Article

SemKoRe: Improving Machine Maintenance in Industrial IoT with Semantic Knowledge Graphs

HICHAM HOSSAYNI [1,*], IMRAN KHAN [1], MOHAMMAD AAZAM [2], AMIN TALEGHANI-ISFAHANI [1], NOEL CRESPI [3]

[1] Schneider Electric, France, (e-mail: {hicham.hossayni, imran2.khan, amin.taleghani}@se.com);
[2] University of Nottingham, Malaysia, (e-mail: aazam@ieee.org)
[3] Institut Polytechnique de Paris, IMT, Télécom-SudParis, 91011 Evry Cedex, France (e-mail: noel.crespi@it-sudparis.eu)
* Correspondence: Hicham Hossayni (e-mail: hicham.hossayni@se.com)

Version September 2, 2020 submitted to Appl. Sci.

Abstract: The recent focus on sustainability and improved efficiency requires innovative approaches in industrial automation. We present SemKoRe, a knowledge graph developed to improve machine maintenance in the industrial domain. SemKoRe is vendor-agnostic, it helps original equipment manufacturers (OEMs) to capture, share and exploit the failure knowledge generated by their customers machines located around the world. Based on our interactions with actual customers, it usually takes several hours to days to fix a machine-related issue. During this time, production stops and incurs cost in terms of lost production. SemKoRe significantly enhances the maintenance process by reducing the failure diagnostic time, and by centralizing machine maintenance knowledge fed by the experts and technicians around the world. We developed flexible architecture to cover our customers' varying needs, along with failure and machine domain ontologies. To demonstrate the feasibility of SemKoRe, a proof-of-concept is developed. SemKoRe gathers all failure related data in the knowledge graph, and shares it among all connected customers in order to easily solve future failures of the same type. SemKoRe received the approval of several substantial clients located in USA, UK, France, Germany, Italy and China, associated with various segments such as pharmaceutical, automotive, HVAC and food & beverage.

Keywords: Failure diagnostics, Industry 4.0, Industrial Internet of Things (IIoT), knowledge graph, machine maintenance, semantic web.

1. Introduction

Industrial Internet of Things (IIoT) has emerged as an enabler of the rapid integration of advanced technologies in the industrial world [1]. Factories are becoming fully connected and smart, thereby allowing manufacturers to improve process efficiency, sustainability, and safety while decreasing costs. Many industries are making heavy investments in smart manufacturing and production systems. In return, they expect optimal and sustainable production with minimum maintenance efforts. This makes maintenance one of the most important aspects of industrial process activities. Formerly considered as part of general enterprise costs, it has become a real source of data and critical for business continuity & performance[2].

Systems such as computerized maintenance management system (CMMS), manufacturing execution system (MES) and enterprise resource planning (ERP) are used to perform maintenance activities in several industries [3]. These systems provide features such as predictive and preventive maintenance, maintenance planning, scheduling, execution, monitoring and traceability. However, these systems have two inherent drawbacks. *First*, they are intended to optimize the maintenance process at given location (a factory or a site). This means that two different factories (or sites) cannot share the details of a specific machine's maintenance operations without a human expert in the loop. Such sharing is useful when machines share the same characteristics and perform the same operations regardless of their locations. While cloud-based offers are a potential remedy, several customers still do not want to have their data on multiple vendor cloud platforms.

The *second* drawback of these systems is that they are not interoperable at the semantic level. Schneider Electric works with several Original Equipment Manufacturers (OEMs) who design, build and ship machines for their customers around the world. In our experience, there is no easy way to align the data coming out of these maintenance systems and to thereby get a uniform understanding. This issue is complicated by the heterogeneity of these systems and the associated silos since each business segment and customer is unique and operates under different regulatory and geographic constraints. Despite these challenges, our customers are increasingly demanding better visibility about the performance of their assets, reductions in maintenance costs & downtime, improved productivity and more agility in their end-to-end processes.

In this paper, we present the early outcomes of our work, SemKoRe: how we use it to construct knowledge graphs of machine failures and exploit it to address various issues. SemKoRe is a vendor agnostic solution that uses formal, shared and explicit models to capture the details of machine domains, the failures of these machines and the applied repairing procedures. SemKoRe is designed to speed-up the maintenance process and to allow for quick recovery from failures. When a new machine is installed in a factory today, there is no existing knowledge of its failures. In any given factory, each failure is only discovered at its first occurrence. The maintenance process includes diagnostics to determine the reasons for a failure, its impact, and to define and apply the correct repair procedures. This process is repeated at different locations for the same machines having the same failures. In reality, failure details are usually captured manually, e.g., using spreadsheets (e.g. Excel). This approach is not fault proof as each person filling out the sheet cannot be expected to provide all the required information and even if that information is given, there will be semantic mismatch, e.g., one person describes issue as "abnormal rotation speed" while another person describes the same issue as "irregular spinning rate". Both mean the same but use different semantics (we provide more details on it based on interactions with our actual customers in Section 2).

SemKoRe helps to avoid this semantic mismatch, and captures all the machine, failure and maintenance data as a knowledge graph, allowing several actors to benefit (Section 4 and Section 5). For example, *Operators & Technicians* can benefit from the knowledge provided by other operators at different sites to address their issues. This will also reduce the risk of mismanipulation of machines by incompetent operators, which is a considerable industrial threat in reality [4]. *OEM Machine Builders* can improve the next generations of the machines they build, thanks to the knowledge captured in SemKoRe that helps them to know why certain machines have higher failure rates. *Analytic teams* can improve their work, as SemKoRe will provide a global view of machine failures and the background of a variety of contexts.

We implement our SemKoRe system using GraphDB (cloud/gateway triplestore), IBM Node-Red (flow-based development tool), Microsoft Azure (for cloud service), Azure IoT Hub (for IoT connectivity), and Docker (to package SemKoRe services) (Section 6). To support the needs of different actors, SemKoRe is developed using the semantic web and ontologies [5]. This approach helps to accommodate future requirements and to provide a clear separation of concerns between the application needs and the domain knowledge which in this case is machine domain and failure domain knowledge. We adopted a distributed architecture in which knowledge collection is performed on the edge layer. The collected knowledge is shared with other actors and machines through a cloud-based instance.

We elaborate the lessons learned in Section 7.1, and offer future research directions in Section 7.2 and conclude in Section 7.3.

2. Motivating Scenario & Requirements

In this section, we talk about some key motivating scenarios, with practical examples, that would help to envision the core idea of the problem domain. Then we present the set of customers requirements that guided us during the elaboration of our solution.

2.1. Motivating Scenario

The following scenario is based on our interactions with real customers who want to improve their existing maintenance process. Let us consider three actors: Bob *the machine operator*, Alice *the maintenance technician* and Joe *the OEM machine builder*. On a given day, Bob is working on factory floor operating several machines when suddenly one machine stops working. Bob spends some time to fix the issue himself but is unable to do so, since Bob's main job is to operate the machine. He might be able to fix small issues due to his experience but he is supposed to call a qualified technician for anything major. He then calls Alice to come to factory floor to check on the machine. When Alice checks the machine she finds that she is also not able to solve the issue so she calls the OEM or Schneider Electric service bureau, where a machine expert guides her through the repair process. Finally, Alice is able to fix the issue and the machine starts working.

The whole process took a long time and while Bob is now able to operate his machine, if the same issue occurs in a similar type of machine located in a different city the same process would likely be repeated because only Alice knows how to quickly solve this particular issue. However, if Alice can describe what she learned from the service bureau and share her experience with the technicians in other sites by using some appropriate mechanism, they could all benefit from this common knowledge.

Another beneficiary of this common knowledge is Joe. Today, when Joe gets reports about the issues with his machines from different customers, he has no easy way to get the finer details that can only come from the technicians like Alice. These details could be useful and help him to understand why some of his machines are facing particular issues. This can help him to improve the design & engineering process of his machines, especially in the case of hundreds or thousands of machines being used worldwide, the scale of problem and timely action in resolving the issue becomes hugely difficult. Another benefit is that using the insights from customer A, Joe can help customer B to quickly respond to machine issues while respecting of privacy and sensitive nature of the information, if both customers have the same type of machines. The importance of the quick resilience after failure aspect is discussed in details by Alcaraz et al. in [6].

2.2. Requirements

Based on the motivating scenario described above, we now present the following set of requirements. The *first* requirement is that the proposed solution should make it easy to capture and share knowledge among various actors. The *second* requirement is that the proposed solution should be usable both on cloud (public or private) and on-premise systems. Indeed many customers are willing to connect their machines and factories to the cloud, while others choose to fully isolate their factories in order to protect their industrial property and to keep their private data locally. The *third* requirement is that the solution should have built-in mechanism to protect the sensitive information about the processes and the business. During our interactions with customers, this requirement came up as the *make or break* point for them. The *fourth* requirement is that the solution should support root cause analysis and make it easy to identify the component(s) that cause the failures. The *fifth* requirement is that the solution should be platform-independent and thus should not depend on any particular hardware or software platform. The *sixth and last* requirement is that the proposed solution should be open and extensible to cover the current as well as future needs. These requirements are also thought to avoid introducing security issues or affecting the machines' performances in the customers sites [7] .

3. Related Works

E. Kharlamov et al. present one of the earlier works, from a major industrial company on capturing industrial information models using W3C standards [8]. This work proposes an application front-end to allow non-semantic experts to develop ontologies. The front-end is a modified Web-Protegé [9], that hides the complexity associated with the desktop Protegé version. The work highlights the benefits of involving domain experts to capture the domain knowledge and to create different services using it. However, the solution does not cover our main requirements.

N. Zaini et al. [10] propose a generic online tool for building collaborative ontology without prior deep knowledge of the domain. An initial ontology is built, populated and enriched by multiple participants in a collaborated manner. The goal is to simplify the ontology based modeling of domain knowledge for the users without ontology expertise. However, the ontology concepts are not sufficiently abstracted, as these concepts are simply renamed. This means that despite simplification, substantial semantic expertise is needed to make the necessary modifications to the ontology. Another important missing element is that there are no checks to ensure consistency which can be an issue in a multiple user environment.

There is a large pool of work on industrial maintenance. D. L. Nunez et al. [11] created a taxonomy of the Prognostics and Health Management in manufacturing. They propose a formal ontology for failure prognostics based on industrial ISO standards for failure mode analysis, failure diagnostics & prognostics (e.g. ISO 13372, ISO 13379, ISO 13381 and others). Failure knowledge is described in ontologies from ISO standards. Semantic Web Rule Language (SWRL) [12] is used to define rules in order to generate warning messages in case of abnormal states. However no approach is described to share the acquired failure knowledge among different users.

In [13], M. Melik-Merkumians et al. used ontologies for fault diagnosis for industrial control applications. They utilized reasoning capability to check the model consistency over the time and to raise early-alarms for critical failures.

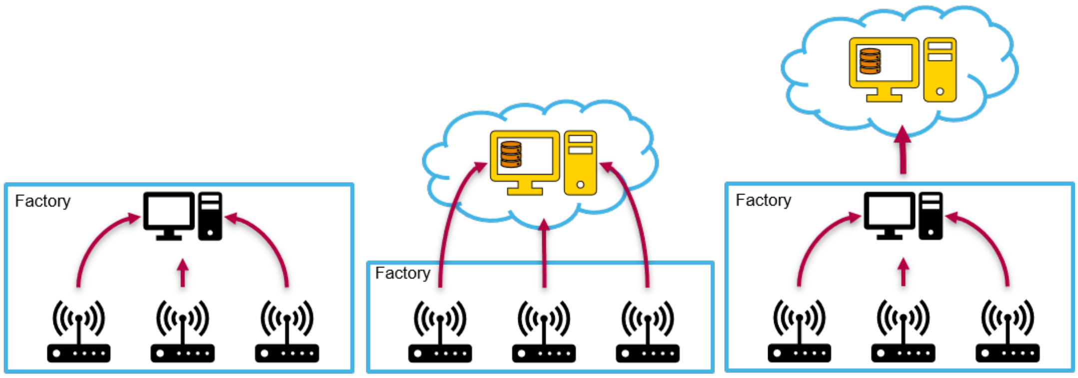
L. Palacios et al. [14] propose an ontology based support for fault diagnosis for aircraft maintenance operations. An aircraft maintenance ontology is modeled and fed by the (manual) alignment of several existing ontologies related to the avionics domain to discover the relations between the causes and failure symptoms, explain the failures and any unscheduled maintenance requirements as well as the possible procedures that can be applied to each situation.

An ontology-based approach is adopted by H. Peng et al. [15] for the fault diagnostics of conveyors. The knowledge about fault symptoms, fault causes and fault solutions was modeled with multiple ontologies. The resulted ontologies were mapped together based on a mathematical formulation of conveyor fault diagnostics. Some reasoning rules were defined in order to infer additional relations between faults, symptoms and potential causes.

In [16], R. Chen et al. used ontologies to model the knowledge of fault diagnosis for rotating machines. Their proposed ontology model describes fault diagnosis knowledge considering the vibration characteristics as main fault factor. The model's reasoning capability is considered by defining some SWRL rules for fault diagnostics.

F. Xu, et al. [17] also relied on ontologies to design a loader fault diagnosis system. It aims to help users find the fault causes, locations and fault maintenance measures of loaders in a reasonable amount of time. Ontology is used to model the loader information and describe the relative failures. This work uses the condition based reasoning (CBR) method to diagnose loader faults by finding similar corresponding situations in the past. When no corresponding case is found, CBR fails and the (SWRL-based) rule based reasoning (RBR) approach is proposed for fault diagnosis.

In [18], S. Wan, et al. developed a Collaborative Maintenance System Planning that allows many stakeholders to collaborate to ensure maintenance process quality. An ontology-based approach is adopted to model a large field of knowledge: the machine domain model, failure knowledge and stakeholders knowledge are modeled together to ensure the interoperability between their systems, the maintenance planning and Resources and Constraints knowledge. However, this centralized solution



(a) Local SemKoRe Architecture **(b)** Cloud based architecture **(c)** Hybrid architecture

Figure 1. SemKoRe Architectures; IoT gateway connected to a real machine: Local server

focuses mainly on preventive maintenance planning. In addition, the managed failure knowledge is relatively basic and does not consider root causes or symptoms.

All the works mentioned above use ontologies to model machine data models and failure knowledge. However, none of these works satisfy all of the requirements that we identified from our motivating scenario. These works developed various ontology models and some exhaustively described potential failures and their characteristics. However, to the best of our knowledge no machine failure ontology is available for reuse or for extension. Also, neither of our two major requirements, i.e., knowledge sharing and data confidentiality have been considered.

Also, many Cloud-based solutions are proposed to enhance the maintenance process for different domains like smart grids, shop-floors, ... etc. Different aspects were analyzed: such as remote maintenance [19], fault detection [20], machines monitoring [19,21], preventive maintenance scheduling [22], predictive maintenance [23], or data confidentiality [24]. However, none of these studies considered sharing experiences or knowledge between different actors for maintenance purpose.

4. Architecture and Ontology Models

In this section, we discuss our contributions & our proposed architectures, along with the developed ontology models.

4.1. High-level Architecture

As mentioned before, our customers require different deployment options, and so we divided them into three categories and developed three architectures. *In the first category*, customers prefer to not connect their machines to the cloud and some even do not want to connect to the Internet due to the sensitive nature of their business, and to protect their data. The architecture proposed for this category is shown in Fig. 1a. *The second category* of customers opted for an entirely connected architecture, in which the machines/gateways in their factories are directly connected to the cloud. For this category, we proposed the architecture shown in Fig. 1b. *The third architecture* targets the customers who refuse to connect their machines to the cloud, but are ready to deploy a local on-premise server between the cloud and their machines. For this use case, we proposed a hybrid architecture Fig. 1c, where most of the collected data stays in the local server, and only an anonymized part of the data is transmitted to the cloud.

In all these cases, each machine is connected to an industrial IoT gateway, e.g., Modicon M262 [1] to collect the run-time data of the machine and the information provided by maintenance personnel and/or operators. Each gateway is connected to a central entity (either a Local SemKoRe or a SemKoRe

---

[1] https://www.se.com/ww/en/product-range/65771-modicon-m262/

Server) which collects the data from all the gateways and then aggregates and shares with the gateways connected to the same type of machine.

Though, the Achilles heel of our proposal remains the case of the first category of customers, i.e. who don't want to connect their factories to the cloud. The unique technical solution to share the maintenance knowledge consists of using physical supports (e.g. USB keys, CDs, ...) with human intervention. However, the internal business case analysis is still in progress. We believe that once the business case is finalized, the technical adaptation of the SemKoRe services will be trivial and will not require lot of efforts.

In this paper, we focus on the cloud-based architecture because it covers all the constraints and features of the other architectures. This architecture is also implemented in a proof-of-concept to demonstrate the feasibility (see Section 6).

4.2. Detailed Architecture

Figure 2 shows the detailed architecture of the SemKoRe.

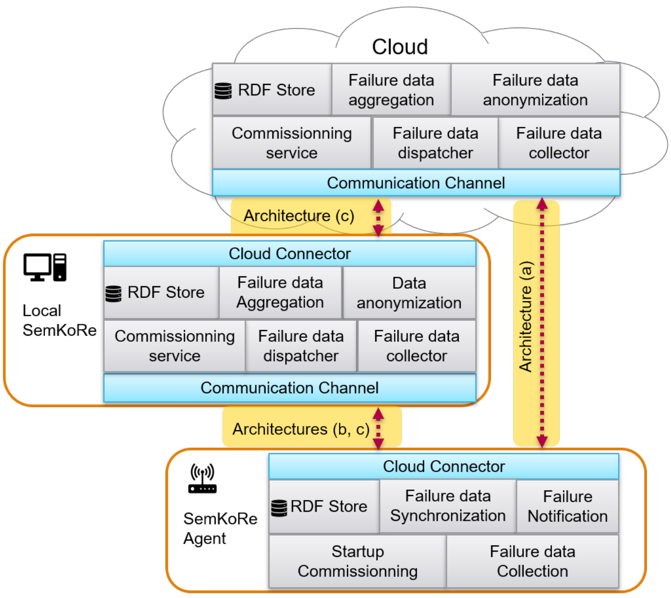


Figure 2. SemKoRe Detailed Architecture

The SemKoRe consists of three entities:

1. A SemKoRe Agent: Runs on industrial IoT gateways connected to the machines. It collects data when failures occur in the connected machines. According to the chosen architecture, the collected data is then shared with either the SemKoRe Server or Local SemKoRe. The SemKoRe Agent is designed to fit in all architectures (see Fig. 1).
2. A SemKoRe Server: Running on the cloud, it manages several failure data producers, i.e., Local SemKoRes or SemKoRe Agents. The collected failure data is validated by an expert and aggregated and afterwards shared with the SemKoRe Agents and/or Local SemKoRes.
3. Local SemKoRe: Lightweight instance of the SemKoRe Server deployed on a local server to manage the machines located in a site or factory. It collects data produced by the SemKoRe Server (if available) and the SemKoRe Agents on local gateways. The data aggregation is done locally, and only aggregated data is shared with the SemKoRe Server. The cloud then merges its aggregations with the Local SemKoRe aggregation, and pushes back the updates to the corresponding entities.

We use in the cloud a message broker to connect the machine gateways to the SemKoRe Server for bi-directional data sharing. We also developed a REST interface on the SemKoRe Server side to directly call remote services e.g., commissioning service.

4.3. Machine Failure Ontology Model

The first part of our work is collecting information on machine failures to be able to answer the following questions:

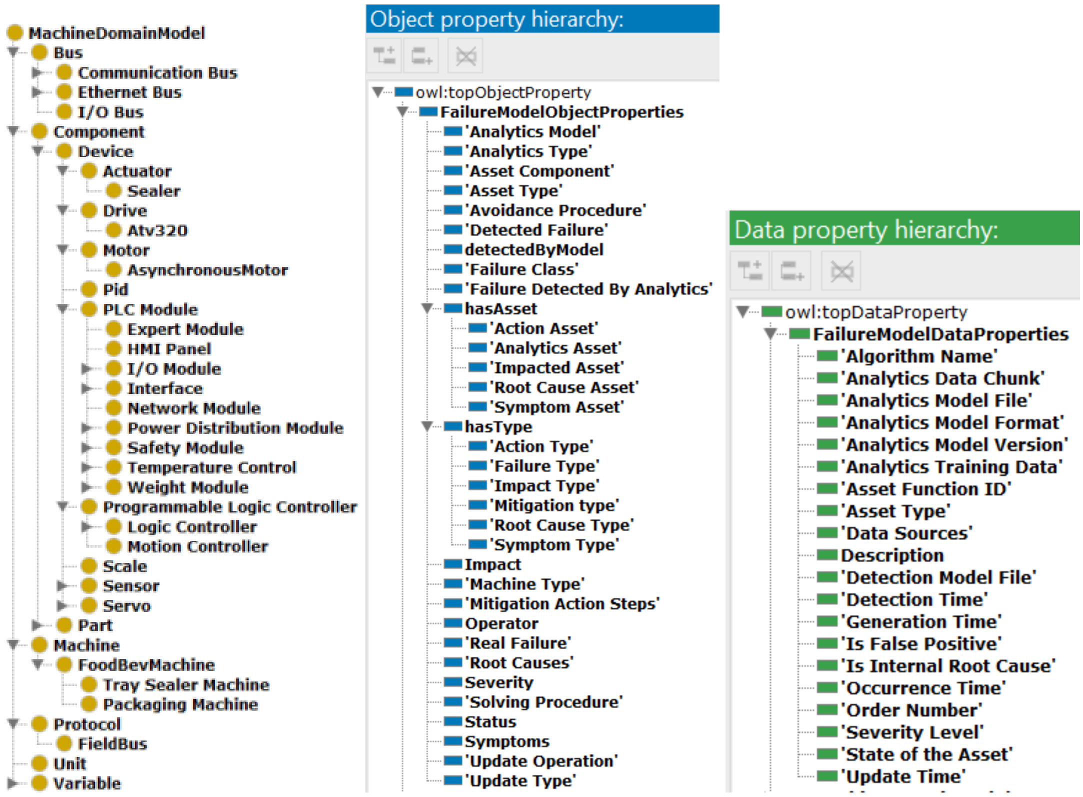
- What are the failure symptoms? Symptoms reflect the perceptible aspects of failures whether they are visual, sonic, odor or heat related.
- What is the impact of the failure? This may or may not be detected easily. Each failure impact is relative to a machine or to one of its components.
- What are the root causes of a particular failure? This question is difficult to answer, since it assumes prior knowledge about the cause-effect relations specific to each machine type. Answering this question requires the knowledge of a machine domain expert.
- After knowing the failure type and impact, how can we repair the machine?
- After knowing the root causes of a specific failure, is there a preventive maintenance procedure that can help us to avoid that failure?

To answer all of these questions, we defined the data model of the machine failures using Semantic Web standards because of the schema-less nature of RDF, RDFS, OWL and the explicit formalism supported by these languages. The failure ontology is created by interacting with the machine builders and by using the initial set of requirements described in section 2.2. It acts as a common data model and will be enriched with new concepts by domain experts over time. Progressively, our design, engineering, configuration and maintenance tools will use this ontology to create the knowledge about the failures and allow us to develop different services over it. SemKoRe targets the industry automation business, in which the failure knowledge can be significantly different from one customer to another. The concept uses a flat ontology model, containing only the most common general concepts required for current needs. The subsequent specialization concepts will be easily and naturally added by domain experts as the knowledge collection progresses.

To develop the machine failure ontology presented in Fig. 3, we adopted the Seven-step method, developed by the Medical Information Center of Stanford University [25]. Its seven steps are as follows:

1. **Determine domain and scope**: This work focuses on industrial machine failures.
2. **Consider reusing existing ontologies**: No existing machine domain or machine failure ontology was found for reuse, therefore we developed both for this work. Regarding upper-level ontologies, there are several candidates like basic formal ontology (BFO), ISO-15926, Gist, and suggested upper merged ontology (SUMO) but we still need to finalize one.
3. **List important terms in the ontology**: After interactions with the machine domain experts, the following important terms were identified *Failure*, *Symptom*, *Impact*, *Root cause*, *Solving Procedure*, among others.
4. **Define classes and class hierarchy**: Several classes were created including the important terms listed above. However, since no specialization concept will be introduced, the ontology is flat. Only the classes relative to types (e.g. *Failure Type, Symptom Type*) are grouped as sub-classes of the *Types* class.
5. **Define object properties of classes**: We defined a set of object properties that link all the defined classes together. For example, the property *hasSymptoms* links a Symptom to a specific Failure. The complete list is illustrated in Fig. 4b
6. **Define data properties of classes**: We also defined several of the data properties of classes, with cardinality and type constraints, as shown in Fig. 4c.
7. **Create instances and check exceptions**: Instances of SemKoRe ontology are divided into two parts. The first part, defined by experts during the design time, concerns the generic concepts that will be used for most industrial use cases, such as Severity level (Catastrophic, Critical, Moderate, Low). The second type of instances concern the data provided by the users during the runtime of the SemKoRe. Users instances include details about the failures and symptoms. To check for exceptions, we used Pellet reasoner [26] to verify the correctness of our ontology model.

Figure 3. Failure Ontology



(a) Machine Model Ontology **(b)** SemKoRe object properties **(c)** SemKoRe data properties

Figure 4. SemKoRe ontology design

In addition to the failure ontology, we also created a machine domain ontology (Figure 4a) to describe the machine components to satisfy our requirement to link failures to specific machine components when and where they occur, as simply knowing about a failure is not useful on its own. We also need to identify the components that caused a failure or that show failure symptoms, so that they can be identified as candidates for repair or inspections. For simplification, we only described two types of machines in our machine domain ontology, *a Tray Sealers Machine and a Packaging Machine*. Each machine type is composed of many components (*PLC, Drive, Actuator, Sensor, and others*), connected through different communication buses. To integrate both ontologies, we created OWL Class *FailureAsset*, to associate failures to the corresponding components in the machine domain ontology.

However, we faced another issue to identify the exact component of a machine that has failed or is impacted by the failure. For example, consider that a machine has two Servo Drives of the same type. These Servo Drives are described in the machine domain ontology as two instances (*SDA and SDB)* of the *"ServoDrive"* class and are associated with the instance of a machine. When a failure occurs in SDA, we should be able to identify it through ontology. Such detailed identification is especially useful when the failure knowledge is shared with the other sites using the same machine type, as it will help them to recognize the exact component responsible or impacted by the failure.

To address this issue, each component in the machine domain ontology has a unique number "FunctionID", to distinguish its role in the machine compared to other components of the same type.

We used both desktop Protégé [27] and Web-Protégé [9] to create our ontologies. The latter allowed us to include machine domain experts in the ontology development process and to gather and organize their feedback.

It is important to mention that our main focus in this work has been on the validation of the idea to the OEMs, that formalized knowledge about machines and their failures can be useful for quick resolution of failures and to improve overall equipment effectiveness (OEE). The ontology, in its current form, will be extended to cover the needs of the OEMs. To guarantee the ontology generality, a potential extension basis could be the adoption of taxonomy of ISO standards for failure mode analysis (similarly to [11]). Relevant actors can be involved to extend these ontologies with new concepts and relationships by using appropriate tools like the ones mentioned in section 7.1.

5. SemKoRe Process

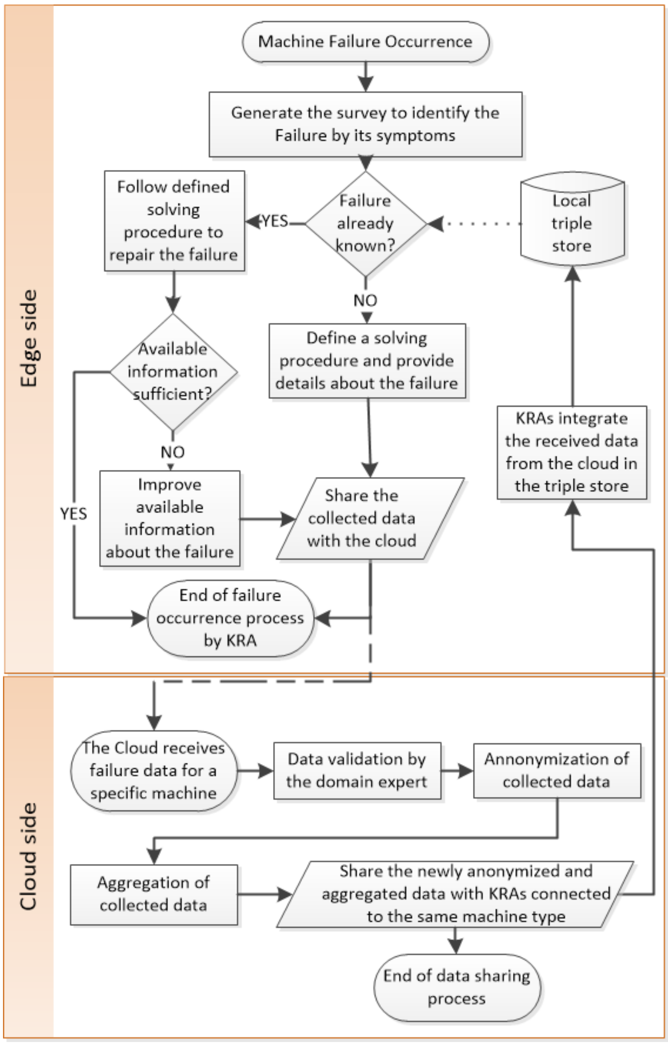


Figure 5. SemKoRe Process

Figure 5 shows the SemKoRe process of failure data collection and sharing. The process is distributed on two layers: on the edge with the SemKoRe Agent, and on the cloud with the SemKoRe Server.

The failure data collection starts when a machine failure occurs. The failure information collection service generates the human machine interface (HMI) for the user (Bob or Alice) to offer the details of the failure Fig. 6. Through the survey, we first try to know if the failure has really occurred or it was only a false positive case triggered by some failure detection service. Then the user is asked to provide details about the symptoms of the failure by selecting known symptoms or by creating new ones, when necessary.

The user checks if the identified failure is already known by the SemKoRe before providing additional details. If the failure already exists, the user follows the instructions to repair the machine. Otherwise, the failure will be documented by Alice or by machine domain experts as shown in Fig. 5. Sometimes, the impacts of a failure may differ from one machine to another. So, existing repair

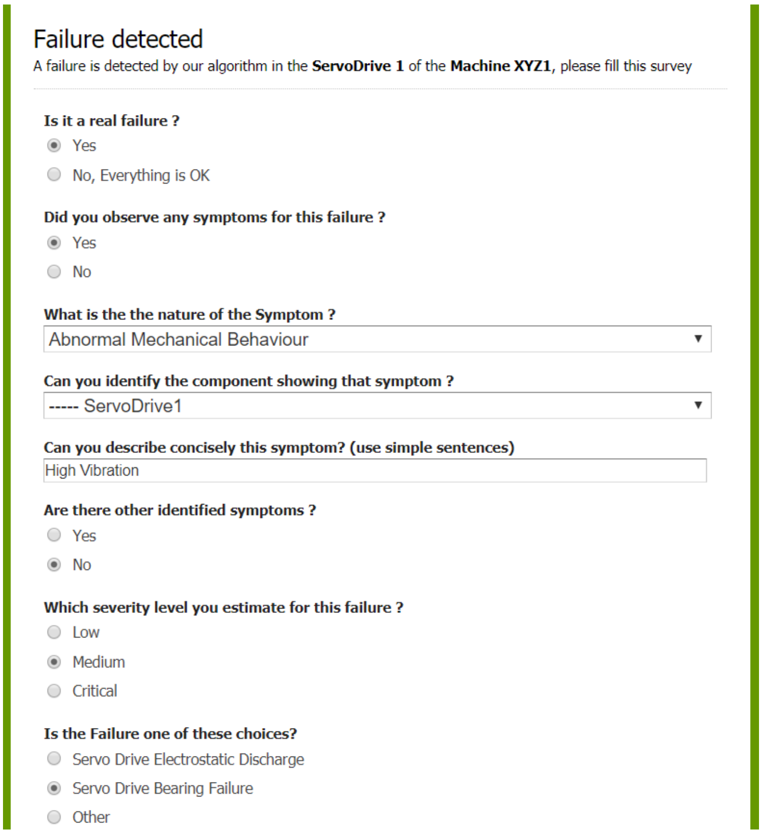


Figure 6. Screenshot of the Failure Survey

procedures might be adapted or new procedures might be created to repair the machine. This process ends, in the edge level, by sharing the collected data with the SemKoRe Server in the cloud.

When the SemKoRe Server receives failure data from a SemKoRe Agent instance, the data must be validated by a machine domain expert before it is integrated into the SemKoRe Knowledge Graph. After the validation process, data is anonymized to protect the data & customer privacy and business sensitive information of the customers (see section 6.3). The anonymized data is then aggregated (see section 6.4) in order to get insights about the occurrence frequency of failures, their impacts and the most adapted/used repairing procedures .

The aggregated data is then shared with the SemKoRe Agents instances connected to the same machine type. On the other side, each SemKoRe Agent instance integrates the data it receives from the cloud into the local triplestores (Graph databases).

6. SemKoRe Implementation

In this section, we describe the services implemented for the SemKoRe. It must be noted that the services deployed on Local SemKoRes are the same as the ones deployed on a SemKoRe Server. As stated previously, we only focus on the architecture where gateways are directly connected to the cloud (since its an overarching architecture and covers all the scenarios we described in Fig. 1).

6.1. Startup Commissioning

Knowing that our Failure ontology will evolve, it is only deployed on the SemKoRe Server. On-premises, the gateways must run the commissioning service in order to get the latest failure model corresponding to the type of the connected machine. The startup commissioning service retrieves two types of information from the SemKoRe Server:

- The machine failure T-Box, containing the concepts defined in the failure ontology; and,

- A-Box data, containing the instances of the T-Box concepts. Knowing that the SemKoRe Server manages data relative to several types of machines, the A-Box retrieved by a gateway contains only the information relative to the machines connected to it.

The startup commissioning service sends a request to the SemKoRe Server with the identity and the type of the connected machines. The SemKoRe Server runs then a Construct SPARQL query to create a sub-graph containing all the data (T-Box and the A-Box) related to the provided machine type. The resulting sub-graph is sent back to the gateway.

6.2. Failure Data Collection

This service runs exclusively in the SemKoRe Agent and is used during or after the maintenance phase. It consists of the following two parts:

6.2.1. Failure Survey

This part collects the failure information using an ordered set of predefined questions. It facilitates the collection of perceptible symptoms of the failures as well as the reporting of new symptoms and failures. During our interactions with machine domain experts, we found that the content of the survey is strongly correlated with the machine types and the kinds of failures they encounter. Our on-field interactions with the operators, technicians and experts highlighted the importance of an intuitive user interface.

6.2.2. Failure Ontology Instantiation

We use the model driven interfaces (MDI) to dynamically generate the user interfaces using the Failure ontology. This procedure has two advantages: *One*, the UIs allow instantiation of the Failure ontology by non-technical users without any knowledge about the Semantic Web or ontologies; and *Two*, the UIs enforce the constraints defined in the ontology model and ensure that all the inputs are valid. These UIs rely on the annotations defined in the ontology such as, *@rdfs:label* and *@rdfs:comment*. The former is used as a human readable label of the input fields shown to the user, while the latter is displayed to explain the nature of the field and the expected input. We defined an additional annotation *@semkore:hidden* to hide a field on the UI in case it should be defined automatically or exclusively by an expert.

The ontology model incorporates two types of constraints: value and cardinality constraints, as described below.

Value Constraints

In the W3C OWL reference [28], a value constraint is used to enforce restrictions on the range of a property when applied to a particular class description. Value constraints can be applied to data properties, for which the value is a data literal, and object properties, for which the value is an individual. We handle each type of property differently:

- Data properties: Users can input a value in the text box which will be validated to make sure that the data type is correct as per the defined constraints; and
- Object properties: A select box is provided with the list of all possible values. For example, for the object property ***Machine*** *hasComponent AllValuesFrom* ***Component***, will lead to a select box with the list of all available ***Component*** instances. With this approach, the probability of getting an invalid input is eliminated altogether.

Only the *owl:someValuesFrom* constraint was managed differently from the W3C standard [28] definition, as it defines a constraint that is applied to *at least* one value, which means that the property could have other values without any restrictions. Since our UIs are targeting non-expert users, and to guarantee the consistency of our model, we considered *owl:someValuesFrom* as being similar to the *owl:allValuesFrom* constraint in our implementation.

Cardinality Constraints

In the W3C OWL reference [28], a cardinality constraint restricts the (*min*, *max*, or *extact*) number of values a data or object property can have. To satisfy the cardinality, single/multiple input fields are generated for each property, allowing to the user to provide the correct number of values for each property.

6.3. Anonymization Service

Privacy protection is a very important concern for our customers. They do not want to share any machine and process-related data in a way that could potentially expose sensitive business information. To address this concern, we implemented a simple service (described below), in the SemKoRe Server in the cloud, to anonymize the collected data before sharing it with other sites or locations.

When a failure occurs, the gateway creates an instance of "Failure Occurrence Class", containing information about the failure, e.g., symptoms, impact, root causes if known, and the failure context, which includes the machine ID, its location, timestamp when failure occurred, and a snapshot of the current parameters. The whole process consists of three steps:

1. The SemKoRe Server removes the machine ID, location, and owner-related information and does not share this information.
2. A human expert reviews and validates all of the failure information before integrating it into the SemKoRe Knowledge Graph. This additional check helps to protect sensitive business information.
3. Finally, all the validated failure information is aggregated and then shared with the connected gateways. This process ensures that no one can deduce the origin of the data, the failure location or the ownership details.

This service is a subject for future SemKoRe versions. The goal is to automate this process so that little to no human involvement is required.

6.4. Failure Data Aggregation

Hosted in SemKoRe Server, this service and aggregates failure data collected from different gateways in order to produce deep insights on the machine failures and their characteristics including symptoms, impacts, and root causes. Once the aggregation is done, the data is shared by the SemKoRe Server with the connected gateways that need it. We have defined a list of simple aggregations that are applied to the failure data:

1. For each machine type, get the list of all failures and their frequency;
2. For each failure, compute the list of all possible symptoms with the frequency of each symptom;
3. For each failure, compute the list of all possible impacts with the frequency of each impact;
4. For each failure, compute the list of all possible root causes with the frequency of each root cause; and
5. For each failure, get the list of solutions with the number of times each solution was successfully used to repair that failure.

For each of these aggregations, a dedicated SPARQL query is executed and the results are injected into the Knowledge Graph. The aggregation service executes after every new data collection to keep the Knowledge Graph up-to-date.

6.5. Failure Data Sharing

The failure data sharing is done through the SemKoRe Server's message broker. Each gateway subscribes to the topic "*.../failure_updates/{machine_type}*", where *{machine_type}* is the type of machine to which the gateway is connected. When failure data is sent to the SemKoRe server (through the REST interface), it is validated, anonymized, aggregated and then published on the message topic corresponding to the right machine type. The gateways receiving this data will simply update the locally stored graph data.

6.6. Implementation Details

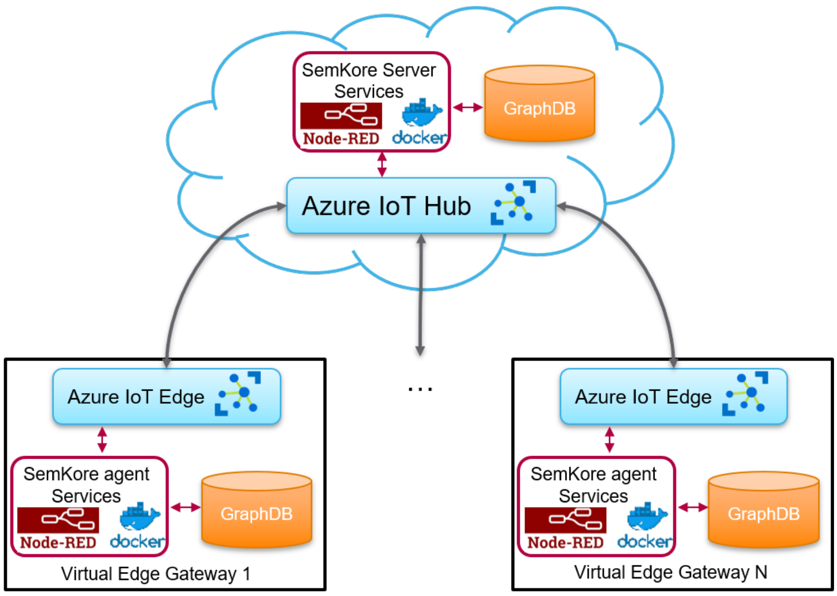


Figure 7. SemKoRe implementation setup. Simulated virtual Edge Gateways with SemKoRe Agents managed by a SemKoRe Server instance hosted on Microsoft Azure cloud.

To demonstrate the feasibility of the SemKoRe approach, we developed a proof-of-concept (Fig. 7) using the following technologies:

- GraphDB: We used GraphDB [29] as triplestore in the cloud and in the gateway. It provides high performance and scalability in addition to the reasoning capabilities.
- Node-Red : Node-Red was used to develop all above-mentioned services for the SemKoRe Server and the SemKoRe Agent. Node-Red is a flow-based development tool for visual programming originally developed by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things [30]. Node-Red is gaining popularity for rapid application development in Schneider's Industrial Automation business.
- Microsoft Azure: The SemKoRe Server services are hosted on Microsoft Azure, which provides high-performance cloud services. The server uses Azure IoT Hub [31] to connect the IoT devices to the cloud using several communication protocols, including the MQTT messaging protocol [32], which we used to simplify the data flow transmission between our SemKoRe agent and server services. For the edge, the Azure IoT Edge service was used to easily connect the edge gateways to the cloud via Azure IoT Hub as shown in Fig. 7.
- Docker: This is a popular container-based virtualization [33] tool. Docker supports many operating systems and hardware architectures, and allows self contained applications to be packaged and executed with a high level of portability and reproducible results [34]. We used Docker to package all of the services of SemKoRe Server and SemKoRe Agent. In reality SemaKoRe agent will reside with many other components. Docker allows us to have the flexibility of deploying different components easily and manage/extend them without impacting others.

As we are still in early stage, we were not able to implement the SemKoRe in real conditions with SemKoRe Agent running on Industrial Gateways connected to real machines. The main obstacle was that the current hardware available for this work had ARM architecture and so it would require a significant effort to port the triplestore and other software components. That effort was beyond the scope of our work. However, after the successful PoC, the business team decided to use an industrial PC as a gateway with enough RAM (8GB), processing (Intel Atom) and storage (64GB) capability. The next iteration of this work will use this industrial PC when it is ready for commercialization. In the meantime, we evaluated our implementation by simulating many virtual SemKoRe Agent instances on a PC equipped with an Intel(R) Core(TM) i7-7820HQ processor, and 16Gb of RAM. We used a single GraphDB server to manage separate triplestores for all the SemKoRe Agent instances. We randomly defined a set of three machine types *{Packaging Machine, Palletizing Machine, Pasteurization Machine}* that we associated to the active SemKoRe Agents. Each machine type was associated with at least two SemKoRe Agent instances which were then connected to the cloud SemKoRe Server. Our current implementation choices will facilitate an easy transition to industrial PCs in future.

We generated random machine failure data by defining a set of potential failures for each machine type. Each failure was then associated to a set of potential characteristics: symptoms, impacts, root causes and solving procedures. For each machine failure occurrence, we randomly picked one of the potential failures of the machine, and then picked a random number of the failure characteristics from the predefined sets.

We were able to demonstrate that the failure data was collected by each SemKoRe Agent and successfully shared with the SemKoRe Server. The data was aggregated and shared back with the SemKoRe Agent instances connected to the same machine type.

7. Conclusions and Future Work

7.1. Learned Lessons

Conducting this study helped us to learn several lessons. The *first* lesson is that the use of semantic web technologies to solve complex industrial problems is still a largely unexplored area. Even today most of the solutions on the market focus on the enterprise and IT side than on the operational side of large industries. This means that there are mature solutions that use semantic web technologies to bridge siloed enterprise data in RDBMS and unstructured data like documents but there is no mature solution that can do the same for data described in operation technology protocols e.g. OPC-UA [35].

The *second* lesson is that technologies such as triplestore are not easily adoptable to typical industrial use cases. Almost all triplestores are focused on big data and huge numbers of triples but, as our work demonstrates, there are several use cases where an efficient solution is needed for typical industrial gateways. Industrial PCs are an option but they are expensive and can only be used by large companies whereas small devices have a very large user base. While machine failures are reality, they do not occur every minute, and so there is no need to use a complex solution that supports billions of triples. Outside the vendor space, the open source community has some options like RedStore [36] but most are not in active development.

The *third* lesson is that the development of industrial grade ontologies is still a herculean task and the existing tool set continues to act as a barrier to entry. In our experience, experts want to formalize their domain knowledge but they have no motivation to learn complex tools such as Protege that do not support collaborative ontology development. WebProtege is a possibility but lacks query, visualization and documentation capabilities. New efforts such as Modom.io [37] and Zazuko [38] take a more simplified approach for non-experts to create ontologies but they are still works in progress.

Regarding ontology governance, there is no standard framework that can be applied to design and develop modular ontologies on an industrial scale. The evolution of ontologies is another area where no clear recommendations and no industrial tools are available to manage the required documentation, evaluation, release and versioning. While some academic works such as [39] exist, they are not mature and often not easy to deploy and use in industrial settings. The Semantic Web community as a whole needs to address these points and improve the developer experience in order to mainstream these useful technologies.

7.2. Future Potential

We have also identified several avenues as the future potential to continue this work. They are mentioned here without any order of priority.

7.2.1. Machine Learning for Data Anonymization

The *first* item is to explore the use of machine learning for data anonymization services. In this work, we used a simple approach with validation by a human expert. But a far more efficient approach would be to investigate the use of artificial intelligence and machine learning to anonymize data based on several contexts.The state-of-the-art anonymization techniques achieve good precision scores (up to 98%), which will make it unnecessary to involve humans. One might consider to collect anonymous data from the beginning to avoid the anonymization overhead and the complexity. However, with this approach we will miss important data that might be useful for things like audit (machine id, maintenance history, configuration) and other applications.

7.2.2. GraphDB Multi-Tenant Support

The *second* item is that currently, one GraphDB tenant is used in the cloud to collect the data of all customers, which could become an issue for scalability and data privacy. A potential solution may be to have separate GraphDB tenants for each customer and then create a common GraphDB instance to collect anonymized and aggregated data from the other customers' instances. Managing these tenants and synchronizing them will be big challenges.

7.2.3. Lightweight Triplestores

The *third* area would be to work on lightweight triplestores for small industrial devices. Many triplestores for embedded/small platforms exist. Most of them are based on the Redland RDF Libraries [40], and are using SQlite as backend storage (e.g. RedStore [36]). However, all these solutions lack reasoning engines and do not support SWRL rules.

7.2.4. Knowledge Graphs Synchronization

Ensuring the knowledge synchronization between the SemKoRe Server and SemKoRe Agent is the *fourth* area. As mentioned before, not all customers are keen to have a cloud connection or can have always-on connection. Therefore, it is necessary to define a synchronization process to ensure that there is no inconsistent knowledge.

7.2.5. UI Enhancement

The *fifth* item of future work is that today UI interfaces are used to report machine failures through manual input from persons like Bob or Alice. This process can be enhanced by using AI/ML algorithms that observe the symptoms and prefill the UI form with accurate details. This can be further extended to automatically fetch the repair instructions from a SemKoRe graph before failures occur.

7.2.6. Ontology extension by non-expert users

In this work, we target a large set of customers from various domains and with different needs. We are not expected to create or to modify ontology for each and every customer. Therefore, the *sixth* item is to develop a framework along with a tool suite and set of services for non-experts to allow them to create and extend their ontology models. We will also need to address more advanced topics like ontology matching, alignment, and conflict resolution to ensure consistency.

7.2.7. Use of upper-level ontologies

Another future item to consider is the use of upper-level ontologies as a basis of the failure and the machine domain ontologies. There is a separate on-going work to decide on the right ontology to provide maximum data integration for the future. For example, today the discussion revolves around using either BFO[2] or ISO15926[3] or Gist[4] as upper-level ontology. Some customers may also be interested in using domain ontologies like SSN[5]. Our view on this point is that once a decision is made, our current ontology can be easily refactored.

7.3. Conclusion

In this paper, we proposed a knowledge graph-based approach, SemKoRe, to enhance the maintenance process for the customers of Machine Builder OEMs. The idea consists of collecting machine failure data generated by different machines owned by many customers in different locations and in different business segments. The SemKoRe approach helps reduce the maintenance costs by sharing maintenance experiences between OEM customers. Based on this early work, our customers showed an interest in using the SemKoRe approach to enhance their industrial maintenance processes. Also, by using the SemKoRe approach, the overall machine building process can be optimized. The machine design phase can benefit from the maintenance feedback to identify any weaknesses of a machine and can improve its design. Also, the collected statistics will allow the performance comparison of a particular machine working in different locations and contexts. Thus, additional services and recommendations can be proposed to the customers in order to optimize their manufacturing process. Some customers also feel that our approach can help them to build Digital Twins to monitor the performance and efficiency of their machines. As mentioned in the Future Potential section, we plan to investigate on several work items to improve the features of SemKoRe.

Author Contributions: Conceptualization, H.H., A.T.; methodology, H.H., I.K.; software, H.H.; validation, H.H., I.K and A.T.; investigation: H.H.; writing–original draft preparation, H.H., I.K., M.A.; writing–review and editing, I.K., M.A., N.C.; supervision: I.K., A.T., N.C.; project administration, A.T., I.K.. All authors have read and agreed to the published version of the manuscript.

References

1. Aazam, M., Zeadally, S., & Harras, K. A. (2018). Deploying fog computing in industrial internet of things and industry 4.0. IEEE Transactions on Industrial Informatics, 14(10), 4674-4682.
2. Cachada, A., Barbosa, J., Leitão, P., Gcraldcs, C.A., Deusdado, L., Costa, J., Teixeira, C., Teixeira, J., Moreira, A.H., Miguel, P., Romero, L. (2018). Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture. 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 1, 139-146.
3. Swanson, L. (2003). An information-processing model of maintenance management. Int. Journal of Production Economics, 83(1), 45-64.
4. Lopez, J., Alcaraz, C., & Roman, R. (2013). Smart control of operational threats in control substations. Computers & Security, 38, 14-27.
5. Berners-Lee, T., Hendler, J., Lassila, O. (2001). "The semantic web." Scientific american 284.5 : 34-43.
6. Alcaraz, C., Wolthusen, S. (2014, March). Recovery of structural controllability for control systems. In International Conference on Critical Infrastructure Protection (pp. 47-63). Springer, Berlin, Heidelberg.
7. Alcaraz, C., Lopez, J. (2012). Analysis of requirements for critical control systems. International journal of critical infrastructure protection, 5(3-4), 137-145.
8. Kharlamov, E., Grau, B.C., Jiménez-Ruiz, E., Lamparter, S., Mehdi, G., Ringsquandl, M., Nenov, Y., Grimm, S., Roshchin, M., Horrocks, I. (2016). Capturing Industrial Information Models with Ontologies and Constraints. International Semantic Web Conference.
9. Horridge, M., Gonçalves, R.S., Nyulas, C., Musen, M.A. (2019). WebProtege: A Cloud-Based Ontology Editor. Companion Proceedings of The 2019 World Wide Web Conference.

---

[2] https://basic-formal-ontology.org/
[3] https://15926.org/home/
[4] https://www.semanticarts.com/gist/
[5] https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

10. N. Zaini and H. Omar, “An online system to support collaborative knowledge acquisition for ontology development,” ICCAIE 2011 - 2011 IEEE Conf. Comput. Appl. Ind. Electron., no. Iccaie, pp. 543–548, 2011.
11. D. L. Nuñez, M. Borsato, “An ontology-based model for prognostics and health management of machines”. Journal of Industrial Information Integration. 6, 33–46 (2017).
12. Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M. (2004). “SWRL: A semantic web rule language combining OWL and RuleML”. W3C Member submission, 21(79), 1-31.
13. M. Melik-Merkumians, A. Zoitl, and T. Moser, “Ontology-based fault diagnosis for industrial control applications,” Proc. 15th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA 2010, 2010.
14. Palacios, L., Lortal, G., Laudy, C., Sannino, C., Simon, L., Fusco, G., Ma, Y., Reynaud, C, “Avionics maintenance ontology building for failure diagnosis support,” IC3K 2016 - Proc. 8th Int. Jt. Conf. Knowl. Discov. Knowl. Eng. Knowl. Manag., vol. 2, no. Ic3k, pp. 204–209, 2016.
15. H. Peng, D. Chang, and Y. Wang, “Fault Diagnosis of Conveyor Based on Ontology,” vol. 157, no. 10, pp. 338–345, 2013.
16. Chen, R., Zhou, Z., Liu, Q., Pham, D.T., Zhao, Y., Yan, J., Wei, Q. , “Knowledge modeling of fault diagnosis for rotating machinery based on ontology,” Proceeding - 2015 IEEE Int. Conf. Ind. Informatics, INDIN 2015, pp. 1050–1055, 2015.
17. Xu, F., Liu, X., Chen, W., Zhou, C., Cao, B. , “Ontology-based method for fault diagnosis of loaders,” Sensors (Switzerland), vol. 18, no. 3, 2018.
18. Wan, S., Li, D., Gao, J., Roy, R., Tong, Y. , “Process and knowledge management in a collaborative maintenance planning system for high value machine tools,” Comput. Ind., vol. 84, no. 8, pp. 14–24, 2017.
19. Mourtzis, D., Vlachou, A., Zogopoulos, V. (2017). Cloud-based augmented reality remote maintenance through shop-floor monitoring: a product-service system approach. Journal of Manufacturing Science and Engineering, 139(6).
20. Xenakis, A., Karageorgos, A., Lallas, E., Chis, A. E., González-Vélez, H. (2019). Towards distributed IoT/cloud based fault detection and maintenance in industrial automation. Procedia Computer Science, 151, 683-690.
21. Mourtzis, D., Vlachou, E., Milas, N., Xanthopoulos, N. (2016). A cloud-based approach for maintenance of machine tools and equipment based on shop-floor monitoring. Procedia Cirp, 41, 655-660.
22. Mourtzis, D., Vlachou, E. (2018). A cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance. Journal of manufacturing systems, 47, 179-198.
23. Schmidt, B., Wang, L. (2018). Cloud-enhanced predictive maintenance. The International Journal of Advanced Manufacturing Technology, 99(1-4), 5-13.
24. C. Alcaraz, I. Agudo, D. Nunez and J. Lopez, “Managing Incidents in Smart Grids à la Cloud,” 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Athens, 2011, pp. 527-531, doi: 10.1109/CloudCom.2011.79.
25. Noy, Natalya F., and Deborah L. McGuinness. “Ontology development 101: A guide to creating your first ontology.” (2001).
26. Pellet reasoner, Online, https://www.w3.org/2001/sw/wiki/Pellet - [Accessed 24-04-2020]
27. Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W., & Musen, M. A. (2001). Creating semantic web contents with protege-2000. IEEE intelligent systems, 16(2), 60-71.
28. Web Ontology Language Reference, Online, https://www.w3.org/TR/owl-ref - [Accessed 24-04-2020]
29. GraphDB, Online, http://graphdb.ontotext.com - [Accessed 24-04-2020]
30. Node-Red, Online, https://nodered.org - [Accessed 24-04-2020]
31. Azure IoT Hub, Online, https://azure. microsoft. com/en-us/services/iot-hub - [Accessed 24-04-2020]
32. Andrew Banks, Rahul Gupta. MQTT Version 3.1. 1. OASIS standard, 2014, vol. 29, p. 89.
33. Docker: Empowering App Development for Developers, Online, https://www.docker.com - [Accessed 24-04-2020]
34. Carl. An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 2015, vol. 49, no 1, p. 71-79.
35. Bruckner, D., Stãnicã, M., Blair, R., Schriegel, S., Kehrer, S., Seewald, M., Sauter, T. “An introduction to OPC UA TSN for industrial communication systems.” Proc. of the IEEE 107.6 (2019): 1121-1131.
36. Haslhofer, B., Roochi, E.M., Schandl, B., Zander, S.. Europeana RDF store report. University of Vienna, 2011.
37. Modom.io, Online, https://modom.io - [Accessed 24-04-2020]

38. Zazuko Ontology Manager, Online, https://zazuko.com/products/ontology-manager - [Accessed 24-04-2020]
39. Alobaid, A., Garijo, D., Poveda-Villalón, M., Santana-Pérez, I., Fernández-Izquierdo, A., Corcho, Ó."Automating ontology engineering support activities with OnToology." Journal of Web Semantics 57 (2019): 100472.
40. Redland RDF Libraries, Online, http://librdf.org [Accessed 24-04-2020]