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 D4.2
Z-BRE4K DSS towards operational optimisation

Work Package 4
WP4 – Design Of Strategies And Integration Of Intelligence

Document type	:	Report
Version	:	V0.8
Date of issue	:	30/07/2019
Dissemination level	:	PUBLIC
Lead beneficiary	:	2 - ATLANTIS ENGINEERING SA

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

Executive Summary

Abstract	The development of the DSS of the Z-BRE4K project leads towards operational optimisation in the end – users shop floors. The system is developed using the criteria that are set by risk assessment methods and standards. Also, criteria exist for the Key Performance Indicators, Failure Modes and Failure Effects. The criteria are used to develop an application based on reasoning engine, a mechanism for recommendation provisions and maintenance scheduling. The reasoning engine is based on Finite State Machines, the recommendation mechanism on MQTT protocol, while the maintenance scheduling in smart asset management algorithms. Technical involvement in the DSS for different data source leads to the creation of a communication API dedicated to the Z-BRE4K project.
Keywords	DSS, Reasoning Engine, Recommendations, Risk Assessment Methods, Strategies, Predictive Maintenance.

Revision history

Version	Author(s)	Changes	Date
V0.1	ATLANTIS	1st Draft – Table of Contents	04/03/2019
V0.2	ATLANTIS	Risk Assessment Methods and Standards	18/03/2019
V0.3	ATLANTIS	Z-BRE4K Strategies	19/03/2019
V0.4	ATLANTIS	Z-BRE4K DSS Approach	22/03/2019
V0.5	ATLANTIS	Prediction Module	23/05/10
V0.6	ATLANTIS	Message Queue	28/05/19
V0.7	ATLANTIS	Event Sources and Roslyn Scripting	05/06/19
V0.8	ATLANTIS	Reasoning Engine	06/06/19
V0.9	ATLANTIS	Conclusions and Next Steps	07/06/19
V1.0	ATLANTIS	Final Version	16/07/19

Abbreviations

Abbreviations	Meaning
API	Application Programming Interface
BFS	Breadth – First Search
CAD	Computer – Aided Design
CAE	Computer – Aided Engineering
CAM	Computer – Aided Manufacturing
CEP	Complex Event Processing
CMMS	Computerised Maintenance Management System
CPU	Central Processing Unit
DFA	Deterministic Finite Automaton
DFS	Depth – First Search
DSS	Decision Support System
ERP	Enterprise Resource Planning
FE	Failure Effect
FM	Failure Mode
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Mode, Effects and Criticality Analysis
FSM	Finite State Machine
GUI	Graphical User Interface
HACCP	Hazard Analysis and Critical Control Points
HAZOP	Hazard and Operability Study
HTTP	Hypertext Transfer Protocol
IDS	Industrial Data Space

ISO	International Organisation for Standardisation
JIT	Just – In Time
JSON	JavaScript Object Notation
KBS	Knowledge Based System
KRI	Key Risk Indicator
MCOD	Multiple Components in One Database
ML	Machine Learning
MP	Matrix Profile
MPI	Matrix Profile Index
MQTT	Message Queuing Telemetry Transport
NDFA	Non – Deterministic Finite Automaton
OEM	Original Equipment Manufacturer
PdM	Predictive Maintenance
PHA	Preliminary Hazard Analysis
PL	Parameter - Length
RPN	Risk Priority Number
SCADA	Supervisory Control and Data Acquisition
SFPS	Single Failure Points
SRT	Shortest Remaining Time
XML	Extendable Mark-up Language

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 Scope and Content of the Deliverable

1.1 Scope and Objective

The present deliverable describes the results of task T4.2 and the work done for the development of a system able to optimise the predictive and Just-In-Time (JIT) maintenance operations during production processes and the assets' conditions. The Decision Support System (DSS), developed during the duration of T4.2, allows the optimisation of the maintenance techniques while implementing the Z-BRE4K strategies for predictive and preventive maintenance. The deliverable also covers the technologies and standards used for the development of the system. Risk assessment methods, Z-BRE4K strategies and a holistic method applied in the DSS, based on the end – users use cases, are described in the deliverable and cover all phases during the development phase.

Based on the technical objectives, described in the Grant agreement, the system should be able to accommodate risk mitigation through failure mode and effects analysis, and to implement machine learning techniques based on events with predictive, preventive and diagnostic analytics capabilities. The combination of the machine learning techniques and risk mitigation analysis leads to a decision support system, which offers user recommendation for the decision process on maintenance procedures. Also, the decision support system is able to create work schedules for maintenance procedures and to re – adapt the production schedule and to comply with the needs and suggestion of the decision support system.

1.2 Content and Structure

Considering the description of the task T4.2, the deliverable D4.2 is organised as follows:

- Section 2 highlights the reasoning engine for predictive maintenance in the Z-BRE4K system. The applied risk assessment methods and standards are described as well as the standards used for Key Risk Indicators (KRIs), Failure Modes (FM) and Failure Effects (FE).
- Section 3 is dedicated to the Z-BRE4K strategy Z-REMEDIATE for repairing, replacing and reusing shop floor resources and the way how it can be evaluated in the DSS. Also, it describes the results of the implementation of the strategy on the DSS. It also describes the strategies which are responsible for the preventive and JIT maintenance in the Z-BRE4K architecture.
- Section 4 provides the whole DSS solution for Z-BRE4K. The section describes the DSS architecture in detail, containing the algorithms implemented in DSS sub – components, component diagrams of the tools and all technical elements of the DSS. Smart asset management, recommendations and rule creation are described and used in the DSS reasoning engine.
  The detailed analysis in section 4 also includes the technical details and the techniques used during development of the DSS. Computerized Maintenance Management System (CMMS) integration, event aggregation and processing, based on event retrieval and message queueing are the main characteristics of the DSS analysis sub - components. Asset state tracking in a rule – based reasoning is described, as well as the

recommendation sub – component. Finally, the scripting mechanism of the rule engine, the algorithm in which the rule engine is based and whom the recommendations are provisioned to are also analysed in section 4.

- Section 5 describes the connection of the DSS with other Z-BRE4K components and a small description of each one.
- Section 6 contains the conclusions of the deliverable concerning the DSS and its implementation on the project. The implementation results are examined, and further improvements and modification are discussed. Also, new DSS features are proposed for further development.

2 Predictive Maintenance Reasoning Engine

A predictive maintenance reasoning engine is a Z-BRE4K application able to support risk assessment with multiple methods based on risk techniques and standards. The application is also able to connect to ontologies and other applications, such as Knowledge Based Systems (KBS) and predictive analytics.

The reasoning engine connects the maintenance domain with ontologies, KBS and predictive analysis in order to train with predictive maintenance models. The models require a large amount of data to be fed in the system and criteria such as data uncertainty, lack and quality of information, involved actors and assets, as well as real – time data response. The reasoning engine will be able to exploit the knowledge that is hidden in all incoming data and systems, based on the mentioned criteria. Its responses will feed the higher-level components for scheduling purposes, resources reallocation and re – adaptation of manufacturing procedures on the shop floor.

The predictive maintenance reasoning engine is implemented based on risk assessment methods and standards for better implementation of the above criteria. Assessing risk leads to better procedures and models in the reasoning engine and create a safer environment, where all criteria are met against the defined risks. Also, the risk assessment methods comply with the reasoning engine for more educated rules and feedback the other DSS components: Recommendations provision and Maintenance Scheduling.

2.1 Risk Assessment Methods and Standards

Risk assessment describes the process of identification of risk factors and hazards that may cause harm in the system called hazard identification, as well as the analysis and evaluation of the risks associated with a specific hazard and finally the determination of the most appropriate actions to eliminate it or control the risks.

The thorough look of a certain problem and possible solutions provided by a risk assessment procedure causes the next steps in all actions to be taken. The steps of the procedures are:

- Understand what happens in each situation and scenario.
- Determine the possible consequences.
- Determine how possible are the consequences to occur.
- Exploit the most suitable way to solve the risk and implement criteria in the DSS reasoning engine to allow the solution.
- Implement the reasoning engine to reduce the number of possible risks over time.

Risk assessment is valuable during the design and implementation of a reasoning engine. Knowing the possible risk scenarios, the criteria of the rule engine can be transformed to accommodate the risk solution in the scenario. Then, the criteria in the rule engine can determine the consequences of the risks and how possible they are in each scenario. Finally, the reasoning engine implements some of the criteria to solve the risks for the best possible solution. After a time period, the reasoning engine is able to re – adapt the implemented criteria, based on the real system input to develop a behaviour which reduces the causes of the risks and helps to stabilise the system in the least risky state.

The system is implemented based on recognised risk assessment methods and standards. Most of the methods are distinct for their significance in manufacturing environments and their competence to recognising patterns from various data used in manufacturing. The most common ISO standards and a method based on analysis of Key Performance Indicators (KRI), Failure Modes (FM) and Failure Effects (FE) are given below.

2.1.1 IEC31010:2009

ISO/IEC 31010[9] is a standard concerning risk management codified by The International Organization for Standardization and The International Electrotechnical Commission (IEC). The full name of the standard is ISO. IEC 31010:2009 – Risk management – Risk assessment techniques. The ISO 31010 standard supports the ISO 31000 standard. It supplies information as to the selection and application of risk assessment techniques. Risk assessment is part of the core elements of risk management defined in ISO 31000, which are:

- Communication and consultation.
- Establishing the context.
- Risk assessment (risk identification, risk analysis and evaluation).
- Risk treatment.
- Monitoring and review.

Risk can be assessed at any level of the analysis of the reasoning engine and the implementation of the DSS. There are 31 risk assessment techniques listed in Annex B of ISO/IEC 31010, the main ones are: brainstorming, Delphi method, Preliminary Hazard Analysis (PHA), Hazard and Operability Study (HAZOP), Hazard Analysis and Critical Control Points (HACCP) and Failure mode and effects analysis (FMEA) or Failure Mode, Effects and Criticality Analysis (FMECA)[15].

2.1.2 IEC60812:2006[15]

Failure mode and effects analysis (FMEA)—also "failure modes", plural, in many publications—was one of the first highly structured, systematic techniques for failure analysis. It was developed by reliability engineers in the late 1950s to study problems that might arise from malfunctions of military systems. An FMEA is often the first step of a system reliability study. It involves reviewing as many components, assemblies, and subsystems as possible to identify failure modes, and their causes and effects. For each component, the failure modes and their resulting effects on the rest of the system are recorded in a specific FMEA worksheet. There are numerous variations of such worksheets. An FMEA can be a qualitative analysis but may be put on a quantitative basis when mathematical failure rate models [**¡Error! No se encuentra el origen de la referencia.**] are combined with a statistical failure mode ratio database.

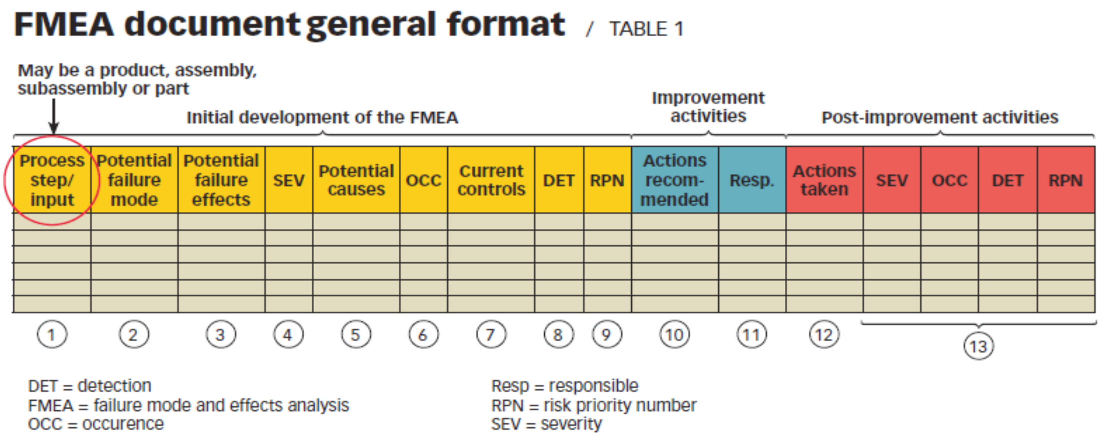
A few different types of FMEA analyses exist, such as:

- Functional.
- Design.

- Process.

Sometimes FMEA is extended to FMECA (failure mode, effects, and criticality analysis) to indicate that criticality analysis is performed too. FMEA is an inductive reasoning (forward logic) single point of failure analysis and is a core task in reliability engineering, safety engineering and quality engineering.

A successful FMEA activity helps identify potential failure modes based on experience with similar products and processes—or based on common physics of failure logic. It is widely used in development and manufacturing industries in various phases of the product life cycle. Effects analysis refers to studying the consequences of those failures on different system levels. Figure 1 shows a document with the FMEA general format.



FMEA document general format / TABLE 1

May be a product, assembly, subassembly or part

Initial development of the FMEA									Improvement activities		Post-improvement activities				
Process step/ input	Potential failure mode	Potential failure effects	SEV	Potential causes	OCC	Current controls	DET	RPN	Actions recom-mended	Resp.	Actions taken	SEV	OCC	DET	RPN
1	2	3	4	5	6	7	8	9	10	11	12	13			

DET = detection
FMEA = failure mode and effects analysis
OCC = occurence

Resp = responsible
RPN = risk priority number
SEV = severity

*Figure 1: FMEA document general format [**¡Error! No se encuentra el origen de la referencia.**]*

The **ground rules of each FMEA** include a set of project selected procedures; the assumptions on which the analysis is based; the hardware that has been included and excluded from the analysis and the rationale for the exclusions. The ground rules also describe the indenture level of the analysis (i.e. the level in the hierarchy of the part to the sub-system, sub-system to the system, etc.), the basic hardware status, and the criteria for system and mission success. Every effort should be made to define all ground rules before the FMEA begins; however, the ground rules may be expanded and clarified as the analysis proceeds. A typical set of ground rules (assumptions) follows:

1. Only one failure mode exists at a time.
2. All inputs and software commands to the item being analysed are present and at nominal values.
3. All consumables are present in sufficient quantities.
4. Nominal power is available.

Major **benefits** derived from a properly implemented FMECA effort are as follows:

1. It provides a documented method for selecting a design with a high probability of successful operation and safety.

2. A documented uniform method of assessing potential failure mechanisms, failure modes and their impact on system operation, resulting in a list of failure modes ranked according to the seriousness of their system impact and likelihood of occurrence.
3. Early identification of single failure points (SFPS) and system interface problems, which may be critical to mission success and/or safety. They also provide a method of verifying that switching between redundant elements is not jeopardized by postulated single failures.
4. An effective method for evaluating the effect of proposed changes to the design and/or operational procedures on mission success and safety.
5. A basis for in-flight troubleshooting procedures and for locating performance monitoring and fault-detection devices.

From the above list, early identifications of SFPS, input to the troubleshooting procedure and locating of performance monitoring / fault detection devices are probably the most important benefits of the FMECA. In addition, the FMECA procedures are straightforward and allow orderly evaluation of the design.

The uses of FMEA are: development of system requirements that minimise the likelihood of failures, development of design and test systems to ensure that the failures have been eliminated or the risk is reduced to an acceptable level, development and evaluation of diagnostic systems and to help with design choices (trade – off analysis). For more details, refer to [15].

2.1.3 ISO 9001:2015

The layout of the standard is similar to the previous ISO 9001:2008 [33] standard in that it follows the Plan, Do, Check, Act cycle in a process-based approach but is now further encouraging this to have risk-based thinking, as it is shown in Figure 2.

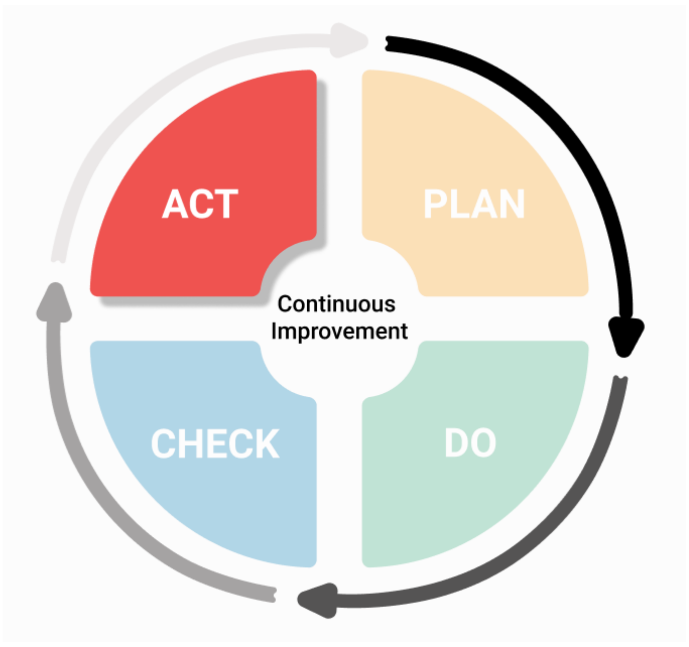


Figure 2: Plan - Do - Check - Act cycle [30]

There are ten quality objectives that define the standard's layout. They are:

- Scope.
- Normative references.
- Terms and definitions.
- Context of the organization.
- Leadership.
- Planning.
- Support.
- Operation.
- Performance evaluation.
- Continual Improvement.

The purpose of the quality objectives is to determine the conformity of the requirements (customers and organizations), facilitate effective deployment and improve the quality management system. A system can be tested against the standard and certified when requirements from all the above objectives are implemented.

During the development of the DSS reasoning engine, definition of requirements that comply with the above objectives should be documented and implemented by developers. The ISO standard facilitates stable development, implementation, testing, acceptance and maintenance of the reasoning engine and the rules based on it.

Also, based on the standard, all the development phases of the reasoning engine should be achieved with adequately defined requirements. The product is review in terms of results and designed based on requirements. Variations of scenarios can also be tested during development for different purposes. The results should be validated against the criteria and appropriate actions to resolve the problems that occurred. Finally, all actions should be documented for future references.

2.1.4 ISO 55001:2014

ISO 55001:2014 [8] is an international standard covering management of assets of any kind. Before it, a Publicly Available Specification (PAS 55) was published by the British Standards Institution in 2004 for physical assets. The ISO 55001 series of Asset Management standards was launched in January 2014. It is based on the four fundamentals of asset management: value, alignment, leadership and assurance. It also describes how to apply requirements in the key domains of asset management.

2.1.5 Key Risk Indicators (KRI), Failure Modes (FM) and Failure Effects (FE)

Key Risk Indicators are useful tools used by managers to define the level of risk on the shop floor procedures. Sometimes they indicate the risk taken by certain activities, the level of uncertainty and how assets and resources can be stretch during the manufacturing processes.

Key risk indicators are modelled based on financial, business and operational actions. They usually are evaluated by managers and their indications lead to decision making on different levels of the manufacturing process. Risks are defined and measured against the most important indicators.

The DSS reasoning engine should be able to incorporate the KRIs in its rule creation process in order to respond to difficult and varied problems on a manufacturing process. One of the main points in using KRIs in the reasoning engine, is to determine the best maintenance practices and prevent major failures during the operation. Also, KRIs can provide suitable numbers for JIT maintenance. Figure 3 shows KRIs in a commercial DSS and how they represent some of the most critical procedures.

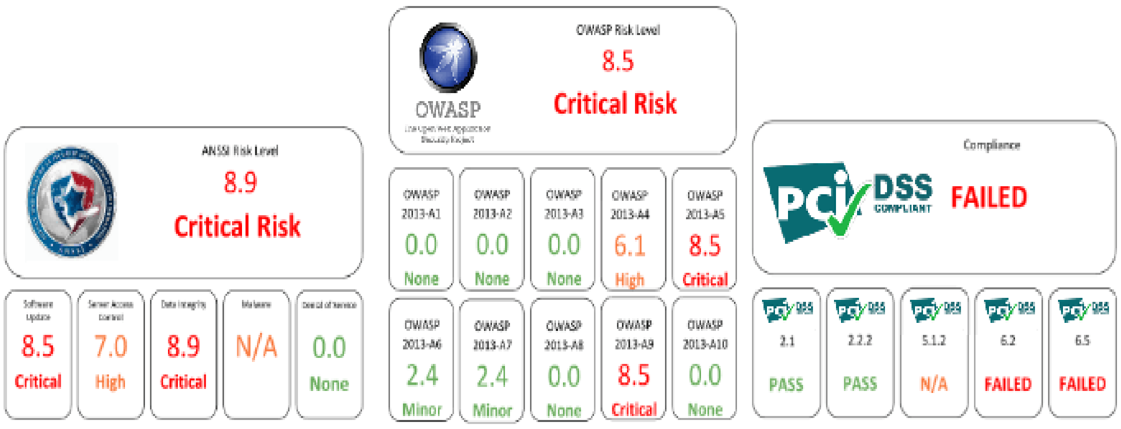


Figure 3: Example of KRIs in a DSS [11]

Failure mode (FM) is the cause of a failure in a system. When there are many possible failures in a system, there also are many different failure modes. Though, some failure modes cause the system to fail more often than others. Their distribution varies based on the severity and criticality of the failure mode.

Failure effects (FE) are the consequences of a failure in a system. They disturb the operation, the status, the functionality of the system. FE have many severity levels and can cause from minor problems in a production line to complete failure and stop of manufacturing operations. Some serious effects can be, injury to the user, inoperability of the product or process, deterioration in product quality, nonadherence to the specifications, failure on other systems in immediate contact with the system that failed must be considered. Figure 4 indicates the failure modes and effects in a cycle process.

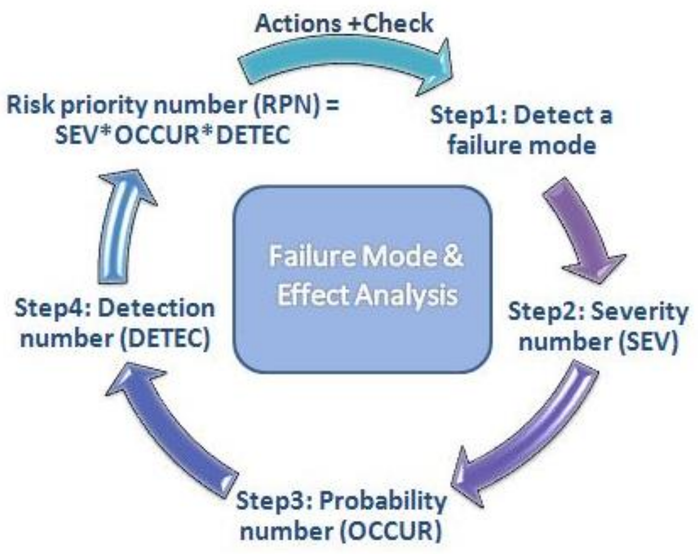

Actions +Check
Step1: Detect a failure mode
Step2: Severity number (SEV)
Step3: Probability number (OCCUR)
Step4: Detection number (DETEC)
Risk priority number (RPN) = SEV*OCCUR*DETEC
Failure Mode & Effect Analysis

Figure 4: Failure Modes and Effects [12]

KRIs, FMs and FEs should be incorporated in the DSS and the reasoning engine which trie to solve problems and create suggestions for maintenance and manufacturing operations. They can be studied and analysed in the system and the defined criteria can be trained against in all indicators to create new, more probable rules and suggestions. Input data from sensor networks is used as a test dataset along with the indicators for improved suggestions from the reasoning engine.

3 Z-BRE4K Strategies in the Decision Support System

The Z-BRE4K DSS includes several strategies for all its purposes. The strategies are defined by the use cases and the scenarios provided by the end – users. The strategies also help to organise the system's architecture and how it should behave in different situations. According to the use case scenarios, the DSS incorporates seven Z-BRE4K strategies in three well defined use cases. Figure 5 shows how the Z-BRE4K strategies are connected in the project, as well as how they are responsible for the various Z-BRE4K components.

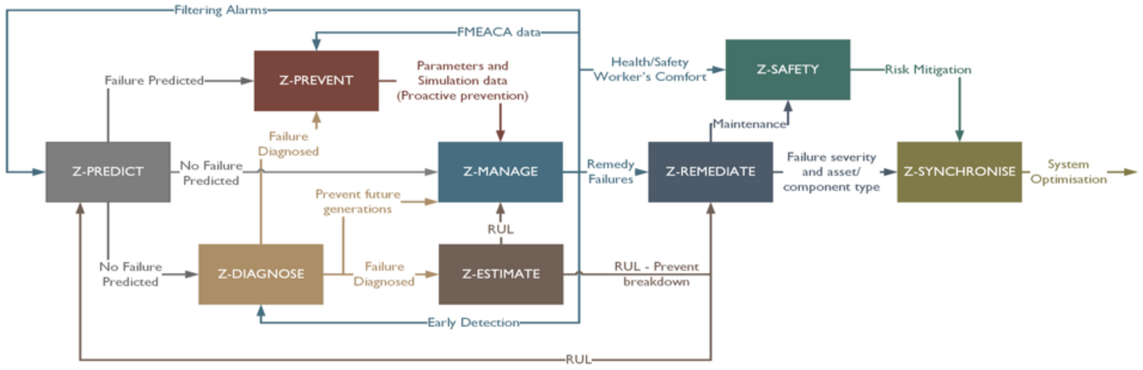


Figure 5: Z-BRE4K Strategies

3.1 Production Processes of Asset Detection and Prediction Condition

A manufacturing process is defined by a set of principles. Those principles, in general, are specifications, safety measures, work progress indicators etc. The manufacturing process is monitored by recording data to Enterprise Resource Planning (ERP) and CMMS systems.

The main key point is the IoT devices (e.g. sensors) network that collects various signals. At each shop floor, various assets (machines or other equipment) are monitored by various sensors. The data that are usually collected, are vibration, temperature, noise data.

Asset detection with proximity-based sensors, asset tracking devices and monitors is one of the latest implementations of a shop floor sensor network. Assets are tracked because they are cost – effective, valuable to the production line and asset shortages create a blockage in the production.

Predictive maintenance (PdM) is one of the critical features of Industry 4.0. The prediction process predicts assets failures (non-conformities and broken assets), shop floor working conditions. Prerequisite for a successful prediction process is the continuous assets monitoring and the signal collection they produce.

Table 1 describes the Z-BRE4K strategies used in production processes of asset detection and prediction conditions.

Table 1:Z-BRE4K Strategies for Production Processes of Asset Detection and Prediction Condition

Z-BRE4K Strategy	Description
Z-Predict	The events detected from the physical layer of the system are engineered into high value data high that will stipulate new and more accurate process models. Such an unbiased systems behaviour monitoring and analysis provides the basis for enriching the existing knowledge of the system (experience) learning new patterns, raising attention towards behaviour that causes operational and functional discrepancies (e.g. alarms) and the general trends in the shop-floor. The more the data pool is being increased the more precise (repeatability) and accurate the predictions will be. The estimations for the future states involve the whole production line – the network of machines and components. The system will predict with high confidence the expected performance of components and their maintenance needs, allowing better production planning and decision making on their remaining useful life (RUL). The ability of Z-BRE4K to optimise the manufacturing processes according to the RUL, production needs, and the maintenance operations is the key innovation to fulfil the industrial requirements.
Z-Diagnose	This strategy is invoked when a current or an emerging failure is detected considering the condition at all three levels – Machine, Product, Shop-floor. In such a scenario, an alarm is being triggered to flag the events that resulted in a failure. By mapping the true reasons, the system will be able to avoid generating the failure or its emergence by weighting the system model. The strategy also involves more actions and processes to deal both with the generation of the diagnosed failure, and its severity increase to the next iterations as well as its impact on the production line. Depending on the criticality of the generated failure, the system will either adapt its parameters to prolong the useful life until the next maintenance, or plan to the production for maintenance. The final decision on the actions is based on the Z-MANAGE strategy.
Z-Estimate	This strategy will combine the information from the Z-DIAGNOSE and Z-PREDICT in order to estimate the remaining useful life of the assets. The estimated values will also be combined with the information from the maintenance operations (physical examination from operators) as well as from the specifications provided by the manufacturer. The latter will be used as the starting point for the estimation process, which after each iteration the deviation of the real-model from the physical model will be reduced having an accurate virtual-model wrapped around the actual state of each machine and its components. The trends for the fatigue and ware rates will provide a confident RUL estimation

3.2 Shop – floor resources

Shop floor resources are considered a milestone for the manufacturing procedure. Assets, other equipment, but even the personnel itself might be identified as resources. Resources allocation is described by the Z-BRE4K strategies given in Table 2 below:

Table 2:Z-BRE4K Strategies for Shop - floor Resources

Z-BRE4K Strategy	Description
Z-Remediate	This strategy involves the decision making in the event of a failure, which after its analysis it will be classified and categorised based on its effect, criticality, and type. Based on the component/assets types (repairable/non-repairable) and their remaining useful life the system will decide for the following; (i) Replace, (ii) reconfigure and/or re-use, (iii) retire, and (iv) recycle. This strategy triggers the Z-SYNCHRONISE and Z-SAFETY strategies from which the maintenance actions will be planned and organized.

3.3 Predictive and JIT Maintenance

Maintenance is a critical operation, from the business and technical point of views. It improves the assets (and/or other equipment) Remaining Useful Life (RUL), reduces the risk of critical failures in the production line, prevents severe damages or unscheduled interruptions on the production.

The typical maintenance operation is the preventive maintenance, which is executed in a strict periodical schedule each year. The Preventive maintenance process, is the least complex one, and is one of the oldest maintenance processes.

Predictive and Just–in–Time (JIT) Maintenance are the most recent developments in manufacturing maintenance. Predictive maintenance is a combination of a failure prediction (when it will happen), and the DSS that exploits and analyses the prediction, and in the end generates alarms and alerts. The addressed people are notified with those alarms and alerts, in order to prevent the failure, if possible.

JIT maintenance is the operation that is executed during a specific time range. The time range is provided by the DSS as a suggestion, and usually is before the failure when the risk factor is quite high. Table 3 below is the description of the predictive and JIT maintenance Z-BRE4K strategies.

Table 3: Predictive and JIT maintenance Z-BRE4K Strategies

Z-BRE4K Strategy	Description
Z-Manage	The overall supervision and optimisation of the system are achieved after the execution of Z-MANAGE strategy. The failures are processed with the Decision support system (DSS) tools and are interfaced with Manufacturing Execution Systems (MES). False positives and false negatives are clustered after the Z-DETECT strategy, which results into fine filtering of these false alarms. To achieve so, the previously acquired knowledge and incidents are also processed to tune the system's performance. Additionally, the production is optimised by better scheduling (Z-SYNCHRONISE), considering the impact of each failure. The optimised scheduling and adaptability of the manufacturing improves the overall flexibility, placing a premium on the production systems, extending their operating life, while preserving increased machinery availability.
Z-Synchronise	The predecessor Z-REMEDIATE strategy will identify the type of action required for diagnosed failures which will be fused with the Z-MANAGE output. This strategy will synchronise all the remedy actions with internal and external supply-chain tiers, as well as with production planning and logistics. It will be responsible to shift the production from one machine to another due to failure or deteriorated condition/performance, acting as the "end-effector" of the Z-BRE4K system.
Z-Safety	Z-BRE4K aims to increase the health & Safety of the factories by employing this strategy. It will be invoked when a maintenance operation will take place to act as an operational "handbrake". Since most of the accidents occur during maintenance actions, the Z-SAFETY will prevent any activation to the machine that is under investigation or repair. The "Safety-Mode" will lift any unauthorised control from the personnel for the whole duration of the maintenance. Apart from reducing the accidents Z-SAFETY will also consider the comfort of the human personnel on the shop floor, e.g. extreme heat or noise may be tolerable for the machines but not for humans. Therefore, the health & safety procedures will be also considered towards the operation feedback of the whole production line. To this end, the design of Strategy will address "safety & health at workplace" Directive 89/391 EEC – OSH and Directive 2009/104/EC "Use of work-equipment".

4 DSS APPROACH FOR Z-BRE4K

The DSS of the Z-BRE4K project implements risk assessment methods for computing KRIs, FMs and FEs in order to produce efficient decision making for improved predictive and JIT maintenance. The system is based on the Z-BRE4K strategies described in the previous section.

There are many types of decision support systems. The DSS developed for Z-BRE4K is a combination between data driven and model driven DSS. The system is developed as a combination between these two types of DSS in order to exploit both the acquired data from the sensor network and the criteria based on the risk assessment methods and the Z-BRE4K strategies.

4.1 System Architecture

The system is modular developed by smaller and independent sub – components. Each of the sub – components are specifically designed to execute a function without interrupting other operations of other sub – components.

The component diagram of the system architecture is given below in Figure 6.

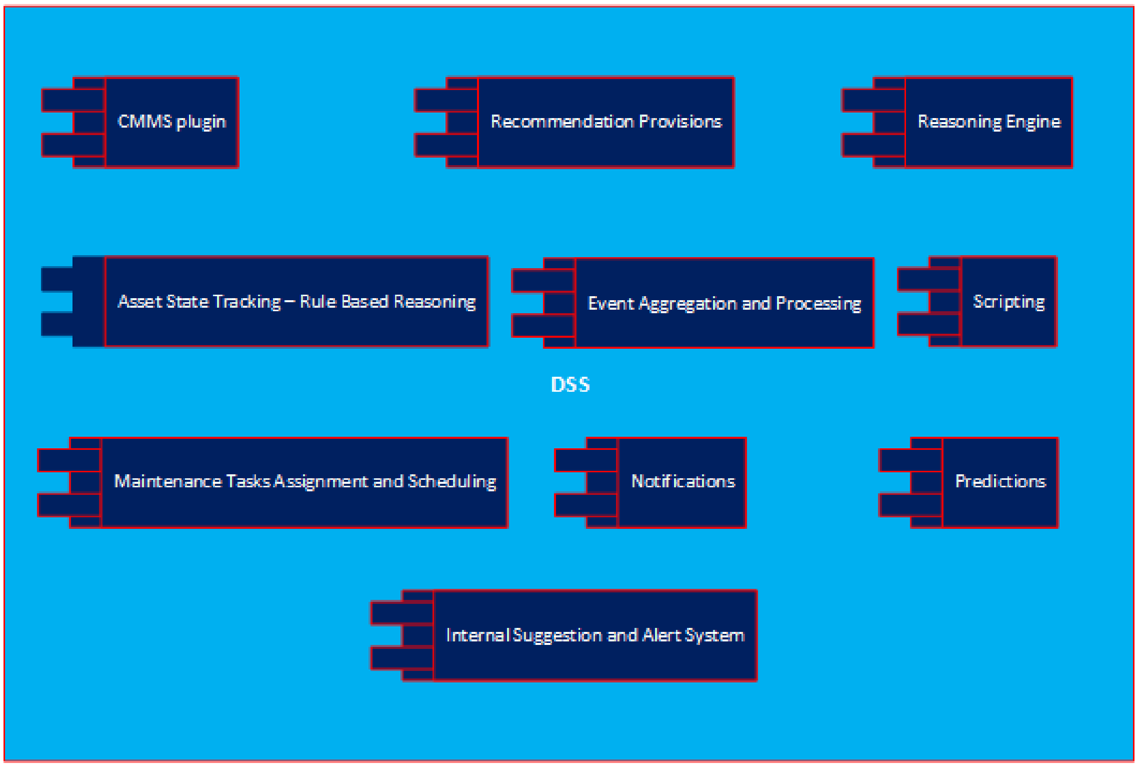

CMMS plugin
Recommendation Provisions
Reasoning Engine
Asset State Tracking – Rule Based Reasoning
Event Aggregation and Processing
Scripting
DSS
Maintenance Tasks Assignment and Scheduling
Notifications
Predictions
Internal Suggestion and Alert System

Figure 6: DSS Component Diagram

The DSS sub – components are

- the CMMS plugin,
- the Recommendation Provision,
- the Reasoning Engine,

- the Asset State Tracking – Rule Based Reasoning,
- the Event Aggregation and Processing,
- the Scripting,
- the Maintenance Tasks Assignment and Scheduling,
- the Notification,
- the Prediction,
- the Internal Suggestion and Alert System.

4.2 CMMS Integration

The CMMS Integration sub-component, along with the Event Processing sub-component, form the lowest tier of the system if seen as a layered architecture. It can be customized via plugins. Its role is to provide a way for the operators responsible for system initialization, to access, define, and organize the (meta)data pertaining to the shop floor, and specifically if / when available, to integrate the system with a CMMS.

The functionality in this sub – component offers benefits such as synchronization tracking by polling endpoints containing shop floor data, ease of use and avoidance of mis-configuration (for example the user does not need to copy – paste "asset codes" when defining rules/state machines) and making advantage of any externally accessible commands the CMMS or other external component can offer (for example, task creation in the CMMS). Figure 7 shows who a CMMS system communicates with the CMMS plugin of the DSS.

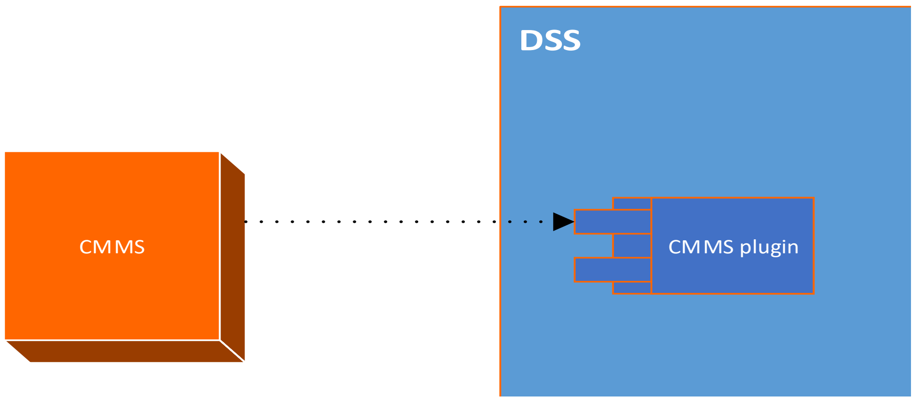


Figure 7: CMMS plugin and communication with CMMS in the DSS

In addition, agent data can be used for recommendations provision. A plugin system allows this sub-component to be agnostic to what particular CMMS or similar component is being used. The sub-component makes the assumption that the shop floor can be modelled using a simplified schema of Asset, Measurement Location, Event Type and Agent entities and expects to be able to load and synchronize these entities by a plugin. Figure 8 is the ontology tree for the CMMS plugin of the DSS system.

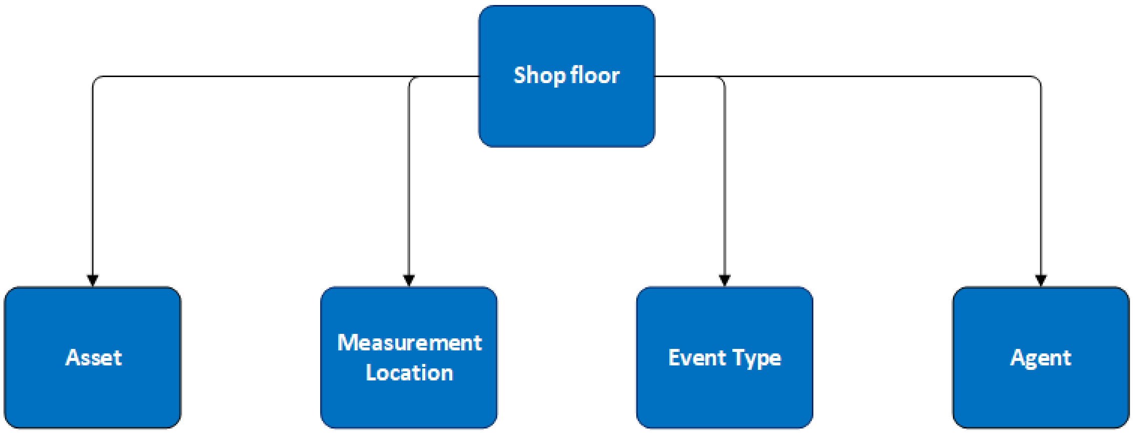


Figure 8: Ontology tree for the CMMS plugin

4.3 Event Aggregation and Processing

This sub – component provides the system's capabilities to access data and event sources, optionally performing Complex Event Processing (CEP) operations. The sub – component is created in the, but aims to be a separate .NET library in order to be easily accessible and reused in other projects, apart from the DSS. It supplies data to sub-components of a higher tier as well as performs event persistence and logging if needed. In addition, this sub - component harmonises the data formats coming from various sources into a common event format. The common event format is used by the rest of the system to create a data format indifferent and agnostic of the details of the data source format. The sub-component is architecturally designed to consist of three tiers.

In the lowest level a set of utility and wrapper methods are provided which aim to:

1. Select amongst several common methods for data retrieval (for example HTTP polling, MQTT, AMQT, an ad hoc internal queue), amongst the provided implementations
2. Assisted in the initialization of the retrieval process for this particular method, by an API providing sensible defaults (for example hiding any programmatic details of setting up an MQTT client)
3. Automatically convert this retrieval process into the System. Reactive observable which is a Microsoft library for creating pipelines of Observables. Data sourced provide data in event format and CEP is performed on the data. It is the .NET implementation of the ReactiveX asynchronous programming and observables pipeline API. It also enables the use of "Asynchronous Processing" techniques, providing a significant performance boost and simplifying multithreaded code development. Asynchronous Processing in this context refers to when operations or threads wait for a result from another operation or thread (for example data parsing waiting on completion of an HTTP call), but do not create bottlenecks and blocks, simply returning control to a dispatcher and

wait to be notified to continue. Bottlenecks and blocks waste resources and indirectly introduce deadlocking problems.
System.Reactive was chosen to simplify development, increase performance and make a robust system, by exploiting an existing implementation of an API and offering the above async processing functionality, as well as the "compositional" (pipelining) functionality that enables CEP operations.

In the second tier, resides a CEP system. Eventhoug, the system implementation is not yet completed, the system is able to delegate the operations available to System.Reactive, but it also provides a more specific API. The API hides details, provides sensible defaults, and integrates with the tiers higher in the layer stack of our system. For example, aggregating by moving averages or throttling of events can be defined on this level. The details of the implementation are hidden by System. Reactive from higher layers.

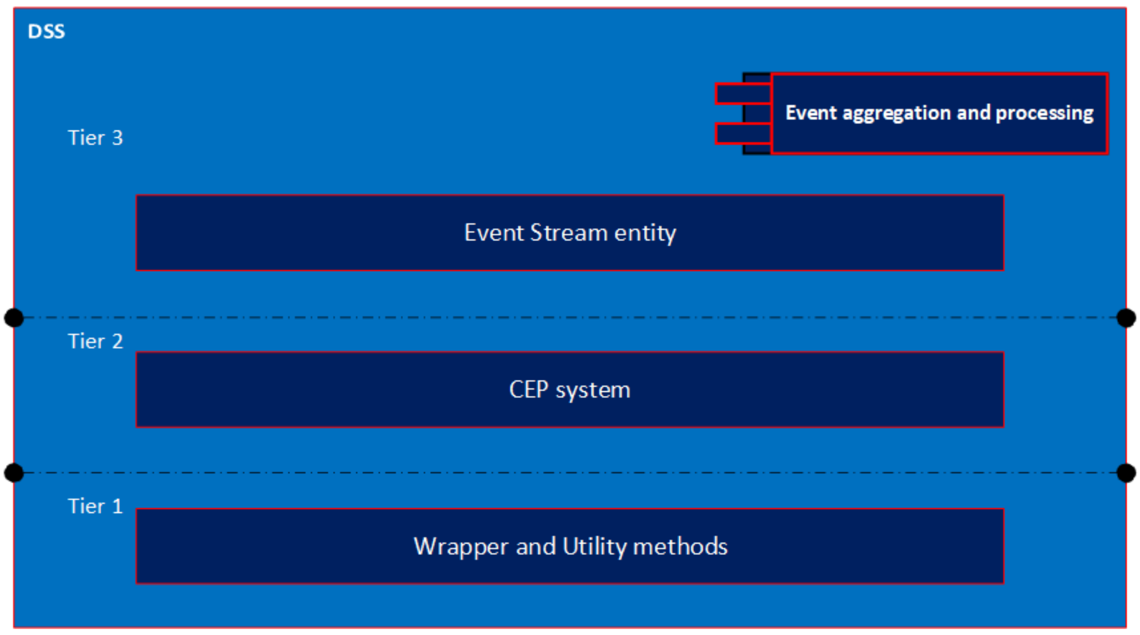


Figure 9: Event aggregation and processing sub - component tiers

In the highest third layer, an "Event Stream" entity is defined, which delegates to the tiers below. An event stream can be thought of as the combination of data retrieval and parsing, along with default logging mechanisms, persistence, and the ability to expose a simple observable that notifies of data updates, but also automates all the previously mentioned functionality with only a simple initial configuration needed by the sub-component user. The Event Stream is configurable by the DSS-plugins to extract shop floor events and harmonize their data format into the common internal format used by the rest of the system.

Figure 9 shows the three – tier architecture of the Event Aggregation and Processing sub – component. Essentially the DSS will treat its plugin – based event sources, as the definition of Event Streams. These, as described above, can take advantage of asynchronous processing, are easily configurable to incorporate common logging, CEP and persistence mechanisms, and to produce events in a common event format.

The event streams are also planned to be accessible at a higher level, such as an asset tracking State Machine directly placing conditions on a named DSS event stream.

4.3.1 Event Retrieval and Processing

An additional system is planned for use, which logically can also be thought of as a part of the Event Retrieval and Event Processing subsystem, but is developed on its own for reasons that are described below.

So far, the described sub-component can be considered to provide the following functionality

- Convert data received as events into a common format.
- Collaborate with shop floor specific plugins to abstract the details of receiving events from each shop floor.
- Enable the use of asynchronous techniques.
- Provide automatic persistence and logging mechanisms.
- Make use of various CEP techniques and set up event pipelines.

However, when thought of as an "API" it is evident that it is targeted to the system programmer and not an and user. In addition, one might expect for the user to people to participate in the CEP pipeline by a "scripting" method.

Therefore, an additional layer related to data and event retrieval exists, which presupposes the base layer, and thus can take advantage of functionality such as receiving events in a common format, and having access to a "list of shop floor Event Sources". The arguments allowed by this functionality are:

- Configuration of the event transformations and CEP pipeline by user scripts
- Automatic dependency resolution
- Domain Specific Language like enhancements towards simplifying scripts specifically for this context
- "Augmented Compilation", where the build errors of the C# compiler when compiling user scripts, are augmented with errors regarding the DSL syntax or parameter set up etc.

The sub component is named the Parameters sub component, which sort of signifies that it is supposed to offer Event Processing style operations but in a higher and more "mathematical" level.

The Parameters sub-component is directly usable from the User Interface of the application. It allows a user to set up a Parameter by choosing a set of Event Streams, configuring it with a *set Event Processing parameter*, such as extra data filters on the common DSS event format, or user assignable TTL settings.

Finally, it adds the option to use a C# script (with specific DSL style syntax that simplifies the code) to convert data in events in an ad hoc way, and additionally to reuse other parameters when defining new ones.

That is the goal is that for an installation that for example has been configured by a plugin to provide three basic even sources X, W, U

The user is able to quickly write the equivalent form as:

$$Y = Math.Sin(5 * X)$$

$$Z = Math.Sqrt(Y + W * U) + 1$$

The system automatically resolves the human form regular expressions and creates the necessary dependency pipeline to compute the above without the effort of writing possible difficult code (such as synchronization code etc.).

Crucial to this is the use of topological sorting for dependency extraction from parameter references in user scripts.

The subsystem is not only used for simplifying end user interaction, but as the gateway for any machine learning sub – components interacting with our rule reasoning and asset state machine tracking functionality. It provides a format for accessing relevant data that is much simpler and closer to the mathematical model used by inference techniques, in comparison to directly accessing the DSS internal events.

4.3.2 Message Queue

We are also implementing our own message queue for the common case scenario where an external component wants to directly provide events to the DSS, without making use of an MQTT broker. The queue is integrated into the asynchronous event parsing methods described above and makes available simplified message queuing techniques while providing a simple rest call like API.

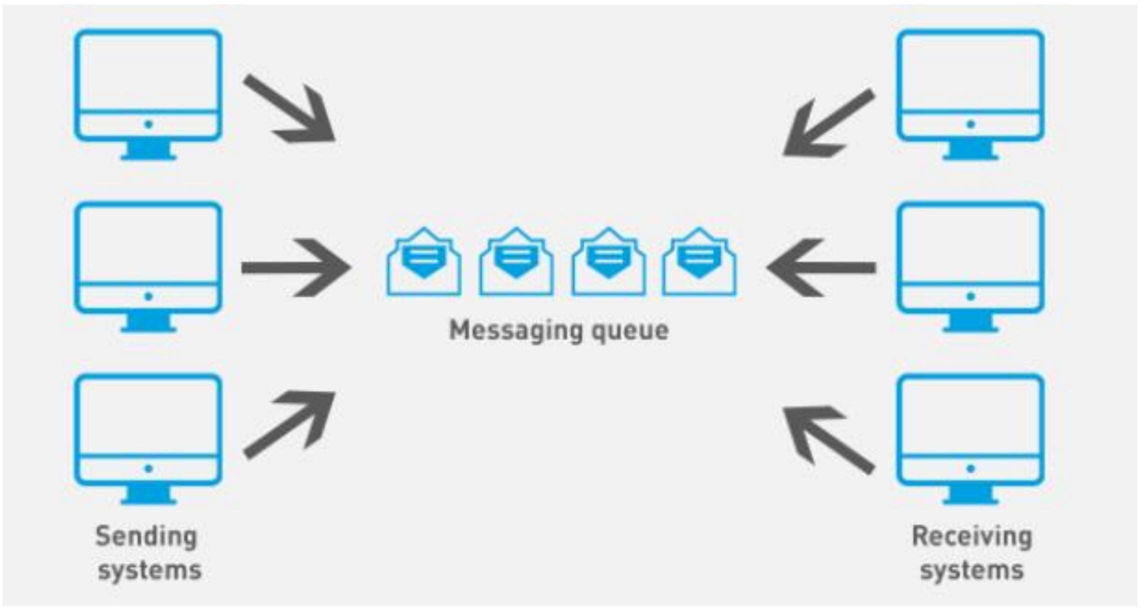


Figure 10: Message queue system [27]

Figure 10 shows the functionality of a message queuing system. Message queuing allows applications to communicate by sending messages to each other. The message queue provides temporary message storage when the destination program is busy or not connected.

A queue is a line of things waiting [27] to be handled - in sequential order starting at the beginning of the line. A message queue is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed.

A message is the data transported between the sender and the receiver application; it's essentially a byte array with some headers on top. An example of a message could be something that tells one system to start processing a task, it could contain information about a finished task or just be a plain message.

The basic architecture of a message queue is simple, there are client applications called producers that create messages and deliver them to the message queue. Another application, called consumer, connect to the queue and get the messages to be processed. Messages placed onto the queue are stored until the consumer retrieves them.

A message queue provides an asynchronous communications protocol, a system that puts a message onto a message queue does not require an immediate response to continuing processing. Email is probably the best example of asynchronous messaging. When an email is sent, the sender continues processing other things without an immediate response from the receiver. This way of handling messages decouples the producer from the consumer. The producer and the consumer of the message do not need to interact with the message queue at the same time.

Decoupling is used to describe how much one piece of a system relies on another piece of the system. Decoupling is the process of separating them so that their functionality will be more self- contained.

A decoupled system is achieved when two or more systems are able to communicate without being connected. The systems can remain completely autonomous and unaware of each other. Decoupling is often a sign of a computer system that is well structured. It is usually easier to maintain, extend and debug.

If one process in a decoupled system fails to process messages from the queue, other messages can still be added to the queue and be processed when the system has recovered. You can also use a message queue to delay processing; A producer post messages to a queue. At the appointed time, the receivers are started up and process the messages in the queue. A queued message can be stored-and-forwarded, and the message is redelivered until the message is processed.

Instead of building one large application, it is beneficial to decouple different parts of your application and only communicate between them asynchronously with messages. That way different parts of your application can evolve independently, be written in different languages and/or maintained by separated developer teams.

A message queue will keep the processes in your application separated and independent of each other. The first process will never need to invoke another process, or post notifications to another process, or follow the process flow of the other processes. It can just put the message on the queue and then continue processing. The other processes can also handle their work independently. They can take the messages from the queue when they are able to process them. This way of handling messages creates a system that is easy to maintain and easy to scale.

4.4 Asset State Tracking – Rule Based Reasoning

The core of the DSS functionality lies in its ability to make real time decisions based on the aggregated events and data describing the shop floor's operation state.

These decisions might derive from criteria such as those in section 2 containing risk assessment methods, or in a general sense contain reasoning that attempts to suggest an action to be performed on the shop floor based on its current or predicted state.

On a high level some of desired attributes of such a system are:

1. The ability to allow managers, operators and experts to accurately and easily express and integrate criteria derived from their domain knowledge or from a set of standards.
2. The ability to easily view the current "shop floor" state in real time without the data that is being used itself, ending up hidden inside a rule-based processing module that may only output specific recommendations.
3. Flexibility when it comes to the system integrating to various installations or external modules providing supplemental functionality, such as the prediction module.
4. Gathering and organizing historical data that can be provided to statistical and machine learning techniques, as well as for report generation
5. An interface to allow machine learning systems to directly manipulate the current "rule set" after a machine learning method has computed more optimal conditions etc.

Several alternative methods to express user supplied rule-based processing on a set of data were tested during development. Certain powerful methods such as variable based on *if-then* rule sets used in some expert systems were for the time abandoned in favour of a State Machine Based model.

Considering (1) and (2) above, if the rule set was supplied in a set of *if-else* statements it could require too much effort from the end-user. Designing a rule that is only activated by a certain system states, would increase the predicate size and require introducing and controlling state variables by the user. Such implementation would potentially make the system error prone in adding or configuring rules, because it requires complete understanding of the moment of firing by the user and "expert system designer" skills which the user group should not possess.

The expectation of a state machine-based model is more natural and more comprehensible for users. Rules are already provided by transition moving from one state to the next, based on the context described by parameters. In addition, this complies with requirement (2) since we get a model for the current system setting which can be visually examined in a relatively easy procedure. More about the theoretical background of the state machines is mentioned in section 4.5.1 below.

There are several drawbacks using State Machines, such as the fact that certain transitions, and therefore *if-then* rules, have to be repeated multiple times when they are to be fired by multiple states, increasing the model's complexity. However, complexity mitigation is designed by providing additional functionality to the transitions. Transitions could be defined to have multiple sources. A transition "prototype" is created and transformed into the final state machine, as well as a guide for users to "walk through" the application to design multiple smaller independent state machines. The more agile, smaller state machines are more efficient versus a more complex or "hierarchical" model. They also help with the rule context because they depend on the separation of concerns in the system.

Thus, the model used is a state machine-based model where the states are declared by the user and are expected to represent a condensed description of the measurements defining the current shop floor state. And the transitions are the equivalent to the *"IF [CONDITION] THEN [ACTION]"* rule set item.

The system allows users to design transitions, extract visualizations of the model, and attach actions via the Recommendation Provision sub – component. It is also accessible by software components such as machine learning subsystems. The system is able to provide to them complete historical data such as the transition log and parameter values, as it is stated in (4) and (5) above.

The transitions level rules take as input the current output of the Parameter Sub-Component. Several common predicates (such a value threshold) are directly provided in the UI as user filters. In addition, the scripting sub – component, described in section 4.6, allows the creation of complex transition firing predicates. Machine Learning (ML) techniques where the model can be applied within the context of a single transition, such as a Support Vector Machine that creates a classification based on recommendation feedback. The set of computed parameters at the time the transition is activated, are integrated to the available option for configuring a transition's triggering condition.

Requirement (3) above is implemented in conjunction with the CMMS integration sub-component.

4.5 Reasoning Engine

The reasoning engine is the core functionality of the Z-BRE4K DSS. It contains all the criteria described in section 2, concerning the risk assessment methods and standards. The engine executes a set of rules, based on the criteria to create suggestion concerning the operations of a shop floor.

The algorithm and logic behind the DSS Reasoning Engine is based on the Finite State Machines (FSM) theory. As the theory indicated, there is an alphabet, words, conditions and transitions. FSMs were chosen for the development of the DSS Reasoning Engine because they provide a valuable and reliable way to compute the transitions and the movement between states. Even if the reasoning engine contains many rules that should be computed simultaneously, the FSMs do not expand in the system and they do not strong computational power. An FSM for the rule engine is shown in Figure 11.

The reasoning engine is based on the parameters, transitions and states to describe each rule. The parameter is the value against which both states and transitions are measured. The parameters can be numeric, alphanumeric, regular expressions etc. They usually are set at the beginning of the systems operation and they are commonly linked with the operations of the system.

The states are described to indicate the current state of the system. Each state is defined by arithmetic values, which are set by the users (maintenance and production managers) through a web UI. Managers take advantage of their experience, by setting the limits for each state. The DSS exploits their knowledge to create a set of rules based on those states [4].

The transitions are triggered when the state changes. They constantly monitor the incoming data. The consumed data is compared with the parameter values and if the transition conditions are true the transition is triggered.

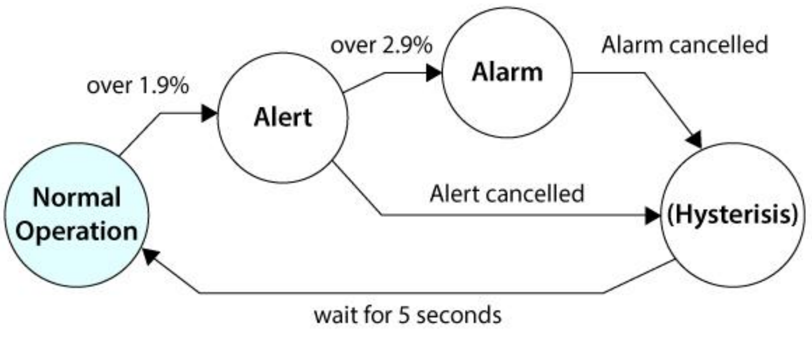


Figure 11: FSM for a rule in the DSS Reasoning Engine [26]

4.5.1 Finite State Machines

A deterministic finite automaton (DFA) D is a tuple $(Q, \Sigma, \delta, q_0, F)$ where [2]:

- Q :is a finite set of states
- Σ :is the input alphabet (any non-empty set of symbols),
- $\delta: Q \times \Sigma \longrightarrow Q$: is the transition function
- q_0: is the initial state and
- $F \subseteq Q$: is the set of final states.

When the transition function is total, the automaton D is said to be complete. Any finite sequence of alphabet symbols $a \in \Sigma$ is a word. Let Σ^* denote the set of all words over the alphabet Σ and ϵ denote the empty word. We define the ***extended transition function*** $\hat{\delta}: Q \times \Sigma^* \longrightarrow Q$in the following way:$\hat{\delta}(q, \epsilon) = q; \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$. A state $q \in Q$of a DFA $D =$

$(Q, \Sigma, \delta, q_0, F)$ is called ***accessible*** if $\hat{\delta}(q_0, w) = q$ for some $w \in \Sigma^*$ e. If all states in Q are accessible, a complete DFA D is called (***complete) initially-connected (ICDFA).*** The language accepted by D, $L(D)$, is the set of all words $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w \in F)$. Two DFAs D and D' are ***equivalent*** if and only if $L(D) = L(D')$. A DFA is called ***minimal*** if there is no other equivalent DFA with fewer states. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, two states $q_1, q_2 \in Q$ are said to be equivalent, denoted $q_1 \approx q_2$, if for every $w \in \Sigma^*, \hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \in F$. Two states that are not equivalent are called distinguishable. The equivalent minimal automaton $D/\approx$ is called the ***quotient automaton***, and its states correspond to the equivalence classes of ≈. It is proved to be unique up to isomorphism.

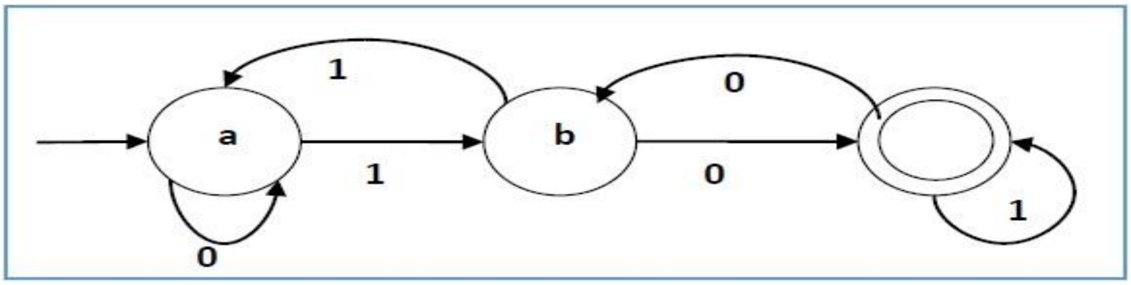


Figure 12:Finite State Machine

A DFA is represented by digraphs called state diagram [24].

- The vertices represent the states.
- The arcs labelled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Figure 12 shows an example of a DFA where:

- $Q = \{a,\ b,\ c\}$
- $\Sigma = \{0, 1\}$
- $q_0 = \{a\}$
- $F = \{c\}$, and

Transition function δ as shown by the following Table 4.

Table 4: Transition δ table

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

A ***non-deterministic finite automaton (NDFA)*** is also a tuple $(Q, \Sigma,\ \Delta,\ I,\ F)$ where I is a set of ***initial states*** and the ***transition function*** is defined as $\Delta : Q\ \times\ \Sigma\ \longrightarrow 2^Q$. Just like with DFAs, we can define the ***extended transition function*** $\hat{\Delta} :\ 2^Q \times \Sigma^* \longrightarrow 2^Q$ in the following way: $\hat{\Delta}(S, \epsilon) = S$; $\hat{\Delta}(S, xa) = \cup_{q \in \hat{\Delta}(S,x)} \delta(q, a)$. The language accepted by $N$is the set of all words $w \in \Sigma^*$ such that $\hat{\Delta}(I, w) \cap F \neq 0$. Every language accepted by some NFA can also be described by a DFA. The ***subset construction*** method takes an NFA A as input and computes a DFA D such that $L(A) = L(D)$. This process is also referred to as ***determinization*** and has a worst-case running time complexity of $O(2^{|Q|})$.

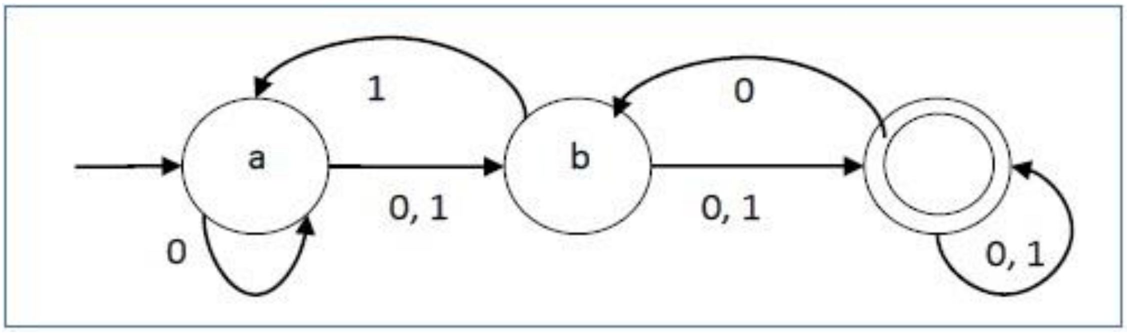


Figure 13: Non - Deterministic Finite Automaton

An NDFA is represented by digraphs called state diagram [25].

- The vertices represent the states.
- The arcs labelled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Figure 13 shows and example of an NDFA where:

- $Q = \{a,\ b,\ c\}$
- $\Sigma = \{0, 1\}$
- $q_0 = \{a\}$
- $F = \{c\}$, and

The transition function δ as shown in the Table 5 below:

Table 5: Non - deterministic Finite Automaton Transition δ

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

Among the DFAs and NDFAs there are conceptual differences. Those differences are summarised in

Table 6 below:

Table 6: Differences between DFAs and NDFAs

DFA	NDFA
The transition from a state is to a single next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called ***non-deterministic*.**
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

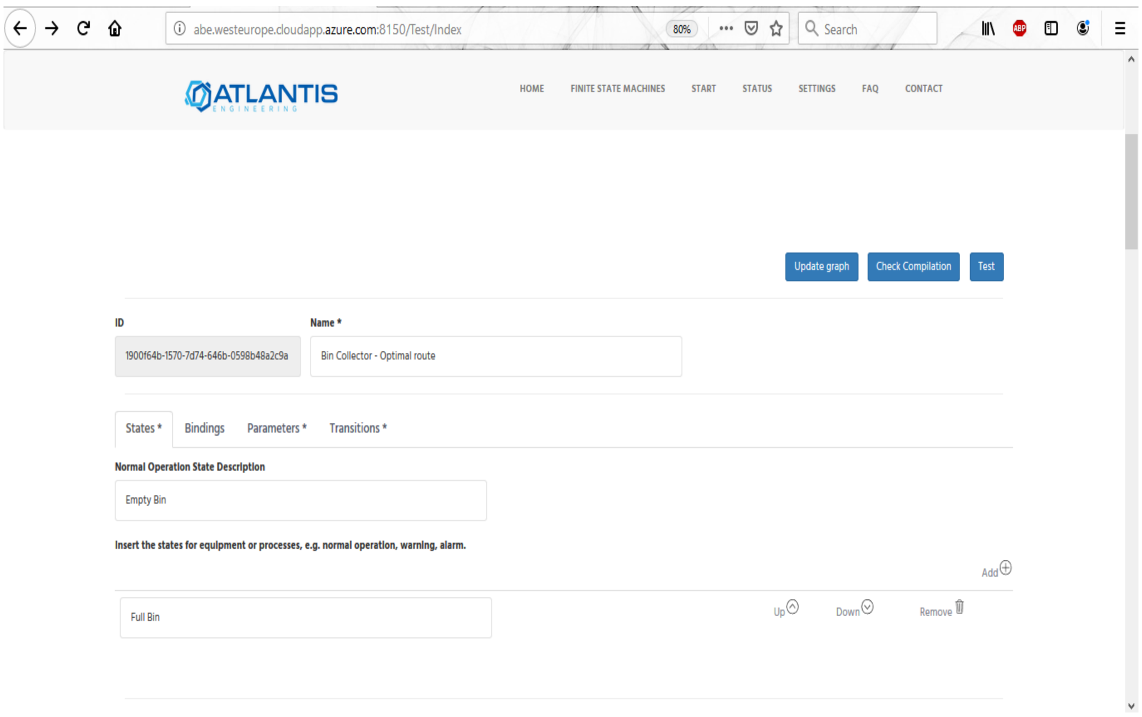
The ***transition density*** of an automaton $A = (Q, \Sigma, \Delta, I, F)$ as the ratio $\frac{t}{|Q|^2|\Sigma|}$, where t is the number of transitions in A . We also define ***deterministic density*** as the ratio of the number of transitions t to the number of transitions of a complete DFA with the same number of states and symbols, i.e.: $\frac{t}{|Q||\Sigma|}$.

The ***reversal*** of a word $w = a_0 a_1 \dots a_n$, written w^R, is $a_n \dots a_1 a_0$. The reversal of a language $L \subseteq \Sigma^*$ is $L^R = \{w^R | w \in L\}$.

A string is accepted by a DFA/NDFA if the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

- A string S is accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, if $\delta * (q_0, S) \in F$
- The language L accepted by DFA/NDFA is $\{S | S \in \Sigma^* and\ \delta * (q_0, S) \in F\}$
- A string $S^{'}$ is not accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, if $\delta * (q_0, S) \notin F$
- The language $L^{'}$ not accepted by DFA/NDFA (Complement of accepted language L) is $\{S | S \in \Sigma^* and\ \delta * (q_0, S) \notin F\}$

Theoretically, the rule creation in the reasoning engine should be allowed only to managers, who are responsible for the production line or the maintenance procedures. They are the ones with the most knowledge of the production lines and the risks that happen continuously. Another reason for selecting FSMs is the flexibility and ease of use to a non – experienced user. The system allows the rule completion to be done relatively easy by trained personnel due to a user interface which follows the flow of the FSMs.



ID: 1900f64b-1570-7d74-646b-0598b48a2c9a

Name *: Bin Collector - Optimal route

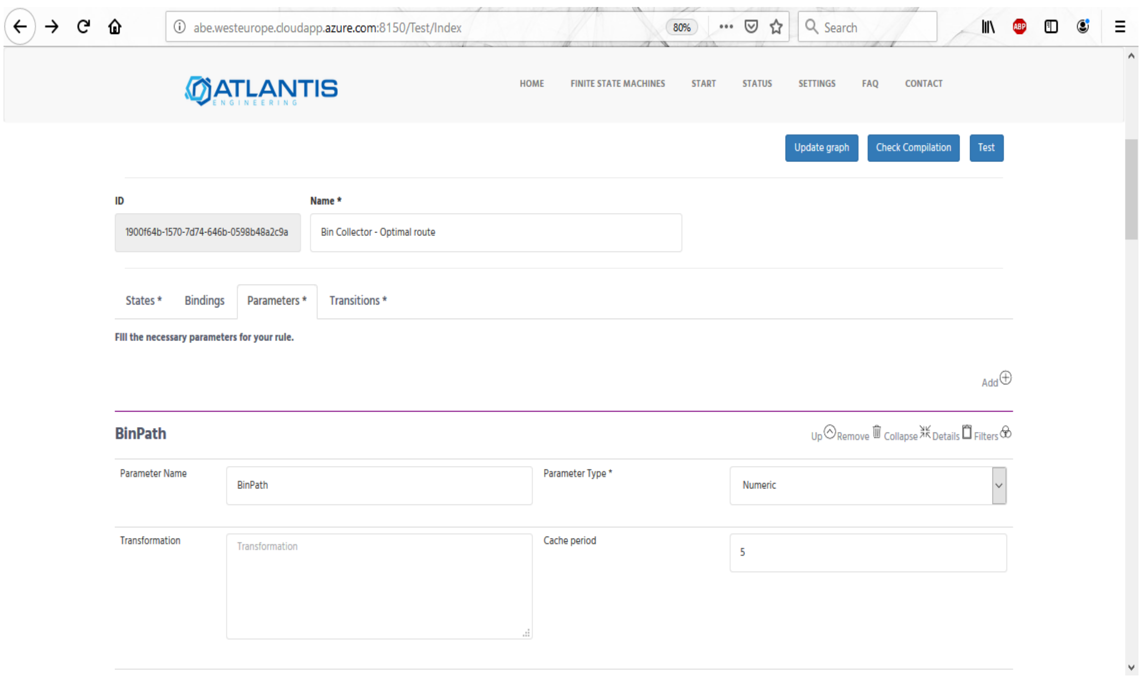
Normal Operation State Description: Empty Bin

Insert the states for equipment or processes, e.g. normal operation, warning, alarm.

Full Bin

Figure 14: States of DSS FSMs

Figure 14 and Figure 15 show the states and parameters screen in the DSS GUI.



ID: 1900f64b-1570-7d74-646b-0598b48a2c9a

Name *: Bin Collector - Optimal route

Fill the necessary parameters for your rule.

BinPath

Parameter Name	BinPath	Parameter Type *	Numeric
Transformation	Transformation	Cache period	5

Figure 15: Parameters of DSS FSMs

4.6 Scripting

The system provides a way for system operators to supply ad hoc methods to evaluate conditions, configure parameters, pre-process data etc., since it is expected that the complexity of the needed expressions for the above will exceed that of what can be represented in a predefined GUI form.

The way this is done is by accepting user defined scripts that can be assigned to specific stages and “junction points” of the DSS pipeline. Since the DSS is supposed to be a solution targeted to domain users, any of these scripts will be able to be hot-loaded into the running instance of the DSS and to be assignable from the User GUI (without any compilation/build/deployment and other “programmer task” related steps.

The chosen method to allow user scripting was to take advantage of a C# real time compilation framework (Roslyn, from Microsoft), C# being the language the DSS was written in itself. This has several advantages in comparison to other scripting methods (Javascript/Python/Lua) such as higher performance and compile time checking.

In addition, the user isn’t asked to write code directly in C# but in a simplified Domain Specific Language like syntax which provides several shortcuts to referring to the DSS entities and makes use of lambda expressions (the syntax of which is generic enough for the learning curve to be lower).

Code in the DSL syntax is automatically checked and compiled to C#, with any errors being supplied to the user upon compilation.

The scripting sub-component can be thought of as an independent module of the application even though it is directly used by other sub-components as a library.

The reasoning sub-component uses this system to declare parameter transformations and ad hoc transition firing conditions. The recommendation sub-component can use it to directly inject parameter and transition data into message templates.

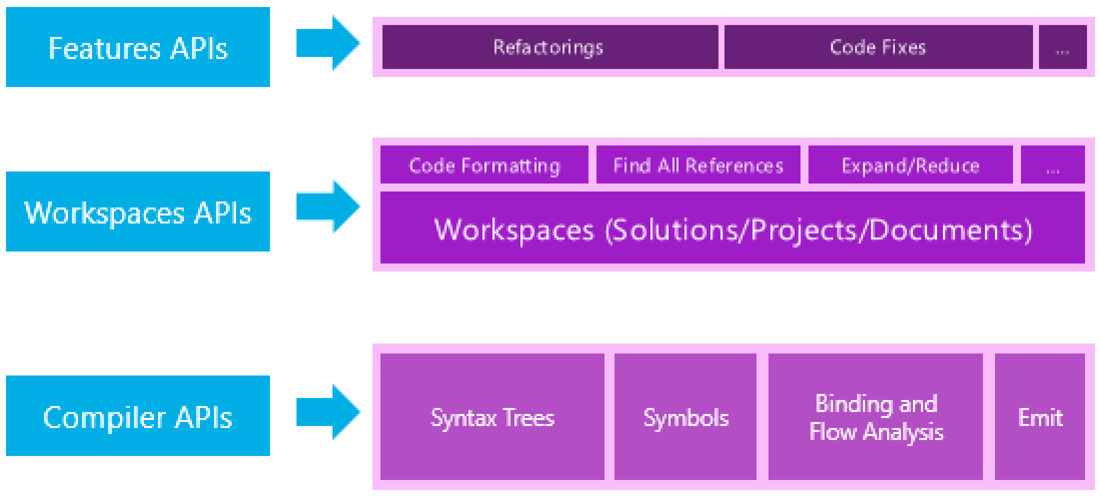


Figure 16: Roslyn compiler APIs [17]

Roslyn compiler APIs are shown inFigure 16 The first version of Roslyn was released in October 2011 as a part of Community Technology Preview (CTP)[17] – an extension for Visual Studio 2010 SP1. The update of CTP in September 2012, despite the large scale, was not very successful. It had the so-called “breaking changes” – changes in Roslyn components, which could potentially crash other components. Besides, not all the features of the CTP APIs were implemented for C# and Visual Basic languages.

Starting with 2015 version, Visual Studio uses Roslyn to compile and build its own projects. However, to date, Roslyn only supports two languages – C# and Visual Basic.

Most of the existing traditional compilers come as “black boxes”, which “magically” convert the source code into an executable file or library. Unlike them, Roslyn allows you to access each stage of the code compilation and application creation process via its own APIs.

Together with compilers, other “black boxes” are often supplied – integrated development environments (IDEs) that can enable you to increase the development speed with convenient tools, such as code highlighting, Intelligence, refactoring tools, performance analysis tools (profilers) and other complex tools. Roslyn takes over these features and also provides an API to them.

The Roslyn compiler pipeline is represented by four phases, each of which has its own object representation:

1. The parser displays information in the form of a syntax tree;

2. The symbol declaration phase displays a hierarchical symbol table;
3. The binding phase returns information in the form of semantic analysis results;
4. The emitting phase provides APIs for generating low-level code in MSIL language (similar to what System.Reflection.Emit does).

Language services use these APIs to perform their own functions. For example, code highlighting uses a syntax tree, while an object browser uses a hierarchical symbol table.

Roslyn diagnostic APIs allow you to handle errors and warnings that occur at all the compilation stages. Roslyn also allows you to process errors through analysis tools written by the user.

Scripting APIs allow executing C# or Visual Basic code without compilation – something similar to the REPL interactive environment in Perl, Python, Haskell, Erlang, and others.

Workspace APIs gives direct access to the application's object model in the compiler without parsing the source code files for the second time. The APIs also allow for projects tuning, management of project dependencies, source code generation without using Visual Studio components.

The syntax tree is the basic structure used by Roslyn for compilation, code analysis, binding, refactoring, code generation and other operations. Roslyn syntax trees have three key properties:

1. They contain all the source information, such as grammatical constructs, tokens, directives, comments and even whitespaces – all this information is contained in the syntax tree;
2. The syntax tree or its part can be converted back to the source code – you can build syntax trees and generate code from them, you can edit the syntax tree and it will generate a corrected code;
3. They are thread-safe and protected from changes. This means that you will not be able to directly change the data in the syntax tree. The tree completely reflects the state of the source code at the time of construction.

These three important attributes of the trees allow you to work with the syntactic structure of the source code, including in custom projects, accessing it through APIs. These properties have also greatly simplified complex refactoring operations, and this happens naturally without direct code editing but only by editing the syntax tree. Each syntax tree consists of the following elements:

- Syntax Nodes – they represent complex syntactic constructs, such as declarations or expressions;
- Syntax Tokens – they represent the simplest constructs for constructing syntax nodes. Syntax tokens consist of, for example, an identifier or operator;
- Syntax Trivia – it represents parts of the source text that are mainly insignificant for the compiler, such as comments, directives or whitespace;

- Spans display positions within the source text of each node, token or trivia, and its length;
- Kinds identify the syntax unit in the tree;
- Errors are processed in the syntax tree in two ways: either by inserting the expected token or by adding a token that is unknown to the compiler as a trivia.

Unlike syntax trees that represent the structure of source code, semantics is the logic in the source code and all its constructs. It includes declarations of variables, classes, objects, fields, methods, function calls and passing parameters to them, types of operands and operation results, and operator priorities. Semantic analysis of source code checks the code (or syntax tree in Roslyn) for compliance with the rules of the language. The semantic model provides the following information about the source code:

- Semantic symbols: source elements or elements imported from libraries (types, methods, properties, fields, events, etc.),
- Resulting type of expression,
- Diagnostic data: errors, warnings, exceptions, etc.

Workspace APIs represent the object model of solutions, projects in solutions and documents in projects. All the objects and methods listed above can be called from any .NET application working with Roslyn as a service and using Roslyn APIs.

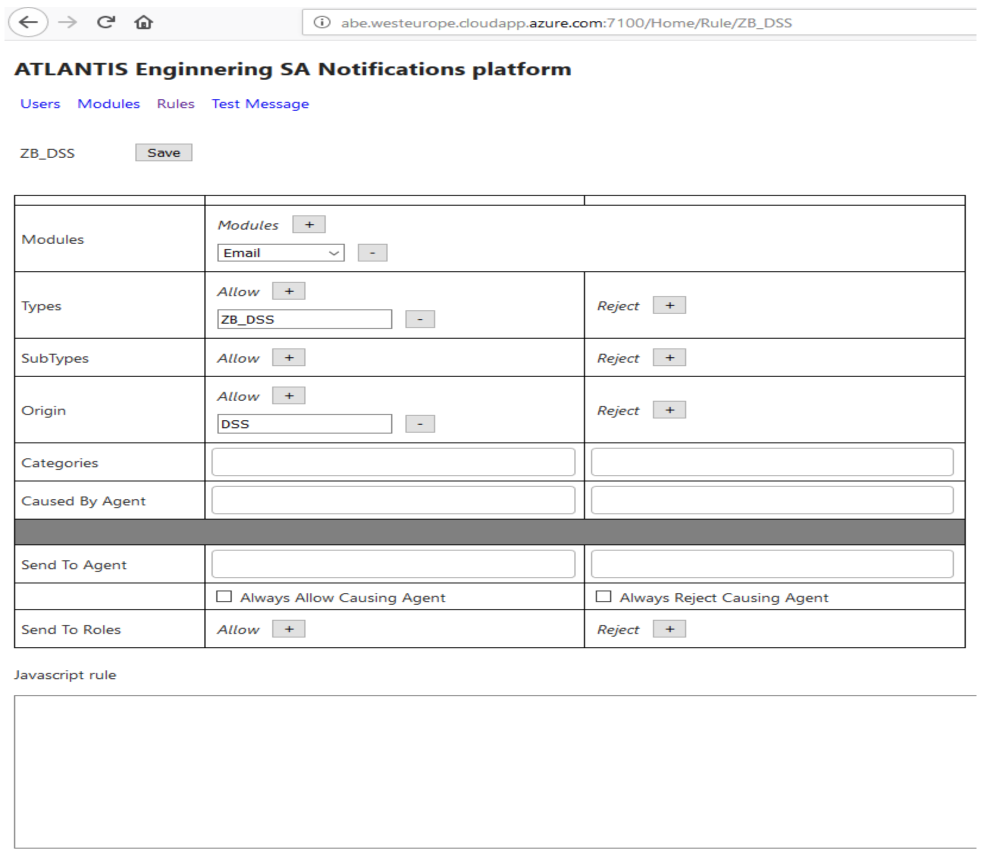
4.7 Notifications

The recommendations sub – component is the output layer of the system. It is responsible for organizing the recommended actions to accompany rules and state machine transitions, and for their proper dissemination to those responsible for executing them on the shop floor.

The system allows the declaration of abstract recommendations which can be directly supplied by experts or imported through a set based on an industry standard. Then, it “attaches” these recommendations to the transitions of our STM model, with the semantics of this action being that whenever transition X is activated supply these recommendations.

Attached recommendations are augmented with data regarding their method of dispersal, such as options to send them via email forward them via MQTT, link them to CMMS commands provided by the CMMS integration sub-component etc. In addition, they can take attributes such as a characterization of their severity, as Alerts, Suggestions and Warnings.

The above system usually works with our notifications sub-component, whose purpose is to set up rules that match incoming messages based on their metadata, to agents of which the attributes and contact data have been loaded by a plugin. In this case, it means that we can set up rules that determine for example which agents can receive recommendations of an “Alert” level and on specific assets, and how they should be notified for them.



abe.westeurope.cloudapp.azure.com:7100/Home/Rule/ZB_DSS

ATLANTIS Enginnering SA Notifications platform

Users Modules Rules Test Message

ZB_DSS Save

Modules	*Modules* + Email -	
Types	*Allow* + ZB_DSS -	*Reject* +
SubTypes	*Allow* +	*Reject* +
Origin	*Allow* + DSS -	*Reject* +
Categories		
Caused By Agent		
Send To Agent		
	☐ Always Allow Causing Agent	☐ Always Reject Causing Agent
Send To Roles	*Allow* +	*Reject* +

Javascript rule

Figure 17: Notification configuration mechanism UI

The notifications component has a separate user interface, as shown in Figure 17 since the functionality of assigning agents to incoming recommendation messages via "access control list" like rules is sufficiently independent and of a lower level for the DSS rule-based system to directly deal with it. An implicit layer exists for indirection (the DSS user does not specify specific agents for which recommendations are to be provided, but only characterizes the recommendations with metadata, it is expected that the actual matching will be done by another admin using the notifications sub-component independently).

4.7.1 Mobile Application – Recommendation Feedback

For the requirements of our system, a type of agent feedback rating received recommendations is needed. This is mainly planned to be used by supervised Machine Learning techniques that will make use of those ratings as training input. However, it could theoretically be used by domain experts directly, and to that end aside from a numerical rating, we have also incorporated a user comments system through which administrators and domain experts can directly communicate with those executing recommendations.

The first edition of the Recommendations sub – component was a one – way notification from the system to the user. It was implemented along with the Notification sub – component to disperse 1 – way messages to agents.

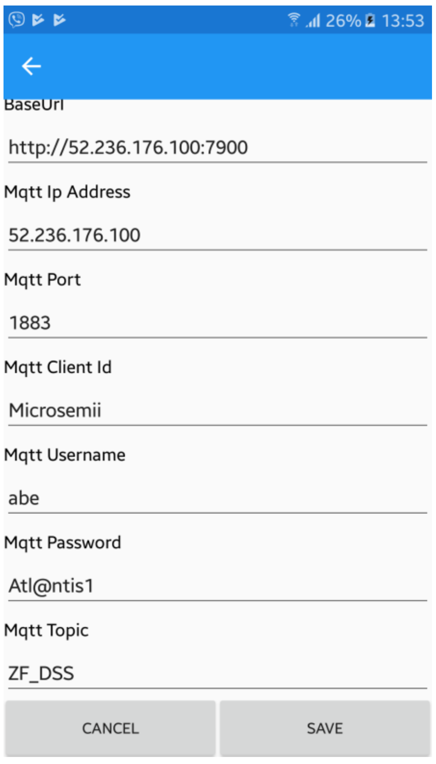
In order to allow users to provide feedback back to the system, we have created a mobile application, section 4.8 and Figure 18, that acts a separate target for sending recommendations. The application allows the user to view received recommendations, rate them and provide user comments. This data is sent back to the server and linked to the recommendation and transition activation. The application is implemented as a Xamarin app for the Android OS and relies on MQTT for communication with the server.

As an aside benefit to using this system, ML techniques applied to the data gathered by the DSS become collaborative since many users can implicitly modify the ML technique's training set independently and in real time.

4.8 Recommendation Provisions

The DSS outcome is the feedback that is sent to the addressed users through the sub – components of recommendation provisions.

The sub – component is based on the MQTT protocol. It sends the suggestions to a RabbitMQ and a feedback android application consumes the suggestions.



BaseUrl
http://52.236.176.100:7900
Mqtt Ip Address
52.236.176.100
Mqtt Port
1883
Mqtt Client Id
Microsemii
Mqtt Username
abe
Mqtt Password
Atl@ntis1
Mqtt Topic
ZF_DSS

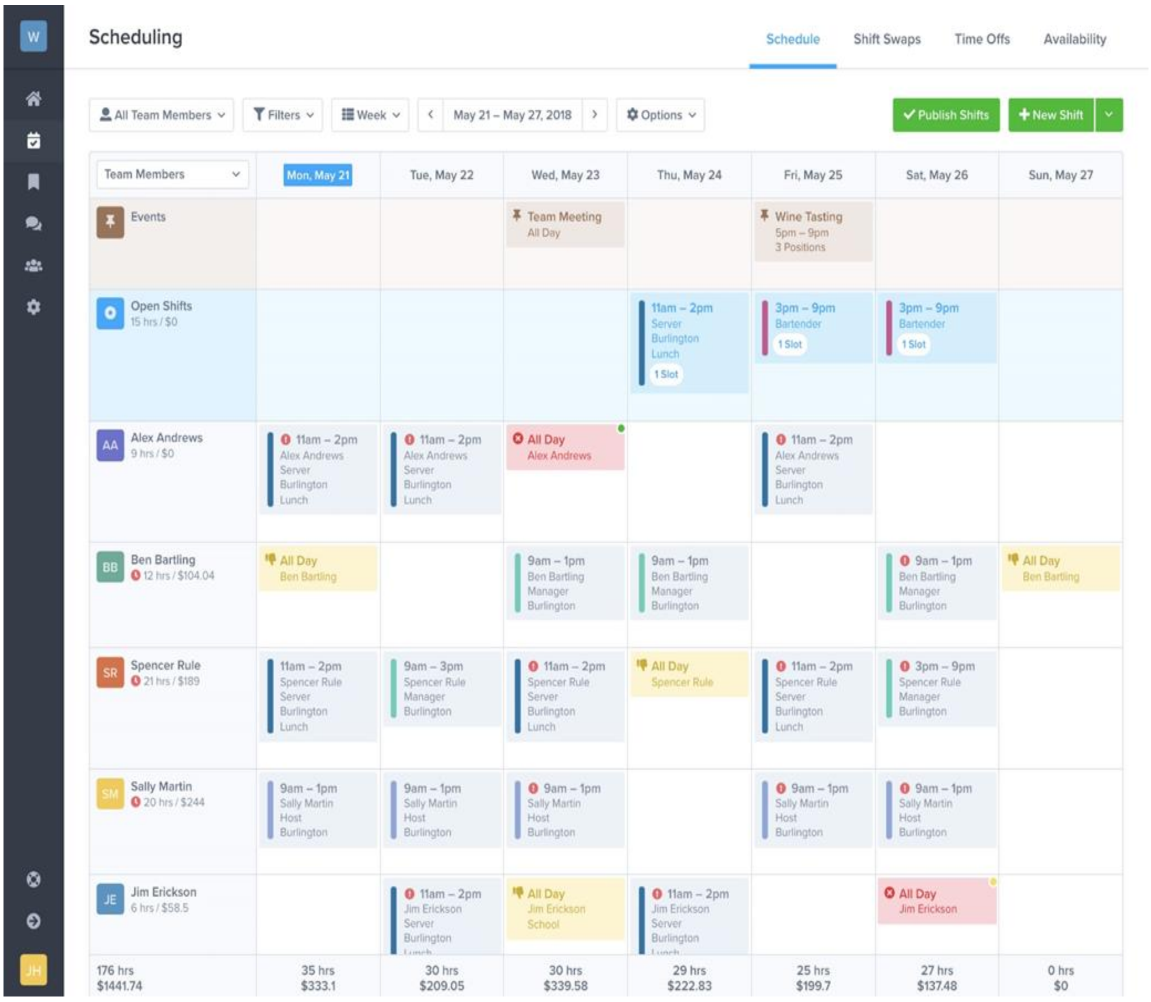
Figure 18: Set up ofthe MQTT client on the user's side for the Recommendation Provision sub - component

Figure 18 shows the set-up screen on the mobile application for the recommendation provisions sub – component. The user connects to the DSS base URL and subscribes to a certain topic with their username and password. The client creates the a connection with the topic and when the connection is established, the user receives the respective messages.

4.9 Maintenance Tasks Assignment and Scheduling

Maintenance tasks assignment and scheduling is a standard process in industrial manufacturing. A maintenance task carries a number of constraints. It is long enough, the appointed for maintenance asset and the addressed personnel have to be available. Additionally, the maintenance process must not interfere with the production scheme, because any interferences could create collisions and interruption in the production line.

Besides the scheduled maintenance tasks, there are unexpected assets break downs, that lead to extra and unexpected maintenance tasks. In that case, the schedule should be updated immediately in order the unexpected task to fit in.



Team Members	Mon, May 21	Tue, May 22	Wed, May 23	Thu, May 24	Fri, May 25	Sat, May 26	Sun, May 27
Events			Team Meeting All Day		Wine Tasting 5pm – 9pm 3 Positions		
Open Shifts 15 hrs / $0				11am – 2pm Server Burlington Lunch 1 Slot	3pm – 9pm Bartender 1 Slot	3pm – 9pm Bartender 1 Slot	
AA Alex Andrews 9 hrs / $0	11am – 2pm Alex Andrews Server Burlington Lunch	11am – 2pm Alex Andrews Server Burlington Lunch	All Day Alex Andrews		11am – 2pm Alex Andrews Server Burlington Lunch		
BB Ben Bartling 12 hrs / $104.04	All Day Ben Bartling		9am – 1pm Ben Bartling Manager Burlington	9am – 1pm Ben Bartling Manager Burlington		9am – 1pm Ben Bartling Manager Burlington	All Day Ben Bartling
SR Spencer Rule 21 hrs / $189	11am – 2pm Spencer Rule Server Burlington Lunch	9am – 3pm Spencer Rule Manager Burlington	11am – 2pm Spencer Rule Server Burlington Lunch	All Day Spencer Rule	11am – 2pm Spencer Rule Server Burlington Lunch	3pm – 9pm Spencer Rule Manager Burlington	
SM Sally Martin 20 hrs / $244	9am – 1pm Sally Martin Host Burlington	9am – 1pm Sally Martin Host Burlington	9am – 1pm Sally Martin Host Burlington		9am – 1pm Sally Martin Host Burlington	9am – 1pm Sally Martin Host Burlington	
JE Jim Erickson 6 hrs / $58.5		11am – 2pm Jim Erickson Server Burlington	All Day Jim Erickson School	11am – 2pm Jim Erickson Server Burlington		All Day Jim Erickson	
176 hrs $1441.74	35 hrs $333.1	30 hrs $209.05	30 hrs $339.58	29 hrs $222.83	25 hrs $199.7	27 hrs $137.48	0 hrs $0

Figure 19: Maintenance tasks scheduling general screen[6]

Figure 19 depicts a generic example of scheduling software.

4.9.1 Smart Asset Management Algorithms

We will incorporate a, dynamically adapted to the current operating conditions, the algorithm for Smart Asset Management (SAM) for the generation of a maintenance plan. To this end it identifies:

- the optimal time for maintenance activities in terms of workload,
- production capacity and schedule,
- availability of the support team,
- the stock level of spare parts,
- and interfacing with the higher level management systems for co-scheduling maintenance and production

The assignment processes

The assignment problem is a fundamental combinatorial optimization problem. In our case, we have a number of skilful technicians and a number of tasks, and we have to assign the tasks to the technicians, based on a number of criteria and constraints. We achieve the assignment by taking into consideration the following:

- ***Personnel availability***
  The software communicates with higher MES/ERP/CRM systems, and has access to the technicians' availability, based on a number of factors like days off, shifts etc.
- ***Priority score***
  The priority score provides the task's weight (importance). It is measured by the following formula:
  $$PriorityScore = (0.7 * AssetCriticality) + (0.3 * TaskCriticality)\%$$
- ***Spare parts availability***
  If the needed spare parts are not available at the moment, the task filtered out, till the spare parts are available.
- ***Technician skills and experience***
  We match the technicians to the tasks based on the needed skills and experience (TE). TE is a percentage value (0% to 100%). We classify the experience based on the Table 7 below:

Table 7: Scheduler Classification for Personnel

Experience	*Percentage*
Unexperienced	*0% - 20%*
Junior	*21% - 40%*
Intermediate	*41% -60%*
Senior	*61% - 80%*
Master	*81% - 100%*

We keep this generalized score, because each company calculates the experience based on different formulas.

While the skills issue is one to one matching, we had to find a way to make sure that the task will be assigned to the technician, who not only has the appropriate skills, but also the appropriate experience. We came up with the following formula:

$$MatchingScore = |TE - PS|$$

The matching score has to be as close to zero as possible.

E.g. let’s assume we have a task with a **PS (PriorityScore) = 67%**, and three technicians that all of them have the necessary skill, but different experiences **TE1 = 90%, TE2 = 22% and TE3 = 60%**. The matching scores are the following:

- ⍰⍰⍰⍰h⍰⍰⍰⍰⍰⍰⍰⍰⍰=90−67=23
- ⍰⍰⍰⍰h⍰⍰⍰⍰⍰⍰⍰⍰⍰= 22−67=45
- ⍰⍰⍰⍰h⍰⍰⍰⍰⍰⍰⍰⍰⍰=60−67=7

We select the third matching score.

We set the matching score as an absolute value, because we want to assign first the most crucial tasks to the most experienced technicians, and then, if they are done with those, we assign to them, easier for them tasks. **So, we have actually implemented a greed search algorithm.**

The scheduling processes

The scheduling takes under consideration the following constraints for a prioritized execution order

1. the Priority Score
2. the start date
3. task's duration

Each task is assigned with a “priority ticket”. The task with the “highest value” ticket is to be executed first and so on. Tasks with the same “ticket value” are executed on First In First Out (FIFO) basis. A priority-based scheduler can be seen in Figure 20.

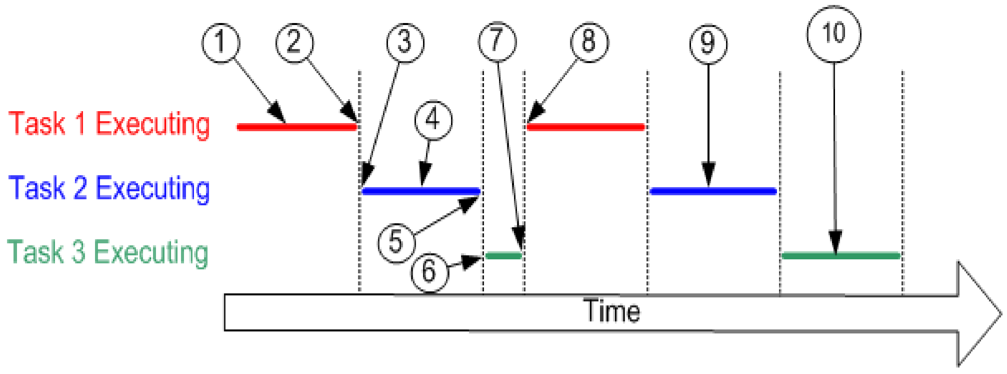


Figure 20: Priority Based Scheduler[21]

Rearrange existing schedule

An Interval Tree is an ordered data structure whose nodes represent the intervals and are therefore characterized by a start value and an end value. A typical application example is when we have a number of available intervals and another set of query interval, for which we want to verify the overlap with the given intervals.

In particular, to make more efficient this type of search we can use an Augmented Interval Tree, a structure in which the information contained in each node is increased by adding, in addition to the bounds of the range, also the information related to the maximum value of the subtree of the node that we are analysing.

The trivial solution is to visit each interval and test whether it intersects the given point or interval, which requires $O(n)$time, where n is the number of intervals in the collection. Interval trees have a query time of $O(\log n + m)$ and an initial creation time of $O(n \log n)$, while limiting memory consumption to $O(n)$. After creation, interval trees may be dynamic, allowing efficient insertion and deletion of an interval in$O(\log n)$ time. If the endpoints of intervals are within a small integer range (e.g., in the range $[1, \dots., O(n)]$), faster data structures exist with pre - processing time O(n) and query time $O(1 + m)$ for reporting m intervals containing a given query point.

4.10 Prediction Module

The prediction module classifies time series as a sequence of events instead of a discrete set. A sequential event set reduces the effort of timestamping the created events and the data annotation of sensor measurements. Time is essential only for critical failures in the fault prediction methods. The key point is the create a discrete series of events in the span of the time series. The Matrix Profile (MP) is deployed for that purpose. It is a data structure that annotates the time series. The main advantage of the MP is that can provide efficient solutions in cost – effective data mining problems concerning time series. The basic idea behind the MP is the estimation of the Euclidean distances of the sub – sequences of the times series. The distances define the similarities between the sub – sequences. The similarities are the basis of the analytics task performed for the predictive maintenance module.

4.10.1 The MP Algorithm for Artificial Events

The algorithm to create artificial events is based on the estimation of MP. The applied technique also is responsible for extracting hidden patterns to predict or timely detect failures. The only requirement of the described technique is the existence of raw measurements from a sensor network on the shop floor.

Based on the pattern – length parameter (PL), an application of MP computes the sub – sequences of length PL and generate the Matrix Profile Index (MPI). The MPI is a directed graph, where each edge points to the most similar sub – sequence. MPI graph is $G = (V, E)$ where $V = v_1, \dots, v_n$ denotes a set of nodes and $E = e_1, \dots, e_n$ defines the edges of the graph G weighted by the values in MP. There edges in the graph that have globally or locally very high weights. A set of thresholds is applied to eliminate the nodes with the weighted edges, due to the fact that in most cases they are just noise in the system. The applied filters are: the edges of the graph that connect two nodes when their distance is X times greater than the distance of the edge connecting the sink node to its nearest neighbour (local rule) are removed from the graph. Also, all edges with a weight more than Y times the mean MP value are filtered out.

The following step estimates the weakly connected components (sub – graphs) of the MPI graph and maps each component to a distinct event. Components with less than Z members are omitted.

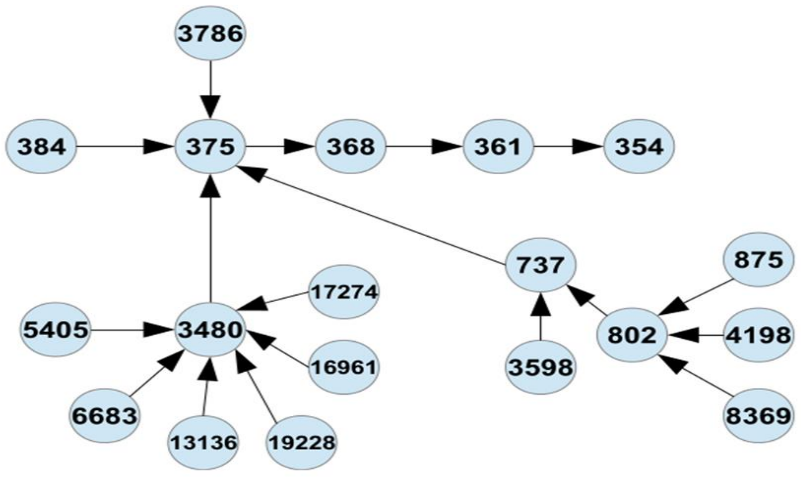


Figure 21: Example of connected sub - components after processing the MPI

Every point in the time series is part of a connected component labelled by the id of that component. The values decide for the three thresholds are based on the experiments conducted on data set from the shop floors. Figure 21 shows the application of MP – based algorithm for $PL = 10$. 19 elements of the original timeseries are mapped to the same artificial event based on their similarity.

4.10.2 Event – based Predictive Maintenance Solution

The created artificial results are mapped to time stamps based on the original time series and partitioned in ranges defined by the occurrences of the fault the predictive maintenance targets (failure modes). The ranges are split into time segments, the size of which (i.e. minutes, hours, days) corresponds to the time granularity of the needed analysis.

There is a rationale behind the time segmentation and dictates that the segments closer to the end of the range may contain fault events indicative of the main event. Considering this rationale a process function can be taught to quantify the risk of the targeted failure in the near future, if the events precede the rationale. The proposed function is a sigmoid function which maps higher values to the segments closer to the machine failure. The steepness and shift of the sigmoid function are configured to better map the expectation of the time before the failure at which correlated events will start occurring. The segmented data in combination with the risk quantification values are fed into a Random Forests algorithm as a training set to form a regression problem (the objective of which is the minimization of the mean squared error). In practice, the event types are hundreds if not thousands. Each event type is essentially a dimension. Therefore, to increase the effectiveness of the approach standard pre-processing techniques can be applied:

1. Rare events (RE) (much rarer than the fault events) are considered as extremely rare, hence they can be removed to reduce the dimensionality of the data.
2. Multiple occurrences (MO) of the same event in the same segment can either be noise or may not provide useful information. Hence, multiple occurrences can be collapsed into a single one.
3. Most frequent events (FE) can be removed, as they usually do not contain significant information since they correspond to issues of minor importance. The events that occur more frequently than the faults are considered as such.
4. Events of minor importance occur and appear in every segment until their underlying cause is treated by the technical experts. Hence, only the first occurrence (FO) of events that occur in consecutive segments can be maintained.
5. Standard feature selection (FS) techniques [23] can also be used in order to further reduce the dimensionality of the data.

Finally, to deal with the imbalance of the labels (given that the fault events are rare) and as several events appear shortly before the occurrence of the fault events, but only a small subset of them is related to them, Multiple Instance Learning (MIL) can be used for bagging the events. A single bag contains fault events of a single hour. Using MIL, the data closer to the fault events (according to a specified threshold), are over-sampled, so that training is improved.

4.10.3 Experiments and results

The experiments were done using historical time series data and converting them to a continuous stream. The ground truth used for the measurements is the information of the timestamps that the machine stopped working due to technical reasons, e.g. damage to a module on the press; this information has been provided by the engineers responsible for the machinery under investigation. Each machine stop represents a failure mode, each prediction represents an alarm and the detected stops are the ones that have at least one preceding alarm within a fixed period before the fault. We assess the efficiency of the supervised learning technique, based on the recall and precision metrics adapted to the PdM context, measured based on the following definitions: Precision is the ratio of the successfully predicted stops to the number of total alarms, and Recall is the ratio of the predicted stops to the number of total stops, where a stop is considered as successfully predicted if there is any prediction made in a specified time gap before a machine stop. Multiple alarms inside the specified time gap for the same machine stop are counted as a single alarm, while the false alarms (i.e. before the time gap) are counted individually. The rationale is that the maintenance engineers are prompted to respond to the first alarm for a specific machine stop, while in the case of the false alarms, they are called to respond to every one of them.

The data used for the assessment of the supervised learning approach are the acoustic emission measurements. The acoustic emission sensors are placed in 6 different spatial positions on the cold forming press, generating data in 6 distinct channels providing measurements in hundreds of different angles per channel. We perform dimensionality reduction by aintaining only the maximum value across all angles per channel per time point. I.e., finally, for each timestamp, there is a single measurement per acoustic channel.

Table 8: Number of distinct artificial types generated per pattern length (PL)

Source	PL-5	PL-10	PL-50
Channel 1	6411	5615	1794
Channel 2	6401	5800	5234
Channel 3	6374	5722	3591
Channel 4	6334	5148	1305
Channel 5	6353	5837	3445
Channel 6	6396	5724	4216

The experiments share a common parametrization and fine-tuning is beyond the scope of this work. Three values of pattern-length (PL) for the Matrix Profile are used (i.e. 5, 10 and 50). Due to the generation of multiple distinct artificial event types, as depicted in Table 8, the event type that represents the machine stops (target event) is the most frequent one, hence the RE, FE and FO pre- processing steps are disabled as they are more beneficial for cases where the frequency of the target event is lower than the other event types. MO is enabled in all the experiments and over-sampling (MIL) is applied. The steepness and the shift of the sigmoid are set to 0.8 and 4, respectively, the threshold for the value of the sigmoid function to set an alarm is set to 0.3, while the time gap for true alarm consideration is set between 1 and 8 hours before a closing machine incident. All measurements refer to 10-fold cross validation. As there are lots of event types, FS pre – processing step is also tested. For partitioning the dataset into 10 folds, we use the number of incidents and not the number of time segments.

Table 9: Experimental results on all the acoustic channels using the supervised learning technique (CD: community detection).

Source	PL	FS	Recall	Precision	F1-score
Channel 1	5	X	0.61	0.61	0.61
Channel 1 CD	5	X	0.57	0.63	0.59
Channel 2	10		0.63	0.53	0.55
Channel 2 CD	5		0.49	0.45	0.47
Channel 3	10	X	0.55	0.61	0.57
Channel 4	50		0.49	0.76	0.59
Channel 5	50	X	0.45	0.80	0.54
Channel 6	50	X	0.48	0.68	0.50

Table 9 presents the recall and precision values, of the results that achieved the best F1-score per channel. The second column depicts the pattern-length used in the Matrix Profile algorithm, while the third one indicates the usage of the FS pre - processing step. The Table also presents the results in two of the channels (i.e. 1st and 2nd) where the community detection (CD) algorithm is used in place of the connected component (CC) algorithm utilized in the MP-based artificial event generation approach. As we observe, Channel 1 and Channel 4 achieved the highest F1-score (0.61 and 0.59 resp.). There is no clear winner between the different pattern-lengths and whether feature selection has applied or not. Regarding the application of the CD, the results are inferior to those achieved by CC, despite the fact that the number of the generated artificial event types is almost the same in both cases.

Table 10: Experimental results on all the acoustic channels using an ensemble of the supervised learning technique.

Source	Strategy	Recall	Precision	F1-score
Ch.4-Ch.1	AND	0.62	0.5	0.55
Ch.1- CD – Ch.1	OR	0.59	0.82	0.67

Next, we employ two simple ensemble strategies with two predictors each: the AND strategy, where two predictors need to raise an alarm, and OR strategy, where an alarm is raised whenever at least one of the predictors' votes for it. We have computed the precision, recall and F1-score of all the possible pairs between all the previous experiments. The results with the highest F1-score per strategy are shown in Table 10. As we observe, the OR strategy was able to enhance the previous results, achieving 0.67 F1-score combining two cases with low recall but high precision. Note, that in this scenario, a random predictor achieved F1-score of 0.31;

moreover, a dummy predictor with recall 1 through raising an alarm every 7 hours cannot exceed F1-score of 0.58.

4.10.4 An unsupervised learning technique based on streaming outlier detection

In this section, we present the streaming distance-based outlier detection algorithm, namely Multiple Component in One Database (MCOD) [13], that was used for early detection of failure on the dataset. We first introduce the background needed for streams and the MCOD algorithm and we conclude this section by presenting the experiments and the results of the unsupervised technique.

Outlier detection techniques are used to identify noise and anomalies in a dataset. In a streaming environment, because of the infinite nature of the data, detecting outliers need to be done in combination with windowing techniques. A window splits the data stream into either overlapping or non-overlapping finite sets of data-points based either on the arrival of each point or the number of points.

For the Philips use case, we use a distance-based outlier detection algorithm, in which the number of a data point's neighbours represent its status as an anomaly or a normal point, as explained in Definition 1. We employ sliding windows. A recent work [5] compares the state-of-the-art distance-based outlier detection algorithms used in streaming settings, in both CPU and memory consumption. The results of the comparison show that MCOD has the best performance in most scenarios. MCOD uses the notion of micro-clusters in order to drastically cut down the number of distance computations needed to assess a data point as normal or anomalous.

Definition 1. *Given a set of objects O and the threshold parameters R and k, report all the objects oi for which the number of neighbours $o_j.nn < k$, i.e., the number of objects $o_j, j \neq i$ for which $dist(o_i, o_{j)} \leq R$ is less than k. The report should be updated after each window slide.*

Note that according to the above definition, outliers may be reported during any time they belong to the window and not necessarily when they are first inserted into it.

4.11 Internal Suggestion and Alert System

The Alert, Suggestion and Recommendation sub-system is at a high level responsible for defining and organizing DSS Recommendations, or semantically similar entities like Alerts or Suggestion, and then dispersing those while also recording them for further analysis. The component can work on its own but also be configured to forward them to other Z-BRE4K components or other systems such as ERPs or CMMS.

5 DSS IN THE OVERALL Z-BRE4K SYSTEM

DSS is a part of the overall Z-BRE4K architecture. It works along with the rest of the Z-BRE4K components. The system receives processed and raw data from the various sources, implements the rules in the reasoning engine, creates suggestions based on the risk assessment methods and sends them to the user. The collaboration between the Z-BRE4K components is essential for creating an uninterrupted data flow on the shop floors.

DSS communicates with the Predictive Maintenance and planning to communicate with FMECA component using the infrastructure of the Semantic Framework. The component diagram with the interacting DSS components is given in Figure 22 below:

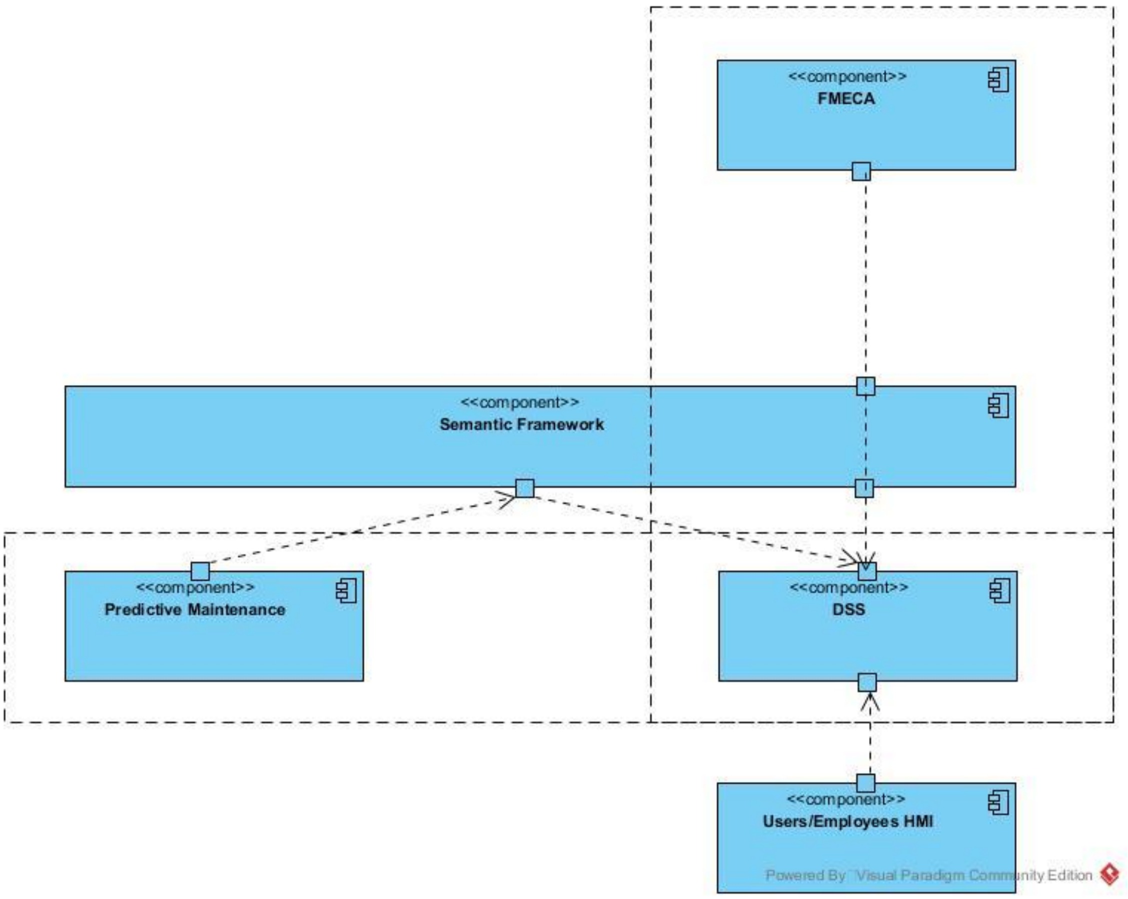


Figure 22: Component diagram with the connected to DSS Z-BRE4K components

The DSS participates in two sections of the Z-BRE4K components. The first section regards the prediction mechanism of the system. The use of predictive and preventive maintenance is critical in a manufacturing environment.

The second section is the risk assessment methods of the Z-BRE4K architecture. Risk assessment is implemented for both DSS and FMECA components.

5.1 Semantic Framework

The semantic framework includes an ontology, which serves as a common knowledge model for all components. The ontology entities model the shop floor operations, procedures and actors. Entities are also used in recognising assets, critical components, failure modes and effects.

Apache Jena will be used for knowledge storage. It provides data retrieval interface through a SPARQL query engine. The input data format are XML files. Output data is rich models about the shop floor procedures and their criticality as RDF triples for the Knowledge Based System and Semantic rules for analysis.

The APIs for the Semantic Framework and their descriptions are given in the Table 11 below:

Table 11: Semantic Framework APIs

Semantic Framework API	Description
SPARQL	The SPARQL Web service is used to send custom SPARQL queries against the SF RDF repository as a general-purpose querying web service. Developers communicate with the SPARQL web service using the HTTP POST method. Each results document can be serialized in many ways, and may be expressed as one of the following mime types: (i) text/xml, (ii) application/rdf+xml, (iii) application/rdf+n3, (iv) application/ sparql-results +xml, (v) application/sparql-results+json or (vi) application/json. The content returned by the web service is serialized using the mime type requested and the data returned depends on the parameters selected.
Knowledge Management	The SF RESTful web services can be accessed directly via API or command line, or may be controlled and interacted with using standard content management systems (CMSs). Each request to an individual web service returns an HTTP status and optionally a document of result sets. Each results document can be serialized in many ways, and may be expressed as RDF, pure XML or JSON.

5.2 Preventive Maintenance

Predictive maintenance Z-BRE4K component provides an event analysis method, finds the causes of failure and breakdown, evaluates the impact and eliminates the potential causes by recognising the events that lead to breakdowns. Another purpose of the predictive maintenance component is the recognition of the chain reactions that cause failure and the rise of alerts for corrective actions.

The main data inputs for the predictive maintenance component are machine simulations: CAD/CAE schemas of machines and assets, electromechanical properties and specifications, embedded data acquisition systems for condition monitoring which provide sensorial data, specifications of SCADA systems and machine simulations files. Also, the predictive maintenance component exploits the historical data from CMMS systems, reports of types of breakdowns and risks and reports of Fault Detection, Fault Diagnosis and Prognosis procedures. The main data output for the Predictive maintenance component is the predicted mean time between failures, the predicted service time and the prognosis.

Predictive Maintenance functionality focuses on the machine simulator which provides visual analysis of the state of the machines, using mathematical models and logical functions. The simulator provides illustrates potential failures and their causes. The results of the simulator ignite the predictive maintenance functionality which provides failure mode, type of failure, possible failure cause and mean time between failures.

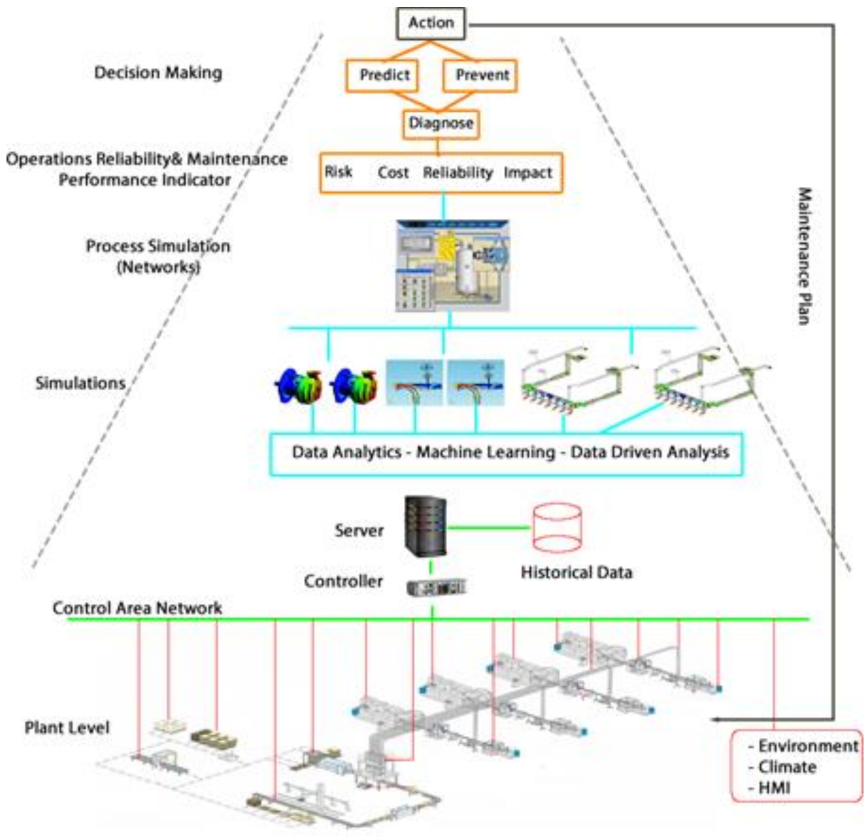


Figure 23: Predictive Maintenance and Machine Simulators functionality Flow[32]

The APIs for the Predictive Maintenance component is given in Table 12 below:

Table 12: Predictive Maintenance APIs

Predictive Maintenance APIs	Description
Supported API for other entities to communicate with Machine Simulator	Description of the supported API: Interfaces with machines, controllers, production planning applications, and BRE4Kdown alarm systems, Machine and component CAE/CAM/Simulator files (API coming directly from those sources – OEM facilitated or from AUTOWARE)
Supported API for other entities to communicate with Predictive Maintenance	Interfaces with Component A (Machine Simulators), machines, controllers, production planning applications, and BRE4Kdown alarm systems, Machine and component CAE/CAM/Simulator files (API coming direct from those sources – OEM facilitated or from AUTOWARE)

5.3 FMECA

Failure effects identified by the FMEA analysis and each one is given a severity class. Failure event frequency is calculated by the failure data and multiplied with the mission time of concern giving the criticality number $C = \lambda * T$, where $\lambda = FM$ and $t = time\ of\ component\ or\ system\ operation$. FMECA also uses Risk Indicators and computes the Risk Priority Number (RPN). FMECA analysis defines Risk as $R\ (Risk) = S\ (Severity) * P\ (Probability)$ and RPN as $RPN = S * P * D\ (Detection)$.

Severity is defined as four distinctive levels: insignificant, marginal, critical and catastrophic. The frequency has five different levels: improbable, remote, occasional, probable, frequent. The FMECA classification of Failure Modes gives the main FM categories: failure during operation, failure to operate at the prescribed time or the given time, failure to cease operation at prescribed time, premature operation and failure due to the lower level component.

The main inputs for the FMECA component are: machines and their assets, FMs that occur in each machine, severity and the effects caused by FMs, failure data and KRIs. The main FMECA outputs contain the calculated risks, RPN, criticality numbers per FM, criticality matrix and alerts.

The basic functionality of the FMECA component is the computation of Risks, RPN and probabilities of a FM. Also, the FMECA is able to create an effect builder which identify the failures which affect system operations. Classification of identified failure modes according to characteristics such as detection, capability and operating provisions is another output of the FMECA component. The causes of the effects, the criticality matrix and alerts are also output data.

FMECA component provides the DSS with the results of FMECA analysis as input in the reasoning engine to create rules for maintenance and re – adaptation scheduling purposes. The FMECA APIs are shown in Table 13 below.

Table 13: FMECA APIS

FMECA APIs	Description
Supported API for other entities to communicate with FMECA	Description of the supported API: Interfaces with sensors and retrieves the sensorial data. Computes the risks based on the analysis and mathematical forms for the data. The data format comes from the supported schema and usually is in XML or JSON. (API coming directly from those sources – OEM facilitated or from AUTOWARE)
Supported API for other components to communicate with FMECA	Interfaces with other components. The API gives permissions to FMECA to retrieve or send data to other components. Input data is the supported FMECA format, output data is in a format supported by other entities. Calls between the components rely on communications protocols defined in the API. E.g. FMECA and DSS communicate between them and the data are in the supported XML or JSON format, based on the ontology entities defined in the Semantic Framework Z-BRE4K component. (API coming directly from those sources – OEM facilitated or from AUTOWARE)

5.4 DSS in the IDS Ecosystem

The ATLANTIS Predictive Maintenance (PdM) solution communicates with the INNOVA+'s IDS connector in order to get access to data from the Philips production line. The use of an IDS connector is crucial in order to meet specific requirements, which are encapsulated at the core of the ID Ecosystem as:

- TRUST - Each participant is evaluated and certified before being granted access to the trusted business ecosystem.
- SECURITY AND DATA SOVEREIGNTY - Apart from architectural specifications, security is mainly ensured by the evaluation and certification of each technical component used in the International Data Spaces.
- ECOSYSTEM OF DATA - The architecture of the International Data Spaces pursues the idea of decentralization of data storage, which means that data physically remains with the respective data owner until it is transferred to a trusted party
- STANDARDIZED INTEROPERABILITY - Each Connector is able to communicate with any other Connector in the ecosystem of the International Data Space.

- VALUE ADDING APPS - The International Data Spaces allows app injection into the IDS Connectors in order to provide services on top of data exchange processes.
- DATA MARKETS - The International Data Space enables the creation of novel, data-driven services that make use of data apps.

The ATLANTIS PdM solution applied in the Philips use case communicates directly with another ATLANTIS component, the Decision Support System (DSS). The latter is responsible for the circulation of the PdM results to the appropriate departments/engineers of the Philips plant. Hence, apart from the data transfer between the Philips plant and the ATLANTIS PdM solution, there are two extra data transfer routines

1. PdM → DSS,
2. DSS → Philips

which can potentially take place using the IDS Connectors.

Figure 24 presents an abstract architecture design of the components participating in the Philips use case.

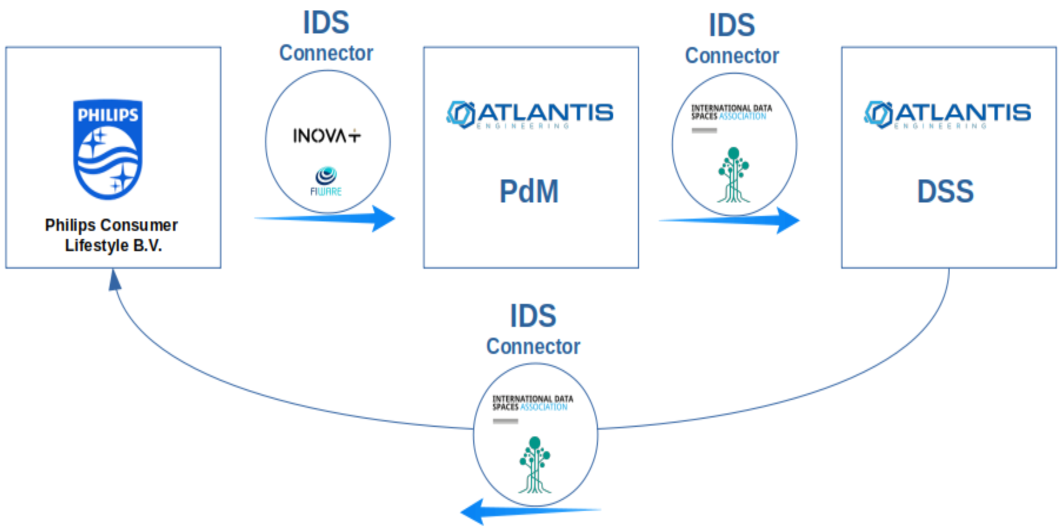

IDS
Connector
IDS
Connector
PHILIPS
Philips Consumer Lifestyle B.V.
INOVA+
FIWARE
ATLANTIS ENGINEERING
PdM
INTERNATIONAL DATA SPACES ASSOCIATION
ATLANTIS ENGINEERING
DSS
IDS
Connector
INTERNATIONAL DATA SPACES ASSOCIATION

Figure 24: DSS in an IDS Ecosystem

The INOVA+'s Industrial Data Space (IDS) connector is based on an open source FIWARE implementation and more specifically the FIWARE Orion Broker component, which is extended with specific capabilities to meet the Philips use case special characteristics, such as persistent data storage and on – demand data request. ATLANTIS on the other side has strategically decided to build its IDS connectors utilizing the Trusted IDS Connector implementation provided by the IDSA. The usage of different IDS Connector implementations on different layers of the same use case is not causing any problem, as it is already stated that the Standardized Interoperability specification is at the core of the IDS Ecosystem (i.e. different IDS Connector implementation can still exist on the same IDS Ecosystem).

As it is presented in the Figure, the data from the Philips plant are transferred to the ATLANTIS PdM solution through the INOVA+'s FIWARE-based IDS Connector. The ATLANTIS PdM component supports the data transfer to the ATLANTIS DSS component using the IDSA's Trusted IDS Connector and finally the ATLANTIS DSS component supports the results circulation back to the Philips plant through an IDSA's Trusted IDS Connector.

The IDSA's Trusted Connector provides a subscription plan for the data exchange functionality, using MQTT servers (queues) for the data transfer on top of the IDS Protocol. Both the ATLANTIS components (PdM and DSS) support the communication with MQTT servers, hence it is possible to create an MQTT topic in which the PdM solution will publish its results and the DSS will subscribe in order to receive them and process them accordingly.

Considering the communication between the DSS and the Philips plant, currently the information is transferred using an ATLANTIS proprietary data transfer approach. However, the DSS component has the needed endpoints implemented to support the data transfer through an IDSA's IDS Trusted Connector.

6 Conclusions and Next Steps

The Z-BRE4K DSS is a system created based on standards and methods concerning risk assessment. The risks that occur in a manufacturing environment are considered by a DSS which is developed with a specification for manufacturing processes and maintenance procedures. Also, the DSS is developed with the use of the Z-BRE4K strategies and different strategies correspond to different actions of the DSS. Also, the Z-BRE4K strategies allow the system to develop a reasoning engine which can correctly suggest solutions to problems that arise, as well as send notifications and re – adapt the maintenance schedule.

The system architecture is based on different sub – components which interact between them but they are developed as stand – alone sub – components. The architecture allows the modular effect of the DSS and a degree of flexibility which does not exist in other systems. A dedicated Z-BRE4K API is used for communication with the rest Z-BRE4K components.

The communication between DSS and FMECA will be implemented through the subscription engine of the FIWARE Orion Broker. DSS and FMECA will play the role of producer and consumer at the same time. They will subscribe to context information, so when an event, a value change happens, the consumer application will get an asynchronous notification. This way, there is no need to continuously repeat query requests (i.e. polling). The Orion Context Broker will let you know the information when it arrives.

The traditional data-driven prognostic approach is to construct multiple candidate algorithms using a training data set, evaluate their respective performance using a testing data set, and select the one with the best performance while discarding all the others. This approach has three shortcomings:

- the selected standalone algorithm may not be robust;
- it wastes the resources for constructing the algorithms that are discarded;
- it requires the testing data in addition to the training data.

To overcome these drawbacks, we will follow an ensemble data-driven prognostic approach which combines multiple – member algorithms with a weighted-sum formulation.

Next steps DSS development includes industry – accepted standards with regard to Remaining Asset Health Estimation, Key Performance Indicators and Auditing Criteria, in a structured manner that enables user integration to the Asset Tracking and Rule Based Reasoning solution, as well as to report generation summarising the shop floor's efficiency over time. The possibility of using an industry standard for auditing will be available to users that will be able to implement the standard without special configuration. In addition, exploitation of the current external Machine Learning component access points will be implemented in the system as an attempt to enhance the product to provide a suite of Machine Learning methods that can possibly be directly applied by general users via a graphical user interface, without heavily depending on a specialized data scientist's offline data-set analysis.

7 REFERENCES

1. Aggarwal, C. (2015). Outlier analysis. *Data mining*, 237-263.

2. Almeida, M., Moreira, N., & Reis, R. (2007). On the performance of automata minimization algorithms. *Technical Report Series*.

3. *Cyber Security Key Risk Indicators. An Automated Report for the C - Suite*. (12 de July de 2019). Obtenido de Outpost24: https://outpost24.com/blog/cyber-security-key-risk-indicators-automated

4. (2018). *D3.9 - Manufacturing Decision Support System II.* Brussels: The COMPOSITION project.

5. *Failure mode and effects analysis*. (12 de July de 2019). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Failure_mode_and_effects_analysis

6. *Free scheduling software designed for hourly employees*. (16 de July de 2019). Obtenido de ZoomShift: https://www.zoomshift.com/services-other

7. *GeeksforGeeks*. (12 de July de 2019). Obtenido de Job Assignement Problem using Branch and Bound: https://www.geeksforgeeks.org/job-assignment-problem-using-branch-and-bound/

8. *ISO 55001:2014*. (12 de July de 2019). Obtenido de bsi: https://www.bsigroup.com/en-GB/Asset-Management/

9. *ISO/IEC 31010*. (12 de July de 2019). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/ISO/IEC_31010

10. Katie, B. (12 de July de 2019). *Asset management system standard published*. Obtenido de International Organization for Standardization: https://www.iso.org/news/2014/01/Ref1813.html

11. *Key Risk Indicators*. (12 de July de 2019). Obtenido de The Institute of Operational Risk: https://www.ior-institute.org/sound-practice-guidance/key-risk-indicators

12. Kiran, D. (2017). Failure Modes and Effects Analysis. En D. Kiran, *Civil Aircraft Electrical Power System Safety Assessment* (págs. 187-216). Tianjin, China: Civil Aviation University of China, Tianjin, China.

13. Kontaki, M., Gounaris, A., Papadopoulos, A., Tsichlas, K., & Manolopoulos, Y. (2016). Efficient and flexible algorithms for monitoring distance - based outliers over data streams. *International systems 55*, 37 - 53.

14. Korovesis, P., Besseau, S., & Vazirgiannis, M. (2018). Predictive maintenance in aviation: Failure prediction from post flight reports. *IEEE Int. Conf. on Data Engineering ICDE*, 1414-1422.

15. (2018). *KRI models and tools.* Brussels: The Z-BRE4K project.

16. Kubiak, T. M. (12 de July de 2019). *Quality Progress*. Obtenido de Conducting FMEAs for Results: http://asq.org/quality-progress/2014/06/34-per-million/conducting-fmeas-for-results.html

17. *Microsoft Roslyn - using the compiler as a service*. (12 de July de 2019). Obtenido de instinctools: https://www.instinctools.com/blog/microsoft-roslyn-using-the-compiler-as-a-service

18. N., N. (12 de July de 2019). *Bright Hub Project Management*. Obtenido de A Comparison of Risk Analysis Versus FMEA (Failure Modes and Effects Analysis): https://www.brighthubpm.com/risk-management/71098-risk-analysis-vs-fmea/

19. organisation, I. (12 de July de 2019). *IEC 31010:2009*. Obtenido de International Organization for Standardization: https://www.iso.org/standard/51073.html

20. *Risk Assessment*. (12 de July de 2019). Obtenido de Canadian Centre for Occupational Health and Safety: https://www.ccohs.ca/oshanswers/hsprograms/risk_assessment.html

21. *Scheduling*. (16 de July de 2919). Obtenido de freeRTOS: https://www.freertos.org/implementation/a00005.html

22. Tay, K. M. (2008). On the use of fuzzy inference techniques in assessment models: part II: industrial applications. *Fuzzy Optimization and Decision Making, Volume 7, September*, 283-302.

23. Tran, L., Fan, L., & Shahabi, C. (2016). Distance - based outlier detection in data streams. *Proceedings of the VLDB Endowment 9(12)*, 1089-1100.

24. *tutorialspoint*. (2018). Recuperado el 16 de April de 2018, de https://www.tutorialspoint.com/automata_theory/deterministic_finite_automaton.htm

25. *tutorialspoint*. (2018). Recuperado el 17 de April de 2018, de https://www.tutorialspoint.com/automata_theory/non_deterministic_finite_automaton.htm

26. Vafeiadis, T., Nizamis, A., Apostolou, K., Charisi, V., Metaxa, I., Mastos, T., . . . Papadopoulos, A. (s.f.). Intelligent Information Management System for Decision Support: Application in a lift manufacturer's shop floor.

27. *What is a messaging queuing service?* (12 de July de 2019). Obtenido de MuleSoft: https://www.mulesoft.com/resources/cloudhub/what-is-a-messaging-queuing-service

28. *What is failure mode?* (12 de July de 2019). Obtenido de Minitab(R) 18 Support: https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/reliability/supporting-topics/basics/what-is-a-failure-mode/

29. *What is message queuing?* (12 de July de 2019). Obtenido de CloudAMQP: https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html

30. *What is the Pland - Do - Check - Act (PCDA) Cycle?* (12 de July de 2019). Obtenido de Learn about Quality: https://asq.org/quality-resources/pdca-cycle

31. Yeh, C., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H., . . . Keogh, E. (2016). Matrix profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. *IEEE 16th International Conference on Data Mining, ICDM 2016*, 1317-1322.

32. (2018). *Z-BRE4K system architecture.* Brussels: The Z-BRE4K project.

33. Χαράλαμπος, Α. (2014). *Η νέα έκδοση το ISO 9001:2015.* Αθήνα: TUV AUSTRIA HELLAS.