Leveraging Semantic Technologies for Collaborative Inference of Threatening IoT Dependencies

Amal Guittoum, François Aïssaoui, Sébastien Bolle, Fabienne Boyer, Noel de Palma

Leveraging Semantic Technologies for Collaborative Inference of Threatening IoT Dependencies

Amal Guittoum
Orange Innovation
Meylan, France
amal.guittoum@orange.com

François Aïssaoui
Orange Innovation
Meylan, France
francois.aissaoui@orange.com

Sébastien Bolle
Orange Innovation
Meylan, France
sebastien.bolle@orange.com

Fabienne Boyer
Univ. of Grenoble alpes - LIG
Grenoble, France
fabienne.boyer@univ-grenoble-alpes.fr

Noel De Palma
Univ. of Grenoble alpes - LIG
Grenoble, France
noel.depalma@univ-grenoble-alpes.fr

ABSTRACT

IoT Device Management (DM) refers to registering, configuring, monitoring, and updating IoT devices. DM faces new challenges as dependencies between IoT devices generate various threats, such as update breaks and cascading failures. Dependencies-related threats are exacerbated by the fragmentation of the DM market, where multiple actors, e.g., operators and device manufacturers, are uncoordinated ensuring DM on interdependent devices, each using its DM solution. Identifying the topology of threatening dependencies is key in developing dependency-aware DM capabilities for legacy DM solutions to tackle dependencies-related threats efficiently. In this work, we apply Semantic Web and Digital Twin technologies to build a decision-support framework that automatically infers the topology of threatening dependencies in IoT systems. We integrate the proposed framework into the in-use Digital Twin platform *Thing in the future* and demonstrate its effectiveness by inferring threatening dependencies in the two settings: a simulated smart home scenario managed by ground-truth DM solutions, such as Orange's implementation of the *USP Controller* and Samsung's *SmartThings Platform*, and a realistic smart home called DOMUS testbed.

CCS Concepts

•Information systems → Resource Description Framework (RDF); Web Ontology Language (OWL); Decision support systems; •Computing methodologies → Ontology engineering;

Keywords

IoT Device Management; Semantic Web; Digital Twin; Ontology; Inference; SHACL; Thing Description; Entity Resolution; Dependencies management; Collaboration.

1. INTRODUCTION

The rapid growth of the Internet of Things (IoT) has an ever-increasing impact on people's lives, leading to promising value-added services in various areas such as smart homes, smart factories, and smart cities. IoT devices represent the main element in creating IoT value by observing, interacting, and implementing functions with minimal human intervention [17]. Therefore, it is critical to ensure their well-functioning through continuous administration. This is referred to as IoT Device Management (DM). In today's IoT systems, DM is provided by multiple actors, which may be device manufacturers, operators, or service providers, each offering their own DM solution [1, 13, 26].

However, these siloed DM solutions are exposed to new threats due to interdependencies among IoT devices. One such threat is cascading failures, where the failure of one device triggers a cascade of undesired state changes in devices that depend on it [32]. 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.

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 [17, 34]. These failures are exacerbated by the uncoordinated execution of DM operations using multiple DM solutions [13] and are usually hard to revert [34].

In addition, dependencies between devices can be exploited to launch attacks that compromise the physical security of customers [33]. These attacks rely on device dependencies in trigger-action platforms such as SmartThings [1], and IFTTT [2] to bring devices to undesired behaviors and are hard to prevent by legacy DM solutions.

Dependency-related threats generate more customer calls to

[1] https://www.smartthings.com/
[2] https://ifttt.com/

customer care applications of DM actors, and their mitigation usually requires human intervention, which increases the cost of DM. For example, the Orange company reports a cost of *20€* for one customer call and *100€* for sending a technician [4], where customers perform *100 calls per week* to request IoT device recovery.

The first step to efficient mitigation and prevention of dependencies -related threats is for DM actors to identify the topology of threatening dependencies. Building such a topology is challenging because IoT dependencies are surprisingly abundant, usually undocumented, rarely static, and governed by different DM actors, i.e., the data describing IoT dependencies are distributed across siloed and heterogeneous DM solutions managed by different DM actors.

To address this challenging task, we propose a framework that allows DM actors to collaboratively infer the topology of threatening dependencies in a managed IoT system. The framework accesses heterogeneous dependencies data from siloed DM solutions and aggregates them using Semantic Web [3] and Digital Twins[3] technologies. Semantic Web technologies are used to enable interoperability across siloed DM solutions. Digital twins allows to build a synchronized view of dynamic IoT dependencies.

The proposed framework 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, dependency-aware planning of DM operations, e.g., *update campaigns* to prevent DM failures, and creating dependency-aware security policies to defend against dependency-related attacks. To the best of our knowledge, this is the first attempt to infer threatening dependencies in IoT systems managed by multiple DM solutions.

The contribution of this work includes (i) The IoT-D ontology providing context-based modeling for threatening dependencies in the form of a knowledge graph (KG); (ii) An entity resolution approach using rules and functions from the advanced features of *Shapes Constraint Language (SHACL) standard*[4] for aggregating extracted dependencies KGs from siloed DM solutions; (iii) A set of SHACL rules to infer the topology of threatening dependencies from the aggregated KGs; (iv) A proof of concept for the proposed framework, integrated into the Orange Digital Twin platform *Thing in the future* (Thing'in)[5] [20].

This paper is organized as follows: First, we present a motivating and representative use case that illustrates dependencies -related threats in a smart home scenario managed by several DM actors. Then, we give our model for threatening dependencies and the proposed framework. After that, we highlight a business use case and evaluation results. Finally, we discuss limitations and related work and provide directions for further research.

[3]"A Digital Twin is a virtual representation of real-world entities and processes": `https://www.digitaltwinconsortium.org/`

[4]`https://www.w3.org/TR/shacl-af/`

[5]Thing'in is a multi-actor open Digital Twin research platform proposed by Orange. It is engaged in building innovative services based on contextual data.

Table 1: Namespaces and prefixes used in this paper

Prefix	Namespace
IoTd	http://www.semanticweb.org/OrangeLab /ontologies/2021/9/IoTD
td	https://www.w3.org/2019/wot/td
hctl	https://www.w3.org/2019/wot/hypermedia
eupont	http://elite.polito.it/ontologies/eupont.owl
dk	http://www.data-knowledge.org/dk/
sh	http://www.w3.org/ns/shacl
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns

2. MOTIVATING USE CASE

We consider a smart home managed by four DM actors. It illustrates the dependencies-related threats that are difficult to overcome with siloed DM solutions. Moreover, it is used in the rest of the paper for explanation purposes.

2.1 Smart Home Architecture

The smart home scenario (see Figure 1) consists of three intelligent systems, namely light control system, temperature control system, and security control system, deployed in a home consisting of a living room and a kitchen:

- Light control 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 control system: Controls the home temperature using a *temperature sensor* and an *air conditioner*. It is mainly based on automation rules[6] 1 and 2 described in Table 2.
- Security control system: Launches *an alarm* when intruders or fires 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 and smoke sensors' services to detect fires. This system is reinforced by automation rules 3 and 4.

Light Bulbs and *the airconditioner* are controllable via the *vocal assistant* and *the smartphone*. A gateway connects devices in the living room to the Internet, while a Wi-Fi repeater connects the kitchen devices. A broker installed in the gateway exposes the sensors services in the form of topics

[6]Automation rules allow automated composition of IoT services in a connected environment. They are in the form IF triggers Then actions.

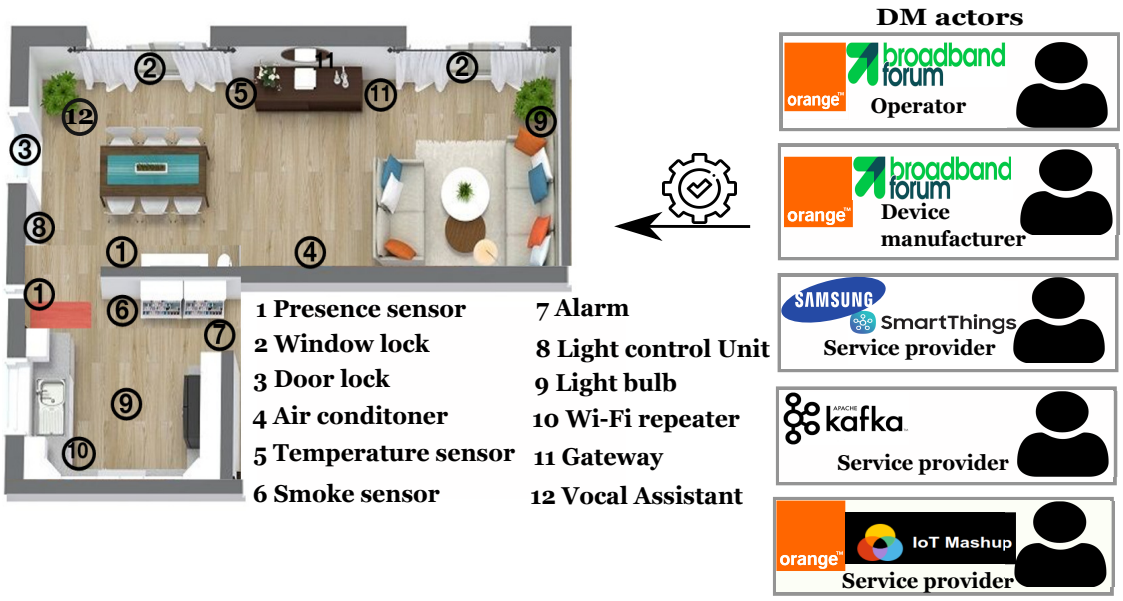

Figure 1: The smart home architecture

such as *temperature topic*. The SmartThings platform[7] [24] enables automation rules 1-3 described in Table 2 and the IoTMashup [8] platform implements the automation rule 4.

Table 2: The smart home automation rules

No.	Type	Automation Rule
1	Comfort	Update the air conditioner regarding the temperature returned by the temperature sensor.
2	Comfort	Open the two windows when the air conditioner is deactivated.
3	Security	Open the door and both windows upon detection of fire.
4	Security	Notifies the User, closes the windows, closes the door, and sets bulbs color to red when detecting an intruder while the User is out of the home.

Devices in the smart home are managed by five DM actors, each proposes its own DM solution: 1) An operator monitoring connectivity devices, e.g., gateway, using *USP controller*[9] [7]. 2) A device manufacturer ensuring DM services on IoT devices such as *firmware update* also using a *USP controller*. 3) A service provider managing automation services using the *SmartThings platform*. 4) A service provider managing service exchange among devices in the broker via *Apache Kafka solution*. 5) A service provider managing also automation services using the *IoTMashup platform*.

[7]SmartThings is Samsung's IoT platform that enables the creation of automation rules.
[8]The IoTMashup is Orange's IoT platform that allows automation across IoT devices.
[9]USP is a standard and a project of the Broadband Forum (BBF) for device management. Orange's USP controller, implementing this standard, is used in the experimentation.

2.2 Illustration of Dependencies-Related Threats

This section illustrates IoT threats stemming from dependencies within the showcased smart home environment.

Example 1. (Cascading Failure) *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 light bulb in the living room fails. This failure propagates to the bulb-dependent devices and services: 1) the light control unit becomes inoperable, 2) the vocal assistant cannot respond to the prompt "Turn off the lights", and 3) the IoTMashup platform can no longer turn the light bulb color to red when it detects intruders (see Rule 4 in Table 2).*

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 the operator reboots the gateway while the device manufacturer 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 [17].*

Example 3. (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 air conditioner and the windows (see Rule 2 in Table 2) to gain access to the home by disabling the air conditioner to open the windows [33].*

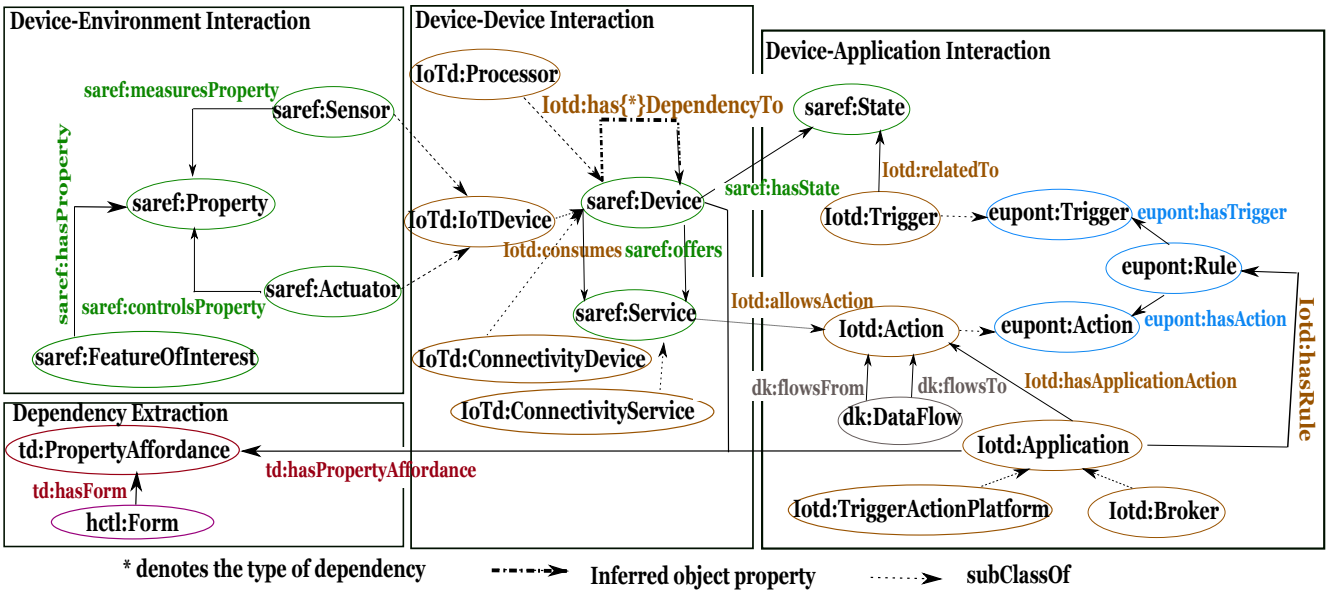

Figure 2: IoT-D ontology

3. 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 IoT-D ontology, which provides a comprehensive representation of threatening dependencies and their context.

3.1 Threatening Dependencies Characterization

We conducted an analysis study of dependencies generating the different threats illustrated in Section 2. The result of this study is a taxonomy of threatening dependencies (See Figure 3). We distinguish two types of threatening dependencies: direct and indirect. Dependencies are direct when

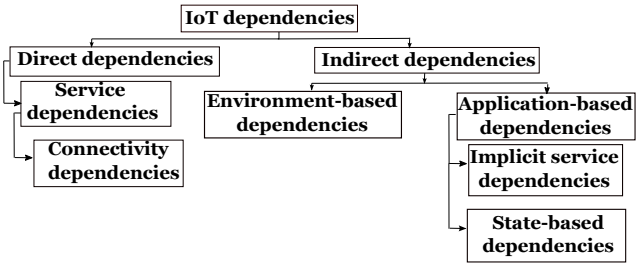

Figure 3: Threatening dependencies taxonomy

IoT devices use 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.

Interactions between sensors and actuators through the physical environment create indirect dependencies between them called *Environment-based dependencies*. For example, the temperature sensor has an environment-based dependency on the air conditioner because it measures the room temperature modified by the air conditioner. 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 2). In addition, IoT applications generate an implicit exchange of services between IoT devices. For example, Rule 1 in Table 2 uses the temperature value returned by the temperature sensor to adjust the air conditioner. Here the air conditioner 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*.

3.2 Threatening Dependencies Data Sources

Threatening dependencies described in Section 3.1 can be derived from a set of information that we call *context data*.

Context data refer to, but are not limited to, connectivity topology,[10] 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: devices services and their interactions within the physical environment are represented in the device manufacturer's USP controller according to the data model* TR-181,[11] *which enables the representation of the* IoT capabilities[12] *of a given device. Interactions at the IoT application level reside in SmartThings and IoTMashup platforms in the form of automation rules represented using a dedicated data model.*[13] *The topology of connectivity is described in the USP controller of the operator according to the data model* TR-181*, which enables the representation of the connectivity topology discovered using the standard* IEEE 1905.1 *[28].*[14]

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*.

3.3 Threatening Dependencies Modeling

We propose an ontology called *IoT-D* (shown in Figure 2) that allows an interoperable representation of threatening dependencies context data in the form of KG, as well as the modalities allowing to extract them. The context data KG is used to infer the KG of threatening dependencies that we will call in the following *threatening dependencies topology*. 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, the use of the KG-based model for IoT dependencies allows for efficient analysis of large, heterogeneous, structured, and unstructured dependency context data [9].

The IoT-D ontology is designed using the ontology engineering methodology NeOn [29] that considers Semantic Web best practices, such as reusing existing ontologies and modularization. It **reuses** the standardized ontology Smart Applications REFerence (SAREF),[15] *Data Knowledge ontology (DK)*,[16] and *End user Programming ontology (EuPont)*.[17] 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.

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 extract the dependency information described in the other modules from devices and applications. It leverages the *Thing Description standard* (TD)[18] with the class *td:PropertyAffordance* to represent heterogeneous data extraction endpoints in an interoperable manner.

Note that the IoT-D ontology conceptualization is representative of IoT systems with no regard to the application domain, which makes it reusable in a wide scope of domains [25]. 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. [19]

4. PROPOSED FRAMEWORK

Relying on the IoT-D ontology, we propose a framework (see Figure 4) that allows on-demand inference of threatening dependencies owned by siloed DM solutions. The framework relies on the digital twin platform *Thing'in* to expose the inferred dependencies topology and to communicate with DM actors. It involves three main steps that rely on Semantic Web standards, namely *Context extraction*, *Entity resolution*, and *Dependency inference*. The first step extracts the

[10] Connectivity topology of a connectivity device such as gateway or WI-FI repeater represents the list of devices connected to it.

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

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

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

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

[15] https://saref.etsi.org/core/v3.1.1/

[16] http://www.data-knowledge.org/dk/1.2/index-en.html

[17] https://elite.polito.it/ontologies/eupont.owl

[18] https://www.w3.org/TR/wot-thing-description/

[19] https://iotdontology.github.io/

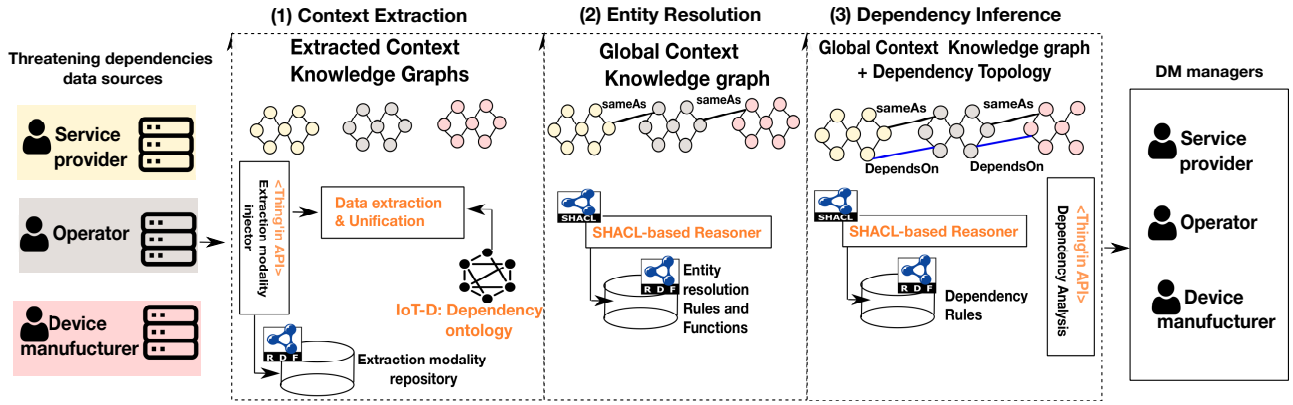

Figure 4: Framework overview - On-demand inference and documentation of threatening dependencies.

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 from the aggregated context KGs. These steps are detailed in the following.

4.1 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 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 format e.g., *json*. It is provided to the framework by DM actors and 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.

Example 5. (Context extraction) *Let us consider the extraction of the connectivity topology from the gateway: the operator injects into Thing'in the extraction modality described in Listing 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 2) is then transformed into KG and stored in Thing'in.*

Listing 1: Connectivity topology extraction modality.

```
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 hctl:forContentType
7     "application/json" ;
8 hctl:hasOperationType
9     td:readProperty ;
10 /*The definition of the extraction link*/
11 hctl:hasTarget
12   "{$USPLink}/dataModel
13    =Device.IEEE1905.NetTopology."
14   ]].
```

Listing 2: The connectivity topology extracted from the gateway by accessing the USP controller of the Operator.

```
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
17    } ...
```

4.2 Step 2: Entity Resolution

4.2.1 Problem Statement

As an IoT device may be managed by multiple DM solutions [13], the extracted context KGs may contain duplicate entities, such as devices with different representations. For example, the temperature sensor may be named *tempSensorModelX* in the device manufacturer's USP controller, while being registered as *TempSensor* in the operator's USP controller. 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 [23]. 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.,

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

4.2.2 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.

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 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 are used to 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 to 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 *sameAs* object property, we perform the ER only between the KGs extracted from the device manufacturers and the KGs extracted from other actors i.e., service providers and operators, since device manufacturers acquire information about all IoT devices and their services.[20] The attribute property *orgIoT:source* [21] is used to define from which actor an entity is extracted. More formally, consider the entity e_i extracted from the operator or the service provider 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 device manufacturer DM solution that satisfies:

$\max \sum_{A_k \in \mathcal{A}^i \cap \mathcal{A}^p} w_k.sim^k(RA_k^i, RA_k^p)$, 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.

Listing 3: SHACL rule for Entity Resolution

```
1 Iotd:EntityResolution
2  rdf:type sh:NodeShape ;
3  sh:targetClass Iotd:IoTDevice ;
4  sh:rule [
5  rdf:type sh:SPARQLRule ;
6  sh:construct """
7 /*Construct the sameAs relationship between the similar device's representations*/
8  CONSTRUCT {
9   $this owl:sameAs ?device .
10          }
11 /*Constraints to check before building the sameAs relationship*/
12 /*Constraints are similarity evaluation between device's representations based on the resolution attributes*/
13   WHERE {
14   /*For each device representation ($this), find its most similar representation (?devices) */
15       {
16        SELECT $this ?device WHERE {
17        {
18 /*Calculate the similarity by calling SHACL functions*/
19          SELECT $this
20        (MAX(0.5*Iotd:similarityFunction(?n2, ?n1)+0.1*Iotd:similarityFunction(?mn2, ?mn1)
21         +0.9*Iotd:similarityFunction(?sn2, ?sn1)) AS ?val) WHERE
22         {
23 /*Select the resolution attributes used in the similarity calculation */
24           $this orgIoT:source ?s .
25           OPTIONAL {$this Iotd:hasDeviceName ?n2 .}
26           OPTIONAL {$this Iotd:hasManufacturerName ?mn2 .}
27           OPTIONAL {$this Iotd:hasSerialNumber ?sn2.}
28           ?device a Iotd:IoTDevice .
29           ?device orgIoT:source ?src .
30           OPTIONAL {?device Iotd:hasDeviceName ?n1.}
31           OPTIONAL {?device Iotd:hasManufacturerName ?mn1.}
32           OPTIONAL {?device Iotd:hasSerialNumber ?sn1.}
33           FILTER (?device!=$this && ?s="other" && ?src="DeviceManufacturer")
34           }
35           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="DeviceManufacturer" &&
46     (0.5*Iotd:similarityFunction(?n2, ?n1) +0.1*Iotd:similarityFunction(?mn2, ?mn1)
47     +0.9*Iotd:similarityFunction(?sn2, ?sn1))=?val)} } }
48     """;] ;
49     .
```

Concretely, consider the SHACL rule presented in Listing 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 and operators (defined by *orgIoT:source="other"*), its most similar representation in the KGs of device manufacturers (defined by *orgIoT:source="DeviceManufacturer"*) is retrieved (line 16) and the *sameAs* relationship is created among them (line 9). Similarity functions (lines 20-21, 46-47) are implemented us-

[20]This is a practical observation.

[21]Refers to the ontology Orange IoT used in Thing'in to manage Digital Twins.

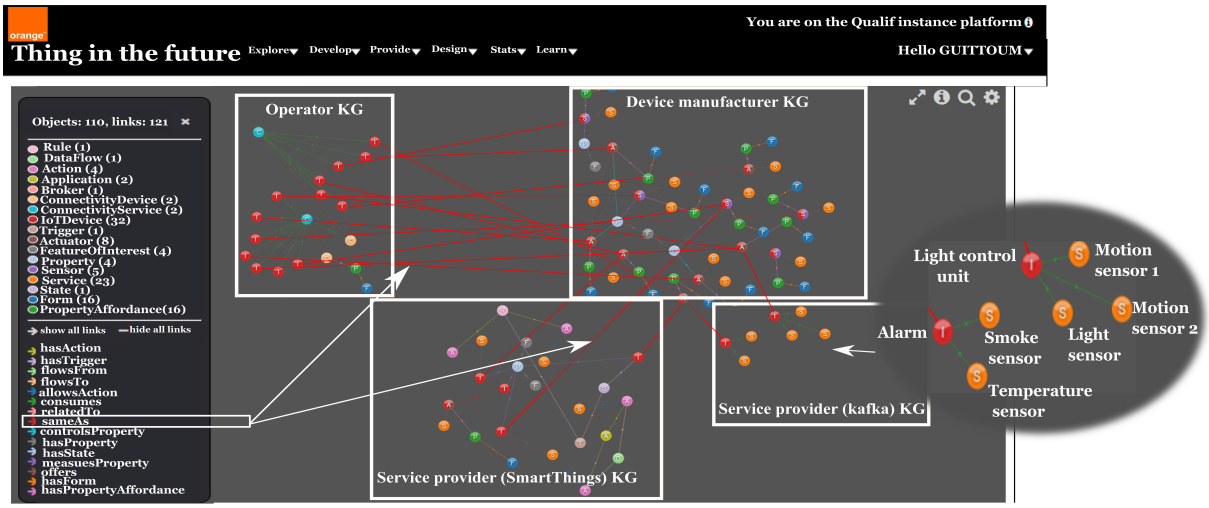

Figure 5: Results of performing the ER on the extracted context KGs (framed graphs) from the use case (red links denotes *owl:sameAs* relationships).

ing SHACL functions (see Listing 4), which are developed using SPARQL extension via registered URI (line 19).[22] To the best of our knowledge, this is the first ER approach based on the SHACL standard. Thing'in view after performing ER on the extracted context KGs from the use case described in Section 2 is shown in Figure 5.

Listing 4: String similarity function as SHACL function

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

4.3 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 Section 3.1. 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:

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

Service dependency inference

The rule presented in Listing 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 14-19).

Listing 5: SHACL rule for Service dependency inference

```
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 /*Constraints are contextual relationships */
14              WHERE {
15 /*Contextual relationships required to infer the service dependency*/
16                  ?device a IoTd:IoTDevice .
17                  ?device core:offers ?service .
18                  $this IoTd:consumes ?service .
19 }""" ;] ;.
```

Connectivity dependency inference

The rule presented in Listing 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 14-21).

Listing 6: SHACL rule for connectivity dependency inference

```
1 IoTd:ConnectivtyDependency
2   rdf:type sh:NodeShape ;
3   sh:targetClass dp:IoTDevice ;
4
5   sh:rule [
6       rdf:type sh:SPARQLRule ;
7       sh:construct """
```

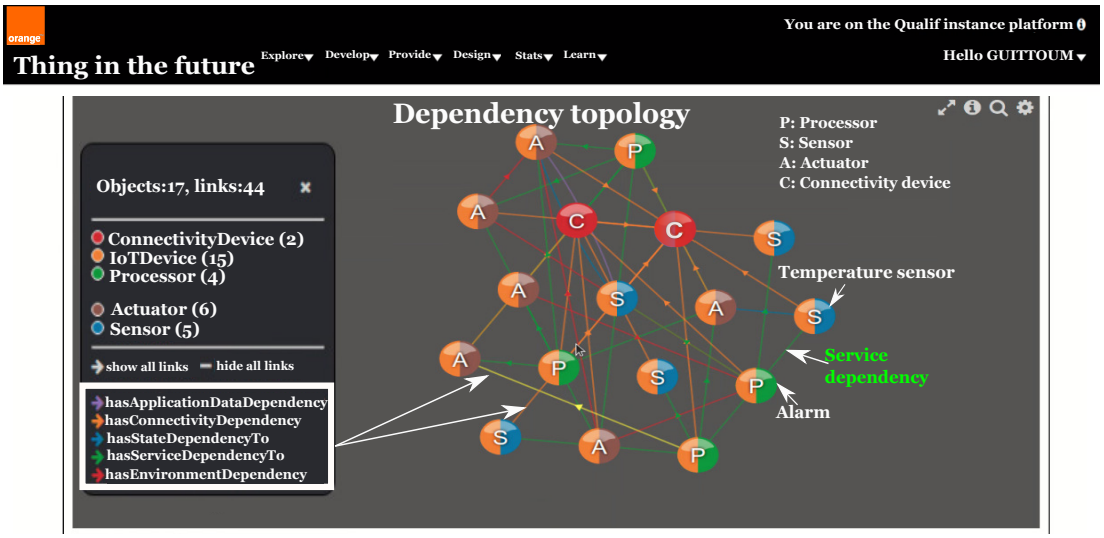

Figure 6: The inferred dependencies topology from the simulated Smart Home Scenario.

```
8       /*Construct the dependency relationship (here is the connectivity dependency) */
9       CONSTRUCT {
10                  $this IoTd:hasConnectivityDependencyTo ?device .
11                }
12 /*Constraints to check before constructing the dependency relationship */
13 /*Constraints are contextual relationships */
14                WHERE {
15 /*Contextual relationships required to infer the connectivity dependency*/
16                    ?device a IoTd:ConnectivityDevice .
17                    ?device core:offers ?service .
18                    $this IoTd:consumes ?service .
19
20                }
21 """ ; ] ;.
```

Environment-based dependency inference

The rule illustrated in Listing 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 15-20).

Listing 7: SHACL rule for environment dependency inference

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

State-based dependency inference

The rule showcased in Listing 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 13-25).

Listing 8: SHACL rule for State-based dependencies inference

```
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      } '' '' '' ;] ;.
```

Implicit service dependency inference

The rule depicted in Listing 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 15-25).

Listing 9: SHACL rule for implicit service dependency inference

```
IoTd:ImplicitServiceDependency
 rdf:type sh:NodeShape ;
 sh:targetClass dp:IoTDevice ;

 sh:rule [
     rdf:type sh:SPARQLRule ;

     sh:construct """
     /*Construct the dependency relationship (here is the implicit service dependency) */
     CONSTRUCT {
                  $this hasImplicitServiceDependencyTo ?device .
               }
/*Constraints to check before constructing the dependency relationship */
/*Constraints are contextual relationships */
               WHERE {
/*Contextual relationships required to infer the implicit service dependency*/
                   ?device a IoTd:IoTDevice .
                   ?device core:offers ?service1 .
                   $this core:offers ?service2 .
                   ?service1 IoTd:allowsAction ?action1.
                   ?service2 IoTd:allowsAction ?action2 .
                   ?dataFlow dk:flowsFrom ?action1.
                   ?dataFlow dk:flowsTo ?action2 .

               }""" ;] ;.
```

Once the global context KG is enriched by the inferred topology of threatening dependencies, it is exposed as a Digital Twins feature. That representation can be easily analyzed and queried by multiple DM actors thanks to Thing'in APIs [23]. The inferred dependency topology from the use case is shown in Figure 6.

5. BUSINESS USE CASE

The inferred threatening dependency topology and its associated context KG can be leveraged in several business use cases regarding the management of dependencies-related threats. This section shows how a cascading failure scenario is efficiently managed and recovery plans are safely performed using our framework.

Example. 6 (Cascading failure management) *Let us take the use case as an example, the user notices that the* alarm *is not responding for intruders detection. He calls the customer care service of the device manufacturer to enquire about the* alarm *recovery. After remote diagnostics using DM, the device manufacturer reports a crash failure on the* alarm *that can be mitigated with the DM operation* reboot*. After rebooting the alarm, the device manufacturer finds out that it is still reporting a crash failure. He assumes that the* alarm *is impacted by a cascading failure and decides to use our framework to foster diagnostics. After analyzing the dependency topology using Thing'in Query API, the device manufacturer notices that the* alarm *uses the services of the* temperature sensor *and the* smoke sensor*, and its state depends on the state of the* presence sensors*. After remote diagnostics of these devices, the device manufacturer discovers a crash failure on the* temperature sensor *that can be mitigated with a* reboot *DM operation. Before rebooting the temperature sensor, he analyzes its context to ensure it can be safely rebooted. He notices that it is used in the SmartThings automation rule by the* airconditioner *(see Rule 1 in Table 2). The device manufacturer notifies the user to disable this rule temporarily. Then, he reboots the* temperature sensor *and the* alarm *successively, and the system returns to the correct state without requiring to send a technician for a physical diagnosis on the devices.*

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

6. 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 home environment, specifically the DOMUS testbed. 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.

6.1 Qualitative Evaluation

6.1.1 Simulated Smart Home Scenario

We evaluated the proposed framework on the simulated smart home scenario in Section 2. IoT devices were simulated using the Open Source USP agents[24]. Regarding the involved DM solutions, We deployed the Orange implementation of the USP controller and the IoTMashup platform locally, and leveraged the cloud developer API of the Smarthings platform 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 6).

6.1.2 Realistic Smart Home: DOMUS Testbed

We evaluated the proposed framework on *DOMUS*[25] (see Figure 8), which represents a connected apartment of $62m^2$ including more than *90* IoT devices installed in five rooms: a living room, a bedroom, a kitchen, a bathroom, and a central 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*,[26] which enables a set of automation rules and allows the devices to access the Internet thanks to *brigdes* devices [27].

Listing 10: DOMUS data extraction modalities.

```
demo:openHab rdf:type dp:Application;
    td:hasPropertyAffordance demo:propappOpenHab1 , demo:propappOpenHab2 .

demo:propappOpenHab1 rdf:type td:PropertyAffordance ;
                     td:hasForm demo:capabilitiesOpenHab1.
demo:capabilitiesOpenHab1 rdf:type hctl:Form ;
            hctl:forContentType "application/json" ;
                      hctl:hasOperationType td:readProperty ;
                      hctl:hasTarget "http://openhab:8080/api/v1/rules/" .
demo:propappOpenHab2 rdf:type td:PropertyAffordance ;
                     td:hasForm demo:capabilitiesOpenHab2.
demo:capabilitiesOpenHab2 rdf:type hctl:Form ;
            hctl:forContentType "application/json" ;
                      hctl:hasOperationType td:readProperty ;
                      hctl:hasTarget "http://openhab:8080/api/v1/connectivityFile/" .
```

[24] https://github.com/BroadbandForum/obuspa

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

[26] https://www.openhab.org/

[27] https://www.openhab.org/docs/concepts/things.html#bridges

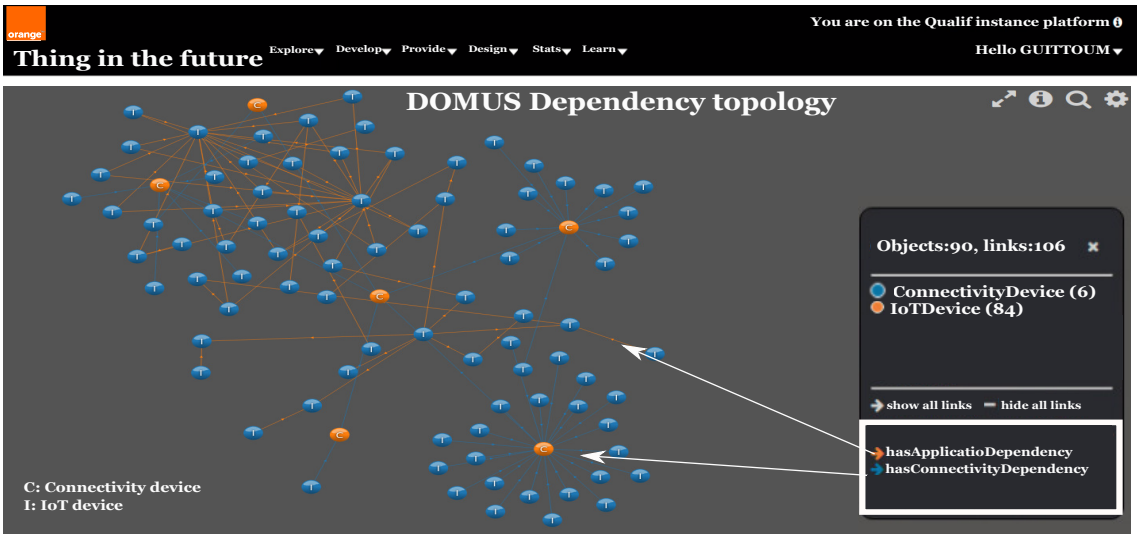

Figure 7: The inferred dependencies topology from DOMUS Testbed.

To infer the DOMUS' dependency topology, we followed a four-step process: 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 10). Third, we defined a mapping that allows the transformation of the extracted context data to KG according to the IoT-D ontology. Last, we ran our framework to infer the threatening dependency topology. The result is depicted in Figure 7 representing the inferred dependency topology. We have automatically identified *106* threatening dependency relationships among DOMUS IoT devices.

6.1.3 *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 [11]. 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 3.

Listing 11: The SPARQL query of the CQ22.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iotd: <http://www.semanticweb.org/OrangeLab/ontologies/2021/9/IoTD#>
PREFIX core: <https://saref.etsi.org/core/>
PREFIX eupont: <http://elite.polito.it/ontologies/eupont.owl#>
PREFIX td: <https://www.w3.org/2019/wot/td#> .
PREFIX hctl: <https://www.w3.org/2019/wot/hypermedia#> .
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>)
}
```

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 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.[28]

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.

6.2 Quantitative Evaluation

6.2.1 *Performance Evaluation*

We carried out a set of performance evaluations by i) measuring the completion time of the ER and dependency inference steps[29] on smart home scenarios with different scales; ii) comparing SHACL to SWRL,[30] another formalism for inference rules used by competing approaches for direct de-

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

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

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

Figure 8: DOMUS Testbed

Table 3: 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

pendencies inference [19] and entity resolution [2]. The 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 Section 2. 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),[31] and SWRL inference is performed using Openllet reasoner with OWL API (version 2.6.5)[32] used by competing approaches. We note that the comparison results are limited by these

[31] https://github.com/TopQuadrant/shacl
[32] https://github.com/Galigator/openllet

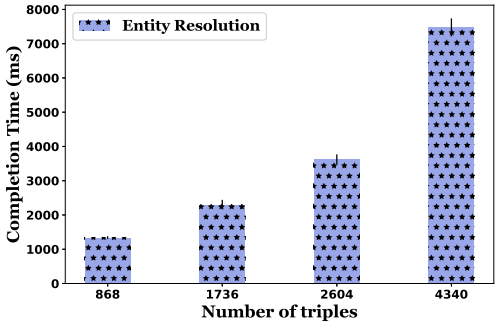

Figure 9: Completion time of the ER step

technological choices. 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 9) 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 failure recovery.

Comparing SHACL with SWRL (see Figure 10) 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 [8]; 2) from the the-

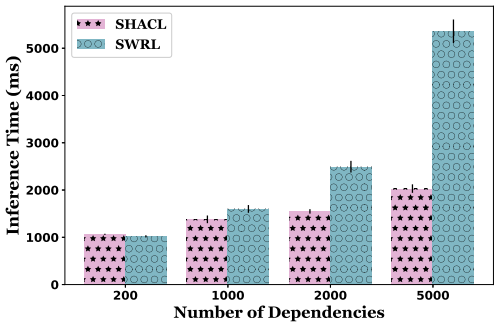

Figure 10: SHACL VS SWRL for dependency inference

oretical complexity perspective, SWRL complexity is exponential [16]. Meanwhile, SHACL complexity depends on the complexity of SPARQL query language, [33] which is polynomial if the graph pattern uses only AND and FILTER operators [21], 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.[34]

6.2.2 *IoT-D ontology Quantitative Evaluation*

We assessed the quality of the IoT-D ontology using the *OntoQA* methodology [31], 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 [11]:

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

$$RR = \frac{|P|}{|P| + |SC|} \quad (1)$$

  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

$$AR = \frac{|AT|}{|C|} \quad (2)$$

  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

$$IR = \frac{|SC|}{|C|} \quad (3)$$

[33] https://book.validatingrdf.com/bookHtml013.html
[34] https://github.com/Orange-OpenSource/ISWC-ReasoningCode

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 [19]. The result (see Table 4) 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: OntoQA Evaluation results

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**

7. LIMITATIONS AND OPEN CHALLENGES

One of the main limitations of our work is that DM actors should manually publish and maintain extraction modalities within our framework. Despite being inspired by the ITU-T Recommendation **Y.4459** [12] 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. An alternative solution is to make use of a multi-agent system [27] where an agent is to be associated with each DM actor. This agent should have proactive capabilities for maintaining the extraction modalities.

Another limitation 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 [6]. They are based on a fully declarative approach for the KG generation process, which is adaptable to additional data sources [22]. This reduces integration costs.

The proposed ER approach performs the ER on the context KGs in two passes: the first pass finds the maximum similarity, and the second identifies the entity associated with the maximum similarity. This process can be optimized by building a machine learning (ML) model to decide the similarity in one pass. Namely, it should decide for two entities whether they are similar. The model can be embedded in a SHACL function to be used in the ER SHACL rule. However, the lack of learning data can be problematic when considering such a model. Another limitation of the proposed ER approach is that it does not guarantee false positives handling, which may affect the overall reliability of the proposed framework. Moreover, the weights of the resolution attribute are currently fixed using the following strategy: 0.9 for device identifiers such as device serial number, 0.5 for device properties such as device name, and 0.1 for device

group properties such as device manufacturer name. To ensure more accurate entity resolution results, these weights should be studied more even by expert study or using ML models such as artificial neural networks (ANN) for parameters optimization [30].

8. RELATED WORK

As far as we know, this is the first attempt to infer threatening dependencies in IoT systems managed by multiple DM solutions. Nonetheless, our work learns from related industry and academia research on multiple axes.

IoT dependencies modeling and extraction are only partly treated in the literature. The works [18], [10], [19] propose static models for IoT dependencies through Satisfiability Modulo Theories (SMT), Markov chain, and semantic ontologies, respectively, to assess IoT risks. However, they do not consider the dynamic nature of IoT dependencies, where dependencies information is extracted and maintained by human intervention. The work [14] comes up with a theoretical proposal for a dynamic graph-based model where explicit dependencies are extracted by analyzing network traffic. However, indirect dependencies, especially application-based ones, are not addressed. Moreover, existing solutions do not consider the practical reality: *IoT is managed by multiple actors*, although dependency information is governed by different DM actors.

The industrial community conducted a few efforts considering the multi-actor aspect of DM. The operator KDDI argues that DM is performed in a horizontal specialization business model, where devices are managed by multiple actors. In their work [26], they propose a federative approach for calculating the cumulative failure rate of a device managed by multiple actors. Orange claims the necessity to federate DM solutions in their presentation [4] for the European Telecommunications Standards Institute (ETSI). Next, they propose in [1] a semantic model for the DM domain to enable the unified management of IoT devices managed by multiple actors. Recently, Jia Yan et al. propose in [13] 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 DM solutions, which may lead to serious security threats. However, dependencies - related threats remain not addressed.

Several ontologies were proposed to model multiple aspects of IoT systems. Many activities were conducted to model the sensing capabilities of sensor networks. The standardized ontology SSN[35] is considered the more valuable effort in this area. Other ontologies have extended this ontology to model other aspects of IoT systems, such as actuation capabilities in *SOSA ontology*, IoT resources in *IoT-Lite ontology*,[36] 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 [19] 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 service and application-based dependencies. Thus, the IoT-D ontology is the first ontology to fill these gaps.

The ER problem has been widely studied for over 70 years in different domains such as knowledge fusion and social network reconciliation [23, 15]. It is also treated in the KG domain for KG completion, where several heuristics are mainly based on ML. However, for cases where training data is unavailable, which is our case, non-learning approaches are suitable [5]. The authors propose in [2] a non-learning ER approach based on SWRL rules that leverages a set of functional keys to identify similar entities. However, 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 SHACL standard, which allow embedding complex similarity computations in ER inference rules.

9. CONCLUSION AND FUTURE WORK

In this work, we shed light on our practical framework that infers threatening dependencies 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 dependency topology and its context KG. The proposed framework leverages established Semantic Web 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 Digital Twin technology to address the dynamic aspect of IoT dependencies. 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.

In future work, we intend to leverage the proposed framework to go one step further in managing dependencies-related threats. Especially considering cascading failure, we intend to enable collaborative and autonomous cascading failure recovery across legacy DM solutions using a cooperative multi-agent system.

10. ACKNOWLEDGMENTS

The authors would like to thank the Maison de la Création et de l'Innovation (MACI) as well as the LIG Lab, responsible

[35] https://www.w3.org/TR/vocab-ssn/

[36] https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/

for the DOMUS platform.

11. REFERENCES

[1] F. Aïssaoui, S. Berlemont, M. Douet, and E. Mezghani. A semantic model toward smart iot device management. In L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, editors, *Web, Artificial Intelligence and Network Applications*, pages 640–650, Cham, 2020. Springer International Publishing.

[2] S. Benbernou, X. Huang, and M. Ouziri. Semantic-based and entity-resolution fusion to enhance quality of big rdf data. *IEEE Transactions on Big Data*, 7(2):436–450, 2021.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[4] S. Bolle, M. Douet, S. Berlemont, E. Mezghani, and F. 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`, 10 2019.

[5] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis. End-to-end entity resolution for big data: A survey. *ArXiv*, abs/1905.06397, 2019.

[6] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In C. Bizer, T. Heath, S. Auer, and T. Berners-Lee, editors, *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*, Apr. 2014.

[7] B. forum. User service protocol tr-369a2, .

[8] M. T. Frank. *Knowledge-Driven Harmonization of Sensor Observations: Exploiting Linked Open Data for IoT Data Streams*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2021.

[9] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool, 2021.

[10] J. Huang, G. Chen, and B. Cheng. A stochastic approach of dependency evaluation for iot devices. *Chinese Journal of Electronics*, 25:209–214, 2016.

[11] A. Z. Ihsan, S. Fathalla, and S. Sandfeld. Diso: A domain ontology for modeling dislocations in crystalline materials. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 1746–1753, 2023.

[12] ITU-T. Recommendation y.4459: Digital entity architecture framework for internet of things interoperability, 2020.

[13] Y. Jia, B. Yuan, L. Xing, D. Zhao, Y. Zhang, X. Wang, Y. Liu, K. Zheng, P. Crnjak, Y. Zhang, D. Zou, and H. Jin. Who's in control? on security risks of disjointed iot device management channels. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1289–1305, New York, NY, USA, 2021. Association for Computing Machinery.

[14] M. Laštovička and P. Čeleda. Situational Awareness: Detecting Critical Dependencies and Devices in a Network. In D. Tuncer, R. Koch, R. Badonnel, and B. Stiller, editors, *11th IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, volume LNCS-10356 of *Security of Networks and Services in an All-Connected World*, pages 173–178, Zurich, Switzerland, July 2017. Springer International Publishing. Part 6: Ph.D. Track: Methods for the Protection of Infrastructure and Services.

[15] B. Li, Y. Liu, A. Zhang, W. Wang, and S. Wan. A survey on blocking technology of entity resolution. *Journal of Computer Science and Technology*, 35:769 – 793, 2020.

[16] J. Mei and H. Boley. Interpreting swrl rules in rdf graphs. *Electronic Notes in Theoretical Computer Science*, 151(2):53–69, 2006. Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005).

[17] E. Mezghani, S. Berlemont, and M. Douet. Autonomic coordination of iot device management platforms. In *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 30–35, 2020.

[18] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 180–188, 2016.

[19] M. Mohsin, Z. Anwar, F. Zaman, and E. Al-Shaer. Iotchecker: A data-driven framework for security analytics of internet of things configurations. *Computers Security*, 70:199–223, 2017.

[20] Orange. Thing'in platform, .

[21] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. M. Aroyo, editors, *The Semantic Web - ISWC 2006*, pages 30–43, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[22] J. A. Rojas, M. Aguado, P. Vasilopoulou, I. Velitchkov, D. Van Assche, P. Colpaert, and R. Verborgh. Leveraging semantic technologies for digital interoperability in the european railway domain. In A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, and H. Alani, editors, *The Semantic Web – ISWC 2021*, pages 648–664, Cham, 2021. Springer International Publishing.

[23] A. Saeedi, E. Peukert, and E. Rahm. Incremental multi-source entity resolution for knowledge graph completion. In A. Harth, S. Kirrane, A.-C.

Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, editors, *The Semantic Web*, pages 393–408, Cham, 2020. Springer International Publishing.

[24] Samsung. Smartthings, .

[25] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil. Iot-o, a core-domain iot ontology to represent connected devices networks. In *20th International Conference on Knowledge Engineering and Knowledge Management - Volume 10024*, EKAW 2016, page 561–576, Berlin, Heidelberg, 2016. Springer-Verlag.

[26] M. Shibuya, T. Hasegawa, and H. Yamaguchi. A study on device management for iot services with uncoordinated device operating history. 2016.

[27] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[28] I. 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.

[29] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López. *The NeOn Methodology for Ontology Engineering*, pages 9–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[30] L. B. Tan and N. D. P. Nhat. Prediction and optimization of process parameters for composite thermoforming using a machine learning approach. *Polymers*, 14(14), 2022.

[31] S. Tartir, I. Arpinar, M. Moore, A. Sheth, and B. Aleman-Meza. Ontoqa: Metric-based ontology quality analysis. 11 2005.

[32] L. Xing. Cascading failures in internet of things: Review and perspectives on reliability and resilience. *IEEE Internet of Things Journal*, 8(1):44–64, 2021.

[33] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, New York, NY, USA, 2015. Association for Computing Machinery.

[34] P. Zdankin, M. Schaffeld, M. Waltereit, O. Carl, and T. Weis. An algorithm for dependency-preserving smart home updates. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 527–532, 2021.