Compas'2022 : Parallélisme/ Architecture/ Système
MIS/UPJV - Amiens France, 5-8 juillet 2022

A framework for on-demand inference of IoT dependencies toward collaborative IoT Device management

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

[1]Orange Innovation, Meylan, France
{firstname.lastname}@orange.com

[2]University of Grenoble Alpes – LIG, Grenoble, France
{firstname.lastname}@univ-grenoble-alpes.fr

Abstract
IoT Device Management (DM) refers to registering, configuring, observing, and updating IoT devices that are deployed in a given environment. In practice, DM tends to be ensured by several actors such as operators, device manufacturers, or service providers, each operating independently via its legacy DM solution. Such siloed DM capabilities are limited in addressing IoT threats related to device dependencies, such as cascading failures. Indeed, these threats spread across devices managed by different DM actors, and their mitigation can no longer be performed without collaborative DM efforts. To achieve such collaboration, DM actors need to be aware of the global topology of dependencies. Determining such topology is challenging, requiring to infer dependencies from the data held by different actors. In this work, we propose a framework that infers such topology on-demand by accessing and aggregating data from legacy DM solutions managed by different actors. Thanks to Semantic Web standards, the framework enables unified data extraction, interpretation, and usage across heterogeneous DM solutions. We leverage the digital twin technology to expose on-demand the global topology of dependencies. We have integrated our solution within Orange's digital twin platform *Thing in the future* and demonstrate its effectiveness by automatically inferring the dependencies topology in a smart home scenario managed by ground truth DM solutions such as Orange's *USP Controller* and Samsung's *SmartThings* platform.

Keywords : IoT Device management, Collaboration, Digital twin, Semantic Web standards

1. Introduction

The rapid growth of the Internet of Things (IoT) is increasingly impacting people's lives, leading to promising added-value services such as smart homes and smart cities [16]. IoT devices represent the main element in creating IoT value by observing, interacting, and implementing functions with minimal human intervention [24]. Consequently, it is critical to ensure their well-functioning and security. This is referred to as IoT Device Management (DM)[1].
In practice, DM is ensured by multiple actors, which can be operators, manufacturers, or service providers, each offering their own DM solution. Therefore, DM is performed in a dis-

[1] IoT Device management is defined by a set of operations that enable remote management of IoT devices throughout their lifecycles, such as firmware updates, reboot, and configuration.

tributed and siloed manner [30, 9, 20]. Although siloed DM solutions are capable of keeping the well-functioning of IoT devices, they are still limited when it comes to the emerging IoT threats, which are related to the dependencies among IoT devices managed by different actors. One such threat is the phenomenon of cascading failure, where the failure of one device triggers a cascade of undesired state changes to devices that depend on it [31]. Failures during the execution of DM operations e.g., *firmware update* on interdependent devices are another dependencies-related threat. Indeed, DM operations cause devices to be temporarily unable to provide their services, which can lead to failures on dependent devices [24, 33]. In addition, dependencies between devices are exploited to launch attacks in connected environments [32]. To combat these threats, DM actors must be aware of the dependencies topology between IoT devices. These dependencies are surprisingly abundant, usually undocumented, rarely static [10], and they are governed by different DM actors i.e., dependencies information is distributed across siloed DM solutions managed by different DM actors.

In this work, we propose a collaborative framework for on-demand inference of the global topology of IoT dependencies, by accessing and aggregating data from legacy DM solutions.

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 present a characterization for IoT dependencies, the proposed framework, and the evaluation results. Finally, we discuss related work and point out our future work.

2. Motivating use case

2.1. Description

The smart home scenario consists of three intelligent systems (described in Appendix Table 1), namely *lighting control system*, *temperature control system*, and *security control system*, deployed in a home consisting of a room and a kitchen (an illustrative view is given in Appendix Figure 4). A gateway connects devices in the room to the Internet, while a Wi-Fi repeater connects devices in the kitchen. A publish/subscribe broker installed in the gateway exposes the sensors services in the form of topics such as *temperature topic*. The SmartThings platform[2] is used to enable a set of automation rules described in Appendix Table 2. From the perspective of DM, devices in the smart home are managed by four DM actors: 1) An operator that monitors the connectivity devices e.g., gateway, using *USP controller*[3]. 2) A device manufacturer that ensures DM services such as *firemware update* also using the *USP controller*. 3) A service provider that ensures automation services, using the *SmartThings platform*. 4) Another service provider managing service exchange among devices through the broker using the *Apache Kafka solution* [1].

2.2. Illustration of the dependencies-related threats

The main problem caused by dependencies is the phenomenon of cascading failure. Let us take the scenario of the light bulb in the room fails. This failure propagates to the light 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 SmartThings platform can no longer turn the light bulb to red when it detects intruders. Another issue is failures when performing DM operations on interdependent devices. Suppose the gateway is rebooted while updating 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

[2] SmartThings is Samsung's IoT platform that enables the creation of automation rules [5].

[3] User Service Protocol (USP) is a standard and an ongoing project of the Broadband Forum (BBF) for device management [8]. USP controller is Orange's implementation of this standard.

being downloaded correctly and the vocal assistant failing after the update [24, 33]. Note that DM failures especially firmware update ones are difficult to mitigate even with the reboot operation [24]. Dependencies are also exploited to compromise the user's security. For example, an attacker can exploit the state dependency between the air conditioner and the windows (see Rule 2 in Appendix Table 2) to gain access to the home, by disabling the air conditioner to open the windows [32].

3. IoT dependencies characterization

We conducted an analysis study of the dependencies that lead to the threats illustrated in Section 2. The result of this study is a taxonomy of threatening IoT dependencies (See Figure 1). We distinguish two types of IoT dependencies: direct and indirect. IoT dependencies are direct when 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 offered by 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 (see Rule 2 in Table 2). In addition, IoT applications generate an implicit exchange of services between IoT devices. Consider an application using the temperature value returned by the temperature sensor to adjust the air conditioner, here the air conditioner implicitly uses the temperature sensor service (see Rule 1 in Appendix Table 2). We call these dependencies *Implicit service dependencies*.

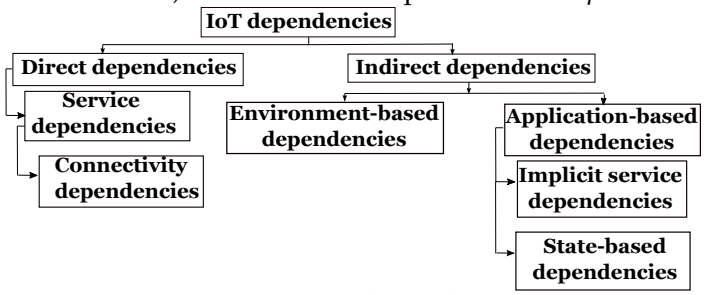


Figure 1: IoT dependencies taxonomy

4. Proposed framework

We propose a collaborative framework (see Figure 2) that enables on-demand inference of IoT dependencies (presented in Section 3) from siloed DM solutions. It relies on Orange's digital twin[4] platform *Thing in the future* (Thing'in)[5] [7] to expose the inferred dependencies topology and to communicate with the DM actors. It is based on the knowledge graph[6] (KG) model for IoT dependencies, with nodes representing devices and edges representing dependency

[4] "A digital twin is a virtual representation of real-world entities and processes, synchronized at a specified frequency and fidelity" [2].

[5] A multi-actor digital twin platform engaged in building innovative services based on contextual data.

[6] "Knowledge Graphs are very large semantic nets that integrate various and heterogeneous information sources to represent knowledge about certain domains of discourse" [14].

relationships between them. More precisely, we define the *IoT-D ontology*[7] (described in Appendix B) that allows representing, in the form of KGs, a set of *context data* from which IoT dependencies can be inferred. Context data refers to data describing connectivity topology, service exchanges between devices, or applicative automation rules. IoT dependencies KG is inferred from context KGs using the three steps relying on Semantic Web standards[8] : *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 dependencies topology from the aggregated context KGs.

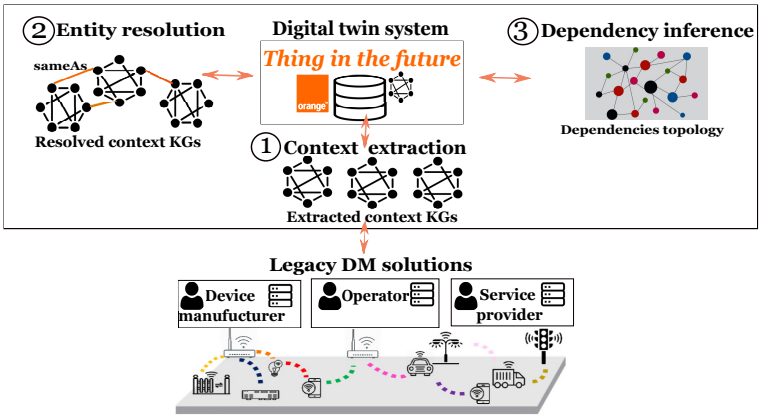


Figure 2: Framework overview

4.1. Step 1: Context extraction

This step aims to extract the context data described in the IoT-D ontology and transform it into KGs. This data is distributed across DM solutions managed by different DM actors. Take the use case example: the connectivity topology is owned by the operator, the service exchange belongs to the service provider managing the Kafka broker, the device capabilities and services are managed by the device manufacturer, and the device interactions at the application layer are owned by the service provider managing the SmartThings platform. To extract this data, we rely on the *Thing Description (TD) standard* [6] to describe the extraction modalities that allow 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 process. Note that the use of the TD standard enables technology-agnostic data extraction, which eases the integration of the heterogeneous DM solutions. An example of the extraction modality provided by the operator to extract the connectivity topology is given in Appendix C. The result of context extraction from the use case is shown in Appendix Figure 6.

4.2. Step 2: Entity resolution

oThe 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, but it is registered as *Temperature* in the broker. These deduplicated 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) prob-

[7] Ontology is a shared vocabulary that describes a set of concepts and relationships between them [17].

[8] Semantic Web standards or technologies, standardized by the World Wide Web Consortium (W3C), "enable people to create data stores on the Web, build vocabularies, and write rules for handling data " [3]. Ontologies and inference rules that we use in this work are part of Semantic Web standards.

lem, which has been studied extensively for over 70 years [13, 29, 22, 12]. To address this problem, we propose an ER approach based on inference rules[9] by leveraging the *Shapes Constraint Language (SHACL) standard* [4]: We assume that extracted KGs contain a set of DM metadata called *resolution attributes* for each entity e.g., device hostname for IoT devices. These attributes are used by a SHACL rule (see Appendix Listing 5) to automatically creates the *sameAs* relationship between the similar entities in the extracted KGs. The similarity is computed using the weighted[10] sum of string similarities e.g., *Jaro similarity* [19] between the resolution attributes. The SHACL rule performs the similarity calculation using SHACL functions. For an example of strict string matching formalized as a SHACL function, see Appendix Listing 4. To the best of our knowledge, this is the first ER approach based on the SHACL standard. The result of performing the proposed ER approach for the use case is shown in Appendix Figure 7.

4.3. Step 3: Dependency inference

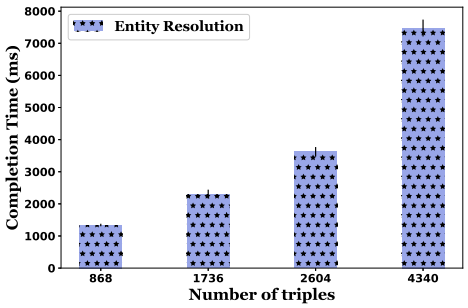
We propose a set of SHACL rules that allow automatic inference of direct and **indirect** dependencies from the resolved KGs. These SHACL rules infer dependencies relationships between devices based on existing relationships in the resolved KGs. For instance, the SHACL rule presented in Appendix Listing 3 infers state-based dependency by creating *hasStateDependencyTo* relationship between two devices when it finds an IoT application that acts on one device according to the state of another. The inferred dependencies topology from the use case is shown in Appendix Figure 8. It is provided as a service for DM actors to enable human-based risk assessment and it can be integrated to other processes to mitigate dependencies-related threats.

5. Evaluation

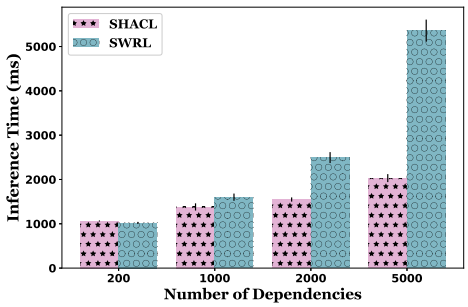
We carried out a set of quantitative evaluations by: 1) measuring the completion time of the ER and dependency inference steps on large-scale smart home scenarios; 2) comparing SHACL to Semantic Web Rule Language (SWRL) [11], another formalism for inference rules used by competing approaches for direct dependencies inference [26] and entity resolution [11]. The comparison was performed according to step 3, dependency inference. The test data sets consist of a set of smart home scenarios with different scales built using a data generator, which injects duplicates into the semantic description of the smart home scenario presented 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 [12](version 1.0.1) and SWRL inference is performed using Openllet reasoner with OWL API [13](version 2.6.5).
We found that the completion time of the dependency inference step is almost negligible (with 32.5 ms for 868 triples and 63 ms for 4340 triples) compared to the ER completion time (see Figure 3a), which 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 to be sufficient for human-based risk assessment. However, this time should be discussed more from the perspective of using the proposed framework for automating the management of cascading failures. The result of comparing SHACL with SWRL (see Figure 3b) 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 perspec-

[9] Inference rule allows deriving new semantic relations in a KG by reasoning on existing ones.
[10] For each resolution attribute, we associate a weight such as 0.9 for the serial number and 0.5 for the hostname.
[11] `https://www.w3.org/Submission/SWRL/`
[12] `https://github.com/TopQuadrant/shacl`
[13] `https://github.com/Galigator/openllet`

tives: 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 [15]; 2) from the theoretical complexity perspective, SWRL complexity is exponential [23]. Meanwhile, SHACL complexity is polynomial[14] [28]. We make available online the source code of the presented evaluation and the generated data sets [15].



(a) Completion time of the ER step



(b) SHACL VS SWRL according to dependencies inference time

Figure 3: Experiments results

6. Related work

As far as we know, this is the first attempt to infer IoT dependencies in multi-actor DM scenarios. Nonetheless, our work learns from related research: IoT dependencies modeling and extraction are slightly treated in the literature. The works [25], [18], [26] 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 [21] 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 distributed across siloed DM solutions managed by multiple actors.

7. Conclusion and Future work

In this work, we shed light on our practical framework that infers IoT dependencies topology from the siloed DM solutions in order to help decision-making when addressing dependencies-related threats. This framework leverages the established Semantic Web standards of the World Wide Web Consortium (W3C) such as TD and SHACL. It makes use of the digital twin technology to address the dynamic aspect of IoT dependencies. It is based on a three-step process involving extracting data from siloed DM solutions, resolving this data, and finally inferring device dependencies. We validated the proposed solution by generating dependencies topology from smart home scenarios. However, we believe that our approach is generic enough to be applied to IoT applications other than smart homes. Moreover, it can be applied in other domains where there is a need to connect siloed and dynamic data sources to unlock innovative use cases. In future work, we plan to explore another collaborative service, which is the collaborative management of the cascading failure using the extracted topology of dependencies. We intend to enable automatic failure recovery in multi-actor scenarios.

[14] https://book.validatingrdf.com/bookHtml013.html

[15] https://github.com/Orange-OpenSource/ISWC-ReasoningCode

Bibliographie

1. APACHE KAFKA. – https://kafka.apache.org/. Online; accessed 30 March 2022.
2. Digital Twin Consortium. – https://www.digitaltwinconsortium.org/. Online; accessed 2 avril 2022.
3. Semantic Web. – https://www.w3.org/standards/semanticweb/. Online; accessed 1 avril 2022.
4. SHACL Advanced Features. – https://www.w3.org/TR/shacl-af/. Online; accessed 21 January 2022.
5. SmartThings rule. – https://developer-preview.smartthings.com/docs/automations/rules/. Online; accessed 6 avril 2022.
6. Thing Description (TD) Ontology. – https://www.w3.org/2019/wot/td. Online; accessed 21 January 2022.
7. Thing in the future. – https://www.thinginthefuture.com/. Online; accessed 23 March 2022.
8. User service protocol tr-369a2. – https://usp.technology/specification/01-index-introduction.html. Online; accessed 22 March 2022.
9. Aïssaoui (F.), Berlemont (S.), Douet (M.) et Mezghani (E.). – A semantic model toward smart iot device management. – In Barolli (L.), Amato (F.), Moscato (F.), Enokido (T.) et Takizawa (M.) (édité par), *Web, Artificial Intelligence and Network Applications*, pp. 640–650, Cham, 2020. Springer International Publishing.
10. Barham (P.), Black (R.), Goldszmidt (M.), Isaacs (R.), MacCormick (J.), Mortier (R.) et Simma (A.). – *Constellation: automated discovery of service and host dependencies in networked systems*. – Rapport technique nMSR-TR-2008-67, April 2008.
11. Benbernou (S.), Huang (X.) et Ouziri (M.). – Semantic-based and entity-resolution fusion to enhance quality of big rdf data. *IEEE Transactions on Big Data*, vol. 7, n2, 2021, pp. 436–450.
12. Christophides (V.), Efthymiou (V.), Palpanas (T.), Papadakis (G.) et Stefanidis (K.). – End-to-end entity resolution for big data: A survey. *ArXiv*, vol. abs/1905.06397, 2019.
13. Ebraheem (M.), Thirumuruganathan (S.), Joty (S.), Ouzzani (M.) et Tang (N.). – Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, vol. 11, n11, jul 2018, p. 1454–1467.
14. Fensel (D.), Şimşek (U.), Angele (K.), Huaman (E.), Kärle (E.), Panasiuk (O.), Toma (I.), Umbrich (J.) et Wahler (A.). – *Introduction: What Is a Knowledge Graph?*, pp. 1–10. – Cham, Springer International Publishing, 2020.
15. Frank (M. T.). – *Knowledge-Driven Harmonization of Sensor Observations: Exploiting Linked Open Data for IoT Data Streams*. – Thèse de PhD, Karlsruher Institut für Technologie (KIT), 2021.
16. Gu (T.), Fang (Z.), Abhishek (A.), Fu (H.), Hu (P.) et Mohapatra (P.). – Iotgaze: Iot security enforcement via wireless context analysis. – In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 884–893, 2020.
17. Guarino (N.), Oberle (D.) et Staab (S.). – *What Is an Ontology?*, pp. 1–17. – Berlin, Heidelberg, Springer Berlin Heidelberg, 2009.
18. Huang (J.), Chen (G.) et Cheng (B.). – A stochastic approach of dependency evaluation for iot devices. *Chinese Journal of Electronics*, vol. 25, 2016, pp. 209–214.
19. Jaro (M. A.). – Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, vol. 84, n406, 1989, pp. 414–420.
20. Jia (Y.), Yuan (B.), Xing (L.), Zhao (D.), Zhang (Y.), Wang (X.), Liu (Y.), Zheng (K.), Crnjak

(P.), Zhang (Y.), Zou (D.) et Jin (H.). – 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*, CCS '21, p. 1289–1305, New York, NY, USA, 2021. Association for Computing Machinery.

21. Laštovička (M.) et Čeleda (P.). – Situational Awareness: Detecting Critical Dependencies and Devices in a Network. – In Tuncer (D.), Koch (R.), Badonnel (R.) et Stiller (B.) (édité par), *11th IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, *Security of Networks and Services in an All-Connected World*, volume LNCS-10356, pp. 173–178, Zurich, Switzerland, juillet 2017. Springer International Publishing. – Part 6: Ph.D. Track: Methods for the Protection of Infrastructure and Services.
22. Li (B.), Liu (Y.), Zhang (A.), Wang (W.) et Wan (S.). – A survey on blocking technology of entity resolution. *Journal of Computer Science and Technology*, vol. 35, 2020, pp. 769 – 793.
23. Mei (J.) et Boley (H.). – Interpreting swrl rules in rdf graphs. *Electronic Notes in Theoretical Computer Science*, vol. 151, n2, 2006, pp. 53–69. – Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005).
24. Mezghani (E.), Berlemont (S.) et Douet (M.). – Autonomic coordination of iot device management platforms. – In *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 30–35, 2020.
25. Mohsin (M.), Anwar (Z.), Husari (G.), Al-Shaer (E.) et Rahman (M. A.). – Iotsat: A formal framework for security analysis of the internet of things (iot). – In *2016 IEEE Conference on Communications and Network Security (CNS)*, pp. 180–188, 2016.
26. Mohsin (M.), Anwar (Z.), Zaman (F.) et Al-Shaer (E.). – Iotchecker: A data-driven framework for security analytics of internet of things configurations. *Computers Security*, vol. 70, 2017, pp. 199–223.
27. Monge Roffarello (A.). – End user development in the iot: A semantic approach. – In *2018 14th International Conference on Intelligent Environments (IE)*, pp. 107–110, 2018.
28. Pérez (J.), Arenas (M.) et Gutierrez (C.). – Semantics and complexity of sparql. – In Cruz (I.), Decker (S.), Allemang (D.), Preist (C.), Schwabe (D.), Mika (P.), Uschold (M.) et Aroyo (L. M.) (édité par), *The Semantic Web - ISWC 2006*, pp. 30–43, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
29. Saeedi (A.), Peukert (E.) et Rahm (E.). – Incremental multi-source entity resolution for knowledge graph completion. – In Harth (A.), Kirrane (S.), Ngonga Ngomo (A.-C.), Paulheim (H.), Rula (A.), Gentile (A. L.), Haase (P.) et Cochez (M.) (édité par), *The Semantic Web*, pp. 393–408, Cham, 2020. Springer International Publishing.
30. Shibuya (M.), Hasegawa (T.) et Yamaguchi (H.). – A study on device management for iot services with uncoordinated device operating history. – 2016.
31. Xing (L.). – Cascading failures in internet of things: Review and perspectives on reliability and resilience. *IEEE Internet of Things Journal*, vol. 8, n1, 2021, pp. 44–64.
32. Yu (T.), Sekar (V.), Seshan (S.), Agarwal (Y.) et Xu (C.). – 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*, HotNets-XIV, New York, NY, USA, 2015. Association for Computing Machinery.
33. Zdankin (P.), Schaffeld (M.), Waltereit (M.), Carl (O.) et Weis (T.). – 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)*, pp. 527–532, 2021.

A. The smart home architecture

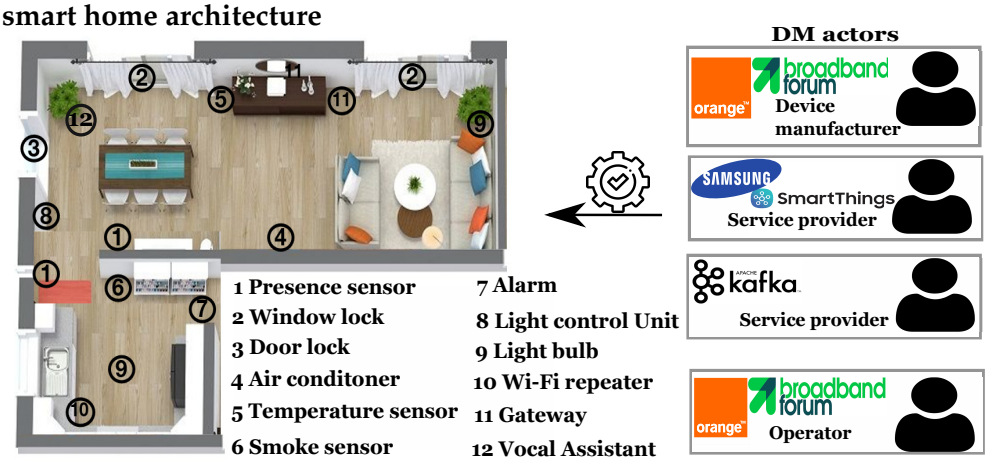


Figure 4: The simulated smart home architecture

System	Description
Light control system	Ensures comfortable and optimal light management in the home. The system relies on a light sensor, two presence detection sensors, bulbs installed in the room and the kitchen, and a central light control unit. The latter uses the brightness measurement service supplied by the light sensor, the presence detection services provided by the presence sensors, and the bulbs control services for controlling light according to the home user's presence. Bulbs are controllable through the vocal assistant.
Temperature control system	Relies on an air conditioner and a temperature sensor to control the home temperature. It is mainly based on the automation rules 1 and 2 described in Table 2. The air conditioner is also controllable via the vocal assistant and the smartphone.
Security control system	Launches an alarm when intruders or fires are detected. It consists of an alarm system that uses the presence detection services provided by the presence sensors to detect intruders. It also uses temperature measurement and smoke detection services to detect fires. This system is reinforced by the automation rules 3 and 4 described in Table 2.

Table 1: The smart home systems

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.

Table 2: The smart home automation rules

B. IoT-D ontology

We designed the IoT-D ontology (shown in Figure 5) that provides an interoperable description of the context from which the various types of dependencies described in Section 3 can be derived. It involves three modules:

- 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[16] 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 representation of service and connectivity dependencies.

- Device-Environment Interaction module, also based on the SAREF ontology, represents the interaction of devices within the physical environment through sensing and actuation, enabling the representation of environment-based dependencies.

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

- Device-Application Interaction module describes device interactions in IoT applications. Based on the EuPont [17] [27] ontology, an IoT application is represented using a set of *IoTD:Rule* and *IoTD:Action* that are executed by *saref:Service*. *IoTD:Rule* is in the form IF *IoTD:Trigger* Then *IoTD:Action* and triggers are related to a change in devices state *saref:State*. 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. It also allows representing implicit service dependencies in the form of data flows between *IoTD:Action*, represented via the DK [18] Ontology class *dk:DataFlow*.

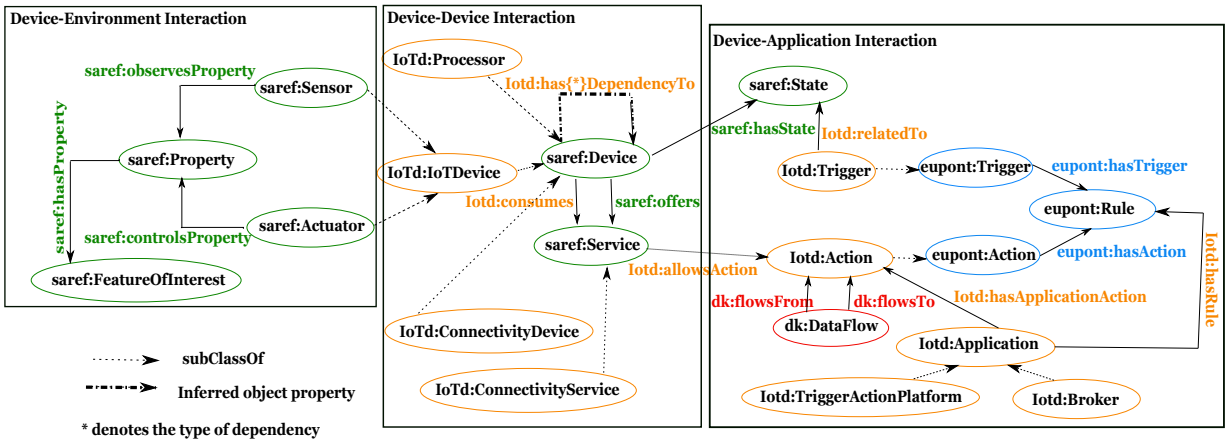


Figure 5: IoT-D ontology

C. Extraction modalities

Consider the extraction of the connectivity topology from the gateway. Thus, the operator injects into Thing'in the extraction modality described in Listing 1 that enables the extraction of the connectivity topology from the operator's USP controller, using the link presented with the property *hasTarget* (line 10-11). The extracted data shown in Listing 2 is then transformed to KG according to the vocabulary of the ontology IoT-D and stored in Thing'in.

Listing 1: Connectivity topology extraction modality.

```
1 /*Declaration of the extraction data
  source here is the gateway*/
2 demo: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."]].
```

Listing 2: Part from data extracted from the gateway by accessing USP controller of the Operator.

```
1 [ {"requested_path":
2 "Device.IEEE1905.NetTopology.",
3 "resolved_path_results": [
4 {
5 "resolved_path":
6 "Device.IEEE1905.NetTopology.",
7 "result_params": [
8  {
9  "param_name": "IEEE1905Device.1.FriendlyName",
10  "value": "TempSensor"
11  },
12  { "param_name": "IEEE1905Device.2.FriendlyName",
13  "value": "bulb1"
14 },
15  { "param_name": "IEEE1905Device.3.FriendlyName",
16    "value": "LightSensor"
17  },
18  {
19  "param_name": "IEEE1905Device.4.FriendlyName",
20 "value": "AirConditioner"
21 } ...
```

[17] https://elite.polito.it/ontologies/eupont.owl
[18] http://www.data-knowledge.org/dk/1.2/index-en.html

D. SHACL rules and functions

This part illustrates SHACL rules and functions used for entity resolution and dependencies inference. SHACL rules are used to create new relations in a KG based on other ones. Meanwhile, SHACL functions are used to perform calculations in SHACL rules. Both are executed by a software component called reasoner.

Listing 3: SHACL rule for State-based dependencies inference

```
dp:StateBasedDependency
  rdf:type sh:NodeShape;
  sh:targetClass dp:IoTDevice;
  sh:rule [
  rdf:type sh:SPARQLRule ;

  sh:construct '' '' ''
    /*Construct the dependency relationship (here is the
  state-based dependency) */
  CONSTRUCT {
     $this dp:hasStateDependencyTo ?device .
        }
/*Constraints to check before constructing the dependency
  relationship */
    /*Constraints are contextual relationships */
  WHERE {
  /*Contextual relationships required to infer the state-based
dependency*/
    ?device a dp:IoTDevice ;
          saref:hasState ?state .
    ?trigger a dp:Trigger
           dp:relatedTo ?state .
    $this core:offers ?service .
    ?service a saref:Service ;
           dp:allowsAction ?action .
    ?rule a eupont:Rule ;
          eupont:hasAction ?action ;
          eupont:hasTrigger ?trigger .
    }
     '' '' '' ;
   ] ;
.
```

Listing 4: SHACL function of strict string matching

```
dp:strictStringMatching
  a sh:SPARQLFunction ;
   /*Define the operators of the function */
  sh:parameter [
    sh:path dp:op1 ;
    sh:datatype xsd:string ;
    sh:description "The first operand" ;
  ] ;
  sh:parameter [
    sh:path dp:op2 ;
    sh:datatype xsd:string ;
    sh:description "The second operand" ;
  ] ;
    /*Define the output type*/
  sh:returnType xsd:double ;
   /*Define the function call */
sh:select """
SELECT
(<http://www.example.org/StrictStringFunction>($op1, $op2)
AS ?result)
    WHERE {
    }
    """ .
```

Listing 5: SHACL rule for Entity Resolution

```
dp:EntityResolution
  rdf:type sh:NodeShape ;
  sh:targetClass dp: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
  {
  {SELECT $this
  /*Calculate the similarity by calling SHACL functions */
(MAX(SIM(?Attribute($this),?Attribute(?device))) AS ?val)
WHERE{
/*Select the resolution attributs used in the similarity
  calculation */
OPTIONAL
{$this dp:hasResolutionAttribut ?Attribute($this) .}
$this orgIoT:source ?s .

OPTIONAL
{?device dp:hasResolutionAttribut ?Attribute(?device).}
?device orgIoT:source ?src .
FILTER (?device!=$this
&& ?s="DM" && ?src="OTHER")
 }
group by $this
}

OPTIONAL
{$this dp:hasResolutionAttribut ?Attribute($this) .}
$this orgIoT:source ?s .
OPTIONAL
{?device dp:hasResolutionAttribut ?Attribute(?device).}
?device orgIoT:source ?src .
Filter(?device!=$this
&& ?s="DM" && ?src="OTHER")
 && SIM(?Attribute($this),?Attribute(?device)) } } }
     '' '' '' ;
     ]  ;
.
```

E. Thing'in views

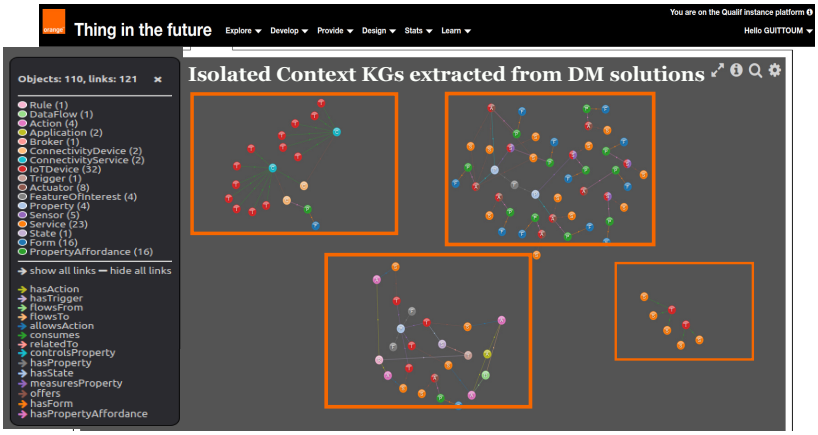


Figure 6: Context extraction

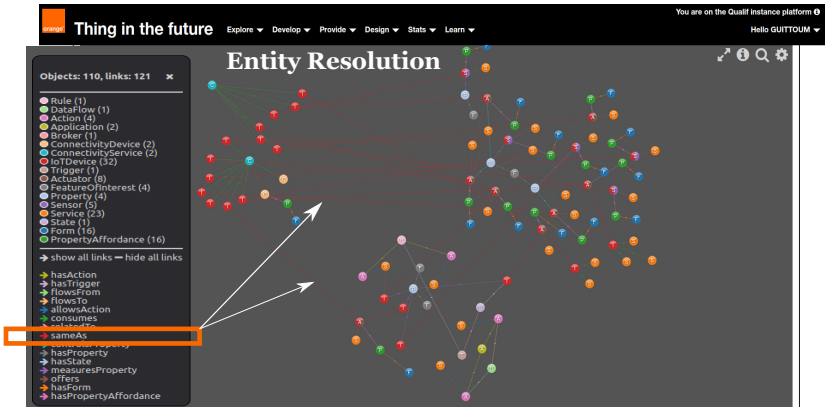


Figure 7: Entity resolution

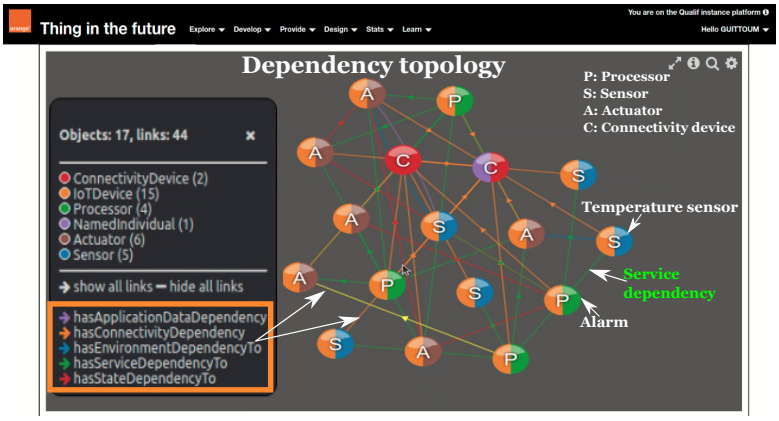


Figure 8: Dependency inference