Process-aware IIoT Knowledge Graph: A semantic model for Industrial IoT integration and analytics

Process-aware IIoT Knowledge Graph: a semantic model for Industrial IoT integration and analytics

Claudia Diamantini[a], Alex Mircoli[a], Domenico Potena[a], Emanuele Storti[a,*]

[a]*Department of Information Engineering, Polytechnic University of Marche, via Brecce Bianche, Ancona, 60121, Italy*

Abstract

The integration of the huge data streams produced by the Industrial Internet of Things (IIoT) can provide invaluable knowledge in the context of Industry 4.0, but is also an open research issue. The present paper proposes a semantic approach to this issue, centered around the notion of process as the backbone. We build an ontology describing the fundamental elements involved in IIoT and their relations, and discuss the construction of the Process-aware IIoT Knowledge Graph, where raw sensor data are enriched with information about process activities and the physical production environment. We also propose a framework for querying the Knowledge Graph, and we demonstrate its capabilities by considering the production of metal accessories as case study.
The published version of this article is available at the Publisher website: https://www.sciencedirect.com/science/article/pii/S0167739X2200320X

Keywords: Industrial Internet of Things, Data Integration, Business Process, Semantics, Ontology, Knowledge Graph

1. Introduction

Industry 4.0 represents the fourth industrial revolution and is characterized by the introduction of digital and other innovative technologies in manufacturing. One of the key enabling technologies is the Internet of Things

*Corresponding author

Email addresses: c.diamantini@univpm.it (Claudia Diamantini), a.mircoli@univpm.it (Alex Mircoli), d.potena@univpm.it (Domenico Potena), e.storti@univpm.it (Emanuele Storti)

(IoT), defined as a network of interconnected devices that collect and exchange data through the Internet [1]. IoT encompasses a wide variety of application domains, with different and contrasting requirements [2, 3]. Focusing more specifically on industrial applications, the term Industrial Internet of Things (IIoT) has been coined. In contrast with other applications, where humans have a central role (connected devices are consumer electronic devices, and applications are devoted to improve human awareness of the surrounding environment) [4], IoT in industry is mainly devoted to connect machines to each other and to control systems, in order to enable self-organization, self-optimization, self-healing, towards a more autonomous and intelligent manufacturing system. In this respect, the focus is on the production line. Information is sensed and exploited on-line and locally in order to react to events, or exchanged with few peers for coordination or remote control during manufacturing. More recently, however, larger integration scenarios have been envisaged, up to the level of the whole factory, or even the enterprise, for collecting and analyzing flows of data enabling optimization and planning. Correspondingly, the reference architecture is emerging as an edge/fog-cloud architecture [5, 6].

Integration at cloud level of the huge data streams coming from the field, and providing a uniform view of the course of events are major issues. Model-based approaches may support the integration effort providing fundamental reference information. In particular, processes are recognized as a major means of integration in an organization. Processes are a set of interrelated activities, aimed at realizing a product or service that contributes to the achievement of the organization's goal. In the development of interrelated activities, many business units are involved. Operating on the same process, business units' resources must be integrated and coordinated for achieving the objective of the process. The process in this sense is therefore an element of homogenization of the organization. Recently, the work in [7] enlightens the fact that process analytics, execution, and monitoring based on IoT data can enable an even more comprehensive view of systems and realize unused potential for optimization. In particular, processes can improve the IoT by bridging the abstraction gap between raw sensor data and higher-level knowledge and optimizing the overall decision making, provided that some challenges are tackled and resolved.

In the present paper, we aim at moving a step forward towards this direction, by providing a semantic model that formalizes and relates the fundamental elements involved in IIoT for cloud-level integration and analytics.

We recognize these elements in (i) sensors, (ii) processes and (iii) related performance indicators, and (iv) the factory, intended as machinery and the environment that contains it. A comprehensive ontology is built following state-of-the-art design principles, in particular reuse, and abstraction.

The ontology acts as a conceptual knowledge layer providing the structure of the *Process-aware IIoT Knowledge Graph*, that includes all instances of elements listed before. In particular, it enables the enrichment of raw sensor data with information about process activities and the physical production environment and, as such, their contextualization. The proposal is enriched with a framework for querying the Knowledge Graph, whose capabilities are demonstrated by considering the production of metal accessories as case study.

The rest of the paper is organized as follows: Section 2 discusses the related literature. Section 3 introduces the case study. Then, Section 4 discusses the layered-based principle adopted to organize the knowledge artifacts, while Section 5 describes the proposed ontology, illustrating the component ontologies and the integration strategy. Sections 6 and 7 are respectively devoted to discuss the construction and exploitation of the Process-aware IIoT Knowledge Graph. Finally, Section 8 draws some conclusions and future work.

2. Related work

Interoperability is a key challenge in IIoT [8], due to the presence of multiple and heterogeneous data sources using different knowledge representations. In this context, ontologies are seen as a promising tool to solve interoperability problems [9], as they provide commonly agreed data models that are understandable by both humans and machines. In the field of IIoT, in particular, ontologies may support a wide range of activities (e.g., design, simulation, planning and scheduling, performance assessment and data integration [10]). Recently, the notion of Knowledge Graph (KG) has also attracted the interest of researchers. A KG is a semantic data model intended to accumulate and convey knowledge of the real world, where graph elements are defined through concepts of ontologies [11]. The following subsections are devoted to present some literature relevant for the present work. In particular, due to the adopted sensor and process perspectives, we detail existing proposals dedicated to these kinds of resources. Then, we discuss

work integrating and exploiting semantic technologies in the context of Industry 4.0.

2.1. Sensor ontologies

Sensor ontologies support several applications, including designing sensor networks, reasoning about available sensors and capabilities and querying sensor data. Given that sensor technologies are mutable over time, and that a manufacturing ontology should be applicable to different industrial domains, sensor ontologies should be general purpose and easily extensible, in order to adapt to different application domains [12]. Eil et al. [13] made an early attempt to define a general purpose ontology for sensor networks by proposing a framework based on the Suggested Upper Merged Ontology (SUMO) [14] to represent sensor classes, attributes and data. Another general purpose ontology is described in [15], where the authors present OntoSensor, which combines concepts and properties from the SUMO ontology with constructs from the Web Ontology Language [16]. To the best of our knowledge, the most comprehensive sensor ontology is represented by the Semantic Sensor Network (SSN) ontology [17], which has been proposed by the W3C Semantic Sensor Network Incubator group. Starting from SSN, Janowicz et al. define the Sensor, Observation, Sample, and Actuator (SOSA) ontology, which provides a "lightweight general-purpose specification for modelling the interaction between the entities involved in the acts of observation, actuation, and sampling" [18]. Similarly to SOSA, Bermudez et el. propose IoT-Lite [19], a lightweight semantic model which is an instantiation of key concepts of the Semantic Sensor Network (SSN) ontology. IoT-Lite does not contains constructs for modeling complex relationships but only offers a core ontology that contains the minimum concepts and relationships that can provide answers to most of the queries needed for IoT applications. IoT-Lite can also be extended by combining it with ontologies for IoT data streams, such as the Stream Annotation Ontology (SAO) ontology [20], although specific ontologies are available for IoT data streams, such as IoT-Stream [21].

2.2. Process ontologies

Process ontologies allow representing the semantics of process elements, which are usually formulated in natural language or semi-formal languages such as *Business Process Model and Notation* (BPMN), by using concepts of a formal ontology. The importance of defining a connection between ontologies and process models has been widely recognized in literature (e.g.,

[22] [23]). A general ontology for representing processes can be found in [24], where the authors combine existing ontologies and BPMN 2.0 in order to define the *BPMN 2.0 Based Ontology* (BBO) for business process representation. Similarly, Thomas et al. [25] propose to formally represent the semantics of element labels in process models by means of a process ontology. In [26] authors present a framework that connects the IoT infrastructure to a context-aware BPM ecosystem by means of IoT-integrated ontologies and IoT-enhanced decision models, while the work in [27] proposes to consider IoT devices as business process resources.

For what concerns the formal representation of industrial processes, Grüninger et al. [28] propose the Process Specification Language (PSL) ontology. The primary objective of the PSL ontology is to build a framework to formalize process information about scheduling, planning, simulation, work flow and project management. Another ontology for modeling manufacturing processes is the MASON ontology proposed by Lemaignan et al. [29]. The MASON ontology formalizes the concept of Process through a class named Operations, which allows to specify several manufacturing-related processes, including productive activities (e.g., pressing, plating, assembly), logistic operations, human operations (e.g., scheduling, programming) and launching operations (e.g., machine setup). Besides the PSL ontology and the MASON ontology, other ontologies developed for modeling manufacturing processes are the MSDL ontology [30] and the *ADAptive holonic COntrol aRchitecture for distributed manufacturing systems* (ADACOR) ontology [31]. Among all the presented ontologies, the PSL ontology provides the most general conceptualization of processes in various domains, but its generality may represent a limit when a specific representation of the manifacturing domain is needed. On the contrary, the MSDL ontology is less general but can be considered the most significant model for representing manufacturing processes, as it captures more specific and rigorous knowledge related to the manufacturing domain [32].

2.3. Integrated Ontologies

Several works extend the ontology scope to the modeling of different parts of the organization and their relations. The approach is modular most of the time and it consists in including and reusing existing resource specific ontologies, like in the present paper. Differences can be observed in the purpose served by the ontology, and hence in the specific ontologies adopted and/or in the structure of concepts. The work presented in [33] shares with the

present proposal the focus on intelligent data analysis. However, it puts emphasis on the adoption of temporal abstraction and temporal reasoning for searching and classifying qualitative temporal patterns in order to obtain information on the condition and state of the manufacturing process which, however, is not modelled. In contrast, in our approach the explicit representation of the production process is introduced, which puts data in the context of the logical workflow, and enables further reasoning capabilities. The idea of providing a context for sensor data is also proposed in [34]. To this end, raw sensor data are annotated with concepts of the *Context Ontology*, that provides the formal representation of the manufacturing domain, and specifically of resources, processes, sensors, time, location and situations. The ontology is built by reusing existing ontologies like the Time Ontology[1], the GeoSPARQL Ontology[2], and the SSN Ontology. The concept of process is barely represented, without further description of its structure, hence the reasoning enabled, mainly devoted to recognise situations that lead to potential failures and correlations among observations, cannot exploit this kind of context. In a very recent paper [35], an ontology network has been proposed, that models in detail the production process, the physical environment, equipments, (recipe) formulas, and materials. However, sensor data is not taken into account, since the scope of the paper is the support to planning and scheduling activities.

To the best of our knowledge, the present paper is the first attempt to link sensor data to the process activities executed when data has been generated.

2.4. Knowledge Graphs

Knowledge graphs are used in various contexts related to Industry 4.0. A preliminary investigation on the impact of KGs in Industry 4.0 can be found in [36], where authors discuss potential applications of KGs in multiple domains, ranging from maintenance, optimization and resources allocation. In [37], Bader et al. define the *Industry 4.0 Knowledge Graph* (I40KG), a KG for modelling norms, standards and specifications of Industry 4.0. The goal of the work is to represent knowledge related to both the single norms/standards and their mutual relations. Xie et al. [38] focus on sensors and propose to model them as subgraphs of a larger KG, which is used as an IoT middleware

[1]https://www.w3.org/TR/owl-time/

[2]https://opengeospatial.github.io/ogc-geosparql/geosparql11/index.html

with the aim of facilitating the integration of multiple sensors. In fact, such an IoT middleware allows communication and interoperability among IoT devices to be realized via attribute manipulation in the KG. The work in [39] exploits knowledge graphs and sensor data for generating production plans across multiple factories, by taking into account both static information (e.g., machine capabilities) and dynamic data (e.g., machine status, machine unavailability due to deterioration).

3. Case study

The proposed case study is a mid-size manufacturing company that produces metal accessories, which are obtained through fabrication and assembly of several metal parts. Such accessories are highly customizable and are produced in different sizes, shapes and colours, which implies that different products may be subject to different processing phases. The company has an industrial plant spread over two buildings, each comprising more rooms identified by capital letters; rooms, in turn, are divided into areas and subareas, identified by numbers. For instance, *1/B/1/2* refers to the subarea 2 of the area 1 in room *B* of the first building.

The overall production process is shown in the BPMN diagram depicted in Figure 1. It can be noticed that the *Washing* activity is optional, while

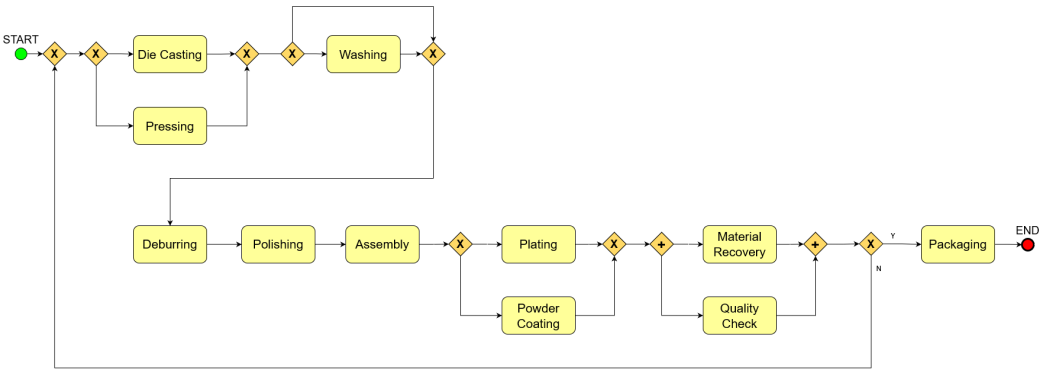

Figure 1: BPMN diagram of the manifacturing process of the case study.

Material Recovery and *Quality Check* are parallel activities. There are also XOR gateways as some processing steps can be performed using different techniques, depending on the product to be produced. Consequently, it is possible to identify macro-activities, corresponding to the set of different

ways in which a specific activity can be performed. For instance, the macro-activity *Painting* can be performed through two different techniques, i.e. *Powder Coating* and *Plating*. The production scheduling is defined by an Enterprise Resource Planning (ERP), which schedules production activities on the basis on internal rules that take into account the estimated lead time, the order priority and the delivery date. An example of production scheduling is reported in Table 1.

Table 1: Example of a production scheduling generated by the ERP.

Order Id	**Product Id**	**Input Parts**	**Output Parts**	**Qty**	**Activity**	**Production Date**	**Estimated Start Time**	**Estimated End Time**
...	...	...	...	...	...	...	...	...
0000246	29	P004	P004	500	Deburring	14/07/2021	09:00:00	11:45:00
0000249	26	RM002	P006	350	Die casting	14/07/2021	09:00:00	10:50:00
0000246	29	P004	P004	500	Polishing	14/07/2021	12:00:00	14:00:00
0000246	31	RM004	P003	200	Pressing	14/07/2021	12:00:00	14:30:00
0000271	26	RM004	P003	450	Pressing	14/07/2021	14:30:00	18:00:00
0000249	26	P006	P006	350	Washing	14/07/2021	14:30:00	15:00:00
0000249	26	P006	P006	350	Deburring	14/07/2021	15:15:00	16:45:00
0000246	29	P004, P005	P043	500	Assembly	14/07/2021	15:30:00	17:45:00
...	...	...	...	...	...	...	...	...

Information about production activities are saved in an event log for quality assessment purposes. Specifically, the human operator and the machine involved are reported, as well as the actual execution times, the total quantity of produced parts and the number of defective parts. An excerpt from the event log is shown in Table 2.

Table 2: Excerpt from the event log.

Order Id	**Product Id**	**Activity**	**Machine**	**Operator**	**Production Date**	**Start Time**	**End Time**	**Total Qty**	**Defective Parts**
...	...	...	...	...	...	...	...	...	...
0000246	29	Deburring	DE002	32	14/07/2021	09:02:14	11:41:52	500	0
0000249	26	Die casting	DC001	17	14/07/2021	09:01:03	10:54:41	366	16
0000246	29	Polishing	PO004	19	14/07/2021	11:59:31	14:11:12	500	0
0000246	31	Pressing	PR001	45	14/07/2021	12:06:10	14:33:24	230	30
0000271	26	Pressing	PR002	26	14/07/2021	14:35:03	18:01:24	454	4
...	...	...	...	...	...	...	...	...	...

Multiple machines, each identified by an incremental number (e.g., DE001 represents the first deburring machine), are available for each production activity. Machines that perform the same activity can even be located in different areas/room, as shown in Table 3, in which mappings between machines and locations are reported.

Table 3: Mappings between machines and locations

Activity	**Machine**	**Location**
Die Casting	DC1	1/A/2/1
	DC2	1/B/1/3
Pressing	PR1	1/A/1/1
	PR2	2/A/1/2
	PR3	1/B/1/2
Washing	WA1	1/B/2/1
	WA2	1/B/2/2
Deburring	DE1	1/B/2/1
	DE2	1/A/2/1
Plating	PL1	2/C/1/2
	PL2	2/B/1/1
...	...	...

Machines are equipped with sensors that monitor the most relevant physical quantities for each production activity, as shown in Table 4, where the types of sensors used in each activity are reported. For instance, a thermometer and a pressure sensor are equipped on the press in order to respectively measure temperature and pressure of mould cavity, which are critical parameters for the *Pressing* activity: in fact, production quality may be affected if pressure is out of specification. It has also to be noticed that different machines related to the same activity may be equipped with different sets of sensors (e.g., DC1 and DC2).

As depicted in Table 5, some sensors are not associated with a single machine but with rooms/areas or groups of machines. For instance, the anemometer is used to monitor the air speed of an entire subarea and evaluate if there are draughts that may have negative effects on *Powder Coating*. In some areas, several environmental sensors are placed with the purpose of obtaining more accurate measurements.

Both sensors equipped on machines and environmental sensors are connected to the Internet. For what concerns the IoT architecture, it is possible to identify three main components:

- *sensors*: there are environmental sensors and sensors equipped on machines. Each sensor collects data at a certain sampling frequency and sends them to the IoT Gateway through the Internet;

- *IoT Gateway*: the gateway receives raw data from sensors and performs a series of preprocessing operations;

Table 4: Mappings among activities and sensors on industrial machinery (excerpt).

Activity	Machine	Sensor	Measure
Die Casting	DC1	position transducer	position of the mould
		thermometer	temperature
		pressure sensor	oil pressure
		force sensor	clamping force
	DC2	thermometer	temperature
		pressure sensor	oil pressure
		force sensor	clamping force
Pressing	PR1	thermometer	temperature
		pressure sensor	pressure of mould cavity
	PR2	thermometer	temperature
		pressure sensor	pressure of mould cavity
Washing	WA1	pressure sensor	water pressure
		surfactant sensor	concentration of degreasing agent
Deburring	DE1	position transducer	position of workpiece
		force sensor	deburring force
Plating	PL1	paint thickness meter	coating thickness
Polishing	PO1	rotational speed sensor	rotational speed
Assembly	AS1	position transducer	position of pieces
		force sensor	assembly force
Powder Coating	PC1	paint thickness meter	coating thickness
		torque sensor	motor torque
Material Recovery	MR1	–	–
Quality Check	QC1	roughness meter	surface roughness
Packaging	PA1	precision scales	weight

- *cloud storage*: data are stored in a cloud database.

More in detail, the pre-processing phase is needed since sensor data are heterogeneous, as different sensor typologies produce different types of data (e.g., an accelerometer produces a terne of values while a thermometer only one). At the end of the above phase, sensor data related to the same physical quantity are transformed into a common schema and are then sent to a cloud database, where they are stored for further analysis. An example of preprocessed sensor data related to temperature is reported in Table 6: timestamps are converted to a common format and each measurement is associated with the location of the temperature sensor, on the basis of the mappings defined in Table 5.

Although the company collects large amounts of manufacturing data using sensors, some types of analysis are extremely difficult to perform without integrating information related to sensors, processes, environments and KPIs. For instance, in case of production defects, it would be very useful to analyze

Table 5: Mappings between buildings and environmental sensors.

Location	Sensor type	Sensor	Measure
1/A/1/1	thermometer	T1	air temperature
1/A/1/1	thermometer	T2	air temperature
1/A/1/1	hygrometer	H1	relative humidity
1/A/1/2	thermometer	T3	air temperature
1/A/2/1	hygrometer	H2	relative humidity
1/A/2/1	anemometer	A1	air speed
...	...		...

Table 6: Example of sensor data related to temperatures

Location	Sensor	Timestamp	Temperature
2/A/1/2	T6	2021-07-14 15:31:42.128	25.4
1/A/1/1	T1	2021-07-14 15:31:42.333	31.1
1/A/1/1	T2	2021-07-14 15:31:42.667	31.3
2/A/1/1	T5	2021-07-14 15:31:42.698	25.4
1/A/2/1	T4	2021-07-14 15:31:43.001	31.2
2/A/1/1	T5	2021-07-14 15:31:43.128	25.3
2/B/1/2	T7	2021-07-14 15:31:43.252	23.9
1/A/1/2	T3	2021-07-14 15:31:43.336	31.5

backwards the entire process in order to find any anomalies in sensor measurements. In fact, if a production defect is found during the quality control phase, it is possible to identify the cause by evaluating, for example:

- the temperatures measured during the *Die Casting* activity: a too low or too high temperature can affect the quality of die casting;
- the position of workpiece during the *Deburring* activity: a wrong position may lead to inaccurate deburring;
- the air speed measured by the anemometer during the *Powder Coating* activity: high speeds could indicate draughts, which may interfere with the painting activity.

4. Layered framework

In this section we discuss how information (or "knowledge artifacts") in the IIoT is categorized based on its typology and the object it represents.

Firstly, we recognize two main perspectives that are intertwined in the IoT environment, namely focusing on processes and sensors:

- the *process perspective* focuses on the definition, planning and execution of business processes, including all the process-related information;

- the *sensor perspective* focuses on the characterization of sensors and their deployment on machines or in the environment. The perspective includes information related to the configuration of sensors and their observations, corresponding to retrieved values.

Performance indicators for the process perspective are typically aimed to measure quantitative parameters related to the whole execution of a process instance, to a given sub-process or even to single activities. For instance, the *Cycle time* measures the end-to-end time needed to complete a (sub)process, while the *Number of defective products* aims to measure how many defective products are identified out of the total amount of manufactured ones for a given (sub)process or activity. Conversely, indicators related to sensor observations correspond to (or are derived from) the physical parameters that are monitored. These are typically low-level simple metrics, such as *temperature* or *relative humidity*. Such indicators are typically not associated to specific processes, although the value of the monitored parameters may significantly affect some process phases. To make an example, in the wood industry, relative humidity levels should range from 55% to 60%. Levels below 40% reduce the moisture content in the wood and may lead to variations in its size (such as shrinking or swelling, warping and cracking), which may be irreversible.

The two perspectives are further described in terms of layers, according to the goal of the representation:

- the *design* layer includes all knowledge artifacts related to the definition of a process model (e.g. BPMN, UML, Petri net diagrams) and the detailed description of sensors, including their characterization in terms of typology, input/output, capabilities, machine in which they are deployed, specific organization environment. The layer manages information that is considered to be stable across multiple executions.

- the *planning layer* focuses on the description of the schedule for a given process instance. This includes information on what activities are planned to be performed at a given time, on a given (set of) machine(s) and by whom. Additional information related to the specific production output are included, depending on the process type.

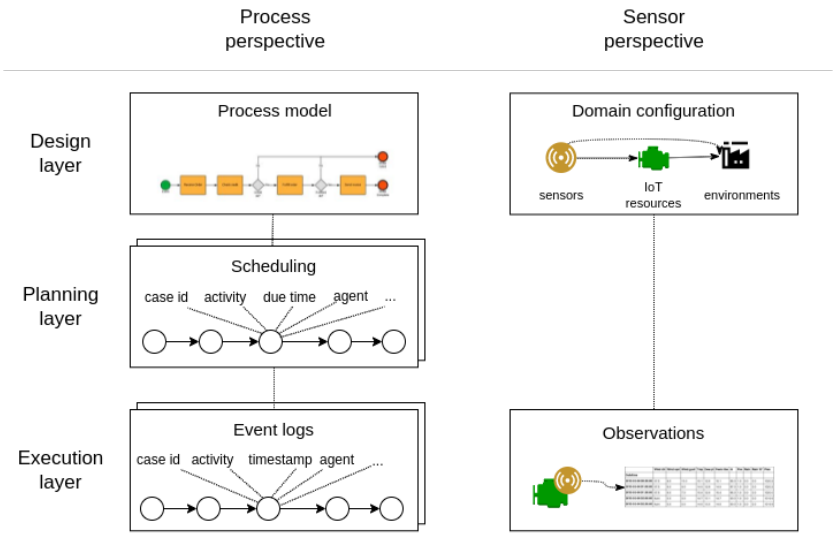

Figure 2: Layers and perspectives.

- the *execution layer* includes the event log, which contains information on past executions of a process instance and the related observations, i.e. the monitored values from the sensors. This layer includes dynamic information which is related to the specific executions.

Figure 2 outlines the relationships between perspectives and layers. In the following, we detail some relevant knowledge artifacts for the different layers.

4.1. Process models

A process model is a description of a process in terms of business activities to be performed and the flow among them in the organization. Although its primary use is descriptive, i.e. to document the current flow of activities in order to track what happens, process models can serve further purposes, including defining prescriptions, namely to establish what steps can/must be taken and under what circumstances.

A process modeling language, such as BPMN, provides the terminology and the rules for defining activities and their relations in the process. In more detail, a BPMN model, as shown in Figure 1, is expressed as a diagram including a set of graphical elements. Among them, *activities* represent atomic activities or sub-processes and are represented as rounded-corner rectangles. They are connected through a *connection*, such as sequence or message flow,

to other activities or to diamond-shaped *gateways*, which determine forking and merging of paths based on the expressed conditions, e.g. parallelism or exclusive choice. *Events* are shown as circles and are used to represent something that happens, such as the start or the end of a process, while *swim lanes* are a visual mechanism of organising and categorising activities, based on cross functional flowcharting.

Other languages, such as Petri Nets or process automata, are characterized by a mathematical formalization and a lower-level notation.

4.2. Process scheduling

For a given process, a scheduling refers to the detailed planning of the activities to be performed for a particular process instance. A scheduling typically includes an identifier of the corresponding production order, along with a set of information for each scheduled activity, such as the following:

- the resource in charge for performing the activity (hereby including both machines and employees);
- the number and type of items that will be taken as input for being processed by the activity;
- the number and type of items that will be produced as output;
- the expected starting and ending date and time of the scheduled activity.

4.3. Event log

Today, all enterprise information systems, including ERPs, CRMs, or workflow management systems, have the capability to collect information on process workflow events, which are typically related to specific activities. The term *event log* refers to such a detailed record of events about a business process, typically represented through accepted standards such as XES. An event log can be viewed as a multi-set of traces, where each trace describes the execution of a particular process instance as a totally ordered sequence of recorded events. Events typically refer to the starting or the ending of an activity, with further information including the resource (i.e. the person or the machine) which initiated or executed the activity, the timestamp of the event and possible data elements recorded with the event, such as the size of the order, the number of input parts processed or output parts produced.

Performance indicators, or KPIs, are typically recorded as well in event logs, to keep track of the cost of an activity, or of some other quantitative measures like the number of defective products that are discarded after performing the activity.

In many real-world settings, obtaining an event log from (often legacy) information systems inside the organization is however far from being trivial. Event data may indeed come from a wide variety of sources, such as database systems, transaction logs (e.g., a trading system), business suites/ERPs, message logs (e.g., from IBM middleware), but they are rarely recorded explicitly. Methodologies and guidelines on how to produce event logs from a relational database have been proposed, e.g., in [40], starting from the assumption that events leave footprints by changing the underlying database. In other approaches, such as [41], a framework is aimed to support domain experts in the extraction of XES event log information from legacy relational databases, by relying on a conceptual representation of the domain of interest in terms of an ontology.

4.4. Domain-specific configuration

Domain knowledge on the configuration of the industrial settings is useful to effectively and correctly interpret and analyse information on process execution and sensor observations.

To provide a spatial context to tasks executed in a process and values retrieved by sensors, information on how the enterprise is physically structured is needed, in terms of environments and sub-environments. In this way, the physical space of the enterprise is represented hierarchically and in a structured way, enabling to describe the location of a given space, e.g. a room, within the building in which it is located. IoT resources, i.e. industrial machines/platforms, are characterized in terms of the type of activity they can perform (e.g., die casting, pressing, washing), where they are located in the enterprise premises and the set of sensors they host. Finally, information on sensors includes the types of measures (e.g. temperature, relative humidity) they can provide, technical details on their functioning and where they are deployed. We assume sensors can be either installed on a machine or directly in the enterprise environment. The former case refers to sensors that provide information on how a machine operates during a process, e.g., to monitor the position of a work piece or the rotational speed of a motor. The location of the sensor is therefore dependent on that of the platform hosting it. The

latter type of sensors, on the other hand, is useful to monitor parameters of the environment where some activities of the process are executed.

4.5. Sensor observations

An observation produced by a sensor corresponds to a detected value of a measurement, taken at a given time. As such, depending on the sampling frequency of the sensor, multiple observations could possibly be produced by a sensor during an executed task. Both the value and the type of measure need to be stored along with further information such as the unit of measurement and the aggregation function. This last is meant to represent the specific mathematical function (e.g., sum, average, min, max) that can be applied to aggregate multiple values of the measure, i.e., multiple observations. As an example, all the observations related to a thermometer sensor hosted by an machine for a particular task can be aggregated through the *average* function to derive the average temperature that was observed during the execution of such a task.

5. A model for IIoT

This section is devoted to discuss a model for Industrial IoT. The model is able to completely represent the knowledge artifacts introduced in the previous section and highlight the relations among them. It is represented as an ontology which reuses and integrates, according to the best practices in ontology engineering, a number of existing ontological models:

- Semantic Sensor Network (SSN) [17] and Sensor, Observation, Sample, and Actuator (SOSA) [18] ontologies for the representation of sensors, their capabilities and related platforms;
- DogOnt [42] for the representation of smart environments;
- KPIOnto [43] for the representation of measures;
- Event ontology [44] for the representation of event logs.

We report in Table 7 the prefixes and the corresponding namespaces for the imported ontologies.

In addition to existing ontologies, we introduce a further module, the Business Process Ontology, for the representation of process models. Finally,

a set of classes and relations have been defined to build the bridge ontology capable to provide the needed connections among the different modules.

In the following, we summarize the most relevant details for each of them and discuss the integration strategy.

Ontology	**Prefix**	**Namespace**
Semantic Sensor Network	ssn	http://www.w3.org/ns/ssn/
Sensor, Observation, Sample, and Actuators	sosa	http://www.w3.org/ns/sosa/
DogOnt	dog	http://iot-ontologies.github.io/dogont/documentation/index-en.html
KPIOnto	kpi	http://w3id.org/kpionto/
Business Process Ontology	bpo	https://kdmg.dii.univpm.it/iot/ontology/bpo/
Event ontology	onp	http://onprom.inf.unibz.it

Table 7: Prefixes and namespaces for the imported ontologies.

5.1. Semantic Sensor Network (ssn)

Semantic Sensor Network (SSN) [17] is an ontology aimed to describe sensors in terms of capabilities, measurement processes, observations and deployments. The ontology is conceptually divided into a set of modules and includes 41 concepts and 30 object properties and an alignment to DOLCE-UltraLight (DUL) upper level ontology is provided. The core is built around a central ontology design pattern describing the relation between sensors, stimulus and observations (i.e., the Stimulus-Sensor-Observation pattern). In SSN, sensors (*ssn:Sensor*) are physical objects that observe, transforming a stimulus into an output representation (*ssn:SensorOutput*). A sensor follows (*ssn:implements*) a method that describes how it observes, e.g. to measure air temperature, the sensor is placed 2m above ground and must be protected from direct solar radiation. A sensor produces observations (*ssn:Observation*) of a property (*ssn:ObservationProperty*), which specifies a simple or more complex result (*ssn:hasSimpleResult* or *ssn:hasResult*) referring to a particular time (*ssn:resultTime*). Furthermore, a sensor is described in terms of measurement properties (e.g., accuracy, detection limits, drift, frequency, measurement range, precision, response time, resolution, sensitivity, selectivity), describing its capabilities in various conditions.

Related concepts which are not sensor specific, such as the sensor location, were left out of the ontology and can be described through suitable external ontologies.

5.2. Sensor, Observation, Sample, and Actuator (sosa)

The Sensor, Observation, Sample, and Actuator (SOSA) ontology [18] is the result of rethinking the SSN ontology based on recent changes in technical development and lessons learned over the last years. In particular, the resulting ontology extends its scope beyond sensors and observations and also considers actuators and sampling in a coherent framework with a flexible representation. As such, we refer to SOSA classes for the representation of sensors, observations and related properties. In particular, we represent sensors through the class *sosa:Sensor*, which can be hosted on a *sosa:Platform*, e.g. a smartphone is a platform typically hosting a number of sensors.

5.3. DogOnt (dog)

The DogOnt ontology [42] is aimed at providing an extensible model for the representation of devices in a smart environment. It focuses on various modeling aspects related to device capabilities, including low-level and communication issues, configurations that it can assume, and the structure of the environment where the device is deployed. As such, it can support a number of applications including reasoning on devices and integration of different technologies. In this work we refer to the DogOnt class *dog:BuildingEnvironment*, which describes smart environments that can be contained (*dog:isIn*) in other larger environments.

5.4. KPIonto (kpi)

KPIOnto is an ontology aimed to describe indicators and their properties [43]. The main class is *kpi:Indicator*, and in this work we rely on a subset of other classes and properties, including the *kpi:BusinessObjective*, the *kpi:AggregationFunction* and the *kpi:unitOfMeasure* (specific units are to be provided externally).

5.5. Business Process Ontology (BPO)

The Business Process Ontology is a lightweight core ontology we have designed to provide the vocabulary describing BPMN models in terms of process elements and relations among them. The ontology has been inspired by various ontological models developed for the representation of process models in different contexts, such as [24] and [25]. Its main class is *bpo:ProcessElement*, which includes both *bpo:Activity* and *bpo:Gateway* (e.g., AND, XOR and OR gateways) as subclasses. Activity is extended by subclasses *bpo:Subprocess* (i.e., a part of the global process model which starts with and ends with

a process element) and *bpo:AtomicActivity*. The control flow of the process is described through the *bpo:flows_directly* object property, which allows to model a sequence between any two given process elements. Finally, a *bpo:Process* consists of a number of process elements (*bpo:consistsOf*).

5.6. Event ontology (onp)

An event ontology was developed to support ontology-based data access to event log data in the context of the Onprom methodology [44]. This is aimed to guide data and process analysts in the conceptual identification of event data and their extraction from the Business Process Management system of the organization.

The ontological model includes all the main concepts needed to represent an event log: *onp:Log*, *onp:Trace*, *onp:Event* together with *onp:Attribute*. An instance of attribute can have a key (*onp:attKey*), a type (*onp:attType*) and a value (*onp:attValue*) as data properties.

As object properties, a log contains (*onp:l-contains-t*) one or more trace, each of which in turn contains (*onp:t-contains-e*) one or more events. A log, a trace or an event may have attributes, respectively though properties *onp:l-has-a*, *onp:t-has-a* and *onp:e-has-a*. Further classes and properties are defined for detailing the attribute types and possible extensions.

5.7. Integration strategy

The above-described ontologies have been integrated to provide a comprehensive model for the representation of an IoT-aware event log. To this aim, a bridge ontology has been designed to provide the needed classes and properties capable to align the different ontological modules. As such, the resulting ontology is equivalent to $\mathrm{O} \equiv \mathcal{O}_{sosa/ssn} \sqcup \mathcal{O}_{kpi} \sqcup \mathcal{O}_{dog} \sqcup \mathcal{O}_{bpo} \sqcup \mathcal{O}_{onp} \sqcup \mathcal{O}_{Bridge}$. In the following, we refer to the prefix *meta:* to refer to the classes and properties that have been defined in the bridge ontology. They are reported below and are shown in Figure 3:

- *IoTResource* is a class which specializes *sosa:Platform*. Capabilities of the IoT resource can specified through *hasCapability* as an *ActivityType*. This last describes the activities that can be performed inside the organization. An IoTResource, as well *sosa:Sensor*, is *locatedAt* a *dog:BuildingEnvironment*.

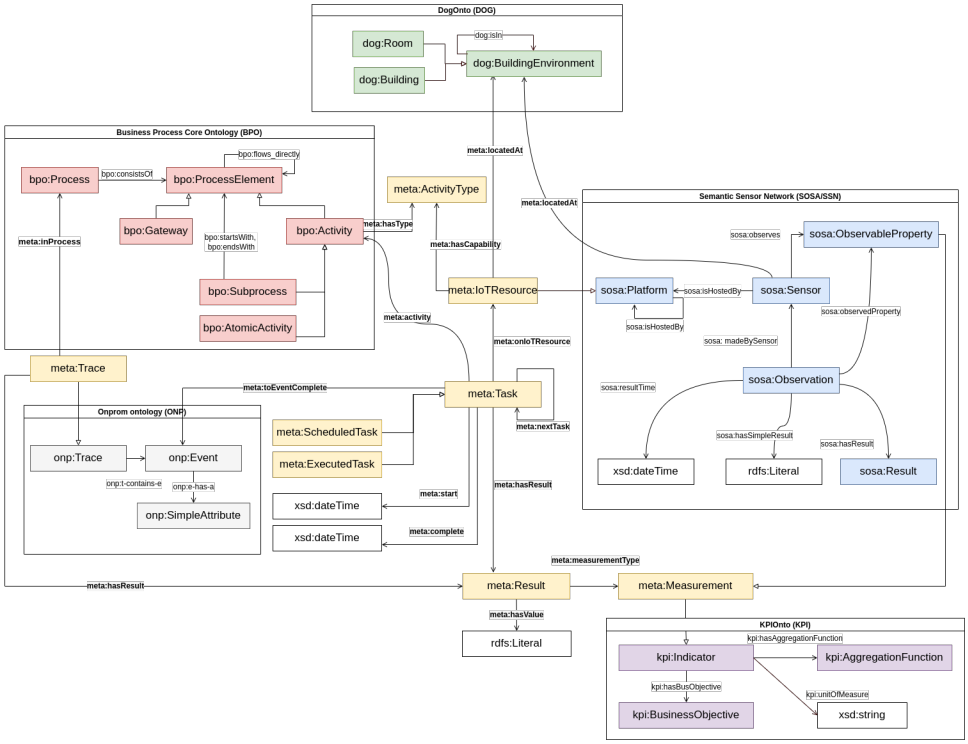

Figure 3: Integrated ontological model.

- *Task* is a generalization of two subclasses, namely *ScheduledTask* and *ExecutedTask*, to manage the representation of both a scheduled process or an event log related to a particular executed instance. A task is scheduled/executed on a given IoT resource (*onIoTResource*) and refers to a specific *bpo:Activity* of the process model. A task refers to the corresponding event through the *toEventComplete* property and has the *start* and *complete* properties representing the timestamp in which it starts and ends. Furthermore, the task is linked to the causally following task in the same schedule/trace through the *nextTask* property, following the approach that will be described in the next section.

- *Result* is a class representing a measure that is recorded in the log and refers to a task or a trace, e.g., the number of defective products that is measured after certain activities are performed. A result has a *Measurement* type and a specific value.

- *Trace* is a subclass of *onp:Trace* and is linked to the particular *bpo:Process* through the *inProcess* property.
- *Measurement* is defined as a subclass of *kpi:Indicator*, and in turn it extends *sosa:ObservableProperty*. It represents observable properties that are directly measured by a sensor or refer to tasks and traces.

The final ontological model is hence capable to overall represent (i) the process and (ii) the sensor perspectives. As for the former, this is achieved by defining a process model and its relations with the related scheduling and executed process instances. The scheduling and the event log are represented through the same approach, i.e. as a Scheduled/Executed trace including a set of scheduled or executed events. Each event is in relation with the particular scheduled/executed activity in the process model, and with the IoT resource which is in charge of its execution (in case of a scheduled event) or has eventually executed it (in case of an executed event).

As for the sensor perspective, the ontology includes information on the domain configuration, in terms of IoT resources that are located at specific places in the organizational environment and may include a number of sensors onboard, which are characterized in terms of observable indicators and all the observations that have been collected over time. Each observation implicitly refers to a given executed event which in turn refers to a particular process instance. This link can be derived by comparing the observation to the event timestamps, as discussed more in detail in the next section.

6. Process-aware IIoT Knowledge Graph construction

The ontological model discussed in the previous section represents the vocabulary for the definition of the Process-aware IIoT Knowledge Graph, which is represented as an RDF graph including all the knowledge artifacts belonging to the layers from the perspectives introduced in Section 4. While some of the artifacts, e.g., the process model, can be directly mapped to corresponding classes and properties in the ontological schema, some others, such as the event log and the observations, require proper care for adapting them to the ontological structure, deriving hidden information or performing data cleaning. In the following, we describe the methodology followed for the representation of the event log and the observations.

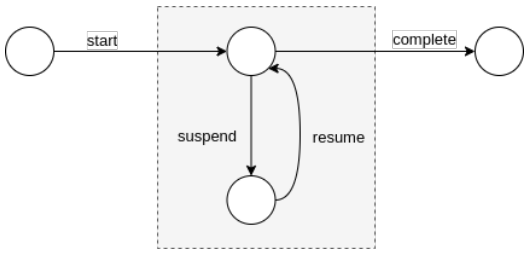

Figure 4: Minimal life-cycle model.

6.1. Representation of the event log

In this work, we assume that the event log contains a recording of multiple events related to each activity in a process. This corresponds to the notion of life-cycle of activities in the XES format for the encoding of event logs, which is a transactional model including all the possible states of an activity. The XES format describes a standard life-cycle model which includes, among others, the state "start" when the execution of the activity commences, and "complete" when it ends. Possible further states include "suspend" if the execution is paused, "resume" if it restarts. In Figure 4 a subset of the standard life-cycle model is shown. The gray box represents the state in which the activity is running. As a minimal assumption, in this work we consider event logs including two events for each activity, namely about the "start" and the "complete" life-cycle transitions. As a consequence, given the two timestamps of such events, we assume the corresponding activity is running between them. To make an example, let $e1$ be an event related to the "start" life-cycle transition of activity *Polishing*, while $e2$ is the event corresponding to the "complete" transition of the same activity. If the timestamps of $e1$ and $e2$ are respectively "2022-07-20T11:30:10" and "2022-07-20T11:36:30", then the activity ran for 6 minutes and 20 seconds.

Representing the event log in the ontological model is however not always straightforward. In fact, the event log includes only linear traces of events, where each trace is about a particular process instance. If the process model is linear as well, e.g., only includes sequences between activities, then the translation is simple. On the other hand, if the process includes parallel branches, then from a linear trace it is not immediately possible to determine the correct sequence of execution. Indeed, it may happen that two directly following events actually belong to activities on separate branches. For this reason, the causal relations among events need to be identified by

comparing the trace against the process model. For doing so, we rely on the approach discussed in [45], which is aimed to make causal relations between events in a trace explicit. This is done by translating a trace in an instance graph [46], where nodes represent events of the trace, while each edge represents a direct succession between two events, such that a causal relation among the corresponding activities holds. As a result, for each pair of "start" and "complete" events $e1$ and $e2$ in a trace, an instance of the class *ExecutedTask* is created in the Knowledge Graph, linked to the "complete" event $e2$ through the *completeAt* property. Such an instance of executed task also has a property *start* equal to the timestamp of the event $e1$, and a property *complete* with a value equal to the timestamps of event $e2$. The instance is then linked to the causally following *ExecutedTask* through the property *meta:nextTask*. Information on the process activity scheduled/executed and the particular related resource are available in the corresponding event as attributes. As such, a property *meta:activity* is added to link the task at hand to the proper instance of *bpo:Activity*. Similarly, a property *meta:onIoTResult* is added between the task and the related IoTResource. Finally, values of possible measurements that are related to the event are explicitly represented as instances of the *meta:Result* class, which includes the value and the type of measurement (as an instance of *meta:Measurement*).

6.2. Linking observations to the event log

An observation produced by a sensor is characterized by at least a value related to a certain *ObservableProperty* that is detected at a particular timestamp. As such, sensors produce only simple information which is not correlated to contextual information available in the environment. However, as a requisite for a correct and meaningful analysis of processes, in case of sensors deployed on a machine it is necessary to map each observation to the specific executed activity (and corresponding events) during which it was generated. To derive this information it is needed to determine: (1) the platform on which the sensor is deployed, (2) the executed task that took place on such a platform, such that the timestamp of the observation is included between the values of its *start* and *complete* properties. Given an observation $\langle obs \rangle$, the following SPARQL construct query creates the triple to link it to the executed task related to the activity on which it has been observed, by performing a comparison of the temporal instants in which the task and the observation are recorded.

```
CONSTRUCT <obs> meta:observedOn ?e
WHERE {
 <obs> a sosa:Observation;
       sosa:resultTime ?t;
       sosa:madeBySensor ?s.
 ?s sosa:isHostedBy ?p.
 ?e a meta:ExecutedTask;
    meta:onIoTResource ?p;
    meta:start ?start;
    meta:complete ?complete.
 FILTER (op:dateTime-less-than(?start,?t)).
 FILTER (op:dateTime-greater-than(?complete,?t)).
}
```

In particular, given the observation $\langle obs \rangle$ taken at timestamp t, at first the platform p is retrieved. An executed task e is then found so that the values of its *start* and *complete* properties are such that $start < t < complete$. A similar query is built for sensors which are deployed directly in the environment and are not hosted by any IoT resource. In this case, the observation will be linked to all tasks that were running when the observation was recorded.

6.3. Discussion

To end this section, we briefly discuss the feasibility of the Knowledge Graph creation and maintenance. First of all, we observe that the approach requires the availability of information in digital format. Some may be unavailable but easily manageable because limited in volume and relatively stable. This category includes information on the layout of the factory, on the machines and their capabilities, on the sensors, their characteristics and positioning. Other information of a more dynamic nature can be easily obtained from information systems capable of producing data about production scheduling and process executions (event logs), whilst the observations produced by the sensors are clearly available in an IIoT context. Explicit process models instead may often be not available. The adoption of an approach such as the one proposed could trigger greater attention towards business process modelling, but in any case, the availability of event logs allows to resort to process mining techniques in order to discover process models from past executions. The previous sections also explain how dynamic data can be easily linked in the Knowledge Graph automatically or with minimal user effort. It is to be noted that the proposed model is able to handle both *conformant*

and *non-conformant* traces. The former refers to traces that adhere to the process model, while in the latter exceptions or deviations from the process model occur. Conformance issues at production level can be considered less relevant than in management processes, given its routine nature, but they cannot be excluded. For this reason, in the paper we adopt the notion of instance graph to determine the causal relations between executed tasks and adopt a method for the construction of instance graphs that is robust with respect to non-conformities.

7. Graph exploration and analysis

In this section, we discuss how the model can be exploited for querying and explorating the Process-aware IIoT Knowledge Graph. The functionalities hereby described are aimed to define the query for extracting and analysing data under a set of user conditions.

The graph includes two different types of quantitative measures that can be analysed, namely those produced by sensors, which are described through instances of the class *ObservableProperty* (e.g., "Temperature"), and those attached to tasks and traces, which directly refer to the class *Measurement* (e.g., "Elapsed time", "Number of defective products", "Power consumption"). Such two typologies are different in the level of granularity at which they are expressed. The former refers in fact to single observations performed by sensors, hence they are atomic in nature. On the other hand, the latter refer to tasks or to whole traces. In some cases, it may be possible to aggregate observable properties from the level of sensor observation to the task level, e.g., by aggregating all observations recorded during a task. In turn, measures at the task level may be aggregated up at the trace level, e.g., by summing up the power consumption for all tasks of a trace.

Measures can be analysed and aggregated along different perspectives, which represent the focus of the analysis. For instance, if the analysts is interested in what is happening in a given room, sensors deployed in such a room (and observations thereof) could be considered. Likewise, if the focus is on a specific trace, the analysis may access measures about the tasks in such a trace, or, again, the observations recorded during its execution. Considering the Knowledge Graph and all the artifacts represented therein, a number of possible analysis dimensions can be considered: process model, process scheduling/execution, IoT resources, activity types, environments.

As such, an analogy with multi-dimensional analysis can be easily done. However, the graph data structure we refer to in this work is such that the extension of OLAP operations (e.g., slice and dice, roll-up/drill-down) to this case is not always straightforward, as several authors point out, e.g., [47, 48].

As a remarkable feature, the graph structure is such that the relations among dimensions are represented explicitly. To make an example, given a specific environment, such as a warehouse, the activity types that are meaningful to consider are only those that can be performed by resources located there. The same holds for the executed tasks, the sensors and the observations that can be considered. As a further consideration, dimension hierarchies are defined only dynamically, by considering the actual relations among instances. For instance, starting from a given environment, it is possible to focus the analysis on sub-environments by following the *isIn* relation. Otherwise, by starting the analysis on a process model, any part of it (e.g., a sub-process) could be considered to deepen the analysis. This interconnectedness can be exploited to make the exploration of the graph, and therefore the analysis of measures, intrinsically more flexible and dynamic, by enabling the user to move from one dimension to another, at the same time constraining the set and types of dimensions that can be meaningfully analysed.

For technically skilled user, exploration and analysis can be performed by directly querying the graph. In such a way, all the expressivity of the query language can be exploited to select, filter, group and aggregate information combining dimensions in any possibly valid way. Less experienced users can take advantage of a guided procedure for setting up the analytical task. An analytical task on the graph requires the execution of a SPARQL query which extracts the information of interest. Formally, a SPARQL query is a tuple (DS, R, GP, SM) where:

- DS is an RDF Dataset;
- R is a result form;
- GP is a graph pattern;
- SM is a set of solution modifiers.

In this work, we assume DS is the RDF serialization of the IIoT Knowledge Graph, while R is the SELECT modifier, which enables to specify the target result. For what concerns GP, the graph pattern of the query is generated

from the general pattern shown in Appendix A, capable to match the whole graph. Such a basic graph pattern is then constrained through a set of filtering conditions allowing to specify dimensions, measures of interest and grouping attributes, as described in the following:

1. selection of the analysis *dimensions*: this operation is conceptually equivalent to the identification of the subgraph of interest from the larger Knowledge Graph. The operation is done by setting constraints to the starting graph pattern;
2. selection of the *measure* to be analysed, which involves setting a condition to further constrain the graph pattern;
3. application of *grouping*: a grouping condition is expressed and added to the graph pattern.

Finally, the SM, namely the solution modifiers which represents the part related to projection, includes the grouping attribute(s) and the application of the aggregation operator on the chosen measure. The following subsections discuss more in detail the steps for query building.

7.1. Graph and subgraph selection

As shown in Appendix A, the basic graph pattern does not constrain the target list, showing all the variables defined in the body. As for the WHERE part, the query defines the generic pattern which extracts all the information which may be analysed. In the following, we refer to such a basic graph pattern as GP.

The subgraph selection step is aimed to add constraints to GP to extract the sub-part of the graph which is relevant according to some user requests. A request is expressed as a condition over a dimension, hereby represented as a class, e.g., Process, Trace, Task, IoTResource, Environment, Sensor. The condition may be as simple as a filtering condition, which imposes that only a particular instance of the class (or a set thereof) should be considered, e.g., only a specific environment or specific trace of a process. More complex conditions can however be expressed, e.g., tasks that happened within a temporal window.

The function `constrainGraph`(GP,*type*,*condition*) takes the query pattern GP, a class *type*, a *condition* as input, and returns a graph pattern GP' which includes the specified constraint. In more detail, the function adds a FILTER condition to the graph pattern which limits the possible values that a certain variable can take. Each variable in the input pattern GP is bound

to a given type, e.g. `?environment` is a variable representing instances of class *dog:BuildingEnvironment*. As such, the function retrieves the proper variable depending on the *type*, and generates the full FILTER condition as follows:

```
FILTER (?variable <condition>)
```

To give an example, if the condition requires that the environment should be `ex:warehouse`, the condition would be written as: `FILTER (?environment = ex:warehouse)`. The function can be executed multiple times in case more than one constraint needs to be expressed.

7.2. Measure selection

Once the subgraph of interest has been identified by the graph pattern GP', the function `getMeasures()` allows to list all the available measures. The function is implemented through SPARQL queries which return the instances available in the graph for (i) the class *ObservationProperty*, for measurements related to sensors, or (ii) the class *Measurement*, for measurements related to tasks and traces. Hereby, we show the latter query for measurement related to tasks:

```
SELECT distinct ?task_measurement
WHERE {
   <Q'>
}
```

Once a measure `<measurement>` has been chosen by the user, the function `addMeasure`(GP',*measurement*) updates the graph pattern in order to include a further filtering condition, which binds the proper variable to the chosen measurement. We show such a condition for a chosen `<measurement>` related to tasks:

```
FILTER (?task_measurement = <measurement>)
```

As a result, the function returns the updated graph pattern GP''.

7.3. Grouping

Given a graph pattern GP'' and a set of classes $\{type_1, \ldots, type_n\}$, the function `groupBy`($GP''$,$\{type_1, \ldots, type_n\}$) returns a graph pattern including, for each grouping type $type_i$, a grouping condition on the variable for

such a type. In order to make the target list compliant with the SPARQL specification, all attributes shown in list must be grouping attributes or attributes on which some aggregation is performed. For this reason, also the variable specifying the name of the measurement is added to the groupBy() function. As a result, the final graph pattern GP''' is produced.

7.4. Target list definition

Given a graph pattern GP''' and a chosen measure m, the function `project` returns a target list including the needed variables. The behaviour of the function is different depending on whether a grouping has been performed or not. If so, the function is called as `project`(GP''',m): the target list will include a variable for each grouping attribute and a variable for the measure m, aggregated by its aggregation function. Conversely, the function is called as `project`(GP''', {$type_1$,...,$type_n$}). Since there is no grouping attribute, the target list will include, for any class $type_i$, the corresponding variable.

Once all the components have been set, the final SPARQL query is built as a SELECT query with the target list defined by this last step, and a graph pattern obtained after the grouping step.

7.5. Applications to the case study

This section is aimed to discuss some applications of the approach to the case study presented in Section 3. The reported queries are representative of the diverse types of analysis of interest and show how the proposed model, together with the availability of a query language, enables powerful analyses on the IIoT Knowledge Graph.

Let us suppose the user aims to check whether the position of workpieces during the *Deburring* activity was nominal or not on trace number 554. Through a set of filters, the query fixes the activity type, the trace and the observational properties of the sensor that are related to the three positional dimensions. Finally, the average position is shown by grouping the observations by task and resource, in case multiple machines were involved for the activity.

```
SELECT ?task ?resource ?obsProp
        (AVG(?obs_simpleResult) as ?value)
WHERE {<Q>
       FILTER (?activityType = :deburring).
       FILTER (?trace = :trace554).
```

```
       FILTER (?obsProp = :pos_workpiece_x ||
                          ?obsProp = :pos_workpiece_y ||
                          ?obsProp = :pos__workpiece_z).
}
GROUP BY ?task ?resource ?obsProp
```

As a second example, the user detects an unexpectedly high number of defective products after the Powder Coating activity performed on 2022-07-20. He/she decides to check the measured values for air speed in the room, as high values are likely to interfere with the painting activity.

The following query sets a number of conditions: the analysis focuses on *Tasks* performed on the given date in the time range 11:30 - 15:30 and the activity type must be *Powder Coating*. Then, she decides to show values of the *air speed* grouped by environment, resource and task. Indeed, more than one machine may have been used for the activity, in different environments and traces.

```
SELECT ?environment ?resource ?task ?obsProp_env
       (AVG(?obs_env_simpleResult) as ?value)
WHERE {<Q>
       FILTER (?activityType = :powder_coating).
       FILTER (?start > "2022-07-30T11:30:00" &&
                ?end < "2022-07-30T15:30:00").
       FILTER (?obsProp_env = :air_speed).
}
GROUP BY ?environment ?resource ?task ?obsProp_env
```

As a further example, let us suppose that, for trace number 908, the quality check identified some defective products so that a second iteration of the process has been performed. The user wants to analyse the differences of pressure of the mould cavity as recorded by pressure sensor hosted by the press machine (possibly a different machine in the two iterations). This can be easily performed by the following query

```
SELECT ?obs_time ?resource ?obs_simpleResult
WHERE {<Q>
      FILTER (?trace = :trace908).
      FILTER (?activityType = :pressure).
      FILTER (?obsProp = :pressure_of_mould_cavity).
}
ORDER BY ?obs_time
```

In this case, no aggregation is needed as all available values are shown. Hence, the user is free to choose the attributes in the target list from those available in the graph pattern.

8. Conclusions and future work

The paper discussed a semantic approach for the integration of data streams coming from the field in the context of Industrial Internet of Things, providing a more comprehensive view of systems and supporting analytic tasks. The novelty of the approach is given by the process perspective taken in the conceptualization of the ontology and the related Process-aware IIoT Knowledge Graph, which allows to bridge the abstraction gap between raw sensor data and manufacturing activities. The proposal is enriched by a methodology for the building and management of the Knowledge Graph, and a framework for querying the graph that allows to easily perform data integration and powerful analyses of interest.

Future work will be devoted to design a comprehensive evaluation of the approach on a real case study and to study the application of the approach on relevant issues in the Industry 4.0 domain, like the analysis of data quality, the discovery of anomalies, and the monitoring and optimization of operations. As a conceptual issue, it will be interesting to study aspects related to the evolution of production, especially those related to changes in process models.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645–1660.

[2] A. C. Marosi, R. Lovas, Á. Kisari, E. Simonyi, A novel iot platform for the era of connected cars, in: 2018 IEEE international conference on future IoT technologies (Future IoT), IEEE, 2018, pp. 1–11.

[3] M. Cameranesi, C. Diamantini, A. Mircoli, D. Potena, E. Storti, Extraction of user daily behavior from home sensors through process discovery, IEEE Internet of Things Journal 7 (9) (2020) 8440–8450.

[4] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, Industrial internet of things: Challenges, opportunities, and directions, IEEE Transactions on Industrial Informatics 14 (11) (2018) 4724–4734.

[5] S. Yi, C. Li, Q. Li, A survey of fog computing: Concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata '15, 2015, p. 37–42.

[6] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, C. Mahmoudi, Fog computing conceptual model, Tech. rep., National Institute of Standards and Technology (2018).

[7] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, G. Fortino, A. Gal, U. Kannengiesser, F. Leotta, et al., The internet of things meets business process management: a manifesto, IEEE Systems, Man, and Cybernetics Magazine 6 (4) (2020) 34–44.

[8] H. Rahman, M. I. Hussain, A comprehensive survey on semantic interoperability for internet of things: State-of-the-art and research challenges, Transactions on Emerging Telecommunications Technologies 31 (12) (2020) e3902.

[9] F. Shi, Q. Li, T. Zhu, H. Ning, A survey of data semantization in internet of things, Sensors 18 (1) (2018) 313.

[10] E. Negri, L. Fumagalli, M. Garetti, Approach for the use of ontologies for kpi calculation in the manufacturing domain, Proceedings of the XX Summerschool of Industrial Mechanical Plants Francesco Turco, Napoli, Italy (2015) 16–18.

[11] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Comput. Surv. 54 (4) (2022).

[12] C. Schlenoff, T. Hong, C. Liu, R. Eastman, S. Foufou, A literature review of sensor ontologies for manufacturing applications, in: 2013 IEEE International Symposium on Robotic and Sensors Environments (ROSE), IEEE, 2013, pp. 96–101.

[13] M. Eid, R. Liscano, A. El Saddik, A novel ontology for sensor networks data, in: 2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, IEEE, 2006, pp. 75–79.

[14] A. Pease, I. Niles, J. Li, The suggested upper merged ontology: A large ontology for the semantic web and its applications, in: Working notes of the AAAI-2002 workshop on ontologies and the semantic web, Vol. 28, 2002, pp. 7–10.

[15] D. J. Russomanno, C. R. Kothari, O. A. Thomas, Building a sensor ontology: A practical approach leveraging iso and ogc models., in: IC-AI, Citeseer, 2005, pp. 637–643.

[16] D. L. McGuinness, F. Van Harmelen, et al., Owl web ontology language overview, W3C recommendation 10 (10) (2004) 2004.

[17] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al., The ssn ontology of the w3c semantic sensor network incubator group, Journal of Web Semantics 17 (2012) 25–32.

[18] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, M. Lefrançois, Sosa: A lightweight ontology for sensors, observations, samples, and actuators, Journal of Web Semantics 56 (2019) 1–10.

[19] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor, Iot-lite: a lightweight semantic model for the internet of things, in: 2016 INTL IEEE conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress, IEEE, 2016, pp. 90–97.

[20] S. Kolozali, M. Bermudez-Edo, D. Puschmann, F. Ganz, P. Barnaghi, A knowledge-based approach for real-time iot data stream annotation and processing, in: 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), IEEE, 2014, pp. 215–222.

[21] T. Elsaleh, M. Bermudez-Edo, S. Enshaeifar, S. T. Acton, R. Rezvani, P. Barnaghi, Iot-stream: a lightweight ontology for internet of things data streams, in: 2019 Global IoT Summit (GIoTS), IEEE, 2019, pp. 1–6.

[22] M. Hepp, F. Leymann, J. Domingue, A. Wahler, D. Fensel, Semantic business process management: A vision towards using semantic web services for business process management, in: IEEE International Conference on e-Business Engineering (ICEBE'05), IEEE, 2005, pp. 535–540.

[23] Y. Lin, D. Strasunskas, Ontology-based semantic annotation of process templates for reuse., in: EMMSAD, 2005, pp. 207–218.

[24] A. Annane, N. Aussenac-Gilles, M. Kamel, Bbo: Bpmn 2.0 based ontology for business process representation, in: 20th European Conference on Knowledge Management (ECKM 2019), Vol. 1, 2019, pp. 49–59.

[25] O. Thomas, M. Fellmann MA, et al., Semantic process modeling–design and implementation of an ontology-based representation of business processes, Business & Information Systems Engineering 1 (6) (2009) 438–451.

[26] R. Song, J. Vanthienen, W. Cui, Y. Wang, L. Huang, Context-aware bpm using iot-integrated context ontologies and iot-enhanced decision models, in: 2019 IEEE 21st Conference on Business Informatics (CBI), Vol. 01, 2019, pp. 541–550. doi:10.1109/CBI.2019.00069.

[27] S. Meyer, A. Ruppen, C. Magerkurth, Internet of things-aware process modeling: integrating iot devices as business process resources, in: International conference on advanced information systems engineering, Springer, 2013, pp. 84–98.

[28] M. Gruninger, C. Menzel, The process specification language (psl) theory and applications, AI magazine 24 (3) (2003) 63–63.

[29] S. Lemaignan, A. Siadat, J.-Y. Dantan, A. Semenenko, Mason: A proposal for an ontology of manufacturing domain, in: IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06), IEEE, 2006, pp. 195–200.

[30] F. Ameri, D. Dutta, An upper ontology for manufacturing service description, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 42578, 2006, pp. 651–661.

[31] S. Borgo, P. Leitão, Foundations for a core ontology of manufacturing, in: Ontologies, Springer, 2007, pp. 751–775.

[32] Q. Cao, C. Zanni-Merk, C. Reich, Ontologies for manufacturing process modeling: A survey, in: International Conference on Sustainable Design and Manufacturing, Springer, 2018, pp. 61–70.

[33] F. Roda, E. Musulin, An ontology-based framework to support intelligent data analysis of sensor measurements, Expert Systems with Applications 41 (17) (2014) 7914–7926.

[34] F. Giustozzi, J. Saunier, C. Zanni-Merk, Abnormal situations interpretation in industry 4.0 using stream reasoning, Procedia Computer Science 159 (2019) 620–629, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[35] M. Vegetti, G. Henning, Ontology network to support the integration of planning and scheduling activities in batch process industries, Journal of Industrial Information Integration 25 (2022) 100254.

[36] M. Yahya, J. G. Breslin, M. I. Ali, Semantic web and knowledge graphs for industry 4.0, Applied Sciences 11 (11) (2021) 5110.

[37] S. R. Bader, I. Grangel-Gonzalez, P. Nanjappa, M.-E. Vidal, M. Maleshkova, A knowledge graph for industry 4.0, in: European Semantic Web Conference, Springer, 2020, pp. 465–480.

[38] C. Xie, B. Yu, Z. Zeng, Y. Yang, Q. Liu, Multilayer internet-of-things middleware based on knowledge graph, IEEE Internet of Things Journal 8 (4) (2020) 2635–2648.

[39] D. Dhungana, A. Haselböck, S. Meixner, D. Schall, J. Schmid, S. Trabesinger, S. Wallner, Multi-factory production planning using edge computing and iiot platforms, Journal of Systems and Software 182 (2021) 111083.

[40] W. M. Van der Aalst, Extracting event data from databases to unleash process mining, in: BPM-Driving innovation in a digital world, Springer, 2015, pp. 105–128.

[41] D. Calvanese, M. Montali, A. Syamsiyah, W. M. van der Aalst, Ontology-driven extraction of event logs from relational databases, in: International Conference on Business Process Management, Springer, 2016, pp. 140–153.

[42] D. Bonino, F. Corno, Dogont-ontology modeling for intelligent domotic environments, in: International Semantic Web Conference, Springer, 2008, pp. 790–803.

[43] C. Diamantini, D. Potena, E. Storti, Sempi: A semantic framework for the collaborative construction and maintenance of a shared dictionary of performance indicators, Future Generation Computer Systems 54 (2016) 352–365.

[44] D. Calvanese, T. E. Kalayci, M. Montali, A. Santoso, Obda for log extraction in process mining, in: Reasoning Web International Summer School, Springer, 2017, pp. 292–345.

[45] C. Diamantini, L. Genga, D. Potena, W. van der Aalst, Building instance graphs for highly variable processes, Expert Systems with Applications 59 (2016) 101–118.

[46] B. F. van Dongen, W. M. P. van der Aalst, Multi-phase process mining: Building instance graphs, in: P. Atzeni, et al. (Eds.), Conceptual Modeling – ER 2004, Springer Berlin Heidelberg, 2004, pp. 362–376.

[47] S.-M.-R. Beheshti, B. Benatallah, H. R. Motahari-Nezhad, Scalable graph-based olap analytics over process execution data, Distributed and Parallel Databases 34 (3) (2016) 379–423.

[48] C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, Graph olap: a multi-dimensional framework for graph data analysis, Knowledge and information systems 21 (1) (2009) 41–63.

Appendix A. Basic graph pattern

```
#Task
?task a meta:Task;
      meta:start ?start;
      meta:complete ?end.
      meta:nextTask ?nextTask;
      meta:hasResult ?task_result;
      meta:onIoTResource ?resource;
      meta:activity ?processElement;
      meta:toEventComplete ?event.

?task_result a meta:Result;
             meta:measurementType ?task_measurement;
             meta:hasValue ?task_result_value.

?task_measurement a meta:Measurement;
                  kpi:hasAggregationFunction ?task_measurement_aggFunction;
                  kpi:unitOfMeasure ?task_measurement_unit.

#Trace
?trace  a meta:Trace
        onp:t-contains-e ?event;
        meta:hasResult ?trace_result;
        meta:inProcess ?process.

?trace_result a meta:Result;
              meta:measurementType ?trace_measurement;
              meta:hasValue ?trace_result_value.

?trace_measurement a meta:Measurement;
                   kpi:hasAggregationFunction ?trace_measurement_aggFunction;
                   kpi:unitOfMeasure ?trace_measurement_unit.

#Process
?process bpo:consistsOf ?processElement.
?processElement bpo:flows_directly ?next_processElement.

OPTIONAL {?processElement a bpo:Activity;
                          meta:hasType ?activityType.}.

#IoTResource
?resource a meta:IoTResource;
          meta:locatedAt ?environment;
          meta:hasCapability ?activityType.
```

```
#ActivityType
?activityType a meta:ActivityType.

#Environment
?environment a dog:BuildingEnvironment.
             dog:isIn ?larger_environment.

#Sensors
?sensor a sosa:Sensor;
        sosa:isHostedBy ?resource;
        sosa:observes ?obsProp.
?obsProp kpi:hasAggregationFunction ?obsProp_aggr_func;
         kpi:unitOfMeasure ?obsProp_unit.

?obs a sosa:Observation;
     sosa:madeBySensor ?sensor;
     sosa:resultTime ?obs_Time;
     sosa:observedOn ?task.
OPTIONAL {?obs sosa:hasSimpleResult ?obs_simpleResult.}.
OPTIONAL {?obs sosa:hasResult ?obs_result.}.

OPTIONAL {?sensor_env meta:locatedAt ?environment;
                      sosa:observed ?obsProp_env.
          ?obsProp_env kpi:hasAggregationFunction ?obsProp_env_aggr_func;
                       kpi:unitOfMeasure ?obsProp_env_unit.
          ?obs_env a sosa:Observation;
               sosa:madeBySensor ?sensor_env;
               sosa:resultTime ?obs_env_Time;
               sosa:observedOn ?task.
          OPTIONAL {?obs_env sosa:hasSimpleResult ?obs_env_simpleResult.}.
          OPTIONAL {?obs_env sosa:hasResult ?obs_env_result.}.
             }.
```