ?contentType WHERE {
  ?propertyAffordance a td:PropertyAfforance.
  ?connectivityDevice a iotd:ConnectivityDevice;
  td:hasPropertyAffordance ?propertyAffordance.
  ?form  a hctl:Form.
  ?propertyAffordance hctl:hasForm ?form.
  ?form hctl:hasTarget ?uri.
  ?form hctl:forContentType ?contentType.
/* The URI of the connectivity device is provided as input  */
FILTER(?connectivityDevice=<URI>) }
```

Listing 4.11: The SPARQL query of the CQ22.

4.2 Quantitative Evaluation

4.2.1 Performance Evaluation

We carried out a set of performance evaluations by i) measuring the completion time of the ER and dependency inference steps[23] on smart home scenarios with different scales; ii) comparing SHACL to SWRL,[24] another formalism for inference rules used by competing approaches for direct dependencies inference [Mohsin, 2017] and entity resolution [Benbernou, 2021]. The

[23] We do not provide an evaluation for the context extraction step, since it depends on DM solutions performance and network characteristics.

[24] https://www.w3.org/Submission/SWRL/

comparison was performed according to step 3, dependency inference. The test data sets are smart home scenarios with different scales generated by duplicating the semantic description of the smart home scenario described in Chapter 1 Section 2.1. We executed tests on an Ubuntu 20.04 with 32Go RAM and Intel Corei7 2.5 GHz processors. SHACL inference is implemented using TopBraid SHACL API (version 1.0.1),[25] and SWRL inference is performed using Openllet reasoner with OWL API (version 2.6.5)[26] used by competing approaches. We note that the comparison results are limited by these technological choices.

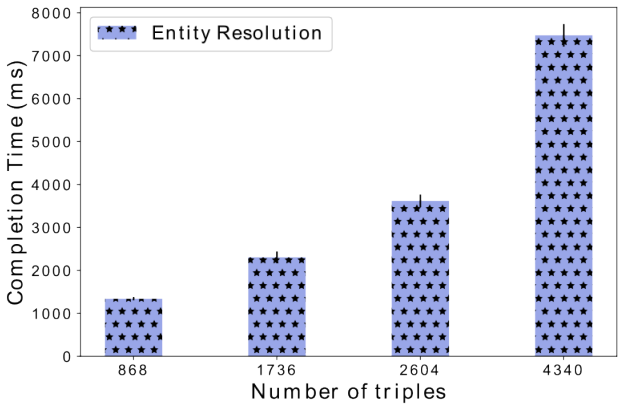


Figure 4.11: Completion time of the ER step.

We found that the completion time of the dependency inference step is almost negligible (on average, 32.5 ms for 868 triples and 63 ms for 4340 triples). The ER completion time (see Figure 4.11) is more time-consuming due to: 1) the graph pattern complexity of the ER rules and 2) the calculations performed by the ER rules in addition to the inference task. Overall, the framework's performance appears sufficient for a human-based decision-support tool. However, this time should be discussed more from the perspective of integrating the proposed framework in automated DM processes e.g., automatic cascading failure management.

Comparing SHACL with SWRL (see Figure 4.12) according to the dependency inference time shows that SHACL performs the best, especially for a large number of dependencies. This can be justified from two perspectives: 1) from the technological perspective, SHACL has the same format as the validated data, which simplifies the technology stack required to implement it, unlike SWRL [Frank, 2021]; 2) from the theoretical complexity perspective, SWRL complexity is exponential [Mei, 2006]. Meanwhile, SHACL complexity depends on the complexity of SPARQL query language, [27] which is polynomial if the graph pattern uses only AND and FILTER operators [Pérez, 2006], which is the case for the dependency inference rules. We make available online the quantitative evaluation source code with the generated smart home data sets.[28]

[25] https://github.com/TopQuadrant/shacl
[26] https://github.com/Galigator/openllet
[27] https://book.validatingrdf.com/bookHtml013.html
[28] https://github.com/Orange-OpenSource/ISWC-ReasoningCode

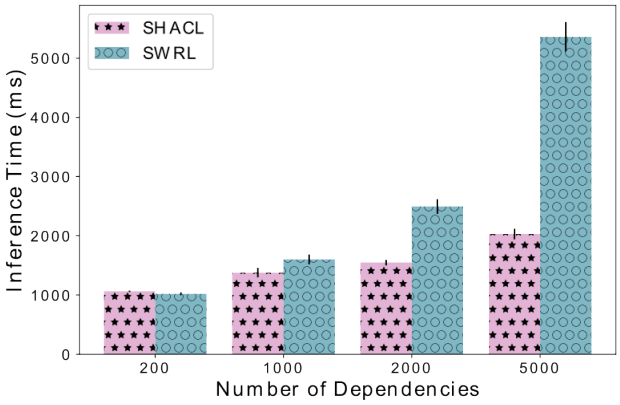


Figure 4.12: SHACL VS SWRL for dependency inference.

4.2.2 IoT-D ontology Quantitative Evaluation

We assessed the quality of the IoT-D ontology using the *OntoQA* methodology [Tartir, 2005], which evaluates the ontology using a set of metrics such as schema metrics. To evaluate the richness of the IoT-D ontology, we have selected the following metrics inspired by the work [Ihsan, 2023]:

- Relationship richness (RR) measures the diversity of relations within the ontology. Formally, it is defined by

$$RR = \frac{|P|}{|P| + |SC|} \tag{4.1}$$

  P is the number of relationships in the ontology, and SC is the number of subclass relationships.

- Attribute richness (AR) shows the richness of concept description through attributes (a.k.a data properties) of the ontology. Formally, AR is defined by

$$AR = \frac{|AT|}{|C|} \tag{4.2}$$

  AT is the number of attributes for all classes, and C is the number of classes.

- Inheritance richness (IR) characterizes the dispersion of information among various levels of the inheritance tree within the ontology. It reflects how knowledge is organized and categorized into distinct classes and subclasses in the ontology. Formally, IR is defined by

$$IR = \frac{|SC|}{|C|} \tag{4.3}$$

We compared the OntoQA results of the IoT-D ontology with the IoTB ontology results. The latter is the only found ontology representing dependencies among IoT devices [Mohsin, 2017]. The result (see Table 4.2) shows that the IoT-D ontology outperforms the IoTB ontology for

all the OntoQA metrics. This signifies that our ontology has more diversity in relationships, and represents a wider range of knowledge and more knowledge per instance compared to IoTB ontology, which is useful for knowledge-based decision support. Moreover, the IoT-D ontology has a lower number of concepts, which means that it is able to represent the same domain of knowledge in a more concise manner, which increases performance and reduces reasoning time.

Table 4.2: OntoQA Evaluation results IoT-D Ontology.

Ontology	C	SC	AT	P	RR	AR	IR
IoTB	30	11	0	21	0,66	0	0,37
IoT-D	22	09	15	22	**0,71**	**0,68**	**0,41**

5 Conclusion

In this chapter, we shed light on our practical framework that infers threatening dependency topology to help legacy DM solutions efficiently address dependencies-related threats. We have identified several business use cases of the proposed framework, such as remote cascading failure management, allowing for reduced DM costs and enhancing customers' quality of experience. Other creative use cases may be developed by exploiting the DKG and its context KG.

The proposed framework leverages established SW standards of W3C and ETSI, such as TD, SHACL, and SAREF, to enable interoperability across siloed DM solutions. It adopts a KG-based model for efficiently analyzing heterogeneous, large, and unstructured dependencies data. It uses the DT technology to address the dynamic aspect of IoT dependencies as well as for knowledge management and information sharing between DM actors. It is based on a three-step process involving extracting dependencies data from siloed DM solutions, resolving this data, and finally inferring and reasoning around threatening dependencies.

We validated the proposed solution by inferring threatening dependencies topology in a smart home scenario managed by multiple DM actors. However, our approach is generic enough to be applied to IoT applications other than smart homes thanks to the IoT-D ontology, which proposes application-agnostic modeling for IoT dependencies. Moreover, our approach can be taken up in other domains where there is a need to connect siloed and dynamic data to unlock innovative use cases.

To go one step further in managing dependencies-related threats, especially cascading failure, we extended the proposed framework to enable collaborative and automatic cascading failure diagnosis and recovery across legacy DM solutions using a cooperative *Multi-Agent System* (MAS) system. This solution will be discussed in the next chapter.

Chapter 5

Solving The Cascading Failure Dilemma using A Semantic Multi-agent System

Summary

This chapter shows our approach enabling autonomous cascading failure diagnosis and recovery on IoT devices managed by siloed DM actors, using a Semantic *Multi-Agent System* (MAS).

Contents

In the previous chapter, we introduced our solution aimed at facilitating the inference of dependencies between *Internet of Things* (IoT) devices to simplify the process of identifying the root causes of cascading failures for *Device Management* (DM) actors. Nevertheless, it is important to acknowledge that this human-centric analysis tool possesses inherent limitations. Consequently, there exists a need to go beyond human-based decision-making towards the development of automated mechanisms that can effectively facilitate the diagnosis and recovery of cascading failures, even in the presence of disparate and siloed DM solutions.

As detailed in Chapter 3 Section 3.2, literature approaches proposed for IoT failure management expose several research gaps. In the realm of failure detection, ML models are commonly employed but face constraints due to data availability and accuracy, given the complexity of IoT systems. In terms of failure diagnosis, knowledge-based methods show promise but often lack root cause identification. Turning to failure recovery, numerous frameworks exist but tend to overlook IoT device diversity and connectivity protocols, relying heavily on costly device replacement. Surprisingly, no comprehensive end-to-end framework integrates detection, diagnosis, and recovery in IoT failure management. Cascading failures are not adequately addressed, and current approaches assume full control over IoT devices, overlooking real-world governance complexities.

Moreover, our study of the capabilities of market DM solutions (see Chapter 3 Section 3.3) in performing failure management shows they lack holistic solutions for IoT failure management, highlighting the need for *collaborative DM* efforts to solve the cascading failure dilemma.

In this chapter, we present a collaborative and end-to-end failure management approach to help DM actors effectively manage failures and solve cascading failure dilemmas using a cooperative *Multi-Agent System* (MAS) (see Chapter 2 Section 2.3). 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* (SW) standards, such as ontologies, in order to simplify failure information exchange and enhance the interoperability among siloed DM platforms. It leverages our previously presented solution of IoT dependency inference 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.

In the following, we illustrate the cascading failure dilemma using the smart home use case presented in Chapter 1 Section 2.1. Then, we present our multi-OSAMA system for Collaborative CFM: First, we show our modeling for OSAMA. Then, we discuss OSAMA's capabilities and functioning. Last, we provide the collaborative CFM protocol and show how it can be used to solve cascading failure dilemmas. We note that the contributions described in this chapter have been published at the in-use track of the 22nd International Conference of Semantic Web (ISWC'23) [Guittoum, 2023a].

1 Illustration of Cascading Failure Dilemma

IoT failures (described in Chapter 3 Section 3.1) can generate multiple cascading failures when occurring on interdependent IoT devices. Managing these cascading failures by siloed DM actors is no longer obvious, especially with their limited failure management capabilities (see Chapter 3 Section 3.3). Take the scenario presented in Figure 5.1 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 on the leak detector (see Rule 5 in Table 1.2). Additionally, the light bulbs are affected as they are state-dependent on the

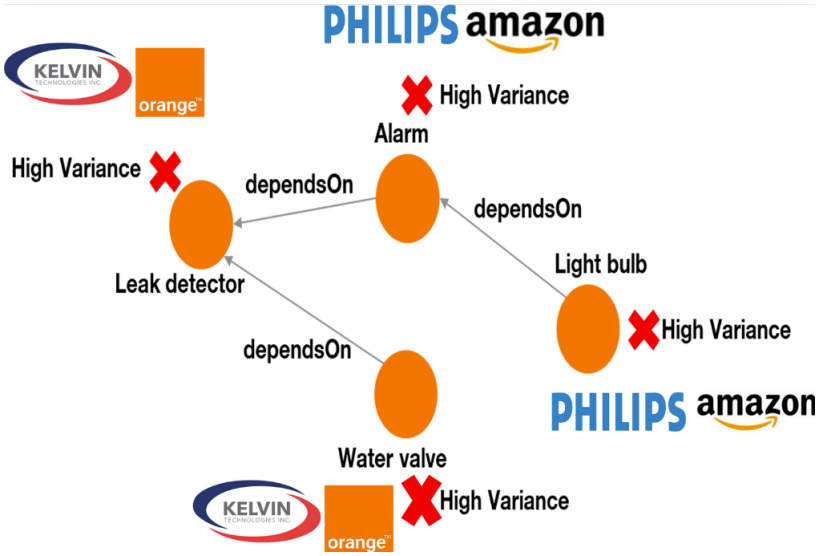


Figure 5.1: An example of cascading failure dilemma.

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 1.1). 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.

Thus, there is a need for a collaborative DM solution for tackling cascading failure dilemmas to effectively identify failure root causes and automate failure recovery.

2 Semantic Multi-OSAMA For Collaborative CFM

MAS enables the collaboration between multiple legacy systems by developing an agent wrapper around them, enabling their participation in collaborative problem-solving and decision-making processes (see Chapter 2 Section 2.3). 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. It is conceived using the multi-agent-oriented programming paradigm including the two dimensions: agent and environment dimensions (see Chapter 2 Section 2.3.1): 1) Within the agent dimension, 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; 2) Within their environment, OSAMAs are provided by four (04) *Artifacts* encapsulating external services that they can explore at runtime to ease CFM (see Figure 5.2):

- *Monitoring Artifact*: Allows to monitor IoT devices and detect failures using legacy DM platforms. Each OSAMA agent has its own monitoring artifact exposing the monitoring capabilities of its legacy DM solutions.

- *Diagnosis Artifact*: Allows to identify failure type and compensatory actions using *Failure Knowledge Base* (FKB). Each OSAMA agent related to a DM actor, owning failure information on its devices, has its own diagnosis artifact exposing an FKB including information about device failures according to their type.

- *Dependency Artifact*: Thanks to Semantic *Digital Twin* (DT), this artifact allows automatic access to an accurate view of dynamic dependency relationships between IoT devices in order to ease cascading failure root cause identification. It is shared between the OSAMA agents allowing them to build collaboratively the global dependency topology and use it for failure root cause identification. It relies on our framework described in Chapter 4.

- *Recovery Artifact*: Allows to execute DM recovery actions on IoT devices using legacy DM platforms. Each OSAMA agent has its own recovery artifact.

In the following, we discuss the *OSAMA* BDI model, *OSAMA* artifacts, and the *Collaborative CFM Protocol*.

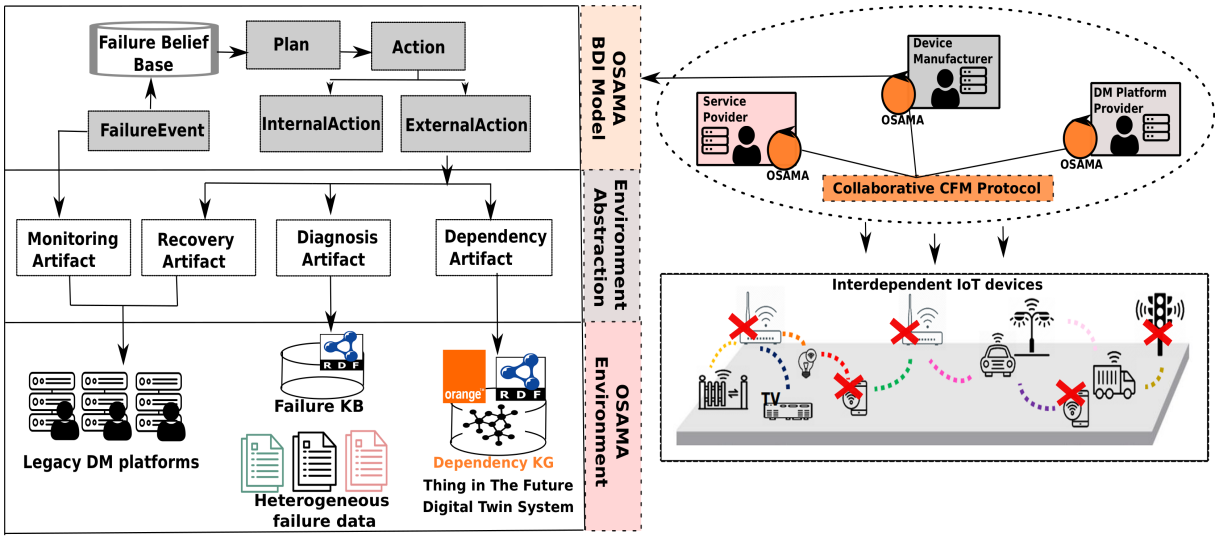


Figure 5.2: Overview of Semantic Multi-OSAMA For Collaborative CFM

2.1 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 [Silva, 2020]. As a reminder (see Chapter 2 Section 2.3.1), the BDI agent model aims at programming rational agents based on human mental attitudes of beliefs, desires, and intentions [Bratman, 1987; Silva, 2020]. 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

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

Table 5.1: 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[1] 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.

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.

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 5.1).

- *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 OSAMA requests.

2.2 Diagnosis Artifact

This artifact embeds an FKB that allows *OSAMAs* to get failure information such as possible compensatory actions e.g., *device reboot*, given a set of failure symptoms such as *failure code*.

More precisely, this artifact uses the SPARQL query described in Listing 5.1, to perform the diagnosis. We assume that each DM actor generates an FKB involving its governed failure information (see Chapter 3 Section 3.3). These FKBs are built using an ontology called *Internet of Things Failure* (IoT-F)[2] (see Figure 5.3) that we developed using the ontology engineering methodology NeOn [Suárez-Figueroa, 2012].

```
SELECT  ?compensatoryAction
WHERE {
?failureMode rdf:type iotf:FailureMode.
?failureMode iotf:hasCompensatoryAction ?compensatoryAction.
?failureMode iotf:happensAt ?deviceType.
OPTIONAL {?failureMode iotf:Symptom [Failure Symptoms ex: ?failureCode].
    }
Filter([a set of failure symptoms])
}
```

Listing 5.1: SPARQL Query for failure diagnosis.

The main purpose of the IoT-F ontology is to allow OSAMA agents to share a global understanding of heterogeneous and distributed failure information. However, it has other intended usages: 1) connecting distributed and heterogeneous IoT failure diagnosis information governed by different actors; 2) assist DM actors in structuring failure information, which is characterized

[2] https://w3id.org/iotf

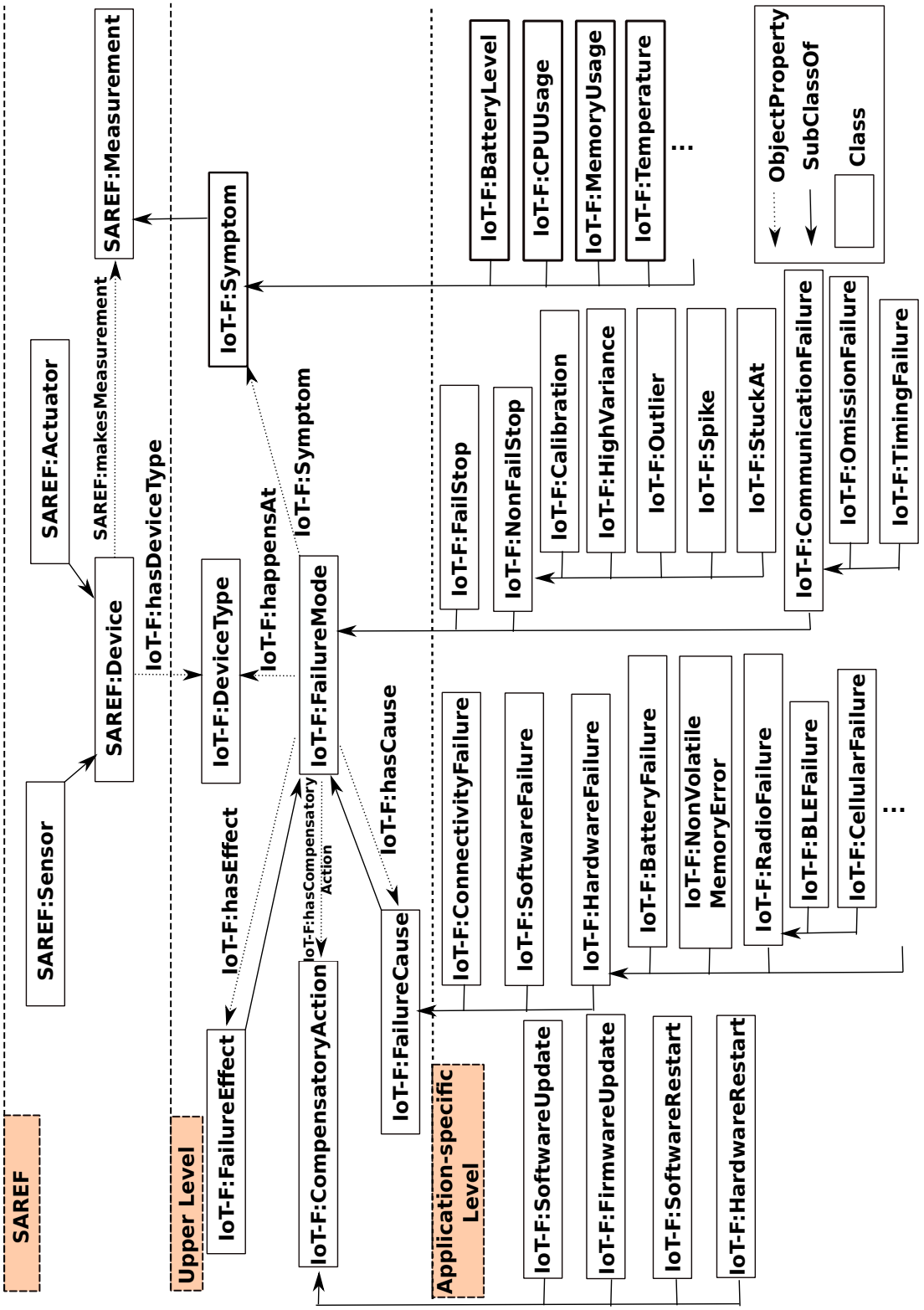


Figure 5.3: IoT-F: IoT Failure Ontology.

by heterogeneity, incompleteness, and ambiguity [Steenwinckel, 2018]; 3) enable efficient search for recovery plans, failure cause, and its impact thanks to the KG structure.

The IoT-F ontology **reuses** the standardized ontology *SAREF*. Its architecture is based on two levels inspired by the work [Emmanouilidis, 2020]:

- *Upper level:* This level is based on the *Failure Mode Effect Analysis* (FMEA) [3] concepts, which are related to a quality assessment methodology providing a generic model for failure description in any domain of interest (see Chapter 3 Section 4.3). We reused FMEA concepts proposed in [Emmanouilidis, 2020], which have been inspired by relevant standards such as IEC60812[4] ISO13372[5], ISO13306[6], and ISO2041 [7]. It includes the concept *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*.

- *Application-specific level:* This level represents failures in IoT systems by specializing each class in the upper level. 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 [Norris, 2022; Chakraborty, 2018; Sharma, 2010; Ozeer, 2019] for IoT failure, failure cause, and recovery actions taxonomies (see Chapter 3 Section 3.1), and market DM models mainly *Matter*[8] and TR-181[9] (see Chapter 2 Section 1.1.1) for IoT failure symptoms specialization.

2.3 Dependency Artifact

This shared artifact allows OSAMA agents to identify failure root causes through the analysis of dependency relationships among IoT devices. It incorporates our framework (see Chapter 4), which enables automatic inference and analysis of dynamic dependency relationships among IoT devices using *semantic DT*. Namely, the dependency relationships are represented through an *IoT Dependency Knowledge Graph* (DKG) exposed as DT for OSAMA agents.

In this DKG, IoT device representations are annotated with information about the OSAMAs that manage them, represented through *Agent Communication Modality Ontology (ACMO)* (see Figure 5.4) that we developed. This annotation allows OSAMAs to communicate with each other while exploring the DKG for failure root cause identification. The ACMO aims to model communication modalities allowing agents within a multi-agent system to reach each other regardless of their implementation technologies.

[3] https://asq.org/quality-resources/fmea
[4] https://webstore.iec.ch/publication/26359
[5] https://www.iso.org/standard/52256.html
[6] https://standards.globalspec.com/std/10272557/en-13306
[7] https://www.iso.org/standard/68734.html
[8] https://csa-iot.org/all-solutions/matter/
[9] https://usp-data-models.broadband-forum.org/

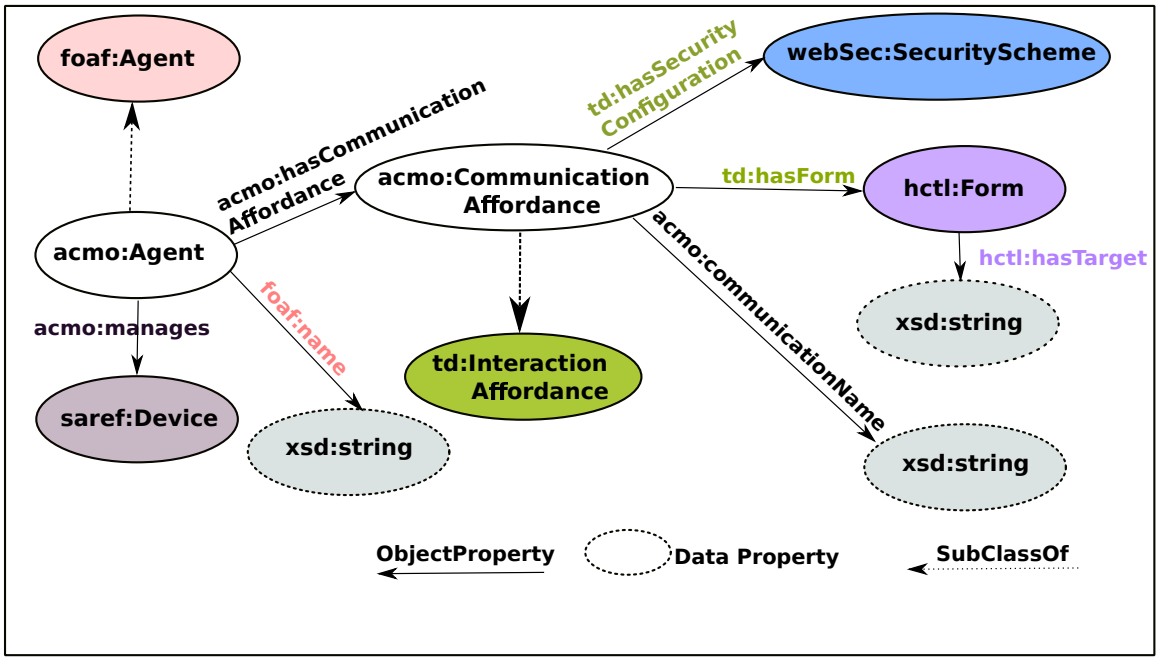


Figure 5.4: Agent Communication Modality Ontology

In order to build the ACMO, we studied communication modalities in different agent platforms and standards such as *FIPA* [10], *JaCaMo* [Boissier, 2020], and *DARPA* [Patil, 1992]. As a result, we found that agent communication modalities are often represented by agent name or agent address. Thus, we chose to represent these communication modalities by the class *acmo:CommunicationAffordance*, which extends the class *td:InteractionAffordance* of the *Thing Description* (TD) ontology (Chapter 2 Section 2.1.1). The *acmo:CommuincationAffordance* can be either the agent's name or agent address including URI, port, or IP address, represented by the class *hctl:Form* and its data property *hctl:hasTarget*. It is associated with each agent represented by the class *acmo:Agent* through the object property *acmo:hasCommunicationAffordance*.

From a technical standpoint, OSAMA agents rely on the Orange DT platform Thing in, providing a set of *REST APIs* to query the DKG when identifying failure root causes.

2.4 Monitoring and Recovery Artifacts

These artifacts embed monitoring and recovery functions of the legacy DM platform to allow OSAMAs 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 OSAMAs 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 (see Chapter 3 Section 3.3) [Sinche, 2020]. This could boost usability and save integration costs by avoiding developing a solution from scratch, which consists of integrating heterogeneous IoT devices through APIs to be accessed by OSAMAs for monitoring and recovery.

[10] http://www.fipa.org/

2.5 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 Chapter 3 Section 3.3): *Service Provider* (SP), *DM Platform Provider* (DMP), and *Device Manufacturers* (MN).

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.

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 , 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).

Let us illustrate the CFM protocol in the cascading failure dilemma presented in Chapter 5 Section 1. 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 DKG 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 OSAMA 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 (see Figure 5.5).

Algorithm 1 Collaborative CFM Protocol

BEGIN
if failure event $evt = (device_i, sourceType, source)$ arrives **then**
  Update the belief base Blf with predicate $failed(device_i)$
  [Local Failure Plan]
  $S \leftarrow getDeviceSymptoms(device_i)$
  if Profile=SP **then**
    $recovery \leftarrow diagnosis(device_i, S)$
  end if
  if Profile=DMP **then**
    $OSAMA_d \leftarrow getDiagnosisAgent(device_i)$
    $requestDiagnosis(OSAMA_d, device_i, S)$
    Wait for proposed recovery action $recovery$ from $OSAMA_d$.
  end if
  $recover(recovery, device_i)$
  if $getDeviceState(device_i) = failed$ **then**
    [Cascading Failure Plan]
    $DKG \leftarrow getDependency(device_i)$
    for $(device_k, OSAMA_k)$ in DKG **do**
      $sendCFMRequest(device_k, OSAMA_k)$
      Wait for response $OSAMA_k$
    end for
    $recover(recovery, device_i)$
    if $getDeviceState(device_i) = failed$ **then**
      Notify customer care service
    end if
    Update the belief base Blf with predicate $recovered(device_i)$
    if $sourceType = OSAMA$ **then**
      $responseCFMRequest(device_i, source)$
    end if
  end if
end if
END

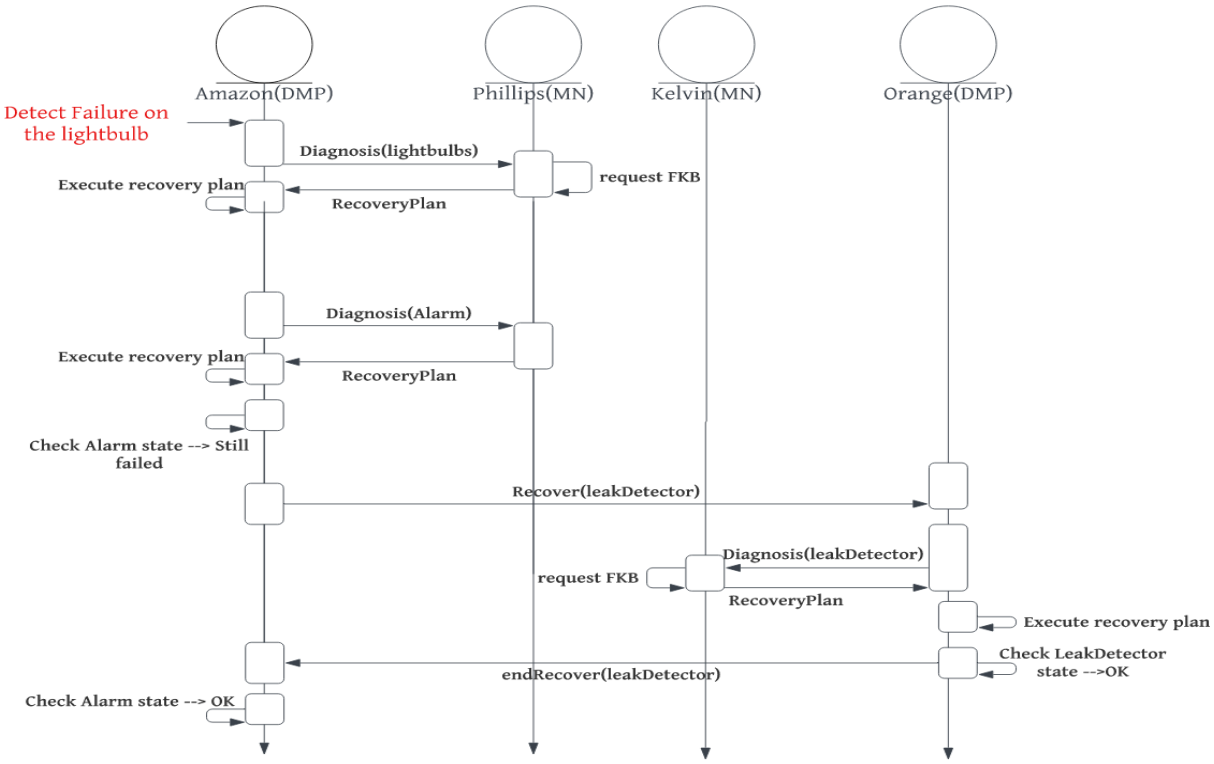


Figure 5.5: CFM protocol illustration.

3 Evaluation

In this section, we present an extensive evaluation of our proposed solution including an evaluation of OSAMA agents and the IoT-F ontology. In our evaluation of OSAMA agents, we aim to answer the following questions: (1) How effectively can our OSAMA agents handle various cascading failure scenarios?, (2) What is their performance in terms of computation time?, and (3) What is the impact of using it instead of the Orange legacy DM solutions in terms of time to repair failures and resource consumption?. To this end, we performed a qualitative evaluation by checking how our *OSAMAs* perform regarding a set of cascading failure scenarios. Moreover, we quantitatively evaluated the performance by measuring the completion time of the collaborative CFM protocol. In addition, we assessed the impact of using our solution on resource consumption in IoT infrastructures, using our extension of the simulator *iFogSim* [Gupta, 2016] that we refer to as *FMSim* [11] (see Appendix A).

To prove the quality of the IoT-F ontology, we provide a qualitative and quantitative evaluation to assess its completeness and richness.

In the following, we present technical details and discuss evaluation results. We note that the code source is available from GitHub [12].

[11] https://github.com/Orange-OpenSource/collaborativeDM-FM-Simulator/
[12] https://github.com/Orange-OpenSource/collaborativeDM-OSAMA-agent/

3.1 Technical Architecture

The technical architecture is described in Figure 5.6, we implemented the *OSAMAs* with the *JaCaMo* framework (version 1.1)[13], which allows adaptable and scalable MAS management and coordination in complex environments [Boissier, 2020]. Within the JaCaMo framework, the CFM protocol described in Algorithm 1 is implemented using the *Jason*[14] BDI technology allowing OSAMA agent to handle failure events in a parallel and coherent manner. The OSAMAs artifacts are implemented with the *Cartago*[15] technology that allows agents to access resources and services within their shared environment.

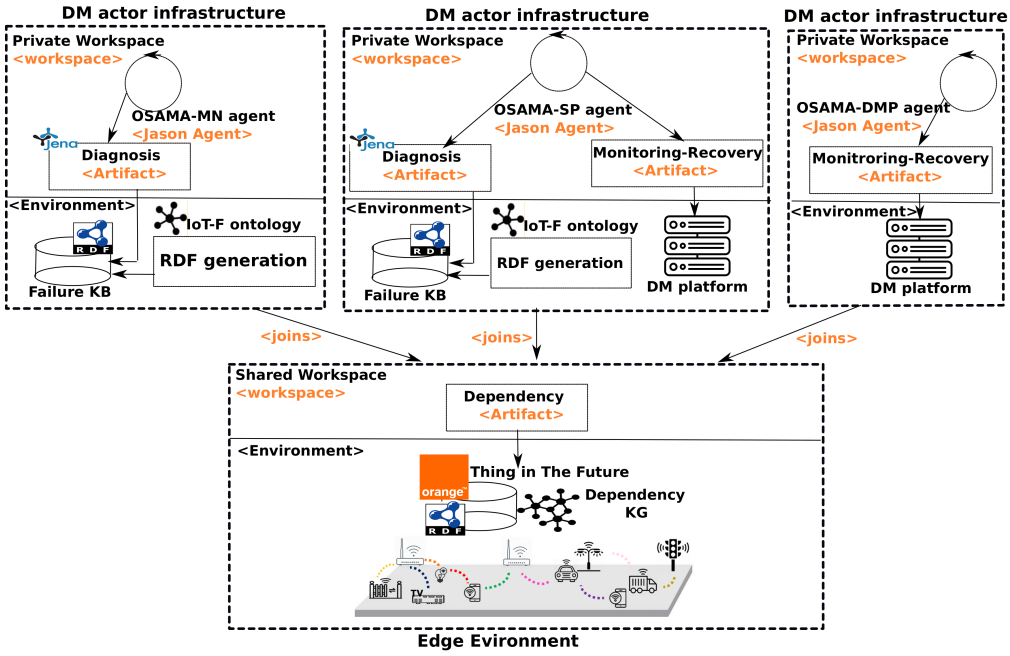


Figure 5.6: Technical Architecture of Multi-OSAMA agents.

We associated to each OSAMA agent a *Private Workspace* that includes its private artifacts such as *Recovery and Monitoring Artifacts* for OSAMA-DMP and *Diagnosis Artifact* for OSAMA-MN. These artifacts are deployed in the DM actor infrastructure. Regarding shared artifacts namely *Dependency Artifact*, it is installed in a shared workspace that can be deployed in an edge device such as *Orange LiveBox*. We leverage the framework *JaCaMo-Rest*[16] to allow agent distribution on multiple nodes. Reasoning in the diagnosis artifact is implemented with *Apache Jena* (version 3.4.0)[17]. To better represent a multi-actor deployment, we deployed the *OSAMAs* associated with the smart home use case (see Chapter 1 Section 2.1) in an Orange cloud infrastructure with the following resource: 1000 MIPS as CPU and 2GB as requested memory. The Figure 5.7 describes the deployment architecture of OSAMA agents present in the use case [18].

[13] https://github.com/jacamo-lang/jacamo
[14] https://jason.sourceforge.net/wp/
[15] https://cartago.sourceforge.net/
[16] https://github.com/jacamo-lang/jacamo-rest
[17] https://github.com/apache/jena
[18] As distributed artifacts are not implemented yet in JaCaMo, we duplicated the deployment of the dependency artifact in each agent.

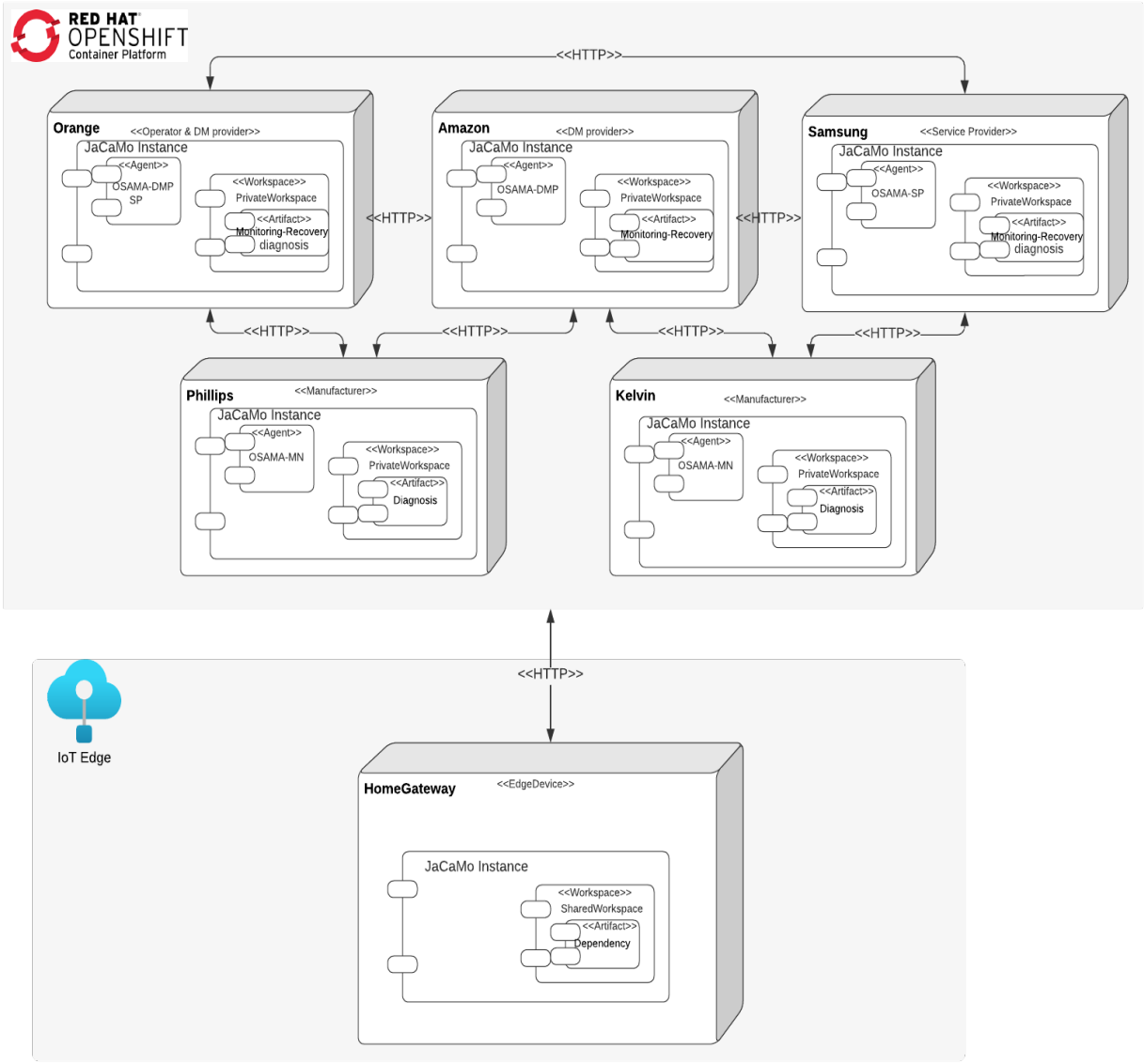


Figure 5.7: Deployment Architecture of OSAMA agents in the Cloud.

3.2 Qualitative Evaluation

3.2.1 OSAMA agents Qualitative Evaluation

The qualitative evaluation has been performed on the smart home use case presented in Chapter 1 Section 2.1, by checking how our *OSAMAs* perform regarding cascading failure scenarios presented in Table 5.2. 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 involved injecting failures in OSAMA agents' belief bases and letting them perform CFM. We validated the accuracy of these results through human-based verification of recovery logs, affirming their correctness.

Table 5.2: 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 Wi-Fi 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

3.2.2 IoT-F Qualitative Evaluation

We followed the same methodology used for the evaluation of the IoT-D ontology through Competency Questions (CQ) and executing SPARQL queries against the instances of the ontology to

determine if the defined ontology can effectively address these CQs. In this study, we identified a total of 08 CQs (see Table 5.3). Subsequently, we formulated a series of SPARQL queries

Table 5.3: Part of the identified CQs.

No.	Competency Question
CQ1	What are IoT device Types available in the FKB?
CQ2	What are failure modes on a given IoT device type?
CQ3	What are symptoms of a given failure mode?
..	..
CQ08	What are possible failure mode and their compensatory actions given a set of symptoms?

for each CQ. The Listing 5.2 presents an example of a SPARQL query allowing to answer the competency question CQ08. We make available online all the competency questions with their associated SPARQL queries.[19]

```
SELECT ?failureMode ?compensatoryAction WHERE {
  ?failureMode a iotf:FailureMode.
  ?compensatoryAction a iotf:CompensatoryAction.
OPTIONAL {?failureMode iotf:Symptom [Failure Symptoms ex: ?failureCode].
    }
Filter([a set of failure symptoms]) }
```

Listing 5.2: SPARQL Query

Finally, we executed the specified SPARQL queries using the *Protégé* SPARQL endpoint on a generated FKB [20] [21]. Notably, we found that the FKB built upon the IoT-F ontology successfully provided answers to all the CQs. This result proves the IoT-F ontology's completeness.

3.3 Quantitative Evaluation

3.3.1 CFM Performance Evaluation

We evaluate the performance of the collaborative CFM protocol performed by the use case OSAMA agents in a cloud-based 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 $5s$ (see Figure 5.8), which we consider acceptable compared to the Orange legacy solution taking *from 15 to 20 min* according to Orange customer care service. Moreover, this performance can be enhanced

[19] https://github.com/Orange-OpenSource/collaborativeDM-IoTF-ontology-documentation/blob/master/cqs-sparql.md

[20] https://github.com/Orange-OpenSource/collaborativeDM-OSAMA-agent/blob/master/QualitativeEvaluation-ISWC/FKB-Kelvin.owl

[21] https://github.com/Orange-OpenSource/collaborativeDM-OSAMA-agent/blob/master/QualitativeEvaluation-ISWC/FKB-Phillips.owl

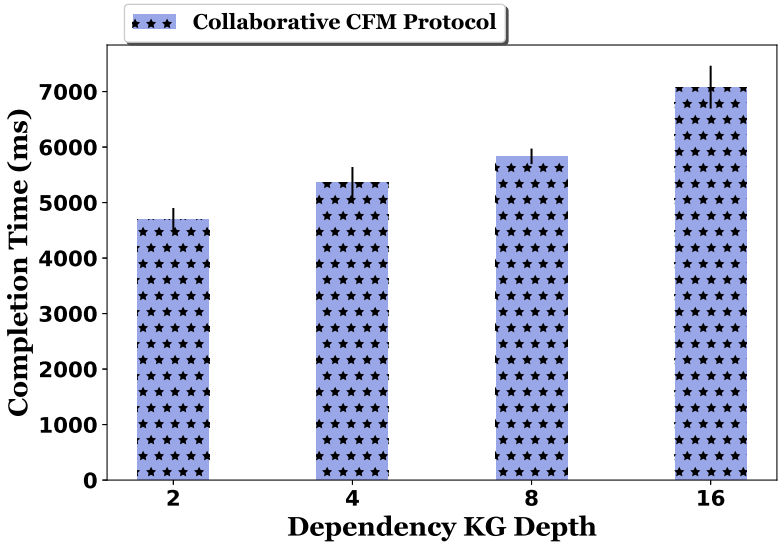


Figure 5.8: CFM completion time as a function of the DKG depth

by reducing message exchange between *OSAMAs* using learning capabilities such as predicting the root cause of a cascading failure or offloading OSAMAs to the edge to reduce latency. The DKG could also be deployed at the edge, as Thing in allows this feature.

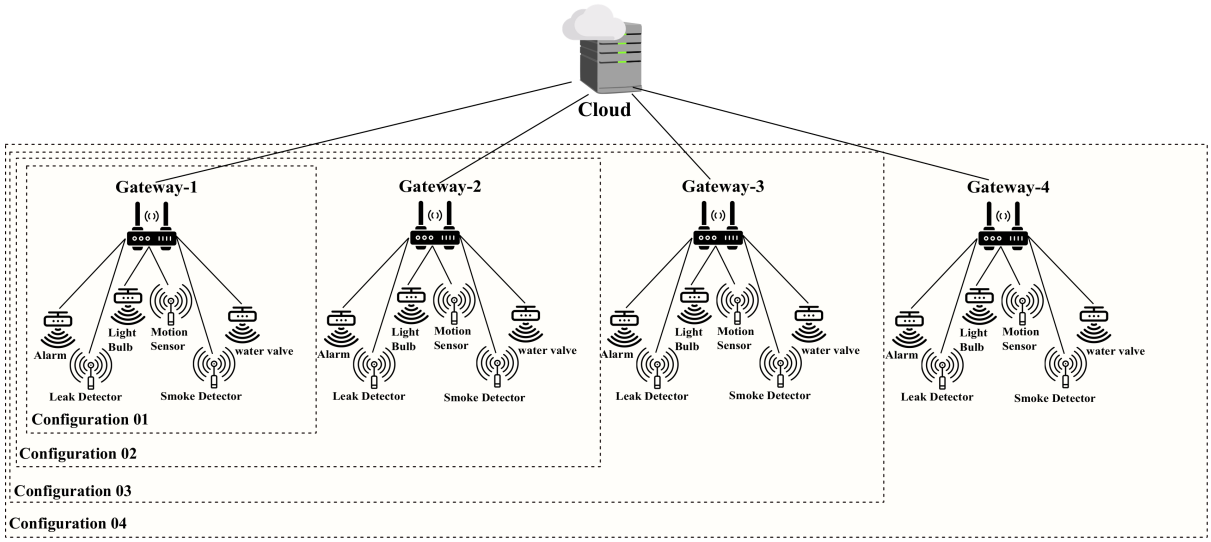


Figure 5.9: Simulation Topology

3.3.2 OSAMA Impact on Resource Consumption

Failures in IoT infrastructures can result in a significant loss of resources, as they make the infrastructure propagate 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

Table 5.4: Resource Characteristics

Resource	Speed (MIPS)	RAM (GB)	Uplink (MBPS)	Downlink(MBPS)
Cloud	44800	40000	10000	10000
Gateways	500	1000	10000	10000

used Discrete Event Simulator for Fog and IoT because of its flexibility, scalability, and accessibility [Perez Abreu, 2020]. 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, a.k.a *Tuple*, to actuators according to a defined application logic. However, *iFogSim* does not support failure simulation on IoT devices.

To this end, we developed *FMSim*, an extension for *iFogSim* allowing failure injection and recovery simulation on IoT devices. We have been inspired by the failure injection method proposed in the simulator *CloudSimPlus* that alters simulation data streams by injecting failure events according to a probabilistic distribution [Nita, 2014]. We simulate failure recovery by stopping failure events from the simulation data stream.

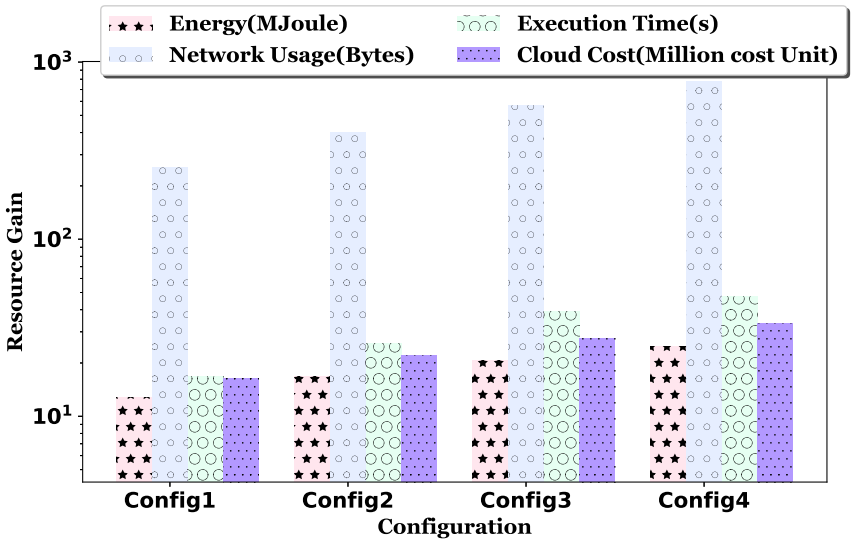


Figure 5.10: Resources gain of using *OSAMA* instead of legacy solution

Table 5.5: Tuple Characteristics

Tuple Type	CPU Length (MIPS)	N/W Length
MOTION	1000	2000
LIGHT_CONTROL	100	100
LEAK	2500	2000
VALVE_CONTROL	14	500
ALARM_CONTROL	14	100
SMOKE	2000	2000

Table 5.6: Simulation Parameters

Parameter	**Value**	**Reference**
Simulation Time	02 days	-
Tuple characteristics RAM Module Fog Device characteristics IoT device Latency Gateway Latency	Table 5.5 10MB Table 5.4 1 ms 100 ms	[Mahmud, 2022; Gupta, 2016; Gupta, 2017; Naas, 2018]
Sensor reading distribution	-Leak detector sensor: 1 detection/day. -Motion sensor: 1 detection/ 4-10 minutes. -Smoke sensor: 1 detection/3 hours.	Human activity trace
Failure Frequency	02 times / day	[Norris, 2022]
Failure Type	High Variance, Stuck-at	-
Failure Distribution	Uniform	CloudSimPlus[22], [Nita, 2014]
Failure Detection Time	Negligible	-
Time to repair Legacy solution	15-20 minutes	Orange Customer Care Service

FMSim allows us to inject cascading failures (scenarios 1–6 described in Table 5.2) in simulated IoT infrastructures with different configurations (see Figure 5.9). The different simulation parameters are described in Table 5.6.

Relying on these configurations, we 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 Figure 5.10 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. Simulation traces are available from GitHub [23] to allow result reproducibility.

3.3.3 IoT-F Quantitative Evaluation

We assessed the quality of the IoT-F ontology using the *OntoQA* methodology, which we have used to evaluate the IoT-D ontology (see Chapter 4 Section 4.2.2).

We compared the OntoQA results of the IoT-F ontology with the FOLIO ontology results. The latter is the only found ontology representing failures among IoT devices [Steenwinckel, 2018]. The result (see Table 5.7) shows that the IoT-F ontology outperforms the FOLIO ontology for the IR and AR OntoQA metrics. This signifies that our ontology represents a wider range of knowledge and more knowledge per instance than FOLIO ontology, allowing richer failure knowledge representation. The IoT-F ontology has a lower RR i.e., it includes more *rdfs:subclass* relationships than object property relationships, as it provides a richer taxonomy of failures and recovery actions compared to the FOLIO ontology.

Table 5.7: OntoQA Evaluation results IoT-F ontology.

Ontology	C	SC	AT	P	RR	AR	IR
FOLIO	29	22	3	19	0,46	0,10	0,76
IoT-F	38	33	13	5	**0,13**	**0,34**	**0,87**

[23] ISWCMaterial/SimulationTraces

4 Conclusion

In this chapter, 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.

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 [Emmanouilidis, 2020], 3) the reuse of legacy platform features within the OSAMA agent for monitoring and recovery to save costs and accelerate integration efforts, and 4) the respect of legacy failure management data and processes governance within each DM actor.

In the upcoming chapter, we will showcase a tangible demonstration of our work: the *Collaborative LAN Troubleshooting*. This demonstration has been developed in collaboration with other teams at Orange Innovation, aiming to exemplify and apply the outcomes of our current research within an industrial context at Orange.

Chapter 6

Collaborative LAN Troubleshooting Demonstration

Summary

In this chapter, we shed light on the concrete demonstration of our work, the *Collaborative LAN Troubleshooting demonstration*. This innovative demonstration was developed in partnership with other teams at Orange Innovation to effectively highlight the outcomes of our current research within an industrial context at Orange.

Contents

1 Context and Motivation

Enhancing the *Quality of Experience* (QoE) stands as a pivotal component of the Orange strategy. In this context, our proposed *Collaborative LAN Troubleshooting demonstration* unveils a prototype of a transformative solution. Indeed, our main goal through this demonstration is to elucidate how we can empower customer care services, led primarily by Orange, through a cutting-edge web-based supervision tool. This tool is designed to provide customer care agents with the means to efficiently manage cascading failures. By doing so, we aim to boost *Orange* capacity to address customer concerns effectively.

More precisely, this demonstration integrates the contributions proposed by the present research to enable both human-based and automated management of cascading failure in a

simulated 3D smart home build with the *Orange Home Simulator*.

In the following, we will present the technical architecture of the demonstration. Then, we discuss a user story to highlight the usefulness of the proposed solutions in enhancing customer care services.

2 Technical Architecture

As mentioned above, the proposed demonstration integrates the results of the presented work into a web-based supervision tool to manage a simulated 3D Smart Home. The technical architecture is presented in Figure 6.1 . It includes six (06) components, namely *Orange Home 3D Simulator*, *Planner*, *Dependency Calculator*, *OSAMA supervision agents*, *Supervision UI*, and *Request scheduler*:

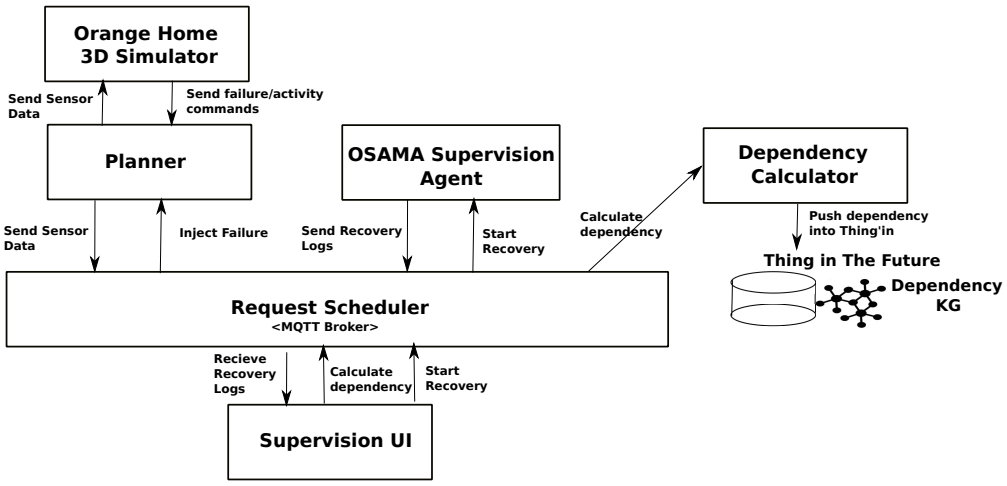
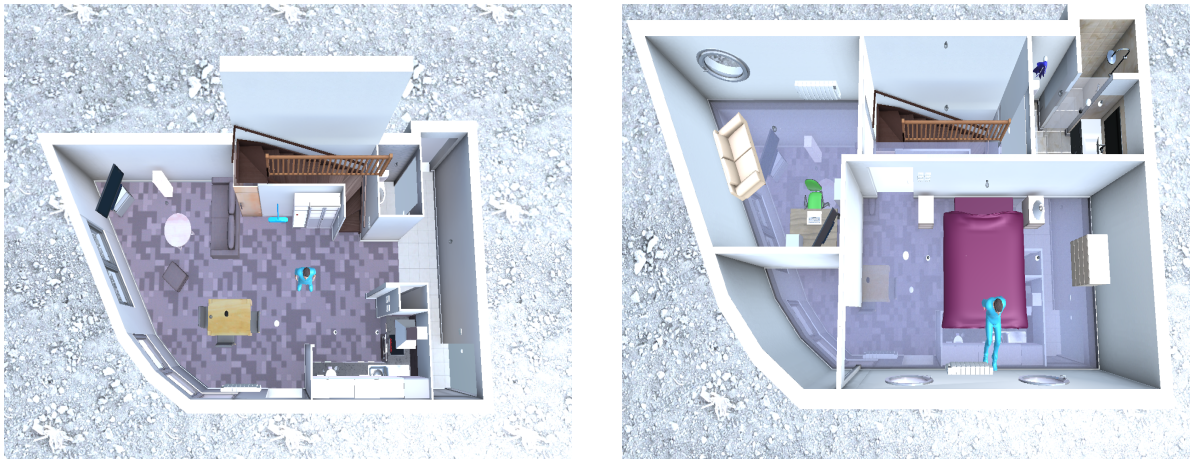


Figure 6.1: Collaborative LAN troubleshooting - Technical Architecture

- *Orange Home 3D Simulator*: represents a simulated 3D home, developed using Unity technology[1], that aims at generating simulated data at a smart home through the simulation of human activity and their interactions with simulated sensors and actuators (see Figure 6.2). It includes two (02) floors with six (06) rooms: a living room, a kitchen, a toilet, a bathroom, an office, and a bedroom, connected by a staircase, a walkway, and an entrance. These rooms include *115 IoT devices* connected with a gateway and WiFi-repeater. The simulated smart home includes a user called *Malcolm* who interacts with the different IoT devices within his daily activities. We assume that devices in the simulated smart home are managed by four (04) actors: *Orange*, *Philips*, *Kelvin Technology*, and *Amazon*.

- *Planner:* represents a *Python script* allowing to plan the activities of *Malcolm* and the IoT devices in the simulated 3D home by sending specific commands to trigger various events. For instance, the command *sendEvent("Relation:ActivitySit", [] , ["Malcolm", "Sofa"])* set the state of *Malcolm* in sitting on the living room sofa.

[1] `https://unity.com/fr`



(a) Floor 01 (b) Floor 02

Figure 6.2: Orange Home 3D Simulator.

- *Dependency Calculator:* represents our framework for IoT dependency inference presented in Chapter 4, which allows on-demand inference of dependency relationships in the simulated 3D home.
- *OSAMA supervision agents:* represents our solution for automatic *Cascading Failure Management* (CFM) presented in Chapter 5, allowing automatic and collaborative CFM on the simulated 3D home. This component includes four (04) *cOllaborative caScading fAilure Management Agent* (OSAMA) supervision agents associated with each DM actor, namely *OSAMA-Orange*, *OSAMA-Philips*, *OSAMA-Kelvin*, and *OSAMA-Amazon*.
- *Supervision UI:* represents a web interface provided for customer care technicians to monitor devices in the simulated 3D home by visualizing device data (see Figure 6.3). Upon failure, the customer care technician may perform failure diagnosis using the dependency calculator (see Figure 6.4) to identify the failure root cause or make the call to the OSAMA supervision agents for automatic CFM (see Figure 6.5).
- *Request scheduler:* represents an *MQTT Broker* to link the Supervision UI with the different components to allow displaying IoT device data and launching dependency calculation and OSAMA-based failure recovery.

3 Customer Care Agent Assistance: A User Story

This section shows how a customer care agent may use the proposed tool to solve cascading failure dilemmas based on the simulated 3D smart home.

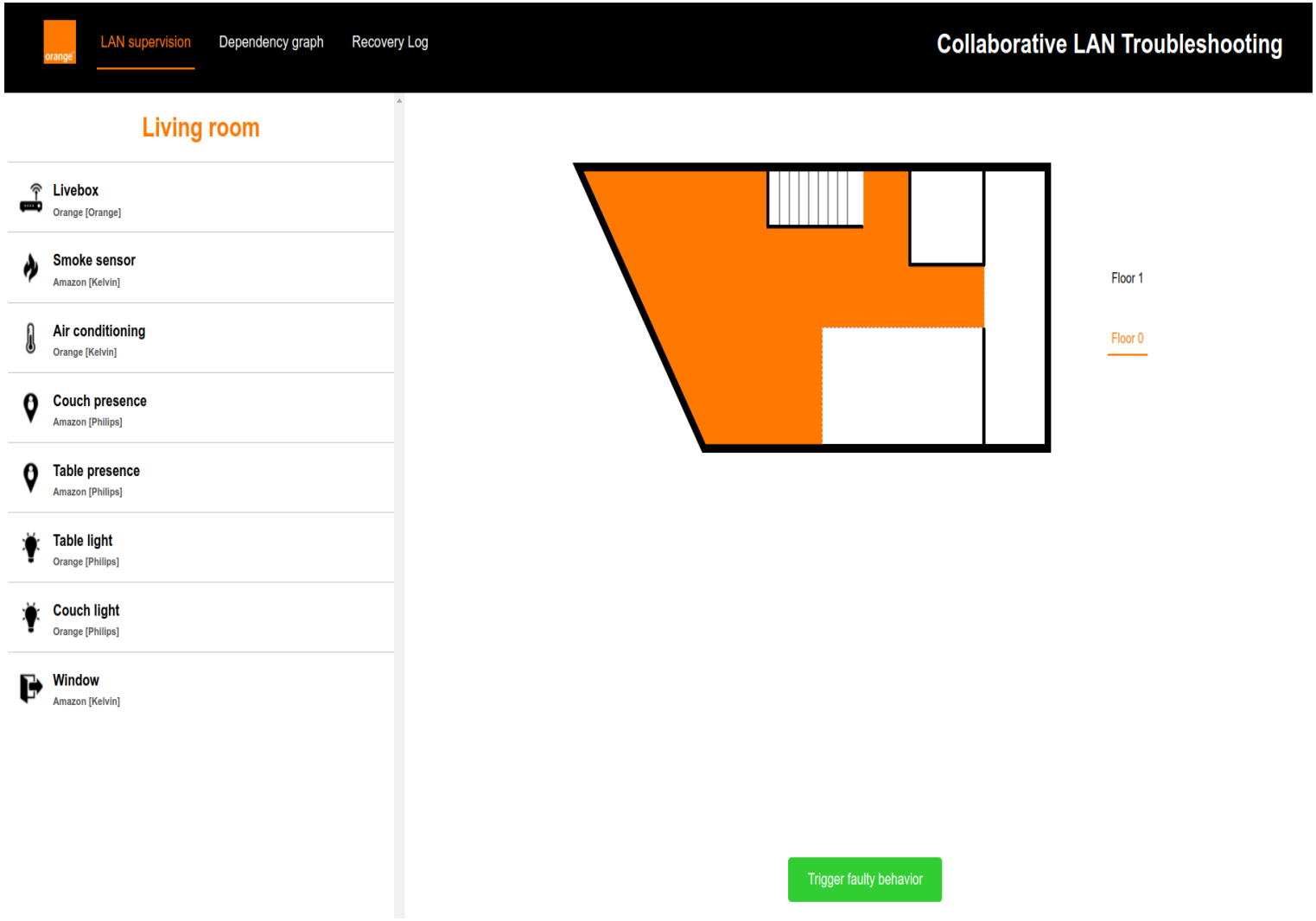


Figure 6.3: Supervision UI describing devices in the 3D Home simulators. Device state and outputs are also displayed within this view to allow the detection of failures in sensor data such as high variance.

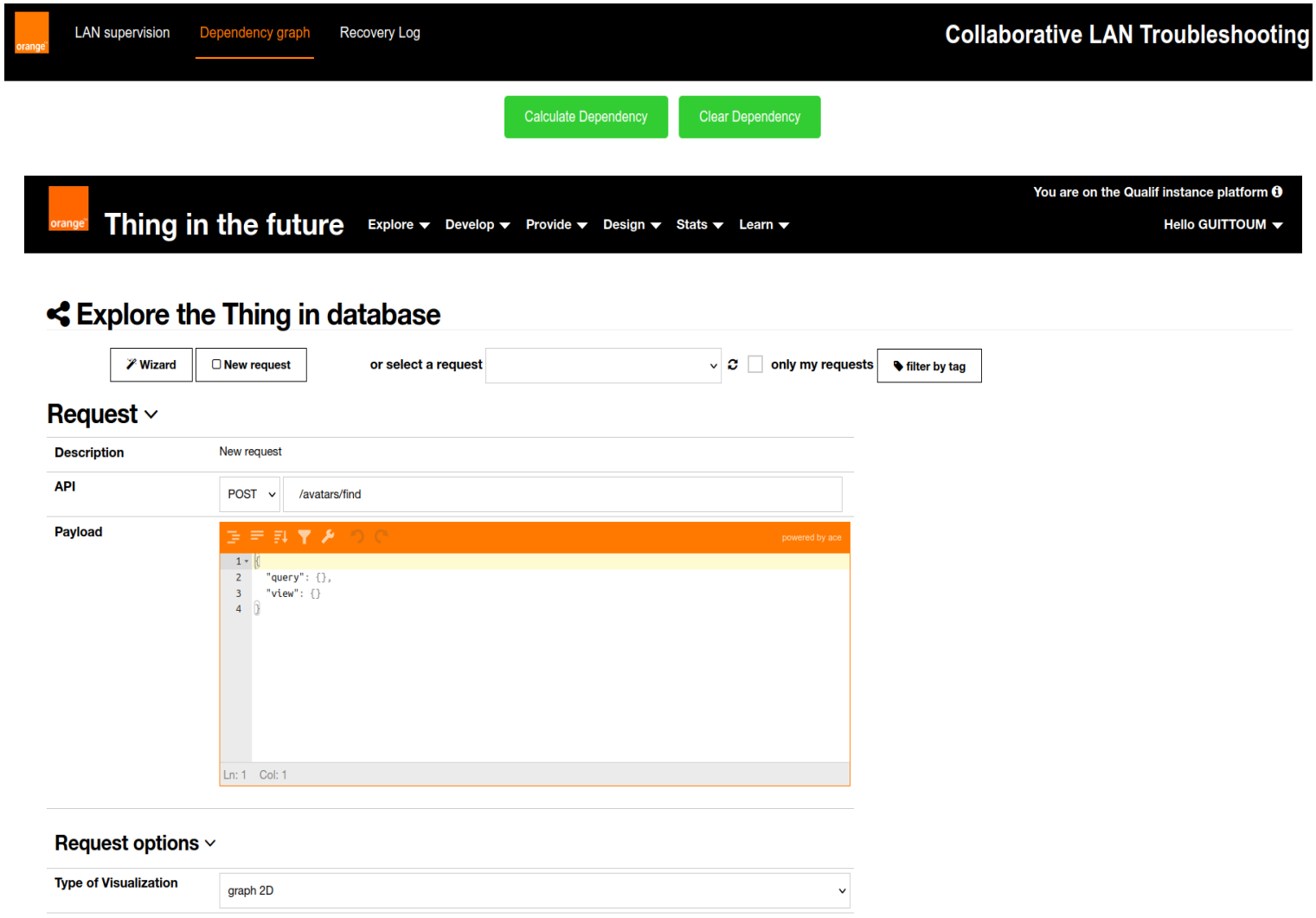


Figure 6.4: Dependency Calculator UI currently embeds the Thing in platform to allow dependency inference on the 3D simulated Smart Home. It allows the query of the *IoT Dependency Knowledge Graph* (DKG) through the Thing in platform.

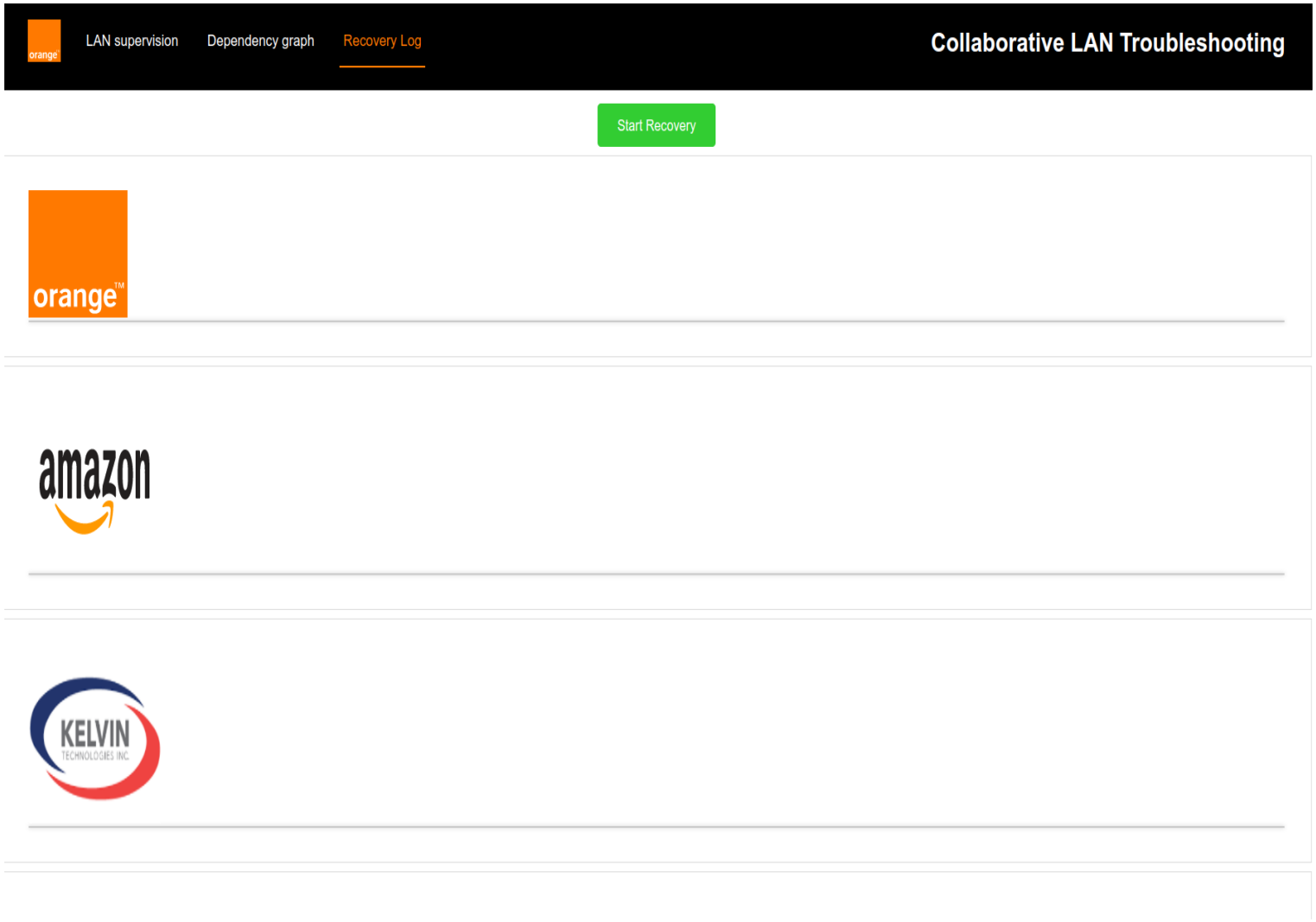


Figure 6.5: OSAMA Supervision Agent UI integrate the four (04) OSAMA agents managing the simulated 3D home namely Orange, Amazon, Phillips and Kelvin. It displays recovery logs including message exchange between the different OSAMA agents.

3.1 Targeted Cascading Failure Scenario

Malcolm, the user of the smart home, noticed the alarm device installed in the entrance of his home starts to behave in a chaotic manner. As the alarm device is managed by *Orange*, *Malcolm* calls the Orange customer care service to inquire about the alarm recovery.

3.2 Dependency Calculation

To address this issue, the customer care agent relies on our proposed tool and initiates the resolution process by calculating the device dependency relationships within the smart home system using the suggested portals. This generates a comprehensive DKG that includes information about devices and their dependencies, stored within the Thing in platform as *Digital Twin* (DT) (see Figure 6.6). Subsequently, the customer care agent formulates a query based on the resulting DKG to specifically identify the devices on which the alarm relies. This query, detailed in the provided Listing 6.1, yields a visual representation (see Figure 6.7) illustrating the alarm's dependency on four smoke sensors installed in the living room, the bedroom, the office, and the kitchen.

```
1  {
2    "query": [
3     {
4             "$iri": "http://thingin.orange.com/
                 demoCollabTroubleShooting/entrance_alarm",
5            "->http://www.semanticweb.org/OrangeLab/ontologies/2021/9/
                 IoTD#hasStateDependencyTo" : "objects"
6      },
7      {
8            "$domain": "http://thingin.orange.com/
                 demoCollabTroubleShooting/",
9             "$alias": "objects"
10       }
11    ],
12 "view": {}
13 }
```

Listing 6.1: Dependency Identification Query.

3.3 Solving The Cascading Failure

To identify the source of the failure, the customer care technician may leverage the *Supervision UI* to monitor the output data originating from the smoke sensors. Nevertheless, this methodology may prove inefficient when tasked with assessing a substantial number of dependent devices. For such cases, the customer care agent can automatically recover this cascading failure through OSAMA supervision agents portal. This portal allows to display recovery logs (see Figure 6.8) allowing to explain the results of the recovery, in order to ensure the system coherence.

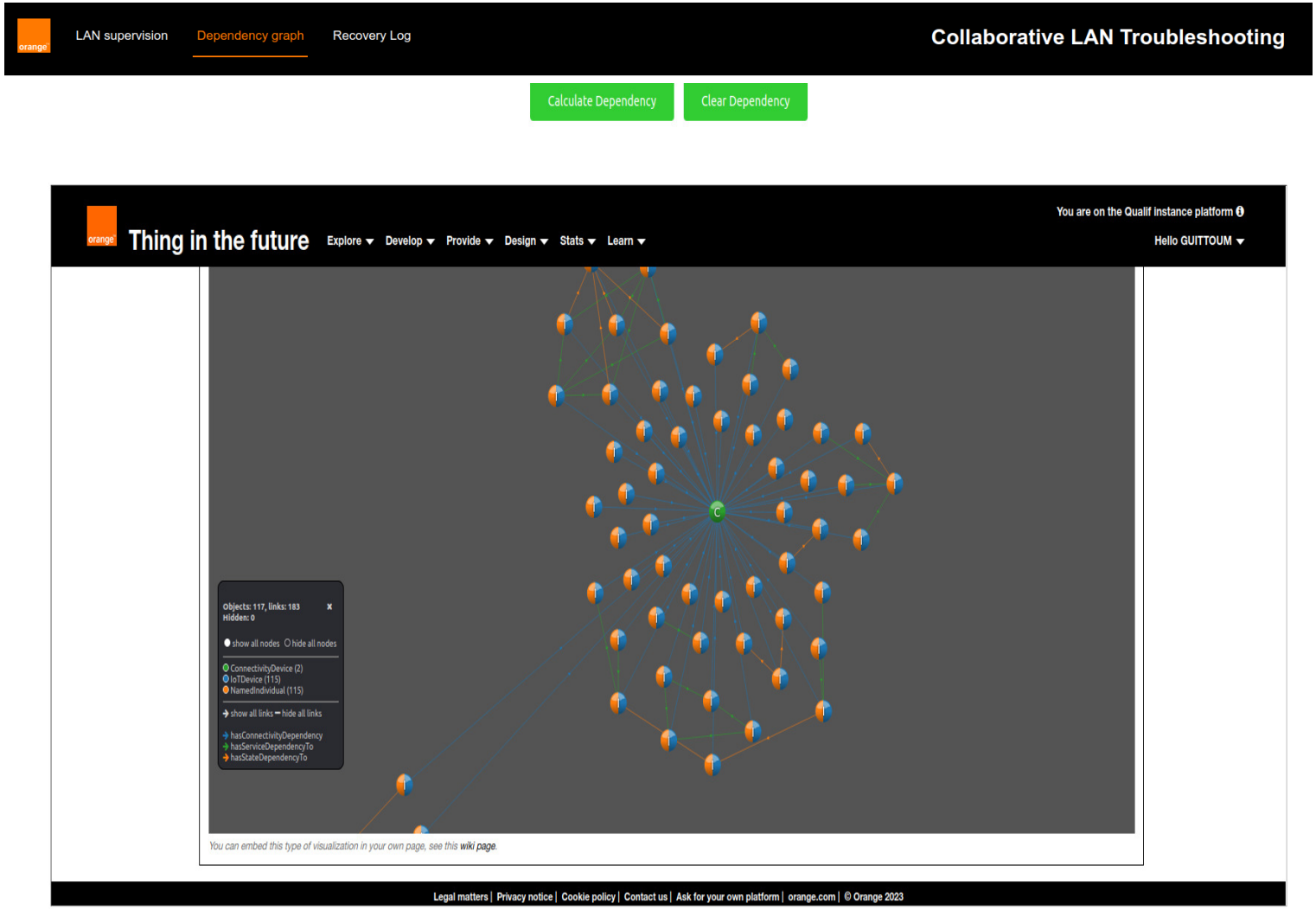


Figure 6.6: The inferred Dependency Topology, from the Simulated 3D Smart Home, described by the DKG. It includes 183 dependency relationships.

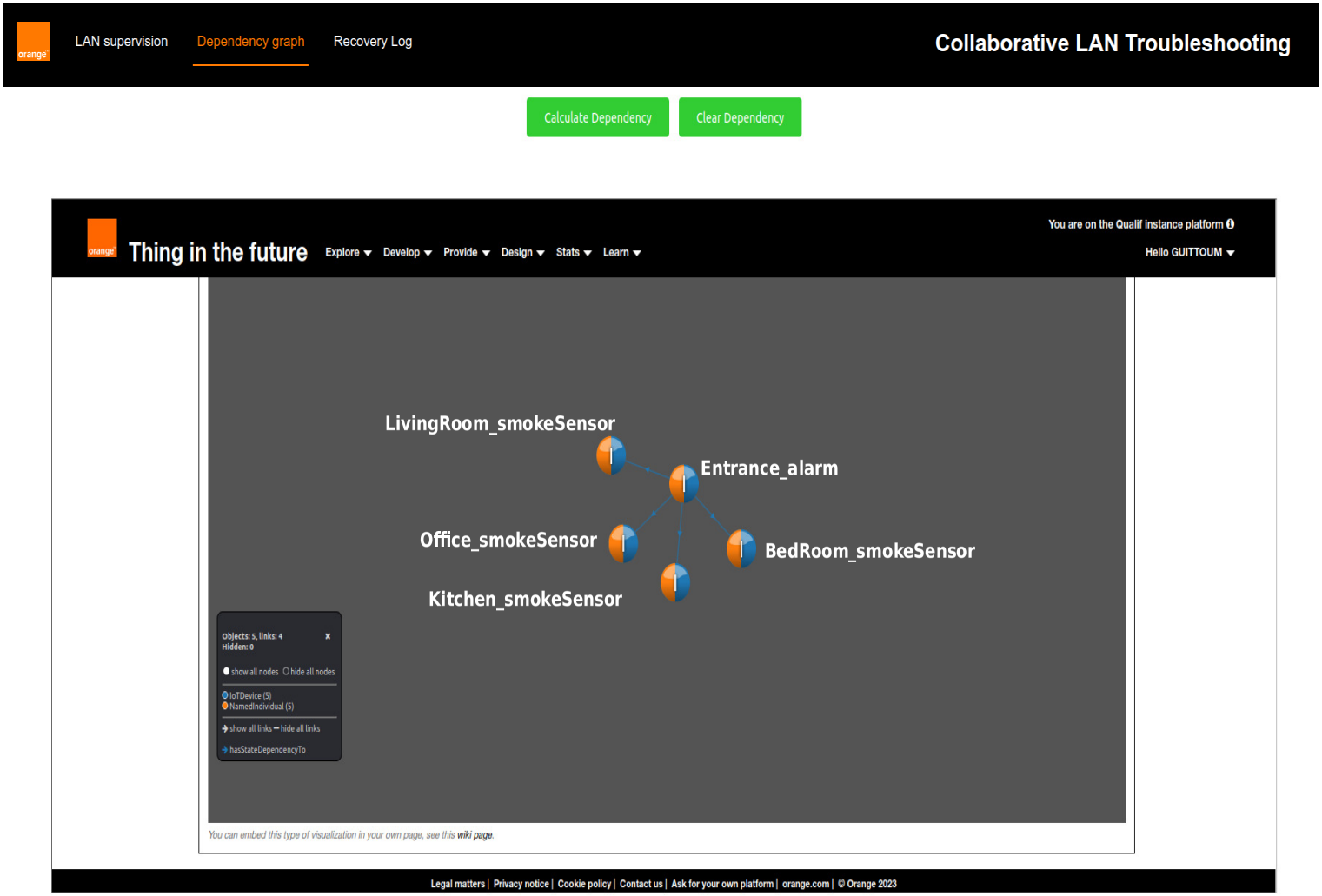


Figure 6.7: Dependency Topology of the alarm device, including the smoke sensors installed in the living room, the bedroom, the office, and the kitchen, which has state dependency to the alarm due to automation rules that launch the alarm upon detection of fire.

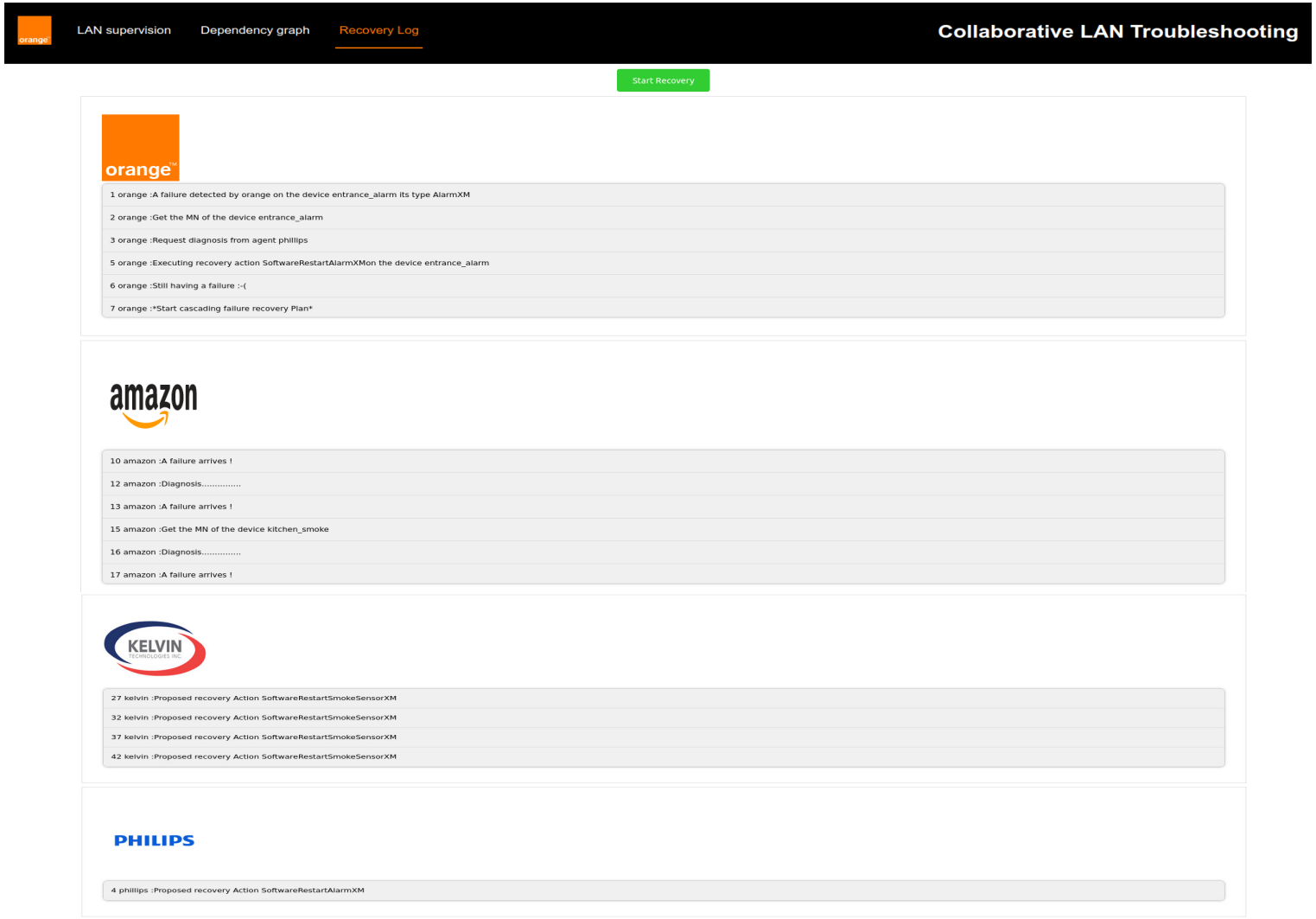


Figure 6.8: Recovery Log OSAMA Supervision Agent UI displaying message exchange between the OSAMA agents to solve the alarm failure.

4 Conclusion

In conclusion, the outlined chapter presented the Collaborative LAN Troubleshooting demonstration. The chapter has established the primary goal of this demonstration, which is to augment Orange's customer care services and then to improve Orange customers' satisfaction. The demonstration includes a web-based supervision tool, enabling the efficient management of cascading failures. We presented the technical architecture of the demonstration and a user story, illustrating the tangible and practical advantages of the proposed solutions in advancing and optimizing customer care services within the Orange company.

Chapter 7

Conclusion

Summary

This chapter concludes this Thesis by summarizing our key contributions. Perspectives and future work to extend the proposed approach are also discussed.

Contents

This Thesis proposed an automatic and collaborative cascading failure management approach for IoT devices managed by different DM actors. The approach aims to help market DM actors reduce customer care costs and time to repair failures for better customer quality of experience. In the following, we provide a summary of the proposed contributions. Then, we discuss directions for future work in order to enrich and optimize the proposed solution and favor its large adoption in the market.

1 Summary of Contributions

This section summarizes our contributions to design an automatic and collaborative *Cascading Failure Management* (CFM) approach for IoT devices managed by different DM actors. We describe our story of building practical solutions and demonstrations to address industrial challenges by relying on both theoretical assets provided by academia as well as cutting-edge technologies available in the market.

First, we started by building an extensive literature review from an industry and research perspective in which we identified research gaps in the IoT failure management domain. Moreover, we learned about cutting-edge technologies and tools and how they could be combined

to fulfill the identified research gaps. Then, we presented an interesting discussion about the combination of *Semantic Web* (SW), *Multi-Agent System* (MAS), and *Digital Twin* (DT) to enable autonomous decision-making on heterogeneous data governed by different actors and organizations.

Then, to ease failure root cause identification, we proposed a framework enabling automatic and collaborative inference of dependency relationships between IoT devices managed by different DM actors. The proposed framework relies on our proposed ontology named *Internet of Things Dependency* (IoT-D), facilitating a unified representation of IoT dependencies across heterogeneous DM solutions, to automatically construct the global dependency KG, a.k.a IoT dependency topology, through a sequential three-step process: Context Extraction, Entity Resolution, and Dependency Inference. The initial step extracts context data from existing DM solutions and transforms it into KGs, the second aggregates the extracted context KGs, and the final step deduces the IoT dependency topology from the aggregated context KGs. The global IoT dependency topology is exposed as a DT view for DM actors using the Orange DT platform *Thing In The Future* (Thing in).

We validated the proposed framework by inferring the IoT dependency topology in simulated and realistic smart home scenarios managed by multiple DM actors.

After that, we proposed the muti-OSAMA agents, a semantic MAS, to enable collaborative and automatic CFM on IoT devices managed by different actors. We developed *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 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 DT technology, 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. We proved the efficiency of this solution by solving several cascading failure dilemmas in simulated smart home scenarios managed by multiple DM actors. We also demonstrated its added value compared to legacy DM solutions according to time to repair failures and resource consumption in IoT infrastructures.

Relying on the proposed solutions, we developed the *Collaborative LAN Troubleshooting* demonstration that integrates our proposed contributions into a web-based supervision tool aiming to help in the management of cascading failures. Through this demonstration, we presented a user story illustrating the tangible and practical advantages of the proposed solutions in advancing and optimizing customer care services within the Orange company.

2 Perspectives

This section presents various paths to address the limitations, enhance the current work, and guide the trajectory for the large adoption of the proposed solutions in the DM market.

2.1 Short term perspectives

2.1.1 Agent-based extraction of IoT dependency topology

One of the main limitations of our framework is that DM actors should manually publish and maintain extraction modalities within our framework for IoT dependency inference. Despite being inspired by the ITU-T Recommendation **Y.4459** [ITU-T, 2020] for information exchange between isolated organizations in the form of digital entities, manual data sharing may impact the coherence of the generated context KGs and the fidelity of the constructed digital twins. Moreover, sharing information describing different dependencies among IoT devices may introduce problems regarding data governance for DM actors.

An alternative solution for these limitations is to provide OSAMA agents with proactive capabilities either for maintaining the extraction modalities or to deploy and maintain distributed digital twins describing dependencies in their systems. In this case, OSAMA agents may query digital twins of each other for failure root cause identification.

2.1.2 More shared artifacts to value Orange Home Services

In the current stage of our solution, we have proposed the shared artifact *Dependency artifact* for DM actors, allowing to value digital twins and federation services provided by *Orange*. Other shared artifacts may be proposed relying on *Orange* platforms and services such as *anomaly detection artifact*, an Orange service to detect anomalies on IoT devices, to help DM actors detect anomalies using network traffic analysis on the Orange *LiveBox*. Another proposed artifact would be the *Flamingo artifact*, integrating the Orange platform *Flamingo* which is an innovative platform allowing to execute DM operations on multiple DM platforms. In this case, DM actors should integrate their DM platforms into the *Flamingo platform* so that the identified recovery actions will be sent to the *Flamingo platform*, where they will be executed in coherent order on the dependent IoT devices. The use of the *Flamingo artifact* may be also useful for devices that are not integrated into a specific DM platform. In this case, proxy capabilities of the *Flamingo platform* may be used to integrate them and enable their remote recovery.

2.1.3 Declarative RDF generation using RML

Another limitation in our proposed solutions is the use of dedicated scripts to generate RDF triples of the context KG, which adds barriers to integrating DM data sources using new technologies and data models. Alternatives to address this problem could explore the use of RDF Mapping Language (RML) rules [Dimou, 2014]. They are based on a fully declarative approach for the KG generation process, which is adaptable to additional data sources [Rojas, 2021a]. This reduces maintenance and integration costs.

2.1.4 Verification of the CFM protocol

Ensuring the coherence of the CFM protocol in recovering cascading failures on interdependent IoT devices is necessary to consider to ensure the consistency of IoT infrastructures managed by OSAMA agents. This may include the consideration of tricky communication situations between OSAMA agents when executing the CFM protocol. For instance, consider the deadlock situations where a set of OSAMA agents are blocked because each OSAMA agent is receiving a CFM request and waiting for another CFM request answer occupied by some other OSAMA agents. We have studied some of these tricky situations using an experimental approach. The results are provided in Appendix B. We found that some of these situations may be handled by default within the *JaCaMo framework*. Other cases should be considered at the development level. A potential path to formally verify the correctness of the CFM protocol would be the exploration of *Model Checking approaches* [Ozeer, 2019].

2.1.5 Tests on realistic scenarios

We validated our solutions on simulated IoT infrastructures using generated failure information. To further prove the efficiency of the proposed solution, tests on realistic IoT infrastructures should be considered, with the use of realistic DM platforms such as *Amazon Web Service (AWS)* and *LiveObjects*, and the consideration of realistic failure information that may be retrieved within customer care services in *Orange* or *Orange partners*.

2.2 Medium term perspectives

2.2.1 Handle uncertainty using Neuro-Symbolic AI

A limitation of our proposed approaches for ER and dependency inference is that it relies on symbolic AI, which involves manipulating symbols based on predefined rules, and is often used to represent and reason about knowledge in a structured manner. However, real-world knowledge is inherently uncertain. Neuro-symbolic AI can incorporate uncertainty into symbolic representations, allowing for the modeling of imprecise or incomplete information. Neuro-symbolic AI is an interdisciplinary approach that combines elements of symbolic reasoning with neural network-based machine learning techniques [Garcez, 2023]. This integration allows for the representation and manipulation of symbolic knowledge alongside the ability to learn from data, making it well-suited to address uncertainty in various domains. To enable this integration for our case, we could leverage the knowledge graph embedding with link prediction and knowledge completion machine learning models to enable ER and dependency inference [Rivas, 2022].

2.2.2 CFM protocol optimization using learning

The performance of the CFM protocol could be optimized by reducing message exchange between OSAMA agents. A promising solution for that is to provide OSAMA agents with learning capabilities. For instance, OSAMA agents associated with the DMP profile may leverage federative learning capabilities to build a model that learns on FKB of multiple MN OSAMA agents

to avoid message exchange between DMP OSAMA agents and MN OSAMA agents for failure diagnosis. Another solution is to use machine learning models to predict recovery actions based on previous cascading failure recovery experiences given a failed device. This would avoid performing failure diagnosis and IoT dependency topology exploration for certain cascading failure scenarios.

2.2.3 Automatic Extraction of failure information

In our solution, we assume that FKB is built manually by MN experts. However, this task could be time-consuming and impact the overall maintainability of the proposed solution. An alternative solution could be to propose an automatic approach for FKB construction. This approach may leverage natural language processing capabilities such as large language models for named entity recognition [Li, 2020b] and relation extraction [Nasar, 2021], or any other text extraction technologies such as optical character recognition. It can be provided as an additional private artifact for MN OSAMA agents to allow them to automatically construct their FKB.

2.2.4 Handling multiple data store queries using Federative SPARQL

In our solution we consider that each device manufacturer acquires one FKB that can be accessed using SPARQL queries. To handle the situation where one device manufacturer publishes multiple FKB that may be distributed on the web, SPARQL Federated Query [1] may be used, allowing OSAMA agents to query multiple and distributed FKB. SPARQL Federated Query is recognized as a W3C extending SPARQL for executing queries distributed over different SPARQL endpoints.

2.2.5 Enabling an effective data governance using the Solid Framework

During the 28th birthday of the Web, Tim Berners-Lee, the inventor of the Web, wrote the blog *Three challenges for the Web, according to its inventor* [2], explaining that one of the main challenges of the Web is that the Web users are losing control of their personal data. Indeed, the existing structure of the Web results in the centralization of power, with dominant entities like Google or Facebook acting as identity providers. These centralized actors offer free services, leveraging access to our personal data, which they then monetize. Not only do they manage our identities within their own platforms, but they also extend this role to numerous third-party applications. Consequently, these major players gather extensive data on our social interactions and service consumption, extending well beyond their own primary offerings. To address this issue, the Solid framework[3] was created by Tim Berners-Lee at MIT, and it is now supported by a W3C community group. Solid is an API standard that lets users store their data securely in decentralized data stores, which can be shared with multiple applications according to specific access control policies managed by the users themselves. This potential benefit of Solid for individual users can be extended for organizations, enabling the decentralization of information

[1] https://www.w3.org/TR/sparql11-federated-query/
[2] https://webfoundation.org/2017/03/web-turns-28-letter/
[3] https://solidproject.org/about

and cooperative processes across siloed organizations. This benefit could open the door for our solution to leverage the Solid framework as a tool for data governance and customized collaboration across siloed DM solutions. Indeed, heterogeneous DM actors may adopt the Solid framework to manage the access of the shared information related to failure and dependencies. Thus, data access may be enabled according to the degree of partnership between different DM actors.

2.2.6 Enhancing the traceability of OSAMA agents

To facilitate the integration of OSAMA agents into customer care services, ensuring their traceability is crucial. In the demonstration detailed in Chapter 6, we illustrated how this could be accomplished by logging the message exchanges of OSAMA agents during the failure recovery process. To manage these logs more effectively, a knowledge graph can be utilized to store the log data. This knowledge graph can be generated using the standardized ontology, PROV-O [4].

2.3 Long term perspectives

2.3.1 Cascading Failure Tolerance, Prediction, and Prevention

Our solution allows for cascading failure recovery when they appear. This means that the failed IoT device or service will still be unavailable until the failure is recovered, which would impact customer quality of service. To address this limitation, a cascading Failure tolerance approach could extend our solution by failure tolerance techniques allowing to keep IoT services available even in the presence of failures. One potential solution is to adapt failure tolerance techniques proposed for distributed systems to the IoT context and the specifications of our solution (see Chapter 3 Section 2). For instance, the use of checkpointing to save the correct state of IoT devices to restore them in case of failure. This technique may be integrated into our solution using a shared artifact that leverages the knowledge graph historization features provided by the Thing in platform to store IoT devices correct states. The historized knowledge graph may be fed by accessing siloed DM solutions. Other failure tolerance techniques may be considered such as *device replacement* by discovering devices providing the same features as the failed device using the Thing in platform. The *device reconfiguration* technique may be used to automatically replace the failed devices by coordinating DM operations on multiple DM solutions. It may be also used to reconfigure the IoT system to enable *cascading failure prevention*. *Device replacement and reconfiguration* may be integrated also as shared artifacts to be accessed by OSAMA agents for *cascading failure tolerance and prevention*. These approaches may be enforced through failure isolation techniques such as using the dependency KG to identify dependencies that could be temporarily deactivated during failure recovery to stop the cascade and maintain some level of operation. Moreover, different *cascading failure management* historical data may be leveraged to predict the occurrence of cascading failure so that they can be prevented.

[4] https://www.w3.org/TR/prov-o/

2.3.2 Integration of end users as an effective DM actor

In its current form, our solution targets DM actors as the main client, which corresponds to a B2B business model. In some practical scenarios, end users are provided with mobile applications that allow them to perform DM operations on their IoT devices. In other cases, IoT devices are deployed by end users and are not integrated within a DM solution. These particular scenarios need to be handled in our solution through the adoption of a B2B2C model, where end users are considered effective actors and involved in the collaborative CFM. An OSAMA agent may be associated with each end user to allow such transformation.

2.3.3 Toward Standardized IoT Failure Management

Since our solution involves the collaborative effort of multiple DM actors, the standardization path may be a potential plan to foster its large adoption in the DM market. This may be achieved by two main means: First, study the position of our solution regarding current standards and initiatives such as *Matter* [5] and *Prpl* [6], which are trendy initiatives in which *Orange* is involved. Second, 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 DM actors and experts. Another potential standardization plan would be the submission of IoT-D and IoT-F ontologies to the *ETSI* as an extension of the standardized ontology *SAREF* modeling the IoT failure management domain.

2.3.4 Exploring other Collaborative DM use cases

As mentioned in Chapter 1 Section 2.2, several issues are generated due to the siloed management of IoT devices by different DM actors. This brings us to investigate the *Collaborative DM* paradigm, which aims to break DM silos and foster collaborative and digital processes to address potential problems. In this work, we addressed the cascading failure problem. Future work may consider the exploration of other use cases of *Collaborative DM*, such as coordinating DM operations to avoid DM failures. This can be easily achieved by adopting our MAS architecture with specific artifacts and collaborative protocols.

[5] https://csa-iot.org/all-solutions/matter/
[6] https://prplfoundation.org

Appendix A

FMSim: IoT Failure Simulator

Summary

This part describes an experimental contribution of the thesis, which consists of the FMSim simulator for IoT failure injection and recovery simulation.

1 Introduction

The IoT has emerged as a transformative force, interconnecting an ever-expanding array of devices and systems to enable seamless communication and data exchange. As IoT deployments become increasingly integral to our daily lives and critical infrastructure, the need for robust and resilient systems is more paramount than ever. This need can be fulfilled through the enhancement of IoT infrastructure by failure management and tolerance approaches. However, validating such approaches at the design level is a challenging task. Indeed, failure injection in IoT devices may result from deliberate actions, such as interrupting the power supply using a smart plug for devices connected to wall sockets or utilizing an external power source modulator circuit for battery-powered devices. However, subjecting these devices to repeated failure injections for assessment purposes can be detrimental, leading to irreversible damage and definite failure. In such scenarios, simulating the failure of a device becomes crucial for inferential analysis. There are some practical techniques for failure simulation such as employing a Faraday cage can block wireless communication for a wireless IoT device, and deliberately shutting down the network interface of an IoT device allows the failure detection mechanism to deduce its malfunction [Ozeer, 2019]. However, these techniques do not allow to simulate all type of failures on IoT devices such as non-fail-stop failures (see Chapter 3 Section 3.1).

This chapter presents the new simulator *FMSim* tailored for IoT environments, specifically designed to simulate IoT failure injection and recovery. This simulator is an experimental contribution of our work that extends the iFogSim simulator [Gupta, 2016], a simulator for IoT devices and applications, to allow failure injection and recovery simulation as well as the

quantification of resource consumption on an IoT or Fog computing infrastructure in the presence of failures. The main goal is to allow the quantification of resource loss generated due to time-consuming failures such as *High Variance Failure* (see Chapter 3 Section 3.1) as well as the impact of failure recovery approaches in reducing such resource loss.

2 IoT simulators: State of the Art

On our road to building the *FMSim* simulator, we studied several simulators in the literature according to several specifications (a.k.a simulation capabilities) representing different aspects that we would like to simulate in order to validate our failure management solution: 1) the capability to simulate IoT devices and their data; 2) the capability to simulate IoT applications, especially the trigger-action platforms; 3) the capability to simulate dependencies between IoT devices; 4) the capability to simulate failures and cascading failures; 5) the capability to simulate recovery actions; 6) the capability to interact with simulator components during the simulation.

Our study relied on relevant research surveys in the area [Nayyar, 2015; DAngelo, 2016; Chernyshev, 2017; Patel, 2019]. The results are depicted in Table A.1 describing the different simulators we studied, compared according to our specifications. We concluded that there is no simulator that covers all our specifications. This gap can be filled by building a multi-level simulator that extends and combines existing simulators to address multiple specifications in the simulation model.

Table A.1: Study of IoT Simulators

Capabilities /Simulators	Sensor	Actuator	IoT application	Dependencies	Failure injection and recovery	Interactive	Open-Source
IoTIFY[1]	✓	✓	✗	✓	✗	✓	✗
Bevywise IoT[2]	✓	✓	✗	✓	✗	✗	✗
Cooja[3]	✓	✗	✗	✓	✗	✗	✓
YAFS[4]	✗	✗	✗	✓	✓	✗	✓
Fogify[5]	✗	✗	✗	✓	✓	✓	✓
TiedNets[6]	✗	✗	✗	✓	✓	✗	✓
Tiger[7]	✗	✗	✗	✓	✓	✗	✓
NS-3[8]	✓	✗	✗	✓	✗	✗	✓
NetSim[9]	✓	✗	✗	✓	✗	✗	✗
CloudSim[10]	✓	✗	✗	✓	✗	✗	✓
CloudSimPlus[11]	✓	✗	✗	✓	✓	✗	✓
iFogSim[12]	✓	✓	✓	✓	✗	✗	✓

We have chosen to extend the open-source simulator *iFogSim* by failure injection and recovery capabilities since it is the simulator that covers almost all of our specifications and it is easy to

[1] https://docs.iotify.io/
[2] https://www.bevywise.com/iot-simulator/
[3] \https://ns3simulation.com/contiki-cooja-simulator/
[4] \https://yafs.readthedocs.io/en/latest/
[5] https://ucy-linc-lab.github.io/fogify/
[6] \https://github.com/TiedNets/TiedNets
[7] \https://graph-tiger.readthedocs.io/en/latest/index.html
[8] \https://www.nsnam.org/
[9] \https://www.tetcos.com/index.html
[10] \https://github.com/Cloudslab/cloudsim
[11] \https://cloudsimplus.org/
[12] \https://github.com/Cloudslab/iFogSim1

extend [Mahmud, 2022]. Moreover, it allows reproducible evaluation of resource consumption in IoT infrastructure using various performance parameters such as energy consumption, latency, response time, and network usage [Bala, 2020].

3 FMSim, an iFogSim extension

iFogSim is an open-source simulation toolkit developed specifically for modeling and simulating fog computing environments. Fog computing extends cloud computing capabilities to the edge of the network, closer to where data is generated and consumed. iFogSim allows researchers and developers to design, model, and evaluate fog computing applications and infrastructures. It is designed to simulate and study fog computing scenarios, considering factors such as data processing at the edge, efficient resource management, and the dynamic nature of fog environments. It provides a framework for modeling various aspects of fog computing, including IoT devices, communication networks, and IoT application workflows. It is designed to be scalable, allowing users to simulate large-scale fog computing infrastructures and evaluate the performance of applications in such environments. It allows for modeling and analysis of the energy consumption of fog nodes, which is crucial for assessing the sustainability and efficiency of fog computing systems. iFogSim is built on CloudSim, another popular open-source cloud computing simulation toolkit. This integration enables the modeling of interactions between fog and cloud computing resources. These relevant features make iFogSim widely used simulators by researchers and practitioners to explore and experiment with various fog computing scenarios, optimize resource allocation, and develop efficient algorithms for task scheduling and management in distributed edge environments.

iFogSim's technical architecture comprises three components, namely: *physical component*, *logical component*, and *management component* (see Figure A.1). Physical components, including sensors that generate tuples, akin to tasks in cloud computing, through event-driven task generation. Intervals between tuples are determined by a specific distribution during sensor creation. Actuators, on the other hand, receive these tuples. The logical components involve AppModule, responsible for receiving, processing, and sending tuples, while AppEdge facilitates linking and synchronization between two app modules and their input/output tuples based on a fractional selectivity. AppLoop calculates end-to-end latency from one AppModule to another. In the management component, the Controller oversees simulation execution, and the Mapping Module Object establishes links between application modules. Controller, Sensor, and Actuator inherit from the abstract component of *CloudSim* simulator *SimEntity*, while Tuple is inherent from *SimEvent*. Relying on the same logic of extending CloudSim to iFogSim, we introduced new simulation entities that inherit from the *SimEntity* and *SimEvent* as well as *iFogSim* component to model failure injection and recovery. We have been inspired by the failure injection method proposed in the simulator *CloudSimPlus* that alters simulation data streams by injecting failure events according to a probabilistic distribution [Nita, 2014].

More precisely, we introduced *FailureInjector* and *FailureRecoverer* extending *SimEntity* to inject and recover failure events represented by *Failure* that inherits from *SimEvent* (see Fig-

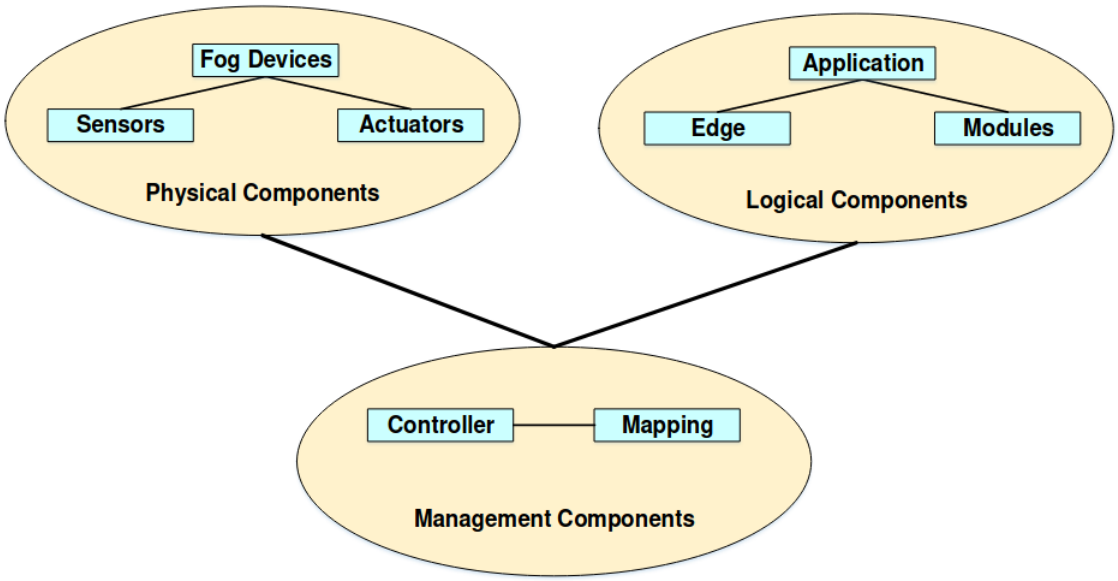


Figure A.1: Components of iFogSim [Gupta, 2016]

ure A.2). In order to integrate these entities into the other simulation entities in iFogSim, we added *FailedSensor* and *FailedActuator* that extend the iFogSim components *Sensor* and *Actuator* to handle failed events and recovery actions received from *FailureInjector* and *FailureRecoverer*. Moreover, we introduced *FailureController* allowing to program failures and their recovery on the simulator according to a specified distribution such as deterministic distribution. We note that only high variance and stuck-at failures are considered in the current version of the simulator. This could be easily extended to consider other failures, such as outliers and fail-stop failures, thanks to our generic conceptual model.

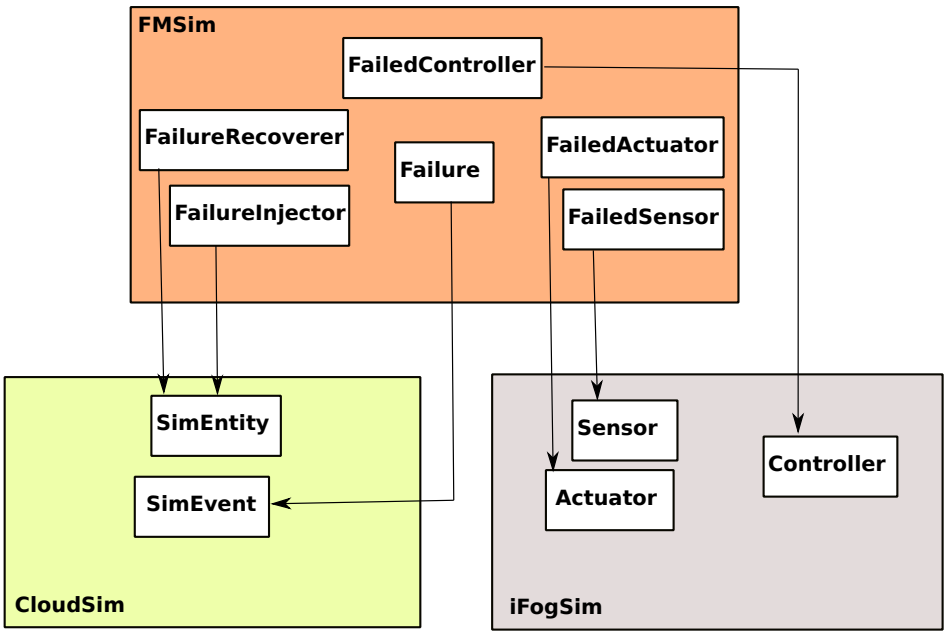


Figure A.2: FMSim conceptual model

4 Conclusion and Perspectives

In this chapter, we presented the FMSim, a simulator for IoT failure management. FMSim extends the well-known simulator *iFogSim* through failure entities. We have been inspired by the failure injection method proposed in the simulator *CloudSimPlus* that alters simulation data streams by injecting failure events according to a probabilistic distribution [Nita, 2014]. We simulate failure recovery by deleting failure events from the simulation data stream. FMSim allows for validating failure management approaches and quantifying their impact on resource consumption of IoT infrastructures.

In its current version, FMSim considers a limited type of IoT failure. However, this may easily handled thanks to its extensible design. Moreover, the current version is not interactive, i.e., failure management approaches can not interact with simulation entities at runtime. The simulation should be programmed after the execution of failure management approaches and the use of time to repair failures as a means to program recovery actions on simulation entities. Future work may handle this limitation by integrating interactive capabilities into simulation entities through *Threads* for example.

We note that the current version of the *FMSim* is available from GitHub [13].

[13] https://github.com/Orange-OpenSource/collaborativeDM-FM-Simulator

Appendix B

Correctness verification of the Collaborative CFM protocol

Summary

In this chapter, we present a correctness verification study of the proposed *Cascading Failure Management* (CFM) protocol through an experimental approach allowing the verification of its behavior regarding a set of challenging scenarios.

Contents

1 Introduction

As we saw in Chapter 5 Section 2.5, *cOllaborative caScading fAilure Management Agent* (OSAMA) agents rely on a distributed collaborative *Cascading Failure Management* (CFM) protocol to manage automatically cascading failures. Designing this protocol proves challenging and error-prone, primarily because of the extensively distributed nature of the IoT ecosystem and DM solutions, necessitating a distributed architecture for effective failure management. Consequently, the behaviors of distinct OSAMA agents run concurrently, adding complexity to the approach's design. As a result, validating the correctness of this distributed failure management approach becomes paramount. In this chapter, we check the correctness of the proposed CFM protocol through an experimental approach allowing the verification of its behavior regarding challenging scenarios. We provide interpretations for the verification results and highlight perspectives for enhancing the proposed CFM protocol. Finally, we provide a global discussion on the verification and the validation of the proposed CFM protocol using *model checking approaches*.

2 Challenging Scenarios

We elaborated a set of challenging scenarios describing tricky situations that OSAMA agents may encounter when executing the collaborative CFM protocol. We ran a series of experiments to verify the resilience of the collaborative CFM protocol regarding these challenging scenarios. Table B.1 provides interpretations of the experiment results and highlights perspectives for enhancing the proposed CFM protocol.

3 Discussion

The previous section showed how OSAMA agents perform within the collaborative CFM protocol regarding challenging scenarios such as *deadlocks*. We concluded that most of the presented challenges could be handled by our proposed solution thanks to our technical choice *JaCaMo*, which is well suited for concurrent and parallel computing allowing OSAMA to handle parallel CFM requests. Other challenging scenarios such as *deadlocks* could be easily handled at the development level, or by enhancing our solution by more features that are described in Chapter 7.

To go one step further in the verification and validation of our solution, future work could investigate the use of *model checking approaches*, which is a formal verification technique employed to ensure the correctness of complex systems by exhaustively exploring their state spaces and verifying whether they satisfy specified properties [Ozeer, 2019]. This approach involves the systematic examination of all possible states and transitions within a system model to determine whether it adheres to desired behaviors or constraints. There are several model-checking approaches, each with its own methodologies and advantages. Symbolic model checking represents states and transitions symbolically, leveraging decision procedures and binary decision diagrams for efficient exploration of the state space. Temporal logic model checking utilizes temporal logic to express properties, allowing for the verification of complex temporal relationships. Explicit state model checking explores the state space explicitly, employing algorithms like depth-first or breadth-first search. Bounded model checking focuses on a subset of the state space within fixed limits, providing efficient error detection within a constrained exploration. Probabilistic model checking extends the approach to handle systems with probabilistic transitions, accommodating uncertainty in system behavior. Hybrid model checking integrates different techniques to handle systems with both discrete and continuous components. The choice of model checking approach depends on the specific characteristics and requirements of the system under consideration, allowing for a tailored verification process [Clarke, 1997].

[1] https://www.w3.org/TR/shacl/

Table B.1: Challenging Scenarios for CFM protocol.

Challenge	Verification	Interpretation
Request Deadlock	✓(partially)	In the situation where two IoT devices (or more) are dependent on each other forming a cycle topology in such case OSAMA agents managing such devices will counter a request deadlock problem, which refers to the situation where two or more OSAMA agents are waiting for the response of each other for the same scenario of cascading failure. Such a challenge is partially solved in our solution, where only cycles of size two are handled by allowing OSAMA agents to check whether the received cascading failure request belongs to a device that the managed device depends on. In such case, the OSAMA agent will not send a request to this device instead it recovers the managed device and answers with a deadlock message. Cycle with a size greater than two needs to be considered. For that, we suggest handling them technically or checking that the dependency graph does not include cycles. This may be achieved through SHACL shapes for graph validation. [1]
Failure of one OSAMA agent or network distribution	✓	Time-out failures may occur when the requested OSAMA agent fails or due to network disruptions. This problem is solved thanks to the JaCaMo primitive *wait* allowing to define a time-out on a request. After the defined time out, the CFM request is stopped and the failure is reported to the customer care service.
Parallel detection of a cascading failure by multiple OSAMA agents	✓	Thanks to the concurrent and parallel exchange of messages enabled by the JaCaMo framework, OSAMA agents can handle parallel detection of a given cascading failure dilemma in a coherent manner. Nevertheless, in this scenario, devices undergo recovery processes multiple times for the same cascading failure. To address this concern, OSAMA may contemplate abstaining from executing recovery actions that arrive within the same time slot for the identical device.
Parallel CFM request arriving on the same OSAMA agent	✓	Thanks to the concurrent and parallel exchange of messages enabled by the JaCaMo framework, OSAMA agents can handle multiple cascading failure dilemmas simultaneously and in parallel manner.
No answers from the FKB	✓	In this situation, OSAMA agent notify the customer care services.
Multiple answers from the FKB	✓	In this situation, OSAMA agent choose the first proposed recovery action. This may be customized to choose the less costly recovery action or the more effective one based on the recovery history.
Parallel execution of DM operation (recovery actions)	To be handled	Parallel handling of multiple cascading failure dilemmas may result in chaotic executions of DM operations on IoT devices. This may generate DM failures on IoT devices (see Chapter 1 Section 2.2) , which may handled by providing OSAMA agents with more collaborative features allowing them to coordinate DM operations on IoT devices (see Chapter 7 Section 2.3.4).
No dependencies related to the failed device or Thing'in is not available	✓	In this situation, OSAMA agent notify the customer care services.
A device has no OSAMA-DMP (is not integrated in a given DM platform)	To be handled	This situation may be handled through the use of the Orange platform *Flamingo* with proxy features (see Chapter 7 Section 2.1.2) to integrate these devices.

Résumé en Français

1 Contexte et Problématique

Orange, en tant qu'opérateur de télécommunication, gère plus de 21 millions d'équipements aujourd'hui principalement dans le domaine de la Maison Intelligente avec les passerelles Internet et des décodeurs TV déployés chez ses clients. Ces équipements doivent être maintenus à jour et réparés à distance grâce à des solutions informatiques appelées *Device Management (DM)*, afin d'assurer une meilleure Qualité d'Expérience (Quality of Experience, QoE) aux clients. Parmi les indicateurs mesurant la QoE, nous trouverons le coût engendré par les appels au service client et sa capacité à répondre, dans les meilleurs délais, aux demandes des clients à la suite d'une panne survenue sur leurs équipements. Le service client classique repose sur une interaction avec le client via des appels téléphoniques: Le client déclare la panne survenue sur son équipement. Son interlocuteur utilise un ensemble d'outils pour identifier la source de la panne, et s'appuie sur le système de DM pour lancer à distance des opérations de réparation sur l'équipement. Si la panne persiste, l'interlocuteur programme une intervention technique. Cependant, ce processus engendre un coût non-négligeable, en termes de temps et d'argent, impactant la QoE de clients.

Les circonstances d'impact sur la QoE de clients et d'augmentation de ces coûts seront exacerbées avec l'avènement de l'Internet des objets (Internet of Things, IoT). En effet, la quantité d'équipement IoT en 2025 est estimée à plus de 70 milliards selon Statista. De plus, dans un système IoT complexe (i.e., usine intelligente, transport intelligent), les équipements IoT seront de plus en plus interconnectés : soit pour assurer la connectivité, soit pour échanger des informations. Ainsi, si une panne est survenue sur l'un des équipements, elle pourra être propagé en cascade et impactera le fonctionnement de ceux qui lui sont dépendants. La résolution des *pannes en cascade* est exacerbé lorsque les équipements sont gérés par des acteurs différent, ex., des opérateurs, des constructeurs d'équipements, et des fournisseurs de service, proposant chacun son propre solution DM qui permet de gérer les pannes sur ces équipements. Ces solutions DM isolée sont limités face au problème des pannes en cascade car ils manquent d'une connaissance sur les liens de dépendances entre les équipements et un contrôle global sur les équipements. Ce qui complique le diagnostique et la correction automatique des pannes en cascade.

Les pannes en cascade amèneront des dysfonctionnements sur d'autres équipements donc une dégradation de la QoE, plus d'appels et une augmentation des coûts du service client. De plus, les pannes en cascade engendrent des pertes d'énergie dans les environnements connectés: les statistiques ont montré que 25-45% d'énergie des systèmes intelligents de chauffage, ventilation et climatisation est gaspillée en raison de pannes [Najeh, 2019].

Dans ce cadre, cette thèse a pour objectif est d'aider les solutions DM existantes dans

la résolution des pannes en cascades. Plus précisément, nous visons à proposer une solution *collaborative et automatique* permettant aux acteurs DM de gérer automatiquement les pannes en cascade sur les équipements IoT interdépendants. Ceci afin d'assurer à la fois une meilleure QoE aux clients (minimiser le temps de réparation), minimiser l'énergie perdue à cause des pannes (enjeu environnemental), et assurer une meilleure maîtrise des coûts de service client (réduire le nombre d'appels et les interventions techniques).

2 Contributions

Pour atteindre notre objectif, nous avons commencé par l'étude de l'état de l'art pour mieux cerner et justifier nos questions de recherche. Et vu le contexte industriel, nous avons étudié la possibilité de réutiliser l'existant chez Orange et dans le marché en termes de plateformes et technologies pouvant servir et accélérer la construction de notre solution. En se basant sur les synthèses ressorties de l'état de l'art, nous avons pu proposer trois contributions qui seront présentées dans ce qui suit.

2.1 Un système de jumeau numérique sémantique pour l'inférence automatique des dépendances entre les équipements IoT

La première contribution permet de faciliter la gestion des pannes en cascade en identifiant automatiquement la topologie de dépendances entre les équipements IoT dans un environnement connecté donné ex. Maison intelligente. La topologie de dépendances, qui décrit les équipements IoT avec les liens de dépendances entre eux, sera utilisée par les acteurs DM pour identifier la source des pannes en cascade et faciliter leur diagnostic.

Notre approche se base sur l'utilisation d'un ensemble de standards de Web sémantique et la représentation des connaissances combinées avec la technologie de jumeaux numériques pour permettre une représentation interopérable et fidèle de la topologie de dépendances entre les équipements IoT. Plus précisément, la solution proposée repose sur une ontologie appelée *Internet of Things Dependency* (IoT-D) qui permet une représentation partagée des dépendances IoT à travers des solutions DM hétérogènes. L'ontologie IoT-D décrit un ensemble de données contextuelles décrivant les dépendances et les interactions entre les équipements IoT. En se basant sur l'ontologie IoT-D, notre solution construit automatiquement le graphe global de dépendances selon un processus en trois étapes (voir Figure B.1), à savoir *Extraction de contexte*, *Résolution d'entité*, et *Inférence de dépendance*. La première étape extrait les données de contexte depuis les solutions DM existantes et les transforme en graphe de connaissance, la deuxième agrège les graphes de contexte extraites, et la dernière infère la topologie des dépendances IoT à partir des graphes de contexte agrégés.

Notre solution s'appuie sur la plateforme de jumeau numérique d'Orange *Thing In The Future* (Thing in) et est conçu pour être intégré aux services client des acteurs DM en tant qu'outil d'aide à la décision pour aider à la gestion efficace des pannes en cascade, tels que l'identification de la cause d'une panne en cascade.

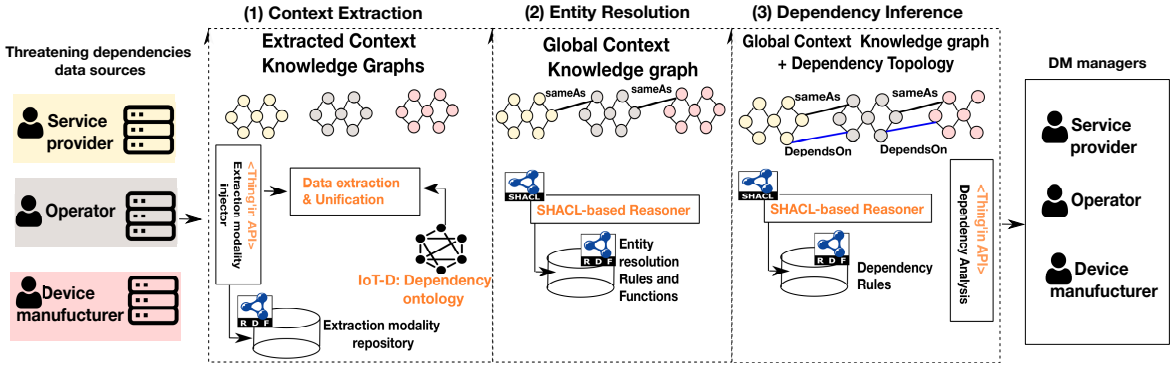


Figure B.1: Un système de jumeau numérique sémantique pour l'inférence automatique des dépendances entre les équipements IoT.

Nous avons développé une preuve de concept qui a permet d'identifier automatiquement la topologie de dépendances entre les équipements IoT dans des maison connectées simulées et réelles telle que l'appartement intelligente DOMUS[2].

2.2 Un système multi-agent sémantique pour la correction automatique et collaborative des pannes en cascade

Pour aller plus loin dans la gestion des pannes en cascade, nous avons proposé une deuxième contribution permettant aux acteurs DM isolés de gérer les pannes en cascade de manière automatique et collaborative, en s'appuyant sur notre première contribution développée pour l'inférence des dépendances IoT. Cette solution consiste en un système multi-agent coopératif, qui fait référence à un réseau d'agents logiciels opérant indépendamment tout en étant faiblement connectés pour résoudre des problèmes complexes qui dépassent les capacités ou les connaissances individuelles de chaque agent. Plus précisément, nous nous appuyons sur *cOllaborative caScading fAilure Management Agent* (OSAMA), un agent sémantique à intégrer dans les plateformes DM existantes afin de les aider à comprendre, collaborer et prendre des décisions efficaces concernant la gestion des pannes en cascade (voir Figure B.2).

OSAMA exploite un ensemble de standard du Web sémantique, telles que les ontologies, afin de simplifier l'échange d'informations sur les pannes et d'améliorer l'interopérabilité entre les plateformes DM isolées. Il tire parti de la technologie de jumeau numérique, modélisant les relations de dépendance dynamique entre les équipements IoT pour l'identification de la cause des pannes. A la détection d'une panne, les agents OSAMA lancent un protocole collaboratif qui leur permet d'identifier automatiquement les causes des pannes et de corriger automatiquement les équipements en panne. Ils adoptent un modèle *Belief Desire Intention* (BDI) pour gérer les pannes en cascade et collaborent selon un *protocole collaboratif* pour corriger les pannes en cascade qui se propagent sur des équipements gérés par des acteurs DM différent.

Dans leur environnement partagé, les agents OSAMA sont fournis par quatre (04) *Artéfacts* encapsulant des services externes qu'ils peuvent explorer à l'exécution pour faciliter la gestion des pannes en cascade: 1) *Artéfact de surveillance* : permet de surveiller les équipements IoT et de

[2] https://www.liglab.fr/fr/recherche/plateformes/domus

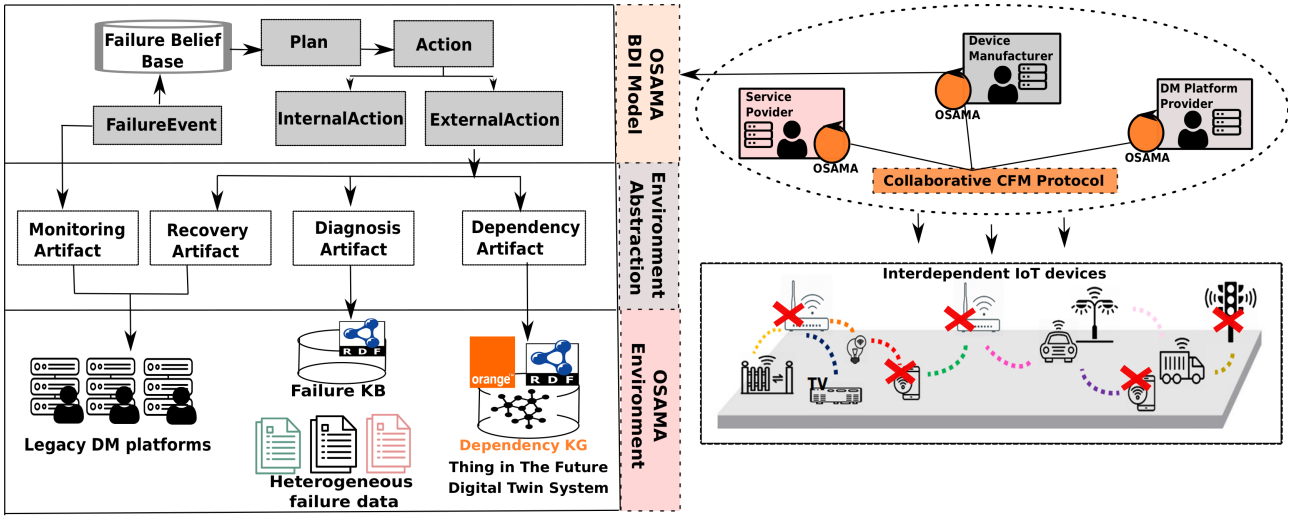


Figure B.2: Un système multi-agent sémantique pour la correction automatique et collaborative des pannes en cascade

détecter les pannes à l'aide des plateformes DM existantes ; 2) *Artéfact de diagnostic* : permet d'identifier le type de panne et ses actions correctives à l'aide d'une base de connaissances sur les panne structurée selon une ontologie appelée *Internet of Things Failure* (IoT-F) ; 3) *Artéfact de dépendance* : s'appuye sur notre solution de l'inférence de dépendance pour accéder automatiquement à une vue des relations de dépendance entre les équipements IoT pour faciliter l'identification de la cause des pannes en cascade ; 4) *Artéfact de correction* : permet d'exécuter des actions correctives sur les équipements IoT à l'aide des plateformes DM existantes. Ces agents *OSAMA* seront installés dans les infrastructures des acteurs DM du marché pour leurs aider dans la gestion automatique des pannes en cascade et l'amélioration de leurs services client.

Nous avons prouvé l'efficacité de cette solution en résolvant plusieurs scenario de panne en cascade dans des maisons connectés simulés gérés par plusieurs acteurs DM. Nous avons également démontré sa valeur ajoutée par rapport aux solutions DM existante en termes de temps de réparation des pannes et de consommation de ressources dans les systèmes IoT.

2.3 La démonstration "Collaborative LAN troubleshooting"

Pour valoriser nos travaux au sein de l'entreprise Orange, nous avons proposé la démonstration pratique *Collaborative LAN troubleshooting* qui illustre l'utilité de nos travaux dans l'amélioration des services client d'Orange. Cette démonstration illustre comment un agent de service client Orange peut utiliser nos solutions pour le diagnostic et la résolution des pannes en cascade à travers un outil de supervision Web (voir Figure B.3). Cet outil permet la supervision des équipements IoT, le calcule automatique des dépendances entre les équipements IoT, ainsi que la correction automatique des pannes en cascade à l'aide des agents *OSAMA*. Ce qui a permet d'illustrer les avantages tangibles et pratiques des solutions proposées pour faire progresser et optimiser les services client au sein de l'entreprise Orange.

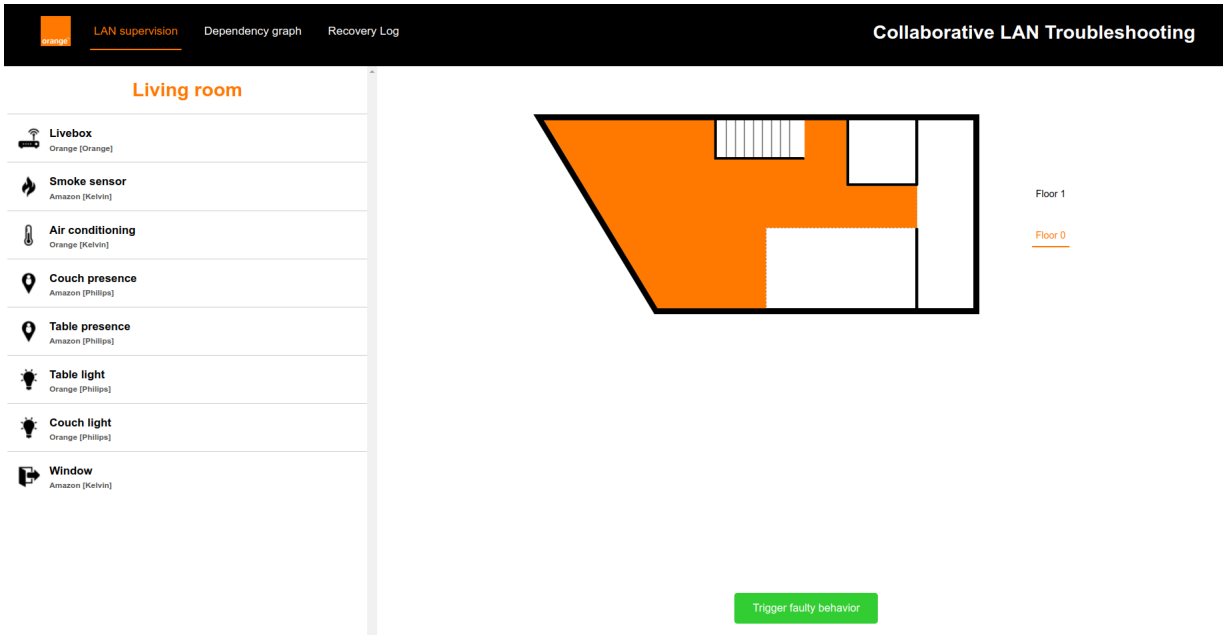


Figure B.3: Interface Web de supervision.

3 Conclusion

3.1 Synthèse

Cette thèse a proposé une approche automatique et collaborative pour la gestion des pannes en cascade des équipements IoT gérés par des acteurs différent. L'approche vise à aider les acteurs DM du marché à réduire les coûts de service client et le temps de réparation des pannes, pour une meilleure qualité d'expérience client.

Les travaux ont débuté par une revue approfondie de la littérature, identifiant les lacunes de recherche et explorant des technologies avancées telles que le Web Sémantique, les systèmes multi-agents, et les jumeaux numériques pour la prise de décision autonome. Ensuite, un nouveau outil d'inférence de dépendances a été introduit, exploitant une ontologie pour fournir une représentation unifiée des dépendances IoT. La validation du cet outil a impliqué des simulations et des scénarios réels, et son intégration avec la plateforme de jumeau numérique d'Orange Thing in a facilité une visualisation tangible du jumeau numérique pour les acteurs DM.

Ensuite, le système multi-agent OSAMA a été proposé pour permettre une gestion collaborative des pannes en cascade, démontrant son efficacité dans la résolution des panne en cascade au sein de scénarios de maison intelligente. Les solutions ont également été présentées dans la démonstration "Collaborative LAN Troubleshooting", illustrant les avantages pratiques pour les services de support client au sein de l'entreprise Orange.

En résumé, les contributions de cette thèse couvrent des bases théoriques, des outils innovants et des solutions pratiques et validées, abordant collectivement les complexités de la gestion des pannes en cascade dans les écosystèmes IoT gérés par divers acteurs DM.

3.2 Perspectives

Notre solution est à l'étape de la preuve de concept, mais nous avons plusieurs plans pour favoriser son adoption à grande échelle.

À court terme, plusieurs perspectives sont proposées. Tout d'abord, pour améliorer l'outil d'inférence des dépendances, la possibilité de doter les agents OSAMA de capacités proactives pour déployer des jumeaux numériques distribués est envisagée. De plus, l'introduction de plus d'artefacts intègrant les outils d'Orange, tels qu'un artefact de détection d'anomalies basé sur l'apprentissage automatique, est suggérée pour élargir les capacités des agents OSAMA. Une étude sur la résilience du protocole collaboratif de gestion des pannes en cascade est proposée pour assurer la cohérence globale de la solution. Des tests de scénarios réalistes à l'aide de plateformes DM de marché telles que *Amazon Web Service* et *LiveObjects* sont préconisés pour valider davantage l'efficacité des solutions proposées.

À moyen terme, les perspectives incluent la gestion de l'incertitude par l'IA neuro-symbolique, l'optimisation du protocole collaboratif de gestion des pannes en cascade en utilisant des capacités d'apprentissage, et l'exploration d'approches automatiques pour la construction des bases de connaissances sur les pannes.

À long terme, l'accent est mis sur la tolérance aux pannes en cascade, la prédiction et la prévention. Des techniques telles que le checkpointing, le remplacement d'équipements et la reconfiguration sont suggérées pour maintenir les services IoT disponibles pendant les pannes et renforcer la résilience globale du système. L'exploration d'une gestion standardisée des pannes IoT est proposée. Cela pourrait être réalisé en étudiant la position de nos solutions par rapport aux standards et initiatives actuelles, telles que *Matter* et *Prpl*. En outre, soumettre nos solutions en tant que projet de standard aux organisations de standardisation auxquelles Orange participe, telles que l'Institut européen des normes de télécommunications (ETSI) ou la Connectivity Standards Alliance (CSA), est envisagé. Un autre plan potentiel de standardisation serait de soumettre les ontologies IoT-D et IoT-F en tant qu'extension de l'ontologie normalisée *SAREF* modélisant le domaine de la gestion des pannes IoT.

Enfin, d'autres perspectives à long terme incluent l'exploration de nouveaux cas d'utilisation de DM collaboratif au-delà de la gestion des pannes en cascade, tels que la coordination des opérations DM pour éviter les pannes DM. Cela pourrait être réalisé en adoptant notre architecture avec des artefacts et protocoles collaboratifs spécifiques.

Bibliographie

[Aïssaoui, 2020] François Aïssaoui, Samuel Berlemont, Marc Douet, and Emna Mezghani. "A Semantic Model Toward Smart IoT Device Management". *Web, Artificial Intelligence and Network Applications*. Ed. by Leonard Barolli, Flora Amato, Francesco Moscato, Tomoya Enokido, and Makoto Takizawa. Cham: Springer International Publishing, 2020, pp. 640–650 (cit. on pp. 2, 20, 25, 33).

[Al-Ali, 2020] Abdul-Rahman Al-Ali, Ragini Gupta, Tasneem Zaman Batool, Taha Landolsi, Fadi Aloul, and Ahmad Al Nabulsi. "Digital twin conceptual model within the context of internet of things". *Future Internet* 12.10 (2020), p. 163 (cit. on p. 37).

[Ali, 2018] Nazakat Ali and Jang-Eui Hong. "Failure Detection and Prevention for Cyber-Physical Systems Using Ontology-Based Knowledge Base". *Computers* 7.4 (2018) (cit. on p. 68).

[Alsabilah, 2021] Nasser Alsabilah and Danda B. Rawat. "Anomaly Detection in Smart Home Networks Using Kalman Filter". *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2021, pp. 1–6 (cit. on pp. 60, 65).

[Alvisi, 1998] Lorenzo Alvisi and Keith Marzullo. "Message logging: Pessimistic, optimistic, causal, and optimal". *IEEE Transactions on Software Engineering* 24.2 (1998), pp. 149–159 (cit. on p. 54).

[Andy Stanford-Clark, 2019] Martin Harris Andy Stanford-Clark Erwin Frank-Schultz. *What are digital twins?* `https://developer.ibm.com/articles/what-are-digital-twins/`. 2019 (cit. on p. 34).

[Antakli, 2023] André Antakli, Akbar Kazimov, Daniel Spieldenner, Gloria Elena Jaramillo Rojas, Ingo Zinnikus, and Matthias Klusch. "AJAN: An Engineering Framework for Semantic Web-Enabled Agents and Multi-Agent Systems". *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*. Ed. by Philippe Mathieu, Frank Dignum, Paulo Novais, and Fernando De la Prieta. Cham: Springer Nature Switzerland, 2023, pp. 15–27 (cit. on p. 45).

[Arel, 2010] Itamar Arel, Cong Liu, Tom Urbanik, and Airton G Kohls. "Reinforcement learning-based multi-agent system for network traffic signal control". *IET Intelligent Transport Systems* 4.2 (2010), pp. 128–135 (cit. on p. 43).

[Armando, 2019] Ngombo Armando, Jose Fernandes, Soraya Sinche, Duarte Raposo, Jorge Sa Silva, and Fernando Boavida. "A Unified Solution for IoT Device Management". Vol. 2019-November. IEEE Computer Society, 2019 (cit. on p. 24).

[Arnaldo Perez, 2019] Arnaldo Perez. *Leveraging the Beliefs-Desires-Intentions Agent Architecture*. 2019 (cit. on p. 40).

[Avsystem, 2023] Avsystem. "User Service Protocol". *Avsystem* (2023) (cit. on p. 22).

[Ayeb, 2020a] Neil Ayeb. "Administration autonomique et décentralisée de flottes d'équipements de l'Internet des Objets". PhD thesis. Université Grenoble Alpes [2020-....], 2020 (cit. on p. 58).

[Ayeb, 2020b] Neil Ayeb, Eric Rutten, Sebastien Bolle, Thierry Coupaye, and Marc Douet. "Coordinated autonomic loops for target identification, load and error-aware Device Management for the IoT". *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. 2020, pp. 491–500 (cit. on pp. 25, 58).

[Babalola, 2016] Adeniyi A Babalola, Rabie Belkacemi, and Sina Zarrabian. "Real-time cascading failures prevention for multiple contingencies in smart grids through a multi-agent system". *IEEE Transactions on Smart Grid* 9.1 (2016), pp. 373–385 (cit. on p. 43).

[Babalola, 2014] Adeniyi Abdulrasheed Babalola. "Implementation of multi-agent system algorithms for distributed restoration and cascading failure blackout prevention in a smart grid system". PhD thesis. Tennessee Technological University, 2014 (cit. on p. 43).

[Bala, 2020] Mohammad Irfan Bala and Mohammad Ahsan Chishti. "Offloading in Cloud and Fog Hybrid Infrastructure Using iFogSim". *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. 2020, pp. 421–426 (cit. on p. 137).

[Bansal, 2011] Sanjay Bansal, Sanjeev Sharma, and Ishita Trivedi. "A Detailed Review of Fault-Tolerance Techniques in Distributed System." *International Journal on Internet & Distributed Computing Systems* 1.1 (2011) (cit. on p. 54).

[Bella, 2022] Giampaolo Bella, Domenico Cantone, M Nicolosi-Asmundo, and Daniele Francesco Santamaria. "The Ontology for Agents, Systems and Integration of Services: recent advancements of OASIS". *Proceedings of WOA*. 2022, pp. 1–2 (cit. on p. 45).

[Benazzouz, 2014] Yazid Benazzouz, Oum-El-keir Aktouf, and Ioannis Parissis. "A Fault Fuzzy-ontology for Large Scale Fault-tolerant Wireless Sensor Networks". *Procedia Computer Science* 35 (2014), pp. 203–212 (cit. on p. 68).

[Benbernou, 2021] Salima Benbernou, Xin Huang, and Mourad Ouziri. "Semantic-Based and Entity-Resolution Fusion to Enhance Quality of Big RDF Data". *IEEE Transactions on Big Data* 7.2 (2021), pp. 436–450 (cit. on pp. 79, 92).

[Berners-Lee, 1998] Tim Berners-Lee et al. *Semantic web road map*. 1998 (cit. on pp. 28, 29).

[Berners-Lee, 2001] Tim Berners-Lee, James Hendler, and Ora Lassila. "The semantic web". *Scientific american* 284.5 (2001), pp. 34–43 (cit. on p. 28).

[Boissier, 2020] Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press, 2020 (cit. on pp. 38, 42, 43, 104, 108).

[Boissier, 2013] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. "Multi-agent oriented programming with JaCaMo". *Science of Computer Programming* 78.6 (2013), pp. 747–761 (cit. on pp. 39, 42).

[Boje, 2020] Calin Boje, Annie Guerriero, Sylvain Kubicki, and Yacine Rezgui. "Towards a semantic Construction Digital Twin: Directions for future research". *Automation in Construction* 114 (2020), p. 103179 (cit. on p. 44).

[Bolle, 2019] Sébastien Bolle, Marc Douet, Samuel Berlemont, Emna Mezghani, and François Aïssaoui. *Towards a Unified IoT Device Management Federative Platform - Presentation at ETSI IoT Week 2019*. https://www.researchgate.net/publication/337160142_Towards_a_Unified_IoT_Device_Management_Federative_Platform_-_Presentation_at_ETSI_IoT_Week_2019. 2019 (cit. on pp. 25, 33).

[Bordini, 2007] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007 (cit. on p. 40).

[Borhani, 2022] Alireza Borhani and Hamid R. Zarandi. "ThingsDND: IoT Device Failure Detection and Diagnosis for Multi-User Smart Homes". IEEE, 2022, pp. 113–116 (cit. on pp. 60, 65).

[Bouchenak, 2005] S. Bouchenak, F. Boyer, D. Hagimont, S. Krakowiak, A. Mos, N. de Palma, et al. "Architecture-based autonomous repair management: an application to J2EE clusters". *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. 2005, pp. 13–24 (cit. on p. 55).

[Bratman, 1987] Michael Bratman. *Intention, Plans, and Practical Reason*. Cambridge: Cambridge, MA: Harvard University Press, 1987 (cit. on pp. 40, 99).

[Budhiraja, 1993] Navin Budhiraja, Keith Marzullo, Fred B Schneider, and Sam Toueg. "The primary-backup approach". *Distributed systems* 2 (1993), pp. 199–216 (cit. on p. 54).

[Castaldi, 2003] Marco Castaldi, Antonio Carzaniga, Paola Inverardi, and Alexander L. Wolf. "A Lightweight Infrastructure for Reconfiguring Applications". *Software Configuration Management*. Ed. by Bernhard Westfechtel and André van der Hoek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 231–244 (cit. on p. 55).

[Celik, 2019] Z. Berkay Celik, Gang Tan, and Patrick Mcdaniel. "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT". *Proceedings 2019 Network and Distributed System Security Symposium* (2019) (cit. on pp. 15, 16).

[Chakraborty, 2018] Tusher Chakraborty, Akshay Uttama Nambi, Ranveer Chandra, Rahul Sharma, Manohar Swaminathan, Zerina Kapetanovic, et al. "Fall-Curve: A Novel Primitive for IoT Fault Detection and Isolation". *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. SenSys '18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 95–107 (cit. on pp. 57, 103).

[Charpenay, 2020] Victor Charpenay and Sebastian Käbisch. “On Modeling the Physical World as a Collection of Things: The W3C Thing Description Ontology”. *The Semantic Web* 12123 (2020), pp. 599–615 (cit. on p. 31).

[Charpenay, 2022] Victor Charpenay, Antoine Zimmermann, Maxime Lefrançois, and Olivier Boissier. “Hypermedea: A Framework for Web (of Things) Agents”. *Companion Proceedings of the Web Conference 2022*. WWW ’22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 176–179 (cit. on p. 45).

[Chatterjee, 2022] Ayan Chatterjee and Bestoun S. Ahmed. “IoT anomaly detection methods and applications: A survey”. *Internet of Things* 19 (2022), p. 100568 (cit. on p. 59).

[Chen, 2015] Rong Chen, Zude Zhou, Quan Liu, Duc Truong Pham, Yuanyuan Zhao, Junwei Yan, et al. “Knowledge modeling of fault diagnosis for rotating machinery based on ontology”. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. 2015, pp. 1050–1055 (cit. on p. 67).

[Chen, 2017] Yingyi Chen, Zhumi Zhen, Huihui Yu, and Jing Xu. “Application of Fault Tree Analysis and Fuzzy Neural Networks to Fault Diagnosis in the Internet of Things (IoT) for Aquaculture”. *Sensors* 17.1 (2017) (cit. on pp. 60, 65).

[Chereque, 1992] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron. “Active replication in Delta-4”. *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*. 1992, pp. 28–37 (cit. on p. 54).

[Chernyshev, 2017] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. “Internet of things (iot): Research, simulators, and testbeds”. *IEEE Internet of Things Journal* 5.3 (2017), pp. 1637–1647 (cit. on p. 136).

[Chi, 2022a] Yuanfang Chi, Yanjie Dong, Z. Jane Wang, F. Richard Yu, and Victor C. M. Leung. “Knowledge-Based Fault Diagnosis in Industrial Internet of Things: A Survey”. *IEEE Internet of Things Journal* 9.15 (2022), pp. 12886–12900 (cit. on p. 60).

[Chi, 2022b] Yuanfang Chi, Z. Jane Wang, and Victor C. M. Leung. “Distributed Knowledge Inference Framework for Intelligent Fault Diagnosis in IIoT Systems”. *IEEE Transactions on Network Science and Engineering* 9.5 (2022), pp. 3152–3165 (cit. on p. 61).

[Choi, 2018] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, et al. “Detecting and identifying faulty IoT devices in smart home with context extraction”. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 610–621 (cit. on p. 57).

[Christophides, 2020] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. “An overview of end-to-end entity resolution for big data”. *ACM Computing Surveys (CSUR)* 53.6 (2020), pp. 1–42 (cit. on p. 79).

[Cimino, 2019] Chiara Cimino, Elisa Negri, and Luca Fumagalli. “Review of digital twin applications in manufacturing”. *Computers in Industry* 113 (2019), p. 103130 (cit. on p. 36).

[Ciortea, 2019] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. “A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web”. *AAMAS 2019 - 18th International Conference on Autonomous Agents and Multiagent Systems*. Montréal, Canada, 2019, p. 5 (cit. on p. 45).

[Ciortea, 2018] Andrei Ciortea, Simon Mayer, and Florian Michahelles. “Repurposing Manufacturing Lines on the Fly with Multi-Agent Systems for the Web of Things”. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’18. Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 813–822 (cit. on p. 45).

[Clarke, 1997] Edmund M. Clarke. “Model checking”. *Foundations of Software Technology and Theoretical Computer Science*. Ed. by S. Ramesh and G. Sivakumar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 54–56 (cit. on p. 141).

[Compton, 2012] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, et al. “The SSN ontology of the W3C semantic sensor network incubator group”. *Journal of Web Semantics* 17 (2012), pp. 25–32 (cit. on p. 66).

[Corno, 2017] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. “A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT”. *Computer* 50.11 (2017), pp. 18–24 (cit. on p. 67).

[Cristian, 1991] Flavin Cristian. “Understanding Fault-Tolerant Distributed Systems”. *Commun. ACM* 34.2 (1991), pp. 56–78 (cit. on p. 57).

[Croatti, 2020] Angelo Croatti, Matteo Gabellini, Sara Montagna, and Alessandro Ricci. “On the integration of agents and digital twins in healthcare”. *Journal of Medical Systems* 44 (2020), pp. 1–8 (cit. on p. 44).

[DAngelo, 2016] Gabriele D'Angelo, Stefano Ferretti, and Vittorio Ghini. "Simulation of the Internet of Things". *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2016, pp. 1–8 (cit. on p. 136).

[Daniele, 2015] Laura Daniele, Frank den Hartog, and Jasper Roes. "Created in close interaction with the industry: the smart appliances reference (SAREF) ontology". *Formal Ontologies Meet Industry: 7th International Workshop, FOMI 2015, Berlin, Germany, August 5, 2015, Proceedings 7*. Springer. 2015, pp. 100–112 (cit. on p. 66).

[Dastani, 2004] Mehdi Dastani, M Birna van Riemsdijk, Frank Dignum, and John-Jules Ch Meyer. "A programming language for cognitive agents goal directed 3APL". *Programming Multi-Agent Systems: First International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited papers 1*. Springer. 2004, pp. 111–130 (cit. on p. 40).

[Datta, 2015] Soumya Kanti Datta and Christian Bonnet. "A lightweight framework for efficient M2M device management in oneM2M architecture". Institute of Electrical and Electronics Engineers Inc., 2015 (cit. on p. 24).

[Derrien, 2019] Sylvie Derrien, Pierre Meye, and Phillippe Raıpin. "Thing in, a research platform for the Web of Things". *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2019, pp. 431–432 (cit. on pp. 35, 36).

[Dignum, 2004] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. "Omni: Introducing social structure, norms and ontologies into agent organizations". *International Workshop on Programming Multi-Agent Systems*. Springer. 2004, pp. 181–198 (cit. on p. 41).

[Dimou, 2014] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". *Proceedings of the 7th Workshop on Linked Data on the Web*. 2014 (cit. on p. 130).

[DOMUS, 2023] DOMUS. *Living Lab DOMUS (LIG)*. https://maci.univ-grenoble-alpes.fr/nos-espaces/living-lab-domus-lig. 2023 (cit. on p. 90).

[Dong, 2023] Yi Dong, Zhongguo Li, Xingyu Zhao, Zhengtao Ding, and Xiaowei Huang. "Decentralised and cooperative control of multi-robot systems through distributed optimisation". *arXiv preprint arXiv:2302.01728* (2023) (cit. on p. 43).

[Dorri, 2018] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. "Multi-Agent Systems: A Survey". *IEEE Access* 6 (2018), pp. 28573–28593 (cit. on pp. 37, 38, 43).

[Emmanouilidis, 2020] C Emmanouilidis, M. Gregori, and A. Al-Shdifat. "Context Ontology Development for Connected Maintenance Services". *IFAC-PapersOnLine* 53.2 (2020), pp. 10923–10928 (cit. on pp. 62, 67, 103, 116).

[Erol, 2020] Tolga Erol, Arif Furkan Mendi, and Dilara Doğan. "The Digital Twin Revolution in Healthcare". *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2020, pp. 1–7 (cit. on p. 37).

[Far, 2022] Saeed Banaeian Far and Azadeh Imani Rad. "Applying digital twins in metaverse: User interface, security and privacy challenges". *Journal of Metaverse* 2.1 (2022), pp. 8–15 (cit. on p. 33).

[Fensel, 2020] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, et al. "Introduction: What Is a Knowledge Graph?" *Knowledge Graphs: Methodology, Tools and Selected Use Cases*. 2020, pp. 1–10 (cit. on p. 76).

[Ferreira, 2017] Jose Ferreira, Joao Nuno Soares, Ricardo Jardim-Goncalves, and Carlos Agostinho. "Management of IoT Devices in a Physical Network". Institute of Electrical and Electronics Engineers Inc., 2017, pp. 485–492 (cit. on p. 24).

[Ford, 2020] David N. Ford and Charles M. Wolf. "Smart Cities with Digital Twin Systems for Disaster Management". *Journal of Management in Engineering* 36.4 (2020), p. 04020027. eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29ME.1943-5479.0000779 (cit. on p. 37).

[Frank, 2021] Matthias Frank. "Knowledge-Driven Harmonization of Sensor Observations: Exploiting Linked Open Data for IoT Data Streams" (2021), pp. 0–236 (cit. on p. 93).

[Fu, 2021] Xiuwen Fu and Yongsheng Yang. "Modeling and analyzing cascading failures for Internet of Things". *Information Sciences* 545 (2021), pp. 753–770 (cit. on p. 61).

[Gandon, 2018] Fabien Gandon. "A survey of the first 20 years of research on semantic Web and linked data". *Revue des Sciences et Technologies de l'Information-Série ISI: Ingénierie des Systèmes d'information* (2018) (cit. on pp. 28, 30).

[Ganzha, 2017] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, and Katarzyna Wasielewska. "Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective". *Journal of Network and Computer Applications* 81 (2017), pp. 111–124 (cit. on p. 19).

[Gao, 2015a] Zhiwei Gao, Carlo Cecati, and Steven X. Ding. "A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part I: Fault Diagnosis With Model-Based and Signal-Based Approaches". *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3757–3767 (cit. on pp. 60, 65).

[Gao, 2015b] Zhiwei Gao, Carlo Cecati, and Steven X. Ding. "A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part II: Fault Diagnosis With Knowledge-Based and Hybrid/Active Approaches". *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3768–3774 (cit. on pp. 60, 65).

[Garcez, 2023] Artur d'Avila Garcez and Luis C Lamb. "Neurosymbolic AI: The 3 rd wave". *Artificial Intelligence Review* (2023), pp. 1–20 (cit. on p. 131).

[García-Castro, 2023] Raúl García-Castro, Maxime Lefrançois, María Poveda-Villalón, and Laura Daniele. "The ETSI SAREF Ontology for Smart Applications: A Long Path of Development and Evolution". *Energy Smart Appliances: Applications, Methodologies, and Challenges*. 2023, pp. 183–215 (cit. on p. 66).

[Gentile, 2019] Anna Lisa Gentile, Daniel Gruhl, Petar Ristoski, and Steve Welch. "Personalized knowledge graphs for the pharmaceutical domain". *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18*. Springer. 2019, pp. 400–417 (cit. on p. 32).

[Gheysari, 2022] Mohmmad Gheysari and Mahsa Seyed Sadegh Tehrani. "The Role of Multi-Agent Systems in IoT". *Multi Agent Systems: Technologies and Applications towards Human-Centered*. Ed. by Shibakali Gupta, Indradip Banerjee, and Siddhartha Bhattacharyya. Singapore: Springer Nature Singapore, 2022, pp. 87–114 (cit. on p. 44).

[Giannoni, 2018] Federico Giannoni, Marco Mancini, and Federico Marinelli. "Anomaly detection models for IoT time series data". *arXiv preprint arXiv:1812.00890* (2018) (cit. on p. 59).

[Giantomassi, 2014a] Andrea Giantomassi, Francesco Ferracuti, Sabrina Iarlori, Sauro Longhi, Alessandro Fonti, and Gabriele Comodi. "Kernel canonical variate analysis based management system for monitoring and diagnosing smart homes". *2014 International Joint Conference on Neural Networks (IJCNN)*. 2014, pp. 1432–1439 (cit. on pp. 60, 65).

[Giantomassi, 2014b] Andrea Giantomassi, Francesco Ferracuti, Sabrina Iarlori, Gloria Puglia, Alessandro Fonti, Gabriele Comodi, et al. "Smart home heating system malfunction and bad behavior diagnosis by Multi-Scale PCA under indoor temperature feedback control". *22nd Mediterranean Conference on Control and Automation*. 2014, pp. 876–881 (cit. on pp. 60, 65).

[Glaessgen, 2012] Edward Glaessgen and David Stargel. "The digital twin paradigm for future NASA and US Air Force vehicles". *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. 2012, p. 1818 (cit. on p. 33).

[GoogleDev, 2023] GoogleDev. *Matter The Device Data Model*. 2023 (cit. on p. 18).

[Grieves, 2014] Michael Grieves. "Digital twin: manufacturing excellence through virtual factory replication". *White paper* 1.2014 (2014), pp. 1–7 (cit. on p. 33).

[Guerraoui, 1996] Rachid Guerraoui and André Schiper. "Fault-tolerance by replication in distributed systems". *Reliable Software Technologies — Ada-Europe '96*. Ed. by Alfred Strohmeier. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 38–57 (cit. on p. 54).

[Guittoum, 2023a] Amal Guittoum, François Aïssaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Solving the IoT Cascading Failure Dilemma Using a Semantic Multi-agent System". *The Semantic Web – ISWC 2023*. Ed. by Terry R. Payne, Valentina Presutti, Guilin Qi, María Poveda-Villalón, Giorgos Stoilos, Laura Hollink, et al. Cham: Springer Nature Switzerland, 2023, pp. 325–344 (cit. on p. 97).

[Guittoum, 2023b] Amal Guittoum, Francois Aıssaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Inferring Threatening IoT Dependencies Using Semantic Digital Twins Toward Collaborative IoT Device Management". *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC '23. Tallinn, Estonia: Association for Computing Machinery, 2023, pp. 1732–1741 (cit. on p. 71).

[Guittoum, 2023c] Amal Guittoum, François Aıssaoui, Sébastien Bolle, Fabienne Boyer, and Noel De Palma. "Leveraging Semantic Technologies for Collaborative Inference of Threatening IoT Dependencies". *SIGAPP Appl. Comput. Rev.* 23.3 (2023), pp. 32–48 (cit. on p. 71).

[Gupta, 2021] Deepti Gupta, Olumide Kayode, Smriti Bhatt, Maanak Gupta, and Ali Saman Tosun. "Hierarchical Federated Learning based Anomaly Detection using Digital Twins for Smart Healthcare". *2021 IEEE 7th International Conference on Collaboration and Internet Computing (CIC)*. 2021, pp. 16–25 (cit. on p. 37).

[Gupta, 2016] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. *iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments*. 2016. arXiv: 1606.02007 [cs.DC] (cit. on pp. 107, 114, 135, 138).

[Gupta, 2017] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments". *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2509 (cit. on p. 114).

[Hannoun, 2000] Mahdi Hannoun, Olivier Boissier, Jaime S Sichman, and Claudette Sayettat. "MOISE: An organizational model for multi-agent systems". *Ibero-American Conference on Artificial Intelligence*. Springer. 2000, pp. 156–165 (cit. on p. 41).

[Hayashibara, 2002] N. Hayashibara, A. Cherif, and T. Katayama. "Failure detectors for large-scale distributed systems". *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.* 2002, pp. 404–409 (cit. on p. 53).

[Hitzler, 2021] Pascal Hitzler. "A review of the semantic web field". *Communications of the ACM* 64.2 (2021), pp. 76–83 (cit. on pp. 28, 31).

[Hitzler, 2019] Pascal Hitzler, Armin Haller, Krzysztof Janowicz, Simon J.D. Cox, Maxime Lefrançois, Kerry Taylor, et al. "The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation". *Semant. Web* 10.1 (2019), pp. 9–32 (cit. on p. 66).

[Howden, 2001] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. "JACK intelligent agents-summary of an agent infrastructure". *5th International conference on autonomous agents*. Vol. 6. 2001 (cit. on p. 40).

[Huang, 2016] Jiwei Huang, Guo Chen, and Bo Cheng. "A Stochastic Approach of Dependency Evaluation for IoT Devices". *Chinese Journal of Electronics* 25 (2016), pp. 209–214 (cit. on pp. 48–50, 52).

[HuaweiTechnologies, 2018] Huawei.Technologies. "Unlocking the Potential of the Internet of Things". *Global Industry Vision* (2018) (cit. on p. 15).

[Hubner, 2007] Jomi F Hubner, Jaime S Sichman, and Olivier Boissier. "Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels". *International Journal of Agent-Oriented Software Engineering* 1.3-4 (2007), pp. 370–395 (cit. on p. 41).

[Hübner, 2005] Jomi Fred Hübner, Jaime Simao Sichman, and Olivier Boissier. ": a middleware for developing organised multi-agent systems". *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2005, pp. 64–77 (cit. on p. 41).

[Huhns, 2001] Michael N. Huhns. "Interaction-Oriented Programming". *Agent-Oriented Software Engineering*. Ed. by Paolo Ciancarini and Michael J. Wooldridge. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 29–44 (cit. on p. 42).

[Ihsan, 2023] Ahmad Zainul Ihsan, Said Fathalla, and Stefan Sandfeld. "DISO: A Domain Ontology for Modeling Dislocations in Crystalline Materials". *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 2023, pp. 1746–1753 (cit. on pp. 91, 94).

[Ikram, 2022] Muhammad Ikram, Salman Ahmed, and Safdar Nawaz Khan. "Cascade Failure Management in Distributed Smart Grid Using Multi-Agent Control". *2022 17th International Conference on Emerging Technologies (ICET)*. 2022, Proc-31-Proc–36 (cit. on p. 43).

[ITU-T, 2020] ITU-T. *Recommendation Y.4459: Digital entity architecture framework for Internet of things interoperability*. 2020 (cit. on p. 130).

[ITU-T Y2060, 2012] ITU-T Y.2060. *Overview of the Internet of things*. 2012 (cit. on p. 15).

[ITU-T Y4702, 2016] ITU-T Y.4702. *Y.4702 : Common requirements and capabilities of device management in the Internet of things*. 2016 (cit. on p. 19).

[Jaimez-González, 2021] Carlos R. Jaimez-González and Wulfrano A. Luna-Ramírez. *Towards a Multi-Agent System Architecture for Supply Chain Management*. 2021. arXiv: 2110.08125 [cs.MA] (cit. on p. 43).

[Jia, 2021] Yan Jia, Bin Yuan, Luyi Xing, Dongfang Zhao, Yifan Zhang, XiaoFeng Wang, et al. "Who's In Control? On Security Risks of Disjointed IoT Device Management Channels". *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 1289–1305 (cit. on pp. 2, 3, 6, 25, 26, 79).

[Jung, 2018] Tobias Jung, Payal Shah, and Michael Weyrich. "Dynamic Co-Simulation of Internet-of-Things-Components using a Multi-Agent-System". *Procedia CIRP* 72 (2018), pp. 874–879 (cit. on p. 44).

[Kapitanova, 2012] Krasimira Kapitanova, Enamul Hoque, John A. Stankovic, Kamin Whitehouse, and Sang H. Son. "Being SMART about Failures: Assessing Repairs in SMART Homes". *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp '12. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2012, pp. 51–60 (cit. on pp. 58, 65).

[Keith D Foote, 2022] Keith D. Foote. *A Brief History of the Internet of Things*. 2022 (cit. on p. 15).

[Khadir, 2022] Karima Khadir, Nawal Guermouche, Amal Guittoum, and Thierry Monteil. "A Genetic Algorithm-Based Approach for Fluctuating QoS Aware Selection of IoT Services". *IEEE Access* 10 (2022), pp. 17946–17965 (cit. on p. 33).

[Khadir, 2020] Karima Khadir, Nawal GUERMOUCHE, Thierry MONTEIL, and Amal GUITTOUM. "Towards avatar-based discovery for IoT services using social networking and clustering mechanisms". *2020 16th International Conference on Network and Service Management (CNSM)*. 2020, pp. 1–7 (cit. on p. 33).

[Kodeswaran, 2016] Palanivel Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. "Idea: A system for efficient failure management in smart IoT environments". Association for Computing Machinery, Inc, 2016, pp. 43–56 (cit. on pp. 58, 65).

[Laštovička, 2017] Martin Laštovička and Pavel Čeleda. "Situational Awareness: Detecting Critical Dependencies and Devices in a Network". *11th IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS)*. Ed. by Daphne Tuncer, Robert Koch, Rémi Badonnel, and Burkhard Stiller. Vol. LNCS-10356. Security of Networks and Services in an All-Connected World. Zurich, Switzerland: Springer International Publishing, 2017, pp. 173–178 (cit. on pp. 49, 52).

[Lazarova-Molnar, 2016] Sanja Lazarova-Molnar, Hamid Reza Shaker, Nader Mohamed, and Bo Norregaard Jorgensen. "Fault detection and diagnosis for smart buildings: State of the art, trends and challenges". *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*. 2016, pp. 1–7 (cit. on pp. 60, 65).

[Ledmi, 2018] Abdeldjalil Ledmi, Hakim Bendjenna, and Sofiane Mounine Hemam. "Fault Tolerance in Distributed Systems: A Survey". *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. 2018, pp. 1–5 (cit. on pp. 53, 55).

[Leitão, 2013] Luis Leitão and Pável Calado. "An Automatic Blocking Strategy for XML Duplicate Detection". *SIGAPP Appl. Comput. Rev.* 13.2 (2013), pp. 42–53 (cit. on p. 79).

[Li, 2020a] Bohan Li, Yi Liu, Anman Zhang, Wenhuan Wang, and Shuo Wan. "A Survey on Blocking Technology of Entity Resolution". *Journal of Computer Science and Technology* 35 (2020), pp. 769–793 (cit. on p. 79).

[Li, 2019] Jiaming Li, Ying Guo, Josh Wall, and Sam West. "Support vector machine based fault detection and diagnosis for HVAC systems". *International Journal of Intelligent Systems Technologies and Applications* 18 (2019), p. 204 (cit. on pp. 60, 65).

[Li, 2020b] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. "A survey on deep learning for named entity recognition". *IEEE Transactions on Knowledge and Data Engineering* 34.1 (2020), pp. 50–70 (cit. on p. 132).

[Liang, 2020] Chao Liang, Bharanidharan Shanmugam, Sami Azam, Asif Karim, Ashraful Islam, Mazdak Zamani, et al. "Intrusion Detection System for the Internet of Things Based on Blockchain and Multi-Agent Systems". *Electronics* 9.7 (2020) (cit. on p. 44).

[Lionel Tailhardat, 2022] Lionel Tailhardat, Yoan Chabot, and Raphaël Troncy. "NORIA-O: an Ontology for Anomaly Detection and Incident Management in ICT Systems". 2022 (cit. on p. 68).

[Lombardi, 2021] Marco Lombardi, Francesco Pascale, and Domenico Santaniello. "Internet of Things: A General Overview between Architectures, Protocols and Applications". *Information* 12.2 (2021) (cit. on p. 15).

[Mahela, 2022] Om Prakash Mahela, Mahdi Khosravy, Neeraj Gupta, Baseem Khan, Hassan Haes Alhelou, Rajendra Mahla, et al. "Comprehensive overview of multi-agent systems for controlling smart grids". *CSEE Journal of Power and Energy Systems* 8.1 (2022), pp. 115–131 (cit. on p. 43).

[Mahmud, 2022] Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. "iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments". *Journal of Systems and Software* 190 (2022), p. 111351 (cit. on pp. 114, 137).

[Maloney, 2019] Matthew Maloney, Elizabeth Reilly, Michael Siegel, and Gregory Falco. "Cyber Physical IoT Device Management Using a Lightweight Agent". IEEE, 2019, pp. 1009–1014 (cit. on p. 24).

[Manyika, 2015] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, et al. "Unlocking the Potential of the Internet of Things". *McKinsey Global Institute* 1 (2015) (cit. on p. 15).

[Mariani, 2023] Stefano Mariani, Pasquale Roseti, and Franco Zambonelli. "Multi-agent Learning of Causal Networks in the Internet of Things". *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*. Ed. by Philippe Mathieu, Frank Dignum, Paulo Novais, and Fernando De la Prieta. Cham: Springer Nature Switzerland, 2023, pp. 163–174 (cit. on pp. 50, 52).

[Mavromatis, 2020] Alex Mavromatis, Carlos Colman-Meixner, Aloizio P. Silva, Xenofon Vasilakos, Reza Nejabati, and Dimitra Simeonidou. "A Software-Defined IoT Device Management Framework for Edge and Cloud Computing". *IEEE Internet of Things Journal* 7 (3 2020), pp. 1718–1735 (cit. on p. 24).

[Mei, 2006] Jing Mei and Harold Boley. "Interpreting SWRL Rules in RDF Graphs". *Electronic Notes in Theoretical Computer Science* 151.2 (2006), pp. 53–69 (cit. on p. 93).

[Mezghani, 2020] Emna Mezghani, Samuel Berlemont, and Marc Douet. "Autonomic Coordination of IoT Device Management Platforms". *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 2020, pp. 30–35 (cit. on pp. 6, 25, 26, 50, 58, 70, 72).

[Minerva, 2020] Roberto Minerva, Gyu Myoung Lee, and Noël Crespi. "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models". *Proceedings of the IEEE* 108.10 (2020), pp. 1785–1824 (cit. on p. 44).

[Moder, 2020] Patrick Moder, Hans Ehm, and Eva Jofer. "A Holistic Digital Twin Based on Semantic Web Technologies to Accelerate Digitalization". *Digital Transformation in Semiconductor Manufacturing*. Ed. by Sophia Keil, Rainer Lasch, Fabian Lindner, and Jacob Lohmer. Cham: Springer International Publishing, 2020, pp. 3–13 (cit. on p. 44).

[Mohsin, 2016] Mujahid Mohsin, Zahid Anwar, Ghaith Husari, Ehab Al-Shaer, and Mohammad Ashiqur Rahman. "IoTSAT: A formal framework for security analysis of the internet of things (IoT)". *2016 IEEE Conference on Communications and Network Security (CNS)*. 2016, pp. 180–188 (cit. on pp. 49, 52).

[Mohsin, 2017] Mujahid Mohsin, Zahid Anwar, Farhat Zaman, and Ehab Al-Shaer. "IoTChecker: A data-driven framework for security analytics of Internet of Things configurations". *Computers & Security* 70 (2017), pp. 199–223 (cit. on pp. 49, 52, 67, 92, 94).

[Moualla, 2022] Ghada Moualla, Sebastien Bolle, Marc Douet, and Eric Rutten. "Self-adaptive Device Management for the IoT Using Constraint Solving". *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*. 2022, pp. 641–650 (cit. on p. 25).

[Moura, 2019] Ralf Luis De Moura, Tiago Monteiro Brasil, Luciana De Landa Ceotto, Alexandre Gonzalez, Luiz Paulo Barreto, and Ludmilla Bassini Werner. "Industrial internet of things: Device management architecture proposal". Institute of Electrical and Electronics Engineers Inc., 2019, pp. 1174–1178 (cit. on p. 24).

[Mugeni, 2023] John Bosco Mugeni and Toshiyuki Amagasa. "A Graph-Based Blocking Approach for Entity Matching Using Contrastively Learned Embeddings". *SIGAPP Appl. Comput. Rev.* 22.4 (2023), pp. 37–46 (cit. on p. 79).

[Naas, 2018] Mohammed Islam Naas, Jalil Boukhobza, Philippe Raipin Parvedy, and Laurent Lemarchand. "An Extension to iFogSim to Enable the Design of Data Placement Strategies". *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. 2018, pp. 1–8 (cit. on p. 114).

[Najari, 2021] Naji Najari, Samuel Berlemont, Grégoire Lefebvre, Stefan Duffner, and Christophe Garcia. "RADON: Robust Autoencoder for Unsupervised Anomaly Detection". *2021 14th International Conference on Security of Information and Networks (SIN)*. Vol. 1. 2021, pp. 1–8 (cit. on pp. 59, 65).

[Najari, 2022] Naji Najari, Samuel Berlemont, Grégoire Lefebvre, Stefan Duffner, and Christophe Garcia. "Robust Variational Autoencoders and Normalizing Flows for Unsupervised Network Anomaly Detection". *Advanced Information Networking and Applications*. Ed. by Leonard Barolli, Farookh Hussain, and Tomoya Enokido. Cham: Springer International Publishing, 2022, pp. 281–292 (cit. on pp. 59, 60).

[Najeh, 2019] Houda Najeh. "Diagnosis in building : new challenges". Theses. Université Grenoble Alpes ; École nationale d'ingénieurs de Gabès (Tunisie), 2019 (cit. on pp. 7, 27, 60, 65, 143).

[Nasar, 2021] Zara Nasar, Syed Waqar Jaffry, and Muhammad Kamran Malik. "Named entity recognition and relation extraction: State-of-the-art". *ACM Computing Surveys (CSUR)* 54.1 (2021), pp. 1–39 (cit. on p. 132).

[Nayyar, 2015] Anand Nayyar and Rajeshwar Singh. "A comprehensive review of simulation tools for wireless sensor networks (WSNs)". *Journal of Wireless Networking and Communications* 5.1 (2015), pp. 19–47 (cit. on p. 136).

[Nguyen, 2022] Luong Nguyen, Mariana Segovia, Wissam Mallouli, Edgardo Montes de Oca, and Ana R. Cavalli. "Digital Twin for IoT Environments: A Testing and Simulation Tool". *Quality of Information and Communications Technology*. Ed. by Antonio Vallecillo, Joost Visser, and Ricardo Pérez-Castillo. Cham: Springer International Publishing, 2022, pp. 205–219 (cit. on p. 37).

[Niati, 2020] Asmaa Niati, Cyrine Selma, Dalila Tamzalit, Hugo Bruneliere, Nasser Mebarki, and Olivier Cardin. "Towards a Digital Twin for Cyber-Physical Production Systems: A Multi-Paradigm Modeling Approach in the Postal Industry". *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '20. Virtual Event, Canada: Association for Computing Machinery, 2020 (cit. on p. 44).

[Nishiguchi, 2018] Yuki Nishiguchi, Ai Yano, Takeshi Ohtani, Ryuichi Matsukura, and Jun Kakuta. "IoT fault management platform with device virtualization". Vol. 2018-January. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 257–262 (cit. on pp. 57, 61, 65).

[Nita, 2014] Mihaela-Catalina Nita, Florin Pop, Mariana Mocanu, and Valentin Cristea. "FIM-SIM: Fault Injection Module for CloudSim Based on Statistical Distributions". *Journal of telecommunications and information technology* 4 (2014) (cit. on pp. 113, 114, 137, 139).

[Norris, 2020] Michael Norris, Z. Berkay Celik, Patrick Mcdaniel, Gang Tan, Prasanna Venkatesh, Shulin Zhao, et al. "IoTRepair: Systematically Addressing Device Faults in Commodity IoT". *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)* (2020), pp. 142–148 (cit. on pp. 57, 61, 65).

[Norris, 2022] Michael Norris, Z. Berkay Celik, Prasanna Venkatesh, Shulin Zhao, Patrick McDaniel, Anand Sivasubramaniam, et al. "IoTRepair: Flexible Fault Handling in Diverse IoT Deployments". *ACM Trans. Internet Things* 3.3 (2022) (cit. on pp. 19, 56–58, 61, 62, 65, 103, 114).

[Noura, 2019] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. "Interoperability in Internet of Things: Taxonomies and Open Challenges". *Mob. Netw. Appl.* 24.3 (2019), pp. 796–809 (cit. on p. 19).

[Noviello, 2023] Francesco Noviello, Munyque Mittelmann, Aniello Murano, and Silvia Stranieri. "Parking Problem with Multiple Gates". *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*. Ed. by Philippe Mathieu, Frank Dignum, Paulo Novais, and Fernando De la Prieta. Cham: Springer Nature Switzerland, 2023, pp. 213–224 (cit. on p. 43).

[Obeid, 2018] Charbel Obeid, Inaya Lahoud, Hicham El Khoury, and Pierre-Antoine Champin. "Ontology-Based Recommender System in Higher Education". *Companion Proceedings of the The Web Conference 2018*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1031–1034 (cit. on p. 32).

[Omicini, 2008] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. "Artifacts in the A&A meta-model for multi-agent systems". *Autonomous agents and multi-agent systems* 17 (2008), pp. 432–456 (cit. on p. 41).

[Opoku, 2021] De-Graft Joe Opoku, Srinath Perera, Robert Osei-Kyei, and Maria Rashidi. "Digital twin application in the construction industry: A literature review". *Journal of Building Engineering* 40 (2021), p. 102726 (cit. on p. 37).

[OrangeBuisness, 2023] OrangeBuisness. "Avec Orange l'internet des objets change le monde pour tout le monde". *OrangeBuisness* (2023) (cit. on p. 15).

[Ozeer, 2019] Umar Ibn Zaid Ozeer. "Autonomic resilience of distributed IoT applications in the Fog". Theses. Université Grenoble Alpes, 2019 (cit. on pp. 53–57, 61, 65, 103, 131, 135, 141).

[Pahl, 2019] Marc-Oliver Pahl and Stefan Liebald. "A Modular Distributed IoT Service Discovery". *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 448–454 (cit. on p. 33).

[Patel, 2019] ND Patel, BM Mehtre, and Rajeev Wankar. "Simulators, emulators, and test-beds for internet of things: A comparison". *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE. 2019, pp. 139–145 (cit. on p. 136).

[Patil, 1992] Ramesh Patil, Richard Fikes, Peter Patel-Schneider, Donald P McKay, Tim Finin, Thomas Gruber, et al. "The DARPA knowledge sharing effort: Progress report". *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*. 1992 (cit. on p. 104).

[Pérez, 2006] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL". *The Semantic Web - ISWC 2006*. Ed. by Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, et al. 2006, pp. 30–43 (cit. on p. 93).

[Perez Abreu, 2020] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. "A comparative analysis of simulators for the Cloud to Fog continuum". *Simulation Modelling Practice and Theory* 101 (2020), p. 102029 (cit. on p. 113).

[Perumal, 2016] Thinagaran Perumal, Soumya Kanti Datta, and Christian Bonnet. "IoT device management framework for smart home scenarios". Institute of Electrical and Electronics Engineers Inc., 2016, pp. 54–55 (cit. on p. 24).

[Pham, 2016] Cu Pham, Yuto Lim, and Yasuo Tan. “Management architecture for heterogeneous IoT devices in home network”. Institute of Electrical and Electronics Engineers Inc., 2016 (cit. on p. 24).

[Piro, 2016] Robert Piro, Yavor Nenov, Boris Motik, Ian Horrocks, Peter Hendler, Scott Kimberly, et al. “Semantic technologies for data analysis in health care”. *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*. Springer. 2016, pp. 400–417 (cit. on p. 31).

[Portela, 2023] Ariel L. Portela, Rafael A. Menezes, Wanderson L. Costa, Matheus M. Silveira, Luiz F. Bittecnourt, and Rafael Lopes Gomes. “Detection of IoT Devices and Network Anomalies based on Anonymized Network Traffic”. *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 2023, pp. 1–6 (cit. on pp. 59, 65).

[Pouvreau, 2023] Quentin Pouvreau, Jean-Pierre Georgé, Carole Bernon, and Sébastien Maignan. “Optimization of Complex Systems in Photonics by Multi-agent Robotic Control”. *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2023, pp. 272–283 (cit. on p. 43).

[Power, 2020] Alexander Power, Supervisor Dr, and Gerald Kotonya. *A Predictive Fault-Tolerance Framework for IoT Systems*. 2020 (cit. on p. 57).

[Pretel, 2022] Elena Pretel, Elena Navarro, Víctor López-Jaquero, Alejandro Moya, and Pascual González. “Multi-Agent Systems in Support of Digital Twins: A Survey”. *Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence*. Ed. by José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, and Hojjat Adeli. Cham: Springer International Publishing, 2022, pp. 524–533 (cit. on p. 44).

[Pynadath, 2003] David V Pynadath and Milind Tambe. “An automated teamwork infrastructure for heterogeneous software agents and humans”. *Autonomous Agents and Multi-Agent Systems* 7 (2003), pp. 71–100 (cit. on p. 41).

[Qorvo, 2021] Qorvo. *Matter Gets Everybody “Talking”*. 2021 (cit. on p. 17).

[Radoglou Grammatikis, 2019] Panagiotis I. Radoglou Grammatikis, Panagiotis G. Sarigiannidis, and Ioannis D. Moscholios. “Securing the Internet of Things: Challenges, threats and solutions”. *Internet of Things* 5 (2019), pp. 41–70 (cit. on p. 18).

[Ray, 2018] P.P. Ray. “A survey on Internet of Things architectures”. *Journal of King Saud University - Computer and Information Sciences* 30.3 (2018), pp. 291–319 (cit. on p. 16).

[Rhayem, 2020] Ahlem Rhayem, Mohamed Ben Ahmed Mhiri, and Faiez Gargouri. “Semantic Web Technologies for the Internet of Things: Systematic Literature Review”. *INTERNET OF THINGS* 11 (2020) (cit. on p. 33).

[Ricci, 2011] Alessandro Ricci, Michele Piunti, and Mirko Viroli. “Environment programming in multi-agent systems: an artifact-based perspective”. *Autonomous Agents and Multi-Agent Systems* 23 (2011), pp. 158–192 (cit. on pp. 40, 41).

[Ricci, 2009] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. “Environment programming in CArtAgO”. *Multi-agent programming: Languages, tools and applications* (2009), pp. 259–288 (cit. on p. 41).

[Rivas, 2022] Ariam Rivas, Diego Collarana, Maria Torrente, and Maria-Esther Vidal. “A neuro-symbolic system over knowledge graphs for link prediction”. *Semantic Web* Preprint (2022), pp. 1–25 (cit. on p. 131).

[Rocha, 2017] Jorge Rocha, Inês Boavida-Portugal, and Eduardo Gomes. “Introductory Chapter: Multi-Agent Systems”. *Multi-agent Systems*. Ed. by Jorge Rocha. Rijeka: IntechOpen, 2017. Chap. 1 (cit. on p. 38).

[Rojas, 2021a] Julián Andrés Rojas, Marina Aguado, Polymnia Vasilopoulou, Ivo Velitchkov, Dylan Van Assche, Pieter Colpaert, et al. “Leveraging Semantic Technologies for Digital Interoperability in the European Railway Domain”. *The Semantic Web – ISWC 2021*. 2021, pp. 648–664 (cit. on p. 130).

[Rojas, 2021b] Julián Andrés Rojas, Marina Aguado, Polymnia Vasilopoulou, Ivo Velitchkov, Dylan Van Assche, Pieter Colpaert, et al. “Leveraging semantic technologies for digital interoperability in the European railway domain”. *International Semantic Web Conference*. Springer. 2021, pp. 648–664 (cit. on p. 32).

[Rose, 2015] Karen Rose, Scott Eldridge, and Lyman Chapin. “The internet of things: An overview”. *The internet society (ISOC)* 80 (2015), pp. 1–50 (cit. on p. 15).

[Saeedi, 2020] Alieh Saeedi, Eric Peukert, and Erhard Rahm. “Incremental Multi-source Entity Resolution for Knowledge Graph Completion”. *The Semantic Web*. Ed. by Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, et al. 2020, pp. 393–408 (cit. on p. 79).

[Sanislav, 2017] Teodora Sanislav and George Mois. “A dependability analysis model in the context of Cyber-Physical Systems”. *2017 18th International Carpathian Control Conference (ICCC)*. 2017, pp. 146–150 (cit. on p. 68).

[Sanislav, 2019] Teodora Sanislav, Sherali Zeadally, George Dan Mois, and Hacène Fouchal. "Reliability, failure detection and prevention in cyber-physical systems (CPSs) with agents". *Concurrency and Computation: Practice and Experience* 31.24 (2019), e4481. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4481 (cit. on p. 68).

[Schraudner, 2021] Daniel Schraudner. "Stigmergic Multi-Agent Systems in the Semantic Web of Things". *The Semantic Web: ESWC 2021 Satellite Events*. Ed. by Ruben Verborgh, Anastasia Dimou, Aidan Hogan, Claudia d'Amato, Ilaria Tiddi, Arne Bröring, et al. Cham: Springer International Publishing, 2021, pp. 218–229 (cit. on p. 45).

[Scioscia, 2009] Floriano Scioscia and Michele Ruta. "Building a Semantic Web of Things: Issues and Perspectives in Information Compression". *2009 IEEE International Conference on Semantic Computing*. 2009, pp. 589–594 (cit. on p. 32).

[Seidita, 2022] V Seidita, F Lanza, AMP Sabella, A Chella, et al. "Can agents talk about what they are doing? A proposal with Jason and speech acts". *CEUR WORKSHOP PROCEEDINGS*. Vol. 3261. CEUR-WS. 2022, pp. 17–29 (cit. on p. 42).

[Seydoux, 2018] Nicolas Seydoux. "Towards interoperable IOT systems with a constraint-aware semantic web of things". Theses. INSA de Toulouse, 2018 (cit. on pp. 33, 66, 67, 77).

[Sharma, 2010] Abhishek B. Sharma, Leana Golubchik, and Ramesh Govindan. "Sensor Faults: Detection Methods and Prevalence in Real-World Datasets". *ACM Trans. Sen. Netw.* 6.3 (2010) (cit. on pp. 57, 103).

[Shibuya, 2016] M. Shibuya, T. Hasegawa, and H. Yamaguchi. "A Study on Device Management for IoT Services with Uncoordinated Device Operating History". 2016 (cit. on pp. 2, 25).

[Shih, 2016] Chi-Sheng Shih, Jyun-Jhe Chou, Niels Reijers, and Tei-Wei Kuo. "Designing CPS/IoT Applications for Smart Buidlings and Cities". *IET Cyber-Physical Systems: Theory & Applications* 1 (2016) (cit. on p. 57).

[Shoham, 1993] Yoav Shoham. "Agent-oriented programming". *Artificial Intelligence* 60.1 (1993), pp. 51–92 (cit. on p. 40).

[Silva, 2020] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. "BDI Agent Architectures: A Survey". *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 4914–4921 (cit. on pp. 40, 99).

[Sinche, 2020] Soraya Sinche, Duarte Raposo, Ngombo Armando, André Rodrigues, Fernando Boavida, Vasco Pereira, et al. "A Survey of IoT Management Protocols and Frameworks". *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 1168–1190 (cit. on pp. 21, 23, 104).

[Singh, 2017] Munindar P. Singh and Amit K. Chopra. "The Internet of Things and Multiagent Systems: Decentralized Intelligence in Distributed Computing". *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1738–1747 (cit. on p. 44).

[Souza, 2019] Vinicius Souza, Robson Cruz, Walmir Silva, Sidney Lins, and Vicente Lucena. "A Digital Twin Architecture Based on the Industrial Internet of Things Technologies". *2019 IEEE International Conference on Consumer Electronics (ICCE)*. 2019, pp. 1–2 (cit. on p. 37).

[standard, 2013] IEEE 1905.1 standard. "IEEE Standard for a Convergent Digital Home Network for Heterogeneous Technologies Amendment 1: Support of New MAC/PHYs and Enhancements". *IEEE Std 1905.1a-2014 (Amendment to IEEE Std 1905.1-2013)* (2013) (cit. on p. 74).

[Stark, 2019] Rainer Stark and Thomas Damerau. "Digital Twin". *CIRP Encyclopedia of Production Engineering*. Ed. by Sami Chatti and Tullio Tolio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 1–8 (cit. on p. 34).

[Steenwinckel, 2018] Bram Steenwinckel, Pieter Heyvaert, Dieter De Paepe, Olivier Janssens, Sander Vanden Hautte, Anastasia Dimou, et al. "Towards Adaptive Anomaly Detection and Root Cause Analysis by Automated Extraction of Knowledge from Risk Analyses". *SSN@ISWC*. 2018 (cit. on pp. 62, 68, 103, 115).

[Stefano Albrecht, 2020] Stefano Albrecht. *Intro: UK Multi-Agent Systems Symposium - Stefano Albrecht, University of Edinburgh & Turing*. 2020 (cit. on p. 38).

[Steindl, 2020] Gernot Steindl, Martin Stagl, Lukas Kasper, Wolfgang Kastner, and Rene Hofmann. "Generic Digital Twin Architecture for Industrial Energy Systems". *Applied Sciences* 10.24 (2020) (cit. on p. 34).

[Studer, 1998] Rudi Studer, V Richard Benjamins, and Dieter Fensel. "Knowledge engineering: Principles and methods". *Data & knowledge engineering* 25.1-2 (1998), pp. 161–197 (cit. on p. 29).

[Suárez-Figueroa, 2012] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. "The NeOn Methodology for Ontology Engineering". *Ontology Engineering in a Networked World*. Ed. by Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi. 2012, pp. 9–34 (cit. on pp. 76, 101).

[Subramaniam A, 2018] Mona Subramaniam A and Tushar Jain. "Nonlinear Observer-based Fault Diagnosis for a Multi-Zone Building**This work is financially supported by MeitY, Govt. of India, under the Visvesvaraya Ph.D Scheme and SERB - DST under the grant agreement no. ECR/2016/001025." *IFAC-PapersOnLine* 51.24 (2018), pp. 544–549 (cit. on pp. 60, 65).

[Tao, 2022] Fei Tao, Bin Xiao, Qinglin Qi, Jiangfeng Cheng, and Ping Ji. "Digital twin modeling". *Journal of Manufacturing Systems* 64 (2022), pp. 372–389 (cit. on p. 33).

[Tao, 2018] Fei Tao, He Zhang, Ang Liu, and Andrew YC Nee. "Digital twin in industry: State-of-the-art". *IEEE Transactions on industrial informatics* 15.4 (2018), pp. 2405–2415 (cit. on p. 33).

[Tartir, 2005] Samir Tartir, Ismailcem Arpinar, Michael Moore, Amit Sheth, and Boanerges Aleman-Meza. "OntoQA: Metric-Based Ontology Quality Analysis". *IEEE ICDM 2005 Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*. 2005 (cit. on p. 94).

[Teng, 2021] Jing Teng, Changling Li, Yizhan Feng, Taoran Yang, Rong Zhou, and Quan Z. Sheng. "Adaptive Observer Based Fault Tolerant Control for Sensor and Actuator Faults in Wind Turbines". *Sensors* 21.24 (2021) (cit. on pp. 60, 65).

[Thanapalasingam, 2018] Thiviyan Thanapalasingam, Francesco Osborne, Aliaksandr Birukou, and Enrico Motta. "Ontology-based recommendation of editorial products". *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17*. Springer. 2018, pp. 341–358 (cit. on p. 32).

[Uschold, 1996] Mike Uschold and Michael Gruninger. "Ontologies: Principles, methods and applications". *The knowledge engineering review* 11.2 (1996), pp. 93–136 (cit. on p. 91).

[Wang, 2023] Xiaofeng Wang, Yonghong Wang, Zahra Javaheri, Laila Almutairi, Navid Moghadamnejad, and Osama S. Younes. "Federated deep learning for anomaly detection in the internet of things". *Computers and Electrical Engineering* 108 (2023), p. 108651 (cit. on p. 60).

[Weyns, 2005] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. "Environments for Multiagent Systems State-of-the-Art and Research Challenges". *Environments for Multi-Agent Systems*. Ed. by Danny Weyns, H. Van Dyke Parunak, and Fabien Michel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–47 (cit. on p. 40).

[Wilhelm, 2021] Yannick Wilhelm, Peter Reimann, Wolfgang Gauchel, and Bernhard Mitschang. "Overview on hybrid approaches to fault detection and diagnosis: Combining data-driven, physics-based and knowledge-based models". *Procedia CIRP* 99 (2021), pp. 278–283 (cit. on p. 60).

[Wooldridge, 2009] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009 (cit. on p. 38).

[Wu, 2017] Jiewen Wu, Freddy Lécué, Christophe Gueret, Jer Hayes, Sara Van De Moosdijk, Gemma Gallagher, et al. "Personalizing actions in context for risk management using semantic web technologies". *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16*. Springer. 2017, pp. 367–383 (cit. on p. 32).

[Xing, 2020] Liudong Xing. "Cascading failures in internet of things: review and perspectives on reliability and resilience". *IEEE Internet of Things Journal* 8.1 (2020), pp. 44–64 (cit. on pp. 61, 70).

[Xing, 2021] Liudong Xing. "Cascading Failures in Internet of Things: Review and Perspectives on Reliability and Resilience". *IEEE Internet of Things Journal* 8.1 (2021), pp. 44–64 (cit. on pp. 5, 7).

[Xing, 2018] Liudong Xing, Guilin Zhao, Yujie Wang, and Lavanya Mandava. "Competing Failure Analysis in IoT Systems with Cascading Functional Dependence". *2018 Annual Reliability and Maintainability Symposium (RAMS)*. 2018, pp. 1–6 (cit. on p. 50).

[Xu, 2018] Feixiang Xu, Xinhui Liu, Wei Chen, Chen Zhou, and Bingwei Cao. "Ontology-Based Method for Fault Diagnosis of Loaders". *Sensors* 18.3 (2018) (cit. on p. 67).

[Yu, 2021] Gang Yu, Yi Wang, Zeyu Mao, Min Hu, Vijayan Sugumaran, and Y. Ken Wang. "A digital twin-based decision analysis framework for operation and maintenance of tunnels". *Tunnelling and Underground Space Technology* 116 (2021), p. 104125 (cit. on p. 44).

[Yu, 2015] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. "Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things". *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. HotNets-XIV. Philadelphia, PA, USA: Association for Computing Machinery, 2015 (cit. on pp. 50, 70, 72).

[Yu, 2022] Wei Yu, Panos Patros, Brent Young, Elsa Klinac, and Timothy Gordon Walmsley. "Energy digital twin technology for industrial energy management: Classification, challenges and future". *Renewable and Sustainable Energy Reviews* 161 (2022), p. 112407 (cit. on p. 37).

[Zdankin, 2021] Peter Zdankin, Matthias Schaffeld, Marian Waltereit, Oskar Carl, and Torben Weis. "An Algorithm for Dependency-Preserving Smart Home Updates". *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2021, pp. 527–532 (cit. on pp. 6, 26, 50).

[Zhang, 2018] Haibin Zhang, Qian Zhang, Jiajia Liu, and Hongzhi Guo. "Fault Detection and Repairing for Intelligent Connected Vehicles Based on Dynamic Bayesian Network Model". *IEEE Internet of Things Journal* 5.4 (2018), pp. 2431–2440 (cit. on pp. 60, 65).

[Zhou, 2015] Anmei Zhou, Dejie Yu, and Wenyi Zhang. "A research on intelligent fault diagnosis of wind turbines based on ontology and FMECA". *Advanced Engineering Informatics* 29.1 (2015), pp. 115–125 (cit. on p. 67).