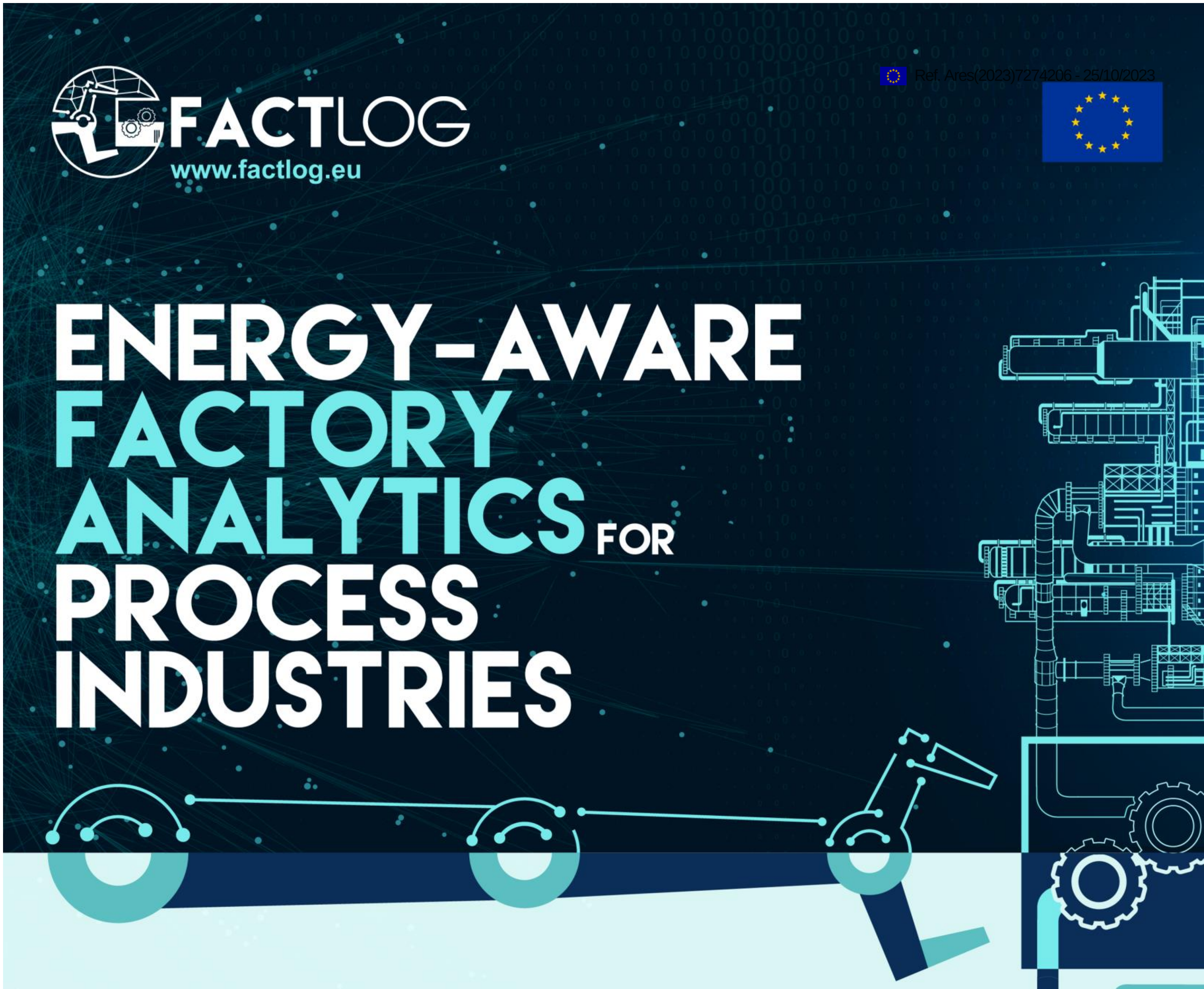
FACTLOG
www.factlog.eu



Deliverable D3.3

Factory Knowledge for Cognition

Version 1.2	**Lead Partner** EPFL
Date 25/10/2023	**Project Name** FACTLOG – Energy-aware Factory Analytics for Process Industries

FACTLOG has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 869951

Call Identifier H2020-NMBP-SPIRE-2019	**Topic** DT-SPIRE-06-2019 - Digital technologies for improved performance in cognitive production plants
Project Reference 869951	**Start date** November 1st, 2019
Type of Action IA – Innovation Action	**Duration** 42 Months

Dissemination Level

X	**PU**	Public
	CO	Confidential, restricted under conditions set out in the Grant Agreement
	CI	Classified, information as referred in the Commission Decision 2001/844/EC

Disclaimer

Executive Summary

The main goal of this deliverable is to boost the cognition process (i.e., the management of enhanced cognitive digital twins) with the factory knowledge, derived mainly in WP4, required for the internal reasoning processes. One of the main challenges will be to understand the role of process and domain knowledge in the cognition process formalized in T3.1, leading to a completely new view on the factory knowledge and consequently on the methods for processing and validating it. The main outcome is a set of interfaces for accessing that knowledge. In addition, feedback about the validity of knowledge will be sent as feedback to support the FACTLOG platform development.

In this deliverable, ontology and knowledge graph models are first investigated. Then cognitive factory services and knowledge graph modelling are identified to provide all the functionalities of knowledge graph models. Then ontology based on the BFO is introduced. Based on the ontology, knowledge graph models are developed. Finally, the integration of knowledge graph models and FACTLOG platform is demonstrated including three approaches: 1) integration based on OWL models; 2) integration based on Neo4j; 3) integration based on HTTP.

Revision History

Revision	Date	Description	Organisation
0.1	18/06/2022	First draft of the deliverable	EPFL
0.3	22/06/2022	Second draft of the deliverable	EPFL
0.3	28/06/2022	Third draft of the deliverable	JSI
0.4	29/06/2022	Final draft ready for internal review	EPFL
0.5	01/07/2022	Peer review	AUEB, TAGES
0.6	05/07/2022	Update based on the reviewers' comments	EPFL
1.0	05/07/2022	Final version ready for submission	EPFL
1.2	25/10/2023	Addressing the comments from EC	EPFL

Contributors

Organisation	Author	E-Mail
EPFL	Jinzhi Lu	Jinzhi.lu@epfl.ch
EPFL	Kiritsis Dimitris	dimitris.kiritsis@epfl.ch
JSI	Jože Rožanec	Joze.Rozanec@ijs.si

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

This document refers to a technical report about ontology and knowledge definition in Task 3.3 and providing a guidance for defining all the entities and relations in the context of five FACTLOG Pilots and technical partners about enhanced cognitive digital for factory. Knowledge Graph Model (KGM) denotes a generic ontology representation and description with all related product and equipment elements which is considered as a domain knowledge defined in this report. The enhanced cognitive twins are developed based on AI methods, algorithms, mechanisms, services and tools with knowledge graph models integrated into an overall modelling application.

In each specialized FACTLOG use case, the enhanced cognitive digital twins are developed based on related products, methods, algorithms and mechanisms with the knowledge graph models which require a unified and high-level abstract ontology definition because of the domain specific knowledge across different pilots. In the whole FACTLOG platform, the knowledge graph models are used to describe the factory knowledge, the platform services and their interrelationships. KGM is used to interconnect and interoperate with external AI tools, Optimization tools, Analytics tools, data visualization tools, etc. These tools require to develop the data interfaces based on the developed ontology for importing and exporting knowledge graph models. According to specific cognition needs, the knowledge graph models can be developed to implement reasoning to support anomaly detection.

The main content deals with the design and implementation of knowledge and development of knowledge graph models for reasoning. To summarize, main objectives of this report are:

- Develop knowledge graph models based on a top-level ontology for cognition services for analytic and dynamic models in the factory scenarios across FACTLOG pilot;
- Provide a reasoning approach for anomaly detection and data analysis for factory scenarios of FACTLOG.
- An integration approach based on Neo4j is introduced to demonstrate how FACTLOG platform is integrated with knowledge graph models to support decision-making.

1.2 Relation with other Deliverables

D2.1 is the input of this deliverable for pilot information.

D3.1 is the input of this deliverable for Enhanced Cognitive Twin definition.

D4.2 is an important reference of this deliverable for ontology definition.

D8.3 is the input of this deliverable for knowledge graph modelling specification.

1.3 Structure of the Document

Section 2 introduces the background of ontology, semantic modelling and knowledge graph modelling. Section 3 introduces the cognitive factory services and knowledge graph models. Section 4 introduces FACTLOG ontology based on BFO for knowledge graph modelling.

Section 5 introduces the knowledge graph models which are developed based on FACTLOG ontology. Section 6 introduces integration of knowledge graph models and cognition services in FACTLOG platform. Section 7 offers conclusions.

2 Ontology for knowledge definition about factory

2.1 Ontology Engineering and Semantic Modelling

2.1.1 Ontology Engineering

Ontology engineering is the general term of methodologies and methods for building ontologies. Ontology engineering refers to “The set of activities that concern the ontology development and the ontology lifecycle, the methods and methodologies for building ontologies and the tool suites and languages that support them”. The results of ontology engineering provide domain knowledge representation to be reused efficiently and prevent waste of time and money which are usually caused by non-shared knowledge. It helps Information Technology (IT) to operate with interoperability and standardization.

Ontology represents the nature of being, becoming, existence, and so on in the way of philosophy. One of the most well-known is: “ontology is an explicit, formal specification of a shared conceptualization of a domain of interest” [12].

Ontology represents the following ideas together [17]:

- Semantic modelling can help defining the data and the relationships between entities.
- An information model provides the ability to abstract different kind of data and provides an understanding of how the data elements are related.
- A semantic model is a type of information model that supports the modelling of entities and their relationships.
- The total set of entities in our semantic model comprises the taxonomy of classes we use in our model to represent the real world.

2.1.2 Semantic Modelling

The main objective of semantic modelling techniques is to define the meaning of data within the context of its correlation, and to model the domain world in the abstract level. The benefits of exploiting semantic data models for business applications are mainly as follows:

- **Avoiding misunderstanding**: by providing a clear, accessible, agreed set of terms, relations as a trusted source and discussions, misunderstandings can easily be resolved.
- **Conduct reasoning**: by being machine-understandable and through the usage of logic statements (rules), ontologies enable automatic reasoning and inference which leads to the automatic generation of new and implicit knowledge.
- **Leverage resources**: by extending and relating an application ontology to external ontological resources, via manual or automatic mapping and merging processes, the need for repetition of entire design process for every application domain is eliminated.
- **Improve interoperability**: semantic models can serve as a basis for schema matching to support systems’ interoperability in close environments where systems,

tools and data sources have no common recognition of data type and relationships.

2.1.3 Basic Ontology Concepts

Ontologies provide formal models of domain knowledge exploited in different ways. Therefore, ontology plays a significant role in many knowledge-intensive applications. Depending on corresponding languages, a number of different knowledge representation formalisms exist.

Ontologies provide formal models of domain knowledge exploited in different ways. Therefore, ontology plays a significant role for many knowledge-intensive applications. Depending on corresponding languages, a number of different knowledge representation formalisms exist. However, they share a minimal set of components as follows:

- Classes represent concepts, which are taken in a broad sense. For instance, in the Product Lifecycle domain, concepts are: Life Cycle phase, Product, Activity, Resources, Even, and so on. Classes in ontology are usually organized in taxonomies through which inheritance mechanisms can be applied.
- Relations represent a type of association between concepts of the domain. They are formally defined as any subset of a product of n sets, that is: $R \subset C1 \times C2 \times ... \times Cn$. Ontologies usually contain binary relations. The first argument is known as the domain of the relation, and the second argument is the range.
- Formal axioms serve to model sentences that are always true. They are normally used to represent knowledge that cannot be formally defined by the other components. In addition, formal axioms are used to verify the consistency of the ontology itself or the consistency of the knowledge stored in a knowledge base. Formal axioms are very useful to infer new knowledge.

For instance, Energy Efficiency at Buildings domain could be that it is not possible to build a public building without a fire door (based on legal issues).

- Instances are used to represent elements or individuals in an ontology.

As a Design Rationale (DR), ontology can be used as follows [6]:

- Level 1: Used as a common vocabulary for communication among distributed agents.
- Level 2: Used as a conceptual schema of a relational database. Structural information of concepts and relations among them is used. Conceptualization in a database is nothing other than conceptual schema. Data retrieval from a database is easily done when there is an agreement on its conceptual schema.
- Level 3: Used as the backbone information for a user of a certain knowledge base. Levels higher than this plays role of the ontology, which has something to do with "content".
- Level 4: Used for answering competence questions.
- Level 5: Standardization
  - Standardization of terminology (at the same level of Level 1)
  - Standardization of meaning of concepts
  - Standardization of components of target objects (domain ontology).
  - Standardization of components of tasks (task ontology)

- Level 6: Used for transformation of databases considering the differences of the meaning of conceptual schema. This requires not only the structural transformation but also semantic transformation.
- Level 7: Used for reusing knowledge of a knowledge base using DR information.
- Level 8: Used for reorganizing a knowledge base based on DR information.

2.2 Semantic Modelling Languages

Several semantic modelling languages are developed to support ontology definition and semantic modelling.

- **XML-based Ontology Exchange Language:** The US bioinformatics community designed XOL for the exchange of ontology definitions among a heterogeneous set of software systems in their domain. Researchers developed it after studying the representational needs of experts in bioinformatics. They selected Ontolingua (a Tool for Collaborative Ontology Construction) and OML as the basis for creating XOL, merging the high expressiveness of OKBC-Lite, a subset of the Open Knowledge Based Connectivity protocol, and the syntax of OML, based on XML. There are no tools that allow the development of ontologies using XOL. However, since XOL files use XML syntax, we can use an XML editor to author XOL files.
- **Simple HTML Ontology Extension:** SHOE is a small extension to HTML which allows web page authors to annotate their web documents with machine-readable knowledge. SHOE makes real intelligent agent software on the web possible. HTML was never meant for computer consumption; its function is for displaying data for humans to read. The "knowledge" on a web page is in a human-readable language (usually English), laid out with tables and graphics and frames in ways that we as humans comprehend visually. Unfortunately, intelligent agents aren't human. Even with state-of-the-art natural language technology, getting a computer to read and understand web documents is very difficult. This makes it very difficult to create an intelligent agent that can wander the web on its own, reading and comprehending web pages as it goes. SHOE eliminates this problem by making it possible for web pages to include knowledge that intelligent agents can actually read.
- **Ontology Markup Language:** OML, developed at the University of Washington, is partially based on SHOE. In fact, it was first considered an XML serialization of SHOE. Hence, OML and SHOE share many features. Four different levels of OML exist: OML Core is related to logical aspects of the language and is included by the rest of the layers; Simple OML maps directly to RDF(S); Abbreviated OML includes conceptual graphs features; and Standard OML is the most expressive version of OML. We selected Simple OML, because the higher layers don't provide more components than the ones identified in our framework. These higher layers are tightly related to the representation of conceptual graphs. There are no other tools for authoring OML ontologies other than existing general-purpose XML edition tools.
- **Ontology Interchange Language**: OIL, developed in the OntoKnowledge project (www.ontoknowledge.org/OIL), permits semantic interoperability between Web resources. Its syntax and semantics are based on existing proposals (OKBC, XOL, and RDF(S)), providing modelling primitives commonly used in frame-based approaches to ontological engineering (concepts, taxonomies of concepts, relations, and so on), and formal semantics and reasoning support found in description logic approaches (a subset of first order logic that maintains a high expressive power,

together with decidability and an efficient inference mechanism). OIL, built on top of RDF(S), has the following layers: Core OIL groups the OIL primitives that have a direct mapping to RDF(S) primitives; Standard OIL is the complete OIL model, using more primitives than the ones defined in RDF(S); Instance OIL adds instances of concepts and roles to the previous model; and Heavy OIL is the layer for future extensions of OIL. OILEd, Protégé2000, and WebODE can be used to author OIL ontologies. OIL's syntax is not only expressed in XML but can also be presented in ASCII. We use ASCII for our examples.

- **DARPA Agent Markup Language+OIL**: DAML+OIL has been developed by a joint committee from the US and the European Union (IST) in the context of DAML, a DARPA project for allowing semantic interoperability in XML. Hence, DAML+OIL shares the same objective as OIL. DAML+OIL is built on RDF(S). Its name implicitly suggests that there is a tight relationship with OIL. It replaces the initial specification, which was called DAML-ONT, and was also based on the OIL language. OILEd, OntoEdit, Protégé2000, and WebODE are tools that can author DAML+OIL ontologies.
- **OWL**: OWL is the result of the work of the W3C Web Ontology Working Group. This language derived from DAML+OIL and, as the previous languages, is intended for publishing and sharing ontologies in the Web. OWL is built upon RDF(S), has a layered structure and is divided into three sublanguages: OWL Lite, OWL DL and OWL Full. OWL is grounded on Description Logics and its semantics are described in two different ways: as an extension of the RDF(S) model theory and as a direct model-theoretic semantics of OWL. Both of them have the same semantic consequences on OWL ontologies.
- **OWL 2**: OWL 2 is an extension and revision of OWL that adds new functionality with respect to OWL; some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressivity. OWL 2 includes three different profiles (i.e., sublanguages) that offer important advantages in particular application scenarios, each trading off different aspects of OWL's expressive power in return for different computational and/or implementation benefits. These profiles are:
- **OWL 2 EL**: It is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.
- **OWL 2 QL**: It is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL).
- **OWL 2 RL**: It is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples. OWL 2 ontologies: The Direct Semantics that assigns meaning directly to ontology structures and the RDF- Based Semantics that assigns meaning directly to RDF graphs.
- **Resource Description Framework and RDF Schema:** RDF, developed by the W3C for describing Web resources, allows the specification of the semantics of data based on XML in a standardized, interoperable manner. It also provides mechanisms to explicitly represent services, processes, and business models, while allowing recognition of nonexplicit information. The RDF data model is equivalent to the semantic networks formalism. It consists of three object types:
  - ✓ Resources are described by RDF expressions and are always named by URIs plus optional anchor IDs
  - ✓ Properties define specific aspects, characteristics, attributes, or relations used

to describe a resource
- ✓ Statements assign a value for a property in a specific resource (this value might be another RDF statement)

The RDF data model does not provide mechanisms for defining the relationships between properties (attributes) and resources—this is the role of RDFS. RDFS offers primitives for defining knowledge models that are closer to frame-based approaches. RDF(S) is widely used as a representation format in many tools and projects, such as Amaya, Protégé, Mozilla, SilRI, and so on.

According to W3C, RDF model has advantages as follows:

- The RDF model is made up of triples: as such, it can be efficiently implemented and stored; other models requiring variable-length fields would require a more cumbersome implementation
- The RDF model is essentially the canonicalization of a (directed) graph and has all the advantages (and generality) of structuring information using graphs
- The basic RDF model can be processed even in absence of detailed information (an "RDF schema") on the semantics: it already allows basic inferences to take place, since it can be logically seen as a fact basis
- The RDF model has the important property of being modular

The union of knowledge (directed graphs) is mapped into the union of the corresponding RDF structures. Since RDF is a standard model for data interchange and is a W3C recommendation designed to standardize the definition and use of metadata-descriptions of Web-based resources, it is well suited to representing data. As knowledge representation, when it comes to semantic interoperability, RDF has significant advantages: The object-attribute structure provides natural semantic units because all objects are independent entities. A domain model—defining objects and relationships—can be represented naturally in RDF. To find mappings between two RDF descriptions, techniques from research in knowledge representation are directly applicable. Therefore, the Z-BRE4K ontology has been implemented in the RDF format.

2.3 Knowledge Graph Modelling

From several academic papers, we found different researchers and companies have proposed many different definitions. To synthesize a coherent definition that helps frame the discussion about KGs, the definitions in references ([8], [13], [5], [7], [14], [18], [2], [10], [4]) were reviewed. They had the following common features:

1. A KG represents interrelationships. All of the definitions specify this feature but do so in different ways.
2. A KG uses techniques to extract knowledge from one or more sources. The kinds of sources differ from one definition to another.
3. The organization is a graph, although the precise meaning of "graph" varies from one definition to another.
4. While a KG must have a schema, not all KG definitions mention it. Those that do mention it specify that the schema defines classes and relations.
5. The KG supports various graph-computing, search, and query interfaces. The supported operations and performance will vary, and the performance will depend on

how trade-offs among scalability, performance, and maintainability are handled as well as on other technical issues.

From these features it is apparent that a KG is not simply another way to represent facts. It involves a software architecture that includes active capabilities for extracting and processing the facts. Jans Aasman [1] characterized the operations of a KG as follows:

- Generation:
    - Collection: Ingestion, web extraction, catalogue extraction, ontology, ...
    - Processing: Schema mapping, entity resolution, cleaning, ...
- Storage
- Applications: Querying, graph mining, recommendation, search, question answering, ...
- Statistical and machine learning techniques are used for all of the above

Accordingly, these lead to the following proposal for a definition of a KG:

A KG is a representation of a set of statements in the form of a node- and edge-labelled directed multigraph allowing multiple, heterogeneous edges for the same nodes. A collection of definitional statements specifying the meaning of the knowledge graph's labels is called its schema.

In common, different graph database can support knowledge graph modelling which enables to integrate with IoT platform, such as Neo4j [9].

2.4 Semantic Reasoning based on Knowledge Graph Models

Based on knowledge graph models, semantic reasoning is the ability to infer new facts from existing knowledge graph data based on inference rules or ontologies. In simple terms, rules add new information to the existing knowledge graph models, adding context, knowledge, and valuable insights. The rules are declarative in nature that declare the desired logic of connections in the knowledge graph models and the process of inferring new facts happens either through pre-materialization, query rewriting or a combination of both.

Reasoning enables to manage consistent among knowledge graph models and speed up the intelligent services by minimizing the amount of information processing that needs to happen outside your database's reasoning engine, as well as the number of required operations. It also brings analysis closer to the knowledge graph models, meaning deeper insights can be gathered from a given knowledge graph model with far less computational effort.

A semantic reasoning engine (otherwise known as a semantic reasoner, inference engine, or rules engine) is a piece of software designed to perform reasoning—to apply rules to a knowledge graph model and conduct semantic inference as we've described. The semantic reasoning tools includes, SQWRL engine [11], SWRL engineer [16], etc.

2.5 Summary and Motivation

In summary, knowledge graph models are always developed based on semantic modelling languages. When developing knowledge graph models, ontology is used to support specification definition of different knowledge graph model elements. Through the ontology, different domain specific knowledge can be integrated into a unified framework.

3 Cognitive Factory Services and KGM

In this section, cognitive factory twins based on knowledge graph models are introduced. Then the function of knowledge graph models in FACTLOG factory scenarios are demonstrated in Section 3.2.

3.1 Cognitive Factory Twins based on Knowledge Graph Models

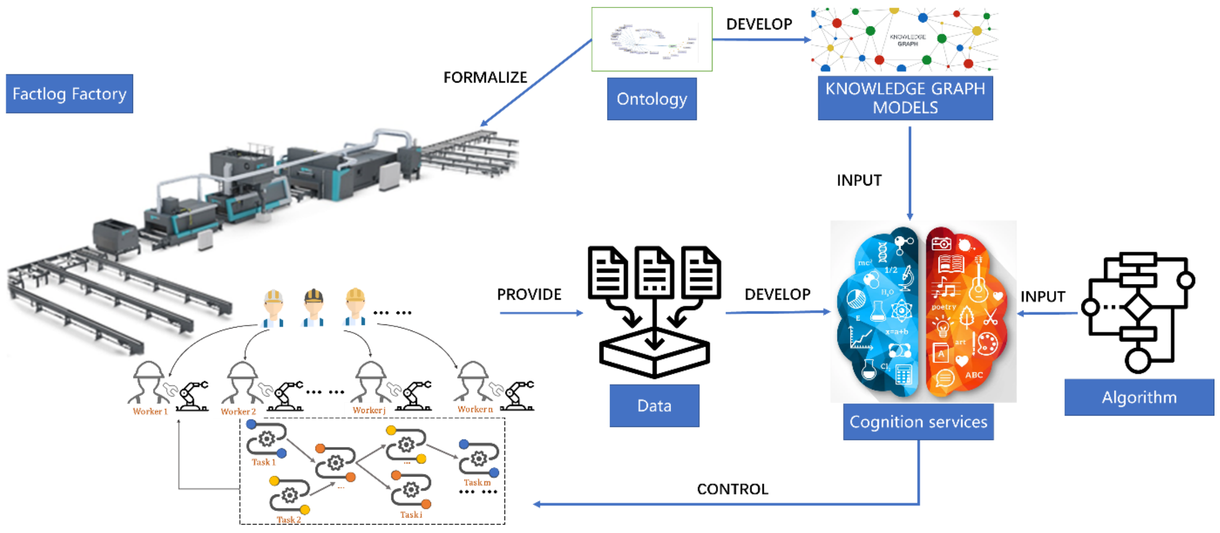


Figure 1: Knowledge graph models and cognition services to FACTLOG factory [15]

As shown in Figure 1, the interrelationships among FACTLOG factories and knowledge graph models are shown:

1. FACTLOG factory provides data to cognition services.
2. Ontology formalizes FACTLOG factory.
3. Ontology is used to develop knowledge graph models.
4. Knowledge graph models, algorithm and data are used to develop cognition services.
5. The cognition services control FACTLOG factory.

3.2 The functions of Knowledge Graph Models in FACTLOG Factory scenario

The functions of knowledge graph models in the FACTLOG factory scenarios are listed:

- Unified description of digital twins and information across FACTLOG platform
- Anomaly detection for cognition
- Visualization of the interrelationships of all the ontology entities and individuals

4 Ontology definition for KGM

4.1 Guidance to KGM for cognition services of factory

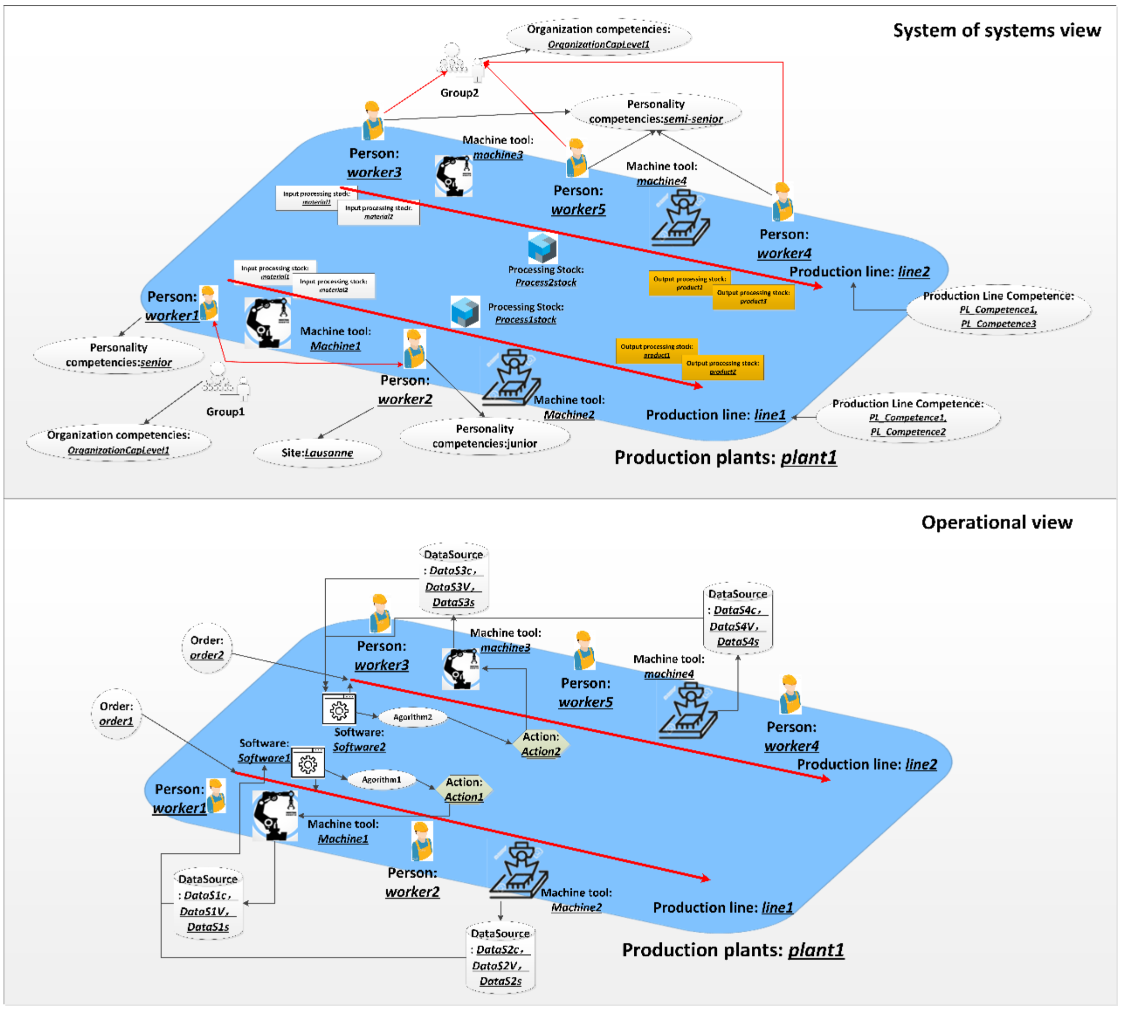

System of systems view
Organization competencies: OrganizationCapLevel1
Group2
Personality competencies:semi-senior
Person: worker3
Machine tool: machine3
Person: worker5
Machine tool: machine4
Person: worker4
Production line: line2
Processing Stock: Process2stock
Processing Stock: Process1stock
Production Line Competence: PL_Competence1, PL_Competence3
Person: worker1
Personality competencies:senior
Machine tool: Machine1
Person: worker2
Machine tool: Machine2
Production line: line1
Production Line Competence: PL_Competence1, PL_Competence2
Group1
Organization competencies: OrganizationCapLevel1
Site:Lausanne
Personality competencies:junior
Production plants: plant1
Operational view
DataSource : DataS3c, DataS3V, DataS3s
DataSource : DataS4c, DataS4V, DataS4s
Order: order2
Order: order1
Software: Software2
Software: Software1
Algorithm2
Algorithm1
Action: Action2
Action: Action1
DataSource : DataS1c, DataS1V, DataS1s
DataSource : DataS2c, DataS2V, DataS2s

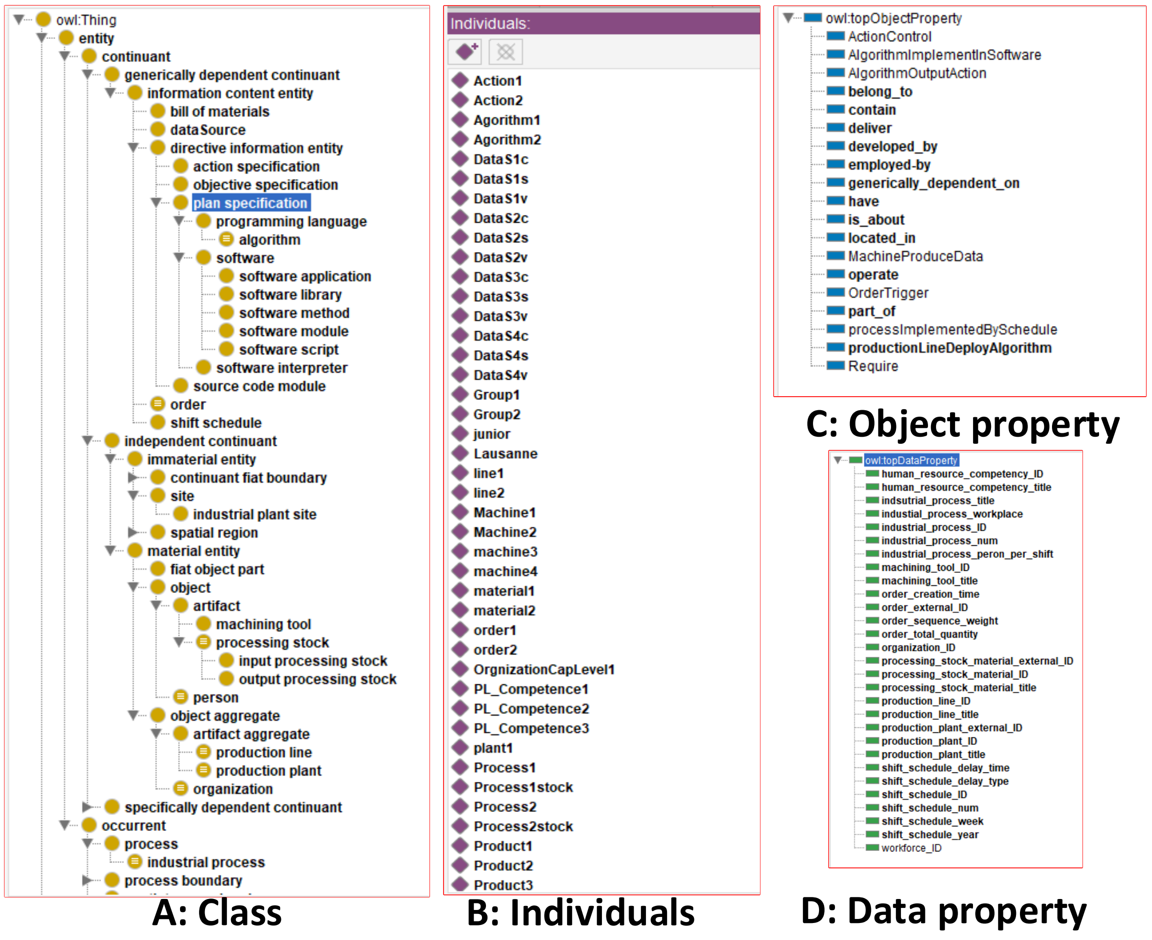
Figure 2: Scenario Overview -A: System of systems perspective; -B: Operational view.

In order to understand how the knowledge graph models, support the formalism of FACTLOG factory, an example is demonstrated as follow:

One scenario is proposed to develop a tutorial of knowledge graph modelling. The scenario is about a shop-plant. In Figure 2-A, the system of systems view is introduced. A *production plants (plant1)* is located in *site (Lausanne)*. The *plant1* includes two *production lines (line1 and line2)*. Two *workers (worker1 and worker2)* work in *line1*. They are in *Group1* which competence is *OrganizationCompLevel1*. The *worker1* has *senior* competence and *worker2* has *junior* competence. Three *workers (worker3, worker4 and worker5)* work in *line2*. They are in *Group2* which competence is *OrganizationCompLevel2*. They have the same person competence: *semi-senior*. *Machine1 and Machine2* are implemented in *line1*. *Machine3 and Machine4* are implemented in *line2*. In *line1*, *process stock(process1stock)* is implemented. The *process1stock* has two *input process stocks (material1 and material2)* and two *output process stocks (product1 and product2)*. The *line1* has competences: *PLCompetent1* and *PLCompetent2* which produce *product1* and *product2*. In *line2*, *process stock(process2stock)* is implemented. The *process2stock* has two *input process stocks* *(material1 and material2)* and two *output process stocks (product1 and product3)*. The *line2* has competences: *PLCompetent1* and *PLCompetent3* which produce *product1* and *product3*.

As shown in Figure 2-B, the operational view is illustrated. When the development process starts, *order (order1)* triggers the *process1* and *order (order2)* triggers the *process2*. The *process1* is monitored and controlled by the *algorithm1* in *software1*. The *algorithm1* provides *action1* to control *machine1*. The *machine1* and *machine2* generate data which provided to *software1*. The *process2* is monitored and controlled by the *algorithm2* in *software2*. The *algorithm2* provides *action2* to control *machine2*. The *machine3* and *machine4* generate data which provided to *software2*.

4.1.1 Knowledge Graph Modelling



```
owl:Thing
  entity
    continuant
      generically dependent continuant
        information content entity
          bill of materials
          dataSource
          directive information entity
            action specification
            objective specification
            plan specification
              programming language
                algorithm
              software
                software application
                software library
                software method
                software module
                software script
              software interpreter
            source code module
          order
          shift schedule
      independent continuant
        immaterial entity
          continuant fiat boundary
          site
            industrial plant site
          spatial region
        material entity
          fiat object part
          object
            artifact
              machining tool
              processing stock
                input processing stock
                output processing stock
            person
          object aggregate
            artifact aggregate
              production line
              production plant
            organization
      specifically dependent continuant
    occurrent
      process
        industrial process
      process boundary
```

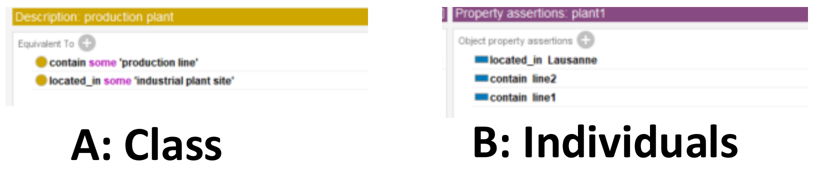
```
Individuals:
Action1
Action2
Agorithm1
Agorithm2
DataS1c
DataS1s
DataS1v
DataS2c
DataS2s
DataS2v
DataS3c
DataS3s
DataS3v
DataS4c
DataS4s
DataS4v
Group1
Group2
junior
Lausanne
line1
line2
Machine1
Machine2
machine3
machine4
material1
material2
order1
order2
OrgnizationCapLevel1
PL_Competence1
PL_Competence2
PL_Competence3
plant1
Process1
Process1stock
Process2
Process2stock
Product1
Product2
Product3
```

```
owl:topObjectProperty
  ActionControl
  AlgorithmImplementInSoftware
  AlgorithmOutputAction
  belong_to
  contain
  deliver
  developed_by
  employed-by
  generically_dependent_on
  have
  is_about
  located_in
  MachineProduceData
  operate
  OrderTrigger
  part_of
  processImplementedBySchedule
  productionLineDeployAlgorithm
  Require
```

```
owl:topDataProperty
  human_resource_competency_ID
  human_resource_competency_title
  indsutrial_process_title
  industial_process_workplace
  industrial_process_ID
  industrial_process_num
  industrial_process_peron_per_shift
  machining_tool_ID
  machining_tool_title
  order_creation_time
  order_external_ID
  order_sequence_weight
  order_total_quantity
  organization_ID
  processing_stock_material_external_ID
  processing_stock_material_ID
  processing_stock_material_title
  production_line_ID
  production_line_title
  production_plant_external_ID
  production_plant_ID
  production_plant_title
  shift_schedule_delay_time
  shift_schedule_delay_type
  shift_schedule_ID
  shift_schedule_num
  shift_schedule_week
  shift_schedule_year
  workforce_ID
```

Figure 3: Knowledge graph models -A: Class; -B: Object property; -C: Data property; -Individuals.

Based on the scenario, knowledge graph model is developed in protégé. The class refers to meta-concepts used in the scenario, for example, production line. Then the individuals of production line class are defined to represent line1 and line2.Object properties are used to represent the relationships between classes. For example, product plant (class) contains product line (class). Then as shown in Figure 4, the relationship rules are defined with object properties and another class. When developing the KG models for the scenario, each individual is connected by object property. The data property is used to define attributes of each individual and class.



Description: production plant

Equivalent To

- contain some 'production line'
- located_in some 'industrial plant site'

Property assertions: plant1

Object property assertions

- located_in Lausanne
- contain line2
- contain line1

Figure 4 Define relationship rules between classes

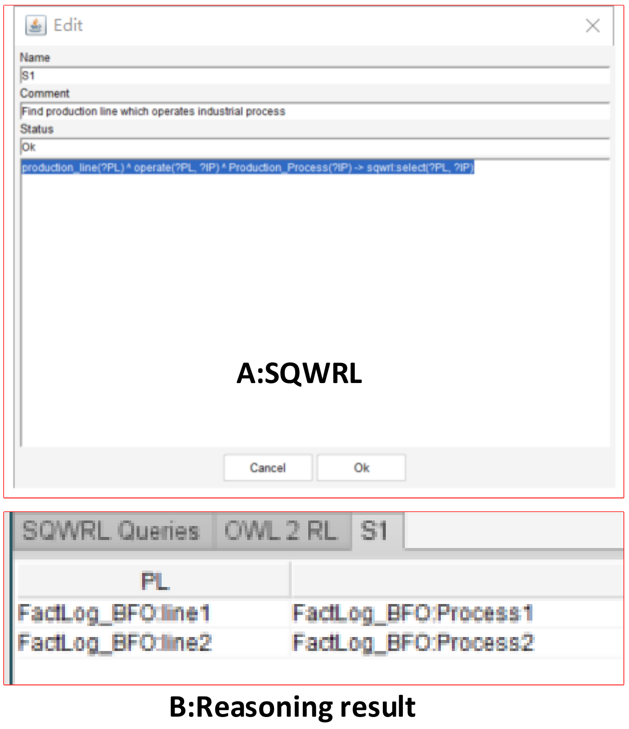
4.1.2 Reasoning for the scenario

4.1.2.1 Reasoning using SQWRL

SQWRL (Semantic Query-Enhanced Web Rule Language; pronounced squirrel) is a SWRL-based query language that provides SQL-like operators for extracting information from OWL ontologies, https://github.com/protegeproject/swrlapi/wiki/SQWRL. The SQWRL is used to reasoning the knowledge graph model in order to get the answer you want. In the scenario, when we finish the knowledge graph models, we want to know how many product lines operate production process. The SQWRL is defined as follow:

production_line(?PL) ^ operate(?PL, ?IP) ^ Production_Process(?IP) -> sqwrl:select(?PL, ?IP)

Then the reasoning is implemented as Figure 5. The results are obtained in protégé.



Edit

Name: S1

Comment: Find production line which operates industrial process

Status: Ok

```
production_line(?PL) ^ operate(?PL, ?IP) ^ Production_Process(?IP) -> sqwrl:select(?PL, ?IP)
```

SQWRL Queries | OWL 2 RL | S1

PL	
FactLog_BFO:line1	FactLog_BFO:Process1
FactLog_BFO:line2	FactLog_BFO:Process2

Figure 5: Reasoning results

4.1.2.2 Query using SPARQL

SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs https://www.w3.org/TR/rdf-sparql-query/ . The SPARQL is also used to query the knowledge graph models built by protégé. The query is implemented in twinkle. If we want to know how many workers, we have in the knowledge graph models and what are their own capabilities, we make use of the SPARQL as follow:

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX sp: <http://purl.obolibrary.org/obo/FACTLOG.owl#>

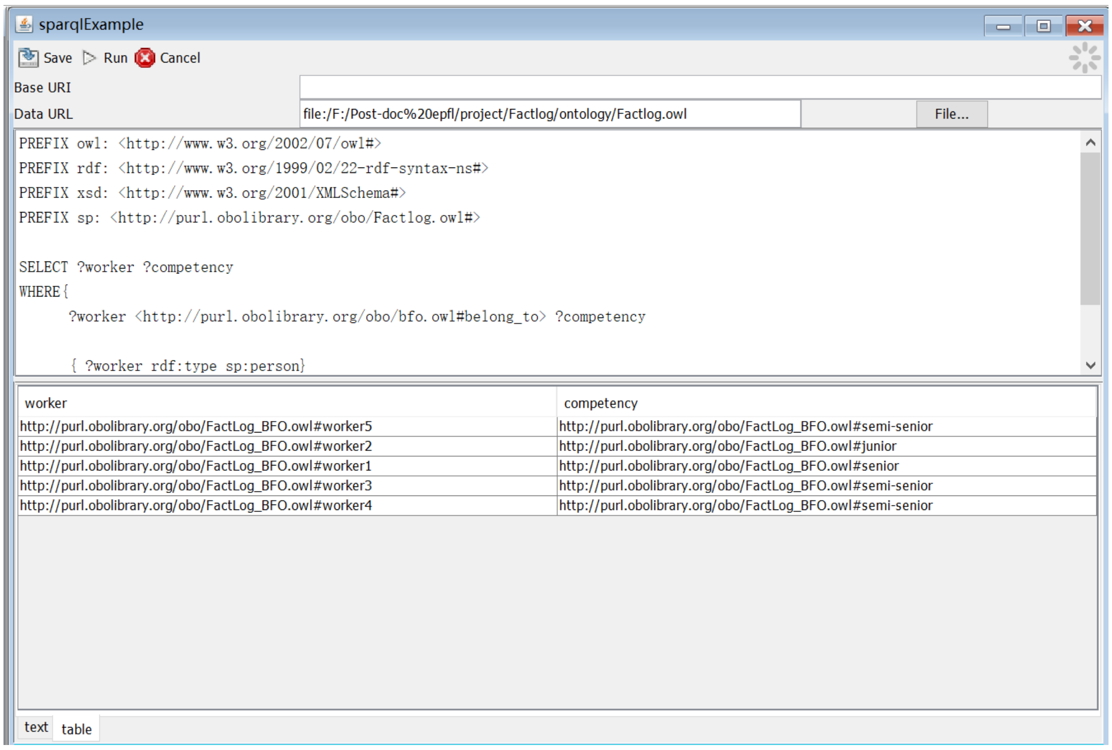
SELECT ?worker ?competency

WHERE{

?worker <http://purl.obolibrary.org/obo/bfo.owl#belong_to> ?competency

{ ?worker rdf:type sp:person}

}



Base URI

Data URL file:/F:/Post-doc%20epfl/project/Factlog/ontology/Factlog.owl

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sp: <http://purl.obolibrary.org/obo/Factlog.owl#>

SELECT ?worker ?competency
WHERE{
      ?worker <http://purl.obolibrary.org/obo/bfo.owl#belong_to> ?competency

      { ?worker rdf:type sp:person}
```

worker	competency
http://purl.obolibrary.org/obo/FactLog_BFO.owl#worker5	http://purl.obolibrary.org/obo/FactLog_BFO.owl#semi-senior
http://purl.obolibrary.org/obo/FactLog_BFO.owl#worker2	http://purl.obolibrary.org/obo/FactLog_BFO.owl#junior
http://purl.obolibrary.org/obo/FactLog_BFO.owl#worker1	http://purl.obolibrary.org/obo/FactLog_BFO.owl#senior
http://purl.obolibrary.org/obo/FactLog_BFO.owl#worker3	http://purl.obolibrary.org/obo/FactLog_BFO.owl#semi-senior
http://purl.obolibrary.org/obo/FactLog_BFO.owl#worker4	http://purl.obolibrary.org/obo/FactLog_BFO.owl#semi-senior

Figure 6: Reasoning results

4.1.2.3 Lessons learned from the Guidance

The ontology models enable to provide a unified description of the entire FACTLOG factory. Through some reasoning approaches, the knowledge graph model can support decision-makings for the FACTLOG services, such as identifying workers’ skill through query.

4.2 FACTLOG Ontology developed for KGM in FACTLOG

4.2.1 Principles

Entities in the FACTLOG semantic framework have been arranged based on the Basic Formal Ontology (BFO) which is a formal ontology framework developed by Barry Smith and his associates [3]. In BFO, there are two varieties which are continuants comprehending continuant entities such as three-dimensional enduring objects and occurrent comprehending processes conceived as extended through (or as spanning) time. To adopt BFO framework will provide availability to merge the other CT domain ontology structured by BFO.

Originated from BFO, ontology design principles of FACTLOG are as follows:

- use single nouns (except data) and avoid acronyms
- ensure unicity of terms and relational expressions
- distinguish the general from particular
- provide all non-root terms with definitions
- use essential features in defining terms and avoid circularity
- start with the most general terms in the domain
- use simpler terms than the term you are defining (to ensure intelligibility)
- do not create terms for universals through logical combination
- structure ontology around is_a hierarchy and ensure is_a completeness
- single inheritance

4.2.2 Methodology

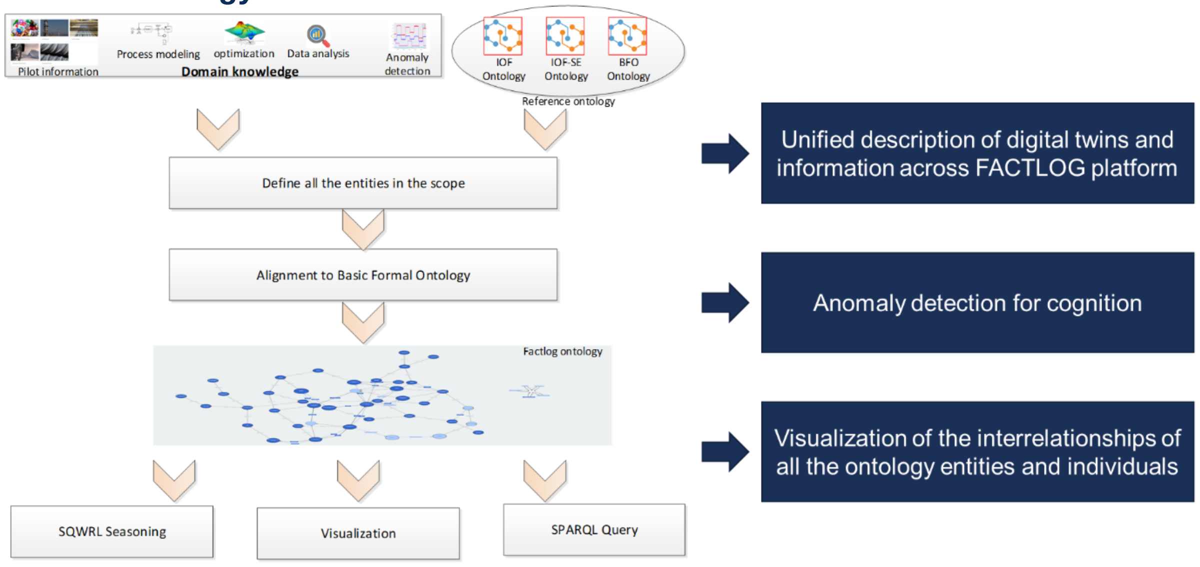

Pilot information
Process modeling
optimization
Data analysis
Anomaly detection
Domain knowledge
IOF Ontology
IOF-SE Ontology
BFO Ontology
Reference ontology
Define all the entities in the scope
Alignment to Basic Formal Ontology
Factlog ontology
SQWRL Seasoning
Visualization
SPARQL Query
Unified description of digital twins and information across FACTLOG platform
Anomaly detection for cognition
Visualization of the interrelationships of all the ontology entities and individuals

Figure 7: Ontology development workflow

In order to design the unified ontology for developing knowledge graph models supporting cognitive capabilities, the one of the well- known systems thinking development methodology (D 8.3) through domain knowledge has been applied to define the domain knowledge including: (i) pilot information; (ii) process modelling; (iii) optimization; (iv) data analysis; (v) anomaly detection. IoF ontology, BFO ontology and IoF SE ontology are three main reference ontology. By composing a top-level overview, abstract concepts form domain specific knowledge from FACTLOG pilots and technical views. After the extraction of entities from FACTLOG pilots, the list of classes was updated in a comparison with existing ontology such as IOF-SE ontology, and IOF ontology. And then, all the entities were rearranged in the BFO structure. Finally, the SQWRL and SPARQL are used to support reasoning and query of the OWL models.

All the ontology concepts are manly used for three aspects:

1. Unified description of digital twin and information across the FACTLOG platform.
2. Ontology reasoning for anomaly detection.
3. Visualization of the interrelationships of all the ontology entities and individuals.

4.2.3 Ontology Framework

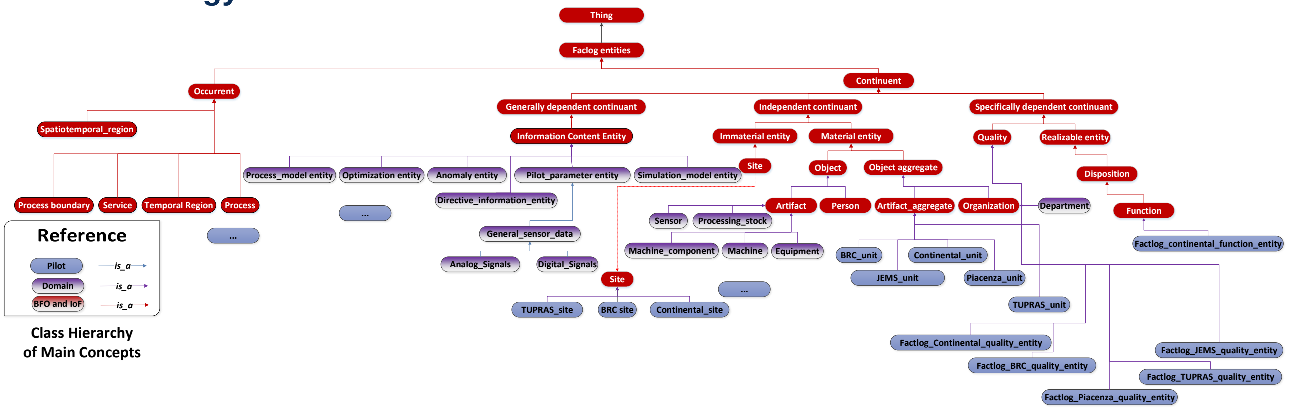


Figure 8: Ontology framework based on BFO

As shown in Figure 8, the FACTLOG ontology is developed based on basic formal ontology. The red blocks refer to BFO and IoF concepts. The purple blocks refer to domain concepts. The blue ones which are defined under BFO and domain concepts are used to define FACTLOG concepts. The whole entire FACTLOG entities include **occurrent** and **continuent** entity.

- A **continuant** is an entity that persists, endures, or continues to exist through time while maintaining its identity.
- An **occurrent** is an entity that unfolds itself in time or it is the instantaneous boundary of such an entity (for example a beginning or an ending) or it is a temporal or spatiotemporal region which such an entity occupies_temporal_region or occupies_spatiotemporal_region.

Under the occurrent entity, several concepts are defined:

- **Process**: an **occurrent** that has temporal proper parts and for some time t, p s-depends_on some material entity at t.
- **Process_boundary**: a temporal part of a process & p has no proper temporal parts.
- **Service**: Service is delivered when the service implements the system function.
- **Spatiotemporal_region**: an **occurrent** entity that is part of spacetime.
- **Temporal_region**: an **occurrent** entity that is part of time as defined relative to some reference frame.

Under the **continuant** entity, several concepts are defined:

- **generically_dependent_continuant** is a continuant that generally depends on one or more other entities.
- **independent_continuant**, a continuant which is such that there is no c and not such that b s-depends_on c at t.
- **specifically_dependent_continuant**, a continuant & there is some independent continuant c which is not a spatial region and which is such that b s-depends_on c at every time t during the course of b's existence

Under **generically_dependent_continuant** entity, several concepts are defined:

- **Information_content_entity**, a generically dependent continuant that is about something.
  - **Process_model_entity**, a virtual concept used to define a process model.
  - **Optimization_entity** , a virtual concept used to define an optimization concept.
  - **Simulation_model_entity**, a virtual concept used to define simulation model concepts.
  - **Pilot_parameter_concept**, a virtual entity to define FACTLOG pilot parameters.
    - **General sensor data**, sensor data used for all the FACTLOG pilots
  - **Directive_information_entity** , a plan specification that describes the inputs and output of mathematical functions as well as the workflow of execution for achieving a predefined objective. Algorithms are realized usually by means of implementation as computer programs for execution by automata.
  - **Anomaly_entity**, a virtual entity to support anomaly detection.
  - **Data_analysis_entity**, a virtual entity used for data analysis.
- **Specifically_dependent_continuant**, is a continuant & there is some independent continuant c which is not a spatial region and which is such that b s-depends_on c at every time t during the course of b's existence
  - **Quality**, a specifically dependent continuant that, in contrast to roles and dispositions, does not require any further process in order to be realized.
    - **FACTLOG_BRC_quality_entity**, quality used in the BRC pilot.
    - **FACTLOG_Continental_quality_entity**, quality used in the CONT pilot.
    - **FACTLOG_JEMS_quality_entity**, quality used in the JEMS pilot.
    - **FACTLOG_Piacenza_quality_entity**, quality used in the PIA pilot.
    - **FACTLOG_TUPRAS_quality_entity**, quality used in the TUPRAS pilot.
  - **realizable_entity**, a specifically dependent continuant that exists essentially or permanently in some independent continuant which is not a spatial region and is of a type instance of which are realized in processes of a correlated type.
    - **Disposition**, a realizable entity & b's bearer is some material entity & b is such that if it ceases to exist, then its bearer is physically changed, & b's realization occurs when and because this bearer is in some special physical circumstances, & this realization occurs in virtue of the bearer's physical make-up.
      - **Function**, a disposition that exists in virtue of the bearer's physical make-up and this physical make-up is something the

bearer possesses because it came into being, either through evolution (in the case of natural biological entities) or through intentional design (in the case of artifacts), in order to realize processes of a certain sort.
  - **FACTLOG_continental_function_entity**, functions used in CONT pilot.
- **Role**, a realizable entity & b exists because there is some single bearer that is in some special physical, social, or institutional set of circumstances in which this bearer does not have to be& b is not such that, if it ceases to exist, then the physical make-up of the bearer is thereby changed.

- **Independent_continuant**, a continuant which is such that there is no c and no t such that b s-depends_on c at t.
  - **immaterial_entity**, which are divided into two subgroups: boundaries and sites, which bound, or are demarcated in relation, to material entities, and which can thus change location, shape and size and as their material hosts move or change shape or size (for example: your nasal passage; the hold of a ship; the boundary of Wales.
    - Site, three-dimensional immaterial entity that is (partially or wholly) bounded by a material entity or it is a three-dimensional immaterial part thereof.
      - **BRC_site**, site used in BRC.
      - **Continental_site**, site used in CONT.
      - **TUPRAS_site**, site used in TUPRAS.
  - **material_entity**, which can preserve their identity even while gaining and losing material parts. Continuants are contrasted with occurrents, which unfold themselves in successive temporal parts or phases.
    - **Object**, a material entity which manifests causal unity of one or other of the types CUn listed above & is of a type (a material universal) instance of which are maximal relative to this criterion of causal unity.
      - Artifact, an Object that was designed by some Agent to realize a certain Function.
        - **Sensor**, a device that produces an output signal for the purpose of sensing of a physical phenomenon.
        - **Processing stock**, is an artifact in an industrial site corresponds to any material in the process of producing or manufacturing finished product.
        - **Machine component**, compositions for constructing machines.
        - **Machine**, a physical system using power to apply forces and control movement to perform an action.
        - **Equipment**, the set of physical resources serving to equip a person or thing implementing used in an operation or activity
      - **Person**, an object that is a human being.
    - **Object_aggregate**, an object aggregates if and only if there is a mutually exhaustive and pairwise disjoint partition of a into objects.
      - **Artifact_aggregate**, a collection of artifacts that designedor aranged by some Agent to realize a certain Function.

    - **BRC_unit**, a company group generally equivalent in size and character to implement BRC services.
    - **Continental_unit**, a company group generally equivalent in size and character to implement CONT services.
    - **JEMS_unit**, a company group generally equivalent in size and character to implement JEMS services.
    - **Piacenza_unit**, a company group generally equivalent in size and character to implement PIA services.
    - **TUPRAS_unit**, a company group generally equivalent in size and character to implement TUPRAS services.
- **Organization**, an object aggregate that corresponds to social institutions such as companies, societies etc. that does something.
    - **Department**, an organizational unit in FACTLOG.

4.3 Ontology for each Pilot and Technical Partner

Based on the ontology framework, ontology for each pilot and technical partners is designed in each deliverable as shown in Table 1.

Table 1 – Mapping between ontology concepts and other deliverables

Ontology concepts	Deliverable	Description
Ontology for Directive_information_entity	Deliverable 4.2	Ontology which describes the general information concepts in FACTLOG.
Ontology for Pilot Description	Deliverable 4.2	Ontology which describes the pilot concepts in FACTLOG.
Ontology for Optimization	Deliverable 4.2	Ontology which describes the optimization concepts in FACTLOG.
Ontology for Processing Modeling	Deliverable 4.2	Ontology which describes the process model concepts in FACTLOG.
Ontology for Data Analysis	Deliverable 4.4	Ontology which describes the concepts of data analysis in FACTLOG.
Ontology for Simulation Model	Deliverable 4.2	Ontology which describes the concepts of simulation models in FACTLOG.

4.4 Ontology for Anomaly Detection

The ontology concepts are defined to support anomaly detection in FACTLOG pilot.

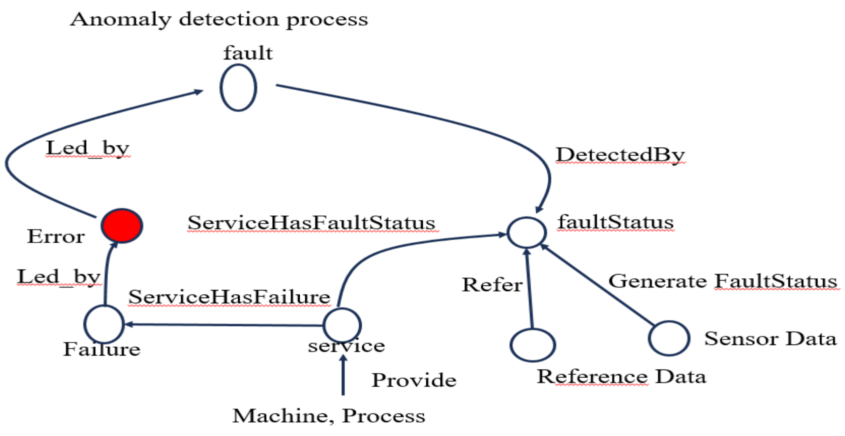


Figure 9: Ontology for describing anomaly detection processes

As shown in Figure 9, several concepts are defined to describe the anomaly detection process.

1. A **service** is defined as what is delivered when the service implements the system function.
2. A **service failure**, often abbreviated here as failure, is an event that occurs when the delivered service deviates from correct service.
3. An **error** is the deviation that at least one (or more) external state of the system deviates from the correct service state, since a service is a sequence of the system's external states.
4. A **fault** is an adjudged or hypothesized cause of an error.
5. An **observation** of a fault is fault status.
6. **Sensor data** refers to the data directly generated from machine or sensor.
7. **Reference data** refers to the data from simulation models for decision-makings.

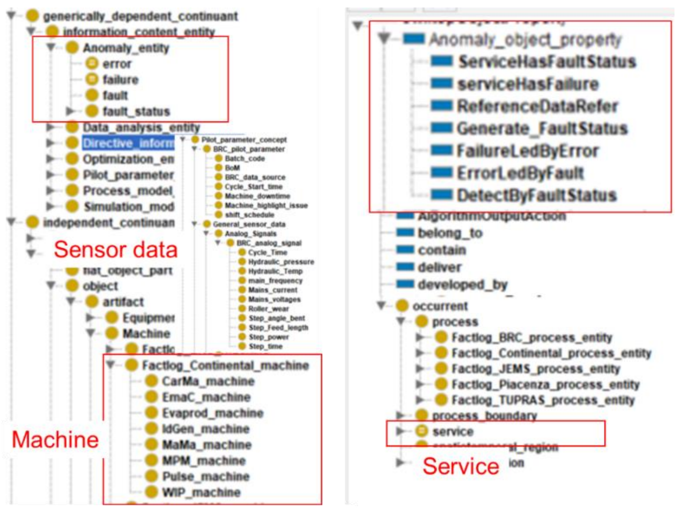


Figure 10: Ontology classes for describing anomaly detection processes

As shown in Figure 10, ontology classes are defined based on the Figure 9.

5 Knowledge Graph Modelling based on FACTLOG Ontology

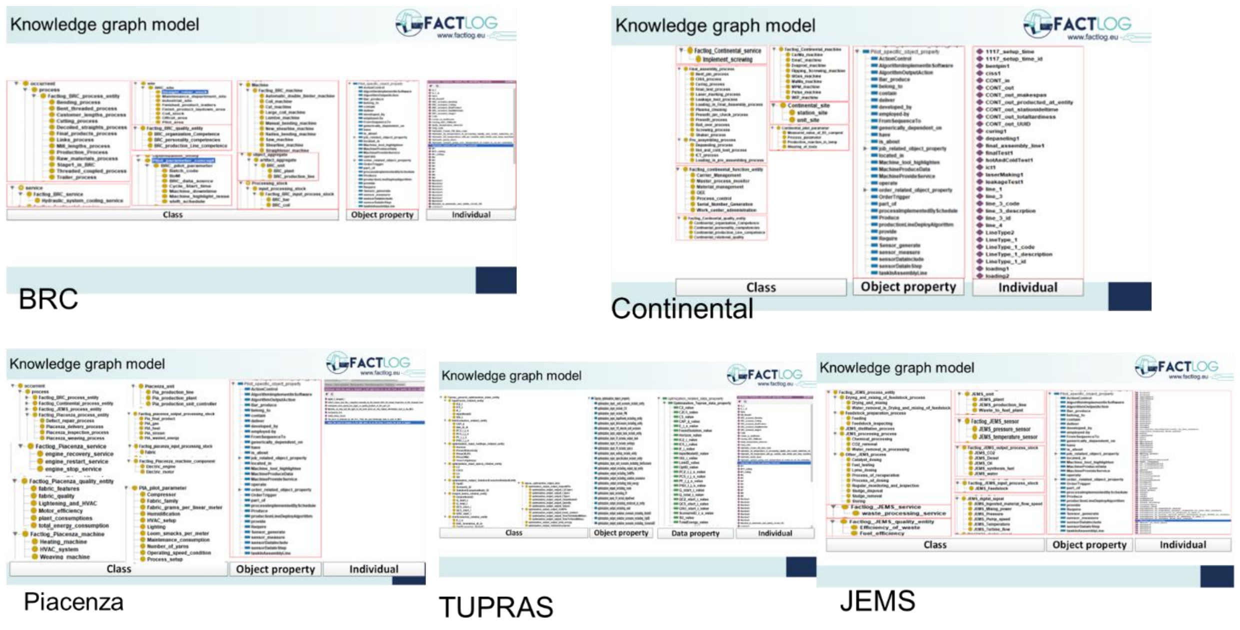


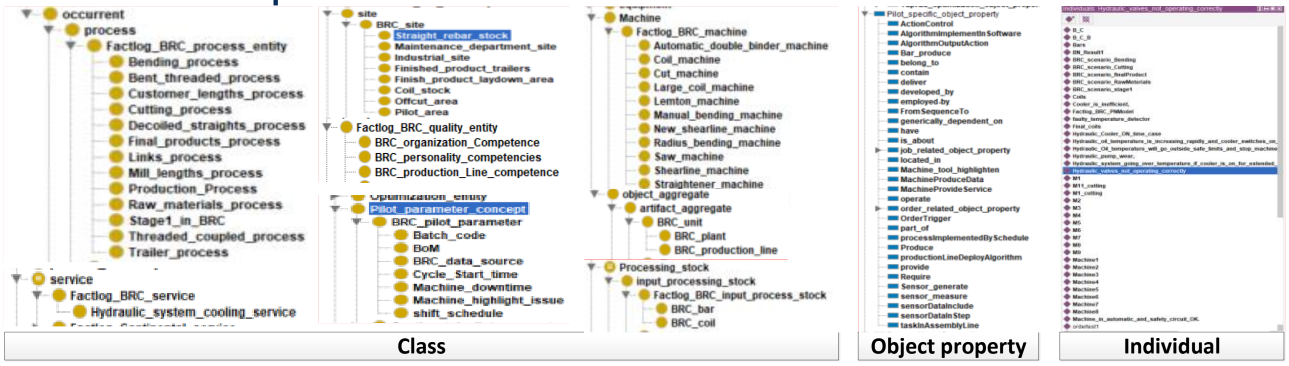
Figure 11: Knowledge graph models based on developed ontology

As shown in Figure 11, knowledge graph models are developed based on the ontology in Section 4. Details are shown in Deliverable 4.2.

5.1 Knowledge Graph Modelling for BRC

In the BRC pilot, four main concerns are first considered when knowledge graph models are developed to formalize the BRC pilot: 1) pilot description; 2) PN model formalism; 3) optimization formalism; 4) anomaly detection (introduced in Section 6.4.1.2).

5.1.1 Pilot description



```
occurrent
  process
    Factlog_BRC_process_entity
      Bending_process
      Bent_threaded_process
      Customer_lengths_process
      Cutting_process
      Decoiled_straights_process
      Final_products_process
      Links_process
      Mill_lengths_process
      Production_Process
      Raw_materials_process
      Stage1_in_BRC
      Threaded_coupled_process
      Trailer_process
service
  Factlog_BRC_service
    Hydraulic_system_cooling_service

site
  BRC_site
    Straight_rebar_stock
    Maintenance_department_site
    Industrial_site
    Finished_product_trailers
    Finish_product_laydown_area
    Coil_stock
    Offcut_area
    Pilot_area
Factlog_BRC_quality_entity
  BRC_organization_Competence
  BRC_personality_competencies
  BRC_production_Line_competence
Optimization_entity
  Pilot_parameter_concept
    BRC_pilot_parameter
      Batch_code
      BoM
      BRC_data_source
      Cycle_Start_time
      Machine_downtime
      Machine_highlight_issue
      shift_schedule

Machine
  Factlog_BRC_machine
    Automatic_double_binder_machine
    Coil_machine
    Cut_machine
    Large_coil_machine
    Lemton_machine
    Manual_bending_machine
    New_shearline_machine
    Radius_bending_machine
    Saw_machine
    Shearline_machine
    Straightener_machine
object_aggregate
  artifact_aggregate
    BRC_unit
      BRC_plant
      BRC_production_line
Processing_stock
  input_processing_stock
    Factlog_BRC_input_process_stock
      BRC_bar
      BRC_coil
```

Class

```
Pilot_specific_object_property
  ActionControl
  AlgorithmImplementInSoftware
  AlgorithmOutputAction
  Bar_produce
  belong_to
  contain
  deliver
  developed_by
  employed-by
  FromSequenceTo
  generically_dependent_on
  have
  is_about
  job_related_object_property
  located_in
  Machine_tool_highlighten
  MachineProduceData
  MachineProvideService
  operate
  order_related_object_property
  OrderTrigger
  part_of
  processImplementedBySchedule
  Produce
  productionLineDeployAlgorithm
  provide
  Require
  Sensor_generate
  sensor_measure
  sensorDataInclude
  sensorDataInStep
  taskInAssemblyLine
```

Object property

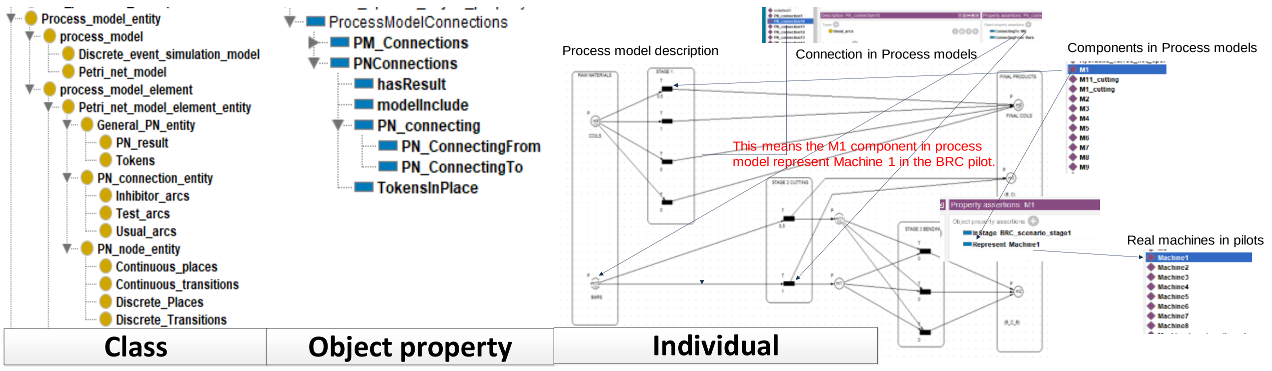
```
Individuals: Hydraulic_valves_not_operating_correctly
B_C
B_C_B
Bars
BN_Result1
BRC_scenario_Bending
BRC_scenario_Cutting
BRC_scenario_finalProduct
BRC_scenario_RawMeterials
BRC_scenario_stage1
Coils
Cooler_is_inefficient.
Factlog_BRC_PNModel
faulty_temperature_detector
Final_coils
Hydraulic_Cooler_ON_time_case
Hydraulic_oil_temperature_is_increasing_rapidly_and_cooler_switches_on
Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine
Hydraulic_pump_wear.
Hydraulic_system_going_over_temperature_if_cooler_is_on_for_extended
Hydraulic_valves_not_operating_correctly
M1
M11_cutting
M1_cutting
M2
M3
M4
M5
M6
M7
M8
M9
Machine1
Machine2
Machine3
Machine4
Machine5
Machine6
Machine7
Machine8
Machine_in_automatic_and_safety_circuit_OK.
ordertest1
```

Individual

Figure 12: Ontology class， object property and individuals for BRC pilot description

As shown in Figure 12, BRC process entities, BRC services, BRC sites, BRC pilot parameter, BRC machine, BRC unit and BRC input process stock are the main ontology concepts defined in the knowledge graph models. General specific object properties are defined in order to support all the pilot description. Individuals are defined to describe an example of the BRC scenario.

5.1.2 PN model formalism



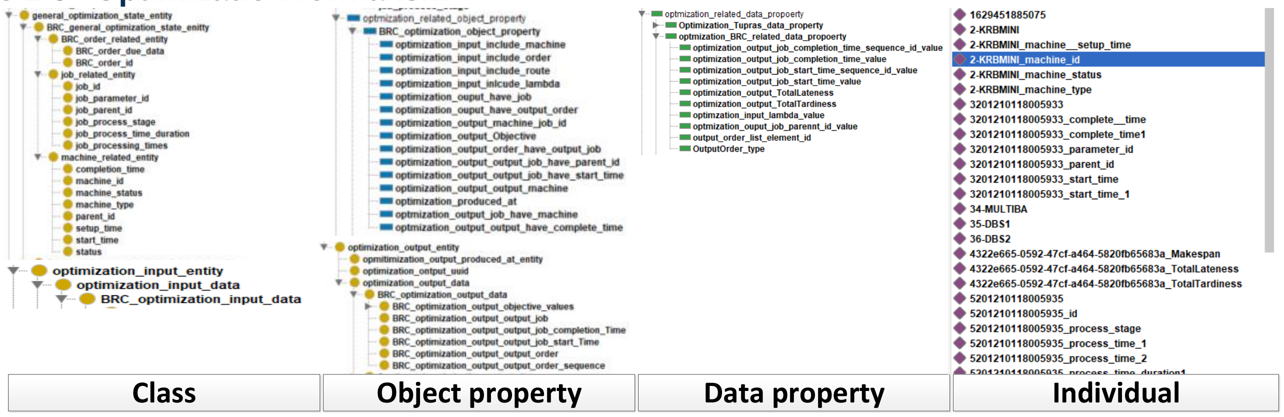
```
Process_model_entity
    process_model
        Discrete_event_simulation_model
        Petri_net_model
    process_model_element
        Petri_net_model_element_entity
            General_PN_entity
                PN_result
                Tokens
            PN_connection_entity
                Inhibitor_arcs
                Test_arcs
                Usual_arcs
            PN_node_entity
                Continuous_places
                Continuous_transitions
                Discrete_Places
                Discrete_Transitions
```

```
ProcessModelConnections
    PM_Connections
    PNConnections
        hasResult
        modelInclude
        PN_connecting
            PN_ConnectingFrom
            PN_ConnectingTo
        TokensInPlace
```

Figure 13: Ontology class， object property and individuals for simulation in BRC pilot description

As shown in Figure 14，classes of petri net model entities and object properties of petri net model connections are defined to describe the petri net model used in the BRC pilot. The PN node entities are used to represent the “model compositions” in the petri net model. The PN connection entities are used to represent the model connections in the petri net model. PN_connectingFrom and PN_connectingTo are used to connect the PN connections and PN nodes. Through this way, the entire PN model is described.

5.1.3 Optimization formalism



```
general_optimization_state_entity
    BRC_general_optimization_state_enitty
        BRC_order_related_entity
            BRC_order_due_data
            BRC_order_id
        job_related_entity
            job_id
            job_parameter_id
            job_parent_id
            job_process_stage
            job_process_time_duration
            job_processing_times
        machine_related_entity
            completion_time
            machine_id
            machine_status
            machine_type
            parent_id
            setup_time
            start_time
            status
optimization_input_entity
    optimization_input_data
        BRC_optimization_input_data
```

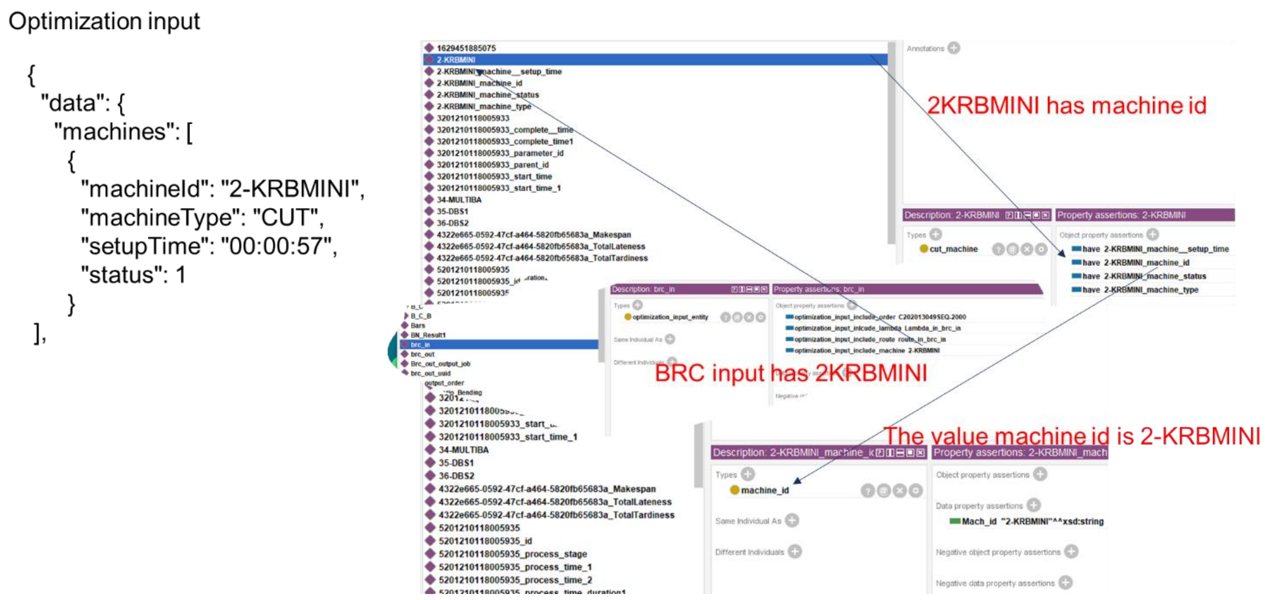
```
optmization_related_object_property
    BRC_optimization_object_property
        optimization_input_include_machine
        optimization_input_include_order
        optimization_input_include_route
        optimization_input_inlcude_lambda
        optimization_ouput_have_job
        optimization_ouput_have_output_order
        optimization_output_machine_job_id
        optimization_output_Objective
        optimization_output_order_have_output_job
        optimization_output_output_job_have_parent_id
        optimization_output_output_job_have_start_time
        optimization_output_output_machine
        optimization_produced_at
        optmization_output_job_have_machine
        optmization_output_output_have_complete_time
optimization_output_entity
    opmitimization_output_produced_at_entity
    optimization_ontput_uuid
    optimization_output_data
        BRC_optimization_output_data
            BRC_optimization_output_objective_values
            BRC_optimization_output_output_job
            BRC_optimization_output_output_job_completion_Time
            BRC_optimization_output_output_job_start_Time
            BRC_optimization_output_output_order
            BRC_optimization_output_output_order_sequence
```

```
optmization_related_data_propoerty
    Optimization_Tupras_data_property
    optmization_BRC_related_data_propoerty
        optimization_output_job_completion_time_sequence_id_value
        optimization_output_job_completion_time_value
        optimization_output_job_start_time_sequence_id_value
        optimization_output_job_start_time_value
        optimization_output_TotalLateness
        optimization_output_TotalTardiness
        optimzation_input_lambda_value
        optmization_ouput_job_parennt_id_value
        output_order_list_element_id
        OutputOrder_type
```

```
1629451885075
2-KRBMINI
2-KRBMINI_machine__setup_time
2-KRBMINI_machine_id
2-KRBMINI_machine_status
2-KRBMINI_machine_type
3201210118005933
3201210118005933_complete__time
3201210118005933_complete_time1
3201210118005933_parameter_id
3201210118005933_parent_id
3201210118005933_start_time
3201210118005933_start_time_1
34-MULTIBA
35-DBS1
36-DBS2
4322e665-0592-47cf-a464-5820fb65683a_Makespan
4322e665-0592-47cf-a464-5820fb65683a_TotalLateness
4322e665-0592-47cf-a464-5820fb65683a_TotalTardiness
5201210118005935
5201210118005935_id
5201210118005935_process_stage
5201210118005935_process_time_1
5201210118005935_process_time_2
```

Figure 14: Ontology class, object property, data property and individuals for optimization for BRC pilot

As shown in Figure 14, optimization classes, object properties and data properties are defined to represent optimization scenario using individuals. In Figure 15, the individuals are used to represent the optimization input data structure. The BRC optimization input has a machine named “2KRBMINI” who has a machine id as 2-KRBMINI.


Optimization input

```
{
  "data": {
    "machines": [
      {
        "machineId": "2-KRBMINI",
        "machineType": "CUT",
        "setupTime": "00:00:57",
        "status": 1
      }
    ],
```

```
1629451885075
2-KRBMINI
2-KRBMINI_machine__setup_time
2-KRBMINI_machine_id
2-KRBMINI_machine_status
2-KRBMINI_machine_type
3201210118005933
3201210118005933_complete__time
3201210118005933_complete_time1
3201210118005933_parameter_id
3201210118005933_parent_id
3201210118005933_start_time
3201210118005933_start_time_1
34-MULTIBA
35-DBS1
36-DBS2
4322e665-0592-47cf-a464-5820fb65683a_Makespan
4322e665-0592-47cf-a464-5820fb65683a_TotalLateness
4322e665-0592-47cf-a464-5820fb65683a_TotalTardiness
5201210118005935
5201210118005935_id
5201210118005935
B_C_B
Bars
BN_Result1
brc_in
brc_out
Brc_out_output_job
brc_out_uuid
output_order
3201210118005933_start_time_1
34-MULTIBA
35-DBS1
36-DBS2
4322e665-0592-47cf-a464-5820fb65683a_Makespan
4322e665-0592-47cf-a464-5820fb65683a_TotalLateness
4322e665-0592-47cf-a464-5820fb65683a_TotalTardiness
5201210118005935
5201210118005935_id
5201210118005935_process_stage
5201210118005935_process_time_1
5201210118005935_process_time_2
5201210118005935_process_time_duration1
```

Description: 2-KRBMINI
Types: cut_machine

Property assertions: 2-KRBMINI
Object property assertions:
have 2-KRBMINI_machine__setup_time
have 2-KRBMINI_machine_id
have 2-KRBMINI_machine_status
have 2-KRBMINI_machine_type

Description: brc_in
Types: optimization_input_entity

Property assertions: brc_in
Object property assertions:
optimization_input_include_order C202013049SEQ-2000
optimization_input_inlcude_lambda Lambda_in_brc_in
optimization_input_include_route route_in_brc_in
optimization_input_include_machine 2-KRBMINI

Description: 2-KRBMINI_machine_id
Types: machine_id

Property assertions: 2-KRBMINI_mach
Data property assertions:
Mach_id "2-KRBMINI"^^xsd:string

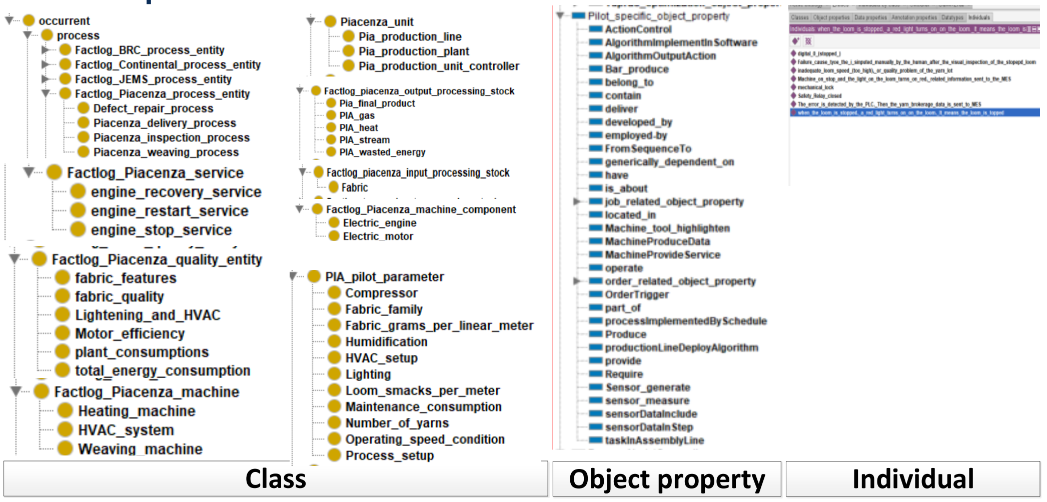
2KRBMINI has machine id
BRC input has 2KRBMINI
The value machine id is 2-KRBMINI
Annotations
Same Individual As
Different Individuals
Negative object property assertions
Negative data property assertions

Figure 15: Individuals for describing the optimization scenario for BRC pilot

5.2 Knowledge Graph Modelling for PIA

In the PIA pilot, three concerns are first considered when knowledge graph models are developed to formalize the PIA pilot: 1) pilot description; 2) optimization formalism; 3) anomaly detection (introduced in Section 6.4.1.3).

5.2.1 Pilot description


Class

```
occurrent
  process
    Factlog_BRC_process_entity
    Factlog_Continental_process_entity
    Factlog_JEMS_process_entity
    Factlog_Piacenza_process_entity
      Defect_repair_process
      Piacenza_delivery_process
      Piacenza_inspection_process
      Piacenza_weaving_process
Factlog_Piacenza_service
  engine_recovery_service
  engine_restart_service
  engine_stop_service
Factlog_Piacenza_quality_entity
  fabric_features
  fabric_quality
  Lightening_and_HVAC
  Motor_efficiency
  plant_consumptions
  total_energy_consumption
Factlog_Piacenza_machine
  Heating_machine
  HVAC_system
  Weaving_machine
Piacenza_unit
  Pia_production_line
  Pia_production_plant
  Pia_production_unit_controller
Factlog_piacenza_output_processing_stock
  Pia_final_product
  PIA_gas
  PIA_heat
  PIA_stream
  PIA_wasted_energy
Factlog_piacenza_input_processing_stock
  Fabric
Factlog_Piacenza_machine_component
  Electric_engine
  Electric_motor
PIA_pilot_parameter
  Compressor
  Fabric_family
  Fabric_grams_per_linear_meter
  Humidification
  HVAC_setup
  Lighting
  Loom_smacks_per_meter
  Maintenance_consumption
  Number_of_yarns
  Operating_speed_condition
  Process_setup
```

Object property

```
Pilot_specific_object_property
  ActionControl
  AlgorithmImplementInSoftware
  AlgorithmOutputAction
  Bar_produce
  belong_to
  contain
  deliver
  developed_by
  employed-by
  FromSequenceTo
  generically_dependent_on
  have
  is_about
  job_related_object_property
  located_in
  Machine_tool_highlighten
  MachineProduceData
  MachineProvideService
  operate
  order_related_object_property
  OrderTrigger
  part_of
  processImplementedBySchedule
  Produce
  productionLineDeployAlgorithm
  provide
  Require
  Sensor_generate
  sensor_measure
  sensorDataInclude
  sensorDataInStep
  taskInAssemblyLine
```

Individual

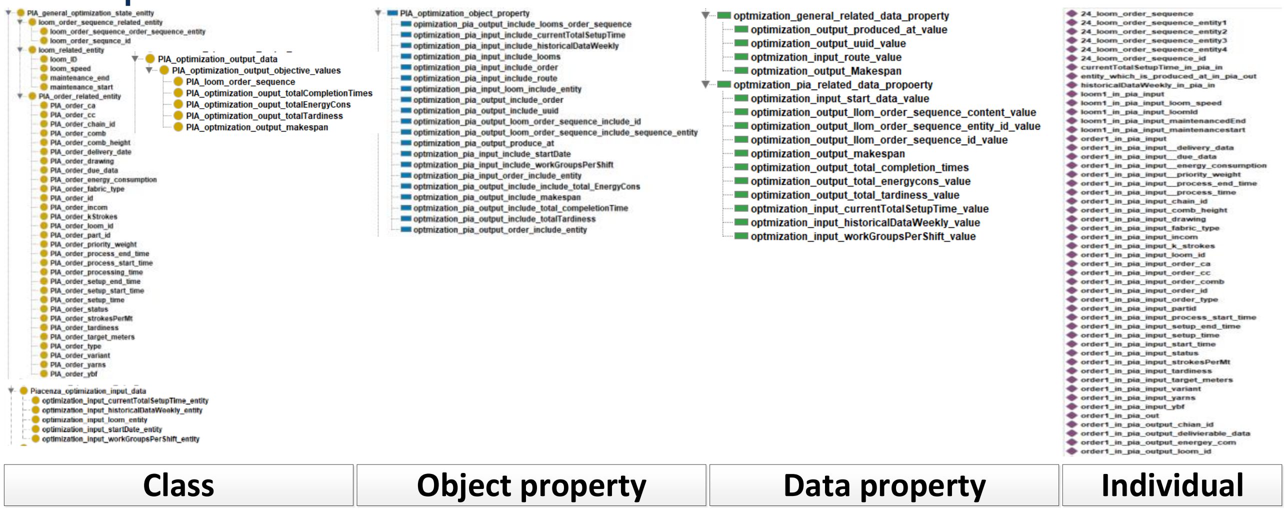
Individuals: when the loom is stopped, a red light turns on on the loom, it means the loom is

```
digital_8_(stopped_)
Failure_cause_tyoe_the_i_sinputed_manually_by_the_human_after_the_visual_inspection_of_the_stopepd_loom
inadequate_loom_speed_(too_high)_or_quality_problem_of_the_yarn_lot
Machine_on_stop_and_the_light_on_the_loom_turns_on_red,_related_information_sent_to_the_MES
mechanical_lock
Safety_Relay_closed
The_error_is_detected_by_the_PLC._Then_the_yarn_brokerage_data_is_sent_to_MES
when_the_loom_is_stopped,_a_red_light_turns_on_on_the_loom,_it_means_the_loom_is_topped
```

Figure 16: Ontology class, object property and individuals for PIA pilot description

As shown in Figure 16, PIA process entities, PIA services, PIA pilot parameter, PIA machine, PIA unit, PIA input process stock and PIA output process stock are the main ontology concepts defined in the knowledge graph models. General specific object properties are defined in order to support all the pilot description. Individuals are defined to describe an example of the PIA scenario.

5.2.2 Optimization formalism



```
PIA_general_optimization_state_enitty
  loom_order_sequence_related_entity
    loom_order_sequence_order_sequence_entity
    loom_order_sequnce_id
  loom_related_entity
    loom_ID
    loom_speed
    maintenance_end
    maintenance_start
  PIA_order_related_entity
    PIA_order_ca
    PIA_order_cc
    PIA_order_chain_id
    PIA_order_comb
    PIA_order_comb_height
    PIA_order_delivery_date
    PIA_order_drawing
    PIA_order_due_data
    PIA_order_energy_consumption
    PIA_order_fabric_type
    PIA_order_id
    PIA_order_incom
    PIA_order_kStrokes
    PIA_order_loom_id
    PIA_order_part_id
    PIA_order_priority_weight
    PIA_order_process_end_time
    PIA_order_process_start_time
    PIA_order_processing_time
    PIA_order_setup_end_time
    PIA_order_setup_start_time
    PIA_order_setup_time
    PIA_order_status
    PIA_order_strokesPerMt
    PIA_order_tardiness
    PIA_order_target_meters
    PIA_order_type
    PIA_order_variant
    PIA_order_yarns
    PIA_order_ybf
Piacenza_optimization_input_data
  optimization_input_currentTotalSetupTime_entity
  optimization_input_historicalDataWeekly_entity
  optimization_input_loom_entity
  optimization_input_startDate_entity
  optimization_input_workGroupsPerShift_entity

PIA_optimization_output_data
  PIA_optimization_output_objective_values
    PIA_loom_order_sequence
    PIA_optimization_ouput_totalCompletionTimes
    PIA_optimization_ouput_totalEnergyCons
    PIA_optimization_ouput_totalTardiness
    PIA_optmization_output_makespan
```

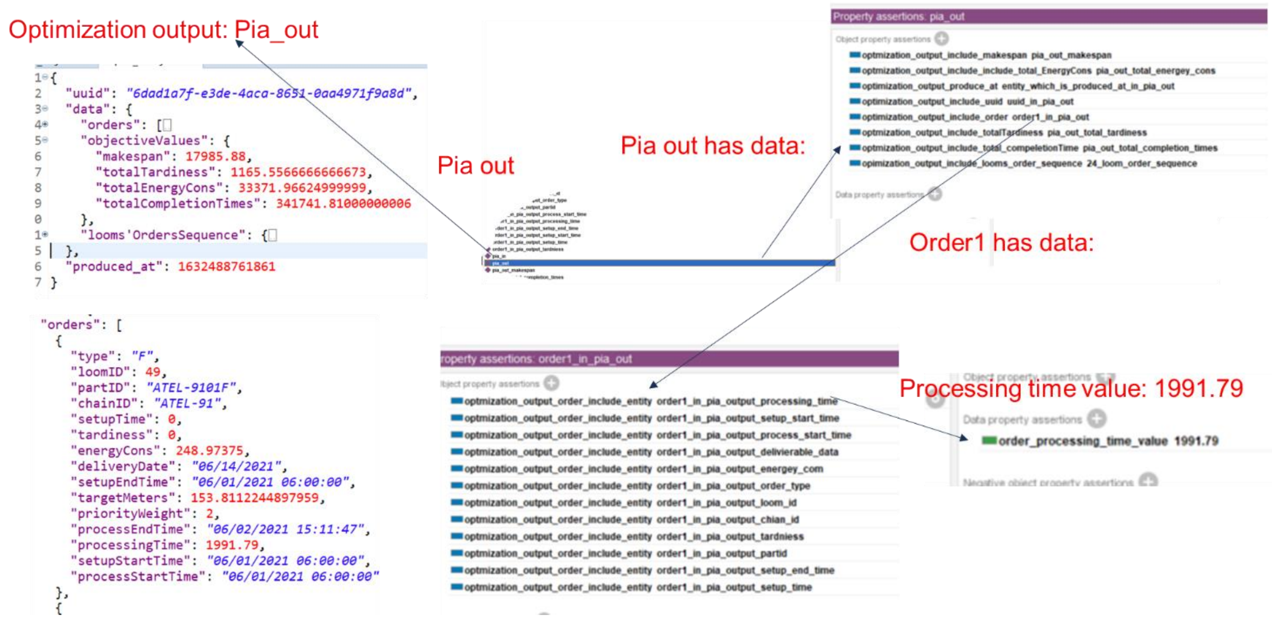
```
PIA_optimization_object_property
  opimization_pia_output_include_looms_order_sequence
  optimization_pia_input_include_currentTotalSetupTime
  optimization_pia_input_include_historicalDataWeekly
  optimization_pia_input_include_looms
  optimization_pia_input_include_order
  optimization_pia_input_include_route
  optimization_pia_input_loom_include_entity
  optimization_pia_output_include_order
  optimization_pia_output_include_uuid
  optimization_pia_output_loom_order_sequence_include_id
  optimization_pia_output_loom_order_sequence_include_sequence_entity
  optimization_pia_output_produce_at
  optmization_pia_input_include_startDate
  optmization_pia_input_include_workGroupsPerShift
  optmization_pia_input_order_include_entity
  optmization_pia_output_include_include_total_EnergyCons
  optmization_pia_output_include_makespan
  optmization_pia_output_include_total_compeletionTime
  optmization_pia_output_include_totalTardiness
  optmization_pia_output_order_include_entity
```

```
optmization_general_related_data_property
  optimization_output_produced_at_value
  optimization_output_uuid_value
  optmization_input_route_value
  optmization_output_Makespan
optmization_pia_related_data_propoerty
  optimization_input_start_data_value
  optimization_output_llom_order_sequence_content_value
  optimization_output_llom_order_sequence_entity_id_value
  optimization_output_llom_order_sequence_id_value
  optimization_output_makespan
  optimization_output_total_completion_times
  optimization_output_total_energycons_value
  optimization_output_total_tardiness_value
  optmization_input_currentTotalSetupTime_value
  optmization_input_historicalDataWeekly_value
  optmization_input_workGroupsPerShift_value
```

```
24_loom_order_sequence
24_loom_order_sequence_entity1
24_loom_order_sequence_entity2
24_loom_order_sequence_entity3
24_loom_order_sequence_entity4
24_loom_order_sequence_id
currentTotalSetupTime_in_pia_in
entity_which_is_produced_at_in_pia_out
historicalDataWeekly_in_pia_in
loom1_in_pia_input
loom1_in_pia_input_loom_speed
loom1_in_pia_input_loomId
loom1_in_pia_input_maintenancedEnd
loom1_in_pia_input_maintenancestart
order1_in_pia_input
order1_in_pia_input__delivery_data
order1_in_pia_input__due_data
order1_in_pia_input__energy_consumption
order1_in_pia_input__priority_weight
order1_in_pia_input__process_end_time
order1_in_pia_input__process_time
order1_in_pia_input_chain_id
order1_in_pia_input_comb_height
order1_in_pia_input_drawing
order1_in_pia_input_fabric_type
order1_in_pia_input_incom
order1_in_pia_input_k_strokes
order1_in_pia_input_loom_id
order1_in_pia_input_order_ca
order1_in_pia_input_order_cc
order1_in_pia_input_order_comb
order1_in_pia_input_order_id
order1_in_pia_input_order_type
order1_in_pia_input_partid
order1_in_pia_input_process_start_time
order1_in_pia_input_setup_end_time
order1_in_pia_input_setup_time
order1_in_pia_input_start_time
order1_in_pia_input_status
order1_in_pia_input_strokesPerMt
order1_in_pia_input_tardiness
order1_in_pia_input_target_meters
order1_in_pia_input_variant
order1_in_pia_input_yarns
order1_in_pia_input_ybf
order1_in_pia_out
order1_in_pia_output_chian_id
order1_in_pia_output_delivierable_data
order1_in_pia_output_energey_com
order1_in_pia_output_loom_id
```

Figure 17: Ontology class， object property, data property and individuals for optimization for PIA pilot

As shown in Figure 17, optimization classes, object properties and data properties are defined to represent optimization scenario using individuals. In Figure 18, the individuals are used to represent the optimization input data structure. The PIA optimization output has an order1 named “order1” who has a processing time and its value is 1991.79.



```
{
  "uuid": "6dad1a7f-e3de-4aca-8651-0aa4971f9a8d",
  "data": {
    "orders": [□
    "objectiveValues": {
      "makespan": 17985.88,
      "totalTardiness": 1165.5566666666673,
      "totalEnergyCons": 33371.96624999999,
      "totalCompletionTimes": 341741.81000000006
    },
    "looms'OrdersSequence": {□
  },
  "produced_at": 1632488761861
}
```

```
"orders": [
  {
    "type": "F",
    "loomID": 49,
    "partID": "ATEL-9101F",
    "chainID": "ATEL-91",
    "setupTime": 0,
    "tardiness": 0,
    "energyCons": 248.97375,
    "deliveryDate": "06/14/2021",
    "setupEndTime": "06/01/2021 06:00:00",
    "targetMeters": 153.8112244897959,
    "priorityWeight": 2,
    "processEndTime": "06/02/2021 15:11:47",
    "processingTime": 1991.79,
    "setupStartTime": "06/01/2021 06:00:00",
    "processStartTime": "06/01/2021 06:00:00"
  },
  {
```

```
order1_in_pia_output_tardniess
pia_in
pia_out
pia_out_makespan
```

Property assertions: pia_out

Object property assertions

```
optmization_output_include_makespan pia_out_makespan
optmization_output_include_include_total_EnergyCons pia_out_total_energey_cons
optimization_output_produce_at entity_which_is_produced_at_in_pia_out
optimization_output_include_uuid uuid_in_pia_out
optimization_output_include_order order1_in_pia_out
optmization_output_include_totalTardiness pia_out_total_tardiness
optmization_output_include_total_compeletionTime pia_out_total_completion_times
opimization_output_include_looms_order_sequence 24_loom_order_sequence
```

Data property assertions

Property assertions: order1_in_pia_out

Object property assertions

```
optmization_output_order_include_entity order1_in_pia_output_processing_time
optmization_output_order_include_entity order1_in_pia_output_setup_start_time
optmization_output_order_include_entity order1_in_pia_output_process_start_time
optmization_output_order_include_entity order1_in_pia_output_delivierable_data
optmization_output_order_include_entity order1_in_pia_output_energey_com
optmization_output_order_include_entity order1_in_pia_output_order_type
optmization_output_order_include_entity order1_in_pia_output_loom_id
optmization_output_order_include_entity order1_in_pia_output_chian_id
optmization_output_order_include_entity order1_in_pia_output_tardniess
optmization_output_order_include_entity order1_in_pia_output_partid
optmization_output_order_include_entity order1_in_pia_output_setup_end_time
optmization_output_order_include_entity order1_in_pia_output_setup_time
```

Object property assertions

Data property assertions

```
order_processing_time_value 1991.79
```

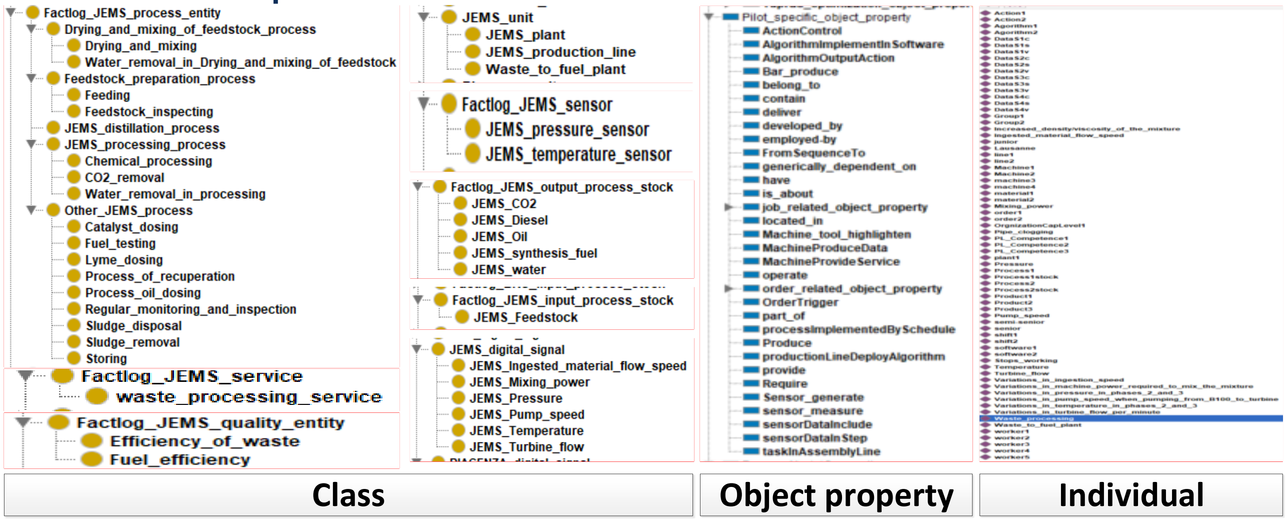
Negative object property assertions

Figure 18: Individuals for describing the optimization scenario for PIA pilot

5.3 Knowledge Graph Modelling for JEMS

In the JEMS pilot[1], two concerns are first considered when knowledge graph models are developed to formalize the pilot: 1) pilot description; 2) anomaly detection (introduced in Section **Errore. L'origine riferimento non è stata trovata.**).

5.3.1 Pilot description



```
Factlog_JEMS_process_entity
    Drying_and_mixing_of_feedstock_process
        Drying_and_mixing
        Water_removal_in_Drying_and_mixing_of_feedstock
    Feedstock_preparation_process
        Feeding
        Feedstock_inspecting
    JEMS_distillation_process
    JEMS_processing_process
        Chemical_processing
        CO2_removal
        Water_removal_in_processing
    Other_JEMS_process
        Catalyst_dosing
        Fuel_testing
        Lyme_dosing
        Process_of_recuperation
        Process_oil_dosing
        Regular_monitoring_and_inspection
        Sludge_disposal
        Sludge_removal
        Storing
Factlog_JEMS_service
    waste_processing_service
Factlog_JEMS_quality_entity
    Efficiency_of_waste
    Fuel_efficiency

JEMS_unit
    JEMS_plant
    JEMS_production_line
    Waste_to_fuel_plant
Factlog_JEMS_sensor
    JEMS_pressure_sensor
    JEMS_temperature_sensor
Factlog_JEMS_output_process_stock
    JEMS_CO2
    JEMS_Diesel
    JEMS_Oil
    JEMS_synthesis_fuel
    JEMS_water
Factlog_JEMS_input_process_stock
    JEMS_Feedstock
JEMS_digital_signal
    JEMS_Ingested_material_flow_speed
    JEMS_Mixing_power
    JEMS_Pressure
    JEMS_Pump_speed
    JEMS_Temperature
    JEMS_Turbine_flow

Pilot_specific_object_property
    ActionControl
    AlgorithmImplementInSoftware
    AlgorithmOutputAction
    Bar_produce
    belong_to
    contain
    deliver
    developed_by
    employed-by
    FromSequenceTo
    generically_dependent_on
    have
    is_about
    job_related_object_property
    located_in
    Machine_tool_highlighten
    MachineProduceData
    MachineProvideService
    operate
    order_related_object_property
    OrderTrigger
    part_of
    processImplementedBySchedule
    Produce
    productionLineDeployAlgorithm
    provide
    Require
    Sensor_generate
    sensor_measure
    sensorDataInclude
    sensorDataInStep
    taskInAssemblyLine

Action1
Action2
Agorithm1
Agorithm2
DataS1c
DataS1s
DataS1v
DataS2c
DataS2s
DataS2v
DataS3c
DataS3s
DataS3v
DataS4c
DataS4s
DataS4v
Group1
Group2
Increased_density/viscosity_of_the_mixture
Ingested_material_flow_speed
junior
Lausanne
line1
line2
Machine1
Machine2
machine3
machine4
material1
material2
Mixing_power
order1
order2
OrgnizationCapLevel1
Pipe_clogging
PL_Competence1
PL_Competence2
PL_Competence3
plant1
Pressure
Process1
Process1stock
Process2
Process2stock
Product1
Product2
Product3
Pump_speed
semi-senior
senior
shift1
shift2
software1
software2
Stops_working
Temperature
Turbine_flow
Variations_in_ingestion_speed
Variations_in_machine_power_required_to_mix_the_mixture
Variations_in_pressure_in_phases_2_and_3
Variations_in_pump_speed_when_pumping_from_B100_to_turbine
Variations_in_temperature_in_phases_2_and_3
Variations_in_turbine_flow_per_minute
Waste_processing
Waste_to_fuel_plant
worker1
worker2
worker3
worker4
worker5
```

Figure 19: Ontology class, object property, data property and individuals for optimization for JEMS pilot

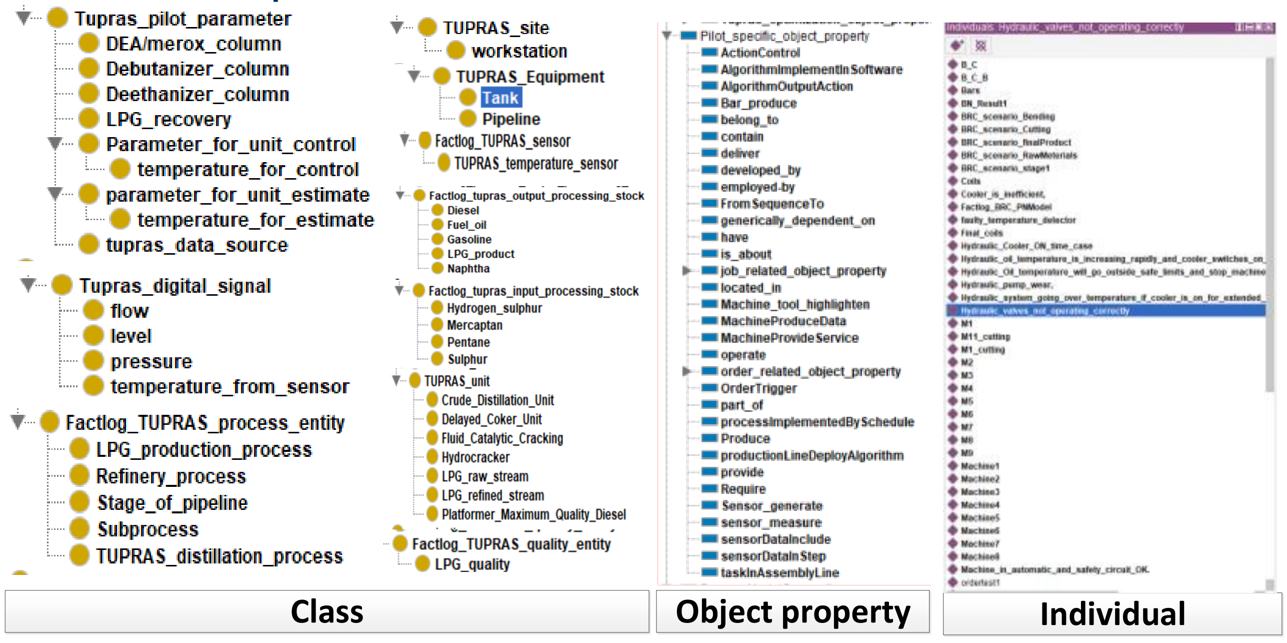
As shown in Figure 19, JEMS process entities, services, pilot parameter, sensor, JEMS unit, quality, digital signal, unit, input process stock and output process stock are the main ontology concepts defined in the knowledge graph models. General specific object properties are defined in order to support all the pilot description. Individuals are defined to describe an example of the JEMS scenario.

5.4 Knowledge Graph Modelling for TUPRAS

In the TUPRAS pilot, four main concerns are first considered when knowledge graph models are developed to formalize the TUPRAS pilot: 1) pilot description; 2) PM model formalism; 3) optimization formalism; 4) Data analysis formalism.

[1] JEMS pilot did not meet its objectives, especially with regards to the integration of the FACTLOG system to its plant since there is not yet an operative plant in Slovenia.

5.4.1 Pilot description



- Tupras_pilot_parameter
  - DEA/merox_column
  - Debutanizer_column
  - Deethanizer_column
  - LPG_recovery
  - Parameter_for_unit_control
    - temperature_for_control
  - parameter_for_unit_estimate
    - temperature_for_estimate
  - tupras_data_source
- Tupras_digital_signal
  - flow
  - level
  - pressure
  - temperature_from_sensor
- Factlog_TUPRAS_process_entity
  - LPG_production_process
  - Refinery_process
  - Stage_of_pipeline
  - Subprocess
  - TUPRAS_distillation_process
- TUPRAS_site
  - workstation
- TUPRAS_Equipment
  - Tank
  - Pipeline
- Factlog_TUPRAS_sensor
  - TUPRAS_temperature_sensor
- Factlog_tupras_output_processing_stock
  - Diesel
  - Fuel_oil
  - Gasoline
  - LPG_product
  - Naphtha
- Factlog_tupras_input_processing_stock
  - Hydrogen_sulphur
  - Mercaptan
  - Pentane
  - Sulphur
- TUPRAS_unit
  - Crude_Distillation_Unit
  - Delayed_Coker_Unit
  - Fluid_Catalytic_Cracking
  - Hydrocracker
  - LPG_raw_stream
  - LPG_refined_stream
  - Platformer_Maximum_Quality_Diesel
- Factlog_TUPRAS_quality_entity
  - LPG_quality

Class

- Pilot_specific_object_property
  - ActionControl
  - AlgorithmImplementInSoftware
  - AlgorithmOutputAction
  - Bar_produce
  - belong_to
  - contain
  - deliver
  - developed_by
  - employed-by
  - FromSequenceTo
  - generically_dependent_on
  - have
  - is_about
  - job_related_object_property
  - located_in
  - Machine_tool_highlighten
  - MachineProduceData
  - MachineProvideService
  - operate
  - order_related_object_property
  - OrderTrigger
  - part_of
  - processImplementedBySchedule
  - Produce
  - productionLineDeployAlgorithm
  - provide
  - Require
  - Sensor_generate
  - sensor_measure
  - sensorDataInclude
  - sensorDataInStep
  - taskInAssemblyLine

Object property

Individuals: Hydraulic_valves_not_operating_correctly

- B_C
- B_C_B
- Bars
- BN_Result1
- BRC_scenario_Bending
- BRC_scenario_Cutting
- BRC_scenario_finalProduct
- BRC_scenario_RawMaterials
- BRC_scenario_stage1
- Coils
- Cooler_is_inefficient,
- Factlog_BRC_PNModel
- faulty_temperature_detector
- Final_coils
- Hydraulic_Cooler_ON_time_case
- Hydraulic_oil_temperature_is_increasing_rapidly_and_cooler_switches_on_
- Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine
- Hydraulic_pump_wear,
- Hydraulic_system_going_over_temperature_if_cooler_is_on_for_extended_
- Hydraulic_valves_not_operating_correctly
- M1
- M11_cutting
- M1_cutting
- M2
- M3
- M4
- M5
- M6
- M7
- M8
- M9
- Machine1
- Machine2
- Machine3
- Machine4
- Machine5
- Machine6
- Machine7
- Machine8
- Machine_in_automatic_and_safety_circuit_OK.
- ordertest1

Individual

Figure 20: Ontology class, object property, data property and individuals for pilot description in TUPRAS pilot

As shown in Figure 20, TUPRAS process entities, TUPRAS services, TUPRAS pilot parameter, TUPRAS unit, TUPRAS quality, TUPRAS digital signal, TUPRAS site, TUPRAS input process stock and TUPRAS output process stock are the main ontology concepts defined in the knowledge graph models. General specific object properties are defined in order to support all the pilot description. Individuals are defined to describe an example of the TUPRAS scenario.

5.4.2 PM model formalism

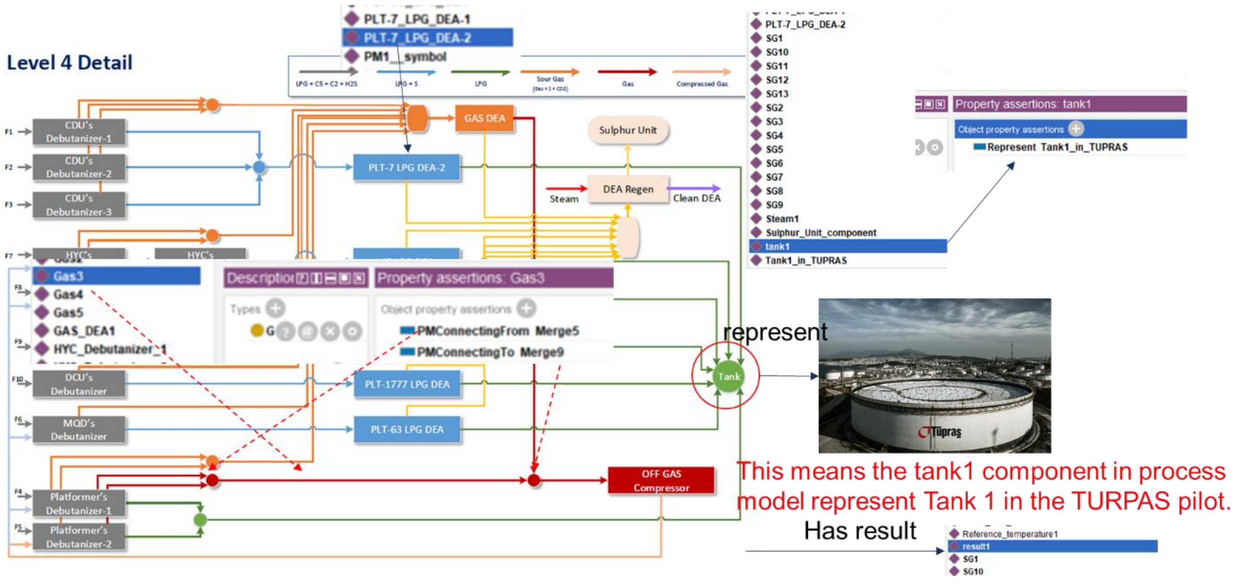
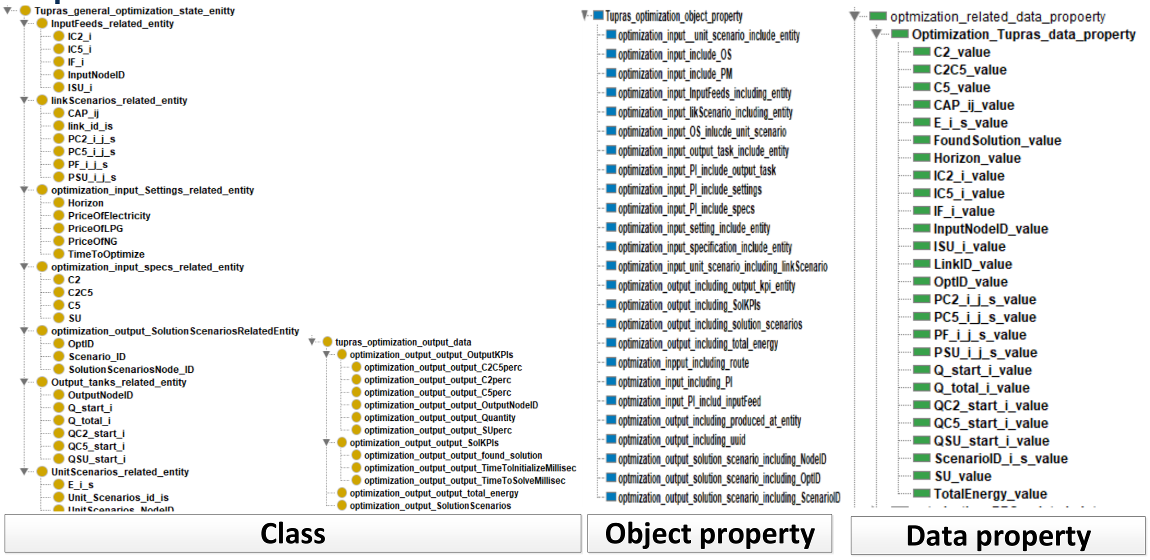


Figure 21: Individuals for process model description for TUPRAS pilot

Using the ontology in Section 5.4.2, the Figure 21 shows the individuals which are used to represent model structure for TUPRAS. For example, the individuals of PM_links are used to represent the connections among PM_nodes. The individuals of PM_nodes are used to represent each model composition in the model. The model composition is defined to represent the real equipment in the TUPRAS pilot.

5.4.3 Optimization formalism



```
Tupras_general_optimization_state_enitty
    InputFeeds_related_entity
        IC2_i
        IC5_i
        IF_i
        InputNodeID
        ISU_i
    linkScenarios_related_entity
        CAP_ij
        link_id_is
        PC2_i_j_s
        PC5_i_j_s
        PF_i_j_s
        PSU_i_j_s
    optimization_input_Settings_related_entity
        Horizon
        PriceOfElectricity
        PriceOfLPG
        PriceOfNG
        TimeToOptimize
    optimization_input_specs_related_entity
        C2
        C2C5
        C5
        SU
    optimization_output_SolutionScenariosRelatedEntity
        OptID
        Scenario_ID
        SolutionScenariosNode_ID
    Output_tanks_related_entity
        OutputNodeID
        Q_start_i
        Q_total_i
        QC2_start_i
        QC5_start_i
        QSU_start_i
    UnitScenarios_related_entity
        E_i_s
        Unit_Scenarios_id_is
        UnitScenarios_NodeID

tupras_optimization_output_data
    optimization_output_output_OutputKPIs
        optimization_output_output_C2C5perc
        optimization_output_output_C2perc
        optimization_output_output_C5perc
        optimization_output_output_OutputNodeID
        optimization_output_output_Quantity
        optimization_output_output_SUperc
    optimization_output_output_SolKPIs
        optimization_output_output_found_solution
        optimization_output_output_TimeToInitializeMillisec
        optimization_output_output_TimeToSolveMillisec
    optimization_output_output_total_energy
    optimization_output_SolutionScenarios
```

Class

```
Tupras_optimization_object_property
    optimization_input__unit_scenario_include_entity
    optimization_input_include_OS
    optimization_input_include_PM
    optimization_input_InputFeeds_including_entity
    optimization_input_likScenario_including_entity
    optimization_input_OS_inlucde_unit_scenario
    optimization_input_output_task_include_entity
    optimization_input_PI_include_output_task
    optimization_input_PI_include_settings
    optimization_input_PI_include_specs
    optimization_input_setting_include_entity
    optimization_input_specification_include_entity
    optimization_input_unit_scenario_including_linkScenario
    optimization_output_including_output_kpi_entity
    optimization_output_including_SolKPIs
    optimization_output_including_solution_scenarios
    optimization_output_including_total_energy
    optmization_inpput_including_route
    optmization_input_including_PI
    optmization_input_PI_includ_inputFeed
    optmization_output_including_produced_at_entity
    optmization_output_including_uuid
    optmization_output_solution_scenario_including_NodeID
    optmization_output_solution_scenario_including_OptID
    optmization_output_solution_scenario_including_ScenarioID
```

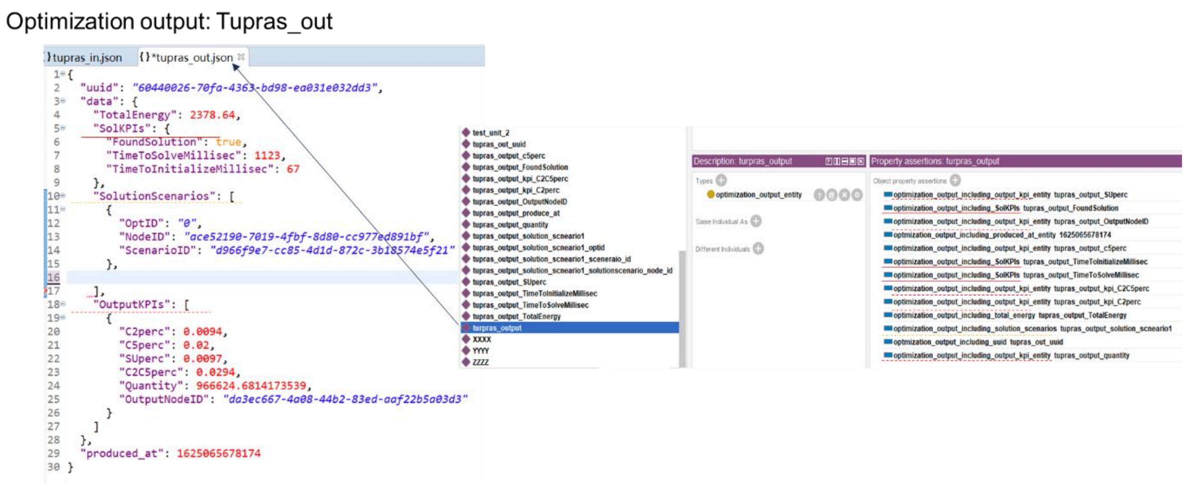
Object property

```
optmization_related_data_propoerty
    Optimization_Tupras_data_property
        C2_value
        C2C5_value
        C5_value
        CAP_ij_value
        E_i_s_value
        FoundSolution_value
        Horizon_value
        IC2_i_value
        IC5_i_value
        IF_i_value
        InputNodeID_value
        ISU_i_value
        LinkID_value
        OptID_value
        PC2_i_j_s_value
        PC5_i_j_s_value
        PF_i_j_s_value
        PSU_i_j_s_value
        Q_start_i_value
        Q_total_i_value
        QC2_start_i_value
        QC5_start_i_value
        QSU_start_i_value
        ScenarioID_i_s_value
        SU_value
        TotalEnergy_value
```

Data property

Figure 22: Ontology class, object property, and data property for optimization for TUPRAS pilot

As shown in Figure 23, optimization classes, object properties and data properties are defined to represent optimization scenario using individuals. In Figure 28, the individuals are used to represent the optimization output data structure. The TUPRAS optimization output has a solKPIs, solution scenario, outputKPIs and etc.



Optimization output: Tupras_out

```
{
  "uuid": "60440026-70fa-4363-bd98-ea031e032dd3",
  "data": {
    "TotalEnergy": 2378.64,
    "SolKPIs": {
      "FoundSolution": true,
      "TimeToSolveMillisec": 1123,
      "TimeToInitializeMillisec": 67
    },
    "SolutionScenarios": [
      {
        "OptID": "0",
        "NodeID": "ace52190-7019-4fbf-8d80-cc977ed891bf",
        "ScenarioID": "d966f9e7-cc85-4d1d-872c-3b18574e5f21"
      },

    ],
    "OutputKPIs": [
      {
        "C2perc": 0.0094,
        "C5perc": 0.02,
        "SUperc": 0.0097,
        "C2C5perc": 0.0294,
        "Quantity": 966624.6814173539,
        "OutputNodeID": "da3ec667-4a08-44b2-83ed-aaf22b5a03d3"
      }
    ]
  },
  "produced_at": 1625065678174
}
```

```
test_unit_2
tupras_out_uuid
tupras_output_c5perc
tupras_output_FoundSolution
tupras_output_kpi_C2C5perc
tupras_output_kpi_C2perc
tupras_output_OutputNodeID
tupras_output_produce_at
tupras_output_quantity
tupras_output_solution_scneario1
tupras_output_solution_scneario1_optid
tupras_output_solution_scneario1_sceneraio_id
tupras_output_solution_scneario1_solutionscenario_node_id
tupras_output_SUperc
tupras_output_TimeToInitializeMillisec
tupras_output_TimeToSolveMillisec
tupras_output_TotalEnergy
turpras_output
XXXX
YYYY
ZZZZ
```

Description: turpras_output

Types: optimization_output_entity

Same Individual As

Different Individuals

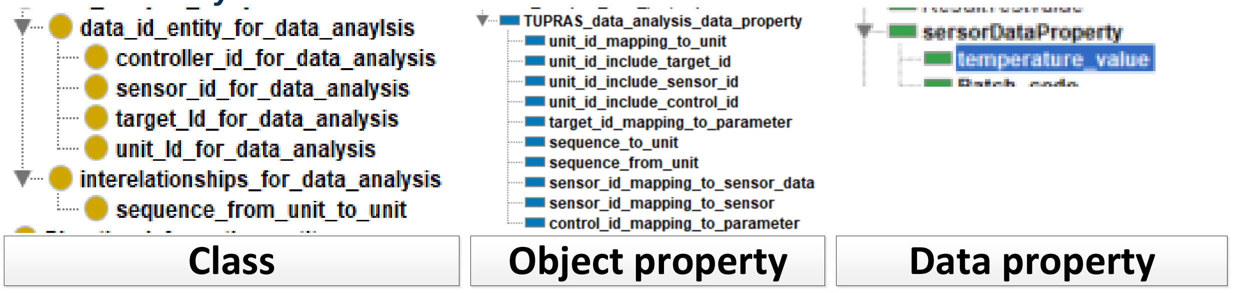
Property assertions: turpras_output

Object property assertions

```
optimization_output_including_output_kpi_entity  tupras_output_SUperc
optimization_output_including_SolKPIs  tupras_output_FoundSolution
optimization_output_including_output_kpi_entity  tupras_output_OutputNodeID
optmization_output_including_produced_at_entity  1625065678174
optimization_output_including_output_kpi_entity  tupras_output_c5perc
optimization_output_including_SolKPIs  tupras_output_TimeToInitializeMillisec
optimization_output_including_SolKPIs  tupras_output_TimeToSolveMillisec
optimization_output_including_output_kpi_entity  tupras_output_kpi_C2C5perc
optimization_output_including_output_kpi_entity  tupras_output_kpi_C2perc
optimization_output_including_total_energy  tupras_output_TotalEnergy
optimization_output_including_solution_scenarios  tupras_output_solution_scneario1
optmization_output_including_uuid  tupras_out_uuid
optimization_output_including_output_kpi_entity  tupras_output_quantity
```

Figure 23: Individuals for optimization in TUPRAS pilot

5.4.4 Data analysis formalism



data_id_entity_for_data_anaylsis
- controller_id_for_data_analysis
- sensor_id_for_data_analysis
- target_Id_for_data_analysis
- unit_Id_for_data_analysis

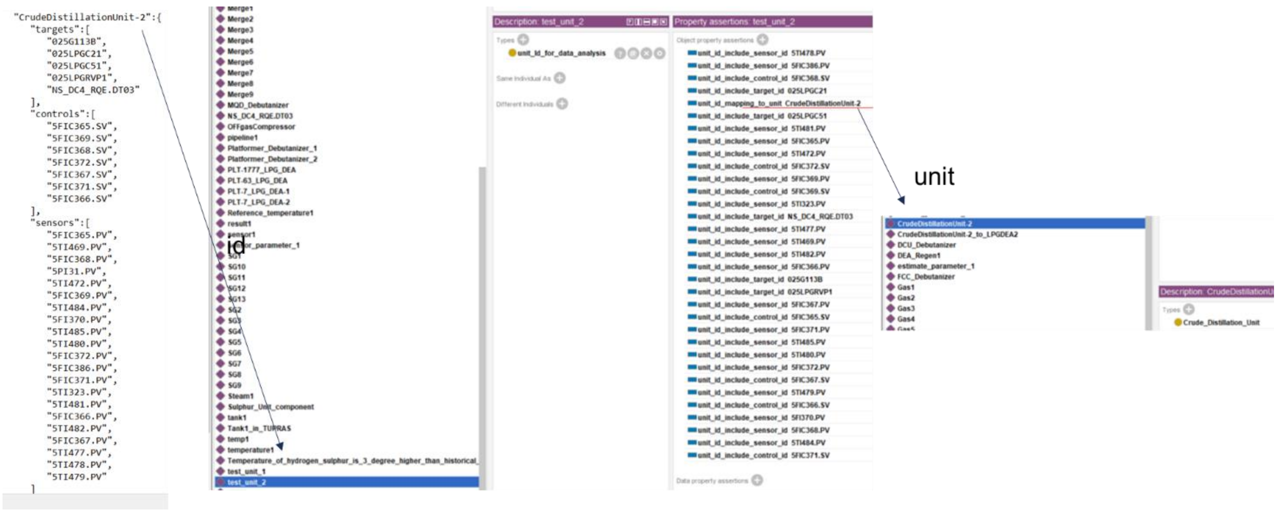
interelationships_for_data_analysis
- sequence_from_unit_to_unit

TUPRAS_data_analysis_data_property
- unit_id_mapping_to_unit
- unit_id_include_target_id
- unit_id_include_sensor_id
- unit_id_include_control_id
- target_id_mapping_to_parameter
- sequence_to_unit
- sequence_from_unit
- sensor_id_mapping_to_sensor_data
- sensor_id_mapping_to_sensor
- control_id_mapping_to_parameter

sersorDataProperty
- temperature_value

Figure 24: Ontology class, object property, and data property for data analysis in TUPRAS pilot

As shown in Figure 24, optimization classes, object properties and data properties are defined to represent data analysis scenario using individuals. In Figure 25, the individuals are used to represent the data analysis scenario. The TUPRAS data analysis scenario has a uuid which is mapping to the unit “crude distillation unit”.



```
"CrudeDistillationUnit-2":{
   "targets":[
      "025G113B",
      "025LPGC21",
      "025LPGC51",
      "025LPGRVP1",
      "NS_DC4_RQE.DT03"
   ],
   "controls":[
      "5FIC365.SV",
      "5FIC369.SV",
      "5FIC368.SV",
      "5FIC372.SV",
      "5FIC367.SV",
      "5FIC371.SV",
      "5FIC366.SV"
   ],
   "sensors":[
      "5FIC365.PV",
      "5TI469.PV",
      "5FIC368.PV",
      "5PI31.PV",
      "5TI472.PV",
      "5FIC369.PV",
      "5TI484.PV",
      "5FI370.PV",
      "5TI485.PV",
      "5TI480.PV",
      "5FIC372.PV",
      "5FIC386.PV",
      "5FIC371.PV",
      "5TI323.PV",
      "5TI481.PV",
      "5FIC366.PV",
      "5TI482.PV",
      "5FIC367.PV",
      "5TI477.PV",
      "5TI478.PV",
      "5TI479.PV"
   ]
```

Merge1, Merge2, Merge3, Merge4, Merge5, Merge6, Merge7, Merge8, Merge9, MQD_Debutanizer, NS_DC4_RQE.DT03, OFFgasCompressor, pipeline1, Platformer_Debutanizer_1, Platformer_Debutanizer_2, PLT-1777_LPG_DEA, PLT-63_LPG_DEA, PLT-7_LPG_DEA-1, PLT-7_LPG_DEA-2, Reference_temperature1, result1, sensor1, sensor_parameter_1, SG1, SG10, SG11, SG12, SG13, SG2, SG3, SG4, SG5, SG6, SG7, SG8, SG9, Steam1, Sulphur_Unit_component, tank1, Tank1_in_TUPRAS, temp1, temperature1, Temperature_of_hydrogen_sulphur_is_3_degree_higher_than_historical_, test_unit_1, test_unit_2

Description: test_unit_2
Types: unit_Id_for_data_analysis
Same Individual As
Different Individuals

Property assertions: test_unit_2
Object property assertions
- unit_id_include_sensor_id 5TI478.PV
- unit_id_include_sensor_id 5FIC386.PV
- unit_id_include_control_id 5FIC368.SV
- unit_id_include_target_id 025LPGC21
- unit_id_mapping_to_unit CrudeDistillationUnit-2
- unit_id_include_target_id 025LPGC51
- unit_id_include_sensor_id 5TI481.PV
- unit_id_include_sensor_id 5FIC365.PV
- unit_id_include_sensor_id 5TI472.PV
- unit_id_include_control_id 5FIC372.SV
- unit_id_include_sensor_id 5FIC369.PV
- unit_id_include_control_id 5FIC369.SV
- unit_id_include_sensor_id 5TI323.PV
- unit_id_include_target_id NS_DC4_RQE.DT03
- unit_id_include_sensor_id 5TI477.PV
- unit_id_include_sensor_id 5TI469.PV
- unit_id_include_sensor_id 5TI482.PV
- unit_id_include_sensor_id 5FIC366.PV
- unit_id_include_target_id 025G113B
- unit_id_include_target_id 025LPGRVP1
- unit_id_include_sensor_id 5FIC367.PV
- unit_id_include_control_id 5FIC365.SV
- unit_id_include_sensor_id 5FIC371.PV
- unit_id_include_sensor_id 5TI485.PV
- unit_id_include_sensor_id 5TI480.PV
- unit_id_include_sensor_id 5FIC372.PV
- unit_id_include_control_id 5FIC367.SV
- unit_id_include_sensor_id 5TI479.PV
- unit_id_include_control_id 5FIC366.SV
- unit_id_include_sensor_id 5FI370.PV
- unit_id_include_sensor_id 5FIC368.PV
- unit_id_include_sensor_id 5TI484.PV
- unit_id_include_control_id 5FIC371.SV

Data property assertions

CrudeDistillationUnit-2, CrudeDistillationUnit-2_to_LPGDEA2, DCU_Debutanizer, DEA_Regen1, estimate_parameter_1, FCC_Debutanizer, Gas1, Gas2, Gas3, Gas4

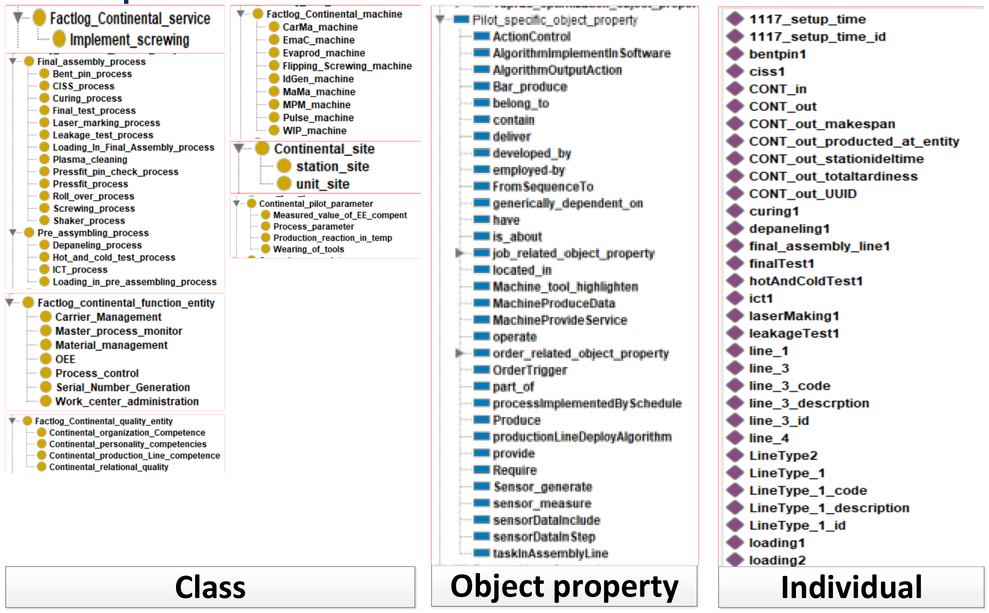
Description: CrudeDistillationU
Types: Crude_Distillation_Unit

Figure 25: Individuals for data analysis in TUPRAS pilot

5.5 Knowledge Graph Modelling for CONT

In the CONT pilot, three main concerns are first considered when knowledge graph models are developed to formalize the CONT pilot: 1) pilot description; 2) optimization formalism; 3) anomaly detection (introduced in Section 6.4.1).

5.5.1 Pilot description



```
Factlog_Continental_service
    Implement_screwing
Final_assembly_process
    Bent_pin_process
    CISS_process
    Curing_process
    Final_test_process
    Laser_marking_process
    Leakage_test_process
    Loading_In_Final_Assembly_process
    Plasma_cleaning
    Pressfit_pin_check_process
    Pressfit_process
    Roll_over_process
    Screwing_process
    Shaker_process
Pre_assymbling_process
    Depaneling_process
    Hot_and_cold_test_process
    ICT_process
    Loading_in_pre_assembling_process
Factlog_continental_function_entity
    Carrier_Management
    Master_process_monitor
    Material_management
    OEE
    Process_control
    Serial_Number_Generation
    Work_center_administration
Factlog_Continental_quality_entity
    Continental_organization_Competence
    Continental_personality_competencies
    Continental_production_Line_competence
    Continental_relational_quality
Factlog_Continental_machine
    CarMa_machine
    EmaC_machine
    Evaprod_machine
    Flipping_Screwing_machine
    IdGen_machine
    MaMa_machine
    MPM_machine
    Pulse_machine
    WIP_machine
Continental_site
    station_site
    unit_site
Continental_pilot_parameter
    Measured_value_of_EE_compent
    Process_parameter
    Production_reaction_in_temp
    Wearing_of_tools
```

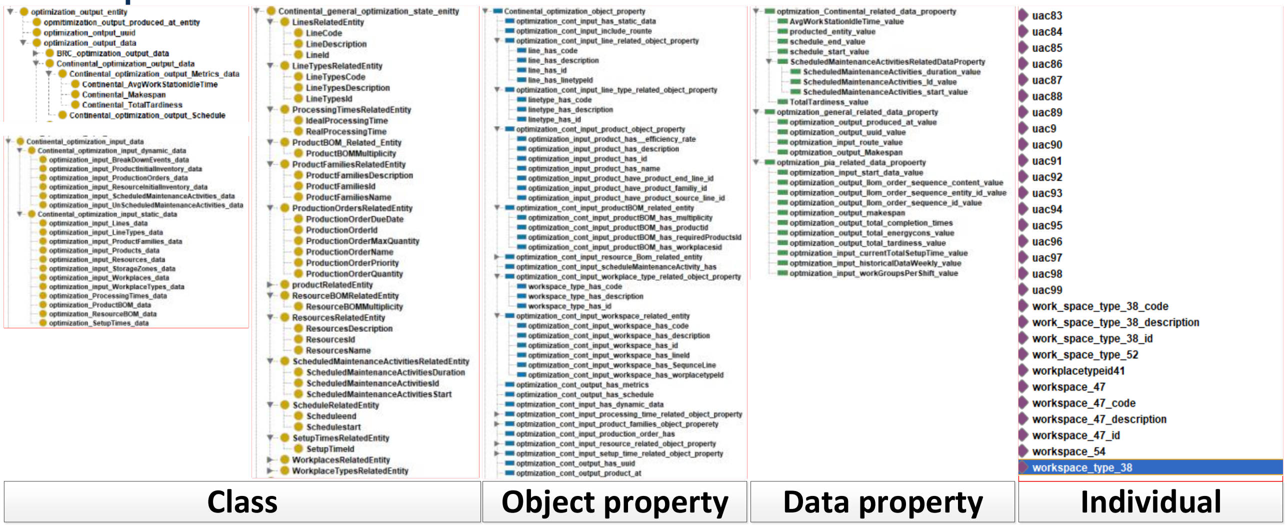
```
Pilot_specific_object_property
    ActionControl
    AlgorithmImplementInSoftware
    AlgorithmOutputAction
    Bar_produce
    belong_to
    contain
    deliver
    developed_by
    employed-by
    FromSequenceTo
    generically_dependent_on
    have
    is_about
    job_related_object_property
    located_in
    Machine_tool_highlighten
    MachineProduceData
    MachineProvideService
    operate
    order_related_object_property
    OrderTrigger
    part_of
    processImplementedBySchedule
    Produce
    productionLineDeployAlgorithm
    provide
    Require
    Sensor_generate
    sensor_measure
    sensorDataInclude
    sensorDataInStep
    taskInAssemblyLine
```

```
1117_setup_time
1117_setup_time_id
bentpin1
ciss1
CONT_in
CONT_out
CONT_out_makespan
CONT_out_producted_at_entity
CONT_out_stationideltime
CONT_out_totaltardiness
CONT_out_UUID
curing1
depaneling1
final_assembly_line1
finalTest1
hotAndColdTest1
ict1
laserMaking1
leakageTest1
line_1
line_3
line_3_code
line_3_descrption
line_3_id
line_4
LineType2
LineType_1
LineType_1_code
LineType_1_description
LineType_1_id
loading1
loading2
```

Figure 26: Ontology class, object property, data property and individuals for CONT pilot

As shown in Figure 26, CONT process entities, CONT services, CONT pilot parameter, CONT site, CONT machine, and CONT function are the main ontology concepts defined in the knowledge graph models. General specific object properties are defined in order to support all the pilot description. Individuals are defined to describe an example of the CONT scenario.

5.5.2 Optimization formalism



```
optimization_output_entity
    opmitimization_output_produced_at_entity
    optimization_ontput_uuid
    optimization_output_data
        BRC_optimization_output_data
        Continental_optimization_output_data
            Continental_optimization_output_Metrics_data
                Continental_AvgWorkStationIdleTime
                Continental_Makespan
                Continental_TotalTardiness
            Continental_optimization_output_Schedule

Continental_optimization_input_data
    Continental_optimization_input_dynamic_data
        optimization_input_BreakDownEvents_data
        optimization_input_ProductInitialInventory_data
        optimization_input_ProductionOrders_data
        optimization_input_ResourceInitialInventory_data
        optimization_input_ScheduledMaintenanceActivities_data
        optimization_input_UnScheduledMaintenanceActivities_data
    Continental_optimization_input_static_data
        optimization_input_Lines_data
        optimization_input_LineTypes_data
        optimization_input_ProductFamilies_data
        optimization_input_Products_data
        optimization_input_Resources_data
        optimization_input_StorageZones_data
        optimization_input_Workplaces_data
        optimization_input_WorkplaceTypes_data
        optimization_ProcessingTimes_data
        optimization_ProductBOM_data
        optimization_ResourceBOM_data
        optimization_SetupTimes_data

Continental_general_optimization_state_enitty
    LinesRelatedEntity
        LineCode
        LineDescription
        LineId
    LineTypesRelatedEntity
        LineTypesCode
        LineTypesDescription
        LineTypesId
    ProcessingTimesRelatedEntity
        IdealProcessingTime
        RealProcessingTime
    ProductBOM_Related_Entity
        ProductBOMMultiplicity
    ProductFamiliesRelatedEntity
        ProductFamiliesDescription
        ProductFamiliesId
        ProductFamiliesName
    ProductionOrdersRelatedEntity
        ProductionOrderDueDate
        ProductionOrderId
        ProductionOrderMaxQuantity
        ProductionOrderName
        ProductionOrderPriority
        ProductionOrderQuantity
    productRelatedEntity
    ResourceBOMRelatedEntity
        ResourceBOMMultiplicity
    ResourcesRelatedEntity
        ResourcesDescription
        ResourcesId
        ResourcesName
    ScheduledMaintenanceActivitiesRelatedEntity
        ScheduledMaintenanceActivitiesDuration
        ScheduledMaintenanceActivitiesId
        ScheduledMaintenanceActivitiesStart
    ScheduleRelatedEntity
        Scheduleend
        Schedulestart
    SetupTimesRelatedEntity
        SetupTimeId
    WorkplacesRelatedEntity
    WorkplaceTypesRelatedEntity
```

```
Continental_optimization_object_property
    optimization_cont_input_has_static_data
    optimization_cont_input_include_rounte
    optimization_cont_input_line_related_object_property
        line_has_code
        line_has_description
        line_has_id
        line_has_linetypeId
    optimization_cont_input_line_type_related_object_property
        linetype_has_code
        linetype_has_description
        linetype_has_id
    optimization_cont_input_product_object_property
        optimization_input_product_has__efficiency_rate
        optimization_input_product_has_description
        optimization_input_product_has_id
        optimization_input_product_has_name
        optimization_input_product_have_product_end_line_id
        optimization_input_product_have_product_familiy_id
        optimization_input_product_have_product_source_line_id
    optimization_cont_input_productBOM_related_entity
        optimization_cont_input_productBOM_has_multiplicity
        optimization_cont_input_productBOM_has_productid
        optimization_cont_input_productBOM_has_requiredProductsId
        optimization_cont_input_productBOM_has_workplacesid
    optimization_cont_input_resource_Bom_related_entity
    optimization_cont_input_scheduleMaintenanceActivity_has
    optimization_cont_input_workplace_type_related_object_property
        workspace_type_has_code
        workspace_type_has_description
        workspace_type_has_id
    optimization_cont_input_workspace_related_entity
        optimization_cont_input_workspace_has_code
        optimization_cont_input_workspace_has_description
        optimization_cont_input_workspace_has_id
        optimization_cont_input_workspace_has_lineId
        optimization_cont_input_workspace_has_SequnceLine
        optimization_cont_input_workspace_has_worplacetypeId
    optimization_cont_output_has_metrics
    optimization_cont_output_has_schedule
    optmization_cont_input_has_dynamic_data
    optmization_cont_input_processing_time_related_object_property
    optmization_cont_input_product_families_object_properety
    optmization_cont_input_production_order_has
    optmization_cont_input_resource_related_object_property
    optmization_cont_input_setup_time_related_object_property
    optmization_cont_output_has_uuid
    optmization_cont_output_product_at
```

```
optmization_Continental_related_data_propoerty
    AvgWorkStationIdleTime_value
    producted_entity_value
    schedule_end_value
    schedule_start_value
    ScheduledMaintenanceActivitiesRelatedDataProperty
        ScheduledMaintenanceActivities_duration_value
        ScheduledMaintenanceActivities_Id_value
        ScheduledMaintenanceActivities_start_value
    TotalTardiness_value
optmization_general_related_data_property
    optimization_output_produced_at_value
    optimization_output_uuid_value
    optmization_input_route_value
    optmization_output_Makespan
optmization_pia_related_data_propoerty
    optimization_input_start_data_value
    optimization_output_llom_order_sequence_content_value
    optimization_output_llom_order_sequence_entity_id_value
    optimization_output_llom_order_sequence_id_value
    optimization_output_makespan
    optimization_output_total_completion_times
    optimization_output_total_energycons_value
    optimization_output_total_tardiness_value
    optmization_input_currentTotalSetupTime_value
    optmization_input_historicalDataWeekly_value
    optmization_input_workGroupsPerShift_value
```

```
uac83
uac84
uac85
uac86
uac87
uac88
uac89
uac9
uac90
uac91
uac92
uac93
uac94
uac95
uac96
uac97
uac98
uac99
work_space_type_38_code
work_space_type_38_description
work_space_type_38_id
work_space_type_52
workplacetypeid41
workspace_47
workspace_47_code
workspace_47_description
workspace_47_id
workspace_54
workspace_type_38
```

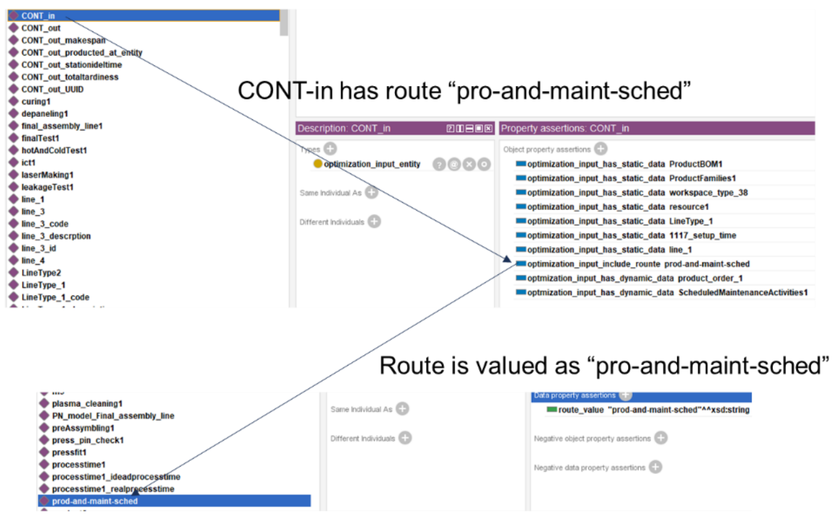
Figure 27: Ontology class, object property, data property and individuals for optimization for CONT pilot

As shown in Figure 27, optimization classes, object properties and data properties are defined to represent optimization scenario using individuals. In Figure 28, the individuals are used to represent the optimization input data structure. The CONT optimization input has a route named “prod-and-maint-sched” whose value is “prod-and-maint-sched”.

Optimization input: CONT in

```
{
"route": "prod-and-maint-sched",
"data": {
"staticData": {
"WorkplaceTypes": [{
"Id": 38,
"Code": "ICT test",
"Description": "ICT test"
},
"LineTypes": [{
"Id": 1,
"Code": "FA",
"Description": "FA"
},
],

 ],
```


CONT-in has route “pro-and-maint-sched”
Route is valued as “pro-and-maint-sched”

```
CONT_in
CONT_out
CONT_out_makespan
CONT_out_producted_at_entity
CONT_out_stationideltime
CONT_out_totaltardiness
CONT_out_UUID
curing1
depaneling1
final_assembly_line1
finalTest1
hotAndColdTest1
ict1
laserMaking1
leakageTest1
line_1
line_3
line_3_code
line_3_descrption
line_3_id
line_4
LineType2
LineType_1
LineType_1_code
```

```
Description: CONT_in
Types
  optimization_input_entity
Same Individual As
Different Individuals
```

```
Property assertions: CONT_in
Object property assertions
  optimization_input_has_static_data  ProductBOM1
  optimization_input_has_static_data  ProductFamilies1
  optimization_input_has_static_data  workspace_type_38
  optimization_input_has_static_data  resource1
  optimization_input_has_static_data  LineType_1
  optimization_input_has_static_data  1117_setup_time
  optimization_input_has_static_data  line_1
  optimization_input_include_rounte  prod-and-maint-sched
  optmization_input_has_dynamic_data  product_order_1
  optmization_input_has_dynamic_data  ScheduledMaintenanceActivities1
```

```
plasma_cleaning1
PN_model_Final_assembly_line
preAssymbling1
press_pin_check1
pressfit1
processtime1
processtime1_ideadprocesstime
processtime1_realprocesstime
prod-and-maint-sched
```

```
Same Individual As
Different Individuals
```

```
Data property assertions
  route_value  "prod-and-maint-sched"^^xsd:string
Negative object property assertions
Negative data property assertions
```

Figure 28: Individuals for describing the optimization scenario for CONT pilot

6 Integration of KGM and Cognition Services in the FACTLOG platform

In this chapter, two approaches are used to integrate the designed knowledge graph models into cognition services in the FACTLOG platform.

6.1 Integration based on OWL models

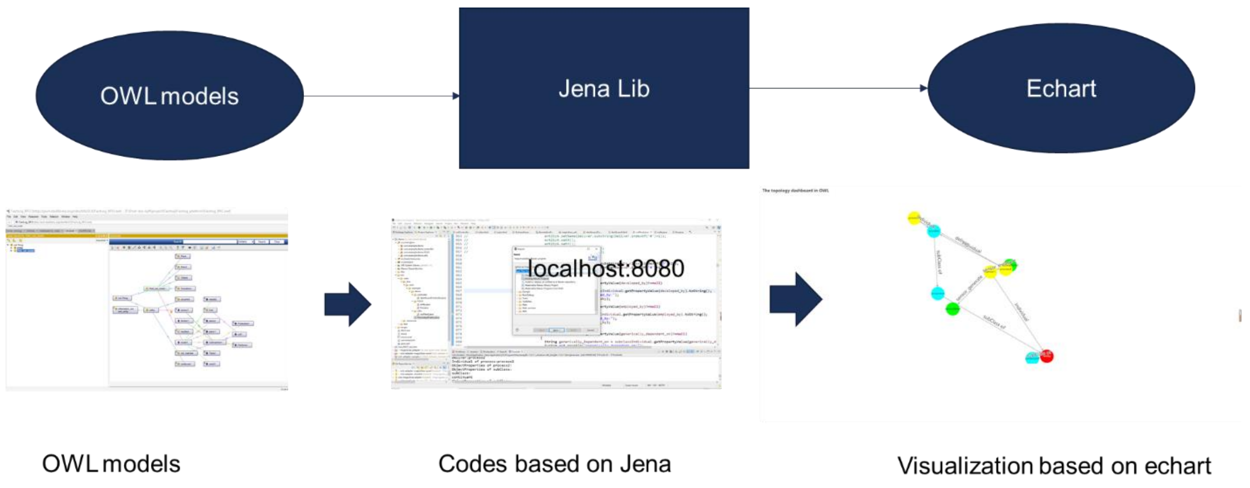


Figure 29: Workflow for visualizing knowledge graph model

As shown in Figure 29, a React and java project is developed based on Jena Lib in order to visualize a knowledge graph model which is built for the FACTLOG project. Totally, there are three steps to realize the visualization:

1. OWL models are developed in Protégé based on the developed ontology.
2. Through the java project we proposed, which details are shown in ANNEX III.
3. The OWL files are shown in the web.

6.2 Integration based on Neo4j

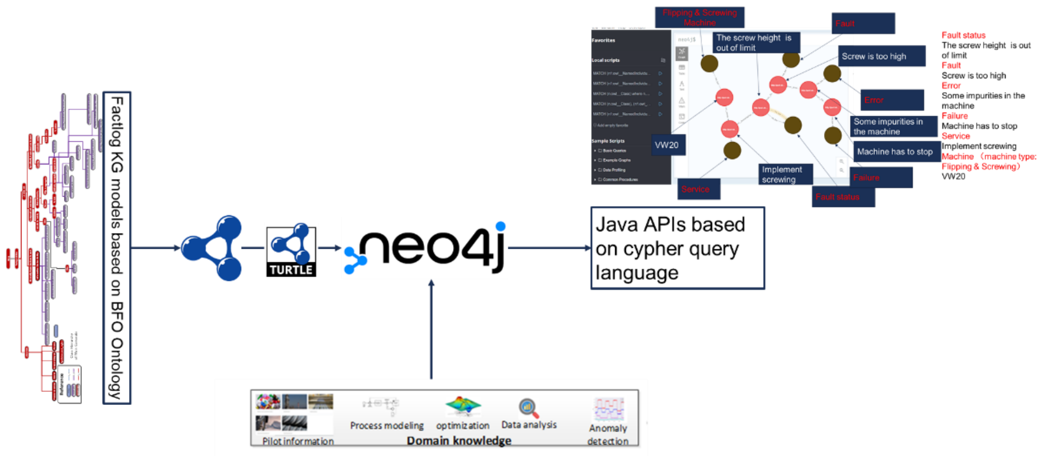


Figure 30: OWL model transformed into Neo4j

As shown in Figure 30, ontology models which are developed based on BFO framework is generated into Neo4j knowledge graph models through turtle. The cypher query language is used to support reasoning of knowledge graph models.

6.3 Integration based on HTTP API

As shown in Figure 14, we developed a Python application that leverages an OWL ontology developed based on the BFO framework to instantiate a knowledge graph. An HTTP API is offered to the external services to enable the training of machine learning models based either on domain knowledge regarding a specific use case, or regarding specifications for building such a model. When detailed specifications are provided to build a certain model, a parser is used to interpret how the features must be built and build them. In all cases, the machine learning model is trained and persisted into the filesystem, so that it can be later accessed along with metadata of interest. Model training is performed asynchronously. All information regarding the characteristics of the model, their instantiation and performance are recorded in the knowledge graph.

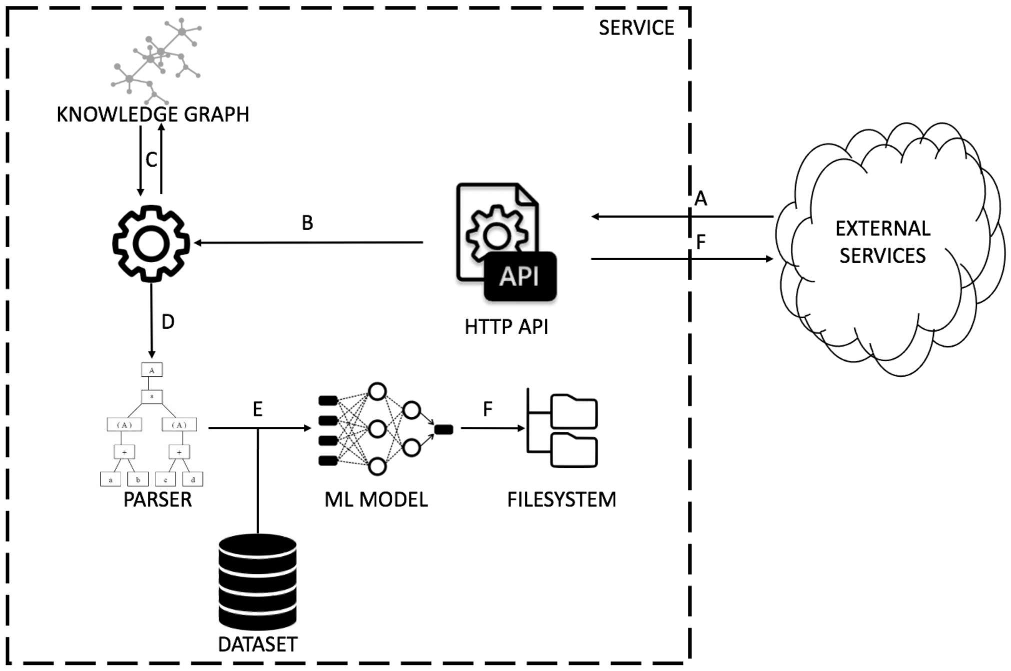


Figure 31. Diagram displaying the components' interaction at the KG-based analytics for process optimization architecture.

6.4 Cognition services for FACTLOG Factories

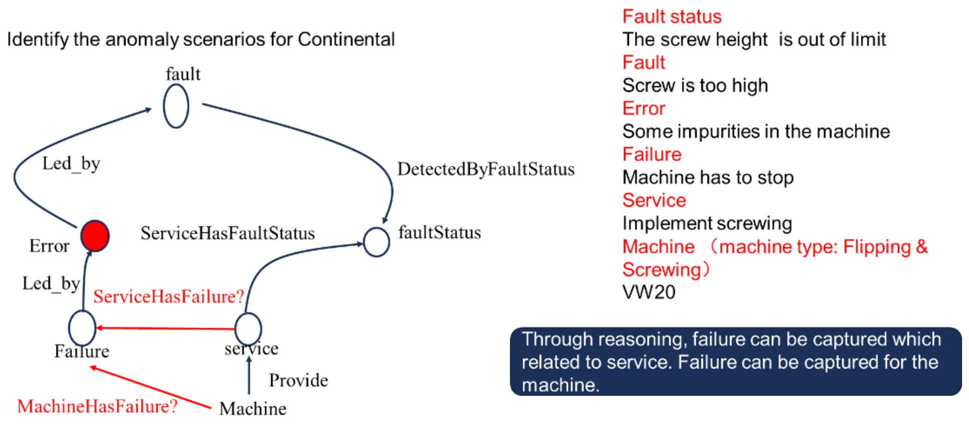
6.4.1 Cognition services based on OWL

6.4.1.1 Anomaly detection for CONT Pilot

Based on ontology for anomaly detection, a knowledge graph model is defined to represent the rational of anomaly detection for the CONT pilot.

Table 2 – Anomaly detection scenario for CONT pilot

Anomaly								
Sensor Data	Possible Error detection (**Error**)	Required additional data inputs （**Reference data**）	Drift Measure （**Fault status)** extral （ontology for natural data）	Possible Causes (**Fault**)	Possible outcomes (**Failure**)	**Service**	Machine	Machine type
-	Some impurities in the machine	-	The screw height is out of limit	Screw is too high	Machine has to stop	Implement screwing	VW20	Flipping & Screwing



Fault status
The screw height is out of limit
Fault
Screw is too high
Error
Some impurities in the machine
Failure
Machine has to stop
Service
Implement screwing
Machine （machine type: Flipping & Screwing）
VW20

Through reasoning, failure can be captured which related to service. Failure can be captured for the machine.

Figure 32: Anomaly scenario for CONT pilot

As shown in Figure 32, based on Table 2, an anomaly scenario is defined. The knowledge graph model is built based on the given data and anomaly ontology. The relationships between failure and two other concepts including service and machine are not defined in the knowledge graph. Thus, a reasoning is executed based on the developed knowledge graph models in order to capture a machine or service has a failure.

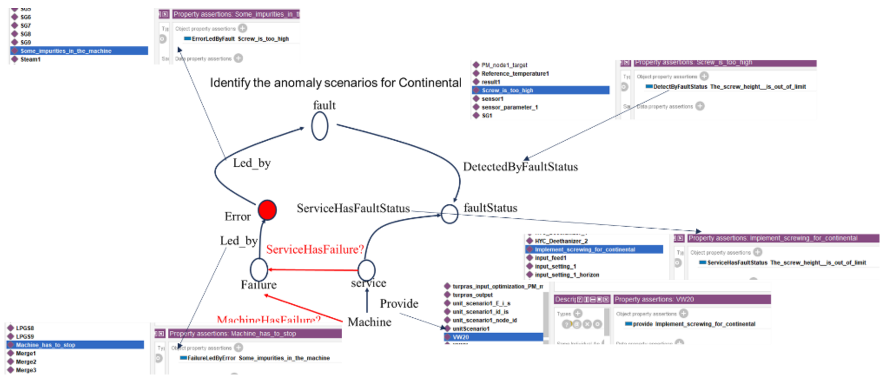
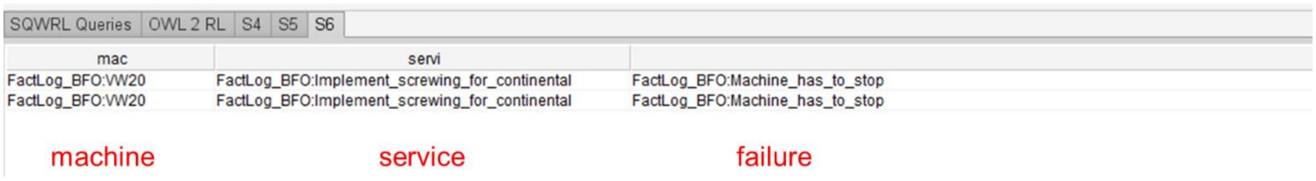

Some_impurities_in_the_machine
Property assertions: Some_impurities_in_t
ErrorLedByFault Screw_is_too_high
Identify the anomaly scenarios for Continental
fault
Led_by
DetectedByFaultStatus
Error
ServiceHasFaultStatus
faultStatus
Led_by
ServiceHasFailure?
Failure
service
Provide
MachineHasFailure?
Machine
Screw_is_too_high
Property assertions: Screw_is_too_high
DetectByFaultStatus The_screw_height__is_out_of_limit
Implement_screwing_for_continental
Property assertions: Implement_screwing_for_continental
ServiceHasFaultStatus The_screw_height__is_out_of_limit
VW20
Property assertions: VW20
provide Implement_screwing_for_continental
Machine_has_to_stop
Property assertions: Machine_has_to_stop
FailureLedByError Some_impurities_in_the_machine

Figure 33: Knowledge graph models for CONT pilot

As shown in Figure 33, knowledge graph models are defined to define the anomaly scenario. Through SQWRL rule, a reasoning is implemented:

```
FACTLOG_BFO:Flipping_Screwing_machine(?mac) ^ FACTLOG_BFO:provide(?mac, ?servi) ^ FACTLOG_BFO:Implement_screwing(?servi) ^ FACTLOG_BFO:ServiceHasFaultStatus(?servi, ?fsta) ^ FACTLOG_BFO:fault_status(?fsta) ^ FACTLOG_BFO:DetectByFaultStatus(?fau, ?fsta) ^ FACTLOG_BFO:fault(?fau) ^ FACTLOG_BFO:ErrorLedByFault(?err, ?fau) ^ FACTLOG_BFO:error(?err) ^ FACTLOG_BFO:FailureLedByError(?imser, ?fal) ^ FACTLOG_BFO:failure(?fail) -> sqwrl:select(?mac, ?servi, ?fail)
```



mac	servi	
FactLog_BFO:VW20	FactLog_BFO:Implement_screwing_for_continental	FactLog_BFO:Machine_has_to_stop
FactLog_BFO:VW20	FactLog_BFO:Implement_screwing_for_continental	FactLog_BFO:Machine_has_to_stop

Figure 34: Reasoning result for CONT pilot

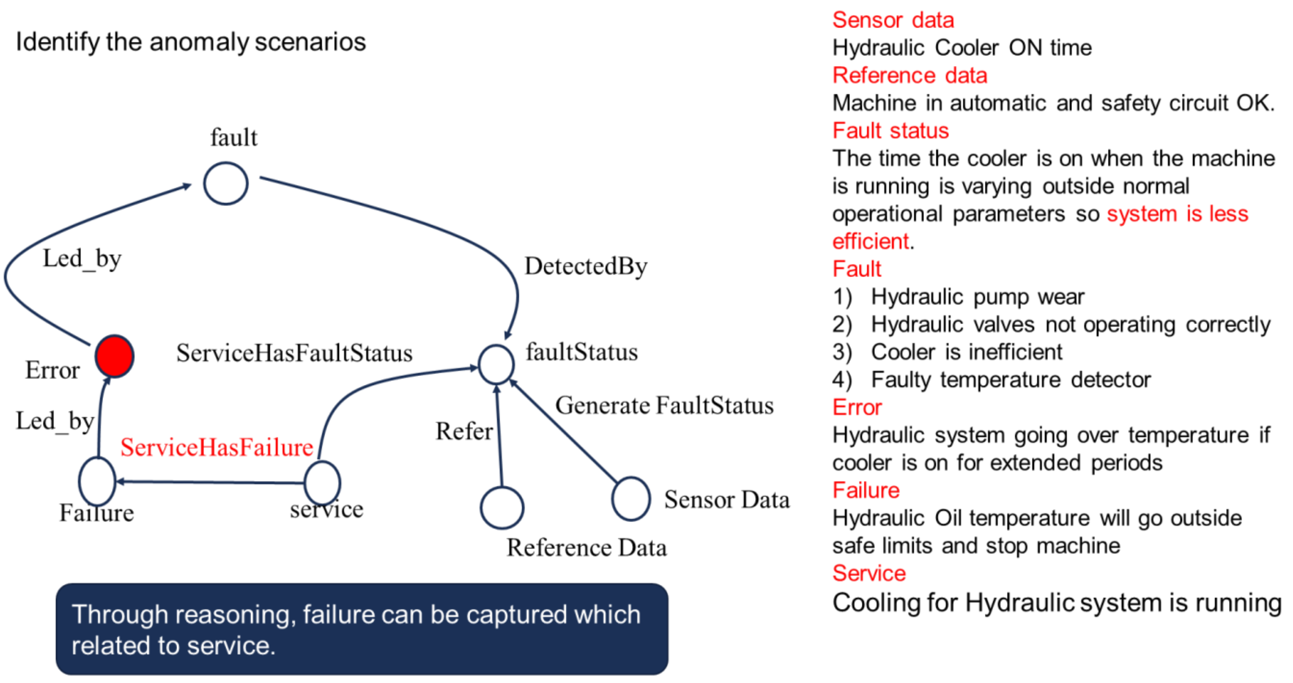
Finally, a reasoning result is obtained. From the result, we can understand VW20 machine has a failure: Machine has to stop.

6.4.1.2 Anomaly detection for BRC Pilot

Based on the ontology for anomaly detection, a knowledge graph model is defined to represent the rational of anomaly detection for the BRC pilot.

Table 3 – Anomaly detection scenario for BRC pilot

Sensor Data **Digital Signals**	Possible Error detection (**Error**)	Required additional data inputs （**Reference data**）	Drift Measure（**Fault status)** extral （ontology for natural data）	Possible Causes (**Fault**)	Possible outcomes (**Failure**)	**Service**
Hydraulic Cooler ON time	Hydraulic system going over temperature if cooler is on for extended periods	Machine in automatic and safety circuit OK.	The time the cooler is on when the machine is running is varying outside normal operational parameters so system is less efficient	1) Hydraulic pump wear 2) Hydraulic valves not operating correctly. 3) Cooler is inefficient. 4) faulty temperature detector	Hydraulic Oil temperature will go outside safe limits and stop machine	Cooling for Hydraulic system is running



Through reasoning, failure can be captured which related to service.

Sensor data
Hydraulic Cooler ON time
Reference data
Machine in automatic and safety circuit OK.
Fault status
The time the cooler is on when the machine is running is varying outside normal operational parameters so system is less efficient.
Fault
1) Hydraulic pump wear
2) Hydraulic valves not operating correctly
3) Cooler is inefficient
4) Faulty temperature detector
Error
Hydraulic system going over temperature if cooler is on for extended periods
Failure
Hydraulic Oil temperature will go outside safe limits and stop machine
Service
Cooling for Hydraulic system is running

Figure 35: Anomaly scenario for BRC pilot

As shown in Figure 35, based on Table 3, an anomaly scenario is defined. The knowledge graph model is built based on the given data and anomaly ontology. The relationships between failure and two other concepts including service are not defined in the knowledge graph. Thus, a reasoning is executed based on the developed knowledge graph models in order to capture a service has a failure.

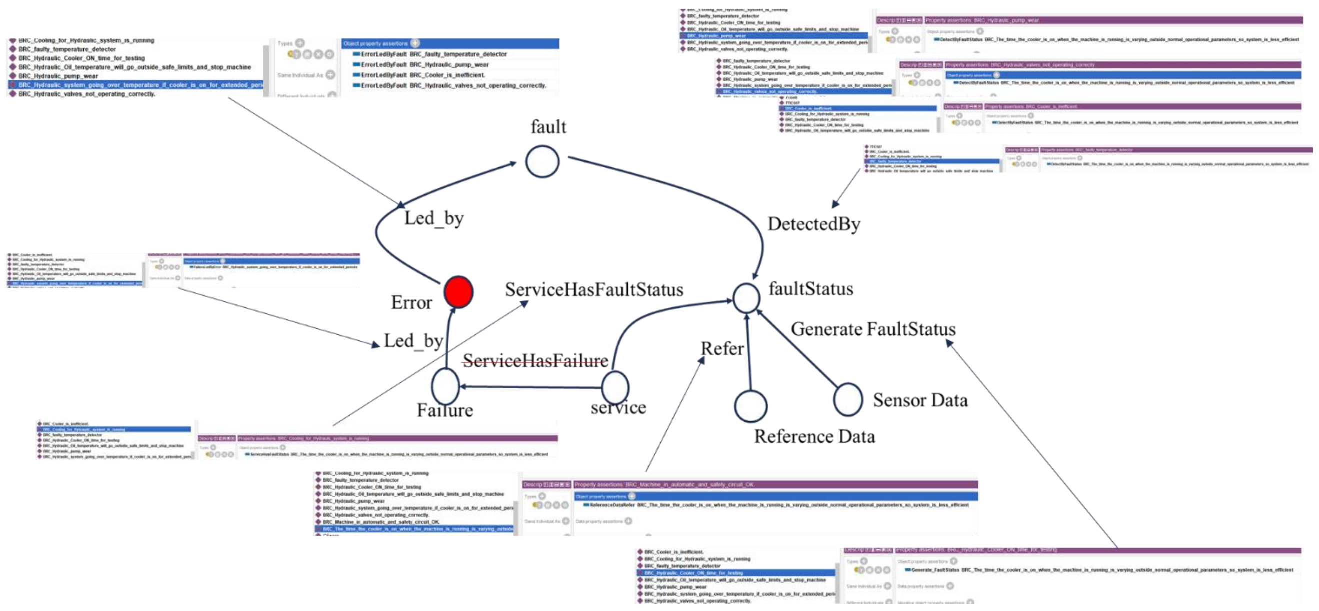
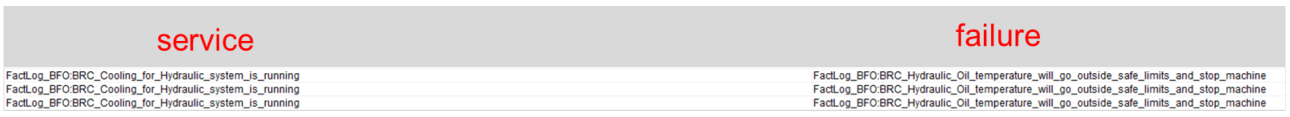


Figure 36: Knowledge graph models for BRC pilot

As shown in Figure 36, knowledge graph models are defined to define the anomaly scenario. Through SQWRL rule, a reasoning is implemented:

```
FACTLOG_BFO:Hydraulic_system_cooling_service(?servi) ^
FACTLOG_BFO:ServiceHasFaultStatus(?servi, ?fsta) ^
FACTLOG_BFO:Hydraulic_Cooler_ON_time(?senData) ^
FACTLOG_BFO:Generate_FaultStatus(?senData, ?fsta) ^
FACTLOG_BFO:fault_status(?fsta) ^ FACTLOG_BFO:DetectByFaultStatus(?fau, ?fsta) ^
FACTLOG_BFO:fault(?fau) ^ FACTLOG_BFO:ErrorLedByFault(?err, ?fau) ^
FACTLOG_BFO:error(?err) ^ FACTLOG_BFO:FailureLedByError(?imser, ?fal) ^
FACTLOG_BFO:failure(?fail) -> sqwrl:select(?servi, ?fail)
```

Finally, a reasoning result is obtained. From the result, we can understand BRC_Cooling_for_Hydraulic_system_is_running service has a failure: BRC_Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine.



service	failure
FactLog_BFO:BRC_Cooling_for_Hydraulic_system_is_running	FactLog_BFO:BRC_Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine
FactLog_BFO:BRC_Cooling_for_Hydraulic_system_is_running	FactLog_BFO:BRC_Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine
FactLog_BFO:BRC_Cooling_for_Hydraulic_system_is_running	FactLog_BFO:BRC_Hydraulic_Oil_temperature_will_go_outside_safe_limits_and_stop_machine

Figure 37: Reasoning result for BRC pilot

6.4.1.3 Anomaly detection for JEMS pilot

Based on the ontology for anomaly detection, a knowledge graph model is defined to represent the rationale of anomaly detection for the JEMS pilot[2].

Table 4 – Anomaly detection scenario for JEMS pilot

Sensor Data	Possible Error detection (**Error**)	Required additional data inputs （**Reference data**)	Drift Measure（**Fault status**) extra (ontology for natural data）	Possible Causes (**Fault**)	Possible outcomes (**Failure**)	**Service**	Machine
• Ingested material flow speed • Mixing power • Temperature • Pressure • Turbine flow • Pump speed	**clogged pipe**	Machine in automatic and safety circuit OK.	• Ingested material flow speed: decreasing ingestion speed • Mixing power: increasing machine power required to mix the mixture • Temperature: temperature above 160C or 300C in phases 2 and 3 • Pressure: increased pressure in phases 2 and 3 • Turbine flow: decreased turbine flow per minute • Pump speed: decreased pump speed when pumping from B100 to turbine	The fault status signals highly increased density/viscosity of the mixture preventing normal operation	The machine stops working / must be stopped.	The machine provides the waste processing.	Waste to fuel plant

As shown in Figure 38, based on Table 4, an anomaly scenario is defined. The knowledge graph model is built based on the given data and anomaly ontology. The relationships between failure and two other concepts including service are not defined in the knowledge graph. Thus, a reasoning is executed based on the developed knowledge graph models in order to capture a machine has a failure.

[2] JEMS pilot did not meet its objectives, especially with regards to the integration of the FACTLOG system to its plant since there is not yet an operative plant in Slovenia.

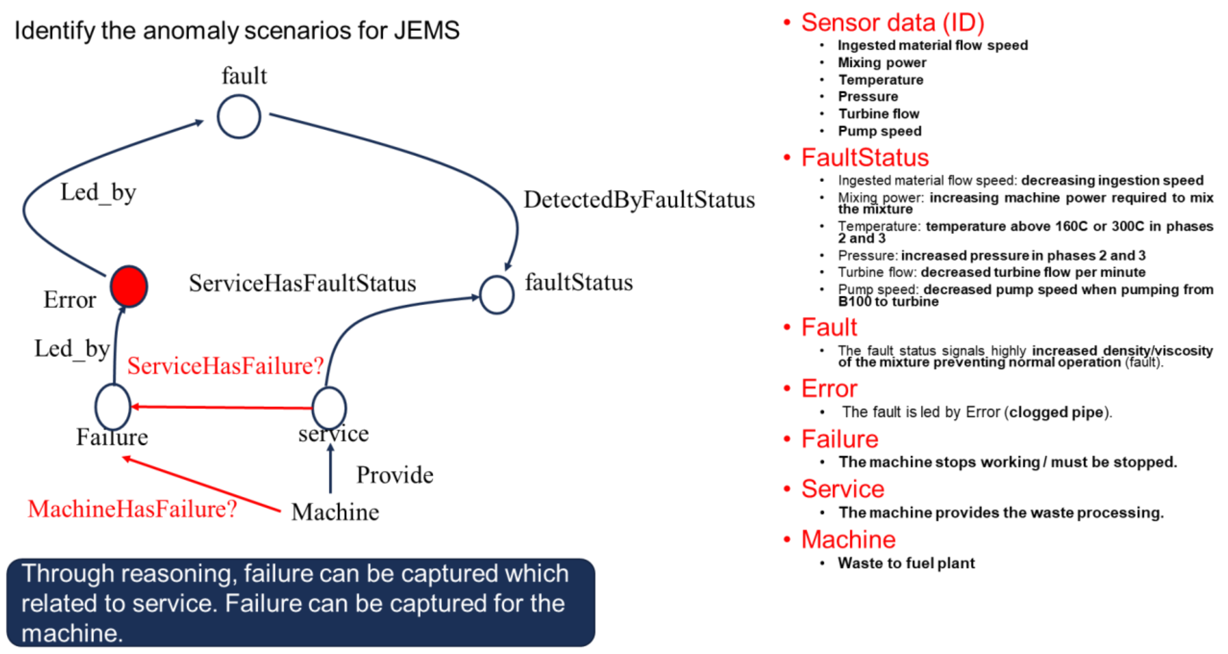

Identify the anomaly scenarios for JEMS
fault
Led_by
DetectedByFaultStatus
Error
ServiceHasFaultStatus
faultStatus
Led_by
ServiceHasFailure?
Failure
service
Provide
MachineHasFailure?
Machine
Through reasoning, failure can be captured which related to service. Failure can be captured for the machine.
Sensor data (ID)
Ingested material flow speed
Mixing power
Temperature
Pressure
Turbine flow
Pump speed
FaultStatus
Ingested material flow speed: decreasing ingestion speed
Mixing power: increasing machine power required to mix the mixture
Temperature: temperature above 160C or 300C in phases 2 and 3
Pressure: increased pressure in phases 2 and 3
Turbine flow: decreased turbine flow per minute
Pump speed: decreased pump speed when pumping from B100 to turbine
Fault
The fault status signals highly increased density/viscosity of the mixture preventing normal operation (fault).
Error
The fault is led by Error (clogged pipe).
Failure
The machine stops working / must be stopped.
Service
The machine provides the waste processing.
Machine
Waste to fuel plant

Figure 38: Anomaly scenario for JEMS pilot

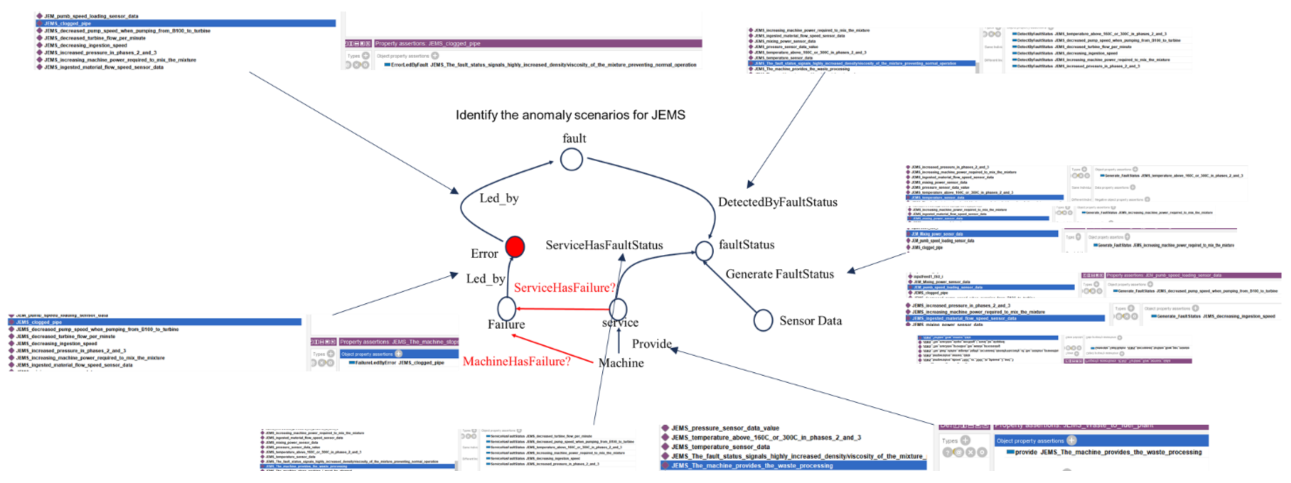
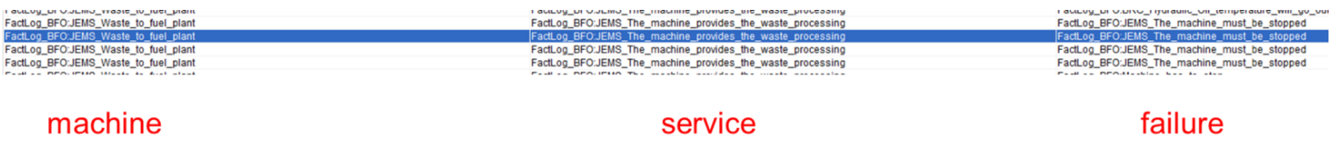


Figure 39: Knowledge graph models for JEMS pilot

As shown in Figure 39, knowledge graph models are defined to define the anomaly scenario. Through SQWRL rule, a reasoning is implemented:

FACTLOG_BFO:Waste_to_fuel_plant (?mac) ^ FACTLOG_BFO:provide(?mac, ?servi) ^ FACTLOG_BFO:waste_processing_service(?servi) ^ FACTLOG_BFO:ServiceHasFaultStatus(?servi, ?fsta) ^ FACTLOG_BFO:fault_status(?fsta) ^ FACTLOG_BFO:DetectByFaultStatus(?fau, ?fsta) ^ FACTLOG_BFO:fault(?fau) ^ FACTLOG_BFO:ErrorLedByFault(?err, ?fau) ^ FACTLOG_BFO:error(?err) ^ FACTLOG_BFO:FailureLedByError(?imser, ?fal) ^ FACTLOG_BFO:failure(?fail) -> sqwrl:select(?mac, ?servi, ?fail)

Finally, a reasoning result is obtained. From the result, we can understand the machine JEMS_Waste_to_fuel_plant provides a service (JEMS_The_machine_provides_the_waste_processing) has a failure: JEMS_The_machine_must_be_stopped.



FactLog_BFO:JEMS_Waste_to_fuel_plant	FactLog_BFO:JEMS_The_machine_provides_the_waste_processing	FactLog_BFO:JEMS_The_machine_must_be_stopped
FactLog_BFO:JEMS_Waste_to_fuel_plant	FactLog_BFO:JEMS_The_machine_provides_the_waste_processing	FactLog_BFO:JEMS_The_machine_must_be_stopped
FactLog_BFO:JEMS_Waste_to_fuel_plant	FactLog_BFO:JEMS_The_machine_provides_the_waste_processing	FactLog_BFO:JEMS_The_machine_must_be_stopped
FactLog_BFO:JEMS_Waste_to_fuel_plant	FactLog_BFO:JEMS_The_machine_provides_the_waste_processing	FactLog_BFO:JEMS_The_machine_must_be_stopped

Figure 40: Reasoning result for JEMS pilot

6.4.1.4 Anomaly detection for PIA pilot

Based on the ontology for anomaly detection, a knowledge graph model is defined to represent the rational of anomaly detection for the PIA pilot.

Table 5 – Anomaly detection scenario for PIA pilot

Sensor Data	Possible Error detection (**Error**)	Required additional data inputs （**Reference data**）	Drift Measure （**Fault status)** extral （ontology for natural data）	Possible Causes (**Fault**)	Possible outcomes (**Failure**)
Mechanical lock	**The error is detected by the PLC. Then the yarn brokerage data is sent to MES**	**Failure cause tyoe the i sinputed manually by the human after the visual inspection of the stopepd loom**	**The error is detected by the PLC. Then the yarn brokerage data is sent to MES**	**Digital 0 (stopped)**	**The error is detected by the PLC. Then the yarn brokerage data is sent to MES**

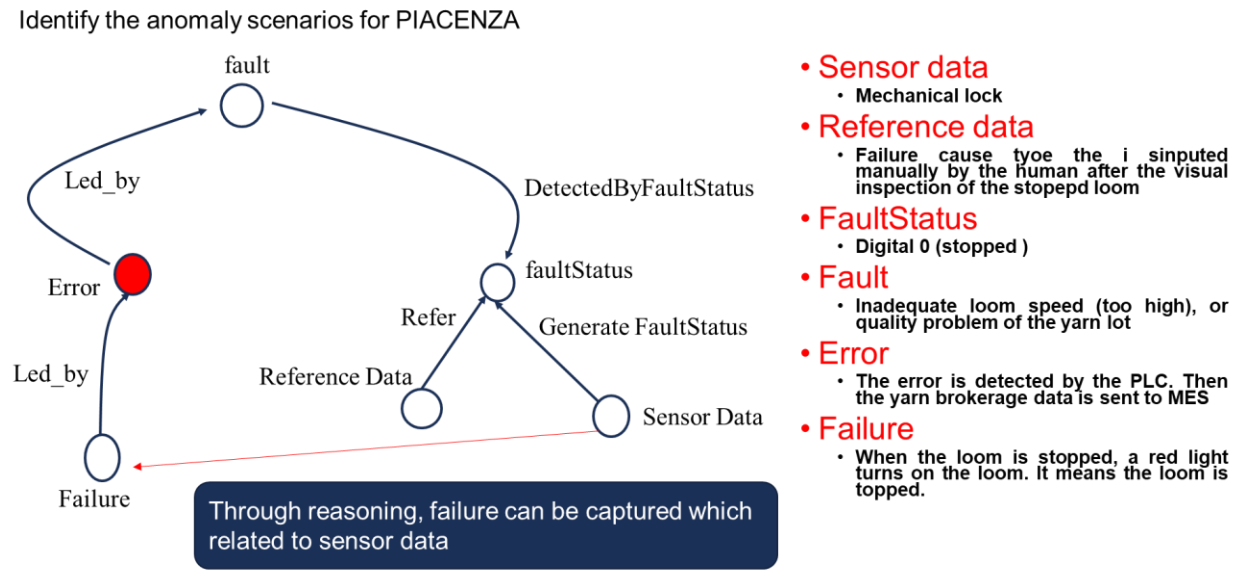

Identify the anomaly scenarios for PIACENZA
fault
Led_by
DetectedByFaultStatus
Error
faultStatus
Refer
Generate FaultStatus
Led_by
Reference Data
Sensor Data
Failure
Through reasoning, failure can be captured which related to sensor data
• Sensor data
• Mechanical lock
• Reference data
• Failure cause tyoe the i sinputed manually by the human after the visual inspection of the stopepd loom
• FaultStatus
• Digital 0 (stopped)
• Fault
• Inadequate loom speed (too high), or quality problem of the yarn lot
• Error
• The error is detected by the PLC. Then the yarn brokerage data is sent to MES
• Failure
• When the loom is stopped, a red light turns on the loom. It means the loom is topped.

Figure 41: Anomaly scenario for PIA pilot

As shown in Figure 41, based on Table 5, an anomaly scenario is defined. The knowledge graph model is built based on the given data and anomaly ontology. The relationships between failure and sensor data are not defined in the knowledge graph. Thus, a reasoning is executed based on the developed knowledge graph models in order to capture a sensor data has a failure.

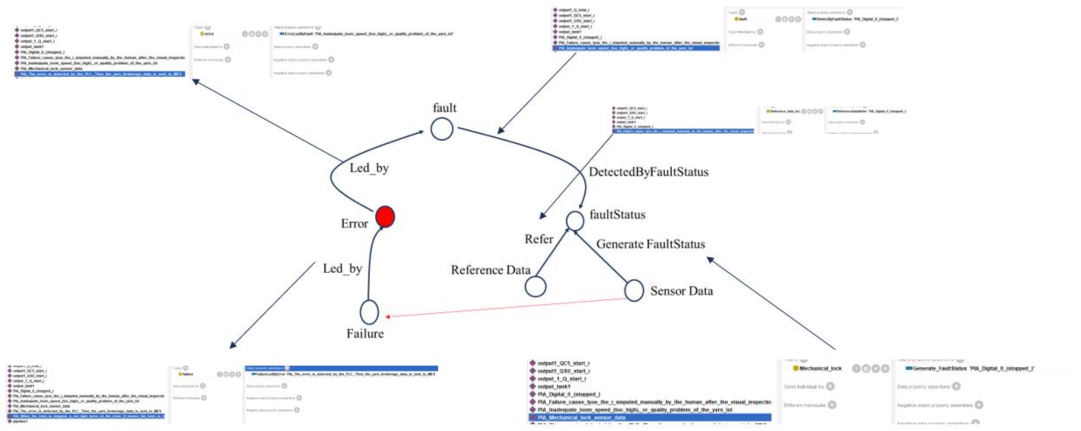
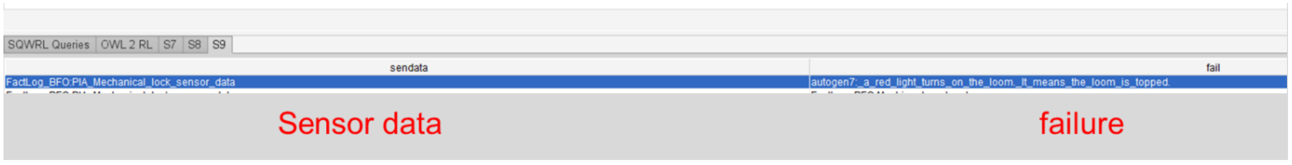


Figure 42: Knowledge graph models for PIA pilot

As shown in Figure 42, knowledge graph models are defined to define the anomaly scenario. Through SQWRL rule, a reasoning is implemented:

FACTLOG_BFO:Mechanical_lock (?sendata) ^ FACTLOG_BFO:Generate_FaultStatus (?sendata, ?fsta) ^ Reference_data_for_anomaly_analysis (?refedata) ^ FACTLOG_BFO:ReferenceDataRefer (?refedata, ?fsta) ^ FACTLOG_BFO:fault_status(?fsta) ^ FACTLOG_BFO:DetectByFaultStatus(?fau, ?fsta) ^ FACTLOG_BFO:fault(?fau) ^ FACTLOG_BFO:ErrorLedByFault(?err, ?fau) ^ FACTLOG_BFO:error(?err) ^ FACTLOG_BFO:FailureLedByError(?imser, ?fal) ^ FACTLOG_BFO:failure(?fail) -> sqwrl:select(?sendata,?fail)



sendata	fail
FactLog_BFO:PIA_Mechanical_lock_sensor_data	autogen7:_a_red_light_turns_on_the_loom._It_means_the_loom_is_topped.

Figure 43: Reasoning result for PIA pilot

Finally, a reasoning result is obtained. From the result, we can understand the sensor data (Mechanical_lock) has a failure: “When the loom is stopped, a red light turns on the loom. It means the loom is topped.”.

6.4.2 Cognition services based on Knowledge Graph Models in Graphical Database

6.4.2.1 *Anomaly detection for CONT Pilot*

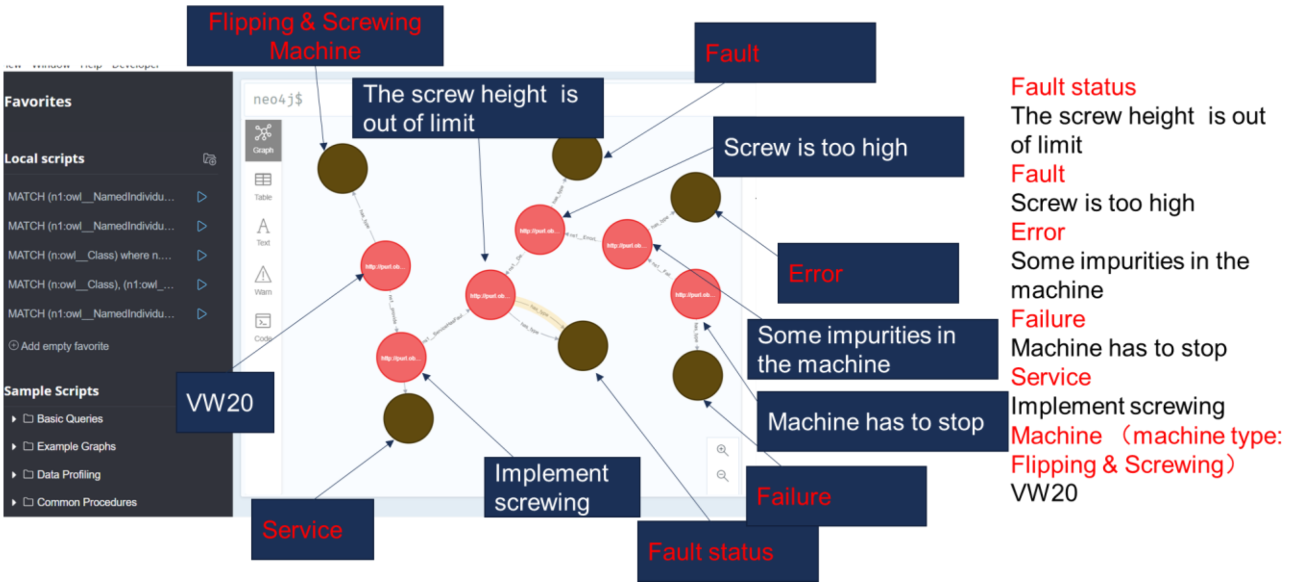

Flipping & Screwing Machine
Fault
The screw height is out of limit
Screw is too high
Error
Some impurities in the machine
Machine has to stop
VW20
Implement screwing
Service
Failure
Fault status
neo4j$
Favorites
Local scripts
MATCH (n1:owl__NamedIndividu...
MATCH (n1:owl__NamedIndividu...
MATCH (n:owl__Class) where n....
MATCH (n:owl__Class), (n1:owl_...
MATCH (n1:owl__NamedIndividu...
Add empty favorite
Sample Scripts
Basic Queries
Example Graphs
Data Profiling
Common Procedures
Fault status
The screw height is out of limit
Fault
Screw is too high
Error
Some impurities in the machine
Failure
Machine has to stop
Service
Implement screwing
Machine （machine type: Flipping & Screwing）
VW20

Figure 44: Reasoning result for CONT pilot

As shown in Figure 44, through the integration of KGM and cognition services, the OWL model (Section 6.4.1.1) for anomaly detection is used to generate Neo4j knowledge graph model. Through the reasoning in Neo4j by using cypher, the *VW20 machine* has fault status *The screw height is out of limit*.

6.4.2.2 *Sensor values forecasting for JEMS pilot*

While the cognitive services exposed for sensor values forecasting work based on induction, leveraging machine learning models to issue predictions based on patterns learned from past data, such models are created based on domain knowledge encoded in knowledge graphs and specific endpoints described and presented in D4.4. In Figure 45 we show a high-level architecture diagram based on the one introduced in Section 6.3, and relating it to our knowledge graph and external services consuming it.

We leveraged the owlready2 library to create an in-memory knowledge graph based on a custom ontology we developed that represents the relevant entities required to create the machine learning model (see Figure 46). The ontology had 31 classes, extended from the BFO upper ontology, and leveraged concepts from four related ontologies: OntoDM, IAO, DAMON, DMOP. Individuals created in the process were persisted, and could be later exported in the OWL format when required, and later imported and merged along with the ontology to the aforementioned Neo4j instance (see Figure 47).

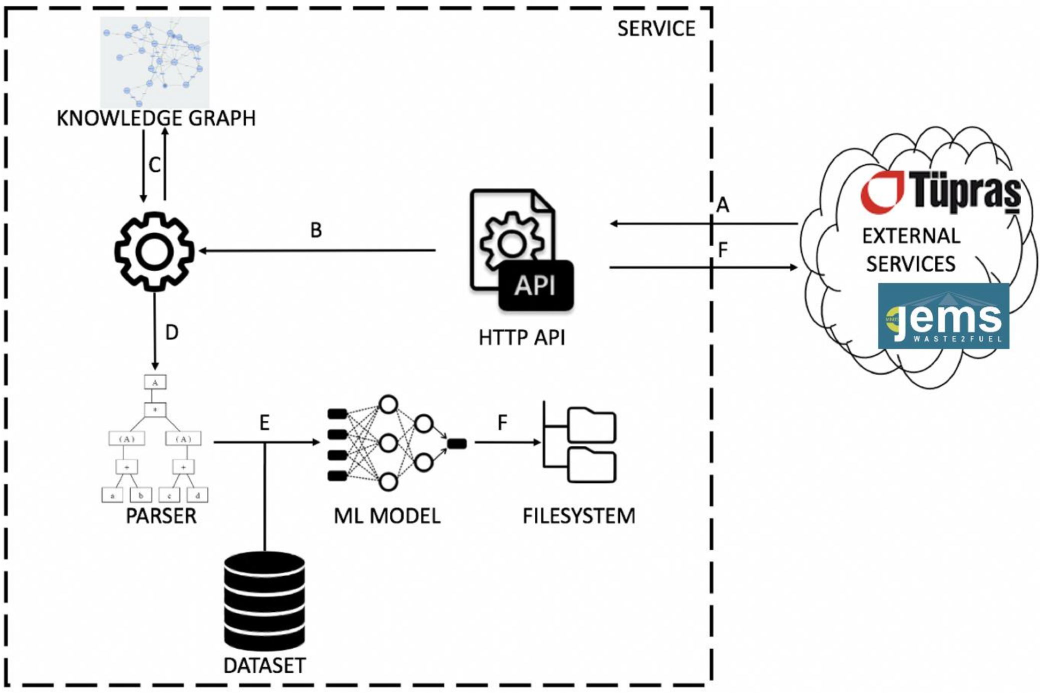


Figure 45. Diagram displaying how the KG-based analytics for process optimization service is used by the use cases we worked for.

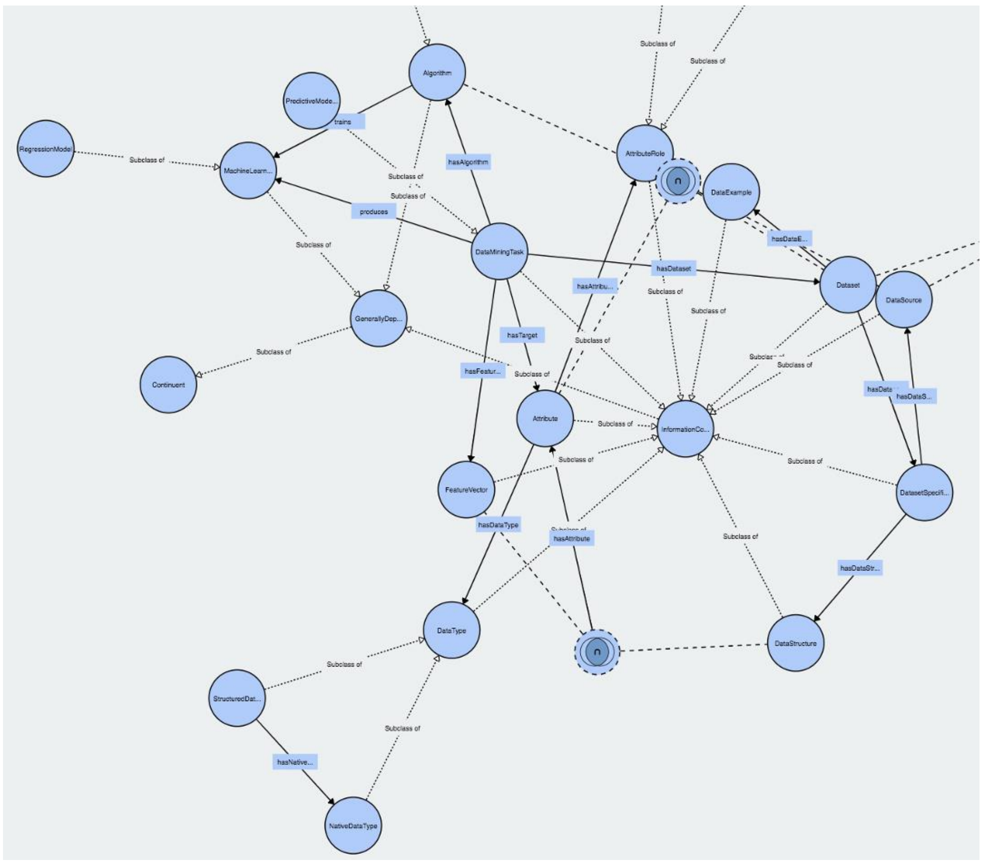
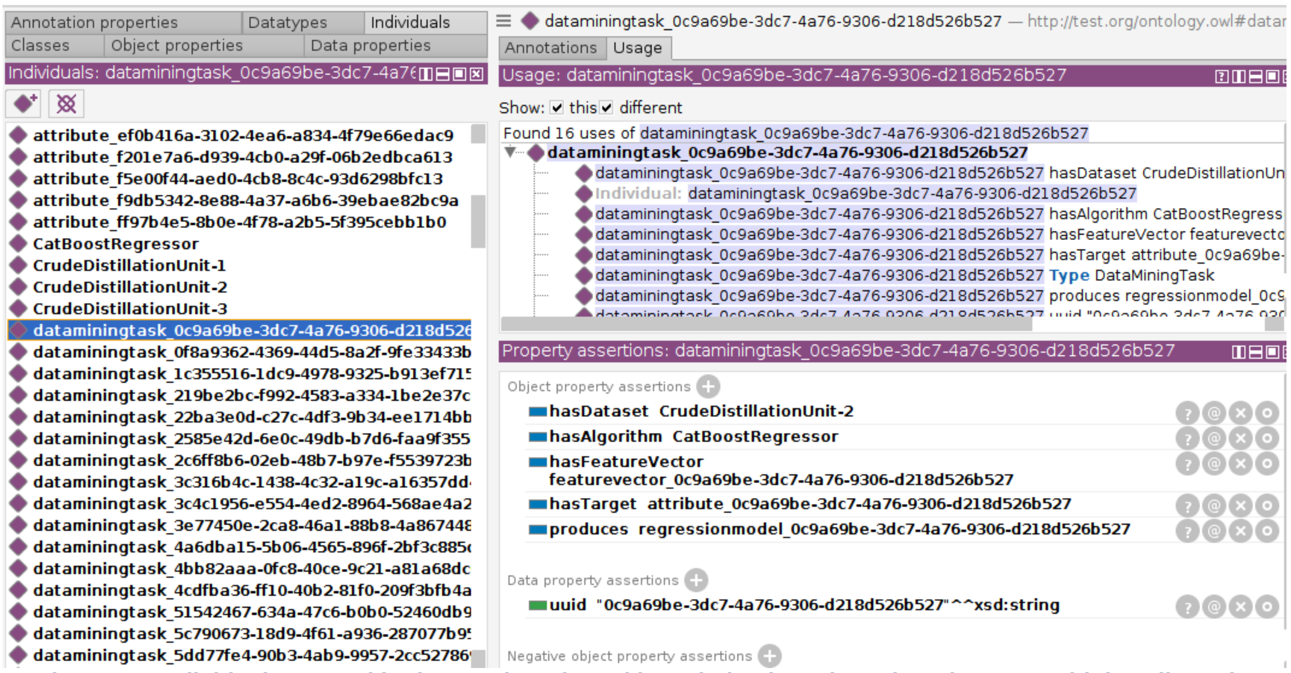


Figure 46. Graph visualization of the ontology we created.



Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Individuals: dataminingtask_0c9a69be-3dc7-4a7

- attribute_ef0b416a-3102-4ea6-a834-4f79e66edac9
- attribute_f201e7a6-d939-4cb0-a29f-06b2edbca613
- attribute_f5e00f44-aed0-4cb8-8c4c-93d6298bfc13
- attribute_f9db5342-8e88-4a37-a6b6-39ebae82bc9a
- attribute_ff97b4e5-8b0e-4f78-a2b5-5f395cebb1b0
- CatBoostRegressor
- CrudeDistillationUnit-1
- CrudeDistillationUnit-2
- CrudeDistillationUnit-3
- dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526
- dataminingtask_0f8a9362-4369-44d5-8a2f-9fe33433b
- dataminingtask_1c355516-1dc9-4978-9325-b913ef71
- dataminingtask_219be2bc-f992-4583-a334-1be2e37c
- dataminingtask_22ba3e0d-c27c-4df3-9b34-ee1714bb
- dataminingtask_2585e42d-6e0c-49db-b7d6-faa9f355
- dataminingtask_2c6ff8b6-02eb-48b7-b97e-f5539723b
- dataminingtask_3c316b4c-1438-4c32-a19c-a16357dd
- dataminingtask_3c4c1956-e554-4ed2-8964-568ae4a2
- dataminingtask_3e77450e-2ca8-46a1-88b8-4a86744
- dataminingtask_4a6dba15-5b06-4565-896f-2bf3c885
- dataminingtask_4bb82aaa-0fc8-40ce-9c21-a81a68dc
- dataminingtask_4cdfba36-ff10-40b2-81f0-209f3bfb4a
- dataminingtask_51542467-634a-47c6-b0b0-52460db9
- dataminingtask_5c790673-18d9-4f61-a936-287077b9
- dataminingtask_5dd77fe4-90b3-4ab9-9957-2cc52786

dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 — http://test.org/ontology.owl#datar

Annotations | Usage

Usage: dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527

Show: ✓ this ✓ different

Found 16 uses of dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527

- dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 hasDataset CrudeDistillationUn
  - Individual: dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 hasAlgorithm CatBoostRegress
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 hasFeatureVector featurevecto
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 hasTarget attribute_0c9a69be-
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 Type DataMiningTask
  - dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527 produces regressionmodel_0c9

Property assertions: dataminingtask_0c9a69be-3dc7-4a76-9306-d218d526b527

Object property assertions

- hasDataset CrudeDistillationUnit-2
- hasAlgorithm CatBoostRegressor
- hasFeatureVector featurevector_0c9a69be-3dc7-4a76-9306-d218d526b527
- hasTarget attribute_0c9a69be-3dc7-4a76-9306-d218d526b527
- produces regressionmodel_0c9a69be-3dc7-4a76-9306-d218d526b527

Data property assertions

- uuid "0c9a69be-3dc7-4a76-9306-d218d526b527"^^xsd:string

Negative object property assertions

Figure 47. Individuals created in the ontology-based knowledge base based on data on multiple calls to the HTTP API.

The service described above enables to use API calls to retrain a model based on particular data (e.g., new data available) and later deployed to replace the stale ones. The architecture supporting the deployment of such machine learning models was described in detail in D3.2, along with the results obtained for the different use cases. Below we reproduce the results obtained for particular motor power, pressure, and temperature sensors (Figures 48-50).

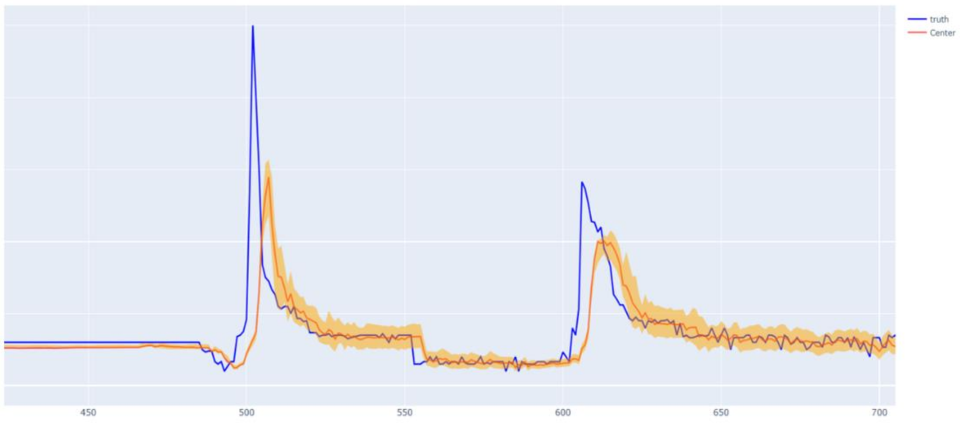


Figure 48. Motor power sensor

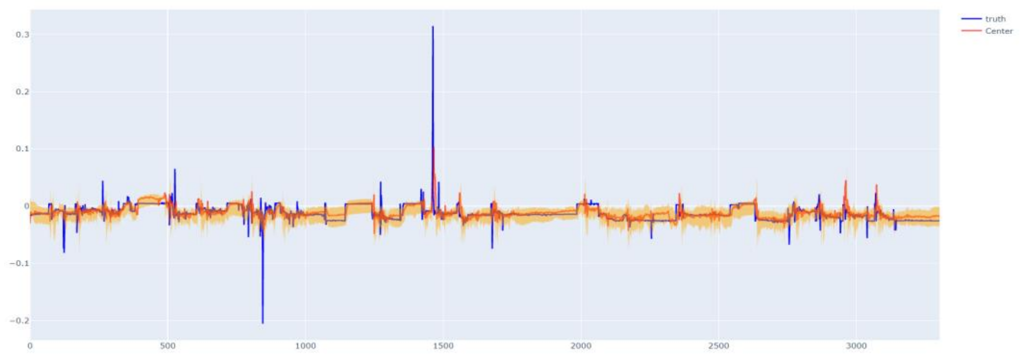


Figure 49. Pressure sensor

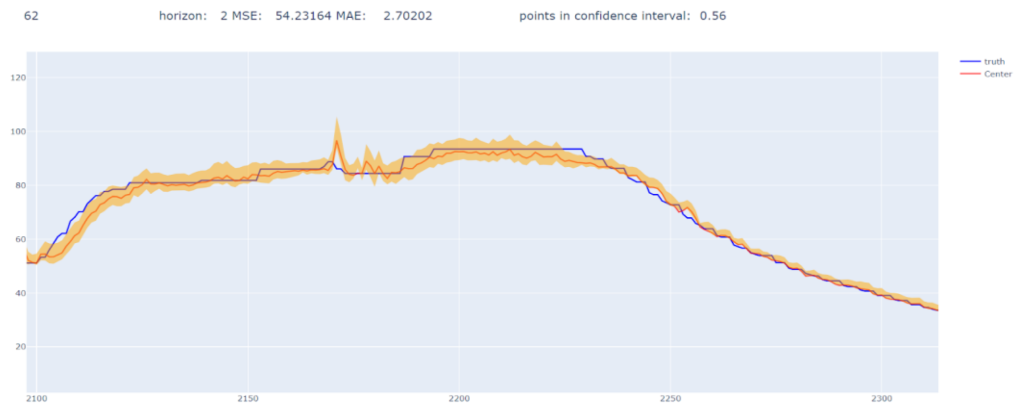


Figure 50. Temperature sensor

Ontology-defined ML model and analytical pipeline enables to apply the initial cognitive services designed in a more generalized approach. While the initial configuration demanded predefined models and feature vector configurations, the upgraded services with knowledge graph semantics enable configuration using API abstraction. This key difference enables the integrator to self-define (automatically configure) the internal analytical data structures, to match the use case requirements. A step towards generalizable services use.

7 Conclusion

This report demonstrates basic background of ontology, ontology engineering and knowledge graph modelling for factory cognition first. Then ontology for the entire FACTLOG project is introduced including pilot description, optimization, process modelling, etc. Moreover, knowledge graph models which are developed for each pilot are introduced. Finally, integration of knowledge graph models and FACTLOG platform is reported including cognition services based on OWL and Neo4j. In summary, we have three important outputs from this deliverable:

- Top level ontology, such as BFO and IOF ontology, is used to construct an ontology framework to support the standardization of our ontology development.
- Five pilots and four technical partners including data analysis, process models, optimization and anomaly detection are defined based on the developed ontology framework.
- The developed knowledge graph models are integrated with FACTLOG platform. Thus, all the FACTLOG components can access the knowledge graph models to capture related data.

Based on the lessons learned from the FACTLOG project, through the top-level ontology, the knowledge graph models have a good scalability. All the pilot concepts and domain concepts of technical partners can be integrated under a unified ontology framework.

References

1. Aasman J (2020) Why Knowledge Graphs Hit the Hype Cycle and What they have in common. In:https://ontologforum.s3.amazonaws.com/OntologySummit2020/Introduction/Why-Knowledge-Graphs-Now--JansAasman_20190904.pdf
2. Aijal J (2020) What is a knowledge graph and how does one work? In: https://thenextweb.com/podium/2019/06/11/what-is-a-knowledgegraph- and-how-does-one-work/. https://www.lifewire.com/what-is-a-foldable-phone-4178374
3. Arp R, Smith B, Spear AD (2016) Building Ontologies with Basic Formal Ontology. The MIT Press, Cambridge
4. Baclawski K, Bennett M, Berg-Cross G, Schneider T, Sharma R, Singer J, Sriram RD (2021) Ontology summit 2020 communiqué: Knowledge graphs. Appl Ontol 16:229–247. doi: 10.3233/AO-210249
5. Blumauer A (2014) From Taxonomies over Ontologies to Knowledge Graphs. Semant Web Co 1–3
6. Chandrasekaran B, Josephson JR, Benjamins VR (1999) What aro ontologies, and why do we need them? IEEE Intell Syst Their Appl 14:20–26. doi: 10.1109/5254.747902
7. Färber M, Ell B, Menne C, Rettinger A, Bartscherer F (2017) Linked Data Quality of DBpedia , Freebase ,. Semant Web 0:1–53
8. Krötzsch M, Thost V (2016) Ontologies for Knowledge Graphs: Breaking the Rules. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp 376–392
9. Medvedev D, Shani U, Dori D (2021) Gaining insights into conceptual models: A graph ‑ theoretic querying approach. Appl Sci 11:1–31. doi: 10.3390/app11020765
10. Mike Bergman (2019) A Common Sense View of Knowledge Graphs. In: http://bit.ly/307PEBs and http://bit.ly/2RAbE6X. https://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/#kg18
11. O'Connor M, Das A (2009) SQWRL: A query language for OWL. In: CEUR Workshop Proceedings
12. Okoye K, Islam S, Naeem U, Sharif MS (2020) Semantic-based process mining technique for annotation and modelling of domain processes. Int J Innov Comput Inf Control 16:899–921. doi: 10.24507/ijicic.16.03.899
13. Paulheim H (2016) Knowledge graph refinement: A survey of approaches and evaluation methods. Semant Web 8:489–508. doi: 10.3233/SW-160218

14. Pujara J, Miao H, Getoor L, Cohen W (2013) Knowledge Graph Identification. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp 542–557

15. Rožanec JM, Jinzhi L, Košmerlj A, Kenda K, Dimitris K, Jovanoski V, Rupnik J, Karlovčec M, Fortuna B (2020) Towards actionable cognitive digital twins for manufacturing. CEUR Workshop Proc 2615:1–12

16. Zou M, Basirati MR, Bauer H, Kattner N, Reinhart G, Lindemann U, Böhm M, Krcmar H, Vogel-Heuser B (2019) Facilitating Consistency of Business Model and Technical Models in Product-Service-Systems Development: An Ontology Approach. IFAC-PapersOnLine 52:1229–1235. doi: 10.1016/j.ifacol.2019.11.366

17. Лобанов ГЮ (2013) Basic Formal Ontology как средство построения онтологии в системной модели аргументации. РациоRu 126–143

18. (2020) Knowledge Graphs and Machine Learning - Towards Data Science. https://towardsdatascience.com/knowledge-graphs-and-machine-learning-3939b504c7bc

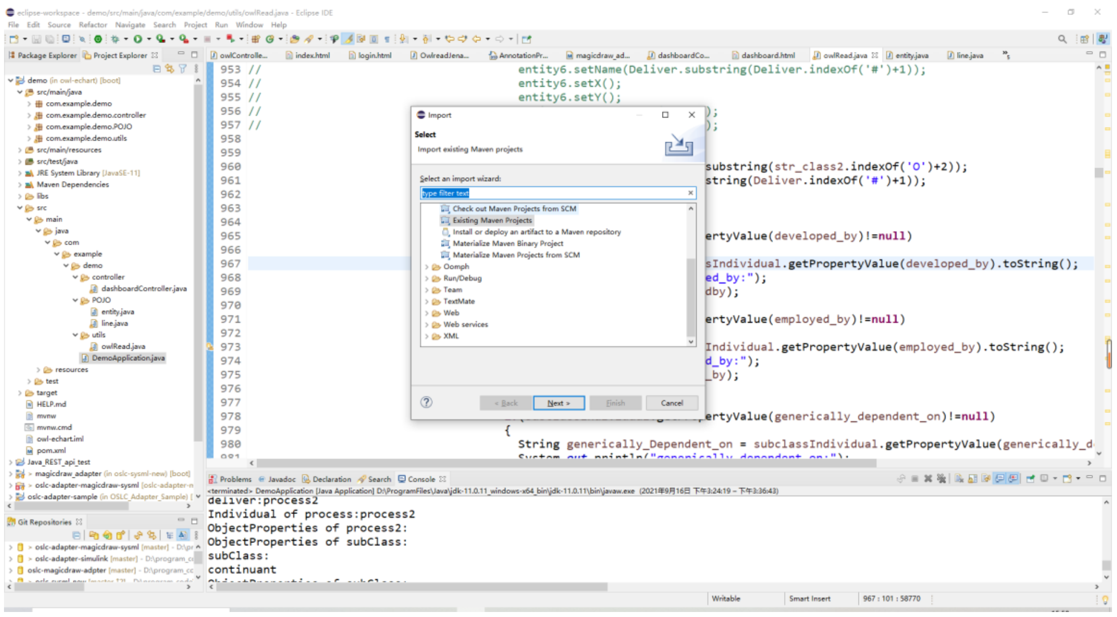
Appendix I – Integrating KGM based on OWL

We have made a project for the visualization for BFO. In this project, two resource classes (Entity and Line) were constructed for the parse of BFO model. The OWL file was read by the *Jena* ontology API. The objects of Entity Class and Line Class stored the topological relation of BFO model. The open source JavaScript visualization library ECHART was used for visualization in this project. Due to the large number of properties of BFO, we only show the properties needed in this particular OWL file in this demo. The details of the project are shown as follow.

HOW TO RUN THE DEMO IN ECLIPSE

(1) Import project

In Eclipse, select the File -> Import. Then the dialog box is shown as Figure 51. Select the *Existing Maven Project* and choose the demo directory and the maven project is import in the Eclipse. It may take some time to load the libraries.



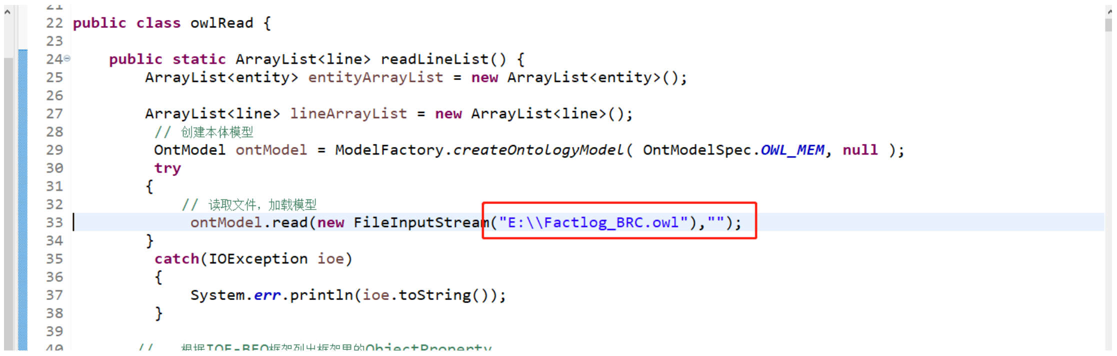
```
entity6.setName(Deliver.substring(Deliver.indexOf('#')+1));
entity6.setX();
entity6.setY();
```

```
<terminated> DemoApplication [Java Application] D:\ProgramFiles\Java\jdk-11.0.11_windows-x64_bin\jdk-11.0.11\bin\javaw.exe (2021年9月16日 下午3:24:19 – 下午3:36:43)
deliver:process2
Individual of process:process2
ObjectProperties of process2:
ObjectProperties of subClass:
subClass:
continuant
```

Figure 51: Eclipse project

(2) modify the file path

Modify the OWL file path in line 33 and line 1828 in owlRead.java. Change file path to your own file path of the OWL file.

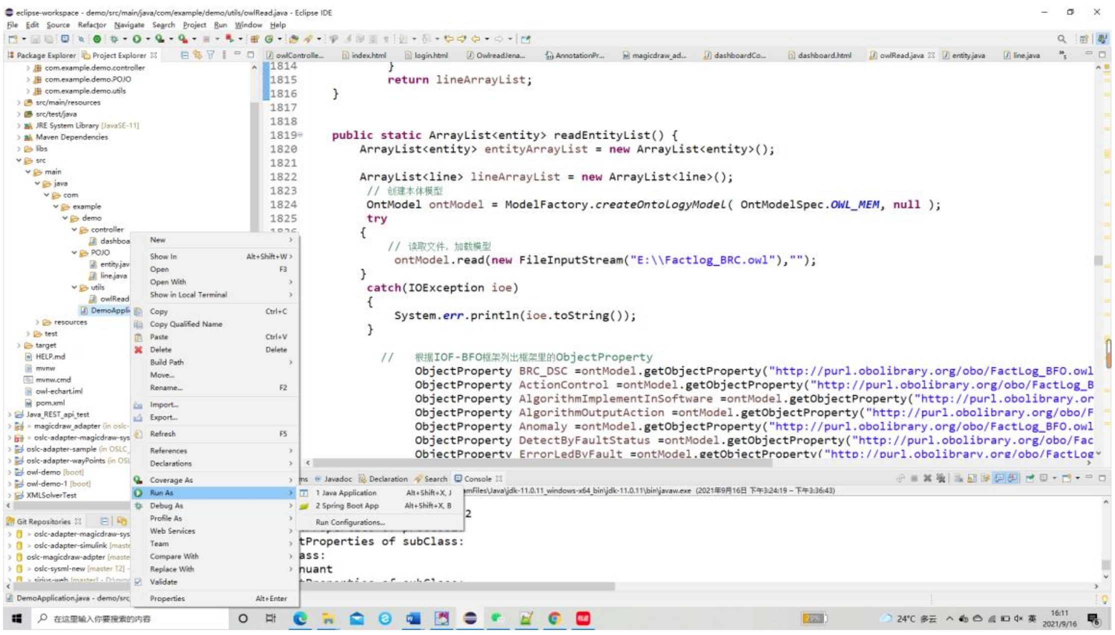


```
22 public class owlRead {
23
24     public static ArrayList<line> readLineList() {
25         ArrayList<entity> entityArrayList = new ArrayList<entity>();
26
27         ArrayList<line> lineArrayList = new ArrayList<line>();
28          // 创建本体模型
29          OntModel ontModel = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM, null );
30          try
31         {
32              // 读取文件, 加载模型
33              ontModel.read(new FileInputStream("E:\\Factlog_BRC.owl"),"");
34         }
35          catch(IOException ioe)
36          {
37              System.err.println(ioe.toString());
38          }
39
```

Figure 52: Change the path for loading OWL files.

(3) run the application

Run the springboot project as Java Application, shown as Figure 53.



```java
        }
        return lineArrayList;
    }


    public static ArrayList<entity> readEntityList() {
        ArrayList<entity> entityArrayList = new ArrayList<entity>();

        ArrayList<line> lineArrayList = new ArrayList<line>();
         // 创建本体模型
         OntModel ontModel = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM, null );
         try
        {
            // 读取文件, 加载模型
            ontModel.read(new FileInputStream("E:\\Factlog_BRC.owl"),"");
        }
         catch(IOException ioe)
         {
            System.err.println(ioe.toString());
         }

           //    根据IOF-BFO框架列出框架里的ObjectProperty
                ObjectProperty BRC_DSC =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/FactLog_BFO.owl
                ObjectProperty ActionControl =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/FactLog_B
                ObjectProperty AlgorithmImplementInSoftware =ontModel.getObjectProperty("http://purl.obolibrary.or
                ObjectProperty AlgorithmOutputAction =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/F
                ObjectProperty Anomaly =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/FactLog_BFO.owl
                ObjectProperty DetectByFaultStatus =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/Fac
                ObjectProperty ErrorLedByFault =ontModel.getObjectProperty("http://purl.obolibrary.org/obo/FactLog
```

```
mFiles\Java\jdk-11.0.11_windows-x64_bin\jdk-11.0.11\bin\javaw.exe  (2021年9月16日 下午3:24:19 – 下午3:36:43)
2
tProperties of subClass:
ass:
nuant
```

eclipse-workspace - demo/src/main/java/com/example/demo/utils/owlRead.java - Eclipse IDE
Run As
1 Java Application Alt+Shift+X, J
2 Spring Boot App Alt+Shift+X, B
Run Configurations...
Console

Figure 53: Compile the whole project

(4) Result Browser

Open a browser and type in the url localhost:8080. Then the result is shown as follow which is localhost:8080.

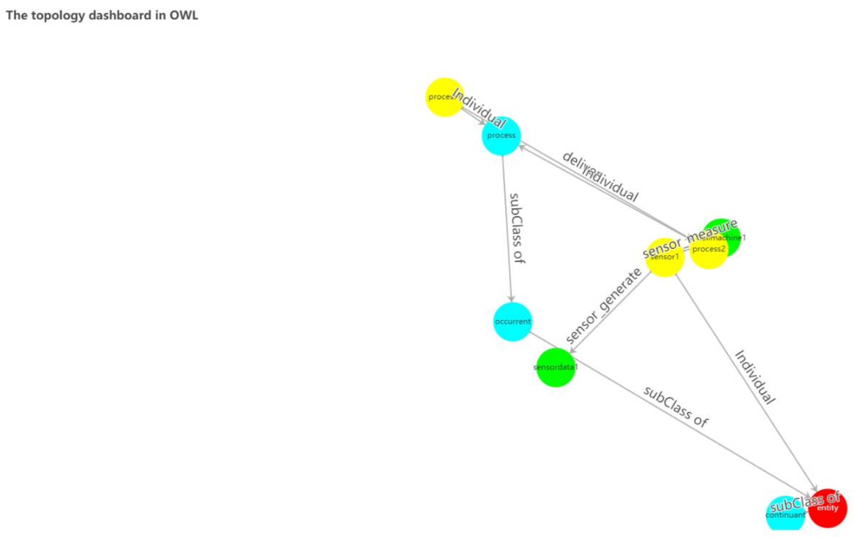


Figure 54: Visualize the OWL information

Appendix II – Knowledge Graph Modeling tools

1. Protégé

   Protégé is supported by a strong community of academic, government, and corporate users, who use Protégé to build knowledge-based solutions in areas as diverse as biomedicine, e-commerce, and organizational modeling https://protege.stanford.edu/ In our project, we make use of it to model knowledge graph models using OWL.

   Unzip the Protege-5.5.0-win and open Protege.exe. Then start your knowledge graph modeling journey.

2. Twinkle

   Twinkle is a simple GUI interface that wraps the ARQ SPARQL query engine. The tool should be useful both for people wanting to learn the SPARQL query language, as well as those doing Semantic Web development http://www.ldodds.com/projects/twinkle/. In this project, we make use of it to implement SPARQL query.

   Unzip the twinkle-2.0-src and open twinkle.jar. Then start your query journey.

3. Neo4j

   Neo4j is the only enterprise-strength graph database that combines native graph storage, advanced security, scalable speed-optimized architecture, and ACID compliance to ensure predictability and integrity of relationship-based queries. In FACTLOG, it is used to integrate knowledge graph models with FACTLOG platform https://neo4j.com/ .