Ecosystem for COllaborative Manufacturing PrOceSses – Intra- and Interfactory Integration and AutomaTION
(Grant Agreement No 723145)

D3.9 - Manufacturing Decision Support System II

Date: 2019-02-26

Version 1.0

Published by the COMPOSITION Consortium

Dissemination Level: Public

Co-funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement No 723145

Document control page

Document file: D3.8_Manufacturing_Decision_Support_System_V1.0.docx
Document version: 1.0
Document owner: ATL

Work package: WP3 – Manufacturing Modelling and Simulation
Task: T3.4 – Decision Support System for Optimising Manufacturing Process
Deliverable type: [OTHER]

Document status: ☒ Approved by the document owner for internal review
☒ Approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Efraimidis Georgios	2019-01-21	ToC and Initial Text
0.2	Papadopoulos Nikolaos (ATL)	2019-01-29	State of the Art
0.3	Nadir Raimondo (LINKS)	2019-01-21	Deep Learning Toolkit
0.4	Vagia Rousopoulou, Thanasis Vafeiadis, Dimosthenis Ioannidis, Alexandros Nizamis (CERTH)	2019-02-08	DFM and SFT
0.5	José Ángel Carvajal Soto	2019-02-08	Learning Agent and LinkSmart
0.6	Ziazios Constantinos, Oustampasidis Dimitrios (ATL)	2019-02-10	Data Persistence, Data Streaming and Processing
0.7	Oikonomou Fanis (ATL)	2019-02-16	Summary and Conclusions
0.8	Charisi Vasiliki (ATL)	2019-02-19	Decision Support System and HMI
0.9	Oustampasidis Dimitrios (ATL)	2019-02-19	Final General Improvements
1.0	Charisi Vasiliki (ATL)	2019-02-19	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Giuseppe Pacelli (LINKS)	2019-02-21	The document is well structured and reflects the work carried out during the task. Only minor comments have been added.
Mathias Axling, CNet	2019-01-22	Approved with minor comments

Index:

1 Executive Summary

In this deliverable, it is described the work for the COMPOSITION task T3.4 – Decision Support System for Optimising Manufacturing Processing. The deliverable is the update of D3.8 – Manufacturing Decision Support System I. It contains all the updates done in the T3.4 from M20 to M30.

The DSS combines information from included Business Process Diagrams (BPD) and Digital Factory Model (DFM) and based on the semantic models produced as well as from all stakeholders involved in the complete supply chain. The task extends the work done in other projects such as SatisFactory (SatisFactory, 2015). The main aim of the DSS is to make a step forward towards a better understanding of the involved manufacturing processes and operations, the contribution of individual links of the supply chain, the effect of process monitoring in productivity, to facilitate communication and knowledge sharing among departments with different roles and responsibilities, the maintenance requirements and procedures and the detection of daily production details and flaws (COMPOSITION, 2016).

The data is received via MQTT and AQMP protocols and formats (XML and JSON) provided from the partners involved in the ongoing processes and is real -time processed. It is be coupled with the associated requests to certain parts of the supply chain, SOP (standard operating procedures) and response strategies, in order to offer feedback to the involved internal or external suppliers, in terms of actionable knowledge and recommendations, including maintenance operations and schedules. The DSS system provides feedback at fixed and mobile terminals in attractive communication interfaces and visualise KPIs and real-time data analytics

In this deliverable an analysis of the current State of the Art for DSS is given, including streaming processes and security options. Also, the COMPOSITION DSS is analysed and examined in detail, along with its algorithms and. a small description of the components that work together with DSS and the integration of them in the system

2 Abbreviations and Acronyms

Table 1: Abbreviation and Acronym Table

Acronym	Meaning
2FA	Two-Factor Authentication
ACL	Access Control List
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
ANN	Artificial Neural Network
API	Application Programming Interface
BDA	Big Data Analytics
BMS	BigData Manufacturing Storage
BPD	Business Process Diagram
BSL	Boston Scientific
CMMS	Computerised Maintenance Management System
CPU	Central Processing Unit
CSP	Credential Service Provider
DFA	Deterministic Finite Automaton
DFM	Digital Factory Model
DLT	Deep Learning Toolkit
DM	Data Mining
DMA	Direct Memory Access
DSS	Decision Support System
EHP	Energy Harvesting Process
FoF	Factories of the Future
FSM	Finite State Machine
GDSS	Group Decision Support System
GUI	Graphical User Interface
HMI	Human Machine Integration
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Provider
JMS	Java Message Service
JSON	JavaScript Object Notation
KBS	Knowledge Based System
KPI	Key Performance Indicator

LA	Learning Agent
MIP	Mixed Integer Programming
MIS	Managing Information System
MQTT	Message Queuing Telemetry Transport
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
MVC	Model-View - Controller
NDFA	Non-Deterministic Finite Automaton
NDSM	Non-Deterministic State Machine
OEM	Object Exchange Model
OGC ST	Open GeoSpacial Sensor Thing
OLAP	Online Analytical Processing
OPC	Open Platform Communications
PC	Personal Computer
PK	Port Knocking
RAM	Random Access Memory
RBS	Rules Based System
REST	REpresentational State Transfer
RPM	Rounds Per Minute
SoA	State of the Art
SOAP	Simple Object Access Protocol
SOP	Standard Operating Procedures
SQL	Structured Query Language
SSO	Single Sign-On
TD	Temporal Difference
ToC	Table of Contents
ubiDSS	Ubiquitous Computing Technology-Based Decision Support System
UC	Use Case
UI	User Interface
WSN	Wireless Sensor Network

3 Introduction

3.1 Purpose, context and scope of this deliverable

The purpose of this deliverable is to provide a detailed description of the Manufacturing Decision Support System used in the COMPOSITION project. The DSS will include information from both Business Process Diagrams (BPD) and Digital Factory Models (DFM) Based on semantic models produced during the project, as well as needs and specifications from all stakeholders involved in the supply chain, DSS will be able to extend its scope and development done during other projects, such as SatisFactory (SATISFACTORY, 2014).In the SatisFactory project the DSS was oriented to worker ease and facilitation. The decision-making process involved gamification actions for worker satisfaction, safety protocols and task in order to cope with accidents on the shop floor or abnormal operations that can be dangerous.

Development and Implementation of DSS provides a better understanding of the involved manufacturing processes and operations, the contribution of individual links of the supply chain, the effect of process monitoring in productivity, to facilitate communication and knowledge sharing among departments with distinct roles and responsibilities, the maintenance requirements and procedures and the detection of daily production details and flaws.

The scope of this deliverable is to provide information about the State-of-the-Art Analysis concerning the Decision Support Systems and their implementation to manufacturing environments. Furthermore, an analysis of DSS in the composition project. System architecture, subcomponents, dependencies and connections between them are analysed in detail, providing the algorithms used in the process. Finally, the connections and interactions of other COMPOSITION components are explicitly examined. The conclusion of the deliverable gives a small review of the steps taken during the creation of the DSS and indicates the steps that should be followed in the future.

3.2 Content and structure of this deliverable

The deliverable follows the structure agreed by all partners concerning all COMPOSITION deliverables. A Summary is provided in the beginning of the deliverable, providing its key points and it is the first chapter of the deliverable.

Chapter 3 introduces the deliverable. The Introduction consists of a description of the purpose, context and scope of the deliverable. Also, a small description of each chapter is given in the introductory chapter.

Chapter 4 gives the State-of-the-Art Analysis about Decision Support Systems. Trends followed in development of DSS, algorithms and methods applied to improve them are examined in this chapter. Analysis and cross examination of the pros and cons of each method is provided. Also, reasons why a certain method or algorithm is chosen against another are available in chapter 3.

Chapter 5 contains the analysis of the actual Decision Support System in the COMPOSITION project. The overall architecture of the system is examined, its subcomponents and their interactions. Each subcomponent is analysed and details about its functionality and purpose in the system are given. Also, dependencies of each subcomponent are described and connections between each subcomponent are analysed. Data flow throughout the DSS is extensively examined. The DSS API is designed in detail, as well as component diagrams and flow charts.

Chapter 6 is the representation of the COMPOSITION components that interact with DSS. The components main functionalities are briefly described in this chapter as well the components' purposes in the project.

Chapter 7 provides the interaction and integration with the rest of COMPOSITION components. Communication protocols and data exchange are analysed in this chapter. Data streaming between different components is given a thorough inspection. Connecting APIs are considered in the length of this chapter. Diagrams can provide further information in the process.

Chapter 8 contains the conclusions of the deliverable. It describes how development proceeded concerning the DSS and the further steps should be taken to further improvement.

Finally, the references used throughout the text are given as well as the tables of figures and tables and any possible annexes.

4 State of the Art Analysis

Decision Support Systems (DSS) have been used for several decades to help decision-makers in the processes concerning help on shop floors and taking the correct decisions in the most suitable time and situations. The State-of-the-Art Analysis for DSS provides the knowledge of modern technologies concerning DSS and new trends that improve DSS. Also, DSS definition and types are presented in this chapter, as well as the new research areas for DSS. Finally, State of the Art Analysis provides information for Stream Processing techniques and Security options in a system.

4.1 Decision Support Systems

Decision Support Systems are information systems that support business or organisation decision-making processes. Management, daily operations and planning as well as maintenance are some of the processes in an organisation that are facilitated by DSSs. Usually higher-level managements need to decide about the above processes in a rapidly changing environment, considering unpredictable factors in advance. DSS aids them in the decision-making process and the problems they face in operational and organisational sectors. The problems are called Unstructured and Semi-Structured decision problems.

4.1.1 DSS Structure

Fully computerised, human–powered and combination DSS exist nowadays. Decision–making processes is different in concept for academia and manufacturing world. While academics consider DSS as tools for decision–making process, managers operate them as facilitators in operational processes. Analytical techniques and models, along with traditional data access and data retrieval functionality are often found in Decision Support Systems. Also, User Interface (UI) and Human Machine Interaction (HMI) are developed for DSS, because end–users need to interact with the system in an easy and appealing way. (Shim, et al., 2002)

Raw data from various sources such as sensors and machines on shop floors, documents from all operations, business models and personal knowledge are the DSS data sources. All information contained in those data sources is combined by DSS and result to the extraction of KPIs and suggestions for decision-makers. Data is usually gathered from organisation’s data inventories (databases – both legacy and relational, Data Warehouses and Data Marts), sales figures and periodic reports, revenue predictions, maintenance reports, task and scheduling reports and standardised procedures on the shop floor (Pirog - Mazur, 2004).

Depending on the described DSS operation, three main DSS components can be identified: database, software application and UI. The database is the place where all data is stored. Data acquisition and sources are already described. The software application is the heart of the system. The software contains all rules and models that analyse the data and the create suggestions for the decision-makers. Software is coded in high-level programming language such as C, C++, C#, Java. Finally, DSS UI allows users to have an appealing communication with the system. Drop-down menus, lists, navigations bars, filling boxes and other user interface components are used to enable DSS rule creation by user, KPIs and suggestion visualisation. Graphical environments are developed using the latest trends on application design using HTML and CSS.

DSS are classified as Decision Science tools or Data Science tools. When they are located as a front–end mechanism in Problem Solving with a well-defined and structured framework which answers business questions, they are Decision Science tools. On the other hand, Data Science is defined as a sophisticated version of Data Driven Analytics and Modelling and when DSS are used as ruled engine to create suggestions for decision-makers according to data analysis and models, they are classified as Data Science tools.

Manufacturing companies collect data on manufacturing processes, supply, customers and production. The purpose is to create and use a DSS that uses the provided information to support management and decision–making process based on

- Historical trends
- Projected forecasts based on trends
- Identification of problem causes in production line

Knowledge acquisition through DSS is another major factor for which manufactures prefer those system. Complex structure of the links and associations between the gathered data and the difficulty to formalise the data itself leads to the use of DSS in the decision-making process, because they can structure the information of the data and extract knowledge based on their rule engine the models that allow the discovery of hidden patterns and connection, between seemingly unrelated data (Miah, 2012).

Furthermore, DSS' structured data exploitation in based on well–defined rules and the established models facilitate knowledge acquisition. Data is ample in manufacturing environment and based on the already existing ruled of DSS, it can be used to train the system for correct prediction. Machine learning and intelligence is the next step of deploying DSS in manufacturing environment. Training the system is possible depending on the existence of data and new rules can be created to adapt to the circumstances shown in the data. Adaptive DSS can be very helpful on shop floor, where unpredicted events often happen during working, help decision–making processed to maximise processes on the shop floor and minimise costs and time waste.

4.1.2 DSS History

Decision Support Systems have been a constant study case since the early days of computerised systems. (Shim, et al., 2002) They have evolved from simple model–oriented systems to multifunctional ones. In the early days of Computer Science, during the 1960's, DSS were based on powerful and expensive mainframe computers and their only functionality was to provide periodic reports in an automated and structured way.

Managing Information Systems (MIS) theory was developed a decade later. MIS theory provided a scientific background for the development of elaborate DSS which supported mainly economical functions such as promotion, prising and logistical values. In the early 1980's academia was involved and interested in DSS provided new and innovative ideas. As a result, DSS development improved vastly during the span of the decade. The 1990's was a decade when Information Technology trended and expanded. Personal computers became common household appliances and more people had access to them. Decision–makers followed the trend of the decade and demanded more efficient systems. The solidified database technologies helped to the size expansion of DSS. Innovative technologies, including higher internet penetration lead to client–server systems that allowed new business operations for DSS. Organisations upgraded their network infrastructure to accommodate those changes. Object Oriented technology and Data Warehouses made their mark on DSS. Combining all the above technologies web driven systems with innovative approaches were developed. Online Analytical Processing (OLAP) is one of the systems developed during 1990's (Kasie, et al., 2017).

Finally, Decision Modelling was integrated into DSS to fully automate decision–making process without any added code and no need for rule/order hierarchy inside the rule family.

4.2 DSS Definition and Description

"DSS are interactive computer-based systems that help decision-makers use data and models to solve ill-structured, unstructured or semi-structured problems." (Shim, et al., 2002)

They provide many possibilities of data analysis while programming effort is kept to a minimum and are usually addressed to managers and higher-level personnel without technical background. Search and data analysis lead to correlation discovery without interference to the MIS, graphical representation of KPIs, suggestion creation and representation. Another significant DSS, according to managers, is the development of complex analysis and alternate scenarios to answer different possible outcomes for an event or a situation. "What if" analysis helps managers to better understand the conditions which lead to certain suggestions by DSS, as well as to enhance their expertise.

Standalone DSS units were common in the past, but now DSS scope has extended to facilitate multiple users. Also, there are different kinds of DSS that focus on models, data, communication or even a combination of all those elements. Depending on what functionality and scope managers prefer in DSS, they are inclined to buy or use a specific DSS that covers all their needs. Thus, the main services of either standalone DSS units or multiple user DSS should be considered.

Data and information are crucial for the existence of DSS, but the most crucial factor is the quality of the data and information provided, instead of the quantity. Unformatted and unlabelled data often causes more problems than solves. The underline data schema of DSS is important for the prediction accuracy and correct suggestion. Furthermore, consistency, relevance, persistence of data augments its quality and eliminate the possibility of information misinterpretation and "bad" decisions by managers (Simon, 1959).

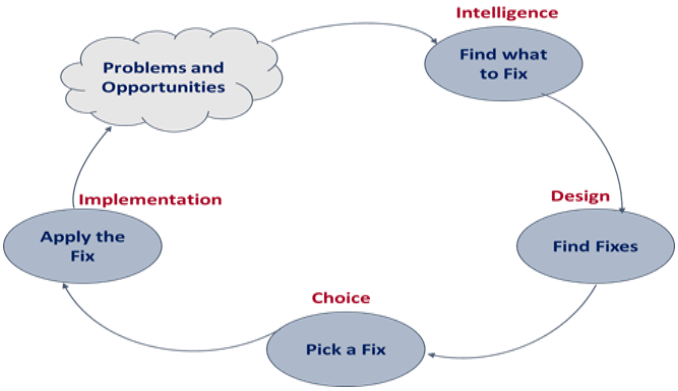

Figure 1: Simon's Decision - Making Process

Figure 1 show Simon's Decision–Making Process. This process is often followed by managers while they are obliged to decide, although they almost follow the steps intuitively. This process allows them to identify the problems, find the existing solutions, pick the one that seems the most suitable solution and apply it. DSS provides the suggestion to the most suitable solution, aggregating data and information coming from them, and rules or models from the decision engine. The described process implies that managers should be able to comprehend the provided data and know the examined operations. The process also implies that data quantity is critical up to a certain point, but data quality is always critical.

Decision-makers should always be able to understand the existing data and discover the connections, principles and value behind it. Also, the assumptions and models used by DSS should also be well known and understood by decision-makers, before asking for more data and information. Exact knowledge leads to better decisions with less effort. Successful decision-makers should prefer knowledge over information and data amplitude.

4.2.1 DSS Types

There are many different types of DSS. The most common ones are Table 2 below: (Felsberger, et al., 2016)

Table 2: DSS Types

DSS Type	Description
Data Driven	They have file drawer, data analysis and analysis information systems. They facilitate Data Warehouse, access and retrieval from large data bases of structured data
Model Driven	There is an underling model that incorporates various disciplines such as: accounting, financial, representation, optimisation model etc. Model Driven DSS manipulate models instead of data in the analysis and the decision-making process. There is no need for large data bases.
Knowledge Driven	They provide suggestion and recommendations to users for certain problems. They also are called management expert systems or intelligent decision support systems. Problem solving is their main focus and they exploit data mining techniques, such as browsing big data for relationship in the content of the data
Document Driven	Retrieval of unstructured documents and web pages with storage and processing technologies for document retrieval and analysis. Documents are: company policies, catalogues, meeting minutes, records, historical documents etc.
Communication Driven	They are also called GDSS (Group Decision Support Systems). Their first purpose is to facilitate the communication between the members of a group in the decision-making process. They support scheduling, document sharing, bulletin boards, electronic communication etc. They are almost always used as project management tools.
Hybrid Systems	Hybrid DSS are composed by diverse types of DSS. There are Web-based DSS that include communication, data and knowledge driven systems. OLAP is a software category with many possible views of information

Intelligent	The latest development of DSS. They combine three several types of DSS into one: • Passive DSS. The classical Data Driven with extensive access to large data bases and rule engine • Active DSS, which provide personalised decision-making process depending on learning processes. Knowledge-Based Systems (KBS) is one of the Active DSS. • Proactive DSS, are also known as Ubiquitous Computing Technology-Based DSS (ubiDSS) and are divided to pull-based proactive, push-based proactive and push-based automated applications.

4.2.2 Modern Technologies Concerning DSS

Modern DSS vary in types as described in the previous chapter, but their common factor is the use of data bases and the access to produced data. Modern companies generate substantial amounts of data daily, and most of it contain useful operational and organisation information. This information can be used during the decision-making process by managers and lead the company to better results. Big data is nowadays a trend in which DSS are based on. Also, connectivity between various kinds of DSS or other systems can be used to enhance learning and suggesting process.

The emerging technologies, derived from the previous description, are: ***grid computing*** – where information sharing is essential between servers (nodes), networks, databases and tools from multiple organisations and integrated and collaborative resource use is allowed; ***cloud computing*** – service-oriented solution which is defined to incorporate virtualisation and utility computing as a parallel and distributed system; ***web services*** – infrastructure to provide stateless, persistence services and to resolve distributed computing issues; ***crowd computing*** – a service platform for mobile phone data from users; ***pervasive computing*** – also known as ubiquitous computing and enables resource computation and utilisation in mobile or environmentally – embedded manner (Miah, 2012)

Big data exploitation for DSS should correspond to the main dimensions of it, called ***Big Data "Vs"***. "Vs" are defined to provide a structure on the data and make it easily accessible and readable. They are:

- ***Volume*** – large amounts of data is generated
- ***Variety*** – data comes from various sources, has many different types and it may be either structured or unstructured
- ***Velocity*** – data is generated continuously and is captured and stored in real-time conditions
- ***Veracity*** – data is always erroneous and inconsistent
- ***Validity*** – data should be measured correctly and fit the perceived criteria
- ***Value*** – the extraction of knowledge and information from the data and how it is used in the organisations' environment.

Implementation of all Big Data "Vs" leads to optimising modes and enhancing improvement of error analysis and prediction of specific faults and situations allows preventive and proactive measures to be taken. Efficiency and effectiveness increases, while rule engines provide operational rules to decision-makers. Furthermore, innovative algorithms and intelligent software applications should be used to produce knowledge and KPIs.

Rules Based System (RBS), Knowledge Based System (KBS), Fuzzy Sets and Neural Networks are some of the algorithm solutions for reasoning and learning. Artificial Intelligence (AI), Data Mining (DM) and data visualisation are also critical in the algorithm development of DSS. Machine Learning (ML) approaches are Artificial Neural Networks (ANN), Genetic Algorithms and Fuzzy Logic.

Rule Engines can be based on classification trees, Branch and Bounce algorithms and Non-Deterministic State Machines. Both Branch and Bounce algorithm and Non-Deterministic State Machine use possibilities to move from one state to another. Branch and Bounce algorithm follows certain steps to reach to a final condition and along the way creates the rule depending on the possibilities. Paths that should be followed are determined by the algorithm. Branch and Bounce algorithm leads to a progressive succession of steps to reach the final decision. Contrary, Non-Deterministic State Machines contain states where a parameter is evaluated and transitions, which comply to certain criteria. While create a decision rule with a state machine the parameter remains the same during the rule. The states represent the possible situations the decision-maker may face

in the process, while the transitions are the possible paths that can lead from a state to another. Each transition can be defined by possibilities, numbers, true or false conditions or other criteria. Moving between different states is defined by the parameter each time, and different transitions can be triggered depending on the conditions each time.

A major difference between Branch and Bounce algorithm and Non-Deterministic State Machines is that while using the Branch and Bounce algorithm the system cannot return to its initial state. On the other hand, non-Deterministic State Machines provides the option to the system to be able to return to its initial state. Simplified forms of Non-Deterministic State Machines are the Deterministic State Machines which often are used in DSS with well-defined operational and organisational objectives. The scenarios have already been fully described and develop, and afterwards the rule engine is deployed. Such systems operate in mature manufacturing environments with well-established practices and known faults and failures. In such cases, the use of the more complex Non-Deterministic State Machines is considered redundant.

4.2.3 DSS Research Areas

DSS have always been in the middle of research areas during the last four decades that have been in use. The most common DSS application are implemented in weather forecasting, production lines, health care organisations, financial and stock market analysis. Also, air traffic control, advance traffic control centres, as well as manufacturing decisions are based on DSS and their suggestions. Oil refinement manufacturing, ATM and bank applications, maintenance decision-making processes are mostly based on DSS.

Large shop floors with many assets are inclined to use DSS instead of CMMS, because CMMS can be absorbed in the DSS. This dual functionality is a research topic in the COMPOSITION project and all the FoF4 European projects. The need to develop smart factories and solve many maintenance, production and safety problem in manufacturing environments drive the research about DSS. Decision-making processes concerning the above factors improve shop floor organisation and maximise benefits while minimising the costs and problems which often happen on shop floors.

The intelligent techniques used in the most recent research studies about DSS are: Data Mining and ANN, Temporal Difference (TD) Learning methods, Genetic Algorithms and Fuzzy Logic. Also, Gaussian Dispersion Model, Mobile Agents and Multi-Agent Systems, as well as the DSS types described in previous chapters are part of the research areas. A general idea of the intelligent techniques and the task appoint to DSS gives Table 3 below:

Table 3: DSS Research Areas

Intelligent Techniques	Task
Data Mining and ANN	Weather Forecasting, Selection and Allocation of Sires and Dams
ANN and TD Learning Method	Sales Prediction
Genetic Algorithm based Fuzzy Neural Network	Measure the Qualitative Effect on the Stock Market
Fuzzy Set and Gaussian Dispersion Model	Air Pollution Control at Coal-Fired Power Plants
Case-Based, Mobile Agent and Multi-Agent	Strategic Choices in term of Technical Interventions on Municipal Infrastructure
Model Based and Rule Based	Measure Enterprise Performance
Knowledge Based System and ANN	Evaluation of Urban Development
Knowledge Based System and Fuzzy Theory	Effective IT Outsourcing Management

MIP-based heuristics and Lot sizing and scheduling	Time Scheduling and Operation Management systems in manufacturing (Figueira, et al., 2015)

Decision Support Systems provide a wide variety on research and development. While they have been studied the previous decades and there are a lot well-documented solutions, the development of big data technologies and the possibilities of storing and accessing so much data, lead to a renewed research environment for DSS. Also, internet adaptation and the advance of IoT opens new horizons in the research areas of DSS in manufacturing environments. DSS and IoT combinations create smart factories and appealing working solutions for both decision-makers and workers. They also lead to more efficient working progress and less repair and maintenance times.

4.3 Stream Processing Technologies

Real–time is an essential part of modern industry. It provides information in real-time for every cause and effect on all industry procedure. It can be used to improve working progress, to avoid lagging times, caused for any reason, improve maintenance procedures and allow the implementation of safety protocols exploiting all the available information (Jain, 2018).

On the other hand, the expiration of data is a significant problem. If certain data will not be used in a certain time frame, its value decreases and the longer it remains idle, the more its value decreases. When data is unexploited, its value is lost, and the actions or decisions based on it can never occur.

Real-time data come continuously into the systems, therefore it can be called streaming data. It needs precise and quick handling because sensors provide data very rapidly and insert it to log files. Only a change in continuous values, can cause severe system changes, but only if it is altered in time.

There are many modern technologies about stream processing in a data pool. It is necessary they are executed in a well-structured data pool accompanied with strict rules and processes in term of ingestion. The main technologies are:

- ***Flink***
- ***Storm***
- ***Kinesis***
- ***Samza***
- ***Kafka***
- ***Rx***
- ***StreamInsight***

4.3.1 Storm

Apache Storm is a distributes real-time processing system. It is designed to directed acyclic graphs and can be used with any programming language. It can compute over one million tuples per second per node, which is very scalable and provides processing guarantees.

Storm is used for machine learning, real-time data analytics and other application with high data velocity. It runs on YARN and integrates Hadoop ecosystems. Yarn is a new JavaScript package manager that replaces the existing workflow for the npm client or other package managers while remaining compatible with the npm registry. It has the same feature set as existing workflows while operating faster, more securely, and more reliably. It is a true real-time processing framework, which does not support batch processing. Storm has low latency and is suitable with data that must be ingested as single entity. Storm provides a bridge between batch processing and stream processing for Hadoop ecosystems.

Storm highlights are:

- Processing power: over 1 million 100-byte messages/node/sec
- Scalability: works on parallel calculations that run across a cluster of machines
- Reliability: guarantees that each data tuple will be processed at least once or exactly once. Messages are only replayed when there are failures

4.3.2 Flink

Flink is an Apache streaming data flow engine which provides facilities for distributed computation over streams of data. It also provides possibilities for batch processes and is equally effective for both batch processing and real-time processing framework, but it puts stream processing first. Flink supports a number of API such as: DataStream API, DataSet API for Java, Scala and Python and SQL-like query API for embedding in Java, Scala static API code. There is also machine learning library implemented in Flink, called FlinkML, SQL query called MRQL and graph processing libraries

Flink is more stream-oriented compared to Spark and Storm, and often is considered as a hybrid between Spark and Strom. "***Apache Spark is an open-source cluster-computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance"***. (Zaharia, et al., 2010) It provides a highly flexible streaming window for continuous stream model, which ensures that both batch and real-time streaming is integrated into one system.

The highlights of Flink technology are:

- Batch and streaming processing in one
- Highly flexible streaming window for continuous stream model
- Integrated with many open source data processing ecosystems

4.3.3 Kafka

Kafka is a publish / subscribe messaging system which integrates data streams. It is fast, scalable and reliable messaging system which is key component to Hadoop stack technology for supporting real-time analytics or modernisation of IoT

Kafka can handle many terabytes of data without incurring. It is different from a traditional messaging system because it is designed to scale very well. It is also designed to deliver three main advantages over AMQP, JMS

Kafka's highlights are:

- Highly Reliable: Kafka replicates data and it can support multiple subscribers. In the event of failure, it automatically balances consumers in the event of failure which is very much reliable in comparison to similar messaging services
- Superbly Scalable: Kafka, which is a distributed system, is able to scale quickly and easily without incurring any downtime
- High Performance: For both publishing and subscribing, Kafka delivers high throughput. It can offer constant levels of performance even when it deals with many terabytes of stored messages
- Durable: Kafka provides intra-cluster replication by keeping messages on the disks which make it durable messaging system

4.3.4 Kinesis

Kinesis is similar to Kafka, although Kafka is free and it should be made an enterprise-class solution for organisations. Amazon's Kinesis is an out-of-the box data streaming solution. It comprises shards from Kafka partitions.

One of the problems Kinesis solves is real-time aggregation of data which is followed by loading the aggregate data into a data warehouse, with putting it into Kinesis streams. This technique durability and elasticity. Kinesis is a managed, scalable, cloud-based service which allows real-time processing of large data streams.

Kinesis highlights are:

- Real-time data processing
- Ingestion of real-time data into data stores like S3, Elasticsearch or Redshift for batch analytics
- Kinesis Analytics analyse data in real-time

4.3.5 Samza

Samza in another Apache stream processing framework, which is tightly associated with Kafka. It is designed to take advantage of Kafka's unique architecture and ensures fault tolerance, buffering and state storage.

Samza uses YARN for resource negotiations, and as a result a Hadoop cluster is necessary with Samza. It can store state using fault-tolerant checkpointing system, implemented as a local key-value store. It helps Samza to offer delivery security, although there is not reliability and accuracy recovery in the event of failure. It supports high-level abstractions making easier to work. On the other hand, it only supports JVM language.

The Samza highlights are:

- Simple API: Samza provides a very simple callback-based "process message" API as compared to MapReduce
- Managed state: Samza manages snapshotting and restoration of stream processor's state
- Fault tolerance: Samza works with YARN whenever a machine in the cluster fails in order to transparently migrate your tasks to another machine
- Scalability: Samza is partitioned and distributed at all levels

4.3.6 Rx

Rx is a library that composes event-based and asynchronous programs. Use of data sequence is extensive in the form of data streams from files, web services, system notification and user defined data. The major characteristics of Rx are (Microsoft, 2018):

- Data push: data is retrieved from sources using various GET methods, having control over how data is retrieved in the application
- Data pull: data push is similar to joining a book club and registering in particular book genres. The same concept applies in data push model, where the program registers to a stream sequence and any changes are managed at the source.
- .NET development: allows the programs to be developed in desktop applications. It can also be released for Silverlight, Windows Phone 7 and JavaScript

4.3.7 StreamInsight

"Microsoft StreamInsight™ is a powerful platform that you can use to develop and deploy complex event processing (CEP) applications. Its high-throughput stream processing architecture and the Microsoft .NET Framework-based development platform enable you to quickly implement robust and highly efficient event processing applications. Event stream sources typically include data from manufacturing applications, financial trading applications, Web analytics, and operational analytics." (Microsoft, 2018). The major characteristics of StreamInsight are:

- Monitor your data from multiple sources for meaningful patterns, trends, exceptions, and opportunities.
- Analyse and correlate data incrementally while the data is in-flight -- that is, without first storing it-- yielding very low latency. Aggregate seemingly unrelated events from multiple sources and perform highly complex analyses over time.
- Manage your business by performing low-latency analytics on the events and triggering response actions that are defined on your business key performance indicators (KPIs).
- Respond quickly to areas of opportunity or threat by incorporating your KPI definitions into the logic of the CEP application, thereby improving operational efficiency and your ability to respond quickly to business opportunities.
- Mine events for new business KPIs.
- Move toward a predictive business model by mining historical data to continuously refine and improve your KPI definitions.

Both Rx and StreamInsight technologies are closer to the data streaming process required for the COMPOSITION DSS and they are deployed in it. Their techniques and characteristics help DSS, which is also a .NET application, to be integrated with the rest of the COMPOSITION components.

5 Decision Support System – DSS

Based on the analysis and the DSS types discussed in chapter 4.2.1, it was decided that a combination of a model-driven and data-driven DSS is developed for the COMPOSITION project. The Decision Support System can handle incidents and transformed data coming from all the previous components to create KPIs and visualise them. Furthermore, the most important functionality of the DSS is to use the incoming data to create rules in the decision engine which will help decision-makers to decide during mainly maintenance procedures.

Decision Support System should be designed to accommodate the needs in a manufacturing environment. DSS integrates Digital Factory Models with sensor data, and other information and knowledge about the products, manufacturing, planning, simulation, communication and controls at all levels of planning and manufacturing. Raw data form sensors on factories are acquired and transformed according to the Digital Factory Model Schema. Processed data is accessed by the DSS through the DFM API. Accessing and processing the transformed data is easier and DSS implementation can be applied without the complicated need of transforming data in a suitable format.

Designing a DSS data and algorithm specification should be considered. Data specifications derive form DFM API. The algorithms suitable to be applied on a DSS in a manufacturing environment are both data mining algorithms to retrieve suitable data from a repository and decision-making algorithms for the decision-making process. The most used data mining algorithms are classifications trees, generic algorithms, support vector machine, Naïve Bayes. Various combinations and modifications of the above algorithms are considered to designing the data mining part of the DSS. Additionally, Nondeterministic Finite-state Automata (NFA) could be used in the decision-making process. The automata provide the possibility to be expanded during the process. Originally, DSS can implement a rule engine based on Finite State Machines, where the rules applied are defined based on the use cases. The initial rules can be used during a period to train data and then NFAs and non-deterministic algorithms can be implemented for the decision-making process with the collaboration of the Deep Learning Toolkit.The results of the analysis made by the Deep Learning Toolkit is fed to the Learning Agent, which is responsible for adapting the results and then refeed the DLT with outcomes from learning processes. The final output is given to the DSS, via the LA. Both the analysis of the Deep Learning Toolkit and the Learning Agent are briefly described in chapters 6.3 and 6.4.

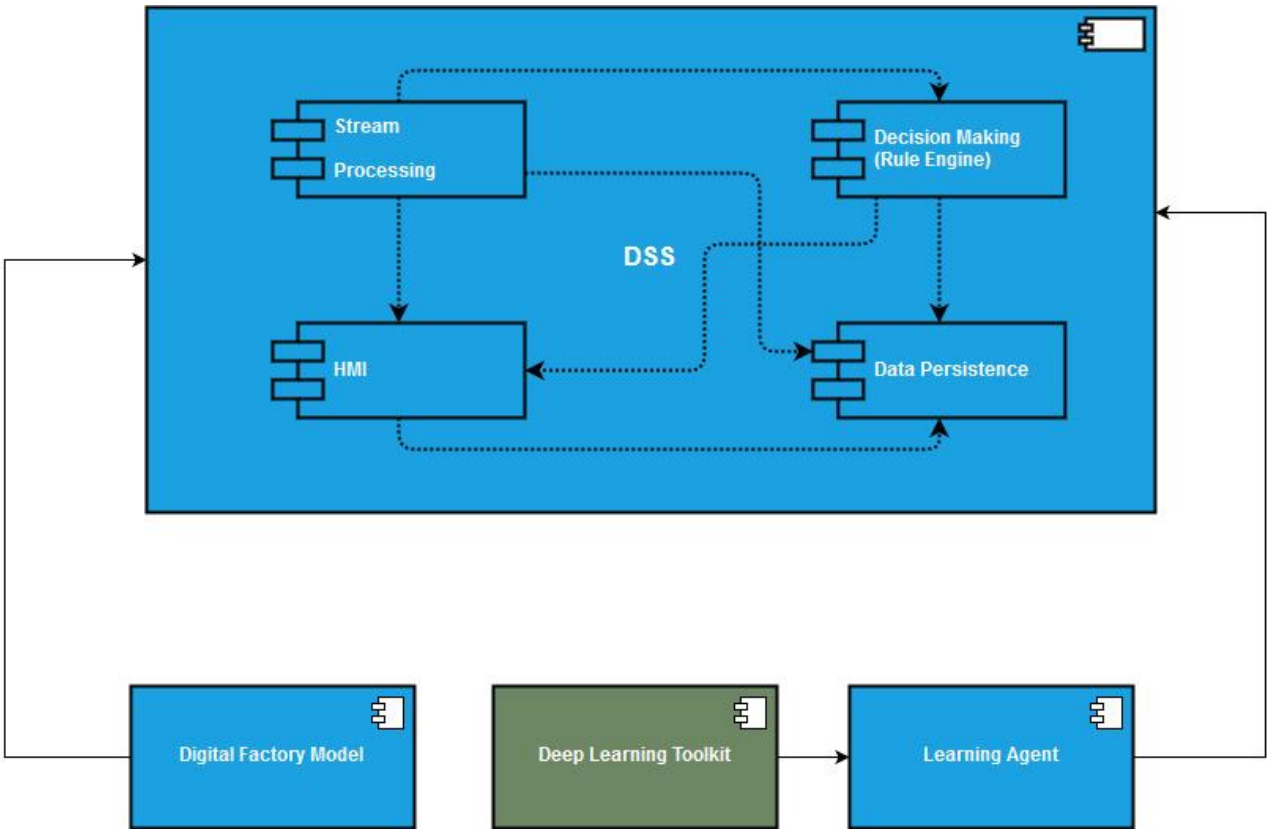

Figure 2: Decision Support System Architecture

DSS architecture relates to the use of common data for all systems. Most manufacturing shop floors use the same sensors and they provide the same data. The differences are spotted to the levels of integration at the

shop floors with the virtual factory models. Digital Factory Models create a virtual model of the factory, the DSS can work with. The process of making the virtual models of the factories follows at the section below.

Briefly, the sub-components of decision support system are:

- ***HMI*** -This component is responsible for the interaction with the user.
- ***Decision Making (Rule Engine)*** - The rule engine of decision support system
- ***Stream processing*** - This component processes data of all external systems and extracts the information necessary for the decision support system
- ***Data persistence*** - This sub-component handles the storage of information for decision support internal use.

The main components from which DSS receives data are:

- ***Simulation and Forecasting toolkit***
- ***Deep Learning Toolkit***
- ***Learning Agent***

Regardless the internal structure the decision support system, it should be deployed as one using a composed docker image.

5.1 Modelling and Persistence

DSS needs a suitable model with persistent data and key in order suitable rules to be defined. DSS model components are: the model itself, its description and its type. The DSS rules are based on these components and persistence in a key factor in applying rules. Persistent data means the result of an action remains, even if the cause of the result changes. Persistence is very important in maintenance procedures, because if a fault occurs during operations, then the cause may not exist after a while (e.g. high voltage cause a burnt resistor), but the result remains until it is fixed, following a maintenance procedure. Depending on maintenance procedures, the most accurate description of the models and their types are shown in the table below:

Table 4: DSS Model Components

Model	Description	Type
Asset	Models the assets on a shop floor according to the DFM schema	Persistent
Failure Mode	Indicates the failure that should be detected by the DSS and it is added on the DFM schema	Persistent
Time Frame	Indicated the time period for which the failure mode on a certain asset has a specific value	Persistent
Value	The actual value of the prediction that the DSS can use in the rule creation	Persistent
Task	Models maintenance tasks on a shop floor to the DFM schema	Persistent
Actor	The actors are modelled based on the DFM provided schema	Persistent

The DSS model are based on: Asset, Failure Mode, Time Frame, Value, Task and Actor based on the main values needed for maintenance procedures. Assets represent the equipment on shop floors that are monitored for faults and malfunctions, failure modes indicate the failures on specific assets during maintenance processes, time frame shows the time period for which the prediction is valid, value shows the prediction of failure that the DSS receives and uses in the rule engine, tasks are the maintenance tasks needed for repairing

assets on shop floors and completing maintenance tasks. Finally, actors are the persons to complete the maintenance tasks, such as supervisors, workers and managers on the shop floor.

All models are persistent type because they can cause a result and then return to their nominal operation status. This is essential for the DSS, because the system knowns the cause and its effects and can produce rules according to these distinctions.

DSS models are consistent with the DFM and its schema, because the communication between these two components should provide all necessary information in a common format that can be used from both. All information and data can be stored in the DFM database and can be retrieved by the DSS using the most suitable API calls. As a result, DSS only processes data and does not store any of it internally. This fact leads to a data independent solution, which can be used in many different applications with minor modifications that could be applied to its models.

5.2 Data Stream and Processing

*"**Stream processing is a computer programming process that allows applications to easily exploit limited parallel programming in real time. Stream processing simplifies parallel programming by imposing restrictions to its use. Given a stream of data to process, a series of operations is applied to each element of the stream. The operations are called kernel functions and are usually pipelined. They allow Direct Memory Access (DMA) for data processing and expose data dependencies and correlations. Stream processing is the processing of data in motion, or in other words, computing on data directly as it is produced or received**"* (Artisans, 2018).

Most of data are born as continuous streams: sensor events, user activity on a website, financial trades, and so on – all these data are created as a series of events over time. COMPOSITION UC-BSL-2 Predictive Maintenance requires the control of machines on BSL shop floor implementing a sensor network for monitoring and gathering data to be sent to other COMPOSITION components. Acoustic sensors are implemented on Boston Scientific Ltd (BSL) shop floor, to measure the noise of RHYTMIA machines. Data continuously comes from the acoustic sensors in the form of .wav files containing noise in the machine. Also, on the BSL shop floor, BRADY oven machines are monitored and provide continuous data in COMPOSITION system. BRADY oven machines use Wireless Sensor Network (WSN) to send data in the system and exploit the possibilities provided by the DSS. WSN continuously sends data, conforming to an Open GeoSpacial Sensor Thing (OGC ST) model. Finally, COMPOSITION UC–KLE-1 Maintenance Decision Support System, applied in KLEEMANN shop floor, requires vibrometer sensors that measure the acceleration of the BOSSI polishing machine in order to monitor the machine operation. Also, data streaming from the SFT providing the prediction of machine failure for the next 24 hours is required by DSS. Both data sources are used for visualisation and rule creation in the DSS and data streaming is necessary in both use cases. DSS uses RX/StreamInsight along with data at rest and stream processing infrastructure in order to pull or push data from repositories.

Before stream processing, this data was often stored in a database, a file system, or other forms of mass storage. Applications would query the data or compute over the data as needed. Data storage is necessary for further exploitation of data. Knowledge extraction in the form of KPIs can provide decision-makers with new and more effective solutions to problems previously encountered. Also, machine learning techniques require

the existence of historical data to exploit it as training dataset.

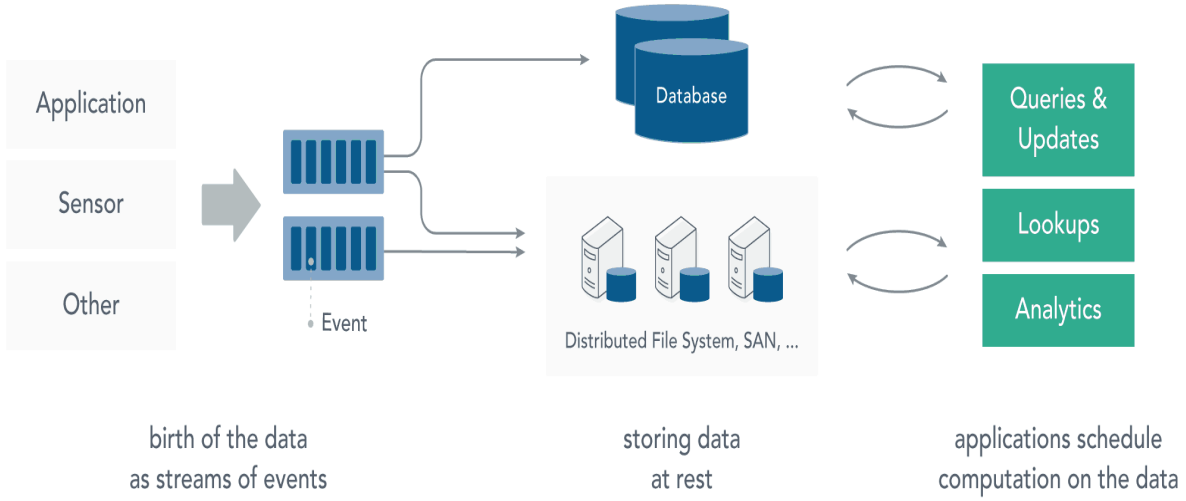

Figure 3: Data at Rest architecture (Artisans, 2018)

Figure 3 represents an overview of a data at rest architecture, common before modern stream processing. Data at rest usually refers to data store in persistent storage such as disk, database etc. and it is yet used for queries and transaction, contrary to data in use which is data currently processed in a CPU or a RAM.

Stream Processing turns this paradigm around: The application logic, analytics, and queries exist continuously, and data flows through them continuously. Upon receiving an event from the stream, a stream processing application reacts to that event: it may trigger an action, update an aggregate or other statistic, or "remember" that event for future reference. Streaming computations can also process multiple data streams jointly, and each computation over the event data stream may produce other event data streams.

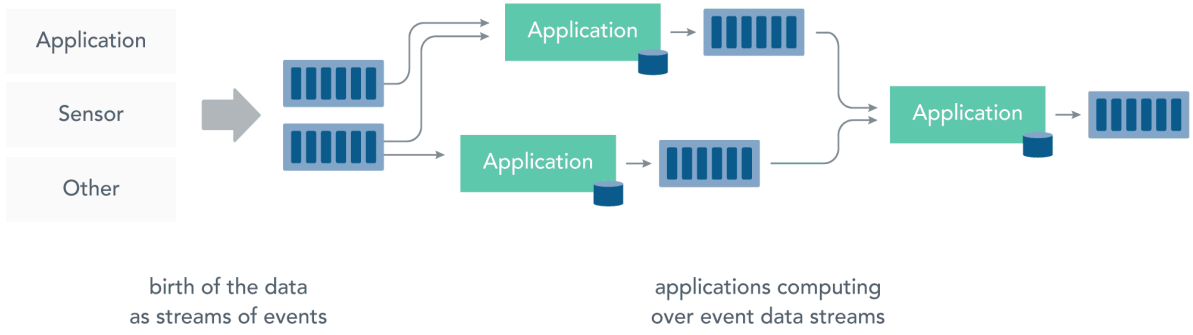

Figure 4: Stream Processing infrastructure (Artisans, 2018)

The systems that receive and send the data streams and execute the application or analytics logic are called stream processors. The basic responsibilities of a stream processor are to ensure that data flows efficiently and the computation scales and is fault tolerant. The stream processing paradigm naturally addresses many challenges that developers of real-time data analytics and event-driven applications face today:

- ***Applications and analytics react to events instantly:*** There's no lag time between "event happens" -> "insight derived" -> "action is taken". Actions and analytics are up-to-date, reflecting the data when it is still fresh, meaningful, and valuable.

- ***Stream processing can handle data volumes that are much larger than other data processing systems:*** The event streams are processed directly, and only a meaningful subset from the data is persisted.
- ***Stream processing naturally and easily models the continuous and timely nature of most data:*** This contrasts with scheduled (batch) queries and analytics on static/resting data. Incrementally computing updates, rather than periodic re-computation of all data fits naturally with the stream processing model.
- ***Stream processing decentralizes and decouples the infrastructure:*** The streaming paradigm reduces the need for large and expensive shared databases. Instead, each stream processing application maintains its own data and state, which is made simple by the stream processing framework. In this way, a stream processing application fits naturally in a microservices architecture.

5.2.1 Stateful Stream Processing

Stateful stream processing is a subset of stream processing in which the computation maintains contextual state. This state is used to store information derived from the previously-seen events. Virtually all non-trivial stream processing applications require stateful stream processing:

- A fraud prevention application would keep the last transactions for each credit card in the state. Each new transaction is compared to those in the state, labelled as valid or fraudulent, and the state is updated with that transaction.
- An online recommender application would keep parameters that describe the user's preferences. Each product interaction generates an event that updates these parameters.
- A microservice that handles a song playlist or e-commerce shopping cart receives events for each user interaction with songs or products. The state keeps the list of all added items.

Figure 5 shows a visualisation of a stateful stream processing.

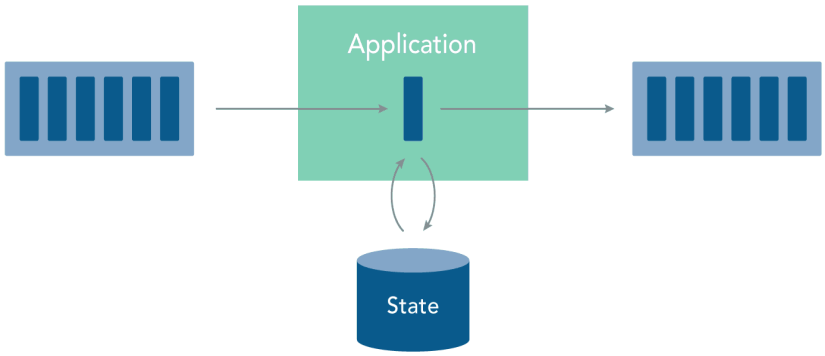

Figure 5: Stateful Stream Processing (Artisans, 2018)

Conceptually, stateful stream processing brings together the database or key/value store tables and the event-driven / reactive application or analytics logic into one tightly-integrated entity. The deep integration between the state and execution of the application / analytics logic results in very high performance, scalability, data consistency, and operational simplicity. Stateful stream processing requires a stream processor that supports state management.

Stream Processing unifies Data Processing, Analytics, and Applications. Both real-time data processing / analytics and event-driven applications are mentioned in previous paragraphs. The question to answer about these two techniques are if they are two different domains with processing and analytics implemented via frameworks like Hadoop or SQL warehouses and applications implemented via application frameworks and databases.

Current approaches to data processing/analytics and event-driven applications have much in common. For analytics to produce results in real-time or near real-time, a system must continuously compute and update results with each record or event. Modern applications and microservices also operate in an event-driven or

"reactive" fashion, meaning their logic and computation is triggered by events (where events are generated, for example, by a user interacting with a website).

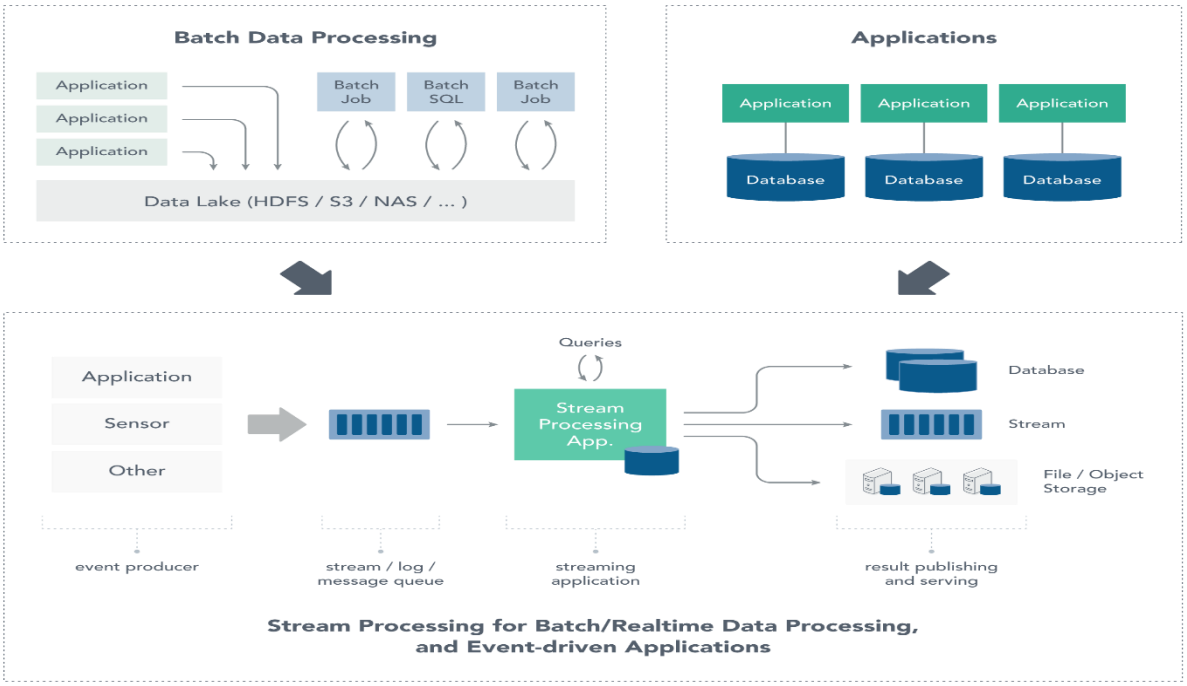

Figure 6: Stream Processing for Real-time data processing and Event - driven applications (Artisans, 2018)

DSS unifies stream processing and batch processing. Stream processing unifies applications and analytics. This simplifies the overall infrastructure, because many systems can be built on a common architecture and allows a developer to build applications that use analytical results to respond to insights in the data–to take action–directly.

For the COMPOSITION project the list of streams:

Table 5: Stream Data Types

Stream	Description	Type
Acoustic data	Data from acoustic sensors	Sensorial data
WSN data	Data from wireless sensors	Sensorial data
Vibrometer data	Acceleration data form vibrometer	Sensorial data
BMS data	Monitoring data from machines	Sensorial data
SFT probabilities	Probabilities of failure for the Bossi machine, analysed in the SFT component	Processed data
LA prediction	Predictions of failure coming from the Learning Agent, based on the training of the data in the Deep Learning Toolkit	Processed data
DFM monitoring data	Binary data based on the monitoring of real-time acoustic data coming from DFM	Processed data

DFM data concerning optimal routes	Data coming from DFM and are given to the DSS notification mechanism to send the optimal route for collecting bins on KLEEMANN shop floor	Processed data

5.2.2 UC-BSL-2 Predictive Maintenance

Data streaming process is used in the COMPOSITION UC (use case). The first UC to apply data streaming process is the BSL-2 Predictive Maintenance UC. In this UC, data is gathered form WSN and EHP (Energy Harvesting) hardware. RHYTMIA oven machine fan parameters are monitored by the system. The measured parameters of Rhytmia oven are: the temperature of the oven, the power in which the machine works in percentage of the nominal power and the log files of the oven for the different events occurring during working times. Afterwards, these parameters are compared to pre-set limits and three different COMPOSITION components analyse the results. BDA (Big Data Analytics) aggregates, annotates, filters and publishes data results for DLT.

DLT provides a prevision for the next possible breakdown with OGC ST and then propagates the possibility to the LA, which applies machine learning techniques for training the system and accepting only the correct predictions. The results are sent to the DSS which applies rules created in the Rule Engine to raise an alarm according to predictions and data. Also, rule application lead to different alerts through the DSS Notification Engine and according to those pre-set parameters, different notifications are sent to actors according to the rules in the Rule Engine. Finally, the rule results and data visualisation are shown in the DSS HMI.

New noise recording sensors, developed by Tyndall, are also implemented for the use case. There are five noise sensors attached to five of the fans of the Rhytmia oven. These sensors record noise samples every five minutes for twenty seconds. The recorded data provides a large amount of .wav files and the noise are monitored easily. In order, to achieve better understanding about the noise levels and their criticality, a transformation is applied on the data. Based on Fourier Transformation the acoustic files are transformed to dB levels. The data providing the noise levels in dBs is sent to the DSS where suitable rules are applied and notification for the users are sent.

The dBs levels data is also sent to the SFT and a monitoring process is created. The monitoring process detects the outliers of the dB and when four consecutive outliers are detected for each fan, it creates and event and sends it to the DSS. The rules in the DSS try to operate with the best practice, so more advanced rules combine the dB levels and the events from the outliers.

Figure 7 shows the Stream Process of the BSL-2 Predictive Maintenance UC. Data flows from the different COMPOSITION components to create the above described format. Each component exploits the data for its own purposes and they provide knowledge outputs. DSS applies rules in the incoming data and creates alarms for different situations and problems occurred.

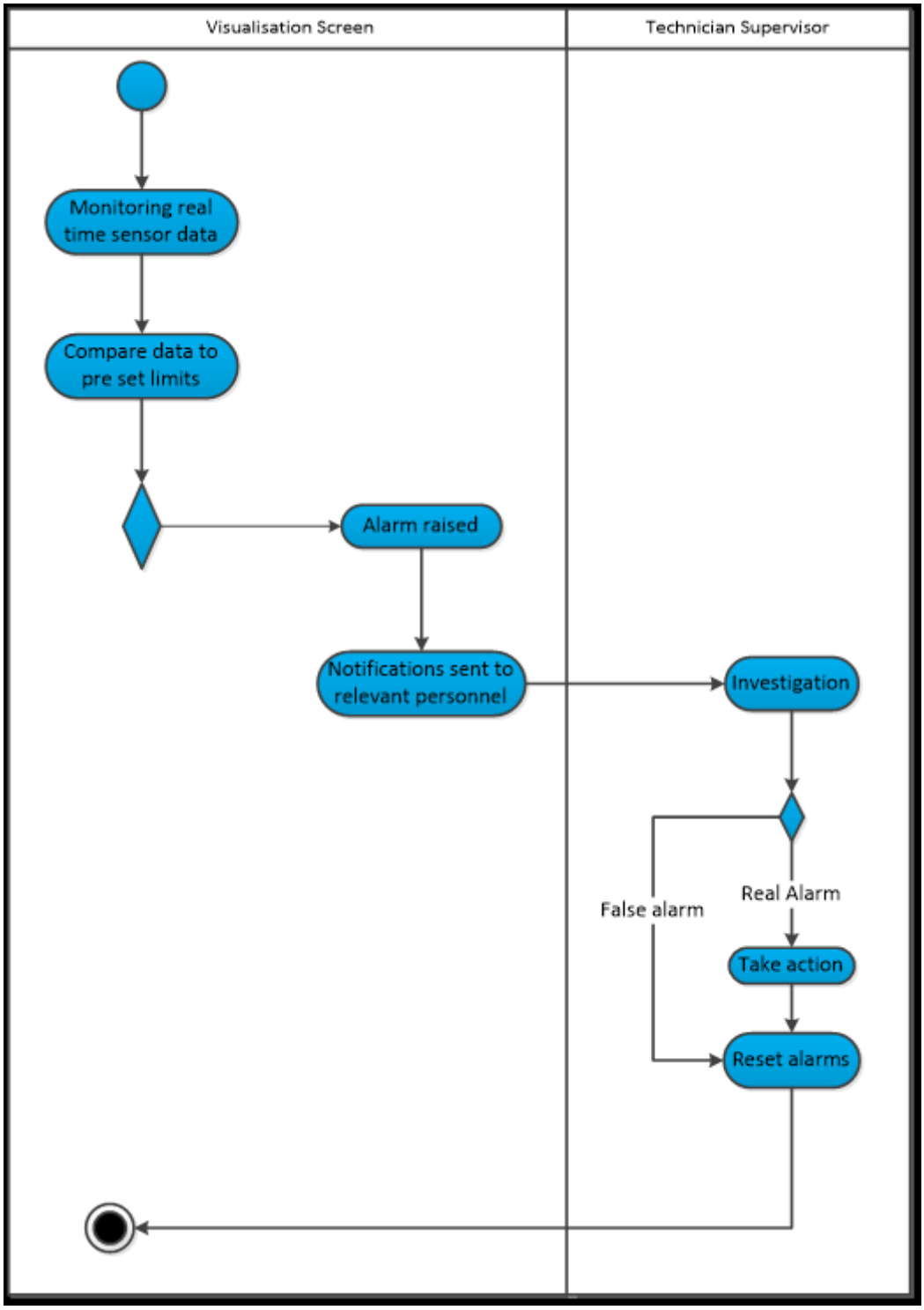

Figure 7: BSL-2 Predictive Maintenance Stream Processing

The component diagram for BSL-2 Predictive Maintenance UC is shown in Figure 8

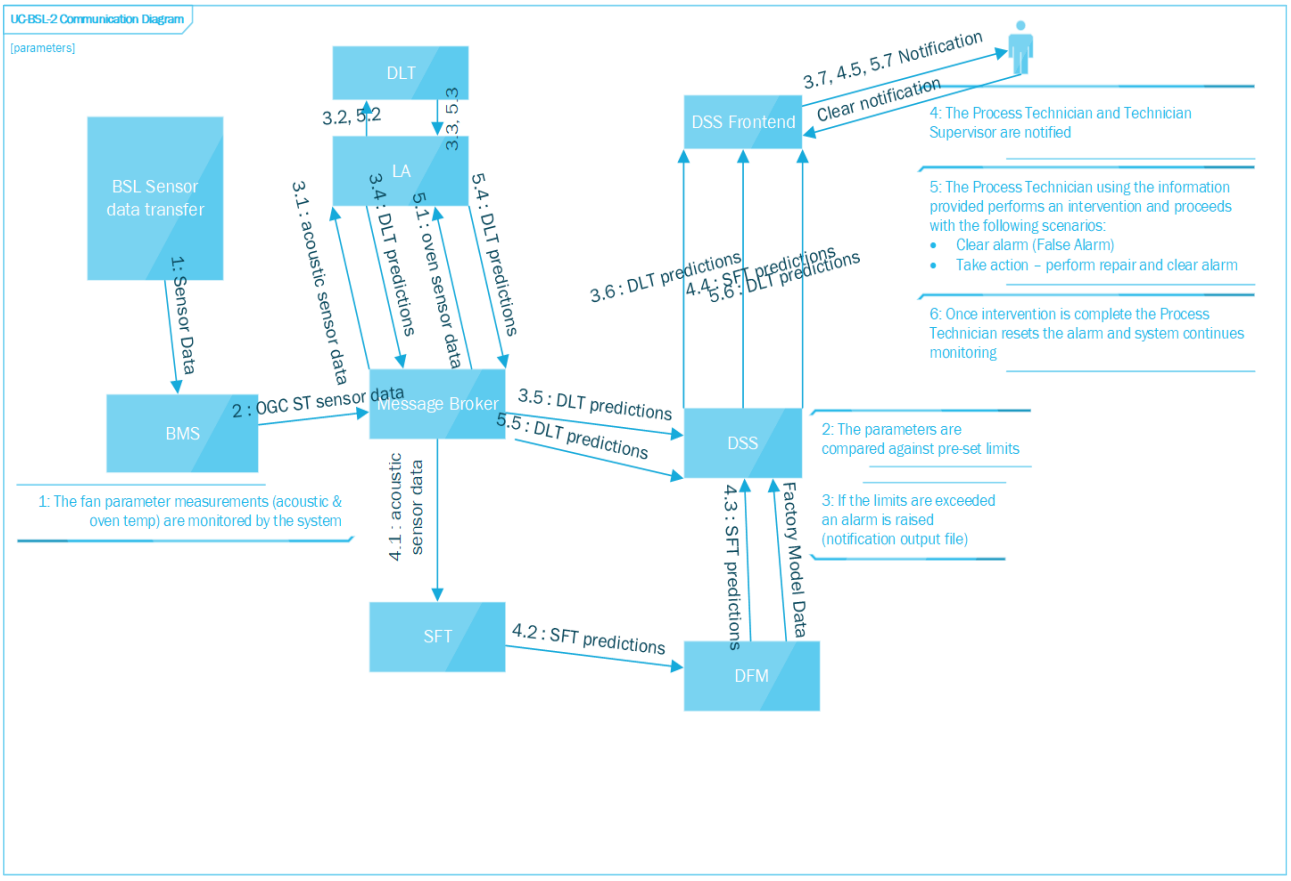

Figure 8: BSL-2 Predictive Maintenance UC Component Diagram

It is shown that all components communicate between them through a message broker. The broker receives the data gathered on BSL shop floor through the BMS and distributes it through MQTT protocol to the rest of the COMPOSITION components.

There are two ways to structure the MQTT topics. They are: system infrastructure and semantics/domain model

- ***System infrastructure:***

  [component]/[Scope]

  Composition/BMS/NXW_51/OGC/1_0/Datastreams/ds_5-1/Observations

- ***Semantics / domain model:***

  [O&M:Procedure]/[DFM:Asset][O&M:ObservedProperty(DFM:Event.Type)]

  Composition/IntraFactory/Prediction/Task_0pf4jcq/Failure

MQTT topics are used for the intra-factory communication, while AMQP topics for the inter-factory communication.

There is also security option in the UC. LinkSmart Service Catalog is used by Security Framework for PK (Port Knocking) management in order to define which ports in the system will be public and available to all, through opening them in the firewall. Also, ACL options for authentication and authorisation are applied to each component. The list provides the authenticated system users and those who are allowed to participate in the US, using its HMIs. SSO options have been discussed for the COMPOSITION project, in order a user to sign-on only once for the different components.

Security is provided to both event and message broker, where each component uses its own credentials to subscribe in the topics that accesses. If the components are able to subscribe with the given credentials, it has all the resources to access the topics and receive the data from them. On the other hand, when a component tries to subscribe to the message broker with faulty credentials, the broker recognises it and access is denied.

5.2.3 UC-KLE-1 Maintenance Decision Support

UC-KLE-1 Maintenance Decision Support follows a similar Stream Processing as UC-BSL-2 Predictive Maintenance. Data is gathered from sensors, such as BOSSI polishing machine vibration sensor data. It is sent to the BMS which distributes it to the BFM. BFM stores the data to be use in SFT and produce prediction results. Also, BMS stores observations made from the incoming data. Finally, SFT propagates the data to DSS for rule creation and knowledge extraction. KPIs will be created for UC-KLE-1 Maintenance Decision Support, as well as data visualisation from the DSS.

There is also data coming from the visual analytics toolkit, which essentially is the transformation of the raw vibration data. The vibration data is measured as acceleration over the three axes, *x,y,z*, and their transformation is the eigenvalues of the accelerations. These eigenvalues show outliers in the data that may be possible failures for the system, if there are continuously recurring.

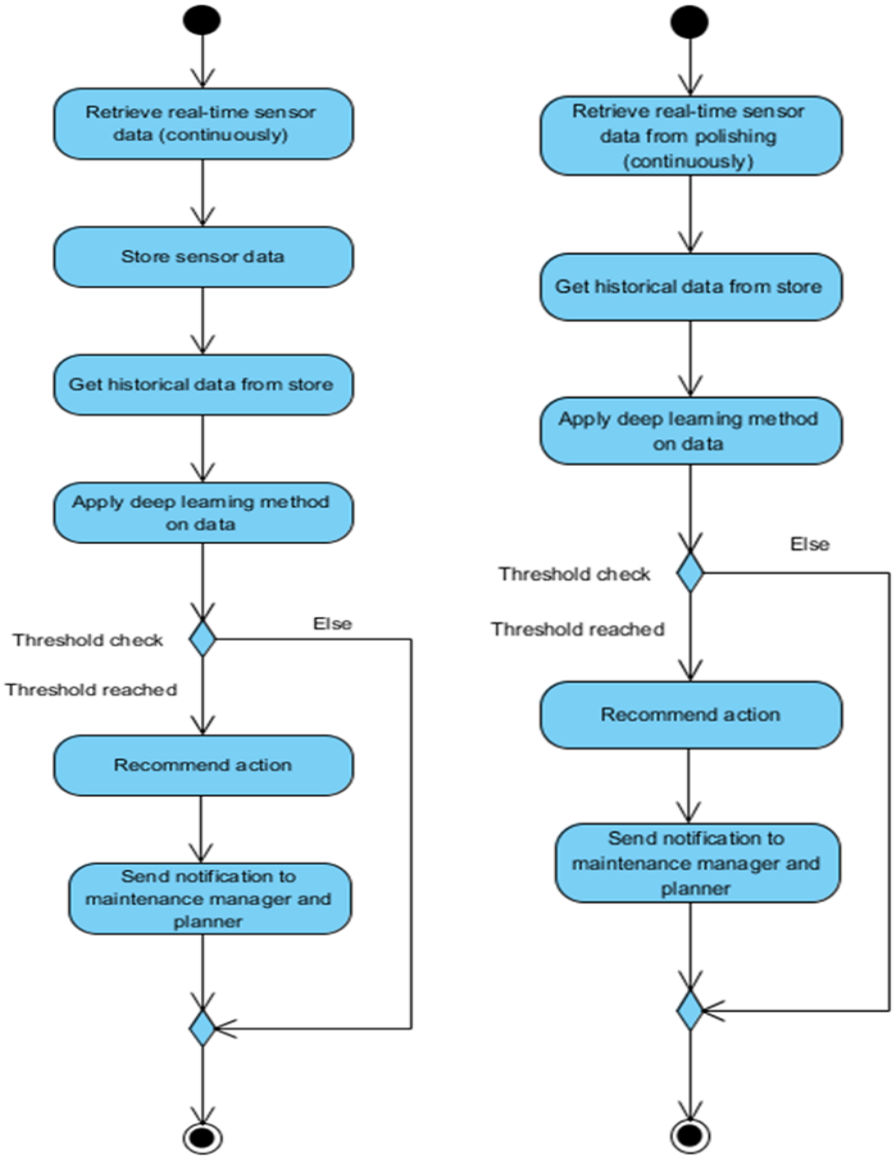

Figure 9: UC-KLE-1 Maintenance Decision Support Stream Processing for both real-time and historical data

Figure 9 shows the Stream Processing for real-time and historical data of KLE-1 Maintenance Decision Support UC. The difference between the two use cases is that a BMS is applied in the UC-KLE-1 for data storage. The BMS is suitable to retrieve the created data and use it as historical data. The historical data can be propagated to all other components to be used a machine learning dataset.

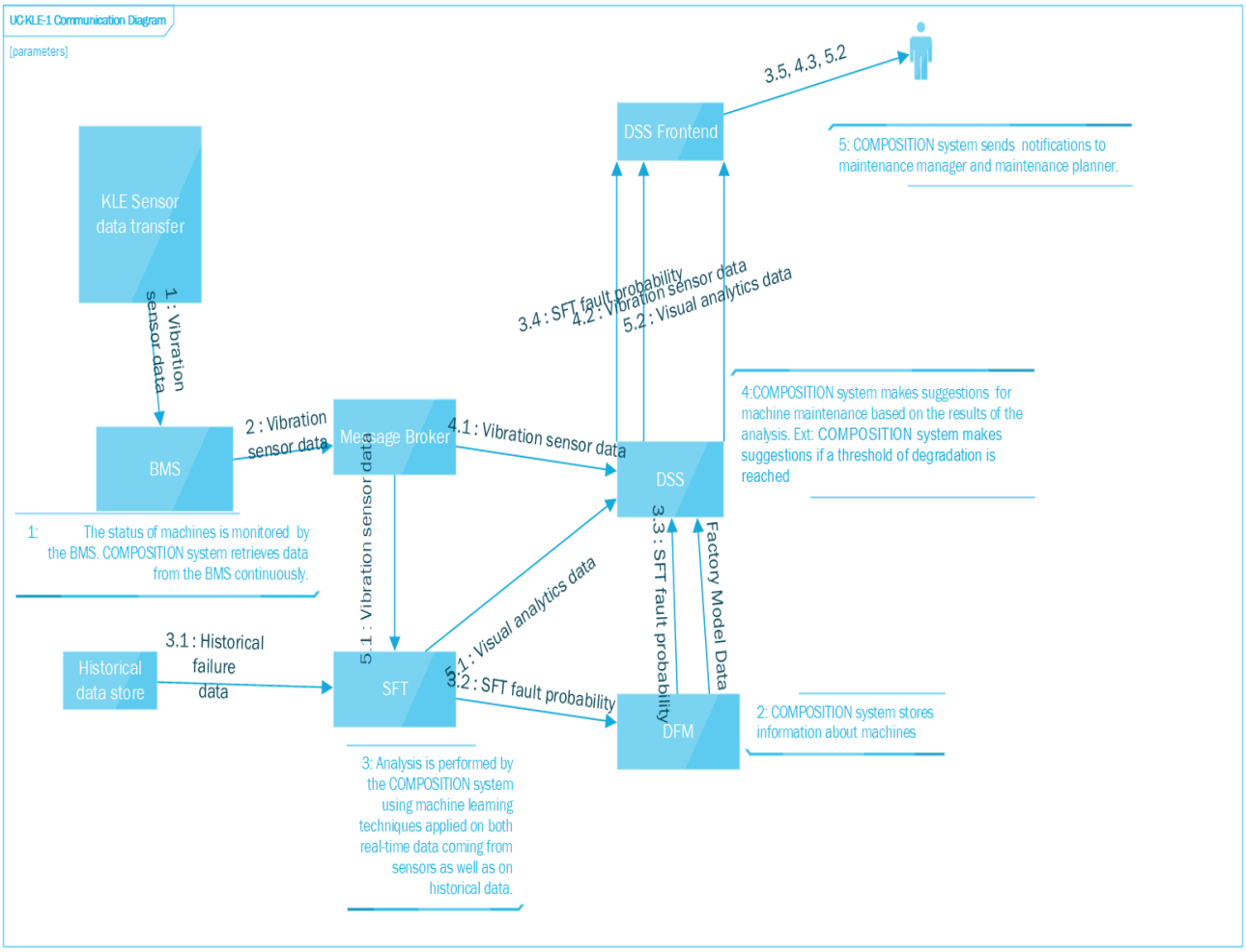

Figure 10: KLE-1 Maintenance Decision Support UC Component Diagram

Figure 10 shows the component diagram of the UC-KLE-1 Maintenance Decision Support. As it shown the data is gathered from the sensor network and is stored in the BMS. Through OGC ST, data is propagated to all other COMPOSITION components. MQTT and AMQP topics are also used in the distribution of information and delivery of data to the components. Information and data are delivered the same way as in UC–BSL-2 Predictive Maintenance for BSL. DSS uses rule engine for decision making process and the extraction of knowledge. KPIs are used for knowledge measurement and data visualisation is another objective of the DSS.

5.3 Decision Making – Rule Engine

The decision making or rule engine sub–component is the core of DSS COMPOSITION component. All information coming to DSS from other components is exploited by the rule engine. The models, established in the Stream Processing sub–component, are used in the rule engine to create the rules. Rules are used to provide accurate suggestions decision–makers. They are graphically represented in the HMI sub–component of DSS in order users to have an easier and more appealing experience while using the COMPOSITION DSS.

The first edition of the rule engine is based on the Finite State Machine (FSM) theory, because an action or an event leads a machine from one state to the other, depending on a transaction based on the action. This preliminary model applies to well-known applications, where there are pre–defined states and transactions. It is also implied that the provided data has been thoroughly examined and studied and its attributes are known.

Each rule engine consists of many Finite State Machines. Each machine corresponds to a rule. To apply a whole rule engine in a shop floor, we create many different state machines to describe all operations. Due to highly complicated environments on shop floors, the state machines can retrieve the same data sources and use them with as different parameters in states for different purposes.

While the amount of data is expanded, the data cannot always be studied well, and the rule engine moves from FSM to Non–Deterministic Finite Automata (NDFA), which cover more options than FSM. The rules are created based on the new data and include possibilities of failure and repetition, as well as unreachable or

forbidden states. To understand operation of the DSS rule engine, the mathematical and algorithmic background is firstly explained and the standard operation of the COMPOSITION DSS rule engine is examined.

DSS is designed, programmed and deployed from the beginning based on finite state machines. It is preferable to create an application from the beginning, without dependencies with previous technologies in the rule engine. It is a .NET application and it is written in C# programming language.

5.3.1 Finite State Machine

"*A finite state machine (sometimes called a finite state automaton) is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. Finite state machines can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics*". (Anderson, 2006; Anon., 2018)

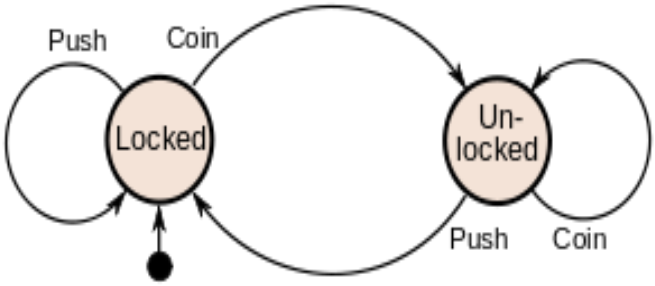

Figure 11: State Diagram for a Turnstile

A system where particular inputs cause changes in state can be represented using finite state machines. This example describes the various states of a turnstile. Inserting a coin into a turnstile will unlock it, and after the turnstile has been pushed, it locks again. Inserting a coin into an unlocked turnstile or pushing against a locked turnstile will not change its state.

There are two types of finite state machines (FSMs): deterministic finite state machines, often called deterministic finite automata, and nondeterministic finite state machines, often called nondeterministic finite automata. There are slight variations in ways that state machines are represented visually, but the ideas behind them stem from the same computational ideas. By definition, deterministic finite automata recognize, or accept, regular languages, and a language is regular if a deterministic finite automaton accepts it. FSMs are usually taught using languages made up of binary strings that follow a pattern. Both regular and non-regular languages can be made of binary strings. An example of a binary string language is: the language of all strings that have a 0 as the first character. In this language, 001, 010, 0, and 01111 are valid strings (along with many others), but strings like 111, 10000, 1, and 11001100 (along with many others) are not in this language.

5.3.2 Formal Definition of Finite State Machines and Non-Deterministic Finite Automaton

A ***deterministic finite automaton (DFA)*** D is a tuple $(Q, \Sigma, \delta, q_0, F)$where (Almeida, et al., 2007):

- 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.

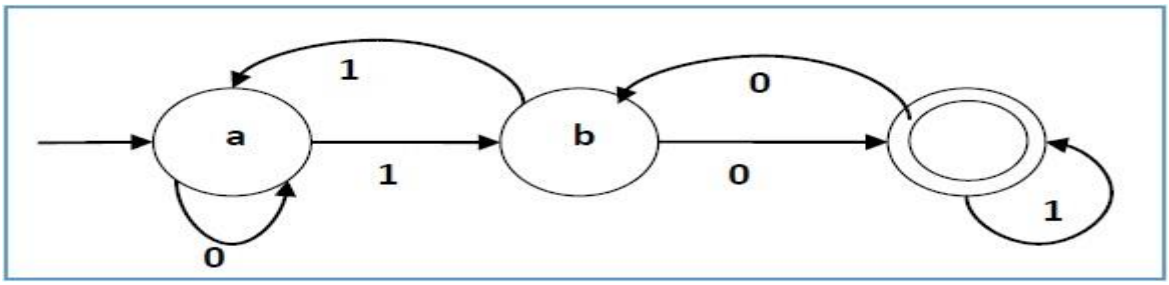

Figure 12: Deterministic Finite Automaton – DFA

A DFA is represented by digraphs called state diagram (Anon., 2018).

- 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 6

Table 6: 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 a 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 $O(2^{|Q|})$.

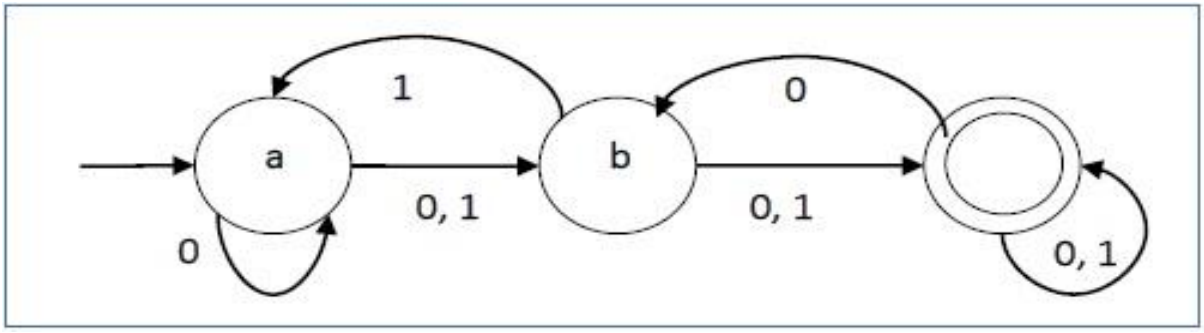

Figure 13: Non - Deterministic Finite Automaton

An NDFA is represented by digraphs called state diagram (Anon., 2018).

- 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 7 below:

Table 7: 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 the table below:

Table 8: 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 $\{S | S \in \Sigma^* and\ \delta * (q_0, S) \notin F\}$

5.3.3 COMPOSITION DSS Rule Engine

COMPOSITION's DSS Rule Engine is based on the theory described in 5.3.1 and 5.3.2. Furthermore, the language, states and transition function are modified to accommodate the creation of the rules. States are defined based on the already existing states of the system. Alphabet is the conditions for each state. Each condition can be mathematical expressions, which when they change the state should change also, regular alphanumeric expressions and strings or a combination between all of them. The transitions are defined from the alphabet and they are a subset of it. The transition function for each transition is evaluated as true or false and when the transition is evaluated true, the system moves from the transition's initial state to the transitions final state.

Algorithm steps of rule creation following the principals of FSM are presented in the Table 9 below:

Table 9: FSM Algorithm for DSS Rule Engine

FSM Algorithm for DSS Rule Engine
1. Define the states for the rule engine
2. Define the transitions for the rule engine
3. Define the conditions for each state
4. Define the action should be taken for each state
5. Describe a certain rule for a specific situation
6. Analyse the rule, discovering the states and transitions needed for the rule
7. Define the set of transition which will be used for the rule
8. Set the values of the conditions
9. Set the limits on the conditions, which lead to preference for one of the transitions
10. Apply the rule on sample data to test its application
11. Apply the rule on real data for real result
12. Use the DSS suggestion on shop floor problems
13. Improve rule with constant feeding it with new data
14. Revise rule when the conditions does not apply to the problem any more

A state diagram is created for each rule. State diagrams graphically represent the FSM, and contain initial and final states, transitions for different conditions and each transition's condition. The more complicated the rule, the more complicated the state diagram also is. The initial rules contain a few states and transitions, even though the transitions are more than the states, because there are different ways from transitioning from one state to the other, back and forth. Figure 14 shows an initial state diagram for a rule in the rule engine.

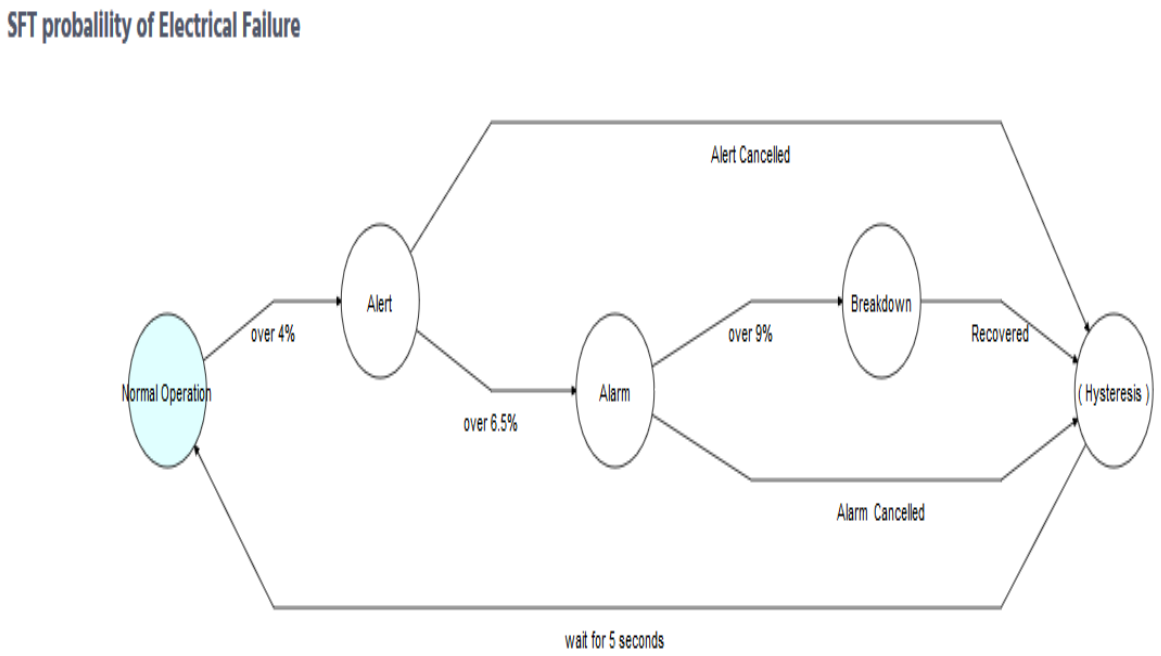

Figure 14: State Diagram for FSM Rule in the Rule Engine

5.3.4 Application of the DSS Rule Engine

The DSS Rule Engine is a web–based application. Usually, decision–makers or higher–level managers on shop floors have access to the application. Its UI is very simplified containing only the necessary graphical elements, but it applies all the logic needed for the rule creation, application and testing.

Decision–makers can create new rules, following the algorithm described above, modify the already existing ones and delete them, whenever the rules do not apply to the data any more. In order a decision–maker to complete a rule, the system allows them to choose conditions and states from drop–down menus, combo boxes, which allow strings and other types of variables. They also can add all necessary states and transitions, defining for each one all the parameters. Figure 15 shows the first screen for the DSS rule engine, where the state diagram of the state machine is created.

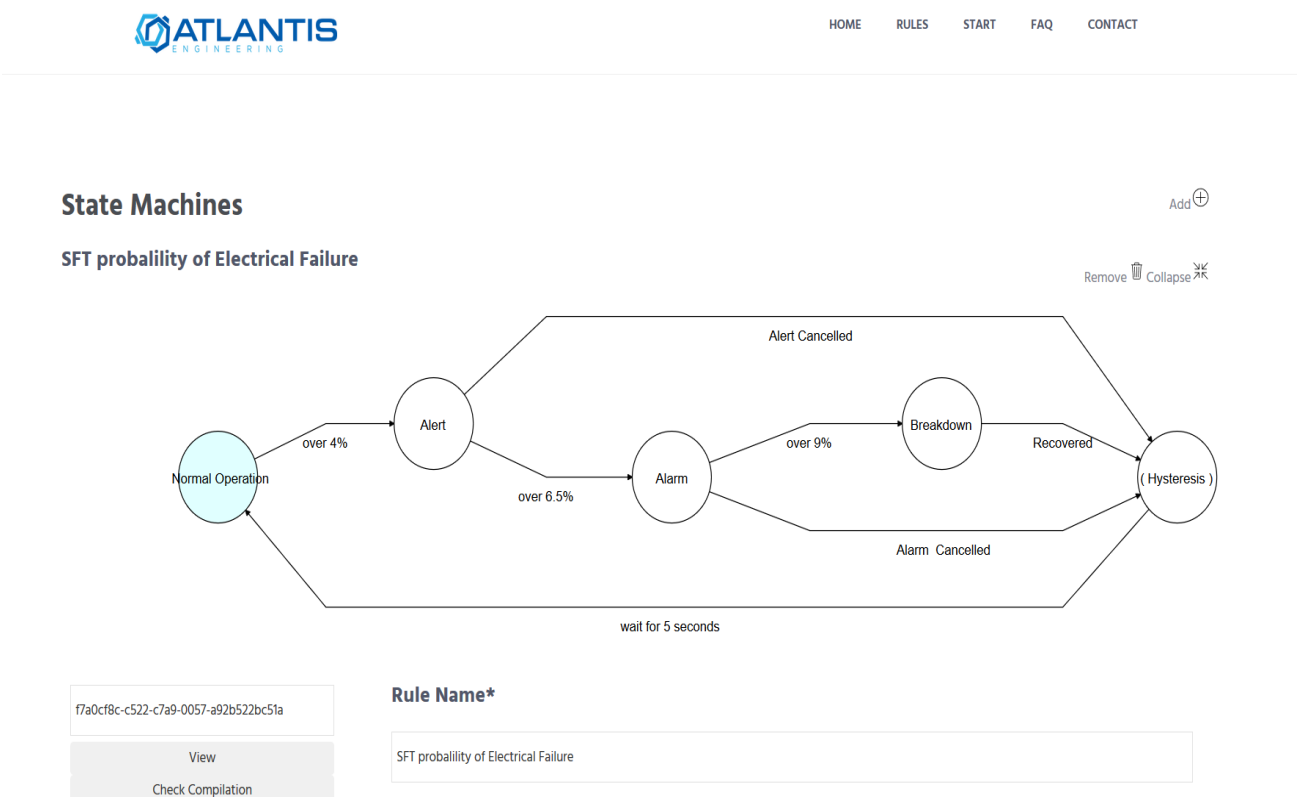

Figure 15: DSS Rule Engine Screen where the state diagram is shown

As shown in Figure 15 above, the rule engine allows to create new rules with the “Add” button, which exists on the upper right corner of the screen. Pressing the “Add” button the dedicated area of a new rule opens. There is a tab menu, in which the user can declare all possible elements of the state machine.

Figure 16 shows the tab menu that created to replace the previous boxes and tables in the DSS UI. The UI was designed and implemented with React components, and it will be discussed in next paragraphs. The first tab is called “States” and in there the states themselves are declared. The second tab is called “Bindings” where the user can define the bindings which apply for each state. The third tab is called “Parameters” and they are the measured or monitored parameters that affect the rule and how it is applied. The forth tab is called “Transitions” and there the rule’s transitions are defined

All the boxes will be further analysed below, and all their functionalities will be thoroughly explained. To save the rule, a decision-maker must press the “Save” button located on the upper left corner of the screen, below the COMPOSITION logo.

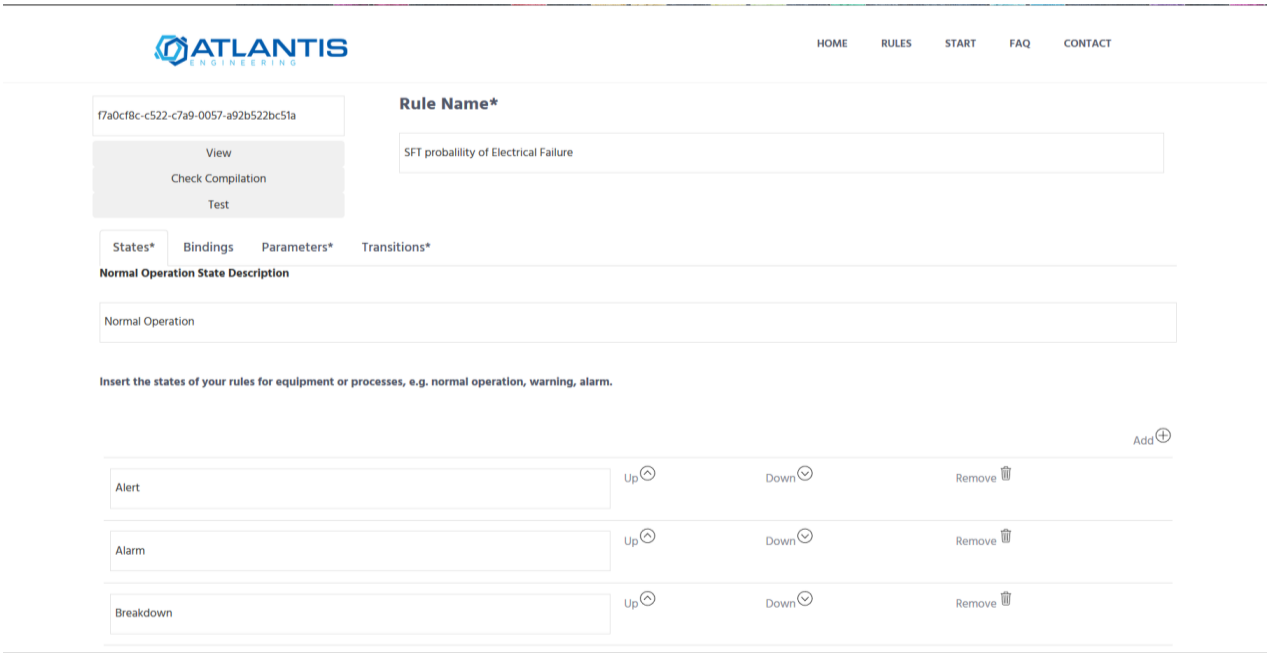

Figure 16: States of DSS Rule Engine

Figure 16 shows how the States are defined in the rule engine. In the displayed rule there are three states called: Alert, Alarm and Breakdown State. Using the “Add” button a decision-maker can add new states for this state machine. Each state has its own “Remove” button and the user can delete the state. The states can be moved up or down, in a way that is convenient for the user, using the “Up” and “Down”. The user can edit each of the states in the textboxes.

Bindings are the maintenance assets defined for each state. They play a significant role in the rule create and implementation because it is their status that changes and changes the status of the state. As it is shown there is already a binding in the rule, called: Bossi. During the rule formation, the decision-maker chooses a second one from the menu, the same way as described for the States. Figure 17 shows the Bindings tab of the Rules screen in the DSS.

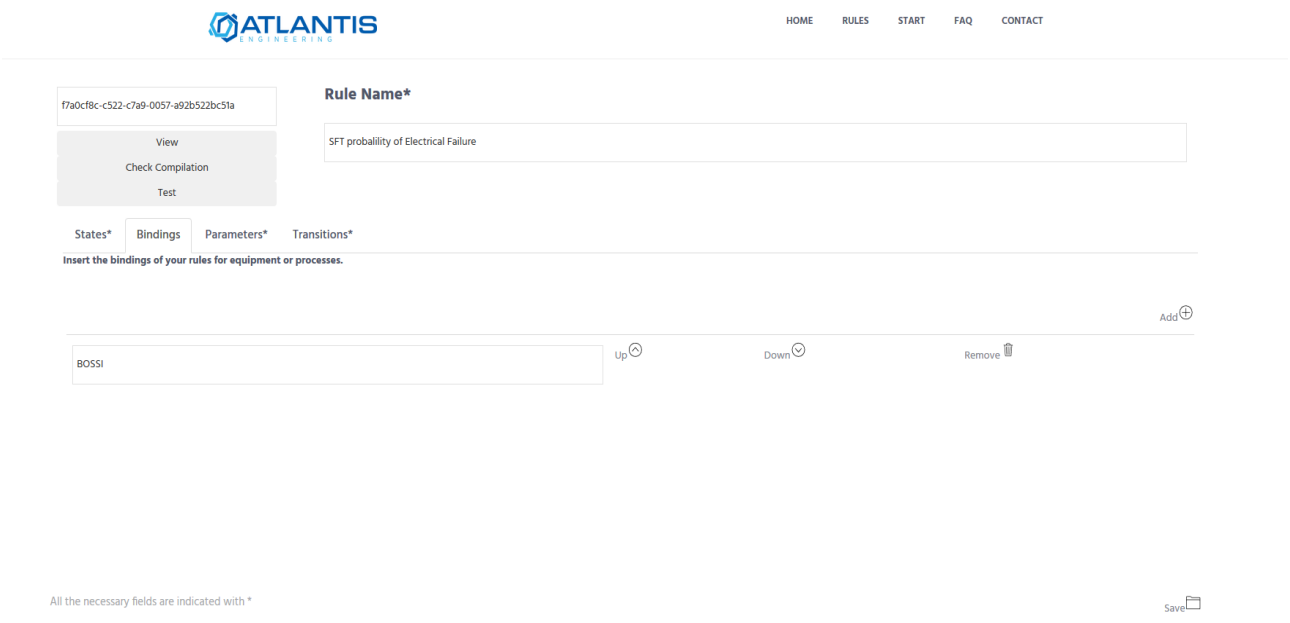

Figure 17: Binding of DSS Rule Engine

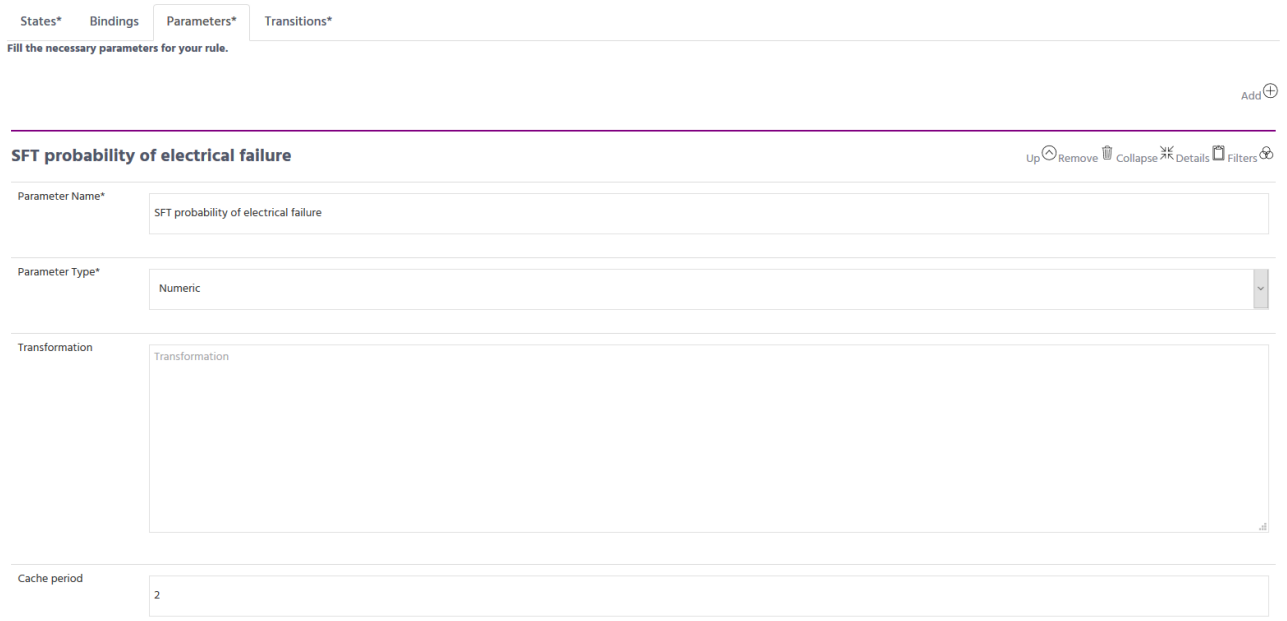

Figure 18: DSS Rule Engine Parameters definition

Figure 18 shows the basic information for the parameter. The field with “Parameter Name” is the name of the parameter defined by the user. there are five choices which can be defined in the parameter. The parameter type can be chosen from a drop-down menu with the following options: Numeric, Alphanumeric, Vector or MessageAcquired. Numeric values are numbers derived from the incoming data, Alphanumeric values are in string format, Vectors are multi-dimensional, while MessageAcquired type is file format, e.g. XML, JSON.

In the Transformation box, an expression can be set to evaluate rule parameters. Usually, the expression is written in C, C# or other high-level programming language. When the evaluated expression is true, the parameter is considered by the rule.

The last element defined in the rule’s parameter settings is the cached data and how long it remains cached in RAM. Decision-makers can set the number of seconds the data can stay in RAM, to extract KPI from it and produce accurate suggestion for the system. The number of seconds is usually large, because it improves the performance of the system. The more information that remains cached, the more accurate the suggestion of the rule.

The menu in the right shows the buttons for “Up”, “Remove”,” Collapse”, “Details”, and “Filters”. The view in Figure 18 is the detailed view. When the user collapses the view, only the name of the parameters is shown. Also, the user has the ability to move the parameters up and down, in suitable way and delete each of them using the “Remove” button.

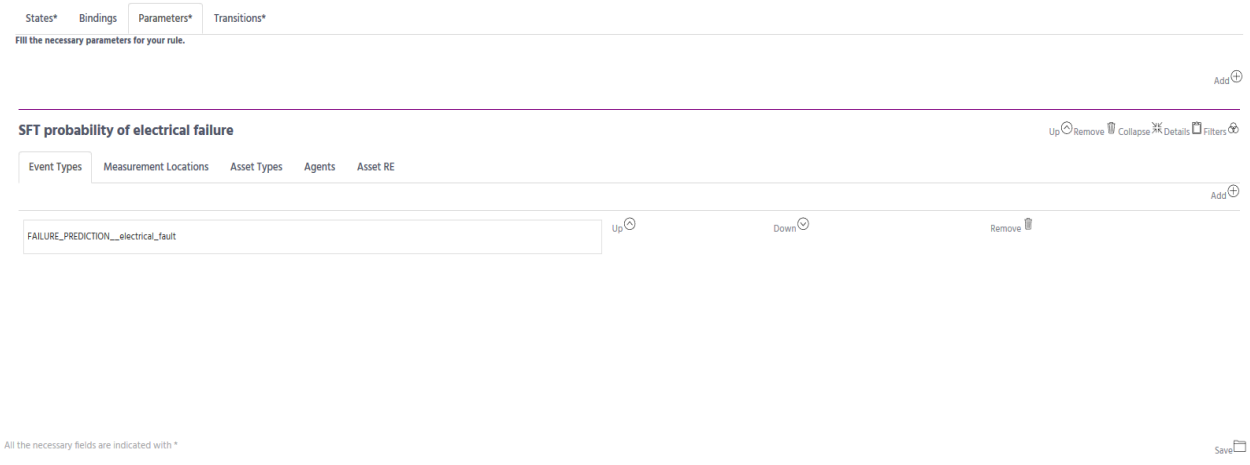

Figure 19: DSS Parameters - Filters view

The filters view is shown in Figure 19. There are five options in the tab menu for the filters. These options are called: Event Types, Measurement Locations, Asset Types, Agents and Asset Re (regular expressions). All of these can be filled by clicking on the respective tab and adding values in the textboxes there.

The Asset Regular Expression is a string value completed by the user and can be applied to anything in correspondence with the rule. Asset Type is usually filed with the assets on the shop floor involved in the rule, Measurement Location is filled with the definition of a measurement point, which is taken into consideration in the rule definition. Event Type defines from what events should the rule be triggered, such as: alerts, alarms, incident etc. Finally, Agent is filled with information contained who is the actor involved with this parameter.

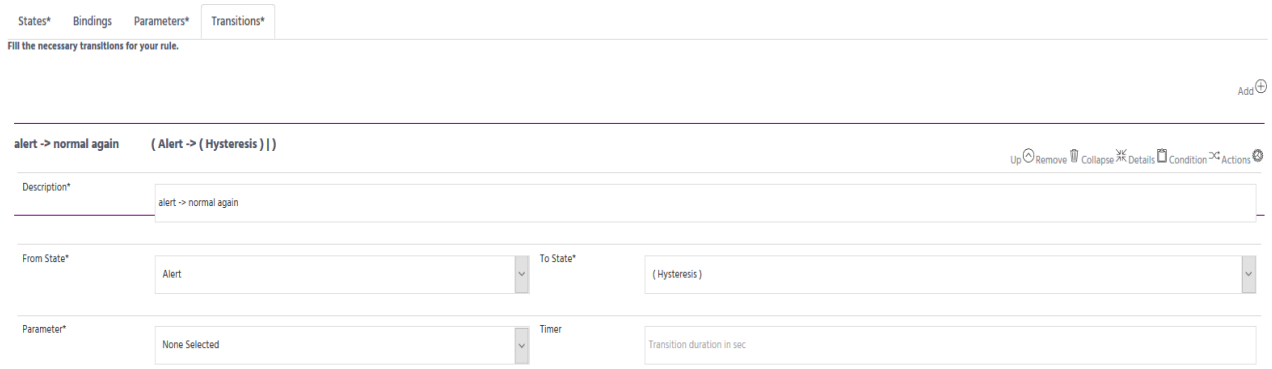

Figure 20: Transitions – Details view

Figure 20 shows a transition called “Alert -> Normal again”. The transition dialogue box contains a box for the name to be filled, three drop-down menus for the starting state, end state of the transition and the parameter used. There is also a box, in which decision-makers fill the second after which the transition is expired, i.e. this transition can only be live and accurate for 20 seconds.

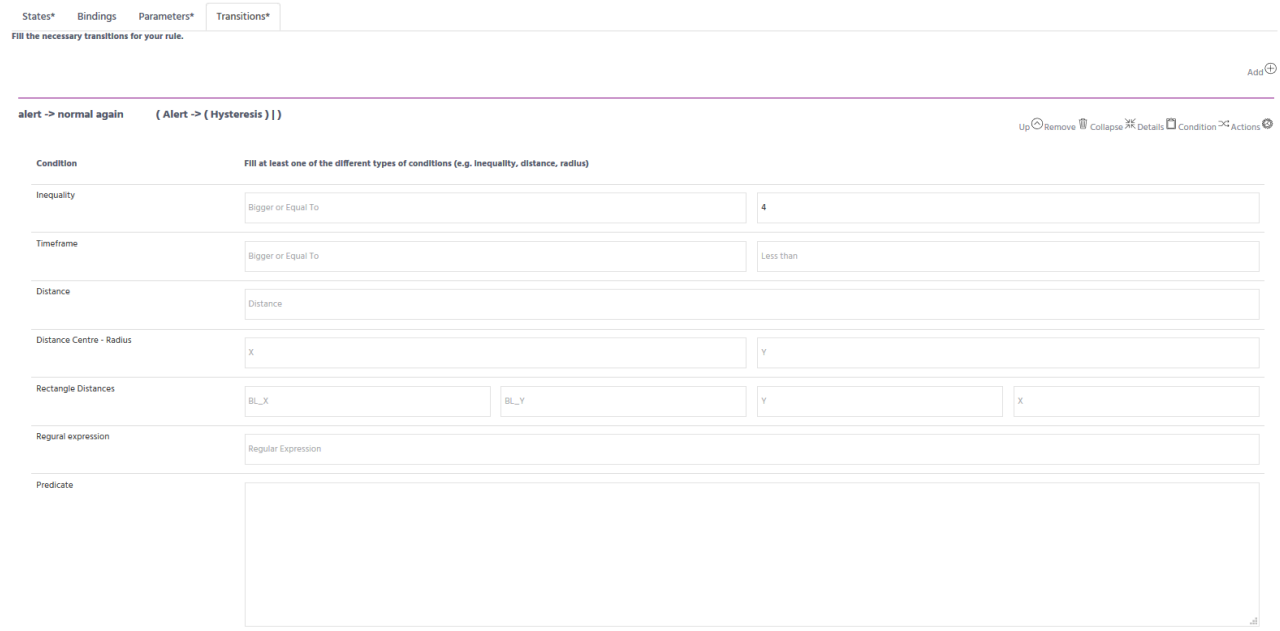

Figure 21: DSS Transitions - Condition View

The Condition view of the DSS rule engine is shown in Figure 21. There are several textboxes, which are optional for the user. In Figure 21, the user has filled the box with a value less than for “Inequality” fields. The rest of the fields can be filled too or be left empty. They refer to time conditions, spatial conditions: such as distances or circles etc. There are also, tow boxes where the user can regularise the condition’s values or uses them as a predicate with mathematical expressions.

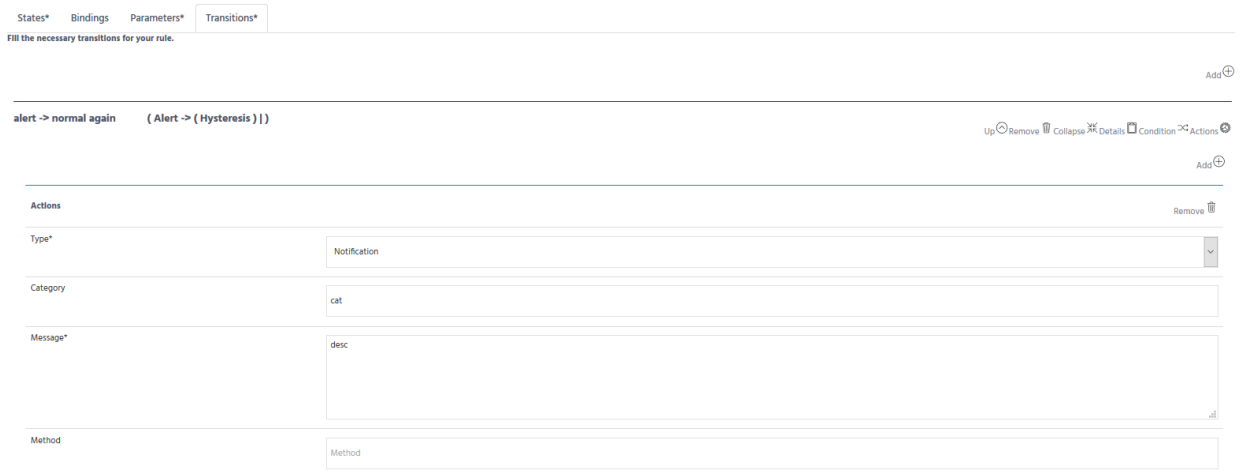

Figure 22: DSS Transition - Actions view

The actions view for each transition can be shown in Figure 22. The users may add actions or remove them. Actions can be: notifications, maintenance tasks, alerts, logs, suggestions or forwards to external systems. The user defines the type of notification, the category, the message and the method of the notification. The method is usually an email or a push notification to a mobile application.

The rule's output is a notification sent to the respective actors in the system containing information and suggestion about the cause that triggered the rule. Notifications can either be push notification, through a Google Services application or emails, SMS. Also, suggestions are shown on the DSS HMI along with graphs for the extracted KPIs and the values that caused the trigger. A log file, in JSON format is also an output of the DSS rule engine. The log file is helpful for data to be stored in a persistent way and be available for further exploitation. An example of a JSON log file, coming from a DSS rule is given below:


```
{
  "stms": [
    {
      "Asset": "CIRCUIT - 0 ROW - 0 REGION - A",
      "State": "INITIAL",
      "LastSTMRun": "2018-04-18T23:29:01.7061936+03:00",
      "Time_In_This_State": 6.2593736
    },
    {
      "Asset": "CIRCUIT - 5 ROW - 2 REGION - B",
      "State": "INITIAL",
      "LastSTMRun": "2018-04-18T23:29:01.938202+03:00",
      "Time_In_This_State": 6.0253714
    },
    {
      "Asset": "CIRCUIT - 5 ROW - 2 REGION - C",
```

```
        "State": "INITIAL",

        "LastSTMRun": "2018-04-18T23:29:01.940202+03:00",

        "Time_In_This_State": 6.0233713999999994

      },
    ],

    "actions_perfomed": []

  }
```


5.4 HMI – Human Machine Interaction

HMI basically integrates the operation of a machine or a process with the feedback to or from the operator.

One aspect is the quality of the graphic user interface and in connection to this, the usability. Another important aspect is the openness of the HMI solution. Is it easy or difficult to exchange essential information with different systems or controllers? Is the application code locked for customization of functions or objects? Will runtime software be able to operate on different hardware platforms? Are design engineers able to use standard .Net objects in their projects? These are issues frequently more discussed in the dialogue between customers and vendors.

The open platform architecture of tomorrow´s HMI solutions will offer a wide range of opportunities for OEMs to enhance the look, the functionality and the connectivity of applications in order to catalyse unique products with substantial integrity. HMI solutions will be less proprietary and offer increased freedom in choice of runtime platform; from compact operator panels to industrial PCs from different manufacturers.

It will be possible to create a scalable master project, which can be applied to different controller brands and panel resolutions with the advantage of only having to maintain one project. Engineers will demand opportunities to use scripting tools, e.g. C# script, to customize the look or functionality of objects. The design tool will offer the possibility to import third party objects and .net controls. Freedom in connectivity and communication is the hallmark of a truly open HMI solution and will include a variety of options ranging from simple real-time exchange of data between controllers up to data storage and OPC communication with other equipment and IT systems

The software in this layer is a set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the Database or the Web. It is this layer, for example, that will wholly contain the MVC architecture of a GUI. The Presenters, Views, and Controllers all belong in here. The models are likely just data structures that are passed from the controllers to the use cases, and then back from the use cases to the presenters and views.

Similarly, data is converted, in this layer, from the form most convenient for entities and use cases, into the form most convenient for whatever persistence framework is being used. i.e. The Database. No code inward of this circle should know anything at all about the database. If the database is a SQL database, then all the SQL should be restricted to this layer, and in particular to the parts of this layer that have to do with the database.

Also, in this layer is any other adapter necessary to convert data from some external form, such as an external service, to the internal form used by the use cases and entities.

Apps are user experiences that have the reach of the web, and are:

- ***Reliable*** - Load instantly and never show the dinosaur, even in uncertain network conditions.
- ***Fast*** - Respond quickly to user interactions with silky smooth animations and no janky scrolling.
- ***Engaging*** - Feel like a natural app on the device, with an immersive user experience.

Also, we should extend beyond screen and inform the users using notifications and other HMI means. The main screen is the shop floor view shown in Figure 23 below has the two main characteristics:

- Summarise the status on the shop flor
- Display the current condition

As it is show for the KLEEMANN shop-floor there are for cards on the screen that show the prediction coming from SFT for the possibilities of normal operation, mechanical failure, electrical failure and hydraulic failure. The predictions are based on historical data from KLEEMANN. SFT gathers the data till the previous day, analyses it and gives a prediction probability for the next 24 hours.

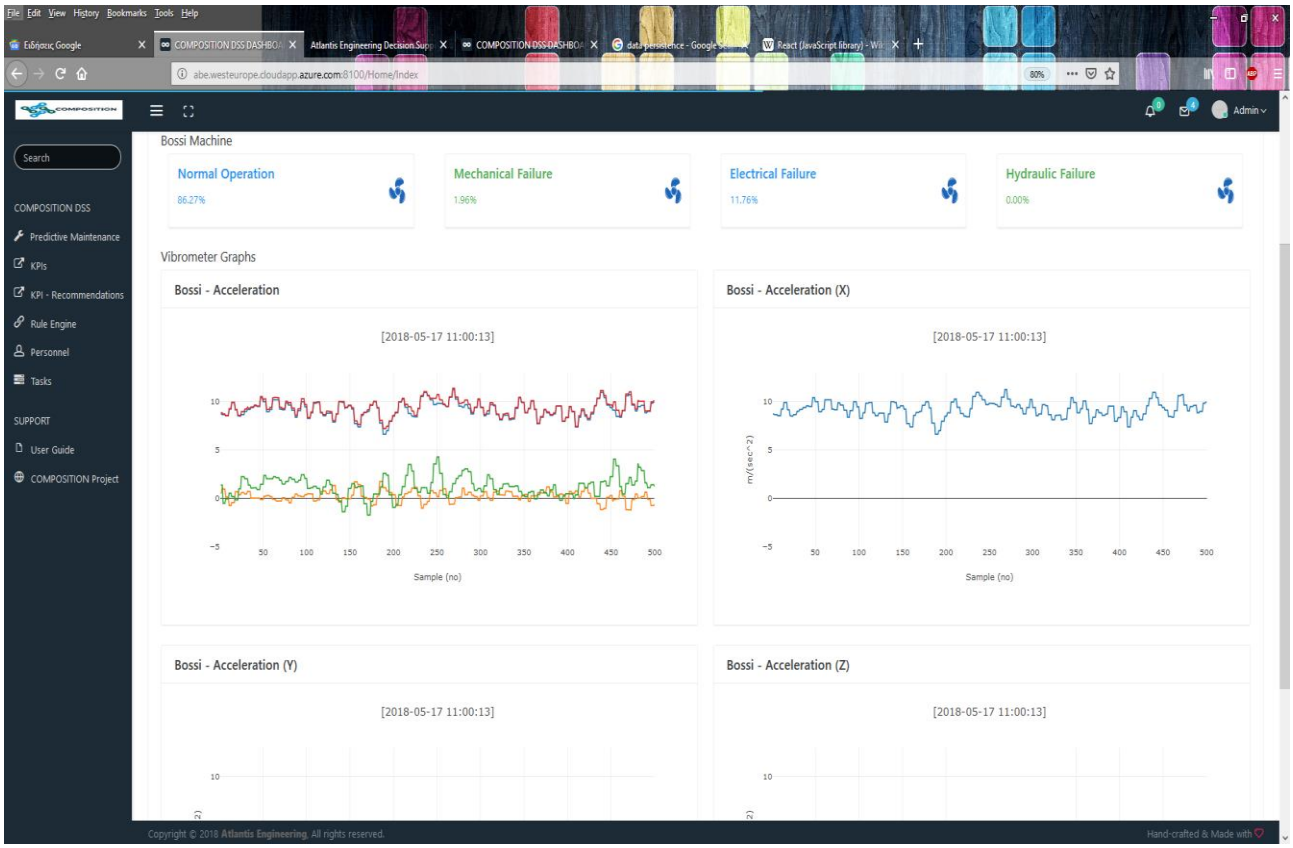

Figure 23: DSS Main Screen for Shop Floor

Figure 24 shows the personnel management screen. The contacts are the personnel on a shop floor. administrators can add a new worker in the system, modify the information concerning the worker or delete a person from the list. Worker's name, email, phone number and position can be set in the system

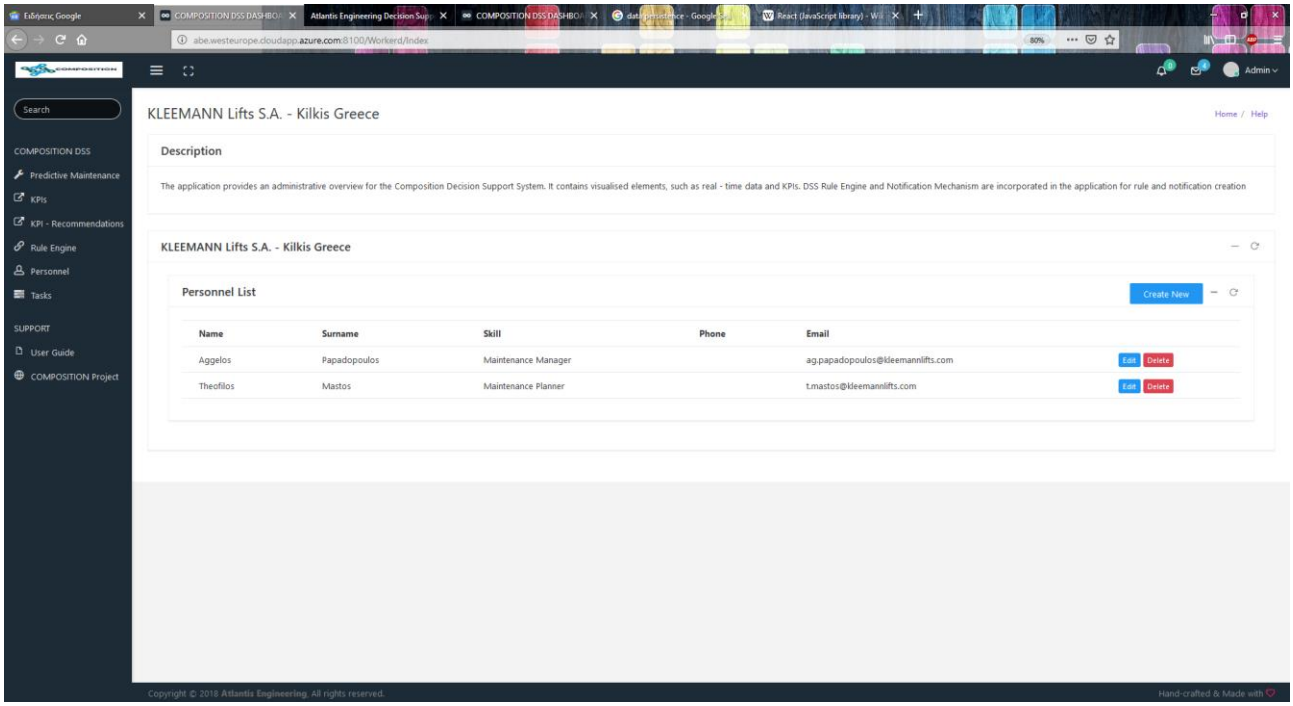

Figure 24: Personnel management screen

Figure 25 shows the DSS task management screen. When a task is inserted in the steps for completed the task can be shown, the title, start date and end data. The aim is to show task in a condensed way in the DSS HMI, for informational purposes, but to not replace the CMSS of the shop floor.

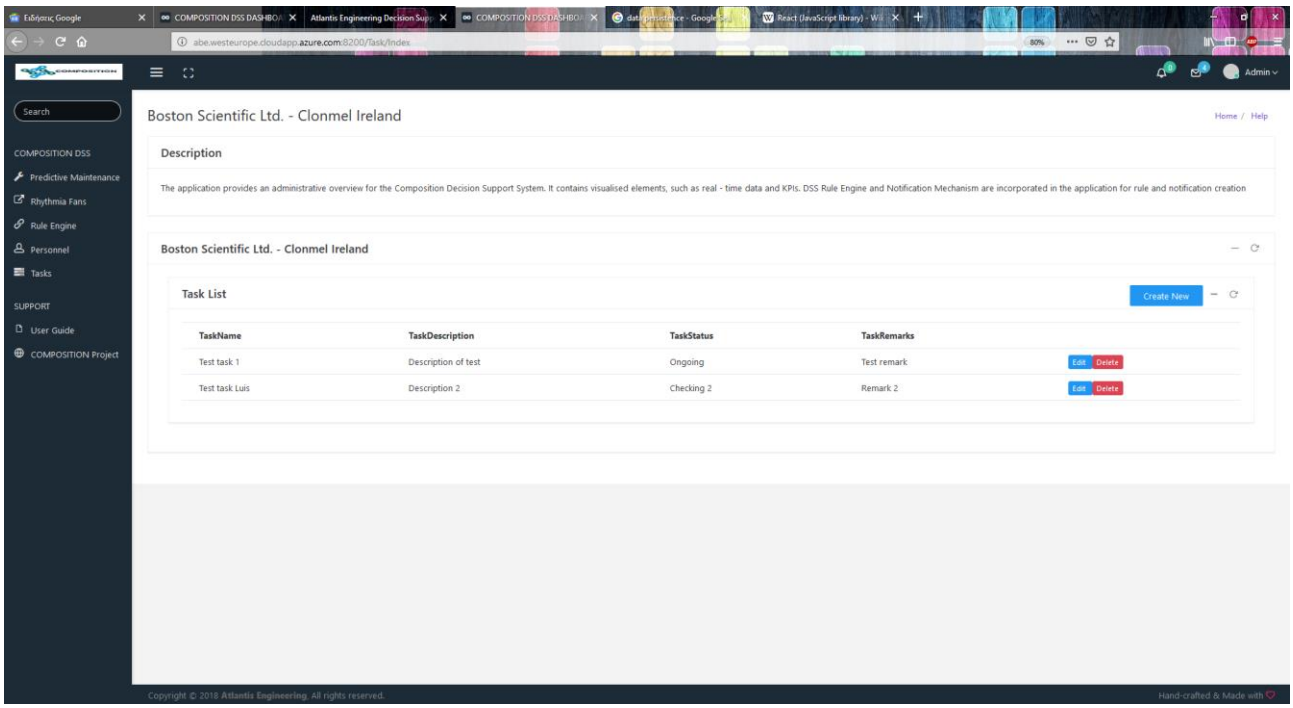

Figure 25: DSS Task Management Screen

During M20 to M30 there was a change in the template used for the HMIs. The new template was decided by all members of the project and it should be implemented for all HMIs. Since HMI integration is a project requirement, all HMIs have all now similar views. The next step if the fully integrated HMIs for the COMPOSITION project. This task will be completed using web components. ***Web Components*** are a set of features that provide a standard component model for the Web allowing for encapsulation and interoperability of individual HTML elements. A custom web component will be developed, which will include all menus and

sub–menus of all COMPOSITION HMIs. Then the component will be implemented to all HMIs and the COMPOSITION user will be able to see the unified HMI for all use cases and shop floors.

5.4.1 React Components

In the months between the two deliverables for the Manufacturing Decision Support System, a new UI was developed for the Rule Engine. The rule engine machine can be view as completely independent product and for that reason, a completely different approach was used for its UI development.

The UI development is based on React.js and React components. The reason to use React is that the components are easily reusable in the project and the can be written in form very similar to HTML. Each component is independent and the combination of all components on the layout gives us the final views.

The Model-View-Controllers (MVC) architecture is enhanced using react components for the views. It allows the programmers to develop less complicated and more independent code. React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. Complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API.

React components implement a *render()* method that takes input data and returns what to display. Usually React.je uses an XML-like syntax called JSX. Input data that is passed into the component can be accessed by *render()* via *this.props*. In addition to taking input data (accessed via *this.props*), a component can maintain internal state data (accessed via *this.state*). When a component's state data changes, the rendered mark-up will be updated by re-invoking *render()*. Using *props* and *state*, we can put together a small Todo application. This example uses *state* to track the current list of items as well as the text that the user has entered. Although event handlers appear to be rendered inline, they will be collected and implemented using event delegation.

For the Rule engine UI, an example of the React components used is given below. The example is the Actions React.js components

```
export  var Action = ({row, changer,deleter})=>
   {
       var input_in_cell_style= {verticalAlign:"top"};
       return (
            <div style={{marginBottom:60,paddingBottom:20}}>
                  <FormControl_Basic
                  legend={<div  class="label1" ></div>}
                  control={
                        <div style={{display:"flex"}}>
                        <div style={{flexGrow:1}}></div>
                        <div>
                              <button    className={"btn    btn-outline-danger"}
                              onClick={deleter}>          <span           aria-
                              hidden="true">&times;</span>             </button>
                              </div>
                        </div>}/>
                  <FormControl_Basic
                  legend={<div  class="label1" >  Type </div>}
                  control={
                  <div>
                        <select     className="form-control"     value={row.Type}
                        name="Type"  onChange={changer}>
```

```
                                <option value="Notification">Notification</option>
                                <option value="Create_Task">Task </option>
                                <option value="Alert">Alert</option>
                                <option value="Suggest">Suggest</option>
                                <option value="Log">Log</option>
                          </select>
                    </div>}/>
                    <FormControl_Basic
                    legend={<div   class="label1" >  Category       </div>}
                    control={
                          <input        className="form-control"        type="text"
                          name="Message_Category"               onChange={changer}
                          value={row.Message_Category} /}/>
                    <FormControl_Basic
                    legend={<div   class="label1" >  Message       </div>}
                    control={
                          <textarea                        className="form-control"
                          rows={5}className="form-control"             type="text"
                          name="Message_Desc"                  onChange={changer}
                          value={row.Message_Desc} /}/>
                    <FormControl_Basic
                    legend={<div   class="label1" >  Method       </div>}
                    control={
                         <input className="form-control" name="method" />}/>
              </div>
      );
}
```

5.5 Data Persistence

DSS Data Persistence sub–component is applied to the system, in order to secure that both incoming and outcoming data is persistent.

Data must comply to a schema that is applied from DFM and DLT and Data Persistence sub-component must be able to handle it as it is. After, using and exploiting the data, the Data Persistence sub-component preserves it format and the applied by the schema rules.

Data Persistence sub-component provides the internal mechanism, in the DSS that allows it to access the data without changing it. It is written in C# as the whole DSS background application. DSS output data is also based on the Data Persistence sub-component and it uses the pre-defined schemas for output data on the COMPOSITION project

5.6 KPIs

KPIs are used to measure performance in manufacturing processes. One of the major roles in creating KPIs for those processes is using data coming from sensors on the shop floor to measure pre-defined values, suitable for manufacturing KPIs. Also, the data can be used as input to DSS and the outputs can provide measurements and statics that create KPIs concerning the decision-making process on the shop floor. Both roles rely heavily on data existence and the relation that might exist between it.

There are different ways to choose a set of KPIs for certain processes. Decision-makers or managers should choose from:

- a fully certified KPI set such as: BREEAM, Open House, Super Building etc
- Select KPIs from existing sets
- Add new KPIs when existing and certified sets do not satisfy the needs for a specific case

Decision-making process is highly iterative and many results may appear in later iterations. New KPIs should cover these iterations. Another fact that should be taken into consideration is the combined knowledge that KPIs provide and how this abstract notion should be reform to KPIs.

The context and content of data on different shop floor lead to the implementation of different KPIs, both as visualisation and decision-making tools. One example is two different factories, use the same machine and the same monitoring sensors. The first one uses the machine in a very stable environment and the slightest changes in its operations lead to problematic situations and the need for maintenance processes or changes in the manufacturing process in order to prevent faults. The second machine is located in a heavy working environment, which implies heavy and unstable use of the machine. Small changes in data do not affect the manufacturing procedure. In both cases, the produced data is the same, but the purpose of the machine is different in the two shop floors and the resulting KPIs for the machine operation should be different.

There are different models applied on shop floors for different procedures. There are models about manufacturing processes, maintenance processes, security and safety processes that should be considered when creating KPIs in a DSS. While there may be some actors, assets and tasks that are the same on all shop floor processes, there are some that exclusively belong to different processes. Applying a set of KPIs, those factors should be implemented.

Finally, creating a set of KPIs should consider the people that will see the KPIs and learn something from it. Different information is considered useful for workers, technicians, safety actors, decision-makers or managers. The suitability of the provided information to different actors should be one of the main aspects to consider while creating or setting KPIs.

COMPOSITION DSS functionality is to create rules in the rule engine in order to help the decision-making process, while there is the possibility of visualisation. Live data is available that can be used for the KPIs. Use case and model analysis should lead to the KPIS which will be used by COMPOSITION DSS and provide knowledge for the shop floor and maintenance procedures. For each use case, the KPIs should provide information to all different actors, or be extracted and become further knowledge for the DSS.

In both UC-BSL- 2 Preventive Maintenance and UC-KLE-1 Maintenance Decision Support a sensor network has been deployed on the shop floors. Sensors send live data to other relevant components and then it is propagated to the DSS, through these components. It is possible the data to be propagated formatted from the previous components or unformatted. In both cases, DSS should exploit the data to create KPIs for all actors on the shop floors and visualise some of the KPIs. The transformed data usually is probabilities of failures on the machines and the raw data shows values straight from the sensors.

The received data is transformed in the DSS rule engine and according to certain conditions, as it is already explained in chapter 5.3.3, extracted KPIs are created. Each actor receives the most suitable KPIs with the needed information in order to complete their tasks.

The most common KPIs defined for the COMPOSITION DSS, in their general form, are given below:

- ***Number of failures***. The number of failures is measured and creates a KPI that shows how many times the machine failed due to a cause detected on the machines. This number can be translated as wrong operational procedure of the machine, non-existing safety procedures for the machine and other factors, better known to managers
- ***Number of maintenance tasks***. The KPI gives the number of times the machine needed maintenance for problems caused and detected by the machines.
- ***Probability of failures:*** this KPI is a data transformation, that indicates how possible it is for a machine to fail due to the data that is detected from the sensors.
- ***Number of alerts:*** DSS can create alerts, while implementing a rule, when the detected data is above or below certain pre-defined thresholds.
- ***Number of alarms:*** DSS also sends the number of alarms created while implementing a rule, when the detected data is above or below a certain pre-defined thresholds. The difference between alerts

and alarms is that alert is a notification which in not critical, and only warns the actors on shop floors that something has caused a malfunction, breakdown etc. On the other hand, alarms are critical notifications that are sent when the criticality of the breakdown is severe and can cause problems to workers and machines alike.

- ***Data values.*** Usually the sensor data values are defined as KPIs concerning the machines

KPI visualisation is indicative for shop floors actors, because they receive necessary information at a glance. DSS can visualise all of the above KPIs in a pleasant and easy way to be available to actors. Further study of the use cases and collaboration with the end users can enhance the available KPIs. New KPIs can be added, which will address the specific needs of the end users. There is also the possibility for applying advanced KPIs, which will be the combination of different KPIs.

5.6.1 Maintenance KPIs

For both KLEEMANN and Boston Scientific there are maintenance KPIs developed for the maintenance procedures on the shop floors. The first KPIs to be developed were the MTTR and MTBF (Mean Time To Repair and Mean Time Between Failures).

The KPIs were based on the historical data provided by KLEEMANN for the BOSSI machine. They can be computed for a certain time period in the past and show the health status for each KPI. Following maintenance procedures and standards, it was defined that the machines are aligned to assets, failures to failure modes and the time periods to timeframes. The given values are the possibilities provided by the prediction from SFT and DLT/LA COMPOSITION components.

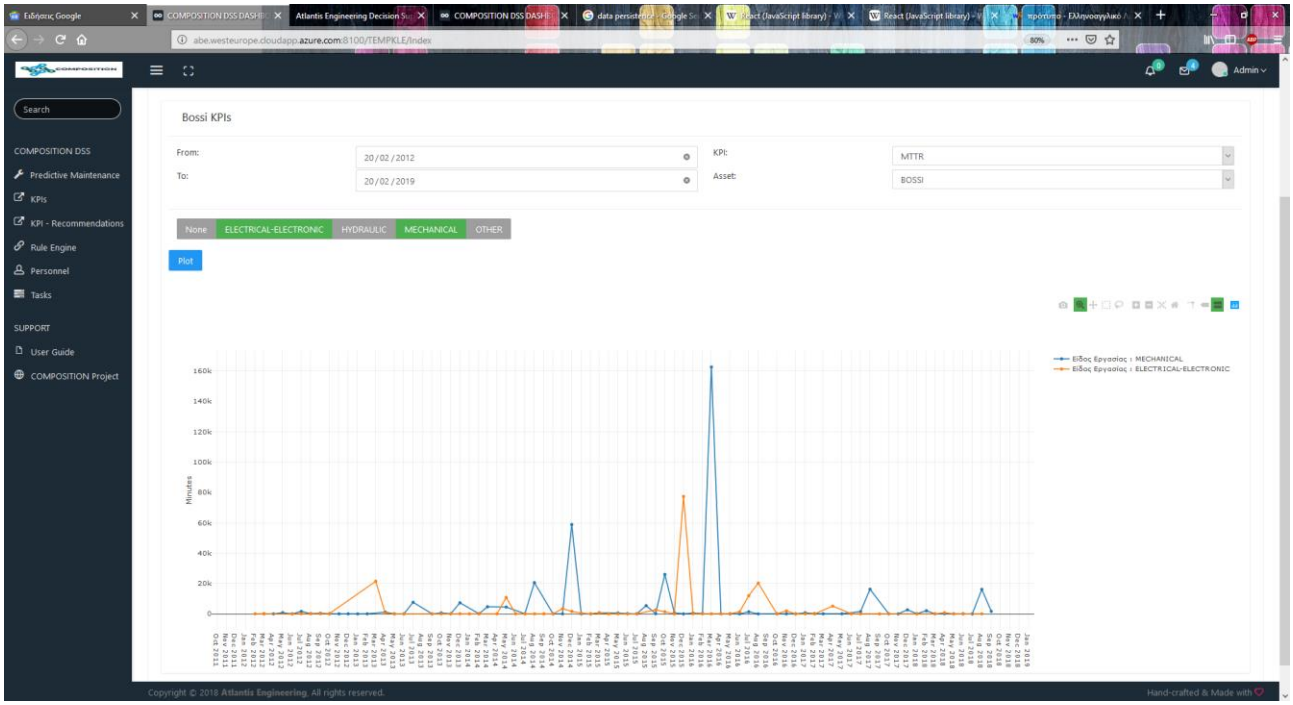

Figure 26: MTTR KPI

Figure 26 shows the MTTR KPI for the KLEEMANN shop floor for the time period between 20/01/2012 and 20/02/2019. The user has chosen to see the KPI for the Electrical and Mechanical failure modes. The user can also change the time period for the concerning data, as well as the related KPI. There are five KPIs provided. MTTR, MTBF, Response Time, Downtime and Count. These KPIs are maintenance related KPIs. The plot is now provided as a time series, but there are future possibilities for showing it as bar or pie charts. Users plot the KPI graph with all the fields completed for the period they need to see.

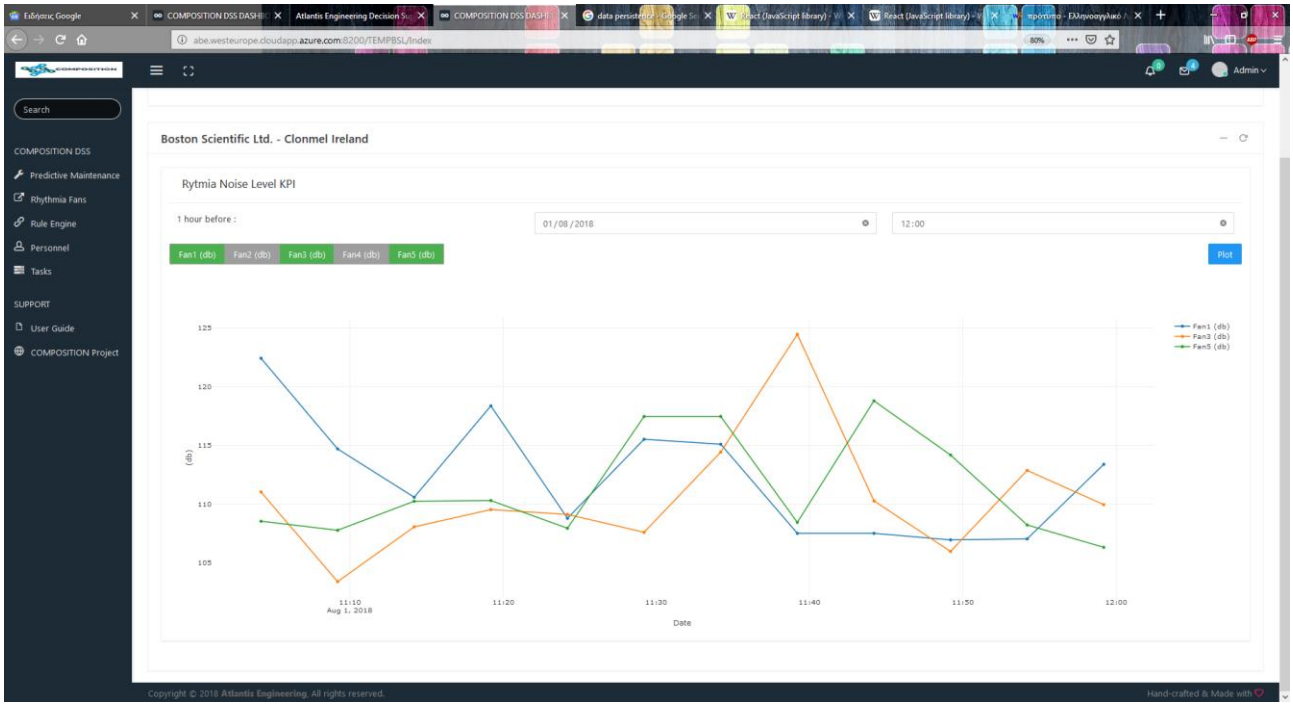

Figure 27: Fan Noise for BSL

Figure 27 shows the comparison of noise for a certain time period on the Boston Scientific shop floor. Users can choose the date and time for which they are interested in observing the noise on the fans. They can also choose which fans they need to see on the diagram. Maintenance KPIs are not provided on the Boston Scientific shop floor, because maintenance procedures are pre-defined and arranged in a way that they can be performed without the interruption of the work process, even if there is a breakdown on a machine. There are always spares machines in the production line. On the other hand, the health status, as indicated by the noise level is useful in monitoring the machine and preventing breakdowns.

During the final stage of COMPOSITION project new KPIs will be added, concerning the notifications that arrive to the users. A feedback application will be given to the users and a counter will be applied to the notification's mechanism. The counter will count each notification the users get on the shop floors and what feedback they send back. A rating system of the feedback will be implemented too, which will provide a new KPI about the satisfaction of the user concerning the notifications they get.

6 COMPOSITION Components Collaboration with DSS

COMPOSITION project provides collaboration with different components, for different use cases. The components need for the Manufacturing Decision Support System to work together are: Simulation and Forecasting Toolkit, Digital Factory Model and Deep Learning Toolkit. A brief representation of these different toolkits and their basic functionalities is given in this chapter.

6.1 Simulation and Forecasting – SFT

6.1.1 Enhance Decision Making in Production

The Simulation and forecasting tool (SFT) component is part of the high-level platform of COMPOSITION, the Integrated Information Management System (IIMS). Simulation and forecasting tool's main purpose is to simulate process models and provide forecasts of events whose actuals outcomes have not yet been observed. The Simulation and forecasting tool will enhance the decision support in both production line and logistics. In order to be able to provide predictions, the Simulation and forecasting tool uses both static and dynamic data.

More precisely, the Simulation and forecasting tool offers to the DSS predictions related to predictive maintenance for both KLEEMANN and BSL production lines. A probability of future faults (hydraulic, mechanical and electrical) based on KLEEMANN BOSSI polishing machine's historical data are available from SFT to DSS. Furthermore, a Machine Vibration Diagnosis Profile methodology that detects abnormal behaviour of vibrations supported from SFT is connected to DSS for both visualization and events notifications. Moreover, SFT provides to DSS predictions for BSL production line based on acoustic sensors' data and methodologies such as Density-based spatial clustering of applications with noise (DBSCAN) a Local Outlier Factor methodology and Support Vector Machines (SVMs) classification. By collecting output from this SFT methods, the DSS can inform the end-user for outliers on the fans' operation from BSL ovens in real-time.

6.1.2 Enhance Decision Making over the Supply Chain

Besides the production line, the Simulation and forecasting tool enhances the decision support over the supply chain. A Tonnage-Route Genetic Algorithm or T-RGA is developed and applied for UC-ELDIA 1. A generative algorithm encounters a function of routes and tonnage and calculates a pair of values that minimizes the function through an optimization process. The Genetic Algorithm finds the optimum pair (number of routes, tonnage) to minimize the proportion: *min (routes/weight)*

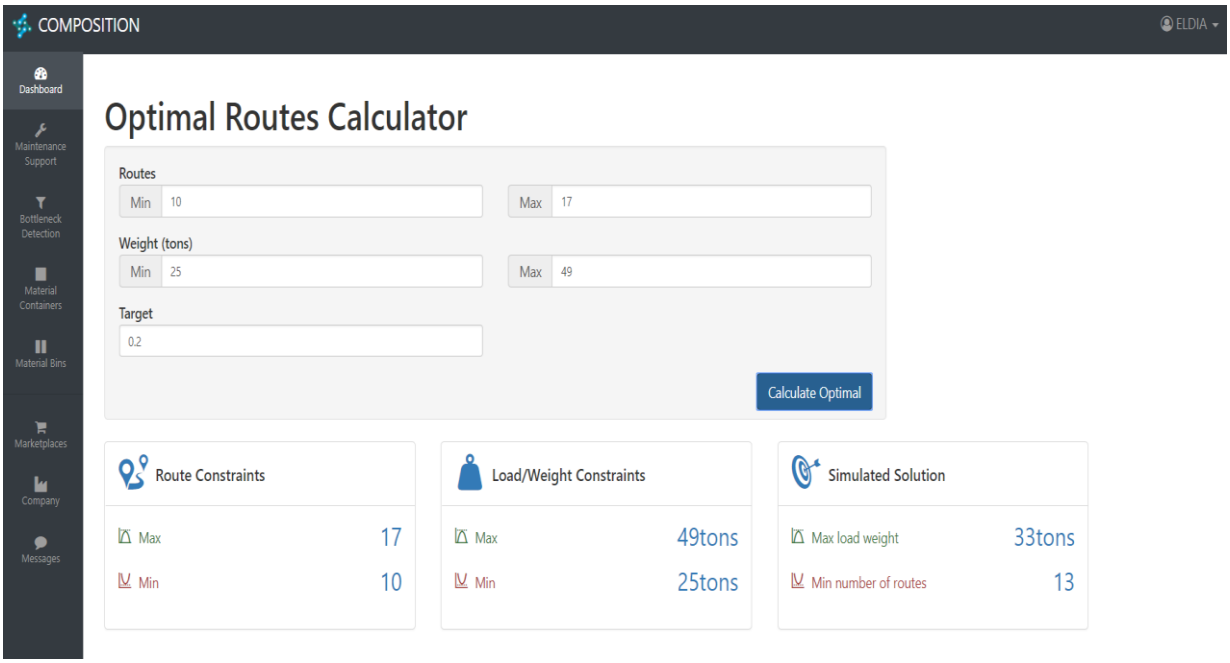

Figure 28: Optimal Routes Calculator for Decision Support in Logistics

Furthermore, algorithms and methodologies from SFT such as Time Series Forecasting and Markov chain supports the decision-making and planning over the supply-chain and waste management scenario from ELDIA. Both methods provide to end-user with estimations of the tonnage that will be transferred in the next months. More precisely the trend analysis informs the user for the trend of the transferred tonnage and the Markov methodology provides to the user the probability to have increased or decreased transferred tonnage in next months based on current trend.

Figure 29: SFT Output for Tonnage Forecasting

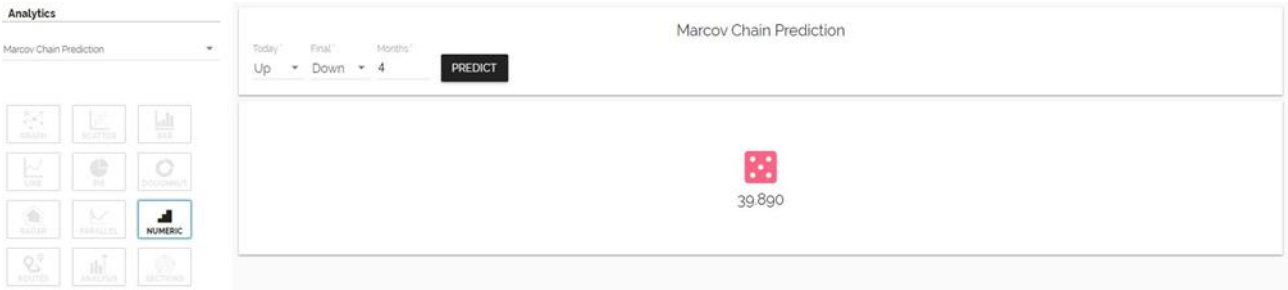

Figure 30: SFT Output for Markov Chain Prediction

Besides the solutions were based on historical data about tonnages, SFT enhance the decision-making about the selection of the time that bins should be empty. A Trend Analysis methodology has been applied for this case.

Figure 31: SFT Output for Fill Level Trend Analysis

The available algorithms and methodologies from SFT have been detailly described in D3.5 and D3.7 reports.

6.2 Digital Factory Model – DFM

The Digital Factory Model is a core component of the COMPOSITION system. The DFM enables the digitalization of industrial aspects. Data which are provided from different system's parts in a heterogeneous format finally are described in a common format using DFM schema. This means that all the data are modelled and provided with the same format to all related components. The Digital Factory Model can describe all the information related to a factory such as buildings, assets, actors, processes and live events.

Every factory can be represented by a DFM instance. IIMS components can build DFM instances or read data from these instances by using the DFM API services which is based on RESTful services and HTTP protocol.

The DSS is strongly connected with the DFM. By using the DFM API services the DSS can get all the static information related to a factory. The data exchange format is based on DFM schema which is built upon well-known standards. Information about actors, assets and equipment are described using B2MML standard.

Moreover, the BPMN diagrams from Task 3.1 are available to the DSS via the DFM API. The DSS can get the corresponding to a factory instance, BPMN diagrams by calling the appropriate service for Business Processes. The Business Process List element of the DFM was covered by OMG's BPMN XML package. This package provides schemas which offer all the necessary means for the representation of a BPMN diagram in a DFM instance.

Furthermore, SFT output become available to DSS through DFM. The SFT predictions are posted to DFM as OGC Observation and they are collected from DSS for visualization or notification purposes.

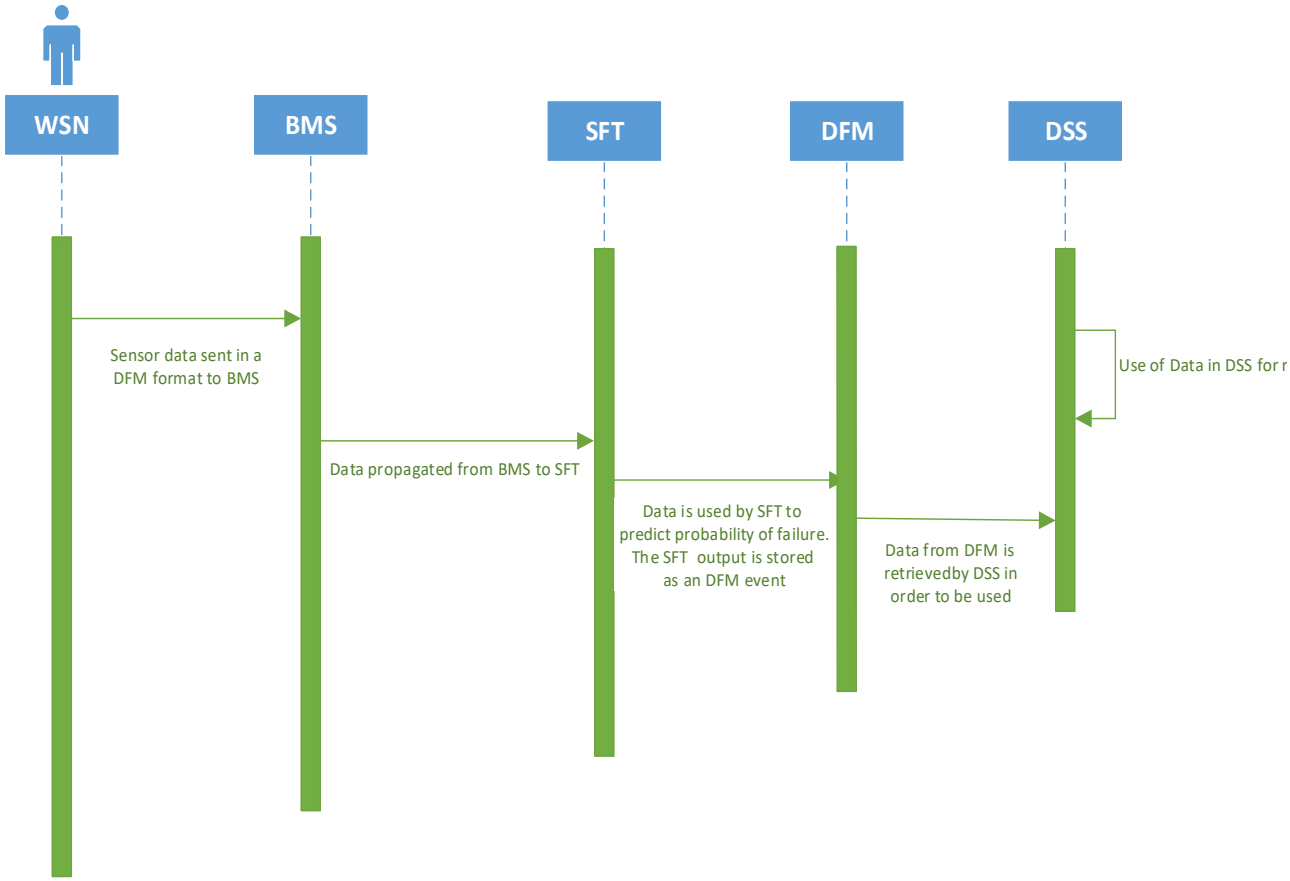

Figure 32: Sequence Diagram for the data route in with DFM and SFT COMPOSITION Components

Figure 32 is the sequence diagram which provides the route the data has to follow in order to reach DSS from SFT and DFM components. The basic logic behind the sequence diagram is that the data is gathered from WSN and is sent to DFM. If data follows the DFM model, it can be processed from all the rest COMPOSITION components. This data is propagated to SFT, where they are transformed from sensorial data to meaningful data (e.g. probability of failure of the BOSSI polishing machine on KLEEMANN shop floor). The transformed

data is sent to DSS through the exchange protocols, previously discussed. DSS uses and transforms the data in the rule engine, according to internal processes and creates rules and KPIs based on it.

6.3 LinkSmart® IoT Learning Agent _LA

The LinkSmart® IoT Learning Agent is the component in charge of collecting, propagating and pre- and post-processing the data to be able to predict and learn in real-time.

The LA provide a processing pipeline infrastructure for real-time, runtime, and on-the-fly data processing. Additionally, the LA provide a CEML framework which allows the real-time data and model management for effectively continuous training and evaluating the models. The model's implementation is provided by the DLT (see6.4)

The data sent by the BMS, it is parsed, collected and aggregated following the deployment pre-processing rules for the realization of BSL-2 use case. The process is described in detail in deliverable D5.2. Afterwards, the data is sent to the DLT for forecasting. The result is the serialized and propagate in JSON or JWS format.

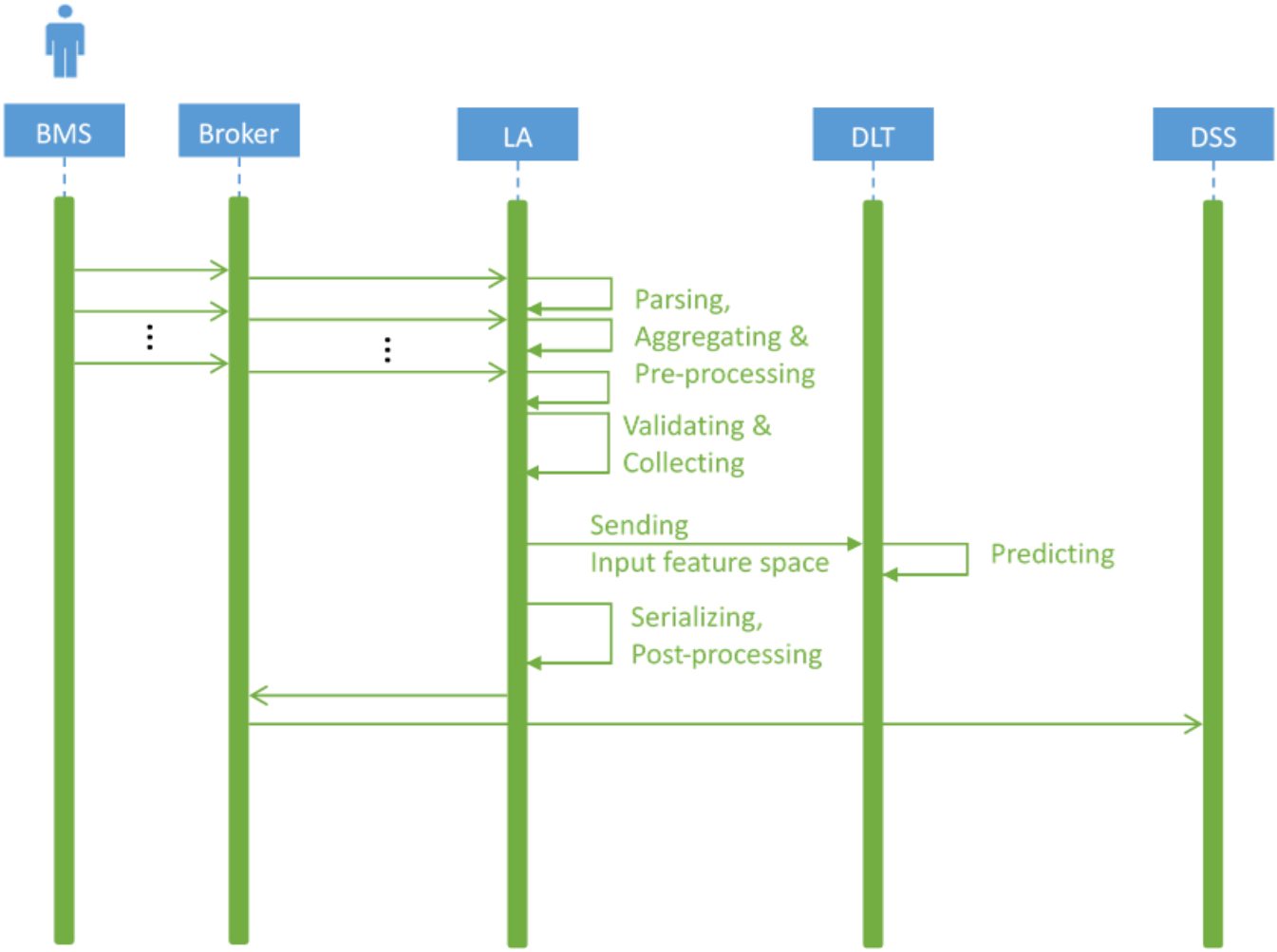

Figure 33: Sequence diagram BMS to DSS for BSL-2 use case

6.4 Deep Learning Toolkit – DLT

Deep Learning Toolkit is developed by ISMB in order to provide a provision on the next possible breakdowns, based on live data analysis. The analysis is based on a trained artificial neural network. The training data set is usually a historical data set (COMPOSITION, 2017).

DLT provides the output data to the DSS in order to create rules in the rule engine to face and solve shop floor breakdowns. Specifically, UC-BSL-2 Predictive Maintenance requires the communication between DLT and DSS.

DLT input data comes straight from the sensor network on the shop floor. BSL Rhythmia ovens are continuously monitored and provide live data to the Intra-Factory Management System. The Big Data Analytics module, now deployed as Learning Agent is responsible for the operational procedures involving the aggregation of the sensor data, in order to create a suitable mapping for the Deep Learning Toolkit. The Deep Learning Toolkit returns previsions to the agent in reaction to incoming live samples.

The Learning Agent module is therefore in charge to dispatching the data to the DSS.

7 Components Integration with DSS

Components integration is ongoing for all COMPOSITION components. The protocols needed for integration are HTTP protocol with REST/SOAP queries and MQTT with direct messaging through topics. Also, a major requirement for the COMPOSITION project is the dockerisation of each component and the Portainer used for them. A central docker server is provided by ISMB, where the components are dockerised and component integration should take into consideration this process.

7.1 Keycloak Integration

DSS embodies a series of technologies, processes and practices designed to protect networks, computers, programs and data from attack, damage or unauthorized access. In a computing context, security includes both cybersecurity and physical security. Ensuring cybersecurity requires coordinated efforts throughout an information system. Elements of cybersecurity include:

- ***Application security***
- ***Information security***
- ***Network security***
- ***Disaster recovery / business continuity planning***
- ***Operational security***
- ***End-user education***

One of the most problematic elements of cybersecurity is the quickly and constantly evolving nature of security risks. The traditional approach has been to focus most resources on the most crucial system components and protect against the biggest known threats, which necessitated leaving some less important system components undefended and some less dangerous risks not protected against. Key cloak backed with blockchain is the hard of application security.

Key cloak is an open source identity and access management solution which is commercially supported by JBoss. I was running Key cloak using the JBoss Key cloak official Docker image (https://hub.docker.com/r/jboss/keycloak/). By default, there is no SSL enabled so I needed to run it behind HAProxy with SSL offload enabled. You can run it without SSL but some .NET classes refused to work without https prefix (but not sure if it's required for this exact scenario). If you run Key cloak behind a proxy you will need to enable the proxy forwarding as described in the docker page to make it work properly. In addition, you need to supply the admin username and password as variables to the docker container to be able to login.

The goal is to have protected pages in the WebApp which require the user to sign-in with OpenID through Key cloak server. In the WebApp we should be able to use the claims/roles given by the Keycloak and should be able to call the WebAPI with a bearer token. WebAPI should be able to know the identity of the calling user and not only the calling service in secure way.

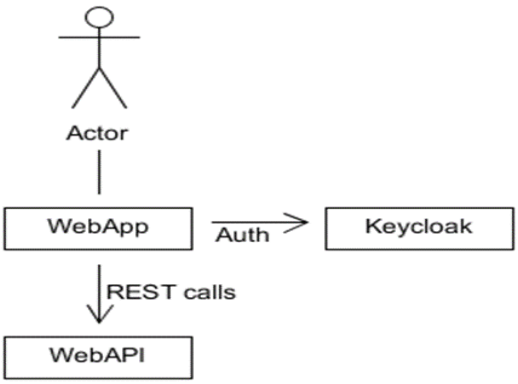

Figure 34: Keycloak Flow Diagram

When using Keycloak as an identity broker, users are not forced to provide their credentials in order to authenticate in a specific realm. Instead, they are presented with a list of identity providers from which they can authenticate. You can also configure a default broker. In this case the user will not be given a choice, but instead be redirected directly to the parent broker.

The following diagram in Figure 35 demonstrates the steps involved when using Key cloak to broker an external identity provider:

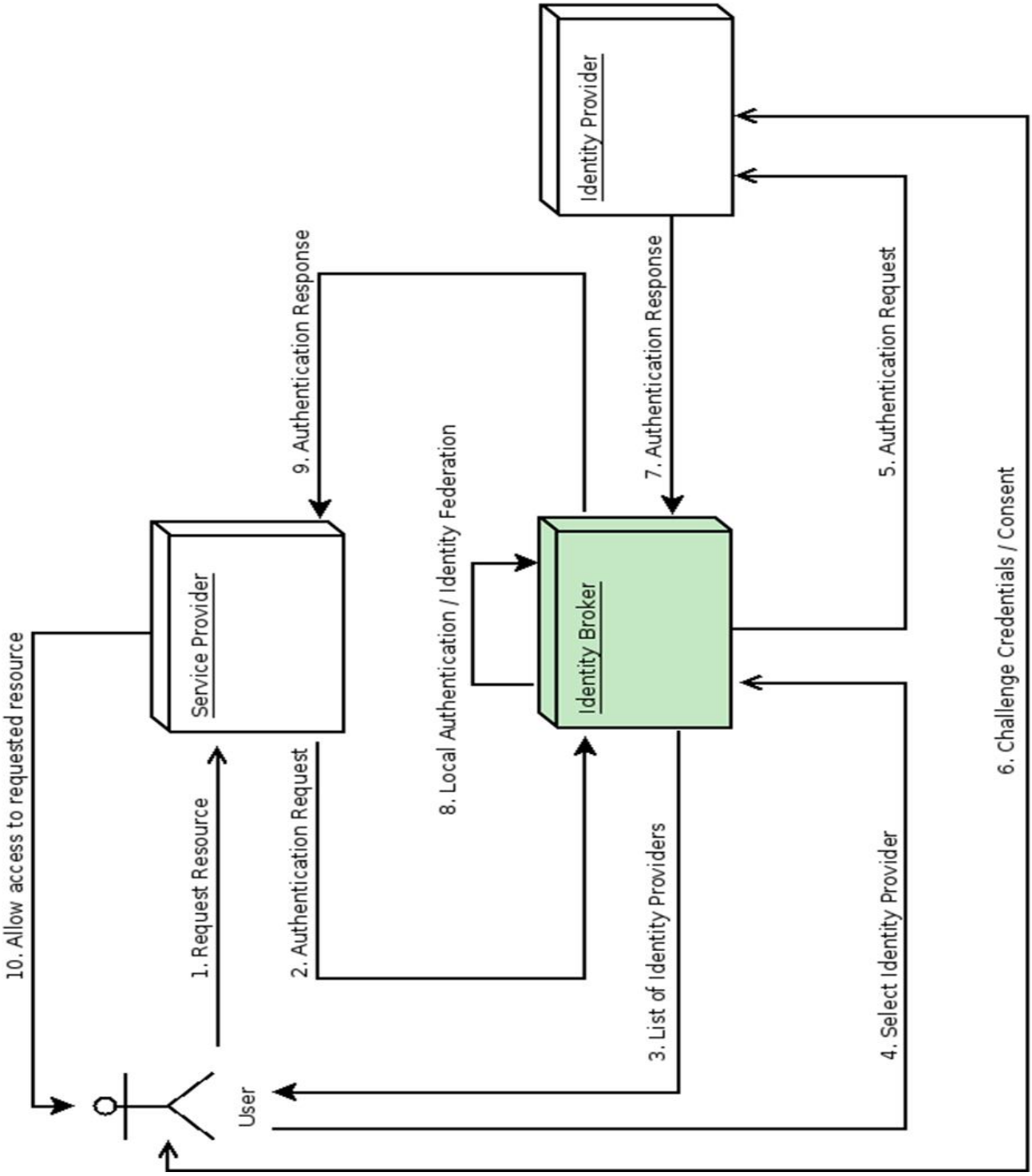

Figure 35: Identity Broker Flow diagram

Table 10: Identity Broker Flow

Identity Broker Flow
1. User is not authenticated and requests a protected resource in a client application.
2. The client applications redirect the user to Key cloak to authenticate.
3. At this point the user is presented with the login page where there is a list of identity providers supported by a realm.
4. User selects one of the identity providers by clicking on its respective button or link.
5. Key cloak issues an authentication request to the target identity provider asking for authentication and the user is redirected to the login page of the identity provider. The connection properties and other configuration options for the identity provider were previously set by the administrator in the Admin Console.
6. User provides his credentials or consent in order to authenticate in the identity provider
7. Upon a successful authentication by the identity provider, the user is redirected back to Key cloak with an authentication response. Usually this response contains a security token that will be used by Key cloak to trust the authentication performed by the identity provider and retrieve information about the user.
8. Now Key cloak is going to check if the response from the identity provider is valid. If valid, it will import and create a new user or just skip that if the user already exists. If it is a new user, Key cloak may ask the identity provider for information about the user if that info doesn't already exist in the token. This is what we call identity federation. If the user already exists Key cloak may ask him to link the identity returned from the identity provider with his existing account. We call this process account linking. What exactly is done is configurable and can be specified by setup of First Login Flow. At the end of this step, Key cloak authenticates the user and issues its own token in order to access the requested resource in the service provider.
9. Once the user is locally authenticated, Key cloak redirects the user to the service provider by sending the token previously issued during the local authentication.
10. The service provider receives the token from Key cloak and allows access to the protected resource.

There are some variations of this flow that we will talk about later. For instance, instead of presenting a list of identity providers, the client application can request a specific one. Or you can tell Key cloak to force the user to provide additional information before federating his identity. As you may notice, at the end of the authentication process Key cloak will always issue its own token to client applications. What this means is that client applications are completely decoupled from external identity providers. They don't need to know which protocol (e.g.: SAML, OpenID Connect, OAuth, etc) was used or how the user's identity was validated. They only need to know about Key cloak.

The policies concerning the DSS and the HMIs are already applied on the Keycloak mechanism. When a user tries to access on of the HMIs is redirected to the Keycloak sing in page. They provide their credentials and if the credentials are correct the are redirected to the HMI page, along with the token of authentication and authorisation. The token is passed through the different HMIs and the user does not need to sign in again for each new HMI they access.

If an unauthorised user tries to access any of the HMIs with already implemented policies, they are denied access. This results only to authenticated users to be able to use the COMPOSITION HMIs and be validated by the Keycloak mechanism. Finally, the above process agrees with the project requirements of single sign-on and safety and security on the COMPOSITION application.

7.1.1 First Log–in Flow

When a user logs in through identity brokering some aspects of the user are imported and linked within the realm's local database. When Key cloak successfully authenticates users through an external identity provider there can be two situations:

- There is already a Key cloak user account imported and linked with the authenticated identity provider account. In this case, Key cloak will just authenticate as the existing user and redirect back to application.
- There is not yet an existing Key cloak user account imported and linked for this external user. Usually you just want to register and import the new account into Key cloak database, but what if there is an existing Key cloak account with the same email? Automatically linking the existing local account to the external identity provider is a potential security hole as you can't always trust the information you get from the external identity provider.

Different organizations have different requirements when dealing with some of the conflicts and situations listed above. For this, there is a First Login Flow option in the IDP settings which allows you to choose a workflow that will be used after a user logs in from an external IDP the first time. By default, it points to first broker login flow, but you can configure and use your own flow and use different flows for different identity providers. The flow itself is configured in admin console under Authentication tab. When you choose First Broker Login flow, you will see what authenticators are used by default. You can re-configure the existing flow. (For example, you can disable some authenticators, mark some of them as required, configure some authenticators, etc).

7.1.2 Default First Log–in flow

Let's describe the default behaviour provided by First Broker Login flow.

- ***Review Profile*** This authenticator might display the profile info page, where the user can review his profile retrieved from an identity provider. The authenticator is configurable. You can set the Update Profile on First Login option. When On, users will be always presented with the profile page asking for additional information in order to federate their identities. When missing, users will be presented with the profile page only if some mandatory information (email, first name, last name) is not provided by the identity provider. If Off, the profile page won't be displayed, unless user clicks in later phase on Review profile info link (page displayed in later phase by Confirm Link Existing Account authenticator)
- ***Create User If Unique*** This authenticator checks if there is already an existing Key cloak account with same email or username like the account from the identity provider. If it's not, then the authenticator just creates a new local Key cloak account and links it with the identity provider and the whole flow is finished. Otherwise it goes to the next Handle Existing Account sub flow. If you always want to ensure that there is no duplicated account, you can mark this authenticator as REQUIRED. In this case, the user will see the error page if there is existing Key cloak account and the user will need to link his identity provider account through Account management.
- ***Confirm Link Existing Account*** On the info page, the user will see that there is an existing Key cloak account with same email. He can review his profile again and use different email or username (flow is restarted and goes back to Review Profile authenticator). Or he can confirm that he wants to link the identity provider account with his existing Key cloak account. Disable this authenticator if you don't want users to see this confirmation page but go straight to linking identity provider account by email verification or re-authentication.
- ***Verify Existing Account by Email*** This authenticator is ALTERNATIVE by default, so it's used only if the realm has SMTP setup configured. It will send mail to the user, where he can confirm that he wants to link the identity provider with his Key cloak account. Disable this if you don't want to confirm linking by email, but instead you always want users to reauthenticate with their password (and alternatively OTP).
- ***Verify Existing Account by Re-authentication*** This authenticator is used if email authenticator is disabled or non-available (SMTP not configured for realm). It will display a login screen where the user needs to authenticate with his password to link his Key cloak account with the Identity provider. User can also re-authenticate with some different identity provider, which is already linked to his Key cloak account. You can also force users to use OTP. Otherwise it's optional and used only if OTP is already set for the user account.
- ***Securing the API***

Table 11: Client Configuration for the WebAPI

Client Configuration for the WebAPI
Client Protocol = openid-connect
Access Type = confidential (again a trusted client)
Standard Flow Enabled = off (the API is only called with a bearer token)
Implicit Flow Enabled = off (the API is only called with a bearer token)
Service Accounts Enabled = on (the API might want to call other services)
Authorization = on
Credentials / Client Authenticator = Client id and Secret
Configure the proper Root URL

We also need a new user to Keycloak but that is easy to create through the admin console. Just remember to set the user active. In the WebAPI we only configure the JSON Web Token support as the bearer token (= access token generated by Keycloak) should be the only way to call the service. Here we have an issue that I have been unable to solve (assuming it can be solved in a better way) since the Audience validation must be disabled. In this case the audience in the access token is actually the WebApp for which the token was originally generated for. However, that might not be a show stopper as we are able to check the validity of the token for this service in a different way explained later on.

7.2 Simulation and Forecasting Toolkit – SFT

Simulation and Forecasting Toolkit communicates directly with DSS, using an API created by SFT. The communication is achieved through HTTP protocol. The communication between DSS and SFT is based on DFM schema. The predictions of SFT are described as DFM events. The DSS is able to read these events in real time using the DFM API services. The output file, as well as the communication endpoint is shown below in

```
{
    "member": [
        {
            "href": "http://www.composition-project.eu/KLE_1_Fault_Probability"
        },
        {
            "resultTime": "09:02:36.656514",
            "result": {
                "uom": "http://www.composition-project.eu/uom#percentage",
                "value": 0.065992865636614744
            },
            "observedProperty": {
                "href": "http://www.composition-project.eu/bossi_fault_probability"
            },
            "id": "electrical_fault_id",
            "procedure": {
                "href": "http://www.composition-project.eu/predictive_maintenanceKLE"
            },
            "type": "CategoryObservation"
        },
        {
            "resultTime": "09:02:36.656514",
            "result": {
                "uom": "http://www.composition-project.eu/uom#percentage",
                "value": 0.06956004756242569
            },
            "observedProperty": {
                "href": "http://www.composition-project.eu/bossi_fault_probability"
            },
            "id": "mechanical_fault_id",
            "procedure": {
                "href": "http://www.composition-project.eu/predictive_maintenanceKLE"
            },
            "type": "CategoryObservation"
        }
    ],
    "phenomenonTime": {
        "instant": "09:02:36.656514"
    },
    "featureOfInterest": {
        "href": "http://www.composition-project.eu/faultProbability"
    },
    "_id": {
        "$oid": "5a7b0431c4e4cd27a4082c41"
    },
    "id": "KLE_1_SFT_Event"
}
```


Figure 36: JSON output for the SFT

SFT output is available to DSS through DFM for the UC-BSL-2 Predictive Maintenance, UC-KLE-1 Maintenance Decision Support and UC-KLE-3 Scrap Metal and Recyclable Waste Transportation. All the predictions' output is presented as OGC Observations in JSON format.

7.3 Digital Factory Model – DFM

Both static and dynamic information of a factory can be available from DFM to DSS. The DSS is able to get/read data from DFM factory instances by using a wide catalogue of web services provided by DFM API. All the transactions are based on HTTP and RESTful web services. The end-points are secured by using basic-auth.

The next figure presents the complete list of web services from DFM API that the DSS can use:

getActorByID	setActor	deleteActor
getActorList	setActorList	deleteActorList
getAssetByID	setAsset	deleteAsset
getAssetList	setAssetList	deleteAssetList
getEquipmentByID	setEquipment	deleteEquipment
getEquipmentList	setEquipmentList	deleteEquipmentList
getProcedureByID	setProcedure	deleteProcedure
getProcedureList	setProcedureList	deleteProcedureList
getSensorByID	setSensor	deleteSensor
getSensorList	setSensorList	deleteSensorList
getBuildingInformation	setBuildingInformation	deleteBuildingInformation
getBPMN	setBPMN	deleteBusinessProcess
getBusinessProcessList	setBusinessProcessList	deleteBusinessProcessList
getFactoryInformation	setObservation	deleteObservation
getObservationByID	setSample	deleteSample
getSampleByID		

Figure 37: DFM supported interfaces

7.4 LinkSmart® IoT Learning Agent

The LA communicates with the DSS through MQTT protocol. Data and information traverse through LA to other COMPOSITION components using MQTT, uses OGC Sensor Things standard. An example of a data outbound schema is given below, in JSON format:

```
{
    "parameters": [{
            "evaluations": [{
                    "@id": 1,
                    "method": "More",
                    "normalizedResult": 0.0,
                    "
                    ready": false,
                    "result": 0.0,
                    "controlMetric": false,
                    "name": "MatthewsCorrelationCoefficient",
                    "target": 0.5,
                    "currentValue": 0.0
                }, {
                    "@id": 2,
                    "method": "More",
                    "controlMetric": true,
```

```
                    "ready": true,
                    "result": 155.0,
                    "normalizedResult": 1.0,
                    "name": "SlideAfter",
                    "target": 100.0,
                    "currentValue": 155.0
                }
            ]
        }, {
            "input": [ <values omitted>]
        }, {
            "evaluations": [1, 2]
        }
    ],
    "datastream": {
        "@iot.id": "5c82aae2-2957-4b94-8c10-a5afa4783a6c"
    },
    "phenomenonTime": "2019-01-28T16:21:37.292Z",
    "result": [1.0, 0.9704771041870117, 160.0]
}
```


The data are received from the DSS through MQTT topics and web sockets and are shown as a probability percentage on the HMIs. In order to detect the failure in the RYTHMIA machines over the next two and half hours, when a value different from 0 is detected in retained in the systems memory, until a larger value arrives.

When the values changes, according to pre-defined thresholds, they trigger a rule from the rule engine. For example, when the probability values are below 40% the system is in its nominal operation and no rule is triggered. Values between 40% to 80% trigger warnings from the rules in the rule engine. When the detected value in over 80%, the probability of breakdown is really high and alerts, warnings and maintenance tasks are created by the rule engine, in order to avoid total breakdown and more maintenance time.

7.5 Deep Learning Toolkit – DLT

DLT communicates with other COMPOSITION components through MQTT protocols. It is also fully dockerised and contained in a portainer instance.

Data and information traverse through DLT to other COMPOSITION components. To eliminate miscommunication and lost packages caused by unformatted data, DLT follows a data schema agreed for all COMPOSITION components. An example of a data outbound schema is given below, in JSON format:

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Composition DLT prediction",
    "type": "object",
    "properties": {
        "phenomenonTime": {
            "type": "string"
        },
        "resultTime": {
            "type": "string"
        },
        "result": {
            "type": "number",
```

```
                "minimum": 0,
                "maximum": 1
            }
        },
        "required": [
            "phenomenonTime": "2012-02-16T08:06:09+00:00",
            "resultTime": "2018-02-21T15:32:31.380800+00:00",
            "result": 0
        ]
    }
```

The data are received from the DSS through MQTT topics and web sockets and are shown as a probability percentage on the HMIs. In order to detect the failure in the RYTHMIA machines over the next two and half hours, when a value different from 0 is detected in retained in the systems memory, until a larger value arrives.

When the values changes, according to pre-defined thresholds, they trigger a rule from the rule engine. For example, when the probability values are below 40% the system is in its nominal operation and no rule is triggered. Values between 40% to 80% trigger warnings from the rules in the rule engine. When the detected value in over 80%, the probability of breakdown is really high and alerts, warnings and maintenance tasks are created by the rule engine, in order to avoid total breakdown and more maintenance time.

8 Conclusions

The work that has been made in task T3.4 – Decision Support System for Optimising Manufacturing Process is examined in this deliverable.

The deliverable describes the new and state of the art technologies, which were used in the Decision Support System and its integration with other COMPOSITION components. The steps that followed during development of DSS is thoroughly described and analysed. Each DSS sub-component has its own technologies that are taken into consideration. Data streaming processes, Finite State Machines and HMI techniques are analysed and the work done on these sub-components is described. Also, the integration techniques are briefly described in the deliverable. The specific technologies used for the DSS are: a finite state machine ontology for the rule engine, which was created from the beginning, Rx/StreamInsight technologies along with data at rest and streaming process techniques for the Stream Processing sub-component, Angular5, HTML and CSS in a MVC structure for the HMI and finally HTTP and MQTT protocols for the connections and interactions with the rest of the components. Data persistence is ensured following good data base design and techniques and the pre-defined data schemas.

During M20 to M30 of T3.4 – Decision Support System for Optimising Manufacturing Process the objectives that were described in D3.8 – Manufacturing Decision Support System were met with new approaches. These objectives can be described with the following actions:

- ***Improvement of streaming processes and data ingestion***. Real-time data is now available to the system and the described techniques can be evaluated and used for much more and real-time data. The WSN is in place and provides a lot of data for different shop floors and the DSS is able to ingest it without missing important information and values.
- ***Extension of rule engine***. The DSS has been extend for new, more complicated rules. Also, data-driven rules are applied, when the WSN provides the necessary data.
- ***KPIs extraction and knowledge management***. There are several KPIs developed and applied in the system. They cover both maintenance and user satisfaction areas. Further development of the KPI mechanism will be achieved in the final phase of the project.
- ***HMI improvement***. New HMIs are implemented based on React and Web components and they will be fully integrated in the final phase of the project.
- ***Data persistence.*** Data should continue to follow the schemas provided for the COMPOSITION project and remain persistent through its run. Even if, in the next phase of the project there are a lot more data provided, it should be formatted the same way as the test data is formatted until now.
- ***Components Integration***. The components integration has started with the HMI integration and continues in the project for all components.

Finally, this deliverable is the result of an ongoing process, including technical challenges and solutions. Task T3.4 – Decision Support System for Optimising Manufacturing Process continues throughout the project remaining time and this deliverable D3.9 – Manufacturing Decision Support System II reflects. the collaborative effort put to complete the task requirements.

9 List of Figures and Tables

9.1 Figures

9.2 Tables

10 References

(Almeida, et al., 2007) Almeida, M., Moreira, N. & Reis, R., 2007. On the performance of automata minimization algorithms. *Technical Report Series: DCC - 2007 - 03.*

(Anderson, 2006) Anderson, J. A., 2006. *Automata Theory with Modern Applications.* 1st ed. Cambirdge: Cambridge Univesity Press.

(Anon., 2018) Finite State Machines. [Online] Available at: https://brilliant.org/wiki/finite-state-machines/

(Anon., 2018) Deterministic Finite Automaton, 2018, https://www.tutorialspoint.com/automata_theory/deterministic_finite_automaton.htm

(Anon., 2018) Non Deterministic Finite Automaton, 2018, https://www.tutorialspoint.com/automata_theory/non_deterministic_finite_automaton.htm

(Artisans, 2018) Artisans, Data, data – artisans, 2018, https://data-artisans.com/what-is-stream-processing

(COMPOSITION, 2017) D5.3 – Continuous deep learning toolkit for real time adaptation

(COMPOSITION, 2016) GRANT AGREEMENT NUMBER — 723145 — COMPOSITION

(Felsberger, et al., 2016) Felsberger, A., Oberegger, B. & Reiner, G., 2016. A Review of Decision Support Systems for Manufacturing Systems. SamI40 workshop at i-KNOW.

(Figueira, et al., 2015) Figueira, G., Amorim, P., Guimaraes, L. & Almada - Lobo, B., 2015. A decision support system for the operational production planning and schedulign of an integrated pulp and papaer mill. *Computers and Chemical Engineering.*

(Fraser, 1997) Fraser, B., 1997. Site Security Handbook. s.l.:SEI/CMU.

(Haykin, 2009) Haykin, S., 2009. Neural Networks and Learning Machines. 3rd ed. Hamilton, Ontario: Pearson - Prentice Hall.

(Jain, 2018) Jain, R., Algoworks, 2018, http://www.algoworks.com/blog/real-time-data-streaming-tools-and-technologies/

(Josang, 2017) Josang, A., 2017. A Consistent Definition of Authorization. Oslo: 13th International Workshop on Security and Trust Management.

(Kasie, et al., 2017) Kasie, F. M., Bright, G. & Walker, A., 2017. Decision support systems in manufacturing: a survey and future trends. Journal of Modelling in Management, 12(3), pp. 432-454.

(Miah, 2012) Miah, S. J., n.d. An Emerging Decision Support Systems Technology for Disastrous Actions Management. s.l.:s.n.

(Microsoft, 2018) Microsoft, 2018. *Microsoft Developer Network.* [Online] Available at: https://msdn.microsoft.com/en-us/library/hh242985(v=vs.103).aspx [Accessed 26 April 2018].

(Microsoft, 2018) Microsoft, 2018. *Microsoft Developer Network.* [Online] Available at: https://msdn.microsoft.com/en-us/library/ee362541(v=sql.111).aspx [Accessed 26 April 2018].

(Pirog - Mazur, 2004) Pirog - Mazur, M., 2004. The Model of Decision Support System for a Manufacturing Company. pp. 46-53.

(SATISFACTORY, 2014) *GRANT AGREEMENT - NUMBER - 636302 - SATISFACTORY,* Brussels

(SatisFactory, 2015) SatisFactory, 2015. *SatisFactory.* [Online] Available at: http://www.satisfactory-project.eu/satisfactory/ [Accessed 25 February 2019].

(Shim, et al., 2002) Shim, J. P. et al., 2002. Past, present, and future of decision support technology. Decision Support Systems, 33(2), pp. 111-126.

(Simon, 1959) Simon, H. A., 1959. Theories of Decision - Making in Economics and Behavioral Science. The American Economic Review, 49(3), pp. 253-283.

(Turner, 2016) Turner, D. M., 2016. Cryptomathic - Digital Authentication -- the basics. [Online]
Available at: https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics ,[Accessed 19 April 2018].

(Zaharia, et al., 2010) Zaharia, M. et al., 2010. *Spark: Cluster Computing with Working Sets.* Berkeley: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing.