Z - B R E 4 K

Grant agreement nº: 768869

Call identifier: H2020-FOF-2017

Strategies and Predictive Maintenance models wrapped around physical systems for Zero-unexpected-Breakdowns and increased operating life of Factories

Z-BRE4K

Deliverable D3.2

Knowledge base system (KBS) with asset/product/process signatures

Work Package 3

Knowledge and predictive modelling

Document type	:	Report
Version	:	V0.3
Date of issue	:	27th June 2019
Dissemination level	:	Public
Lead beneficiary	:	12 - HOLONIX

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement nº 768869.

EXECUTIVE SUMMARY

Abstract	This document reports the work carried out in the framework of Task 3.2, 'Knowledge base system with asset/product/process signatures'. Within this task a Knowledge Based System (KBS) to store data about machines/systems behaviour has been developed. Different pieces of information will be stored, reference models and snapshots describing critical components conditions as well as machines parameters. A triple store DB has been implemented to store triplets according to the ontology developed in T3.1 Connectors and reasoner will also be implemented within this task. Next step will be a continuous improvement of the KBS with a testing and integration validation expected with WP5 in M31.
Keywords	Knowledge Based system, triple store, data storage, ontology, predictive maintenance, machines/asset, reasoner, connectors.

REVISION HISTORY

Version	Author(s)	Changes	Date
V0.1	Alice Reina (HOLONIX)	Deliverable outline	18/09/2018
V0.2	Serena Albertario, Weian Xu (HOLONIX)	Deliverable first draft	21/06/2019
V0.3	Daniel Gesto, Jovana Milenkovic (AIMEN); Konstantinos Greventis (ATLANTIS); Veerendra Angadi (BRUNEL)	Review	24/06/2019
V0.4	Serena Albertario (HOLONIX)	Final version after review	24/06/2019

TABLE OF CONTENTS

EXECUTIVE SUMMARY **2**
REVISION HISTORY **3**
TABLE OF CONTENTS **4**
LIST OF FIGURES **5**
LIST OF TABLES **5**
ABBREVIATIONS **5**
1 INTRODUCTION **6**
1.1 Task 3.2 objectives *6*
1.2 Deliverable contents *6*
2 STATE-OF-THE-ART ON KBS **7**
3 KBS IN THE FRAMEWORK OF Z-BRE4K PROJECT **8**
3.1 Software toolkit for ontology implementation *10*
3.2 Triple store DB *11*
3.3 Connectors *14*
3.4 Reasoner *18*
4 CONCLUSION **21**
5 REFERENCE **22**

List of Figures

Figure 1: KBS structure and communications 9
Figure 2: Knowledge Base System component diagram 10
Figure 3: Architecture of Apache Jena 11
Figure 4: Upload RDF file with Fuseki UI 12
Figure 5: Snipet of a RDF file containing some of sensor data 13
Figure 6: Snipet of application where the generated RDF file is POSTed to the Fuseki data load endpoint 13
Figure 7: Representation of a sensor data set within triple store 14
Figure 8: Sample of entity for SACMI/CDS machine within Orion context broker 15
Figure 9: Snipet of code to execute the HTTP PATCH request versus Orion Context Broker writing in JavaScript (Node.js) 16
Figure 10: Sample of a Subscription 17
Figure 11: Snipet of a simple Node.js application 17

List of Tables

Table 1: Competency questions of use case SACMI 18
Table 2: Examples of rules 19

Abbreviations

Abbreviation	Name
AI	Artificial Intelligence
DB	Data Base
KB	Knowledge Base
KBS	Knowledge Base System
RDF	Resource Description Framework
UI	User Interface
DBMS	Database management system
HTTP	HyperText transfer protocol

1 INTRODUCTION

This document describes the activities' results done in Task 3.2 "Knowledge base system (KBS) with asset/product/process signatures.

WP 3 deals with the design and implementation of knowledge and predictive modelling.

To summarize, the main objectives of WP3 are:

- Semantic modelling to empower KBS for cognitive manufacturing.
- Incorporation of risk analysis, KRIs and FMECA.
- Incorporation of predictive/prescriptive analytics to support decision making and associated Z-Strategies.

1.1 Task 3.2 objectives

The main objective of Task 3.2 is the development of a KBS software system, to store data of asset/product/process behaviours along time. The work has been done starting from already existing solution and adapting them to ensure full interoperability with other Z-BRE4K software/hardware systems. It will include context-aware ontology to support predictive maintenance and extended operating life of assets in production facilities and the relevant decision support engine. The reasoner, as base of the KBS, will be created using Task 3.1 result: ontology and using annotations to describe the optimal conditions of assets and product as well as reference to the context and processes where they operate.

1.2 Deliverable contents

The document's main purpose is to present the Knowledge Based System that will be implemented during the Z-BRE4K project. KBS will be supported by I-Like system, an already existent platform for Lifecycle Knowledge Management, an item-level PLM system that stores information about individual instances of asset, products and machines. It extracts knowledge to be consumed along different manufacturing processes. It correlates design-time information with information collected from sensors that measures parameters of products and the surrounding environment during all the pre and post production phases of its life.

Here below the structure of the deliverable is shown:

- Introduction
  - Objectives of Task 3.2
  - Contents of the deliverable
- State-of-the art on KBS
- KBS in the framework of Z-BRE4K project
  - SW toolkit for ontology implementation
  - Triple store DB
  - Connectors
  - Reasoner
- Conclusion
- Reference

2 STATE-OF-THE-ART ON KBS

The origin of Knowledge Based Systems (KBS), or expert systems, dates back at 1976, when the term was first introduced to distinguish them from commonly used Data Bases (DB).

Aim of those systems is to provide structured knowledge on a specific domain, easy to be understand by people and software, in order to favour the interoperability of the systems. This type of information is structured and codified and it's also called "*object model* or *ontological knowledge*".

The development of KBS in mid '80 is in fact the main reason for the use of ontologies in computer science and artificial intelligence (AI). According to Gruber definition [1] "Ontology is an explicit specification of a conceptualization". Through the use of ontologies, knowledge can be understood, shared and communicated across people and computers.

Regardless of the content stored, a KBS should always aim to represent knowledge explicitly (as tools, data, and ontologies) rather than implicitly (computer code, vague human experience) - all for the benefit of the end users.

A KBS consists of three main components:

- **Knowledge Base (KB)** containing knowledge about the problem to solve stored as ontologies;
- **Inference Engine** to use the knowledge stored to obtain solutions to problems. The inference engine applies logic rules (as assertions and conditions) to the KB to derive answers;
- **User Interface (UI)** the component that people interact with to find and extract knowledge stored in the system.

It's possible to indicate a forth component of a KBS: a **knowledge acquisition module** to develop and update the system.

The first KBS for manufacturing has been developed in order to support design, to help the formation of products concepts, starting from the analysis of specifications, to support process planning, to understand functional requirements for production and to help operational planning. Starting from these already existent systems, there have been some efforts in developing ontologies, at different levels, to support information exchange, to reuse and create new knowledge in manufacturing context.

Nonetheless, the maintenance domain is still under research. In the framework of Z-BRE4K project a semantic model has been proposed to deal with predictive maintenance (to have more detail please refer to D3.1). A KBS, whose details are reported in this document, is implemented to be applied to different manufacturing contexts.

3 KBS IN THE FRAMEWORK OF Z-BRE4K PROJECT

Note: during Z-BRE4K project, HOLONIX focus his attention to the end user SACMI/CDS, while INOVA, ATLANTIS and IMEC on PHILIPS. INNOVALIA, AIMEN and BRUNEL on GESTAMP. In this document, HOLONIX will refer his work made for SACMI/CDS, providing details of iLike integration and including supported functionalities, context and process references, as well as asset/product/process signatures.

In the framework of Z-BRE4K project, a Condition Monitoring software is proposed in order to support different actors of a company to evaluate machines and production systems conditions in order to avoid critical and potential failures. This module in fact, will support production managers during the daily production monitor and assessments, but it will be also used by workers to detect the main machine issues in order to act in short time.

The proposed method novelty lies on the development of an approach to gather various types of data (component, machine, production, quality, product and business data) and to develop an integrated digital representation based on semantic knowledge representation formalisms.

In this context, the reference modeled developed during Task 3.1, using ontologies and annotations to describe the optimal conditions of assets and products as well as reference to the context and processes where they operate. The ontology describes the basic entities of Z-BRE4K and model relevant structures of manufacturing systems and processes, establishing a methodological framework to model not only the actors and procedures at the shop floor, but also machinery and their critical components, their failure modes and their criticality, their signatures of healthy and deteriorated conditions, etc. Asset/product/process signatures are also supported as collection of reference structures and several conditions snapshots representing optimal and deteriorated conditions being detected through the production operations.

In this regard, KBS includes context aware ontology related to support predictive maintenance and assets RUL (remaining useful life, in production facilities) and the relevant decision support system that endows wisdom to the Z-BRE4K approach by making use of historical data and the acquired knowledge for all the operations occurred.

The Condition Monitoring Module, whose main core is the KBS can work as stand-alone, providing directly end-users with information to support maintenance planning, but can be also integrated with the overall Z-BRE4K platform, as presented in Figure 1.

In order to reach a semantic achievement, information has to be stored according in triplets, representing a subject-predicate-object relationship. One of the common frameworks of implementing triple store is the Resource Description Framework (RDF) which is a standard model for data interchange on the web. According to this model, data from the shopfloor (machine data, but also maintenance related information) are converted by an RDFizer component in triplets and store in Triple store repository. On top of the Triple store DB an inference engine (reasoner) supports the extraction of knowledge according to end users requirements and the needs of the other Z-BRE4K components.

Semantic Web Service components will support management of the RDF repository in the way of search, authentication, dataset, revision, and SPARQL.

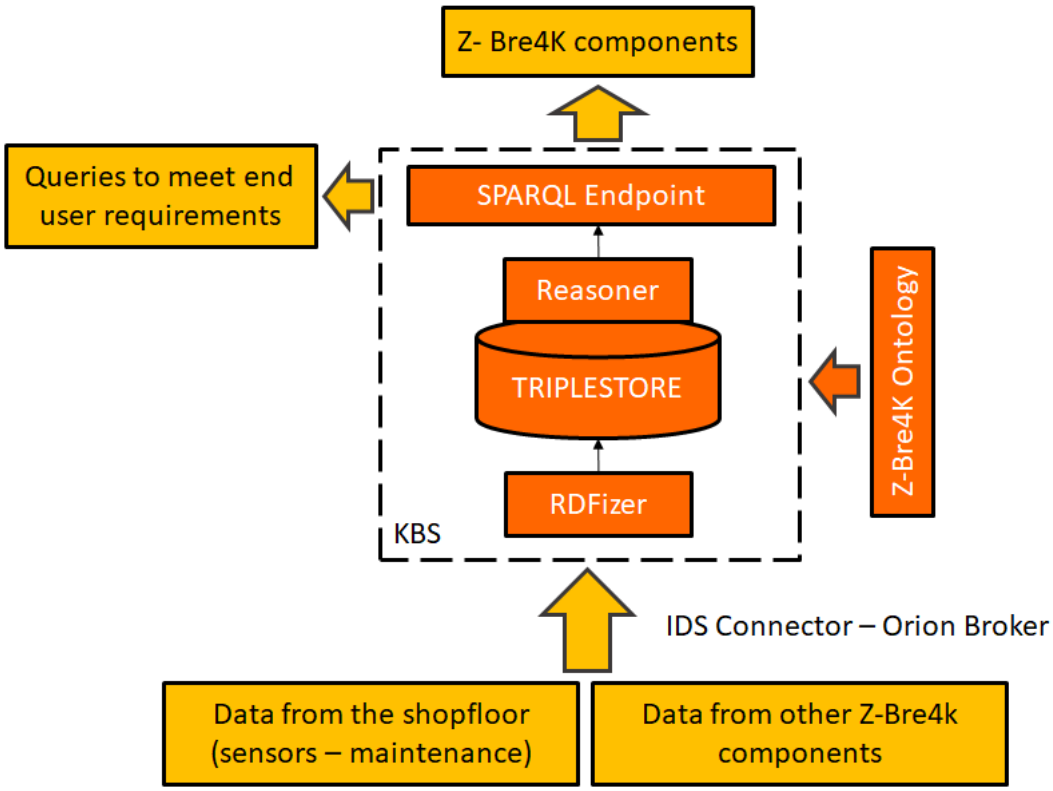

Figure 1: KBS structure and communications

In Z-BRE4K Semantic Framework, Z-BRE4K ontology provides:

- ✓ the main structure of triplex as a reference model,
- ✓ semantic parameters and enables query to satisfy requirements of the maintenance planning support tool.

The main inputs for the KBS are:

- Data from the shopfloor (data concerning machines, processes, production data logs, actors, maintenance activities, etc.)
- Result of analysis from the other Z-BRE4K components (e.g. FMECA, DSS, VRfx, etc)

The main outputs are:

- System knowledge provided directly to the end users
- System knowledge provided to other components

The KBS main functionalities are:

- Transform information in RDF triplets
- Store triplets in a Triple store DB
- Facilitate Reasoning and inference
- Integration of various data sources with semantic interoperability

- Enable queries to meet end users` requirements

In Figure 2 the KBS component diagram is reported:

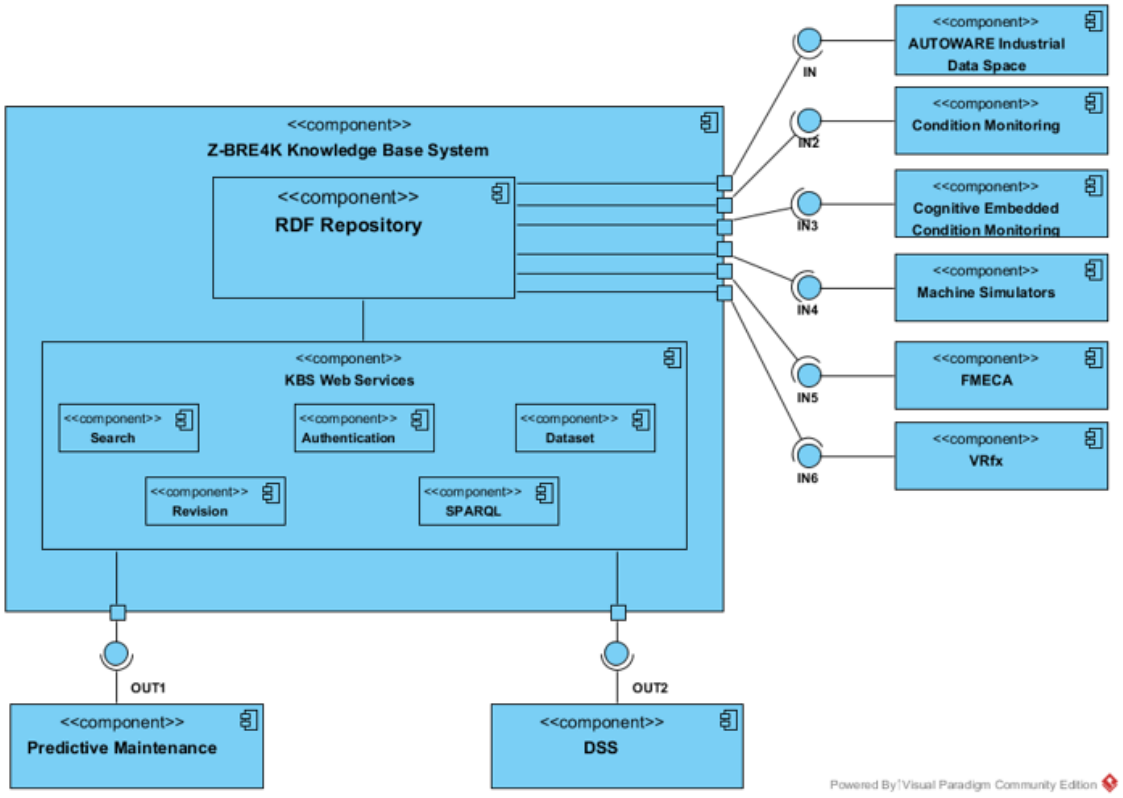

Figure 2: Knowledge Base System component diagram

3.1 Software toolkit for ontology implementation

Apache Jena (or Jena in short) is a free and open source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data.[2]

Figure 3 depicts that the Apache Jena Application framework consists of several different module with different purposes.

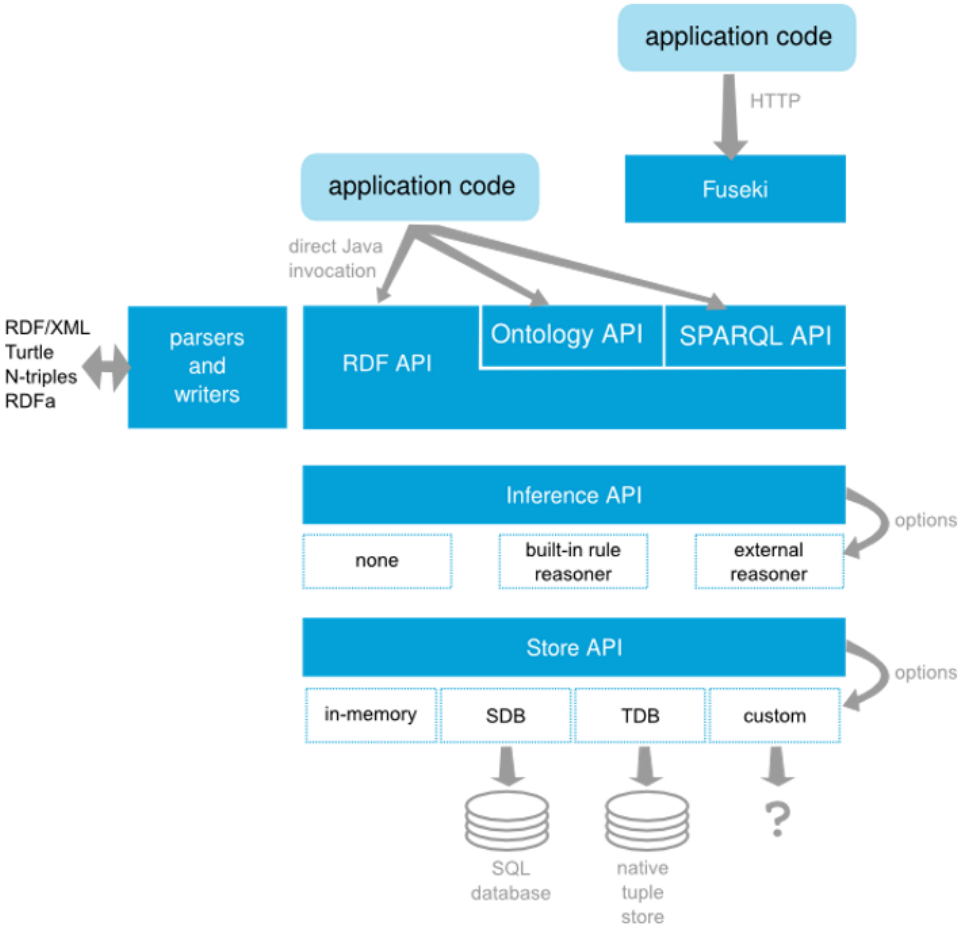

Figure 3: Architecture of Apache Jena

In Task 3.2 the chose has been Apace Jena Fuseki [3]. It can run as an operating system service, as a Java web application (WAR file) and also as a standalone server. It provides security (using Apache Shiro) and has a user interface for server monitoring and administration. It allows also a simpler interface with HTTP RESTful API endpoints that are easily interfaced and independent of programming language. All the programming languages capable of make some HTTP request can interact with KBS instead on only and pure JAVA codes.

3.2 Triple store DB

As the name suggests, the triple store DB is a Database where data are organized in triples which is the basic data unit. It has the format: subject – predicate – object. A triple store is designed to store and retrieve identities that are constructed from triplex collections of strings (sequences of letters). These triplex collections represent a subject-predicate-object relationship that more or less corresponds to the definition put forth by the RDF standard [6]

As mentioned above, in the context of Z-BRE4K project the Apache Jena Fuseki framework has been chosen. This framework integrates its own triple store database which is TDB. The TDB triple store database is a robust, transactional persistent storage layer. It works with SPARQL query language, an equivalent of SQL query language for traditional relational databases.

As for a traditional Relational DBMS, a triple store database can be managed and interrogated with SPARQL query language.

There are mainly two options to insert data into triple store with Apache Jena Fuseki framework:

1. Insert directly triples with an update query over HTTP RESTful API
2. Prepare a RDF document containing all the necessary data then load the RDF document to Apache Jena Fuseki always over HTTP RESTful API then Apache Jena Fuseki will transform RDF fields into triples and persist them to TDB automatically.

A third way could be:

3. Prepare a RDF file and upload it with UI that comes with deployment of Fuseki. This is purely manual work and it should not be used unless for the demonstration purposes.

Figure 4: Upload RDF file with Fuseki UI

The approach with generation of RDF document has been chosen in the context of Z-BRE4K as indicated also by the schema of Figure 1.

The most important reason to this choice is due to the fact that some of data source, for instance some of data set of SACMI/CDS has a quite high value generation frequency (once every second). Inserting directly data set as triple with Fuseki's HTTP API endpoint has a response time that can not guarantee the correct functioning of the data insert.

In the SACMI/CDS use case, the machine is able to generate about 190 messages every minute.

The Fuseki's HTTP API endpoint to insert directly triples has a response time of 300ms which means about 180 triples every minute. Considering also the fact that every message generate four triple records, it's necessary to move to an alternative solution. Without considering the fact that every message has to be transformed into 4 triples. In this context the best solution is to create a RDF document then upload it to the Triple Store and let Fuseki do the transformation of triples.

As represented at Figure 5, the raw data, either from shopfloor or from third party modules, are made available to triple store through a RDFfizer where the raw data are transformed into the RDF format following the Ontology generated in task T 3.1.

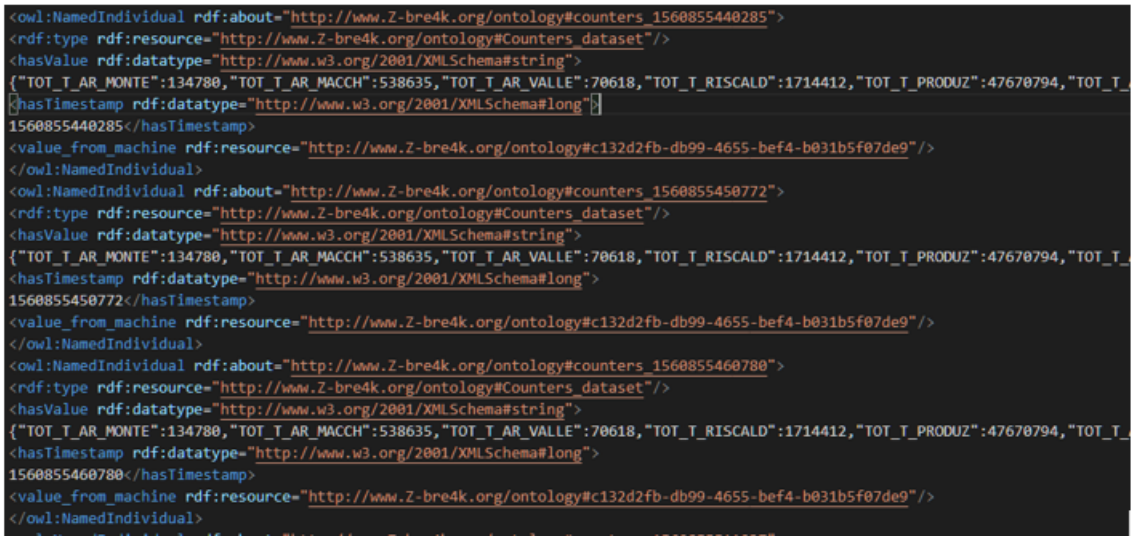

Figure 5: Snipet of a RDF file containing some of sensor data

After the RDF document is generated, it can be sent to Apache Jena fuseki through HTTP endpoint. Apche Jena Fuseki will persist them to the triple data base as triples (Figure 6).

```
fs.readFile("temp.rdf", (err, data) => {
    if (err) {
        reject(err);
        console.log(err);
    } else {
        let req = rp.post(
            {
                "url": this.ApacheJenaFusekiUpdateRDFEndpoint,
                "headers": {
                    "Authorization": "Basic YWRtaW46cHcxMjM="
                }
            },
            function(err, resp, body) {
                if (err) {
                    reject(err);
                } else {
                    resolve();
                }
        });
        let form = req.form();
        form.append('file', fs.createReadStream("temp.rdf"));
    }
});
```

Figure 6: Snipet of application where the generated RDF file is POSTed to the Fuseki data load endpoint

The step of transformation from RDF to triple is executed automatically by Apache Jena Fuseki (Figure 7).

```
:exStatus_1560855509865
        a                       owl:NamedIndividual , :EX_status_dataset ;
        :hasTimestamp           1560855509865 ;
        :hasValue               "\n{\"%M2\":27.020000457763672,\"%M8\":36.5,\"PRESS_I\":45
        :value_from_machine     :f5a8d53f-ef1d-4108-9001-9427050ef809 .
```

Figure 7: Representation of a sensor data set within triple store

3.3 Connectors

The primary data exchange strategy within Z-BRE4K is to share data through Fiware Orion context broker.

The Orion Context Broker is an implementation of the Publish/Subscribe Context Broker GE, providing an NGSI interface. Using this interface, clients can do several operations [4]:

- ✓ Query context information. The Orion Context Broker stores context information updated from applications, so queries are resolved based on that information. Context information consists on *entities* (e.g. a car) and their *attributes* (e.g. the speed or location of the car);
- ✓ Update context information;
- ✓ Get notified when changes on context information take place.

Taking in consideration the SACMI/CDS use case, the Orion context broker has been deployed to a cloud server owned by HOLONIX under docker environment. A set of context information has been initialized: five entities has been initialized, one for every machine with HOLONIX IoT gateway equipped on board.

```
{
    "id": "24c83f7d-688f-42d6-a106-411a5bf21c6f",
    "type": "sacmi_cds_machine",
    "alarms": {⟷},
    "alarmsTimestamp": {⟷},
    "counters": {⟷},
    "countersTimestamp": {⟷},
    "exStatus": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "exStatusTimestamp": {⟷},
    "huStatus": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "huStatusTimestamp": {⟷},
    "setup": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "setupTimestamp": {⟷},
    "state": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "stateTimestamp": {
        "type": "long",
        "value": null,
        "metadata": {}
    },
    "status": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "statusTimestamp": {⟷},
    "thStatus": {
        "type": "json",
        "value": {⟷},
        "metadata": {}
    },
    "thStatusTimestamp": {⟷}
```


Figure 8: Sample of entity for SACMI/CDS machine within Orion context broker

As shown in the sample of Figure 8 every entity consists of definition of data set available as attributes of entity and their data type.

Once defined the entity, a data source can start to update the context of the entity by publishing the new values to the Orion context broker by executing a simple HTTP PATCH request to the Orion context broker passing as body of request the new values for the attributes of the entity.

To be accepted by Orion Context Broker, an update message must follow the rules of NGSI information model of Fiware [5] Every attribute must have specified the type of data it contains and, if it's available, the metadata information about the attribute.

```
genericNGSIPatch(path, messageBody) {
    return new Promise(async (resolve, reject) => {
        let requestPath = this.OrionBrokerBasePath + path;
        var options = {
            method: "PATCH",
            uri: requestPath,
            body: messageBody,
            resolveWithFullResponse: true,
            json: true
        };

        try {
            let resp = await rp(options);
            resolve(resp.data);
        }
        catch (err) {
            console.log("[x] Error happend... PATCHing");
            reject(err);
        }
    });

}
```

Figure 9: Snipet of code to execute the HTTP PATCH request versus Orion Context Broker writing in JavaScript (Node.js)

For the publish/subscribe nature of Orion Context Broker, to receive updates on interested attributes of a given entity (machine), a data consumer has to make an explicit subscription to the Orion Context Broker declaring the interested attribute list and conditions that satisfy the reception of attributes' updates, as well as a HTTP POST endpoint where the Orion Context broker will notify the newly updated attributes.

The subscription is done by executing a very simple HTTP POST request versus Orion Context Broker.

```
{
    "id": "5d08ad76bb51ef7a9f83649c",
    "description": "One subscription to rule them all",
    "expires": "2040-04-05T14:00:00.00Z",
    "status": "active",
    "subject": {
        "entities": [
            {
                "id": ".*",
                "type": "sacmi_cds_machine"
            }
        ],
        "condition": {
            "attrs": [
                "exStatusTimestamp",
                "huStatusTimestamp",
                "thStatusTimestamp"
            ]
        }
    },
    "notification": {
        "attrs": [
            "exStatus",
            "huStatus",
            "thStatus"
        ],
        "attrsFormat": "normalized",
        "http": {
            "url": "http://ec2-54-93-248-56.eu-central-1.compute.amazonaws.com:1028/accumulate"
        }
    },
    "throttling": 1
}
```

Figure 10: Sample of a Subscription

In the sample shown in Figure 10 the subscriber is interested into three attributes of all the entities of type *sacmi_cds_machine*. Upon every update, of their related timestamp, the subscriber will receive an JSON message containing three attributes declared with subscription request. The message will be posted to the HTTP POST endpoint url declared during the subscription phase.

Once the subscription has been created, the Orion Context Broker will notify to the consumer's HTTP POST endpoint every time a new value set is available and the consumer will receive the desired data and execute its own application logic.

```
initServer() {
    console.log("Initializing the WebServer")
    app.post('/subscribe', function (req, res) {
        console.log(req.body.data[0]["exStatus"]);
        res.end();
    });

    let port = 3000;
    app.listen(port);
    console.log('Listening at http://127.0.0.1:' + port)
}
```

Figure 11: Snipet of a simple Node.js application

Figure 11 shows a simple Node.js application containing the implementation of HTTP POST endpoint where the Orion Context Broker will publish the selected attributes. In this case, for example, the Status variable set and the content of the attribute will be printed as simulation of application logic.

3.4 Reasoner

Starting from Task 3.1 and its results, it has been possible to start defining some rules at the base of the KBS.

Taking into consideration SACMI/CDS use case, HOLONIX analysed all competency questions made to the end user and defined from which ones it's possible to derive some rules. To have more details about the ontology definition process and all the competency questions please refer to Deliverable D3.1.

Here below, Table 1, shows the competency questions chosen for SACMI/CDS. HOLONIX chose all the questions, through which, it was possible to obtain needs. From needs, it's possible to establish rules that could be implemented via KBS. All questions selected enable to highlight a threshold value or an event that could generate a notification.

Table 1: Competency questions of use case SACMI

#	Competency questions	Rule's creation explanation
1	What kinds of actions are required *before* the failure?	Having a complete list of all required actions, both before and after the failure, it could be possible to track all these required activities in the system. Once, one or more activities were not carried out, it could be possible, through KBS, to send a notification.
2	What kinds of actions are required *after* the failure?	
3	Which is the minimum value of the signal?	Defining the *minimum* and *maximum* value of a signal, it could be possible, once the threshold would be exceed, to send a notification through the KBS.
4	Which is the maximum value of the signal?	
5	How often signals are stored in repository?	Defining a value for which a signal has to be stored in the repository, if this value/threshold is not respected, the KBS could send a notification.
6	Which is the sampling frequency of the signal?	Defining a frequency to be respected, the KBS could send a notification once this value is not respected.

Table 2 represents an example of rules that could be implemented via KBS.

Table 2: Examples of rules

#	Competency questions	Rule's creation examples
1	What kinds of actions are required *before* the failure?	List of actions to do *before* the failure: - Pumb gearbox - Oil change The operator, during his work, didn't make the pump gearbox. KBS sends a notification.
2	What kinds of actions are required *after* the failure?	List of actions to do *after* the failure: - Check oil temperature - Replace additive The operator didn't check the oil temperature. KBS sends a notification.
3	Which is the minimum value of the signal?	Minimum value of oil temperature is foxed at: 90° The signal registered temperature equal to: 85° KBS send a notification.
4	Which is the maximum value of the signal?	Maximum value of oil temperature is fixed at: 125° The signal registered temperature equal to: 130° KBS send a notification.
5	How often signals are stored in repository?	Signals have to be stored in repository every 10 minutes. The system register no signals in this timeline. KBS send a notification.
6	Which is the sampling frequency of the signal?	Frequency is set equal to 10 minutes. The system register a frequency equal to 5 minutes.

		KBS send a notification.

4 CONCLUSION

The main goal of Deliverable D3.2 is the definition of the Knowledge base system that HOLONIX has been implementing until month M21 in Z-BRE4K project by HOLONIX.

During next months of the project, a continuous implementation of the system will be done, with an improving related to reasoner part.

With the integration of all activities (WP5) expected per month M31, a testing and a validation process of the overall solution will be executed.

5 REFERENCE

[1] Gruber, Thomas R. "A translation approach to portable ontology specifications." Knowledge acquisition 5.2 (1993): 199-220.

[2] https://jena.apache.org/getting_started/index.html

[3] https://jena.apache.org/documentation/fuseki2/

[4] https://github.com/telefonicaid/fiware-orion/blob/master/README.md

[5] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model

[6] https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html