D2.6- Final Requirements and Architecture Specifications

Deliverable ID	**D2.6**
Deliverable Title	**Final Requirements and Architecture Specifications**
Work Package	**WP2 – Requirements Engineering and Reference Architecture**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**1~~0~~2/12/2018**
Status	**~~Draft~~Final Version**
Lead Editor	**FRAUNHOFER**
Main Contributors	**Hassan Rasheed, Alexander Schneider(FRAUNHOFER)**

Published by the MONSOON Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723650.

Document History

Version	Date	Author(s)	Description
0.1	15-05-2018	H. Rasheed (FIT)	First Draft with TOC and section 4.3 in chapter 4
0.2	11.06.2018	A. Sheik (ISMB)	Added section for Monitoring Tools
0.3	13.06.2018	A. Bernard (AP)	Update regarding the aluminium domain use cases
0.4	13.06.2018	M. De Pieri (LCEN)	Added contribution to 4.3.1.4
0.5	13.06.2018	V. Bonnivard (PROB)	Added section 4.3.1.7.1
0.6	14.06.2018	D. Ionnanidis (CERTH)	Sections 4.3.1.6 and 4.6 are included
0.7	15.06.2018	R. Rossini (ISMB)	Added section 4.3.2.3
0.8	18.06.2018	P. Bednar (TUK)	Description for various sections in chapter 4
0.9	18.06.2018	G. Charbonnier (CAP)	Added various sections in chapter 4
0.91	20.06.2018	H. Rasheed (FIT)	Consolidation of the content, added summary
0.92	27.06.2018	R. Schlutter (KIMW)	Added Plastic use cases
0.93	28.06.2018	H. Rasheed (FIT)	Incorporated reviewer's comments, some formatting
0.94	9.07.2018	A. Bernard (AP)	Paragraph regarding the global optimization
0.95	20.07.2018	C. Pastrone (ISMB)	Comments and modifications regarding the global optimization
0.99	14.11.2018	A. Schneider (FIT)	Added business case 2 description and further refinements and preparations for release
0.99.1	20.11.2018	H.Rasheed (FIT)	Comments are addressed, missing links for figures are fixed
0.99.2	10.12.2018	H.Rasheed (FIT)	Incorporated comments and added section 4.6
1.0	12.12.2018	H.Rasheed (FIT)	Version ready for submission

Internal Review History

Version	Review Date	Reviewed by	Summary of comments
~~1~~.0.92	28-06-2018	M. De Pieri (LCEN)	Minor wording and language amendments
0.92	27-06-2018	J. Jiménez (UNE)	
0.99.2~~1.0~~	11~~27.06~~12.2018	José Jiménez (UNE)	Minor wording corrections. No technical content to be corrected.
0.99.2	12-12-2018	M. De Pieri (LCEN)	No additional comments after integrations in November

Table of Contents

1 Introduction

This deliverable D2.6Final Requirements and Architecture Specifications describes the updated requirements, the final version of the architecture description of the MONSOON platform, including stakeholders which might be relevant for this project and the platform, as well as a scenario description for the selected use cases in the aluminium and plastics domain.

1.1 Scope

Work package 2 handles Requirements Engineering and the Reference Architecture and deals with the different phases of an evolutionary requirements engineering process, which include user requirements engineering and refinement, architecture design specification and refinement, and model descriptions and development. We follow an iterative approach which means that in this series of deliverables, an initial and an updated version of requirements and architecture are described.

The deliverable is divided into several chapters, chapter 2 gives a short overview of the identified stakeholders, and chapter 3 describes the methodology and principles used for the human centred development approach and the implementation in the MONSOON project.~~In the appendix there is also a list of the initial requirements.~~[JAJC1]
Chapter 4 first describes the methodology for creating an architecture description and then, applying these methods, describes the architecture of the MONSOON software platform.

1.2 Related documents

ID	Title	Reference	Version	Date (in due month)
[RD.1]	Updated Process Industry Domain Analysis and Use Cases	D2.3	1.0	M15
[RD.2]	Initial Requirements and Architecture Specifications	D2.5	1.0	M6
[RD.3]	Updated Real Ttime Communications Framework	D3.2	1.0	M14
[RD.4]	Updated Virtual Process Iindustries Resources Adaptation	D3.5	1.0	M14
[RD.5]	Initial Runtime Container	D3.7	1.0	M14
[RD.6]	Updated Big Data Storage and Analytics Platform	D4.4	1.0	M14
[RD.7]	Initial Multi-Sscale Model based Development Environment	D4.6	1.0	M14
[RD.8]	Initial Integrated MONSOON Platform	D6.2	1.0	M17
[RD.9]	Report on the standardization landscape and applicable standards	D8.5	1.0	M6
[RD.10]	Initial Online and deep machine learning techniques	D5.3	1.0	M15

2 Stakeholders

The following section gives an overview to the different stakeholders and roles in the Aluminium and Plastic domains that are considered relevant for MONSOON. The stakeholders and roles mentioned here are relevant because they are either involved in a part of the process which MONSOON aims to support, or could be potential end users of the platform or are considered sources of knowledge that could support the creation of scenarios.

Table 1 - Stakeholders Aluminium Domain

Stakeholder / Role Aluminium Domain	Description
Global process manager and team	Responsible for supporting overall process decision (structural changes, investments, etc.) – translate executive orientations into process decisions and defining technical support prioritization between smelters.
Plant process manager	Responsible for the global process optimization and robustness of one smelter. He also manages process interactions and aims to minimize process inconsistencies between departments.
Process coordinator	Responsible for defining process adjustment tactics for ~~the~~ one area within the smelter (e.g. paste plant). He is also in charge of monitoring and addressing long term trends and making the corresponding decisions.
Day process supervisors	Responsible for managing trends and process drifts for one part of the process at day to week timescale. He also can take decisions based on predictive analysis in relation to the performance of its area.
On-shift area process supervisors	Responsible for doing process adjustments related to anomalies, making sure that the schedule and targets are being achieved.

Table 2 - Stakeholders Plastics Domain

Stakeholder / Role Plastic domain~~:~~	Description
Industrial engineer	Responsible for the planning phase as he is in charge of defining the parameters of the machines in accordance to customer requirements, targets and particular characteristics of the plastic ~~pieces~~ parts that will be produced.
Production Manager	In charge of the production flow and responsible for analysing the performance of equipment and production in order to address the necessary maintenance of equipment in his area of the plant.
Line Supervisor	Responsible for managing the operators team and support them with more detailed knowledge when finding solutions

	for anomalies in the process.
Machine Operator	Responsible for making sure that the machines and production of parts are running as expected within their corresponding shifts. In some cases, also control the quality of the parts by controlling their size, diameter and resistance.

Table 3 - Other Stakeholders

Stakeholder / Role Others	Description
Software Developer	Developing software which is part of the MONSOON platform.
Operator	Responsible for software deployments and operation of software.
System Administrator	Responsible for setting up and configuring the IT system and making sure that the network and server infrastructure is working properly.
Data Scientist	Responsible for validating data, for developing, training and validation of machine learning algorithms, predicative functions, etc.

3 Requirements

3.1 Purpose, context and scope of this section

The purpose of this section is to give a systematic formalization of an updated set of relevant stakeholder requirements and sub-system requirements. These requirements will guide the developments in the technical work packages

3.2 Methods and Principles of Human-Centred Development

Requirements are descriptions of how the system should behave, application domain information, constraints on the system's operation, or specifications of a system's property or attribute. Requirements engineering is a continuous iterative process driven by an adopted user-centred design (UCD) approach as opposed to a stage or phase realized once in the beginning of a project.

The general approach to requirements gathering involves the following activities:

- Elicitation. Discovering, extracting and learning needs of stakeholders. It includes a domain analysis that helps to identify problems and deficiencies in existing systems, opportunities and general objectives. Scenarios are part of this activity.
- Modelling. Creating models and requirements, looking for a good understanding of them and trying to avoid incompleteness and inconsistency.
- Negotiation and agreement. To establish priorities and to determine the subset of requirements that will be included for the next phase.
- Specification. Requirements expressed in a more precise way, sometimes as a documentation of the external behaviour of the system.

- Verification/Validation. Determining the consistency, completeness and suitability of the requirements. It could be done by means of static testing (using regular reviews, walkthroughs or other techniques) and prototyping.
- Evolution and management. The requirements are modified to include corrections and to answer to environmental changes or new objectives. It is important to ensure that requirement changes do not produce a large impact on other requirements. Requirement management means to face those modifications properly, to plan requirement identification and to ensure traceability (source, requirements and design traceability).

It is important to underline that most of these activities are performed in parallel.

From the scenarios and storylines, a systematic formalisation of all relevant user requirements and subsystems requirements will be derived and detailed in the different iteration cycles through user workshops and interviews.

We started with a common approach to elicit requirements based on singular, well defined requirements based on the Volere schema [RD.2] in a software tool called JIRA.

This proved to be valuable but as the project progressed we learned that due to the complexities involved there was a need to structure them in a better way. The main reason for this was that it was hard to maintain the relations between the single requirements and that a lot of context was missing especially for people getting involved in e.g. implementation tasks what to focus on. The coherence of the requirements was lost.

That´s why we decided to introduce use case descriptions [Cockburn, A., 2000] in addition to the scenarios and business case descriptions that we already had.

This provides a workflow oriented view on the tasks that should be supported by the MONSOON platform and tools and by that we are able to give context and coherence to the requirements. In this way it also becomes easier to implement functionalities which was an added benefit. The use cases will be further refined as the implementations will continue as we have chosen to follow an iterative process in the MONSOON project.

The use cases have been documented in [RD.1] and have influenced the final version of the architecture as described below.

4 Architecture

4.1 Methodology

This section presents the key concepts related to the methodology used to develop the architectural design of the software system developed in MONSOON. We follow standards and best practices as described in the following subchapters. In addition, there have been workshops to discuss, produce and refine the architecture design.

4.1.1 Software Architecture Design Fundamentals

The process used is based on IEEE 1471 "Recommended Practice for Architectural Description for Software-Intensive Systems" [IEEE1471, 2000] and ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description" [IEEE 42010, 2011], by which it was superseded. The latter establishes a methodology for the architectural description (AD) of software-intensive systems.

It implies a process which includes the following steps:

- Identify and record the stakeholders for the architecture and the system of interest
- Identify the architecture-related concerns of those stakeholders
- Select and document a set of architecture viewpoints which can address the stakeholder concerns
- Create architecture views (one view for each viewpoint) which contain the architectural models
- Analyse consistency of the views
- Record rationales for architectural choices taken

Viewpoints are collections of patterns, templates and conventions for constructing one type of view. One example is the functional viewpoint (and therefore a functional view) which contains all functions that the system should perform, the responsibilities and interfaces of the functional elements and the relationship between them. These functions can be described using UML diagrams. Moreover, it also describes which stakeholders need to be involved and how to apply their needs in the architecture as stated in the "architectural perspectives" chapter by Rozanski and Woods [Rozanski & Woods, 2005].

4.1.2 Definitions

The following definitions are on the basis of the ISO/IEC/IEEE 42010:2011 [IEEE 42010, 2011] standard and the definitions provided by Rozanski and Woods [Rozanski & Woods, 2005].

Architecture: Comprises of the concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"

Architectural Description: A collection of products to document an architecture

Stakeholder: An individual, group or organization that has at least one concern relating to the system-of-interest

Concern: An interest in a system which is relevant to one or more stakeholders. It might be a requirement (functional or non-functional) or an objective a stakeholder has regarding the system.

View: A set of models and descriptions representing a system or part of a system from the perspective of a related set of concerns

Viewpoint: Collection of patterns, templates and conventions for constructing one type of view

Model: A simplified representation of an aspect of the architecture, could be in form of a UML diagram

The relationships between these concepts and the system-of-interests are shown in Figure14.

According to the specification of the ISO/IEC/IEEE 42010:2011 standard the main concepts, architecture view and architecture viewpoint, are defined as follows.

- Architecture viewpoint: "Work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns"
- Architecture view: "A representation of a whole system from the perspective of a related set of concerns."

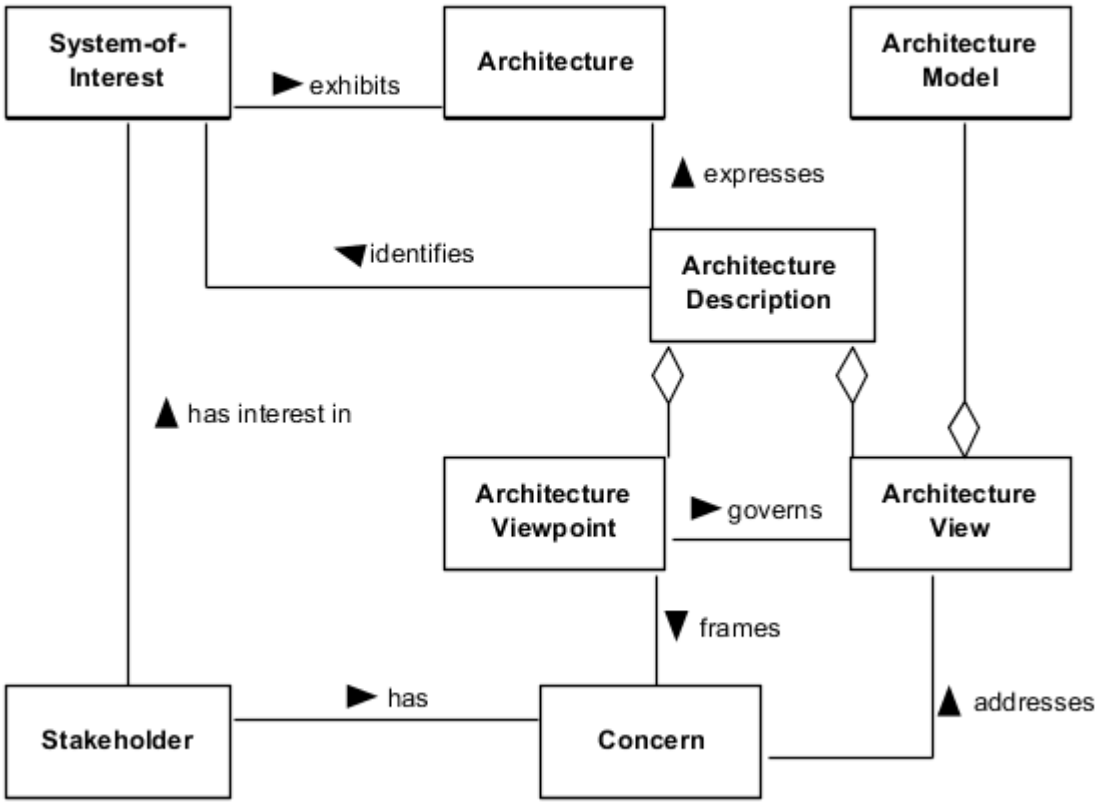


Figure 1 - Architecture description concepts (Adapted from ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description" [IEEE 42010, 2011])

A viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address e.g. Functional viewpoint or Deployment Viewpoint. A view conforms to a viewpoint and so communicates the resolution of a number of concerns (and a resolution of a concern may be communicated in a number of views).

According to [Rozanski & Woods, 2005] using vision and point of view to describe the system architecture can bring many benefits such as:

- Separation of concerns: Separating different models of a system into distinct (but related) descriptions helps the design, analysis and communication processes by allowing designers to focus on each aspect separately.
- Communication with stakeholder groups: Different stakeholder groups can be guided quickly to different parts of the AD based on their particular concerns, and each view can be represented using language and notation appropriated to the knowledge, expertise, and concerns of the intended readership.
- Managements of complexity: Treating each significant aspect of the system separately, the architecture can focus on each in turn and so help conquer the complexity resulting from their combination.
- Improved developer focus: Separating into different views those aspects of the system that are particularly important to the development team, you help ensure that the right system gets built.

4.1.3 Software Architecture Design Process

In a software architecture design process there are several principles we should follow to ensure a high quality of the architecture design. The different stakeholders should be engaged and their concerns taken into account. There might be conflicting or incompatible concerns from different stakeholders which must be dealt with. Also an effective way to communicate decisions and solutions should be implemented and the whole architecture design process should be flexible and pragmatic to be able to deal with the changing requirements and the iterative approach in this project. Also the process should be technology-neutral.

4.1.3.1 Architecture Definition Activities

Rozanski and Woods have based the architectural design process on the following definition:
"Architecture Definition is a process by which stakeholder needs and concerns are captured, an architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." [Rozanski & Woods, 2005] (p.56) the foundation for our process is the IEEE 1471 standard and we have used the process proposed by Rozanski and Woods [Rozanski & Woods, 2005] which is aligned to this standard:

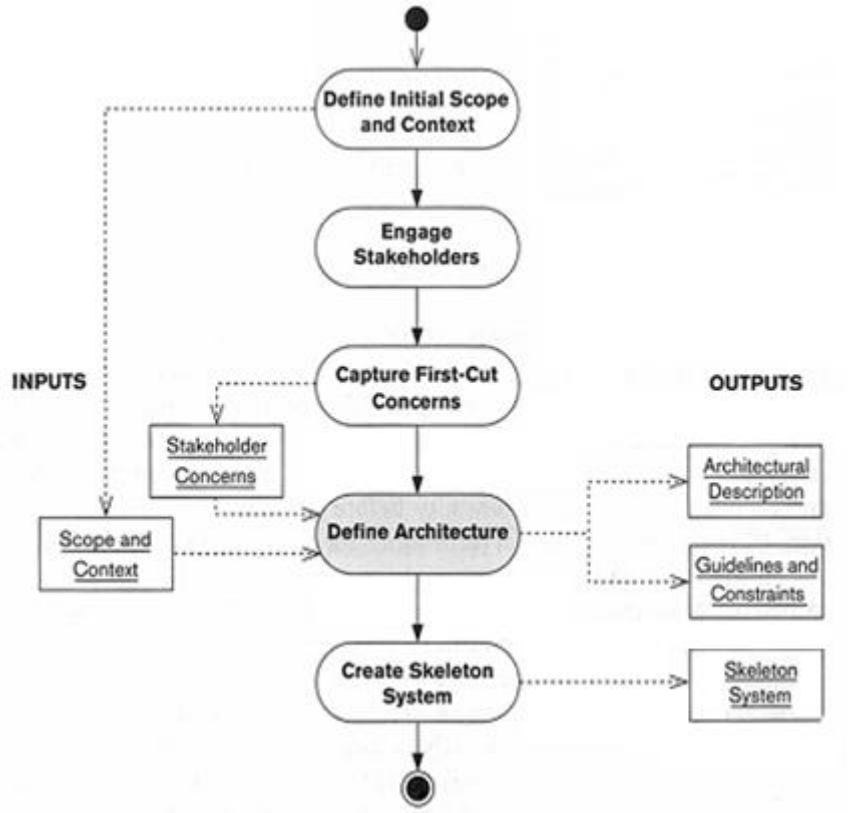


Figure 2 - Activities supporting architecture definition [Rozanski & Woods, 2005]

The process to be implemented in the MONSOON project should reflect this approach. We start with the initial scope and context and the involvement of stakeholders in the process of the scenario development and use cases in WP2 and the subsequent requirements process. The stakeholders are included to express their needs and desires and capture quality properties that increase the success of the platform. The requirements from workshops, vision scenarios, and to-be use cases together with requirements from other sources are the input for the current architecture design phase where we create a first draft of the architectural description. Based on this architectural description, the first prototype should be created, which can be seen as a skeleton system with minimal functionality developed above that. These development efforts reveal some experiences and lessons learnt which in turn constitute a valuable source for the derivation of additional requirements and the revision of already existing ones.

4.1.3.2 Viewpoint Catalogue

The project decided on the following viewpoints from which the views of the architectural document are derived.

- Context viewpoint: The context viewpoint describes interactions, relationships and as well dependencies between the system-of-interest and its environment. The environment includes those external entities with which the system interacts, such as other systems, users, or developers.
- Functional viewpoint: This viewpoint describes the functional elements needed to meet the key requirements of the architecture. It will present proposals in a descriptive way and UML diagrams will assist in the understanding of the proposal. It will describe responsibilities, interfaces, and interactions between the functional elements.
- Information viewpoint: The information viewpoint describes the data models and the data flow as well as the distribution. The viewpoint also defines which data will be stored and where. The description of where data will be manipulated is also part of this viewpoint.
- Deployment viewpoint: This viewpoint describes how and where the system will be deployed and what dependencies exist, considering for example hardware requirements and physical restraints. If there are technology compatibility issues, these can be addressed in this viewpoint as well.
- Development viewpoint: This is the viewpoint which addresses concerns from the developers' point of view. It describes how the software development process is supported, e.g. what conventions should be followed and how the artefact management will look like.

To address quality properties and cross-cutting concerns, architectural perspectives will be used. A typical example is security: it should be considered how the data is secured and which functional elements need to be protected. Another perspective which is interesting is availability, e.g. of the hardware, the functionalelements or the data.

4.2 Context View

The MONSOON vision is to provide Process Industries with dependable tools to help improve in the efficient use and re-use of raw resources and energy. MONSOON aims to establish a data-driven methodology supporting the exploitation of optimization potentials by applying multi-scale model based predictive controls in production processes. MONSOON features harmonized site-wide dynamic models and builds upon the concept of the cross-sectorial data lab, a collaborative environment where high amounts of data from multiple sites are collected and processed in a scalable way. The data lab enables multidisciplinary collaboration of experts allowing teams to jointly model, develop and evaluate distributed controls in rapid and cost-effective way, create predictive functions with the help of machine learning algorithms, or do simulations. It is important to highlight two main parts of the MONSOON framework: the "Real Time Plant Operation Platform" and the "Cross Sectorial Data Lab". These components are coloured in green in Figure 3**Error! Reference source not found.**. These two parts have different users though: the cross sectorial data lab can be used by data scientist or the global process manager and his/her team, while Real Time Plant Operation Platform is used during runtime and can potentially be used by employees working on the shop floor.

In order for the MONSOON platform to work and to be used, it needs to interface with the software used in the production site. Also companies' and national regulations must be compliedwith and standards should be followed (see D8.5 for more information about standards).

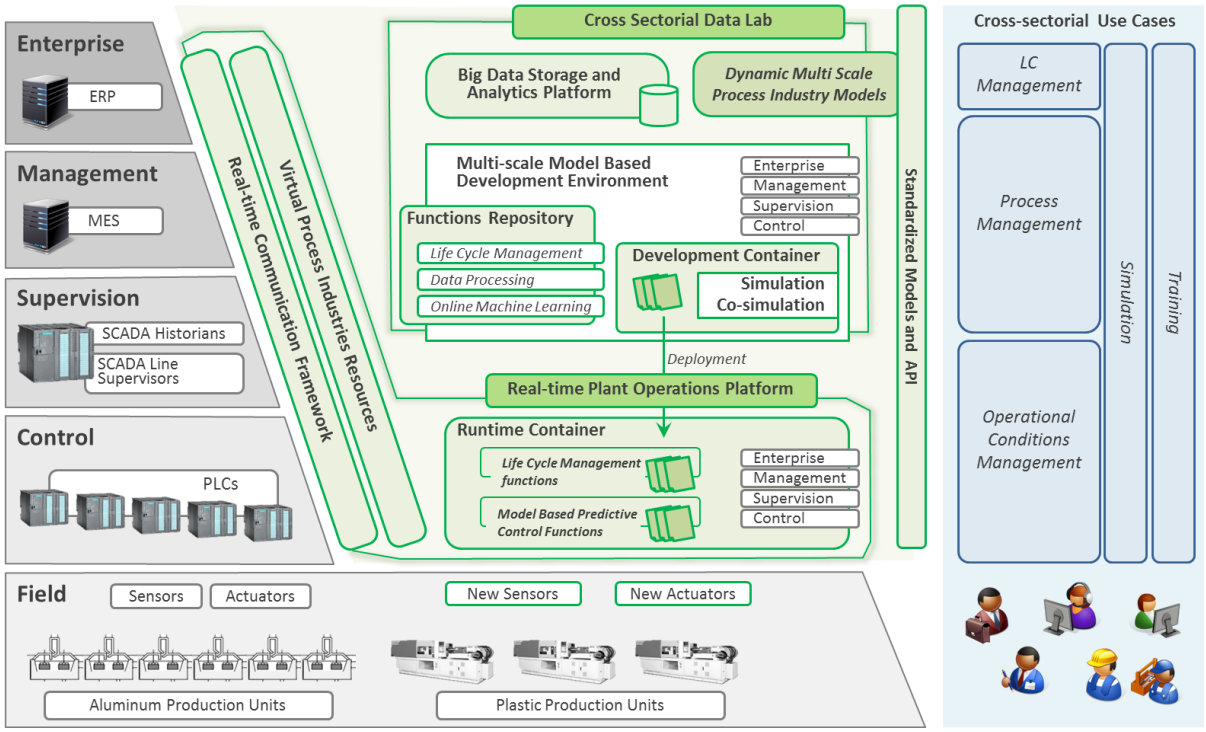


Figure 3 - High level architecture overview

4.3 Functional View

The MONSOON platform consists of two major components: the cross sectorial data lab and the real-time operations platform as can be seen in Figure 3. Each of them contains several subcomponents which are connected in various ways. For example, data from the real-time operations platform is fed to the big data storage which is contained in the cross sectorial data lab and contents from the cross sectorial data lab are deployed in the runtime container which is part of the real-time operations platform. There are also components which are needed by both the main parts of the system, such as the dynamic multi scale process industry models, the tool for trend analysis and the resource optimization toolkit. The components and theirresponsibilities, interfaces, and interactions with other~~the~~ elements will be described in detail in the following sections.

4.3.1 Cross Sectorial Data Lab

The Data Lab provides a collaborative environment where high amounts of data from multiple sites, and possibly from multitude of industry sectors, are collected, stored and processed in a scalable way. It enables multidisciplinary collaboration of experts allowing teams to jointly model, develop and evaluate distributed controls in rapid and cost-effective way. The Data Lab eases the definition of predictive control and life cycle management functions, allowing to work in a simulated environment or to exploit co-simulation by mixing stored data with data flowing in real-time from the real systems. The architecture of the Cross Sectorial Data Lab platform is depicted in Figure 4 and consist of the following main components:

- Big Data Storage and Analytics Platform
- Development Tools
- Semantic Framework
- Life Cycle Management Plugin
- Simulation Toolkit
- Resource Optimization Toolkit
- Function Repository

Additionally, the overall architecture defines a common Configuration and Monitoring Framework used for the monitoring and configuration of the resources and services on both deployments. All these components are described in more detail in the following subsections.

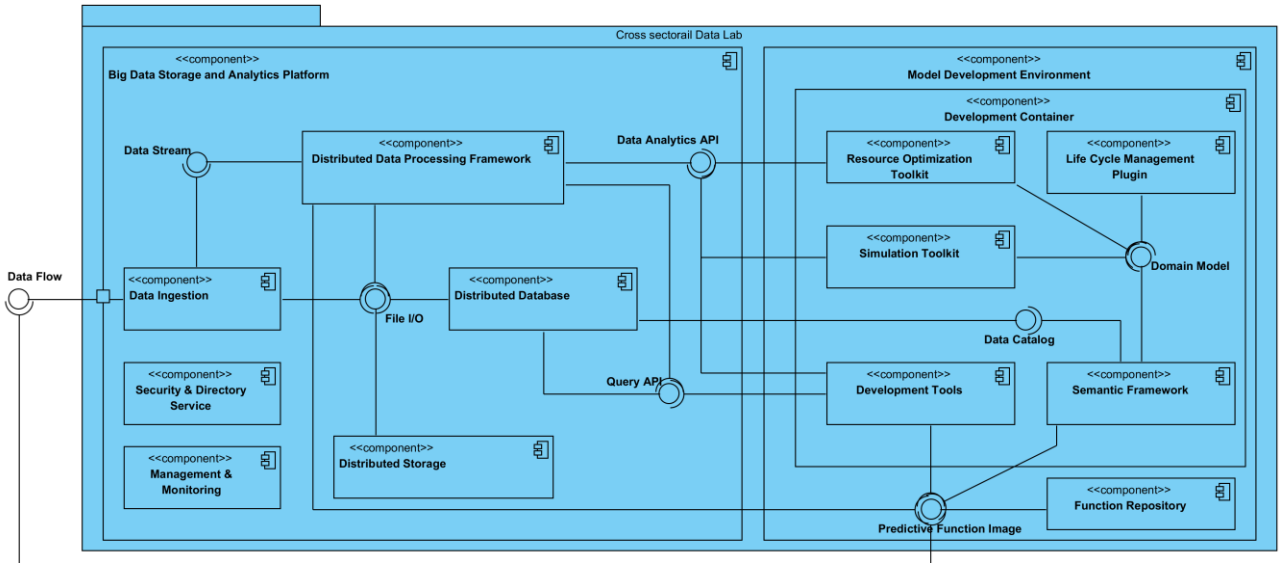


Figure 4 - Functional view of Data Lab

4.3.1.1 Big Data Storageand Analytics Platform

The Big Data Storage and Analytics Platform provides resources and functionalities for storage as well as batch and real-time processing of the operational data from multiple site characterized as Big Data. The platform combines and orchestrates existing technologies from the Big Data and Analytic landscape and sets a distributed and scalable run-time infrastructure for the developed data analytics methods. It provides main integration interfaces between the site Operational Platform and the cloud Data Lab platform and the programming interfaces for the implementation of the data intensive analytics methods. The Big Data Storage and Analytics Platform consist of the following sub-components:

- Distributed Storage: provides a reliable, scalable file system with similar interfaces and semantics to access data as local file systems.
- Distributed Database: provides a structured view of the data stored in the platform and support different interfaces for accessing the data.
- Distributed Data Processing Framework: allows the execution of applications in multiple nodes in order to retrieve, classify or transform the arriving data. The framework provides Data Analytics APIs for processing large datasets via parallel and distributed computations.
- Data Ingestion: implements an interface for real-time communication between the Data Lab and Operation platforms. It also supports batch uploading of the historical data between the Data Lab and Operation platform.
- Security & Directory Service: provides user management and content authorization capabilities for the platform services.
- Management & Monitoring: provides the management, monitoring and provisioning of the platform services on the hosted environment.

4.3.1.2 Development Tools

The Development Tools provide the main collaborative and interactive interface for data engineers, data analysts and data scientists to execute and interact with the data processing workflows running on the Data Lab platform. Using the provided interface, data scientists can organize, execute and share data, and code and visualize results without referring to the internal details of the underlying Data Lab cluster.

The graphical web-based interface of Development Tools is in the form of analytical notebooks. Notebook consists of notes – the paragraphs of code that can be executed directly in a browser window. Results are visualized directly in the environment itself. Data scientists can divide the code into the several logically grouped sections. Scripts then connect to multiple data-processing back-end interpreters. An interpreter is a plug-in which enables a language or data-processing back-end. It enables the data scientists to use multiple technologies by invocation of specific interpreter for particular language or environment directly in the notebook.

In order to facilitate development during the whole predictive function life cycle, ~~I.E~~i.e., apply data-cleaning methods, train models, evaluate their performances, and manage all the I/O of their application (handling loggers, handling file write/read operation) we encourage every data-scientists and data-engineer to contribute to a shared development library:

- During first exploratory phase, data-wrangling is often necessary and can be very specific to the task on hand. When such methods are not yet implemented into the shared Python library, the data scientists are expected to add those functionalities, so that all their work is reproducible.
- When exploring datasets, the scientists use various models to fit the data and write those result downs using notebooks, which is a very handy format to share the results to all the other members of the project. When the models perform sufficiently well, the code they need to function can be added to the MONSOON analysis Python library. The models can then be fitted any data with required shape by all users of the library.
- Various tools implemented in the development library can come in handy when creating predictive function as a Python script. In order to make the packaging of those scripts easier, command line tools are available in order to make sure that all projects have the same structures, regardless of their use case.
- Finally, a private P~~p~~ython repository server is available for MONSOON projects, with native support for official Python package manager. It facilitates the distribution of dependencies across partners.

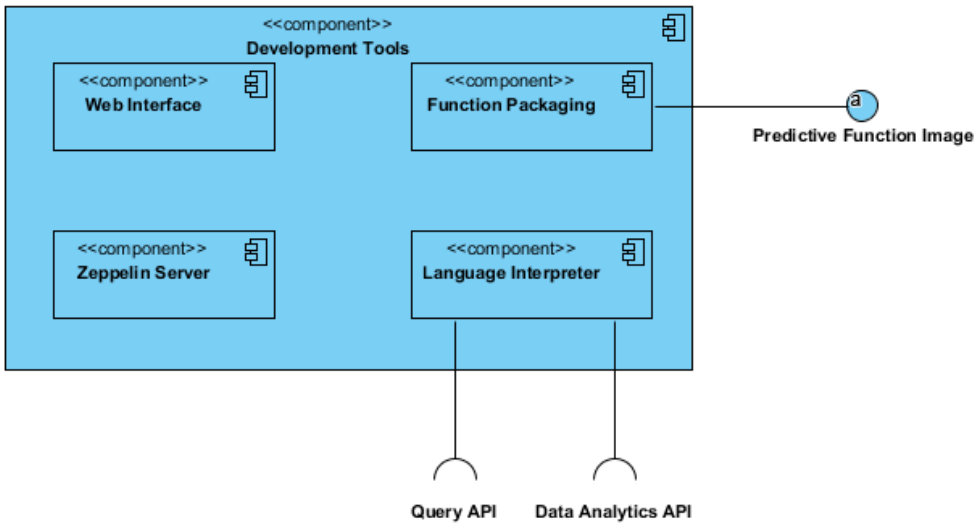


Figure 5 - Development Tools

Predictive functions created in the Development Tool will be packaged together with data pre-processing scripts in the Predictive Function Images. These images will contain all dependencies, can cover entire data analytics pipeline (pre-processing, models building, evaluation). Containers also enable more straightforward installation and replacement of containers. Predictive Function Images and sent into the Predictive Function Repository where they will be stored and could be deployed into Runtime Container.

4.3.1.3 Semantic Framework

The main goal of semantic modelling for the MONSOON project is to provide a common communication language between domain experts and stakeholders and data scientists. On one ~~hand~~ side, data scientists need a deep knowledge about the business objectives and modelled phenomena acquired from the stakeholders and domain experts; and on the other side, stakeholders and domain experts need to interpret the results of the analysis. In order to make this communication more effective, Semantic Framework will define various formalisms and graphical notations. Semantic Framework will then provide user interfaces for creation and editing of the semantic models and web service interface, which will allow to use knowledge expressed in the semantic models for optimization and simulations of the production processes.

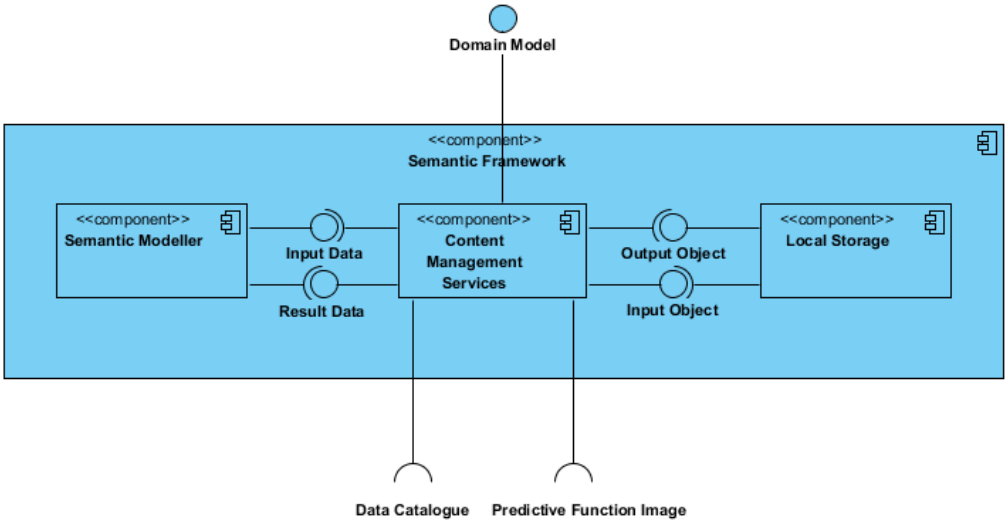


Figure 6 - Semantic Framework

The semantic models will cover the following main concepts:

- Decomposition of the production process to the phases
- Overall business KPIs such as energy and material consumption, product quality or environmental impact and mapping which process phase influences which KPI.
- Logical view of data elements related to the particular production phase including measurements, control signals or diagnostic events.
- Physical view which maps logical data elements to the physical data storage.
- Logical view for predictive functions which will specify data elements for input attributes and output (predicted) values. Additionally, this view will specify relation between performance of the predictive function (e.g. accuracy estimated on the test/validation data set) and values of KPIs.
- Physical view of predictive functions, which will provide information about the training data, algorithms and settings used for the training of the particular predictive model.

Semantic framework consists from three main components:

- Semantic Modeller - all boundary classes that represent the application screens and graphical interfaces. Its main purpose is to manage, visualize and manually create (edit) semantic models.
- Content Management Services (CMS) - contains all necessary components to create, edit, store and filter semantic model information via REST API, which accept data in JSON format. Its main purpose is to communicate with Semantic Modeller. Semantic modeller uses REST API to send semantic models data which need to be saved, edited, deleted using Input Data interface and to receive data for visualization using Result Data interface. However, thanks to REST API interface it is also able to communicate with external applications, so we are also able to automatically feed semantic models data using Data catalogue interface or Predictive Function Image interface to CMS. Semantic framework is also capable to access data from other applications using Domain Model interface.

- Local Storage – is a component used to store data about all semantic models. It uses document-oriented database (MongoDB), which stores data using JSON-like documents with schemas. To communicate with CMS it used Output Object interface and Input Object interface.

4.3.1.4 Lifecycle Management Plugin

Life cycle management plugin aims at assessing the environmental performance of the investigated processes. This tool is supposed to support HSE department in monitoring and quantifying the environmental burdens arising from production activity; results can be used for a wide range of purposes, from CSR activities to internal R&D or communication activities.

From a general workflow perspective, process data collected by monitoring equipment are grouped, pre-processed and elaborated to obtain as an output a set of environmental KPIs[1]. Type of aggregation and pre-processing depends on the shape and nature of raw data and on the kind of application which is investigated.
The KPIs are then stored in a specific database, and they can be visualized using a dedicated dashboard. The dashboard updates automatically and allows to browse the results, providing additional details about KPIs breakdown contribution and past trends.

Next section describes more in detail technical features of the components that have been implemented to ensure integration of the LC plugin with the MONSOON architecture.

4.3.1.4.1 Integration with Operational Platform

The Life Cycle management plugin interacts the same way any other predictive function would do with the runtime container deployed on production site. As every runtime container component, the LC plugin is deployed as a Docker container. This container, domain specific, is created from a base Docker image common to all domains, available from the function repository in the Data Lab. In order to integrate seamlessly with the runtime container, the LC plugin share several directories of its file system with the host server file system thanks to Docker volumes. It means that the plugin is able to read and write in specific directories of the host server. This feature allows the Data Orchestrator component to place input data into the input directory of the LC plugin. A time-based scheduler triggers the execution of the LC plugin at the end of each shift of production. Shift duration may vary between domains.

As mentioned above, results of the plugin are stored in the Runtime Container Time Series database. Indeed, using a custom library written in Python, we can read, update, insert or delete entries in the database directly from the Life Cycle Management Plugin. As soon as results are stored in the database, they can be downloaded thanks to the HTTP Rest API of the distributed database, and end users can look at the associated visualisations with the operational data visualization dashboard tool.

End users may also extract results from web interface (without programmatic knowledge) thanks to the Time Series Database graphical interface and the operational data visualization dashboard (Download as CSV or JSON)

4.3.1.5 Simulation Toolkit

The main goal of the Simulation Toolkit is to support validation and deployment of the predictive function in order to optimize overall key-performance indicators defined for the production process. The validation is based on the a) estimation of accuracy statistics of predictive function (e.g. ROC curves, confidence tables, etc.) on the specified validation set of historical data or real-time data stream of operation data and b) computation of changes for key-performance indicators (KPIs) from the accuracy statistics defined for the overall production process. This estimation of overall impacts can be used to test various "what if" scenarios

[1] Type and amount of KPIs are depending on the investigated application; their description is out of the scope of this deliverable but can be found in *D 7.1 – Initial Evaluation Framework specifications*

or for the automatic discrete optimization of the production process by finding the optimal combination of predictive functions for various process phases.

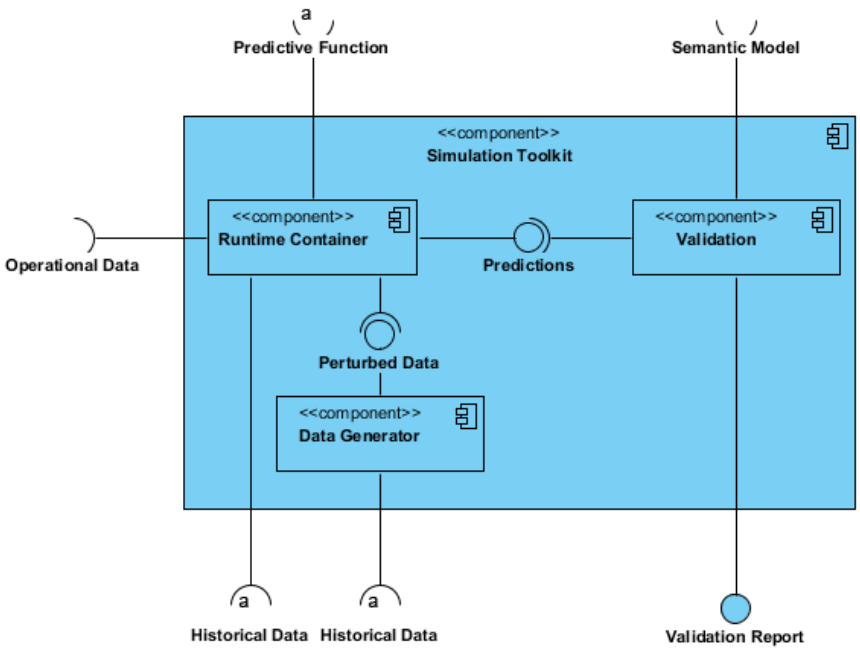


Figure 7 - Simulation Toolkit

Simulation framework loads selected Predictive Function from the Function Image Repository and create a new instance of the function in the Run-time container running in the Data lab environment. Simulation framework is then used to stream selected validation data set to the Run-time container and get the function's predicted values. Simulation framework computes selected evaluation statistics (e.g. Error rate, precision, recall, ROC curve, etc.) And store the statistics in the validation report. Semantic framework loads the validation report and compute KPIs using the parametrized cost matrices and allows visualization of the KPIs in the context of the production process in order to compare multiple predictive functions applied to the same production step.

Besides the performance of the predictive function, Simulation Toolkit will use information about the "costs" associated with the particular input attribute, which reflect how difficult is to obtain particular data (i.e. measure, integrate etc.). Input costs and availability in the specified production environment will put additional constrains for the process optimization.

4.3.1.6 **Resource Optimization Toolkit**

The resource optimization toolkit is the component that will combine the information from the data analytics APIs and after processing it, will provide decision support functionalities for optimizing the process. The decision support will be done in two ways, first it will enhance the operator's awareness by providing relevant information and second by proposing actions that can potentially improve the process characteristics. An overview of the toolkit's internal architecture is provided in Figure 8.

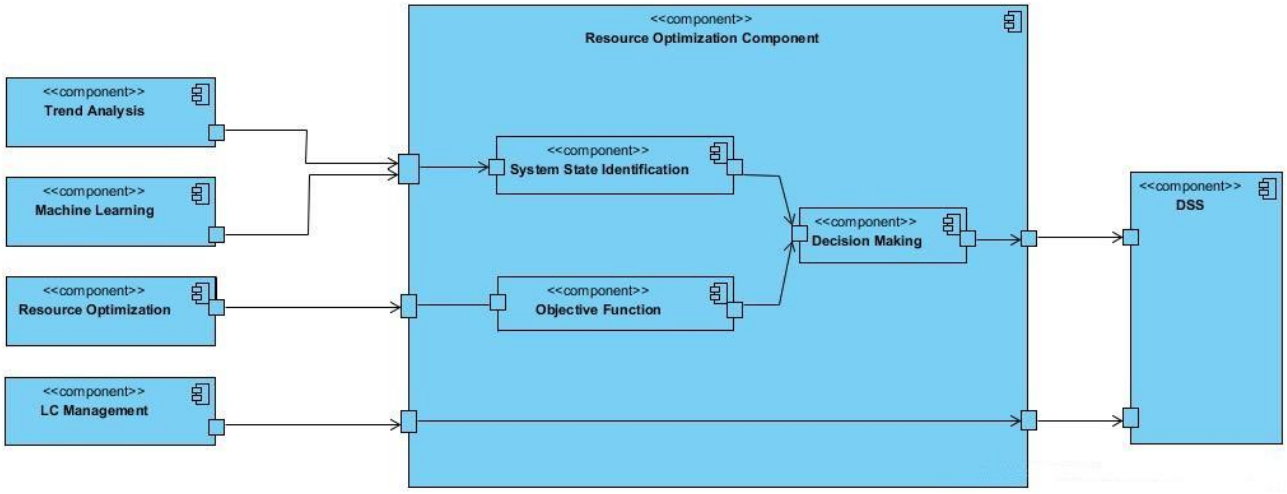


Figure 8 - The resource optimization toolkit

More specifically, information from individual components such as the 1) trend analysis, 2) equipment misbehaviour/stoppage and the 3) lifecycle management are visualized for the better process understanding from an operator.

In addition, inputs from the above components are used to form and solve an optimization problem, which results are provided as a potential optimization action. Information such as the trend analysis and the equipment misbehaviour/stoppage are inputs to the **system state identification** (see Figure 4) which in combination with the **objective function component**, formulate an optimization problem, solved in the **decision making component**. The objective function component provides forecasting capabilities for the variables of interest. The optimization problem is solved within the decision making component and an optimal action is proposed.

4.3.1.7 Function Repository

The Predictive Function repository is in charge of the storage of the ready-to-be-deployed analytics functions. These functions will be packaged as Docker images, and pulled from the Runtime Container. They will be run from there, in a real-time fashion. The Predictive Function repository must always contain the latest version of the analytics function, obtained after a successful build (e.g. after all tests are passed). Every time a function is modified (e.g. after an automatic re-training of a model, or after improvements of the model), and is ready to be deployed, it must be stored in the Predictive Function repository.

4.3.1.7.1 Predictive Function

Predictive functions are packaged in the developer containers as Docker images[2]. The Docker image mainly contains Prediction Algorithm that performs machine learning tasks such as classification, regression or clustering using pre-trained models. These images are deployed in the runtime container. The predictive functions in Docker containers connect to databases and fetch the required data, perform pre-processing and machine learning tasks and write the prediction results back to the databases. They also can store intermediate trained parameters for future use. There can be more than one predictive functions and database per runtime container. Each of these instances can serve different purposes. The predictive functions are stateless and they can be deployed at any time anywhere as long as the predictive functions can reach the databases for accessing and storing the data.

[2] https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

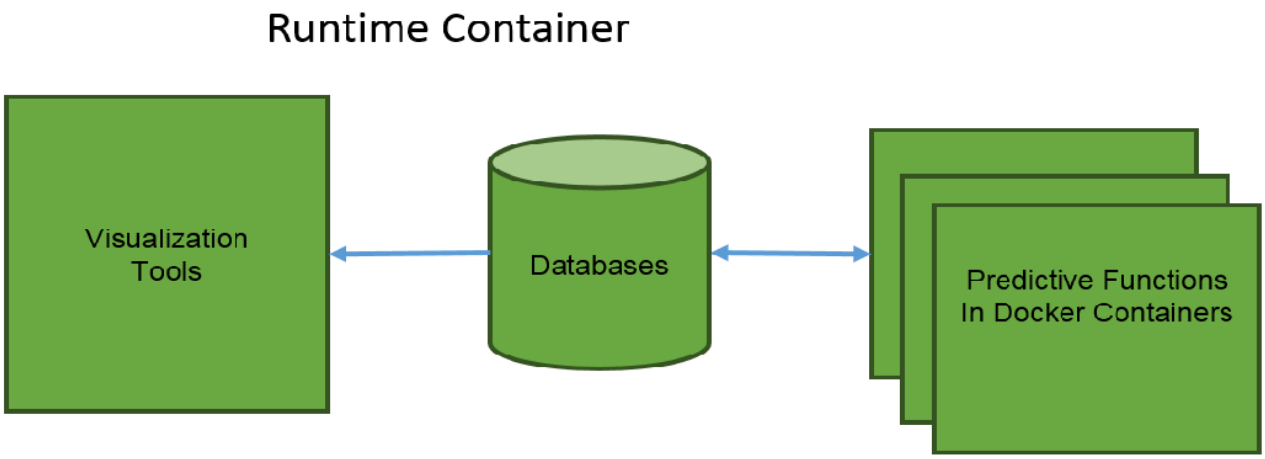


Figure 9 - Predictive Function

4.3.2 Real-time Plant Operations Platform

This platform is in charge to (a) communicate with the heterogeneous existing systems already used in process industries, (b) support data collection, storage, and interaction with the process industry systems respecting required real-time / dependability constraints and under the assumed data intensive conditions. The relevant information acquired from the plant is communicated to the "Cross Sectorial Data Lab".

The architecture of the Plant Operations Platform is depicted in Figure 10. It consists of the following main components:

- Real-time Communication Framework
- Virtual Process Industries Resources Adapter
- Runtime Container

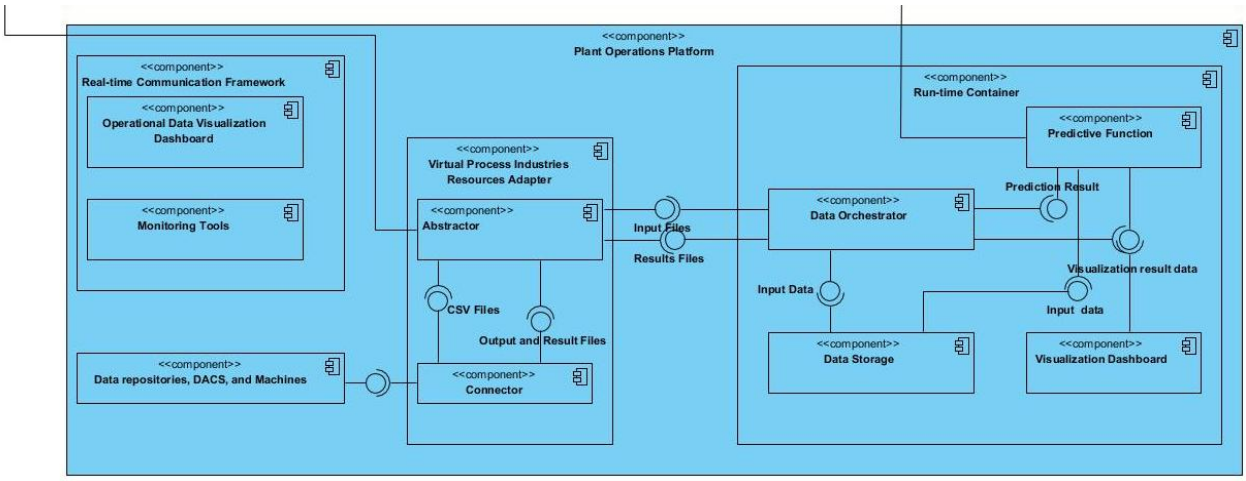


Figure 10 - Real-time Plant Operations Platform

In the following subsections, the main components of this platform are presented.

4.3.2.1 Monitoring Tools

Monitoring of Plant wide resources in the MONSOON project relates to the detection of problems and faults in the systems that compromise the components of the project itself. In this respect, the following presents the list of software tools, used in the project along with their purpose.

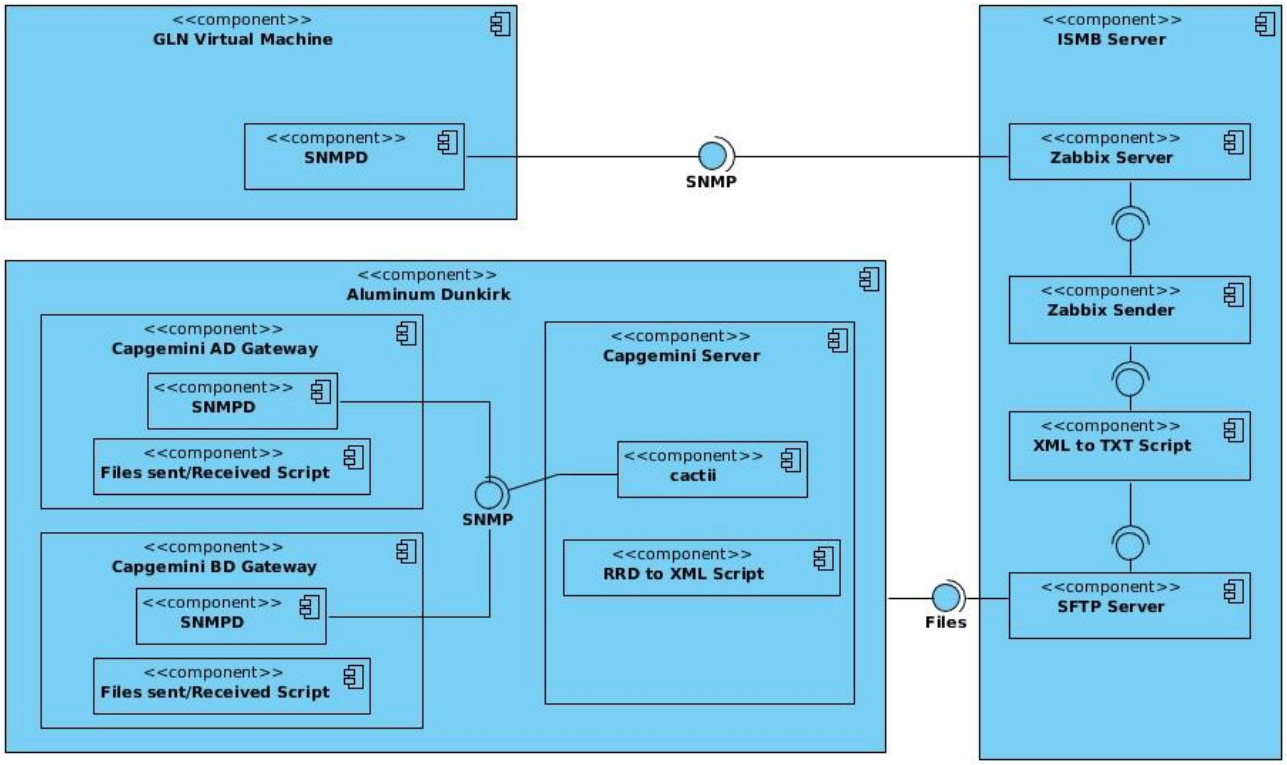


Figure 11 - Real-time Communication Framework (Monitoring of managed devices)

4.3.2.1.1 SNMPD

SNMPD is an agent to receive process and respond to Simple Network Management Protocol (SNMP) requests coming from management software. SNMP is a protocol to collect and organize information about managed devices on IP networks. The protocol defines a management station and a number of managed devices. The management station polls the managed devices at configured intervals for various metrics, which indicate the health of each managed device. In this project SNMPD is the software which acts as the SNMP agent, and is installed at all managed devices as shown in Figure 11.

4.3.2.1.2 Cactii

Cactii[3] is an open source tool, released under GNU GPL, for network monitoring and graphically representing the collected network data. Its core depends on a Round Robin Database for data storage purposes. The RRD in turn is a data logging mechanism for time series data, based on a circular database concept to enable a constant footprint of the database. Finally, Cactii also provides visualization of the data stored in the RRD using a web-based interface. Cactii acts as the SNMP management and visualization software for the aluminium domain, and it is installed at Capgemini premises as depicted in Figure 11.

4.3.2.1.3 Zabbix

Zabbix[4] is an open source enterprise level product for monitoring the availability and performance of IT infrastructure components. It is released under the GNU GPL V2. The software is created specifically for monitoring and tracking of network related services, resources and hardware.In the scope of this project Zabbix acts as the management and visualization software of all the managed devices in the project. In this context, it acts as (i) the SNMP management software for the Plastic Domain, (ii) the monitored data collection tool for the Aluminium domain and (iii) the data visualization and alarm generation tool for both ~~the~~ domains – in the case of malfunctioning of the managed devices. The Zabbix server is installed at ISMB

[3] http://www.cacti.net/
[4] http://www.zabbix.com/

premises, and is able to monitor directly the managed devices in the Plastic domain, while for the Aluminium domain, managed devices are monitored at Capgemini's servers and the monitored data is transferred to the Zabbix server.

4.3.2.1.4 Zabbix Sender

Zabbix sender is a utility within the Zabbix software suite, which allows sending resource monitoring and performance data to the Zabbix server for visualization and processing purposes. The utility allows sending data records stored periodically, stored in textual formats. In the MONSOON platform this utility is mainly used for passive monitoring and resource data collection purposes from the Aluminium Domain. Zabbix sender is installed on the ISMB server, where it receives Zabbix formatted text files, and transfers the collected data to the Zabbix server for visualization and alert purposes. This approach allowed to eradicate the concerns and about security, in particular, related to opening of ports at the Capgemini network.

4.3.2.1.5 SFTP Server

A SFTP server is used as an interface for the transfer of data containing the health of the managed devices in the Aluminium domain. At regular intervals, the transfer of XML files is initiated by the Capgemini server towards the ISMB server, using secure communications for the transfer of this data.

4.3.2.1.6 Shell Scripts

At the ISMB server, some shell scripts have been developed. The purpose of these shell scripts is (i) to convert the data sent by the Capgemini server to the SFTP server, and (ii) to initiate Zabbix sender whenever new files arrive for data ingestion in the Zabbix server database.

4.3.2.2 Operational Data Visualization Dashboard

The purpose of the operational data visualization dashboard is to present the monitored and collected data at a unified place and in a graphical manner, which is easy to understand and grasp. The Zabbix server provides such a feature to perform multiple functions of such a visualization dashboard.

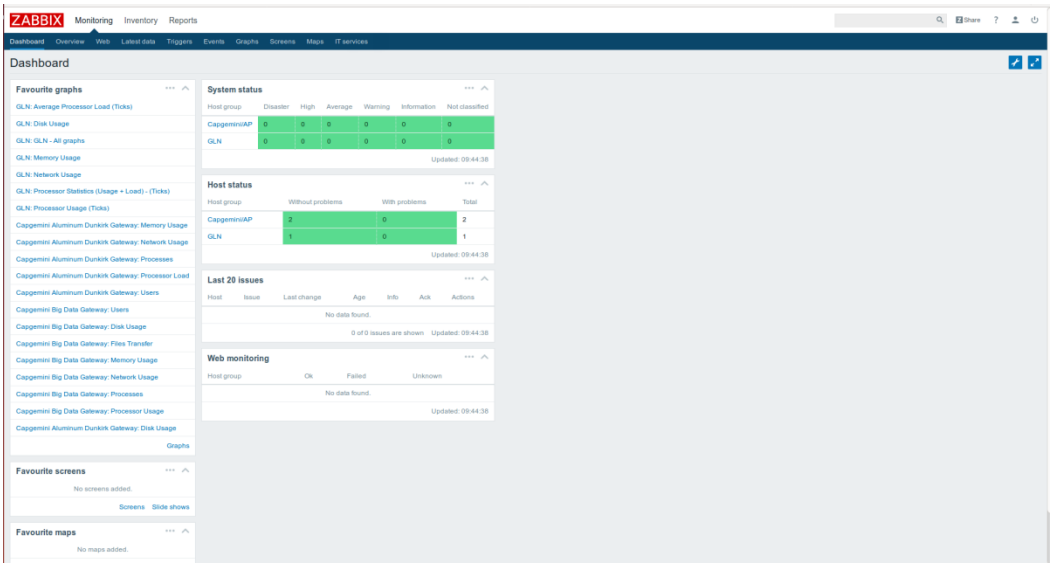


Figure 12 - Operational Data Visualization Dashboard

In particular, apart from data visualization services, the Zabbix dashboard also allows to generate alerts and alarms whenever any of the monitored parameters suggest that the managed device(s) have malfunctioned or are not in an optimal state. Finally, such alerts can also be configured and set up to send notifications in real-time via email or other suitable methods of communications. The dashboard allows to view a complete network architecture map using an interactive interface, such that, it allow an efficient method to identify and locate problem(s), if any, and to inquire about various parameters of the managed devices. The interface of

the Zabbix server is accessible via a web browser and is accessible via the internet. Figure 12 presents a snapshot of the visualization dashboard.

4.3.2.3 Virtual Process Industries Resources Adapter

The main function of the Virtual Process Industries Resources Adapter (VPIRA) is data integration, mediation and routing. The updated overview of the VPIRA architecture is depicted in Figure 13.

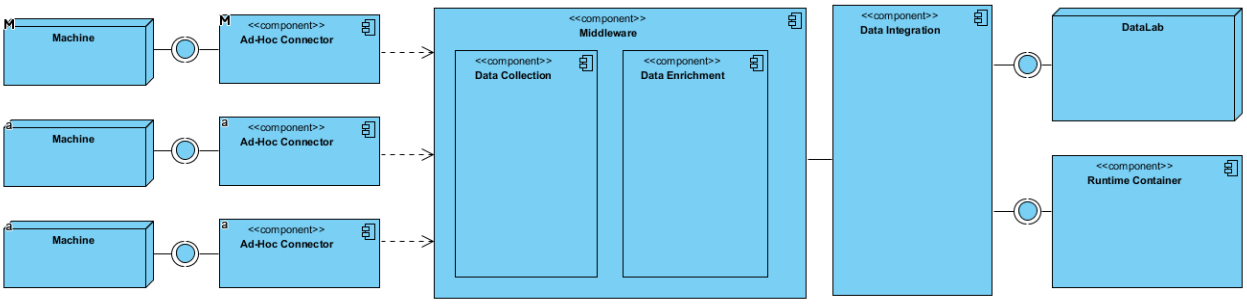


Figure 13 - Virtual Process Industries Resources Adapter

As shown, different connectors will provide data to the VPIRA main component, devoted to the data pre-processing. This module called 'abstractor' is in charge to prepare the data according to the specific request arrived from BigData Storage and Runtime container as well as to send the elaborated data to the BigData Storage' and to the Runtime Container. In particular, Figure 14 summarizes the current data flow about VPIRA for both Plastic and Aluminium domain highlighting the following main layers.

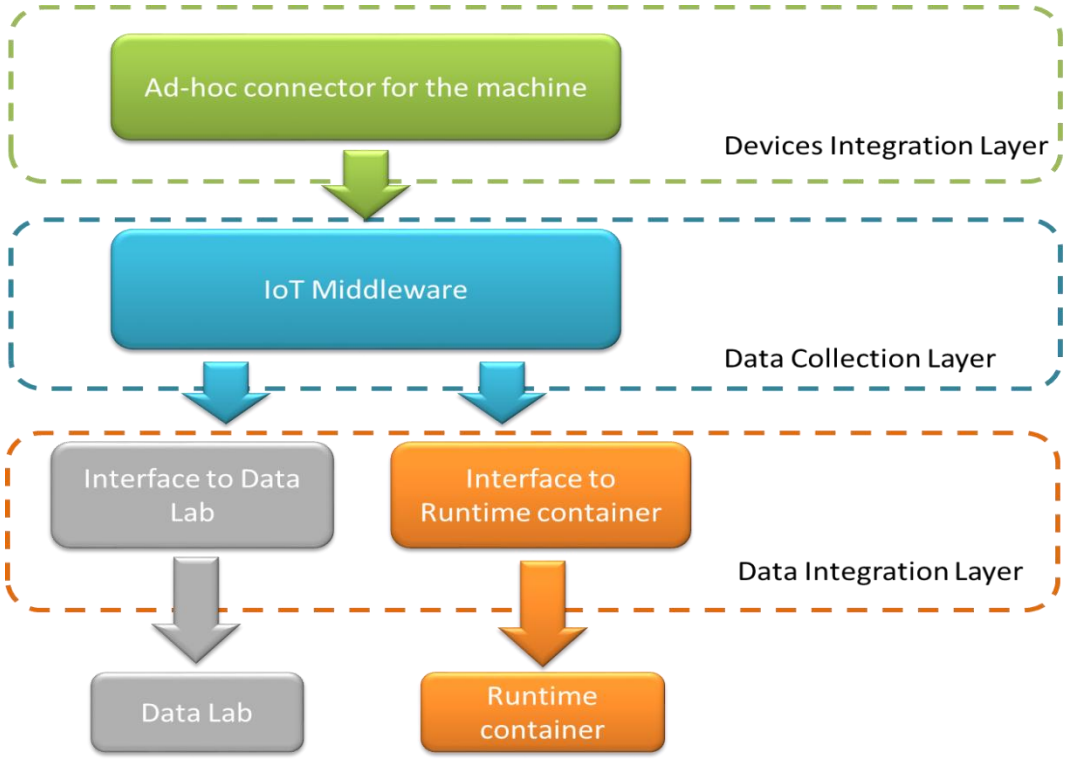


Figure 14 - VPIRA Data flow

Devices Integration Layer: this level aims to retrieve data from the shop floor, i.e., from machines in the different plants, ERP, Historian PI. The interfaces and communication with these sources have been descripted in the previous sections with respect to the two domains of interest. In other words, this layer

contains many connectors as the number of the sources. In this stage, the VPIRA component, can communicate with Euromap63, Historian PI, OPC-UA and Mesal.

Data Collection Layer: after the data has been read through the connectors, this dedicated layer ensures that raw data can be properly collected, decoded and enriched with required semantic annotations according to the semantic models defined in the Cross-sectorial Data Lab.

Data Integration Layer: the last layer envelops the data processed and feeds the real-time container and the Big Data Storage platform. In this version VPIRA will communicate to both sides according to WP4 and WP3 specifications.

4.3.2.4 Runtime Container

The Runtime container is the environment where predictive functions are executed over real-time collected data. The Figure 15 shows a functional view of the Runtime Container with main functionalities.

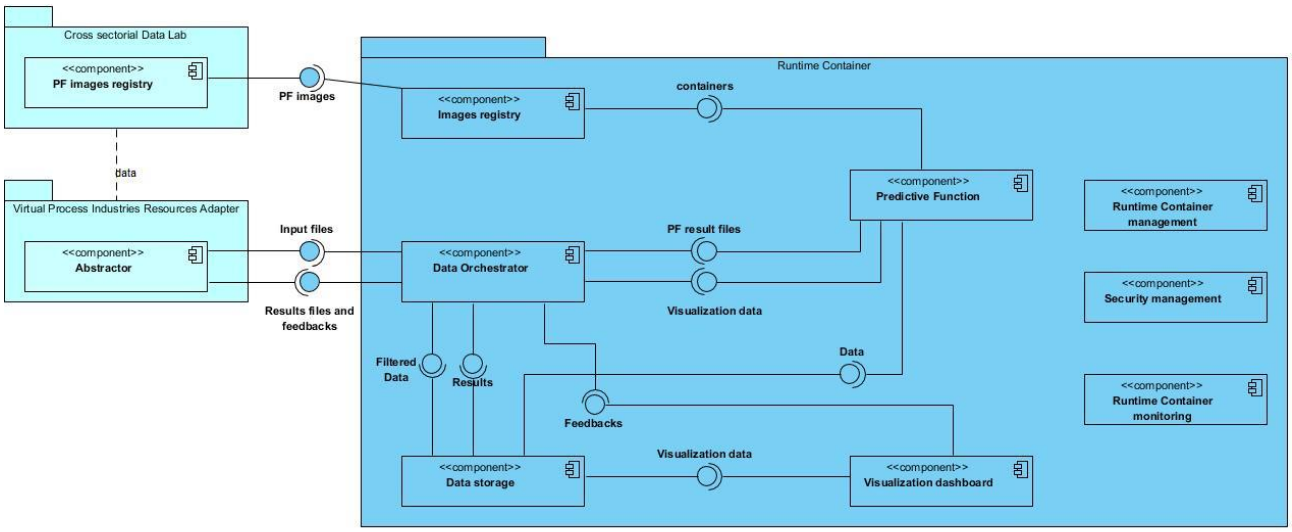


Figure 15 - Runtime Container

The main functionalities of the Runtime Container are given by the following main components:

- Data Orchestrator is the central component of the Runtime Container~~RC~~. Its goal is to be the interface between the Runtime Container environment and the outside world and also to coordinate the data communication inside the Runtime Container. It also filters the raw data according to the needs of the deployed predictive functions.
- Data Storage stack is composed of different type of databases and communication tools between them. These databases have different purposes:
  - Storing the raw data
  - Storing the Predictive Function results
  - Storing the Visualization data.
- Predictive Function stack is composed of one or several predictive functions which take their input data from the Database storage stack and send results to the shop-floor and storage via the Orchestrator.
- Visualization Dashboard is the component used by the expert, operators, etc. to see/analyse the prediction results and to act on the shop-floor behaviour by sending feedbacks (tuning instructions by example) to the site component via the Orchestrator.
- Registry is a library of images of predictive functions ready to be deployed in the Predictive Function stack. The images are prepared in the Data lab and then stored in the registry before deployment.

- Runtime Container Management stack is composed of various components which permit to administrate properly (Deploy, start, stop, etc.) the other components of the Runtime Container.
- Security stack is mainly composed by a LDAP directory for the authentication of the different component of the Runtime Container.
- Monitoring Stack is composed of several tools which monitors the good behaviour of the Runtime Container. The monitoring views are available in dashboards.

4.4 Information View

Raw data retrieved by connectors from the site plant, before transmission to the Datalab or to the Runtime Container, are transformed into a homogeneous format by the Abstractor (one component of the Virtual Process Adapter). The used format in Monsoon is JSON. The Figure 16 shows the global Information view of Monsoon.

Predictive functions, defined and packaged in the Data lab, are deployed and run into the Runtime Container. The results of predictions are sent back to the Data lab in order to improve the quality of predictions and the possible feedbacks (instructions to the machines or to operators) are sent back to the shop-floor.

4.4.1 Data Model/Format

The data model used for transformed raw data and Prediction results is based on JSON format which is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

The current raw data retrieved from site-plant are time-series (datetime-value couples for one variable or datetime-values tuples for several variables). The Abstractor takes these data as input and generates a flow of aggregated information (one per variable). The flow is sent to the Data lab and to the Runtime Container.
The format of output transformed raw data from the Abstractor is like: one file per variable with format

```
[{"value":<value>,"timestamp":"<timestamp>"}, {"value":<value>,"timestamp":"<timestamp>"}, …]
```

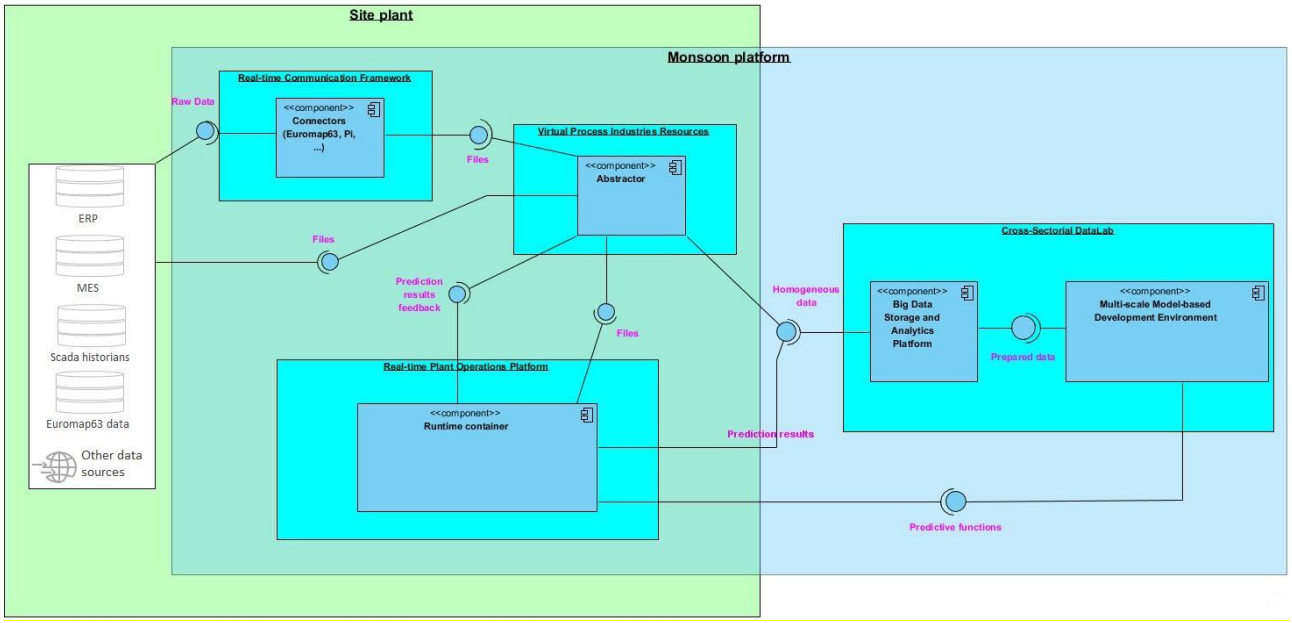


Figure 16 - Information View

The format of a Prediction result/feedback is given in Annex 1.

4.4.2 Data Flow Management Components

The data flows are mainly managed by several instances of Apache NiFi platforms.Apache NiFi is a software project from the Apache Software Foundation designed to automate the flow of data between software systems. The software design is based on the flow-based programming model. It supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.

Three main instances of Apache NiFi are used globally in Monsoon environment:

- The Abstractor, one component of the VPIRA, transforms and communicates data between the shop-floor, the Runtime Container and the Data lab,
- The Data Orchestrator organizes the data flow inside the Runtime Container,
- The Datalab NiFi organizes the data flow inside of the Data Lab.

4.4.3 Predictive Function Life-Cycle

Figure 178shows a standard life-cycle of a Predictive Function from the conception to the deployment.

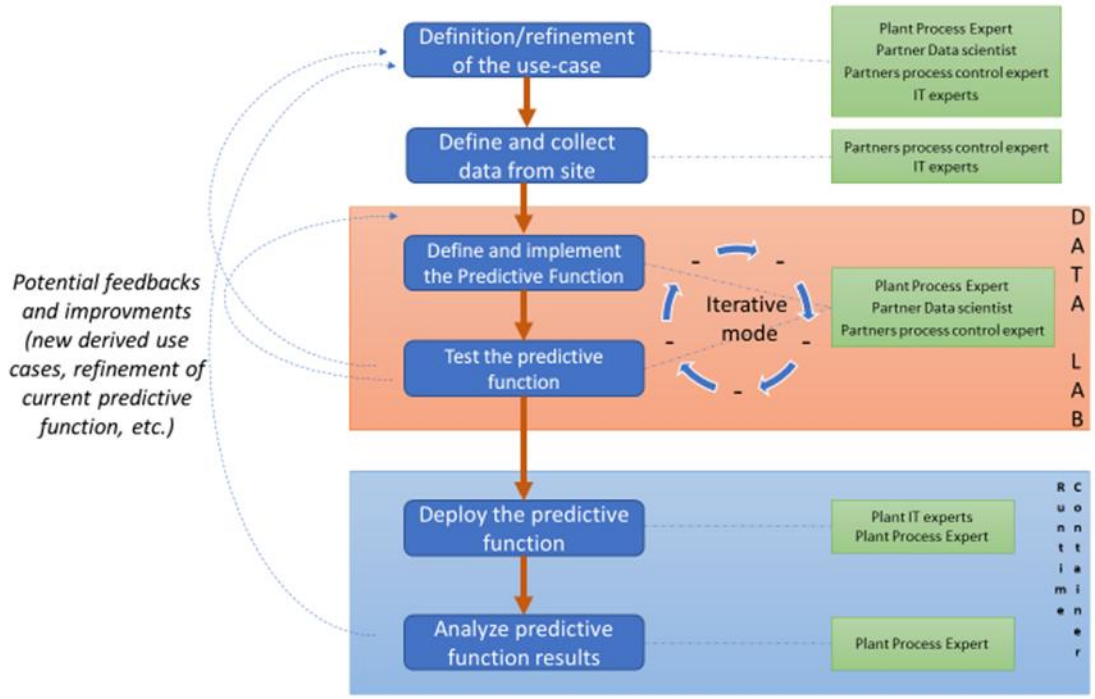


Figure 17 - Predictive Function standard life-cycle

4.5 Deployment View

Deployment view can be divided into various configurations according to the intended usage such as Developing Environment, Testing Environment or Production Environment. Each environment can be further divided into two main parts of the MONSOON platform – Operations Platform and Data Lab platform. Data Lab platform and Runtime Container platform are never deployed together. Data Lab platform should be deployed on a public infrastructure, in order to make it available for all cross-sectorial partners, whether Runtime Container platform should be deployed on each industrial partner site.

4.5.1.1 Plant Operational Platform

The Figure 18 shows the generic plant operational platform. The Monsoon production environment consists in one (or several) server(s) hosting the different components (Abstractor, Runtime Container components, Monitoring component, etc.). The plant operational platform is based on Docker.

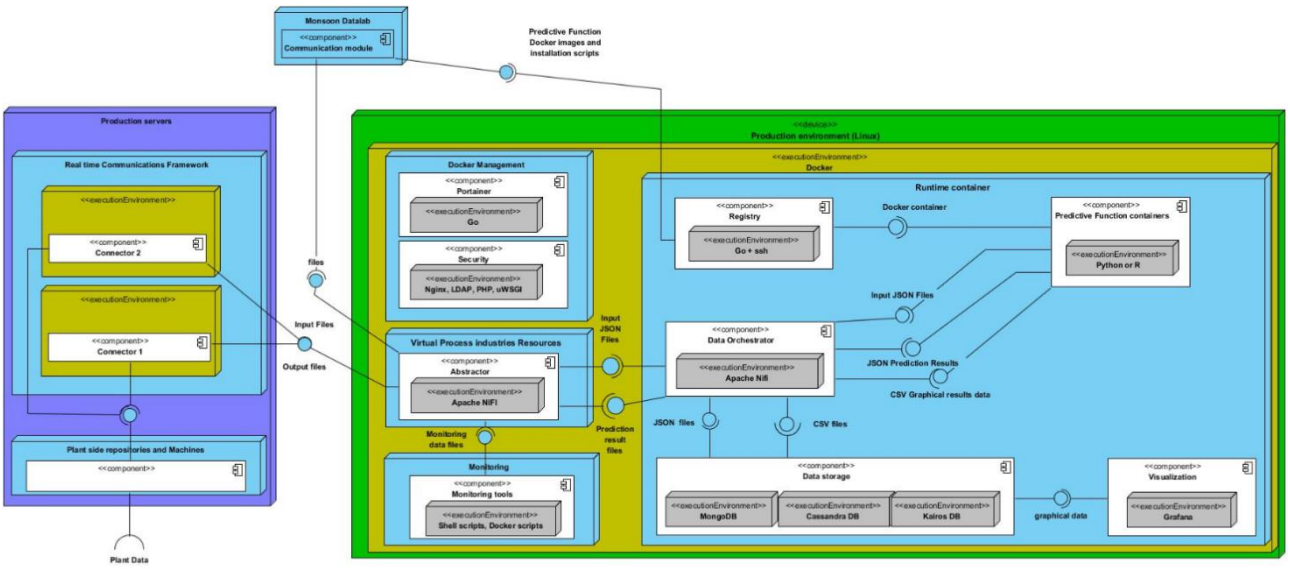


Figure 18 - Plant Operational Platform Deployment

The plant operational platform interacts:

- with the shop-floor servers by receiving raw data files and sending back Predictive Function results and feedbacks
- With the data lab environment by sending raw data files and Predictive Function results and receiving Predictive Function images (Docker images).

4.5.1.2 Cross Sectorial Data Lab Platform

The Data Lab architecture component deployment is based on Docker container technology. The underlying infrastructure is based on the physical servers and virtualized environment built on top of the physical layer. Following sections describe the deployment of the particular components and concrete technologies used in their deployment as shown in Figure 19. Section 4.5.1.3 contains description of their deployment using Docker containers.

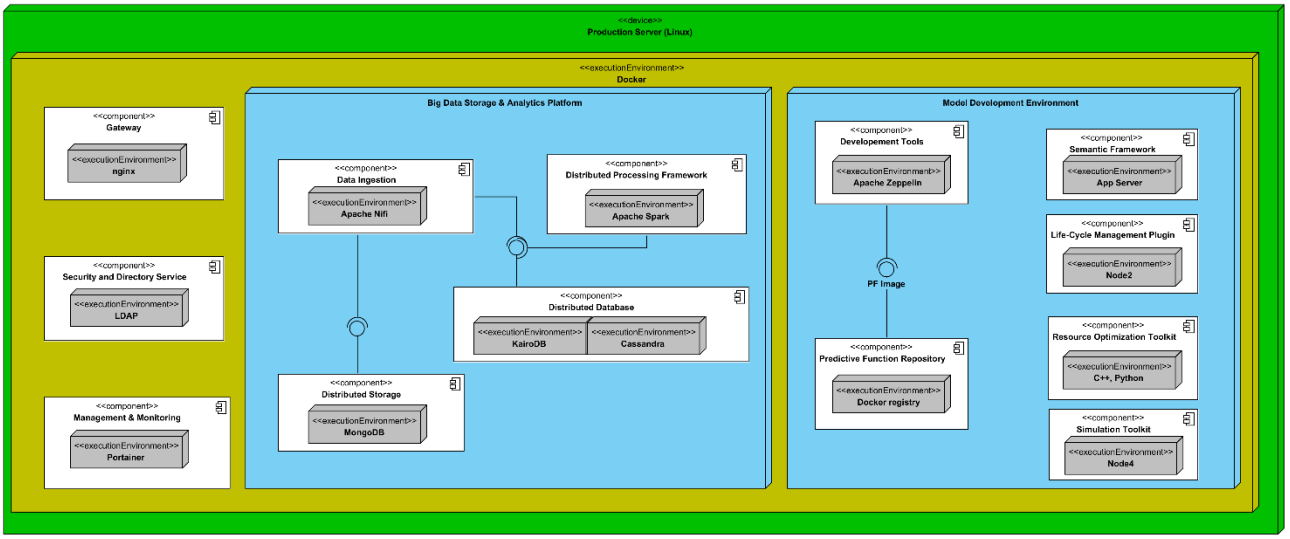


Figure 19 - Data Lab Deployment

Big Data Analytics platform

- *Data Ingestion* – Apache NiFiis used for data ingestion; operational data can be stored either in MongoDB or KairosDB
- *Distributed Storage* – MongoDB NoSQL scalable database is used to store the data in the distributed fashion
- *Distributed Database* – for this component, KairosDB database is used. KairosDB is a fast, scalable time-series database. On the back-end, KairosDB uses Cassandra to store time series data – a popular NoSQL database
- *Distributed processing framework* – as most of the predictive functions developed using Development Tools could be written in Python. Apache Spark interpreter for Python is also supported in Apache Zeppelin component, which is used in Development Tools

Model development environment:

- *Development Tools* – Apache Zeppelin is used as a basis for development tools. Enables to run the code in form of notebooks using various interpreters including Python. Predictive functions developed in the Zeppelin can be packaged into the image and stored in the Docker Registry
- *Predictive Functions Repository* – as the predictive functions images built using the Development Tools are Docker images, this component is represented by Docker Registry
- *Semantic Framework* – developed Semantic Modeller deployment consists of a Client node with Redux and an Application server with Node.js (JavaScript runtime) and uses MongoDB database as a storage component

Other components:

- *Management and Monitoring* – Portainer is used to manage the Docker environment including containers, volumes and networks
- *Gateway* – enables secure access and discovery of running services, route access from outside to user interfaces of particular components, e.g. Zeppelin, Semantic Modeller or Portainer interface
- *Security and Directory Service* - LDAP server for management of the user identities is used. LDAP is integrated with the Apache Nifi, Apache Zeppelin, Portainer and Semantic Modeller environments

4.5.1.3 Containerization of Data Lab Platform

The Data Lab Platform promises to combine and orchestrate existing technologies and open source frameworks from the Big Data landscape to establish a distributed and scalable run-time infrastructure for the data analytics methods. The initial deployment was performed with multiple virtual machines on an in-house physical infrastructure. It turned out that the overall deployment time and configuration management is the most critical aspect in realizing and operationalizing such a platform. It would be optimal to devise a uniform deployment strategy taking into account different deployment options for the platform such as on-premises, cloud/external provider or hybrid. It has also been learned that different demonstrative and use-case scenarios in both aluminium and plastic domains pose different infrastructure and data requirements. Hence, it is useful to define different deployment pipelines or modes for the platform where the right set of platform services are deployed and orchestrated accordingly instead of full stack deployment. Towards this goal, the Big Data Storage and Analytics Platform has been containerized to adapt a common deployment ground with the objective of easing the usage of common platform technologies and make integration with other services or applications easy. The containerization based on Docker framework is depicted in Figure 20.

The Data Lab platform components can be deployed on a single Docker host or in distributed fashion in Docker swarm mode utilizing distributed computing resources by turning of a pool of Docker hosts into a single virtual host.

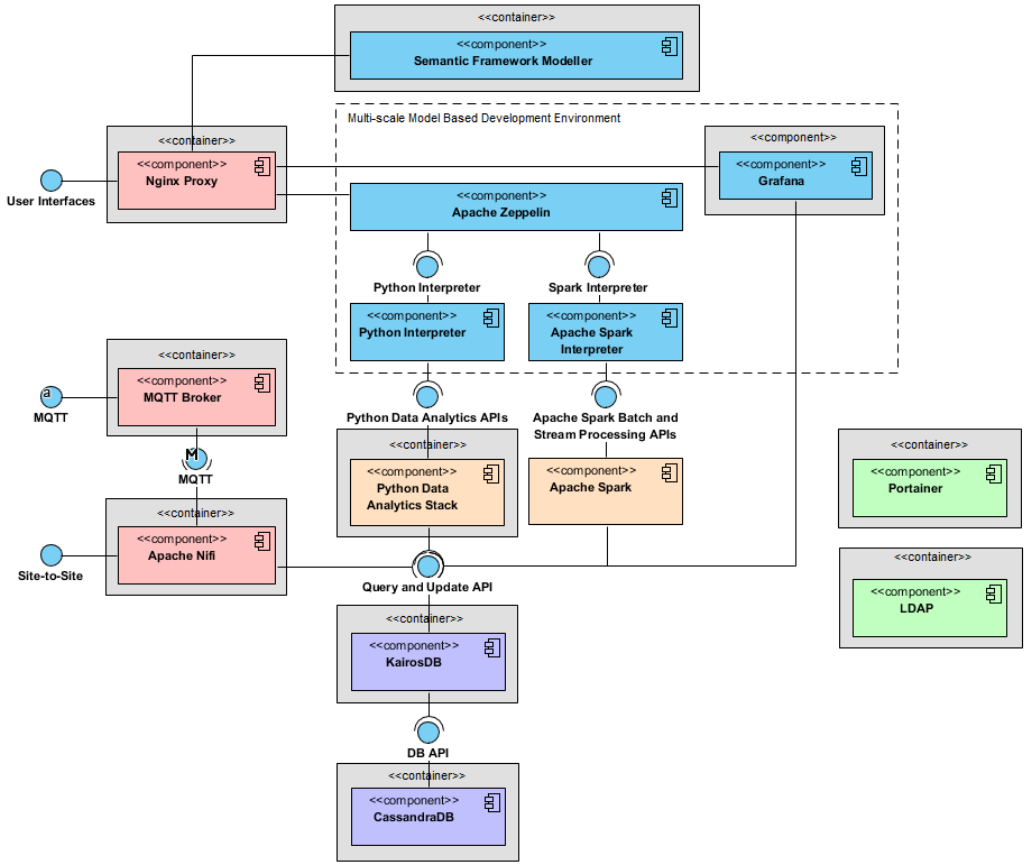


Figure 20 - Containerization of Data Lab Platform

From Network configuration perspective, Docker supports creation of user-defined bridge networks which are best when there is a need for communication of multiple containers deployed on the same Docker host. Current deployment of MONSOON Data lab platform uses two different Docker networks:

- *monsoon_backend* – network connecting containers running all backend services
- *monsoon_frontend* – network enabling connection to user interfaces of Portainer container, Apache Zeppelin UI and Semantic Modeller UI through Gateway container (*Nginx Proxy*). Gateway isthe only component, that has Internet connection through the TUK network and proxy Data Lab services to site clients

Components running management and authentication services:

- *LDAP* – running in container, OpenLDAP authentication server, which stores the user authentication information which is used by Zeppelin, Portainer or Semantic Modeller. Besides the server, separate container runs administration interface of the LDAP server, which is used to manage the authentication information in the Data Lab
- *Portainer* – running in container, Portainer management system for management of the Data Lab containers, images, volumes and networks, user access integrated with LDAP

Components running database services:

- *KairosDB* – container running KairosDB database for time series data storage. As the KairosDB stores the data in CassandraDB database, this one should be also deployed in dedicated container. Data stored in the database are used in Grafana visualization component
- *CassandraDB* – separate container running a Cassandra database used by *KairosDB* container

Components running data ingestion and connectivity services:

- *Apache* NiFi– container running data ingestion service based on Apache NiFi, which forwards data from site or operational data in MQTT format to visualization components, processing or to storage
- *Nginx Proxy* – container running gateway is used to connect the Data Lab to the Internet and enables access to the user interfaces of particular components

Containers running data analytics and modelling services:

- *Semantic Modeller* – container running Semantic Modeller, accessible via web-interface using LDAP provided credentials. The component stores the data in MongoDB, which is running in separate container
- *Grafana* – container running Grafana for data visualization, uses the data from *KairosDB* container
- *Apache Zeppelin* – does not run in container, integrated with LDAP. Zeppelin connects to multiple *Interpreter*s which runs the code on the particular back-end. Although Zeppelin and respective interpreters does not run in Docker containers, dockerized *Python Data Analysis Stack* container can used as the back-end for running of the code
- *Python Data Analysis Stack* –container running all needed tools for data analytics, including Python environment, various frameworks and libraries for data processing, modelling and visualization. The container can be invoked from the Zeppelin tool. Docker here provides isolated environment for multiple users – during each session, dedicated container is created with respective Python interpreter.
- *Apache Spark* – distributed computing framework, deployed in standalone mode, can be used from Zeppelin notebooks

4.6 Development View

This section will describe the methodology used for developing and testing applications in MONSOON project.

4.6.1 Developing and Testing Environment

Since we use different technologies and different languages to build components of MONSOON platform, we do not have a single way of developing ~~them~~ and testing them. Indeed, developing a web application with user interface does not require same testing as developing a Predictive Function which will always run in backend, neither does it requires the same environment.

That being said, each service deployed in MONSOON project will be deployed as a Docker container inside a Swarm cluster. As such, every component can be developed and tested locally by developers. We also provide a production-like environment for testing purpose. This environment consists in a Swarm cluster running on several virtual machines with limited resources.

4.6.2 Continuous Integration Tools

GitLab is a web-based Git-repository manager providing wiki, issue-tracking and CI/CD pipeline features, using an open-source license, developed by GitLab Inc. An instance of GitLab has been set up at CERTH. The choice for GitLab is motivated by the fact that CI services are fully integrated into it and is enabled by default in all projects. Thus, the need for separate plugins and complex scripting of pipelines is bypassed, as GitLab's CI pipelines are defined by simple shell scripts, supporting bash, Windows PowerShell and Windows command prompt scripts.

To set up a CI pipeline one simply needs to add a *.gitlab-ci.yml* file to the root directory of a project and configure a Runner. The *.gitlab-ci.yml* file contains the configuration of CI for a given project, i.e. the jobs that will run, as well as various instructions, which define constraints on how and when these jobs will run. Runners pick the jobs defined in *.gitlab-ci.yml* and execute them in the Runner environment. A Runner can be anything from a virtual or physical machine to a Docker container. Runners communicate with GitLab

through an API, so the Runner simply needs to have network access to the GitLab server. Conveniently, Runners can be set up for multiple projects, instead of just one, in which case they are called Shared Runners. In this way every time a developer commits/pushes code to the repository, the CI pipeline (the jobs defined in the *.gitlab-ci.yml* file in the project root directory) is triggered. All pipelines are listed under the project's Pipelines page.

4.6.3 Code Organization

A Git instance has been set up to address the need for source code management, and each developer partner has been given access to the instance. A brief description of the structure of the repositories is presented hereunder.

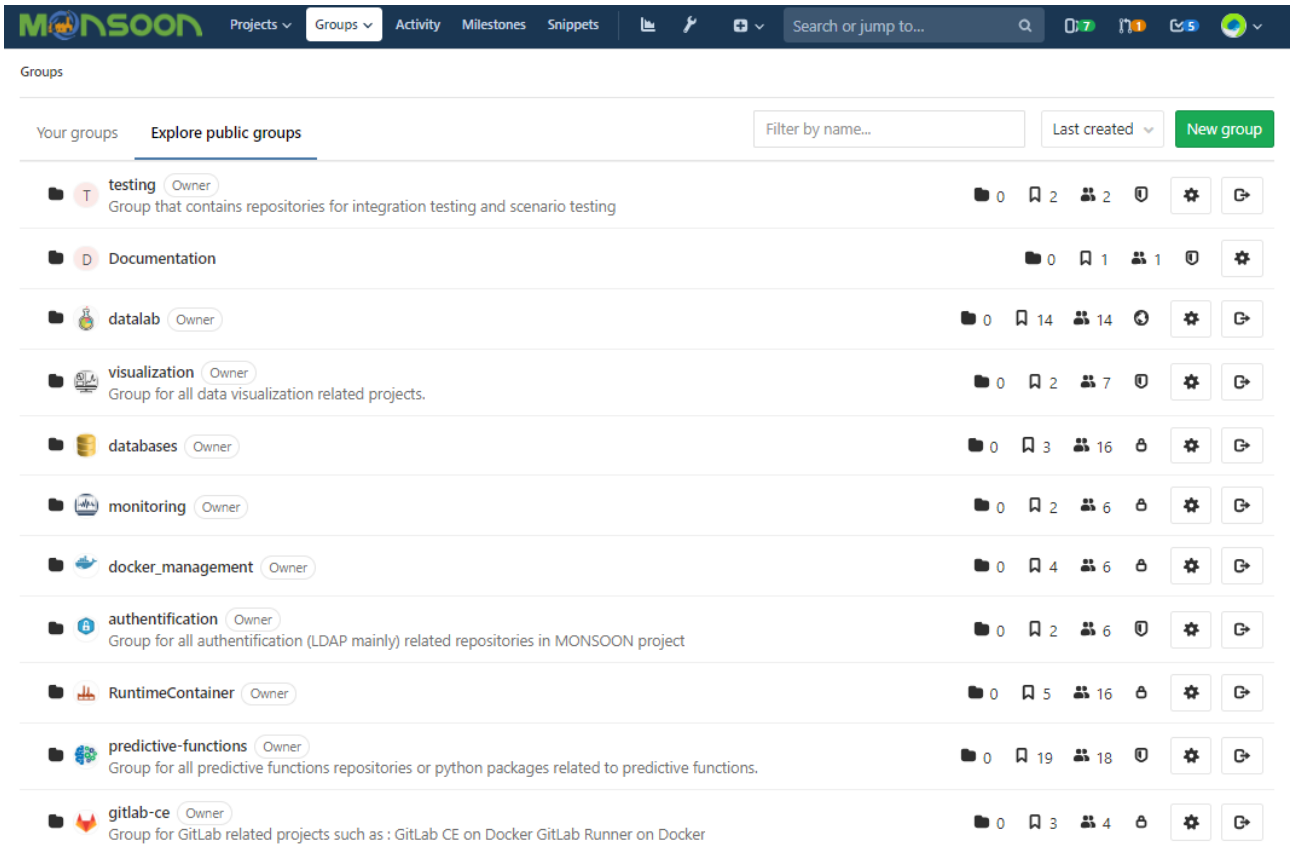


Figure 21 – Project Contained within overarching groups

The MONSOON project comprises many different software components from simulation and visualization to predictive function modules and more. As such, and according to best practices each specific task will have its own relevant repository, i.e. the repository that contains the code that implements the software component that addresses the problem/Proof-of-Concept (PoC) introduced for the particular task. Furthermore, permissions will be set and access will be restricted to the developers of the particular software component. For similar tasks, which can be grouped together under the same overarching concern (such as the PoC for the predictive functions), a common privileges configuration will be set as GitLab allows for the grouping of different projects and the sharing of access rights across projects in the same group. This feature is shown in Figure 21.

Finally, dependencies across projects will be dealt with submodules. The submodule mechanism is a fundamental feature of Git. In case of a dependency relationship between projects, Git allows for a repository to be cloned into a second repository. Thus, the former exists within the latter as a subdirectory with access

to it, all the while keeping commits to both separate. It is noted that while the Git instance runs through the Redmine interface, the repositories will be mirrored to GitLab and all work is expected to be done mainly through it to take advantage of its CI features.

4.6.3.1 Development and testing of Predictive Functions

The development of Predictive Function is split in three stages:

Exploration stage: Most of exploratory analysis are conducted in the Data lab environment. Data Scientists can make use of Zeppelin notebooks features to run code and display results as table, strings, charts or even HTML objects. Since Zeppelin environment is based on Docker, users can provide their own Python interpreter with a Docker image publicly or privately available. Thus, there is no need to globally manage available development environments. Each data scientist is responsible for activating the relevant Docker image for its project. DataLab users can reach Time Series Database from Zeppelin to fetch data in an object oriented fashion within Python using PyMonsoon2 data science library which is developed in a common effort by all partners. During this stage, well known analysis will be conducted such as univariate description of variables, multivariate descriptions, unsupervised classification, dimensionality reduction, statistical test.

Model conception stage: This stage consists in the elaboration of the statistical or machine learning model. As said above, we chose Python as main language for predictive functions.During this stage, data scientists still use PyMonsoon2 library to fetch data, and use datascience frameworks such as Scipy, Scikit-Learn, and Tensorfow (cf. **Error! Reference source not found.**).Models are elaborated within Zeppelin notebooks. A first notebook is used to implement the training of the model and its validation. Once model is validated, it can be dumped using serialization libraries (pickle serialization most of the time). Another notebook will then be created to validate performance of predictive function against historical data. At any time, process experts can access the DataLab to visualize notebooks and give feedback to data scientists. Notebooks can also be exported as HTML files and displayed in team collaboration software such as Confluence.

Predictive Function packaging stage: Once models have been developed by data scientists and validated by process experts, all code written in notebooks will be refactored as a Python package. We intensively use GitLab to manage Predictive Functions lifecycles. As such, packages are hosted on GitLab and Continuous Integration is achieved using GitLab CI/CD. In order to standardize packages a template can be used to generate Predictive Functions repositories. Using a single command line, data scientists can spin up a new GitLab repository with Continuous Integration enabled, running unit tests and code quality tests, as well as building Docker image on each commit.A supported base image is available for Data Scientists, and all Predictive Functions images are created from this base image.

4.6.3.2 Development and testing of Semantic Framework component

During the development of Semantic Framework component, the development and testing environment deployment consisted of a simple client node and an application server running inside a container. As the Semantic Framework uses Distributed Storage component (realized by MongoDB database), the environment required also testing instance of the MongoDB. When using Docker for deployment, developers can download use Semantic Framework Docker image and deploy it together with MongoDB in local environments. Such deployment was used also for testing purposes. When the Semantic Framework is deployed in the Data Lab production environment, the testing database server could be separated and Distributed Storage component from Data Lab production deployment can be used instead.

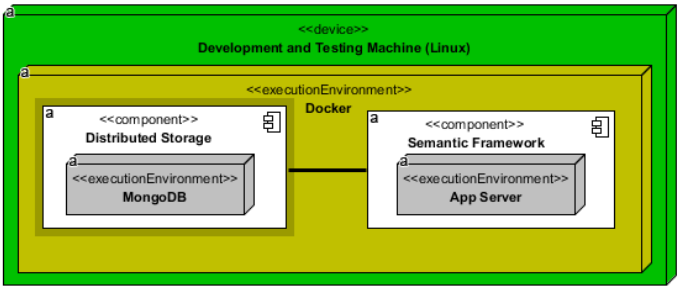


Figure 222 – Development and testing environment – Semantic Framework

4.6.3.3 Development and testing of Development Tools component

In case of development and testing of Development tools, local development environment should contain Function Repository, Data Storage as well as Distributed Data Processing component deployed. In MONSOON, development tool is based on Apache Zeppelin. Currently, Zeppelin is not deployed in the Docker container, but running standalone. Development environment then contains its own instance of Apache Zeppelin. When testing the different Zeppelin interpreters and their back-ends, Zeppelin instance can use Dockerized environments, e.g. Python Data Analytics Stack as in production deployment, or can connect to Distributed Processing Frameworks such as Apache Spark. For development and testing purposes, Spark can be deployed locally on the development machine, or in single-node standalone mode as in distributed environment.

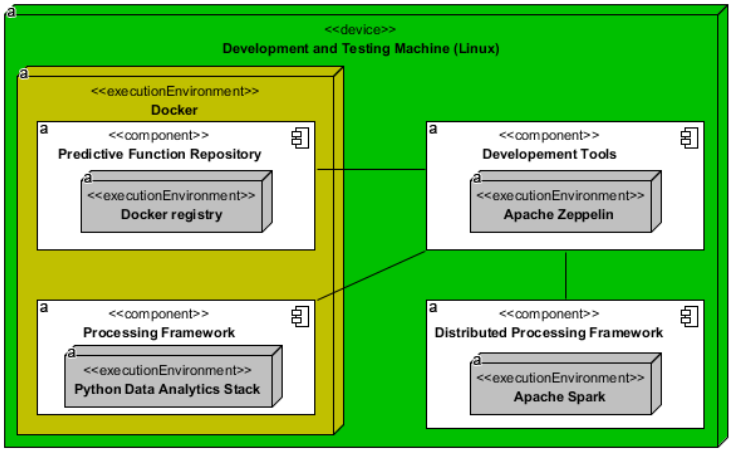


Figure 23 – Development and testing environment – Development Tools

4.6.3.4 Testing of Data Ingestion component

When testing of Data Ingestion tools, those can be deployed locally on development machine. As the Data Ingestion component communicates mostly with Distributed Storage and Distributed Database components, those have to be deployed as well. In MONSOON platform deployment, those components are realized by the respective Docker containers. So, in case of testing of throughput of Apache NiFi used for data ingestion, developers can use a NiFi image to locally deploy a NiFi container, including the required containers with MongoDB and KairosDB (including CassandraDB, as it is used by KairosDB to store the data) which are used for storage/database components. Tests then could be executed in local environment, with the same configuration of the components and their interlinking as in the production deployment of the Data Lab.

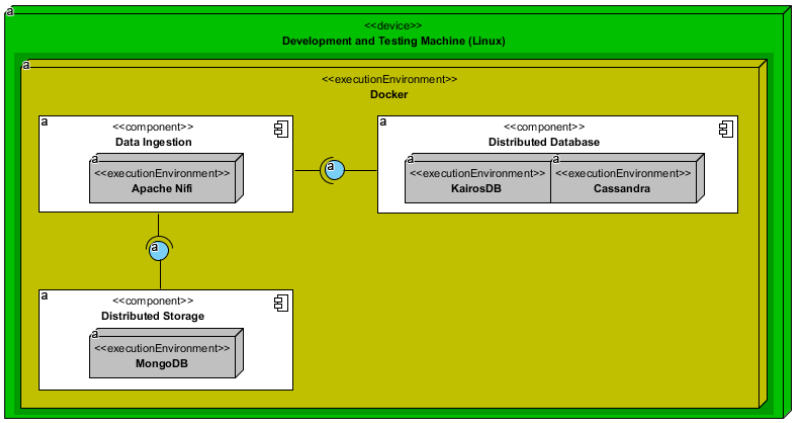


Figure 24 – Development and testing environment – Apache NiFi

5 Technical Scenarios and Use Cases

5.1 Technical Scenarios and Use Cases Aluminium domain

5.1.1 Focus on iteration #1

5.1.1.1 Introduction regarding the use cases

Scope of iteration #1 use cases

The business use cases of the Iteration #1 of MONSOON focused on the green production process (Anode Paste Plant) of the anode life cycle.

See D2.2 – 2.1.5 and D2.3 – 2.1.1 for further details on the green anode production process.

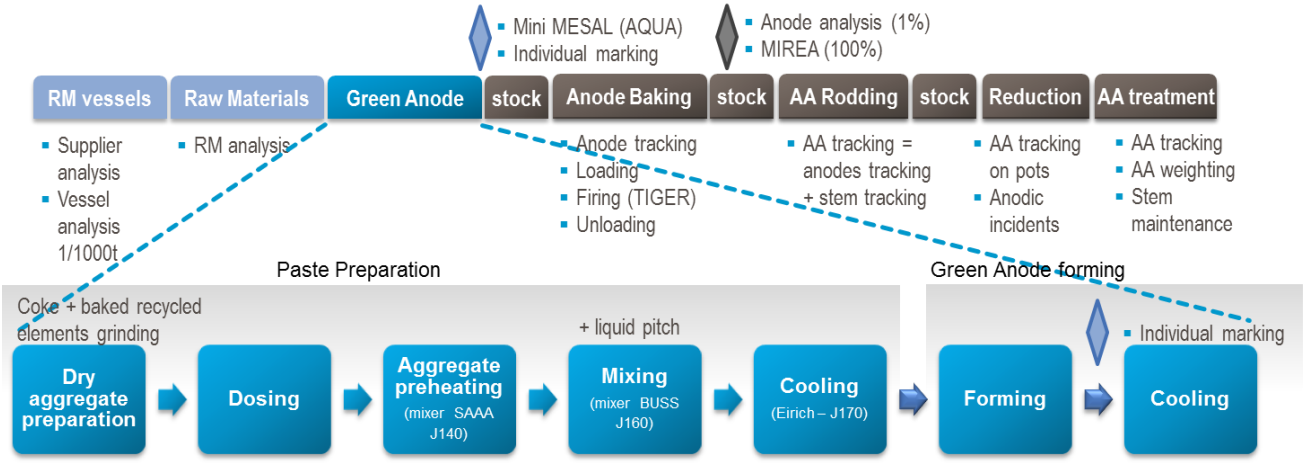


Figure 23 - Highlight on the green anode production process (raw material and anode paste plant)

Anodes contribute as the biggest variable impact to the aluminium production cost.
Optimization goals may focus on the anode paste plant to produce better anodes in terms of quality.
The characteristics expected for good quality anodes are amongst:

- Higher consistency of anode properties
- No anode cracking or slabbing
- No spikes or mushrooms
- Good anode current distribution

- Low anode voltage drop
- Low consumption figures

All of them can be linked to the most important electrolysis parameter, which is the Current Efficiency.

In the anode Paste Plant (PP), we can optimize the PP process by tackling two types of improvement axis:

- Lower conversion cost:
  - Higher throughput
  - Less scrap
  - Fewer breakdowns
  - Lower energy consumption
  - Lower maintenance cost
- Lower raw material cost:
  - Coke / pitch
  - Lower pitch content

The recent technical reports concluded that Aluminium Dunkerque plant produces anodes of high quality with limited options for drastic improvements of their final performance in the pots (high anode density, high current efficiency in pots and finally the low anode net consumption).

Nevertheless a first objective identified for the predictive functions that will be developed during MONSOON iteration #1 is to reduce anode density variability at the anode paste plant.

5.1.1.2 **Predictive Maintenance use case**

<u>Scope: some machines of the mixing chain (J)</u>

The variability on the anode density is in particular a consequence of the mixing chain and forming equipment stoppage and start-up. In this context, the objective is to anticipate the breakdowns and/or highlight equipment/process deviation that impact green anode final quality (e.g. anode density).

During the early stages of data analysis, the mixing chain equipment (the BUSS mixer and the Eirich cooler) and the forming equipment were determined as the main contributors to anode PP stoppages. We decided to focus our studies during the first iteration on the mixing chain (in particular the BUSS mixer).

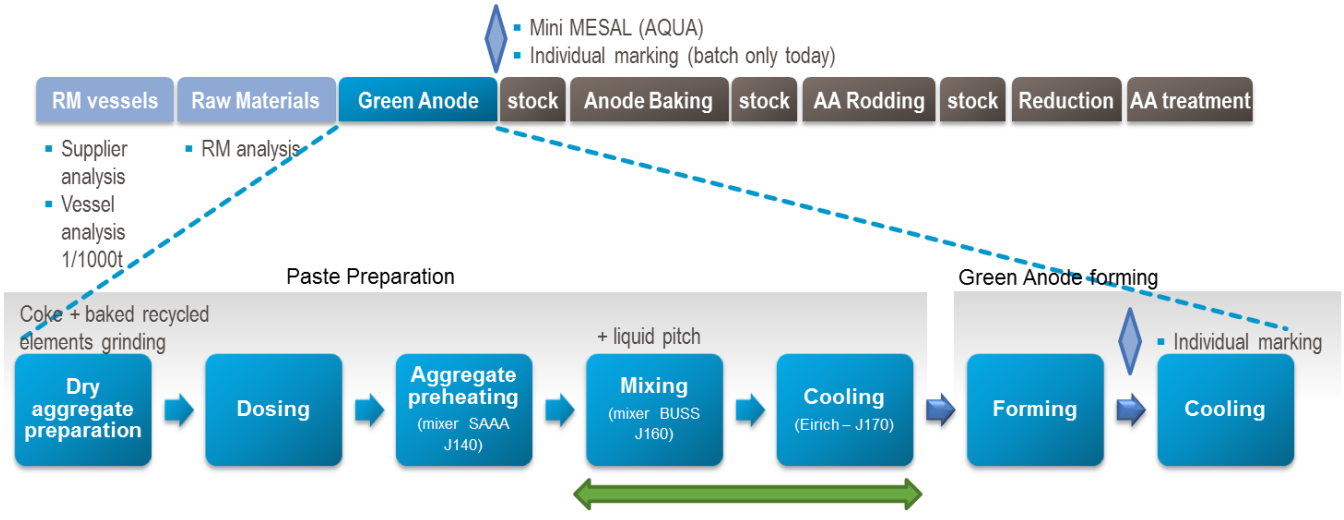


Figure 24 - Scope of the predictive maintenance use case

To elaborate the predictive functions, production data was retrieved from PI system. Furthermore, for classification purposes, we used the description of all PP stops.

The data flow between AD and the MONSOON platform has been thoroughly described in §2.2 of the deliverable D3.2. The type of data is described in the deliverable D3.5, paragraph 3.1.1. The descriptions of the data used during the 1st iteration are given in deliverables D5.1, paragraphs 2.1.4 and 2.1.5, for both the equipment stoppage scenario (also called POC2) and for the equipment misbehaviour scenario (also called POC1) and D5.3, paragraph 3.2.2.

Main objective: minimize interruption time &anode quality loss due to stops

Context:

1. The failure of any upstream part of the chain means a full stop to the whole chain
2. Anode quality (density) is impacted by:
   - Stops and restarts of the equipment
   - Possible misbehaviour from the equipment

Our goals:

We propose to correlate historical sensor data to faults (or unexpected behaviour) in order to bring to light precursors. We want:

- Anticipation (to allow the operator to make adjustments) – An anticipation of 30 to 45 min is an appropriate timescale to intervene on the failing machine.
- The type of the fault
- Description of the precursors

Approaches to address the predictive maintenance use case

This use case was divided into two scenarios: the equipment stoppage and the equipment misbehaviour, studied with different methods and approaches.

The first scenario (or also called POC1) was interested in detecting equipment stoppages. For this purpose, we decided to use a supervised method, the Trend analysis, predicting certain faults from the equipment chains given the sensors data and the full annotation.

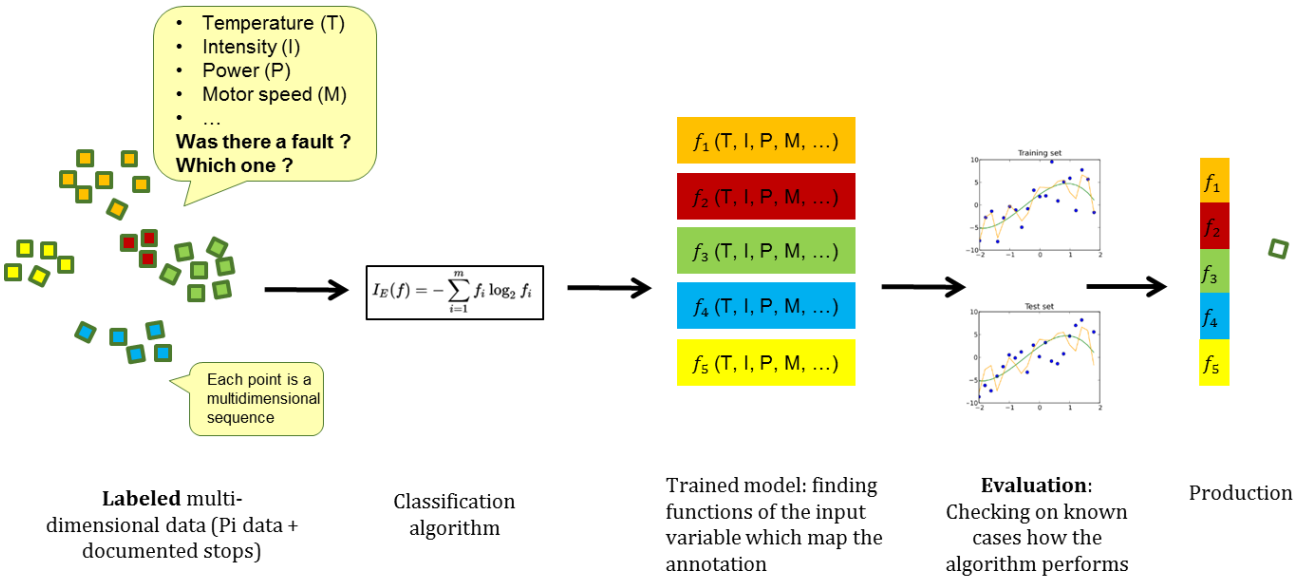


Figure 25 - Illustration of supervised learning using full annotation for prediction

By annotation, we mean the labelling of:

- every fault occurring in the historical data
- every maintenance event in which a fault was identified
- also highly important to have access to "significant" events in the process (stops, maintenance, …)

The second scenario regarding the equipment misbehaviour (or POC2) was tackled with an unsupervised method, using machine learning techniques. This approach enables us to identify "unexpected behaviours" from the equipment chains with the sensors data and without or only partial annotation.

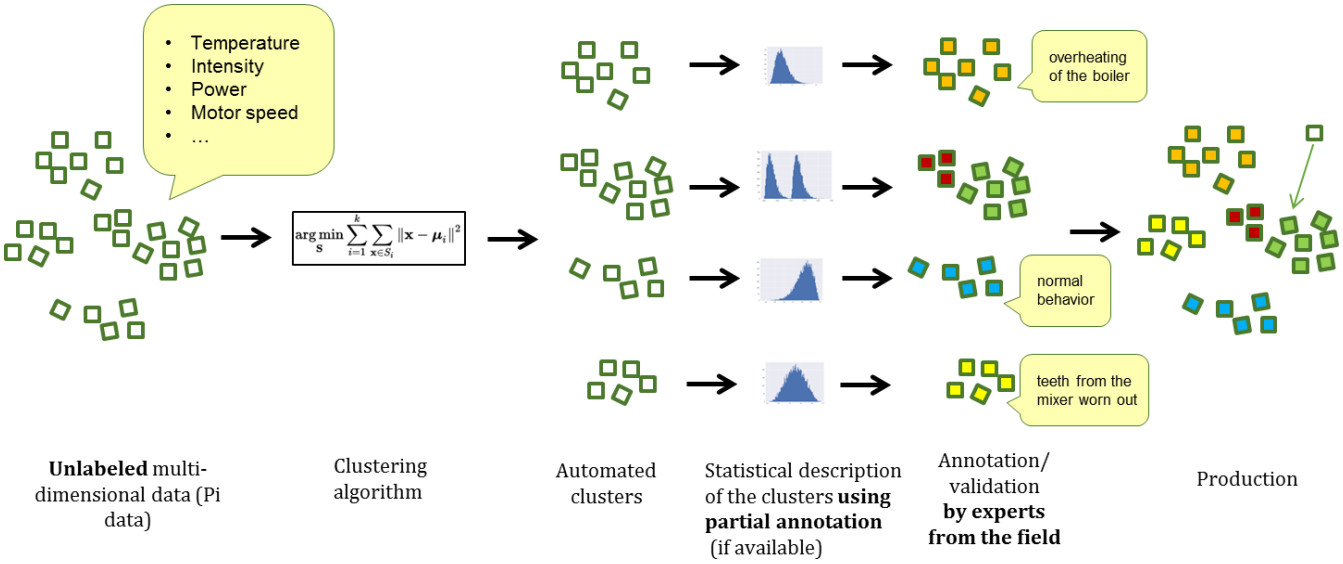


Figure 26 - Illustration of unsupervised learning "unexpected behaviours"

5.1.1.3 Anode predictive quality (green process part) use case

Scope of the green anode predictive quality

In parallel to the equipment stoppages, we decided to study relations between process parameters from the green anode production chain. The first objective is to qualify the green anode quality.
The second objective is to anticipate process deviations via predictive alerts and take countermeasures (e.g. adjust parameters settings) to improve paste and anode quality.

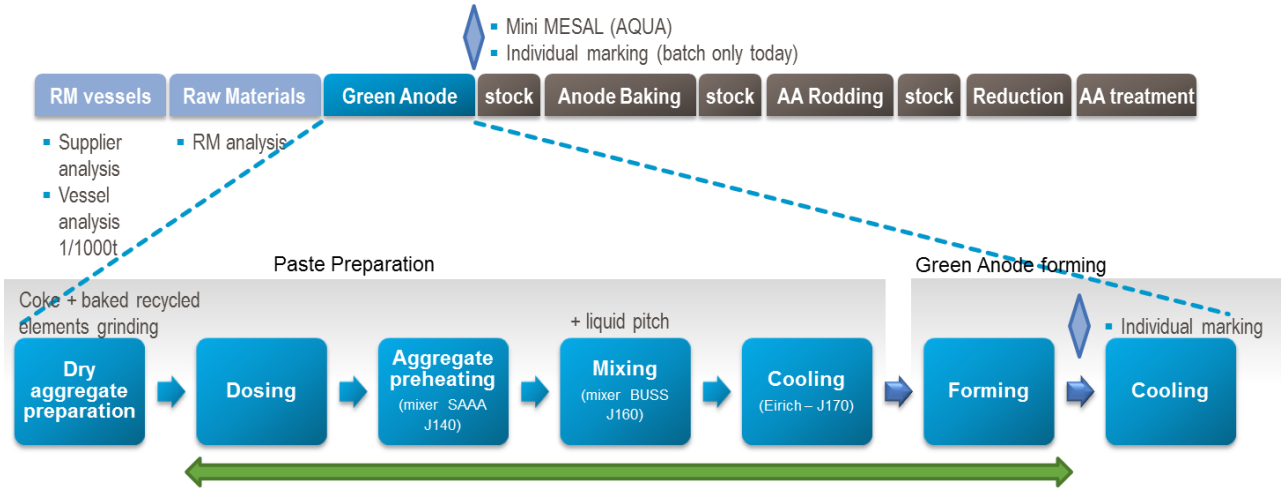


Figure 27 - Scope of the anode predictive quality on green process part

To elaborate the predictive functions, production data was retrieved from PI system and MESAL™ (MES system for Aluminium).

The data flow between AD and the MONSOON platform has been thoroughly described in §2.2 of the deliverable D3.2. The type of data is described in the deliverable D3.5, paragraph 3.1.1. The descriptions of

the data used during the 1st iteration are given in deliverables D5.1, paragraphs 2.1.4 and 2.1.5, and D5.3, paragraph 3.1.2.

Main objectives

In the context of this use case, we need to address the whole green anode production process at once since any part of the process contributes to its outcome which is the green anode.

The problem of anode quality can be addressed through different angles:

1. Understand the source of variability in density of a batch of green anodes
2. Predict the density* of the green anode

* The concept of density needs to be refined. Three parameters are actually to be constrained:

- Height
- Weight
- Density (geometrical or "dry")

For the first iteration, because the green anode height and weight were not available in real-time, we limited the quality criteria to only the geometrical density.

Approach to address the anode quality use case

We decided to use classification techniques, with a supervised approach.

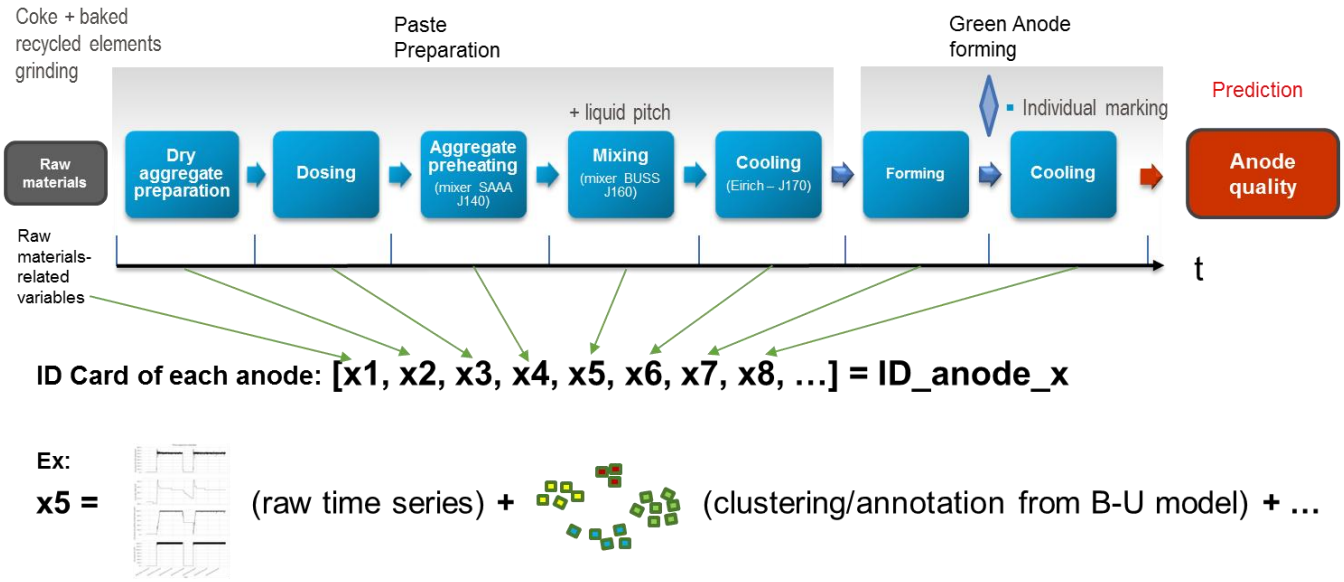


Figure 28 - Green anode predictive quality use case approach principle

We implemented time lags to ensure the correspondence of the paste to the anode produced at the end of the PP (based on operation team expertise).

5.1.2 Outlook for the scenarios to be studied during iteration #2

5.1.2.1 Predictive Maintenance use case

Scope: some machines of the mixing chain (J)

The analysis started on the mixing chain will be deepened with a focus this time on the Eirich machine. This comment applies to the two scenarios above-mentioned, equipment stoppages and equipment misbehaviour.

Main objectives

The objectives during the second iteration are the same, that is to say:

- Anticipation (to allow the operator to make adjustments) – An anticipation of 30 to 45 min is an appropriate timescale to intervene on the failing machine.
- The type of the fault
- Description of the precursors

Approaches to address the predictive maintenance use case

The approaches selected during the first iteration will be kept for the second iteration, that is to say:

- Trend analysis method for the equipment stoppages scenario
- Machine learning, with unsupervised approach for the equipment misbehaviour scenario

5.1.2.2 Anode predictive quality (green process part) use case

Scope of the green anode predictive quality

During the first period, we retrieved data from the Historian Pi that we can consider as raw data. But during this period we were able in parallel to the data scientist activities to develop new MESAL™ modules that will help us to rearrange the data and give data related to a produced anode. So the purpose of the second iteration is to "adapt" the predictive function already developed for batches of 30 minutes of production (almost 15 anodes) to the single anodes.

We will confirm the performances of the batch algorithm when it will run to predict the quality of the single anode.

The production data will still come from both PI system and MESAL™. The MESAL™ system will provide an "Anode sheet", which will gather all the process variables from each piece of equipment through which the paste would have passed and also the characteristics of the anode (geometrical density, height and weight, even the temperature).

Main objectives

Contrary to the first iteration, we hope to be able to enlarge the quality criteria to the anode height and the anode weight, taking into account that these two parameters are included in the Anode sheet. In this context, we will have improved rejection criteria to predict the quality of the green anodes.

Approach to address the anode quality use case

The approach selected during the first iteration, classification techniques with unsupervised approach, will be kept for the second iteration.

5.1.2.3 Electrolysis process optimization

Optimization scenarios on aluminium pots

Because of internal issues and project reorientation, in addition to the scenarios in the carbon area, we foresaw potential of optimization scenarios in the electrolysis area.

The first scenario we would like to study is the prediction of the liquid heights (bath + metal) in the electrolysis pot. The control of these heights may have many impacts on the pot productivity:

- The bath volume affects the alumina dissolution. If the bath height is too low (and consequently the bath volume), the alumina concentration in the bath will reach the saturation point and the alumina will fall to the bottom of the pot creating pot instability.
- The metal height can influence the thermal balance (the metal volume takes part in heat dissipation and the height variability can impact this phenomenon) and the electric stability (orientation of the electromagnetic forces creating liquid movements).
- The overall liquid heights if not controlled can lead to anode submersion and so metal pollution by attack of the iron pin by the bath, or provoke collapses of the covering bath into the liquids (that will also fall at the bottom of the pot like the non-dissolved alumina).

The second scenario concerns the prediction of the pot thermal balance. The pot temperature must be kept between ranges in order to prevent:

- The temperature to rise, if so the bath chemistry can change and aluminium production decreases by re-oxidation of the aluminium (lowering of the current efficiency)
- The temperature to decrease, so the alumina dissolution is less favourable and could lead to create sludge at the bottom of the pot (must be avoided to ensure the high performances of the pot).

And finally, we would like to be able to anticipate the anode effects which occurred when the pot is underfed with alumina. The alumina is supplied through openings made by the chisel in the covering bath. If these opening close due to significant pot temperature decrease, the alumina shots cannot reach the liquid bath to be dissolved.

To elaborate the predictive functions, production data will be retrieved from the ALPSYS system.

As for the Carbon area data, the data flow between AD and the MONSOON platform has been thoroughly described in §2.2 of the deliverable D3.2.

Main objective:

Our identified objectives are to predict the liquid heights (bath+metal) and eventually the pot thermal status and the opening status of the holes in the bath based on the information given by the chisel-bath contact system (one piece of equipment of the pot).

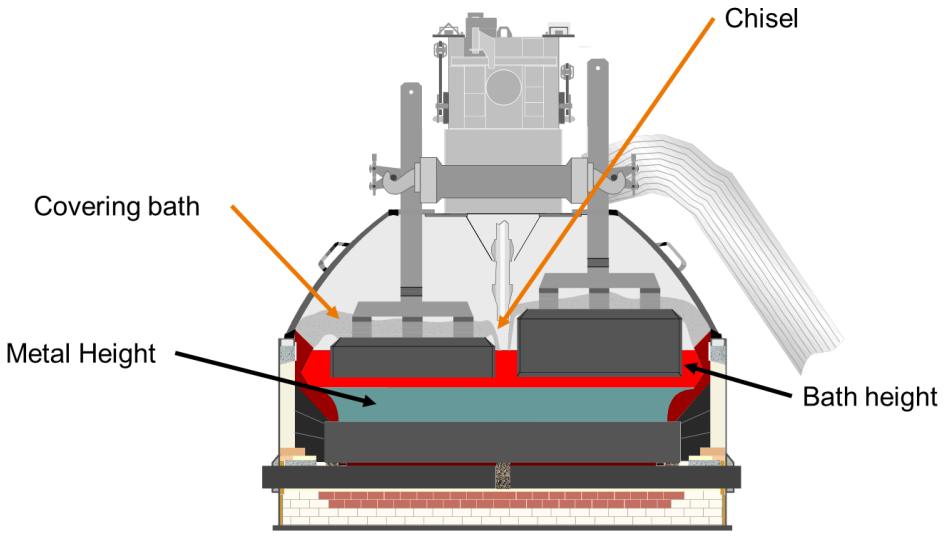
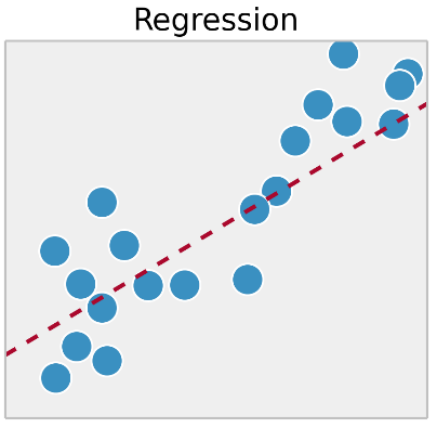


Figure 29 - Pot

Approaches to address this optimization use case

These use-cases can be addressed using regression approaches.



5.2 Technical Scenarios and Use Cases Plastic domain

5.2.1 Focus on iteration 1

5.2.1.1 Introduction regarding business case 1

The business use case 1 of the first iteration is focused on the production of the capsules and decreasing of waste parts. The capsules are manufactured in very large quantities (app. 1.000.000 capsules per day). The long-term rejection rate of the capsules is currently at 3 % and should be decreased to 2.25 % which means a reduction of 10.000 produced waste parts per day.

The characteristics expected for a good quality of the capsules are:

- To observe the diameter of the capsule, its tolerance and its orality
- To prevent the occurrence of short shots
- To ensure that all holes in the bottom of the capsules are free
- To avoid scratches on the capsules
- To ensure the readability of the capsule number

The first two characteristics are linked to the injection molding process and are dependent from the injection phase and the packing phase and its specific technical parameters. The occurrence of closed holes in the bottom of the capsule is mainly related to mold damages by core breaking.The occurrence of scratches is mainly dependent from the periphery (falling process and conveyor belts) behind the injection molding machines. The production process can be optimized by keeping the technical parameters as constant as possible and by monitoring the produced capsules with the help of an optical quality control system which monitors all the defects of the capsules related to its cavity number. These data have to be evaluated by the MONSOON platform.

The Production process of the capsules can be optimized in two ways:

- Lower rejection rate due to process deviations of the injection molding process
- Lower rejection rate due to mold damage

So a first objective identified for the predictive functions that will be developed during MONSOON iteration #1 is to reduce anode density variability at the anode paste plant.

5.2.1.2 Predictive Maintenance use case

On the one hand the variety of the capsule quality is a consequence of deviations of the injection molding process and the input plastic material which leads to coincidently occurring deviations of the capsule quality.

On the other hand mold damages lead to a systematic rejection of the related cavities and its produced capsules.

In the first step the stability (deviation) of the injection molding process was evaluated by the setup of an automatic data export of the technical process variables from the injection molding machines. The result of this is described in D 5.3, chapter 4. The KraussMaffei machine uses a Euromap 63 connector to export the cycle based data. The HUSKY machines use an OPC UA connector to export the cycle based data. These two connectors are currently used to retrieve data from the injection molding process to detect machine stoppages and long-term-trends of technical variables. The data flow between GLN and the MONSOON platform has been thoroughly described in the deliverable D3.2. The type of data is described in the deliverable D3.5. The descriptions of the data used during the 1st iteration are given in deliverables D5.1.Main objective: decrease long term rejection rate from 3 % to 2.25 %.

Context:

The failure of the injection molding process or the injection molding tool leads to rejected parts which are not immediately recognized by the machine worker. The production is ongoing meanwhile the waste parts are rejected until there is a revision of the injection molding process by the machine worker or maintenance of the injection molding tool where the damage is being recognized.

Our goals:

- The correlation of historical and actual sensor data to faults of the injection molding process
- Anticipation of the production of waste parts
- Anticipation of adjustments to be done regarding the injection molding process
- Anticipation of mold maintenance due to mold errors which are "immediately" recognized by the optical quality control system

5.2.2 Plastic business case 2

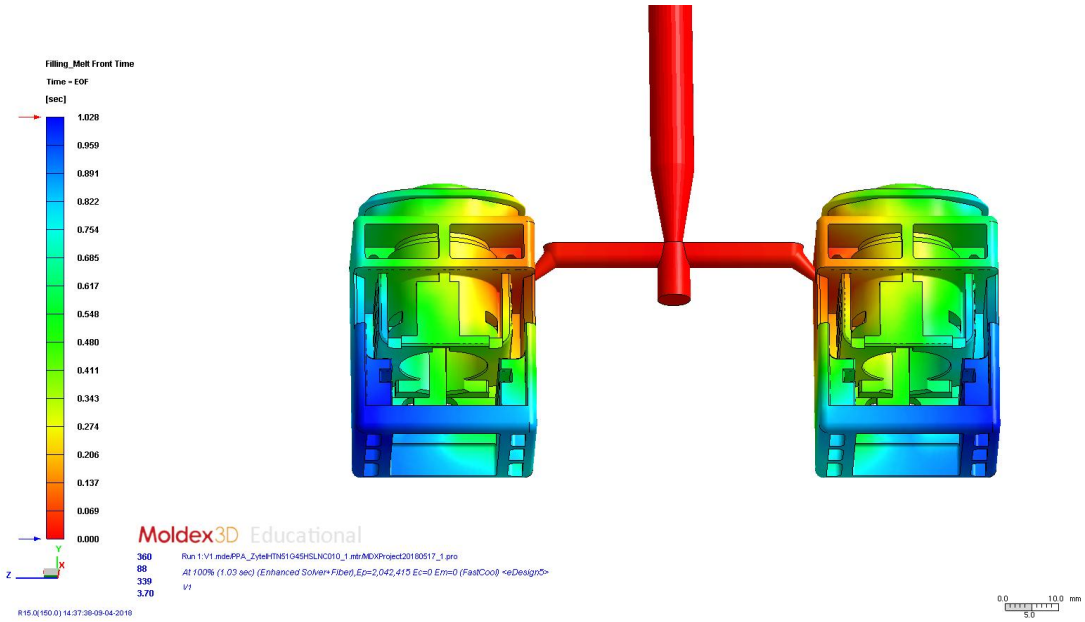
For the business case 2 we had to change the study case to a different plastic part – HOUSING PSS1 but the focus remains the same as it was for the SUBASSY scenario: production of electric connector parts for the automotive industry. This connector part has low output rates compared to the coffee capsules of business case 1 but have very high requests regarding the quality criteria.

The quality control of the HOUSING PSS1 is done by manual inspection and it has a rejection rate of 3 %. Most of causes of the NOK parts are related to process stabilization, dirtiness, out of tolerance values and unfilled parts.

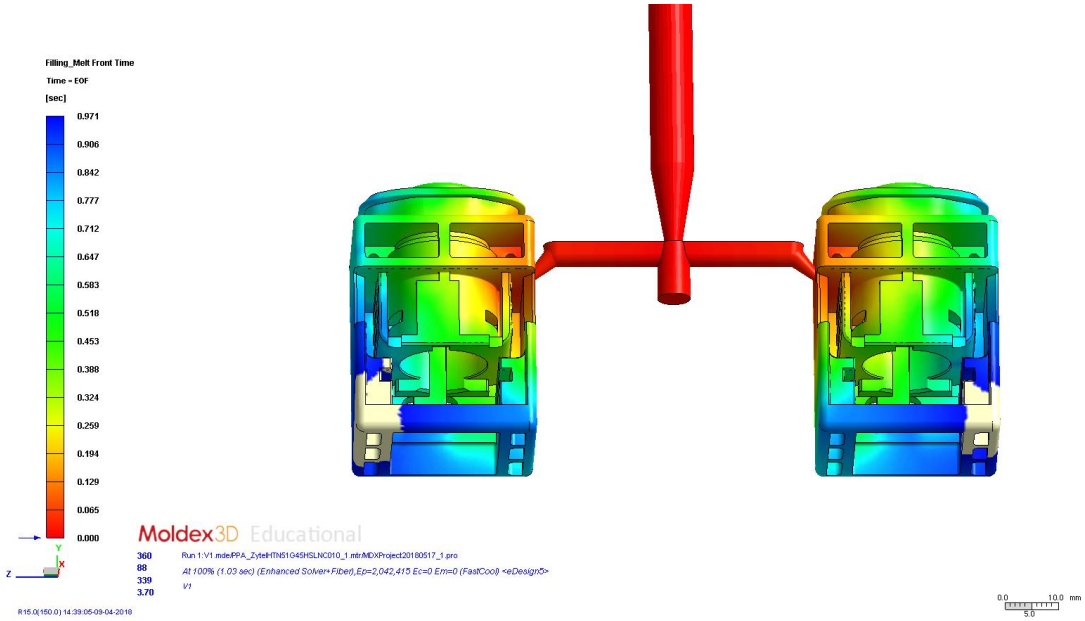
Along the production is running, several plastic parts are being produced for buffer stock to allow preventive maintenance during the batch production. However, because of the mold tool wear after a long period of production the producing parts start to become faulty and a handled re-work is needed. Although the production manager plans several maintenance periods during the batch production, several issues can occur during the production: broken mold parts, electrical issues, cooling errors, etc.

Injection Molding Simulation of the Housing PSS1

To evaluate the filling of the housing and to evaluate possible sensor positions a simulation of the rheological filling was done. The housing has one injection gate. The gate system contains a hot runner system which leads to a cold runner system which fills two identical parts. Therefore one pressure sensor per cavity near the gate and a temperature sensor at the end of the flow part is enough. The closure of a gate leads to a complete non-filling of a part so that the pressure sensor does not measure any data. Thus an additional temperature sensor near the gate is not necessary.

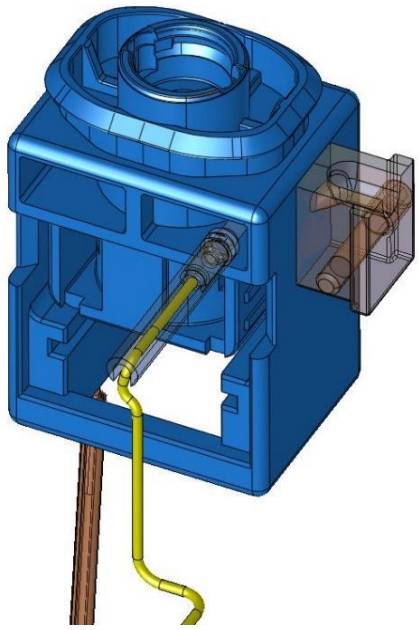


Housing 100 % filled

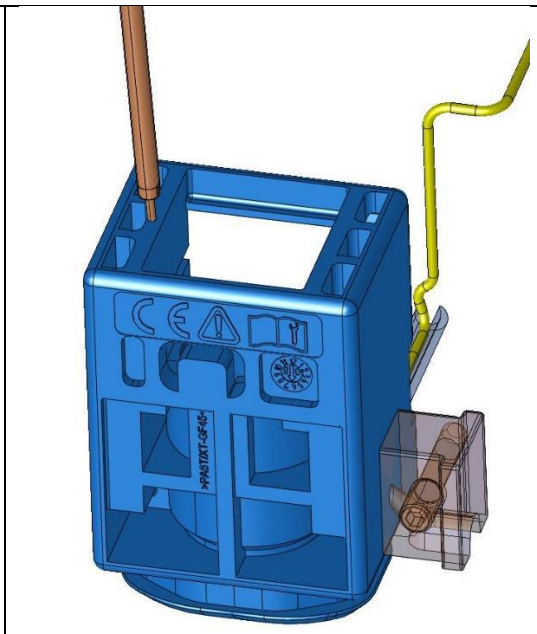


Housing 95 % filled

From the two figures it can be seen which area of the housing is filled last. This is the area where the temperature sensor should be placed to ensure the complete filling of the cavity and to avoid short shots. The pressure sensor should be placed near the gate (the red bar) in the cavity.



Position of the pressure sensor



Position of the temperature sensor

Due to the restrictions of the mold, which has many single parts and moving elements, the shown sensor positions of the pressure ~~sensor~~ and the temperature sensors are the reachable optimum.

Impacts

In the case unfilled parts, they are directly related to the combination of injection parameters: monitoring the temperature in the end filling section of the plastic part and the pressure near the injection point, it can be understandable the flaws between the injection parameters.

For this business case the implementation of temperature and pressure sensors will allow to the injection technicians to better control the behavior of the mold tool and to adjust accordingly the injection parameters to assure a more efficient and profitable process and reduce the NOK parts.

Based on the combination of the sensors information with a set of predictive tools, the MONSOON platform will allow the production manager to visualize the all scenario and define a solid plan with assurances for better production rates and low rejection percentage: less stoppages because of faulty mold, less raw material consumption, waste reduction and decrease of client claims.

The expectations are very associated to the cost reduction. The usage of a predictive philosophy in the management strategy will improve the company standards and also will help to become a more competitive player in the plastic segment – not only as a plastic producer but also as added value provider to the market.

5.3 Technical Scenarios and Use Cases for Global optimization (applicable to both domains)

The aim of the MONSOON platform is also to

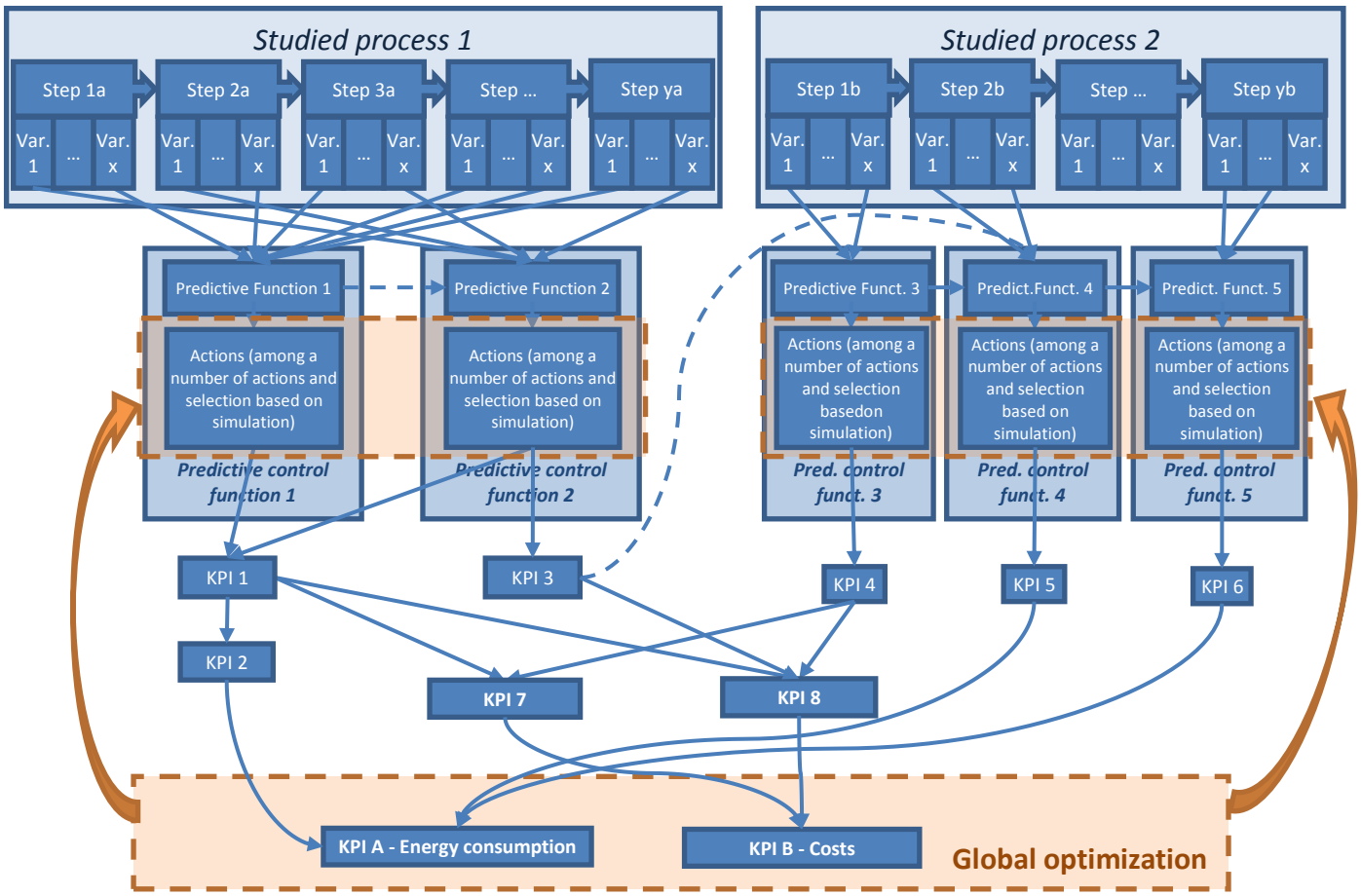
- support the design, development, test and deployment of predictive control functions enabling plant or site wide optimization i.e., a global optimization possibly involving different processes at plant or site level;
- assess the impacts of each local and global optimization performed with the indications given by predictive functions.

More specifically, the impacts can be either environmental improvements or deteriorations, or production enhancement or worsening, but with a common characteristic: to be as far as possible global to the whole plant.

Before starting the design and development of the predictive functions, the process experts can describe the production processes and equipment, the data related to these processes and equipment, the KPIs to be monitored and the improvement objectives related toe.g., processes efficiency and equipment maintenance. The KPIs related to the specific processes and the KPIs linked to global optimization can thus be thoroughly described at the beginning of the algorithms design. For this purpose,experts can use the semantic platform developed by the MONSOON project. In this way, the design of the predictive control functions takes into account from the beginning these required KPIs.

The sketch below depicts how the global optimization is considered within the MONSOON project, based on a generic example with process 1 and process 2, including the relevant steps and variables to be monitored. The picture also introduces a list of KPIs:

- local KPIs (i.e., KPI1 – KPI8) specifically associated to the processes
- global KPIs (i.e., KPI A and KPI B) that are somehow related to the local KPIs while being linked to more than one process.



In this context, MONSOON solution has been designedto support predictive control functions exploiting online data analysis and being able to give a list of possible actions to be done to locally improve/optimize the processes and the production. The MONSOON platform will be then able to estimate for each set of proposed actions the potential impact on local and global KPIs defined by the process experts in the Semantic framework.

For example, when considering process 1, MONSOON can support designing

- Predictive Function 1 - with specific optimization goals associated to KPI1 and KPI2;
- Predictive Function 2 - with specific optimization goals associated to KPI3.

After predicting a possible deviation in the process 1, the predictive control function 1can suggest different possible actions to be implementede.g.

- action A - to change the parameter X of the machine M1
- action B - to modify the recipe of the raw materials
- action C - to advise to plan a maintenance of the subpart of the machine M2

Such actions would directly impact the local KPIs - KPI1 and KPI2 - but also indirectly impact the global ones. Similarly, the predictive control function 2 can suggest another set of actions impacting on the local KPI3 as well as on the global KPIs. In fact, MONSOON predictive control functions will be able to estimate the impact of each action on both local and global KPIs.
When considering process 2 optimization, a different set of actions could be suggested by the predictive control function 3, 4 and 5, where each action would have a specific estimated impact on both local and global KPIs.

In such resulting scenario, the global optimization tool offers the capability tosupport the selection of the most proper set of actions among the ones proposed by the considered predictive control functions in order to reach the optimum thread-off between:

- meeting process-specific local optimization objectives, and
- meeting identified global optimization objectives

Referring again to the above picture, the global optimization tools would support introducing a new global optimization objective that is to minimize

- KPI A – energy consumption
- KPI B – costs

While still considering the local optimization objectives considered for process 1 and 2.
The optimization tool will thus take into consideration all the provided predictions, the different proposed actions and the associated estimated local and global impact, to finally propose a list of possible actions and their performance. In this way, the tool will provide to the operation team all the information and the knowledge needed to decide which actions to perform.

If we revisit the example above, the final output of the predictive control function 1 could be a set of estimated impacts:

- predicted impact of action A: increase of KPI 1 by 2%, KPI 2 by 2%, KPI A by 1%, KPI B by 0,2%
- predicted impact of action B: increase of KPI 1 by 1,5%, no impact on KPI2, no impact on KPI A, decrease of KPI B by 0,3%
- predicted impact of action C: increase of KPI 1 by 1%, KPI 2 by 1%, decrease of KPI A by 0,5%, increase of KPI B by 5%

Having similar output from the different predictive control functions, the optimization tool will be able to select a list of possible combinations of actions such that local and global objectives are met, i.e.

- local processes are improved while
- Correct impact on global KPIs is achieved.

The final control decision will be then performed by the operation team exploiting all the info provided by the optimization tool.

6 Conclusions

This deliverable describes the updated and refined requirements, the process how they are defined as well as the way how they are handled in the project, generally following the standard ISO 9241-210 [ISO, 2010]. It also describes the tools and methods used. It is the second and final iteration ofrequirements based on the more knowledge about the domain, the process, and the intended uses of the MONSOON system.

The architecture has been refined and updated based on the gathered technical requirements. The elaboration of the architecture follows the standards of the Institute of Electrical and Electronics Engineers [IEEE1471, 2000] and [IEEE 42010, 2011], as well as the description from [Rozanski & Woods, 2005]. The architecture is described usinged five different views: context view, functional view, information view, deployment view and development view, which are derived from the Viewpoint Catalogue in chapter 4.1.3.2. All of the platform components are presented with their internal architecture and description. The prospective deployment of the platform using open-source technologies is also elaborated and demonstrated in bothaluminiumand plastic domains.Hencethe final architecture promises its applicability in cross-sectorial domains for varying data requirements andcharacteristics.

Acronyms

Acronym	Explanation
API	Application programming interface
DB	Database
DFS	Distributed File Systems
SQL	Structured Query Language
DBMS	Database management system
RDBMS	Relational database management system
JDBC	Java Database Connectivity: API for java for accessing databases
ODBC	Open Database Connectivity: standard API for accessing DBMS
BLAS	Basic linear algebra subprograms
PMML	Predictive Model Mark-up Language
PF	Predictive functions
REST	Representational state transfer
KPI	Key Performance Indicator
LCA	Life cycle assessment
ROC	Receiver operating characteristic
SCADA	Supervisory control and data acquisition
PLC	Programmable logic controller
OLE	Object Linking and Embedding
TCP	Transmission Control Protocol
IP	Internet Protocol
CPU	Central processing unit
FTP	File Transfer Protocol
MQTT	Message Queue Telemetry Transport
GUI	Graphical user interface
HTML	HyperText Markup Language
HSE	Health, Safety and Environment
CSR	Corporate Social Responsibility
ROC	Receiver Operating Characteristics
CI/CD	Continuous Integration/Continuous Delivery

List of Figures

List of Tables

References

Carroll, J. M. (2000): Making Use: scenario-based design of human-computer interactions. MIT Press.

Clark, Peter G. Lobsitz, Richard M. & Shields, James D.(1989): Documenting the evolution of an information system. IEEE Software, pp. 1819-1826.

Cockburn, A. (2000) Writing Effective Use Cases (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

MONSOON-D2.5 (2017) D2.5 Initial Requirements and Architecture Specifications

Dzida, W. (1999): Developing Scenario-based Requirements and Testing them for Minimum Quality. HCI, Erlbaum.

Dzida, W.; Freitag, R. (1998): Making Use of Scenarios for Validating Analysis and Design. IEEE Transactions on Software Engineering 24(12), pp. 1182-1196.

Emery, F.E.; Trist, E.L. (1960): Socio-Technical Systems. In: Management Sciences, Models and Techniques, Vol. 2, London.

Hickey, A.M.; Dean, D. L.; Nunamaker, J. F. (1999): Establishing a Foundation for Collaborative Scenario Elicitation. The DATA BASE for Advances in Information Systems. 30, 3-4, pp. 92-110.

Hickey, A.M.; Dean, D. L. (1998): Prototyping for Requirements Elicitation and Validation: A Participative Prototype Evaluation Methodology. Proceedings of the 1998 Americas Conference on Information Systems, Baltimore MD, pp. 798-800.

IEEE Standard 1471-2000 (2000). IEEE Recommended Practice for Architectural Description of Software-Intensive Systems

ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description"

ISO (2010) 9241-210:2010 Ergonomics of human-system interaction -- Part 210: Human-centred design for interactive systems

Robertson, S.; Robertson, J.R. (1999): Mastering the requirement process. Addison Wesley, London, ACM Press Books.

Rozanski, N.; Woods, E. (2005), "Software Systems Architecture", ISBN: 0321112296

Schmidt-Belz, B.; D. Fleischhauer (1999): Scenario-based System Validation by Users. Human-Computer Interaction - INTERACT '99, Edinburgh, Swindon: British Computer Society.

Sommerville, I. (2011): Software Engeneering. Boston, Pearson.

Sutcliffe, A. (2003): Scenario-based Requirements Engineering. 11th IEEE International Requirements Engineering Conference (RE '03), Monterey Bay, California, US.

Vafeiadis, T.; Bora-Senta, E.; Kugiumtzis, D. (2011): Estimation of linear trend onset in time series. Simulation modelling – Practice and Theory, Vol. 19, Issue 5, p 1384 - 1398.

Vafeiadis, T.; Krinidis, S.; Ziogou, C.; Ioannidis, D.; Voutetakis, S.; Tzovaras, D. (2016): Robust malfunction diagnosis in process industry time series. 14th IEEE International Conference on Industrial Informatics (INDIN'16), Futurescope, Poitiers, France, 18-21, pp. 111-116,

Vafeiadis, T.; Ioannidis, D.; Krinidis, S.; Ziogou, C.; Voutetakis, S.; Tzovaras, D.; Likothanassis, S. (2016): Real-time incident detection: An approach for two interdependent time series. European Signal Processing Conference (EUSIPCO'16), Budapest, Hungary, 29 August – 02 September, pp. 1418-1422.

Weidenhaupt, K.; Pohl, K. (1998): Scenario usage in System Development: a Report on Current Practice. IEEE Software 15(2), pp. 34-45.

Annex 1: Prediction result/feedback JSON Schema

```
{
  "type": "object",
"definitions": {},
  "$schema": "http://json-schema.org/draft-06/schema#",
"minProperties": 1,
  "properties": {
    "DataAnalyticFunction": {
      "type": "object",
      "required": ["@plant", "@name", "Result"],
      "properties": {
        "@plant": {
          "type": "string"
        },
        "@name": {
          "type": "string"
        },
        "Result": {
          "type": "object",
          "required": ["Contexts", "DataAnalyticResults"],
          "properties": {
            "Symptoms": {
              "type": "object",
              "minProperties": 0,
              "properties": {
                "Symptom": {
                  "type": "array",
                  "minItems": 1,
                  "items": {
                    "type": "object",
                    "required": ["@name", "@value"],
                    "properties": {
                      "@name": {
                        "type": "string"
                      },
                      "@value": {
                        "anyOf": [{
                          "type": ["string", "number"]
                        }]}}}}},
            "Contexts": {
              "type": "object",
              "required": ["Context"],
              "properties": {
                "Context": {
                  "type": "array",
                  "minItems": 1,
                  "items": {
```

```
        "type": "object",
        "required": ["@name", "@value"],
        "properties": {
         "@name": {
          "type": "string"
         },
         "@value": {
          "anyOf": [{
           "type": ["string", "number"]
          }]}}}}},
    "FailuresMode": {
     "type": "object",
     "required": ["FailureMode"],
     "properties": {
      "FailureMode": {
       "type": "object",
       "required": ["@name", "@value", "Symptoms"],
       "properties": {
        "Symptoms": {
         "type": "object",
         "required": ["Symptom"],
         "properties": {
          "Symptom": {
           "type": "array",
           "minItems": 1,
           "items": {
            "type": "object",
            "required": ["@name", "@value"],
            "properties": {
             "@name": {
              "type": "string"
             },
             "@value": {
              "anyOf": [{
               "type": ["string", "number"]
              }]}}}},
        "@name": {
         "type": "string"
        },
        "@value": {
         "anyOf": [{
          "type": ["string", "number"]
         }]}}}},
    "DataAnalyticResults": {
     "type": "object",
     "required": ["DataAnalyticResult"],
     "properties": {
      "DataAnalyticResult": {
       "type": "array",
       "minItems": 1,
       "items": {
```

```
"type": "object",
"required": ["@date", "@value"],
"properties": {
 "@name": {
  "type": "string"
 },
 "@unit": {
  "type": "string"
 },
 "@date": {
  "type": "string"
 },
 "@status": {
  "type": "integer"
 },
 "@value": {
  "anyOf": [{
   "type": ["string", "number"]
  }]}}}}}}}}}}}}
```