MOdel based coNtrol framework for Site-wide OptimizatiON of data-intensive processes

D2.5 - Initial Requirements and Architecture Specifications

Deliverable ID	**D2.5**
Deliverable Title	**Initial Requirements and Architecture Specifications**
Work Package	**WP2 – Requirements Engineering and Reference Architecture**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**29/03/2017**
Status	**Final version**
Lead Editor	**FRAUNHOFER**
Main Contributors	Daniela Fisseler, Olga Michel Chico, Hassan Rasheed, Shreekantha Devasya **(FRAUNHOFER FIT)** Peter Bednar **(TUK)** Abel Betraoui, Vincent Bonnivard **(PROB)** Rosaria Rossini **(ISMB)**

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	2017-01-04	DFi, OMC (FIT)	First Draft with TOC and chapter 4.1
0.2	2017-01-24	DFi (FIT)	Added responsibilities
0.3	2017-02-20	OMC (FIT)	Added information in chapter 3
0.4	2017-03-01	RRo (ISMB)	Added inputs in Section 4.2 and 4.3
0.5	2017-03-02	VB, AB (PROB)	Added inputs in Section 4.3.8 and 4.3.9
0.6	2017-03-06	DFi (FIT)	Added inputs in Section 4.3, added input from TUK
0.7	2017-03-08	DFi, SDe (FIT)	Added inputs in Sections 4.3, 4.4, 4.5 and 5.1 from CAP (JGR, VGD) and in Sections 4.3, 4.5 and 4.6 from CERTH (DIo, TVa, DKa)
0.8	2017-03-09	PBe(TUK)	Updated structure of document, diagrams
0.9	2017-03-13	DFi (FIT)	Integrated contributions from TUK, CERTH, CAP and KIMW
0.91	2017-03-15	DFi (FIT)	Integrated contributions from CAP, PROB (homogenization of component names), AP
0.92	2017-03-16	DFi, HRa, OMC (FIT)	Added chapter 4.6.2 Added changes to Chapter 3 Added content Chapter 2
0.93	2017-03-17	DFi (FIT)	Added missing paragraphs from FIT, LCEN, ISMB, KIMW Version Ready for internal review
0.94	2017-03-27	DFi, OMC (FIT)	Included feedback from reviewers
1.0	2017-03-29	DFi (FIT)	Final version submitted to the EC

Internal Review History

Version	Review Date	Reviewed by	Summary of comments
0.93	2017-03-22	Massimo DE PIERI, Elisabetta REDAVID, Gian Luca BALDO (LCEN)	Approved with minor fixes and comments
0.93	2017-03-22	Jose Antonio Jimenez Caballero (UNE)	Approved with minor fixes and comments

Table of Contents

Document History 2
Internal Review History 2
Table of Contents 3
1 Introduction 4
1.1 Scope 4
1.2 Related documents 4
2 Stakeholders 5
Description 5
3 Requirements 7
3.1 Purpose, context and scope of this section 7
3.2 Methods and Principles of Human-Centred Development 7
3.3 User-Centred Design Procedure in the MONSOON project 10
3.4 Overview of Requirements 15
4 Architecture 17
4.1 Methodology 17
4.2 Context view 20
4.3 Functional View 21
4.4 Information View 36
4.5 Deployment View 37
4.6 Development View 39
5 Technical Scenarios and Use Cases 43
5.1 Use Cases Aluminium domain 43
5.2 Technical Scenarios and Use Cases Plastic domain 50
6 Conclusions 53
Acronyms 54
List of figures 54
List of tables 55
References 55
Appendix 57

1 Introduction

This deliverable D2.5 Initial Requirements and Architecture Specifications describes the initial requirements, the first 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 use cases in the aluminium and plastics domain. There will be an update of all these topics in D2.6 Final Requirements and Architecture Specifications which is due in month 21.

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 will 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 deliverable, only an initial version of requirements and architecture is described which are to mature during the course of the project.
The deliverable is divided into several chapters, chapter 2 gives a short overview of the stakeholders identified so far, 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.
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
[RD.1]	Initial Process Industry Domain Analysis and Use Cases	D2.2	1.0	2016-11-30
[RD.2]	Initial Real time Communications Framework	D3.1	1.4	2016-12-27
[RD.3]	Initial Virtual Process industries Resources Adaptation	D3.4	1.0	2017-01-31
[RD.4]	Initial Big Data Storage and Analytics Platform	D4.3	0.8	2017-03-15
[RD.5]	Test and Integration Plan	D6.1	0.9	2017-03-16
[RD.6]	Report on the standardization landscape and applicable standards	D8.5	1.0	2017-03-17

2 Stakeholders

The following section gives a brief introduction to the different stakeholders and roles in the Aluminium and Plastic domainthat are considered relevant for MONSOON. The list should be considered as an initial version which resultedfrom the first iteration of user and stakeholder interviews. This list might be updated and changed throughout the different iterations within the project. In this regard,it is not yet explicitly defined who the final end users of the platform are.Guided by the material gatheredin following iterations and the continuousUser Centered Design activities (See: Section 3.3), the end users will beconcretizedas the project evolves and the scenarios become more concrete.

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 that derive in requirements as mentioned in Section 3.2.3 of this document.

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 / RolePlastic 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 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 / RoleOthers:	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 initial set of relevant stakeholder requirements and sub-system requirements. These requirements will guide the developments in the technical work packages, and therefore, this will be the first common source of requirements for the MONSOON consortium.

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. This deliverable is the first result of the process of requirements engineering that the MONSOON project has started. 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. An incomplete requirements analysis tends to lead to problems later in the system development and refinement phases. Therefore it is important to continue the user-centred design process outlined in this document. As a consequence, this document should be considered as an initial version of the requirements that will be the basis for updated and changed requirement reports as new requirements arise or outdated disappear in the iterations of the 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 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.

3.2.1 ISO 9241-210 Standard

The ISO 9241-210 [ISO, 2010] "Ergonomics of human-system interaction" gives guidance on human-centred design activities throughout the life cycle of computer-based interactive systems.

Essential principles in this process are:

- Multi-disciplinary design
- Iteration of design solutions
- Appropriate allocation of function between user and technology
- Active involvement of users and stakeholder and a clear understanding of user and tasks requirements

The multi-disciplinary design is given by the expertise of the people from the consortium members, which include psychologists, computer scientists, usability engineers and designers. The iteration of solutions is implemented in the MONSOON work plan.

The human-centred approach implies an iterative life cycle in a project. A system is perceived as a socio-technical system. Scenarios are part of the system specification; they explicitly deal with the usage of a technical system, the context of use, and the allocation of function between the technical system and human users. Later, when a prototype is available, users can try it out and gain personal experience with the system. Iterative cycles allow advancing from specification to implemented prototypes, from experience and evaluation to improved specifications and improved prototypes.

One of the core tasks of user-centred design is to negotiate and facilitate the communication across the well-known user-developer gap while acknowledging the different forms of expression and different requirements on each side. The literature has a lot of examples demonstrating that end users have to bridge the large gap in understanding especially in projects that apply a waterfall model. [Clark, Lobsitz & Shields, 1989] show that evolutionary or iterative approaches drastically reduce this gap.

The user-centred design process reflects an iterative process with no sharp start and end points: eliciting the "context of use" requires intensive user and stakeholder involvement continuously for the whole duration of the process, and the requirements elicitation likewise extends well into the design proposal phase. There are four essential human-centred activities recommended by the ISO standard (ISO-9241-210):

1. to understand and specify the context of use
2. to specify the organizational and user requirements
3. to produce design solutions
4. to evaluate design regarding requirements

The human-centred design approach implies an iterative life cycle in a project. Iterative cycles allow advancing from specification to implemented prototypes, from experience and evaluation to improved specifications and improved prototypes. As mentioned before, a system is perceived as a socio-technical system, which means that the novel technology is a fit between a technical system and its usage (Emery &Trist., 1960). The design proposals are based on the current understanding of the context of use. These proposals provide an idea on how to meet identified or assumed requirements. The evaluations of the design proposals yield a richer understanding of the context of use and new or modified requirements and thus guide the evolutionary improvement of the design.

3.2.2 The Volere Schema

The ISO 9241-210 standard does not prescribe specific methods to achieve these goals; they are to be chosen according to the current state of the art and what is appropriate under individual project circumstances. Based on practical experiences from other R&D projects, we have devised a scenario-based approach, combined with user and stakeholderworkshops,as well as with expert analysis, based on the structure proposed by [Robertson and Robertson, 1999] for mastering requirements.

The Volere process recommended by Robertson and Robertson ensures that all important aspects of requirements are carefully addressed and that the methods applied have proven their value in practical work. The details of the applied process within MONSOON are explained in the subsequent chapters, whereas the important aspects of the requirement description according the Volere schema are addressed in Section 3.3.4. It has been proven to be of great value to put in the effort to define the global constraints affecting the project and the fine-grained distinction of different types of functional and non-functional requirements. Furthermore the definition of customer satisfaction and dissatisfaction helps in prioritizing the requirements and the proper definition of a Fit Criterion and the rationale so that the reader understands the reasoning behind it helps in evaluating if the requirement has been implemented correctly. The philosophy of Robertson and Robertson is very much in line with ISO 9241-210 and allows a structured processing of the requirements assuring that they remain always applicable and testable.

3.2.3 Sources for the Derivation of Requirements

The requirement derivation process has to be founded on specific sources for information. The two classical sources are scenarios and field studies, which consist of interviews and ethnographicalmethods like participatory observations of the domain context and experimental testing of existing solutions. Additionally, requirements can come from analysis of new developments within the domains represented by the industrial partners in the MONSOON project.

3.2.4 Scenario Discussion

Scenarios have proven their potential to communicate project goals and design solutions among all stakeholders and are widely used to discover and understand users' goals and system requirements. Scenarios can be used at all stages of a project, and for various purposes. In particular, scenarios are the first tangible artefact a project can possibly produce, and are therefore suited to start user involvement very early. Scenarios can capture and illustrate features of a system, modes of its usage, and the benefits for users. Scenarios can be written at several levels of detail, they can tell about the current as well as future states of a socio-technical system. Scenarios can focus on normal usage, but can also be used to explore critical cases, limitations or even catastrophes. Scenarios are useful to support discussion among project team members as well as with prospective developer-users. Scenarios can also be seen as part of the documentation and specification of a system. There is a huge amount of literature concerning scenario-based approaches (see: Carroll (2000), Sutcliffe (2003), Weidenhaupt et al. (1998) and Dzida et al. (1999)).

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.

The basis will be user-centric requirements originating from the ecosystems of heterogeneous stakeholders.

3.2.5 User Workshops and Interviews

Workshops aim at acquiring feedback regarding new ideas and providing invaluable information about the potential market acceptance of the product or the idea. During a workshop, participants are asked questions by a moderator. They enable the collection of information about the working environment, which includes a survey of the existing IT infrastructure, and thus, the identification of preliminary requirements and restrictions from the ability to interface with these systems.

Another approach is conducting one-to-one interviews which can be used at all stages of a project, and for various purposes in order to collect feedback that is unbiased from group effects that can occur when different kinds of personalities are involved in group discussions. During their work, it is more convenient for users to explain what they are actually doing and why. In order to create a more comfortable and conversational situation, it is common practice to perform semi-structured interviews. These interviews loosely follow specific guidelines and a list of questions, but leave room for a spontaneous adaptation of the progress and development of the conversation.

The combination of observation and interviews has proven its potential to deliver the best insight into understanding the processes among all stakeholders. Observation is of great value especially in the initial stages of a project. It is often more target-oriented to observe users in their domain context and their activities than to ask them in interviews.

3.3 User-Centred Design Procedure in the MONSOON project

An essential property of the UCD approach is that it has to be adapted to the specific requirements of the individual project. This chapter gives an overview on how the standard procedure has been instantiated and adapted to the MONSOON project.

3.3.1 Initial Vision and Context Scenarios

A scenario is an acknowledged way of communicating the vision of a particular system, as well as to explain and document requirements. Deliverable 2.1 “MONSOON platform usage scenarios” documents the work undertaken in task T2.1 “Scenario Thinking” and provides high-level user requirements in the form of vision and context scenarios of the future use of the MONSOON platform.

From the main vision scenario more specific context scenarios have been derived. The context scenarios were tentative, trying to capture the context of use for a certain user role and to illustrate how the MONSOON platform might support them. Such context scenarios illustrate the benefits and functionality of a system for certain user groups with their typical tasks and goals [Dzida, 1999]. They describe the users' view of the usage of a system within the current context of work and the envisaged improvement of tasks. Scenarios normally do not explicate details of interaction, which is left for a later stage when mock-ups are available

Creating scenarios of end user behaviour and interaction with platform functionality is an extremely useful instrument for identifying key technological, security, socio-economic and business drivers for future end user requirements. The scenarios provide a vision framework for the subsequent iterative requirement engineering phase.

It is important to note that the context scenarios were meant as means for discussion with users and stakeholders. The scenarios do not contain requirements, but help the users and experts generate the requirements.Following the UCD cycle, the scenarios will be continually refined in the next steps.

3.3.2 User Workshops and Interviews

User Workshop and interviewshave been conducted in order to get insights into roles, processes, exceptions, problems and stakeholders involved in the operation of the technology in the Aluminium and Plastic Domain. The interviews started with a short introduction into MONSOON and its goals. The intention was to get a broad understanding of the work performed in each area (context of use). This allowed to better focus the following activities within MONSOON. Additionally, we wanted to gather an understanding of the context in order to be able to derive requirements in the future.The selected interview partners covered different areas of the working processes that are relevant to the MONSOON project.

The documentation of this information was done by taking direct notes or, capturing the environment by taking pictures.

3.3.3 Requirements derivation

The main task after the completion of workshops and interviews was the consolidation of the information gathered from the discussions. The output of the discussions and interviews are user and stakeholder statements. The analysis of the original users and stakeholder statements in the respective workshops led to the elicitation of a first set of requirements at different levels of detail and their aggregation in a structured way. Such feedback to technical scenarios may relate to various aspects of the system and its use, and have been classified according to the Volere schema (see:[Robertson and Robertson, 1999]).

Functional requirements give the specification of the product's functionality, derived from the fundamental purpose of the product.They frame the solution space for the problem that is being addressed by describing how the system should behave in particular situations: its inputs, its outputs and actions it needs to perform in order to accomplish its fundamental purpose. In some cases, functional requirements also define, in an explicit manner, what the system should not do or detail the needed exceptions. (see: [Sommerville, 2011]). Although there are not established subcategoriesin order to further sortfunctional requirements,the following lines show high levelgroupsthat aim to illustrate someexamples:

- Descriptions of what the product has to do
- Explanation of what services it should provide
- Details of what processing actions or operations it must take
- Description of how the system should react to particular inputs.

On the other hand, non-functional requirements are the properties of the product, the qualities and characteristics that make the product attractive, usable, fast or reliable. They are not directly concerned with the specific operations and services that the product should perform or deliver, but they relate to emergent properties such as modifiability, scalability and interoperability. At the same time, non-functional requirements can as well reflect constrains or restrictions of the system. They are just as relevant and critical as functional requirements to ensure the success of the solution.

Non-functional requirements can be grouped according to following subcategories:

- Look and feel requirements (intended appearance for end users)

- Usability requirements (based on the intended end users and the context of use)
- Performance requirements (how fast, big, accurate, safe, reliable, etc.)
- Operational requirements (intended operating environment)
- Maintainability and portability requirements (how changeable it must be)
- Security requirements (security, confidentiality and integrity)
- Cultural and political requirements (human factors)
- Legal requirements (conformance to applicable laws)

Look and feel, usability and cultural requirements are of secondary relevance for the assessment of requirements for a software platform, but are of high importance for the assessment of qualities and aspects of the user interfaces to be developed. The current set of requirements can be found in Chapter 3.4 of this deliverable and thus has become accessible for all users and also traceable for evaluation of design solutions.

3.3.4 Requirement description (Volere Schema)

The workflow to ensure that all necessary details and procedures in the Volere schema are adhered to is rather complex, and it was decided to support this process with a tool for all partners within the project.

We decided to use JIRA, which is a web based bug tracker that allows implementing and tracking the workflow of the Volere schema. Figure 1 shows a screenshot of JIRA with a list of open requirements.

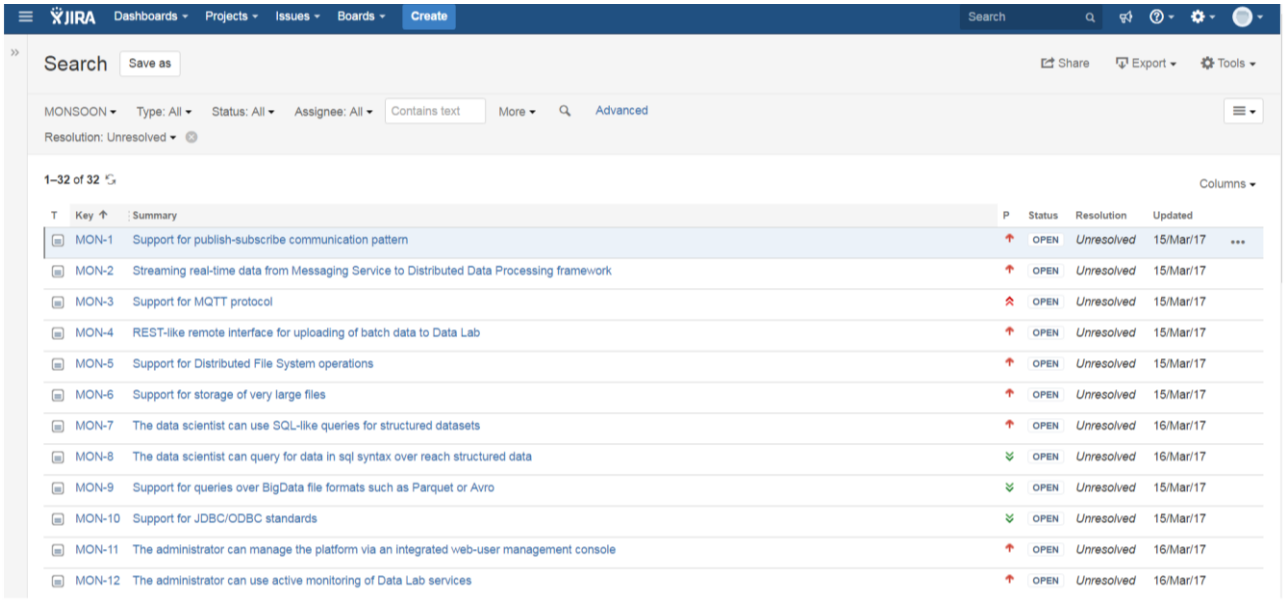

Figure 1 - Screenshot of JIRA with a list of requirements

Priority is a very important field that defines the relevance of this requirement in relation to the other requirements. It allows classification of the specified requirement in five categories: "Blocker", "Critical", "Major", "Trivial", and "Nice to have". The rating was carefully assigned and was the last step of the requirement specification before it passes the quality check. The priority of a requirement is based on several important aspects included in the Volere schema:

• The source, defining if this requirement was raised by primary or secondary stakeholders, or through discussions within the consortium, by vision and technical scenarios.

- The assessment of customer satisfaction and dissatisfaction if this requirement is achieved or missed, respectively.
- The estimation if the requirement is within the scope of the project.
- The component that the requirement is associated to.

The summary of a requirement contains a one-sentence description of the requirement. The description tells about the intent of the requirement and should be clear and brief.

The rationale of a requirement expresses the reason behind the requirement's existence. The rationale provides the reason why the requirement is important and the contribution it makes to the product's purpose. The rationale contributes to the understanding of the requirement.

The Fit Criterion is the quantified goal that the solution (i.e. the realization of the requirement) has to meet. This field describes how to determine if the requirement is met. It should be written in a precise quantifiable manner. The Fit Criterion sets the standard to which the developer constructs the product.

Figure 2 shows a screenshot of JIRA with a requirement in edit mode:

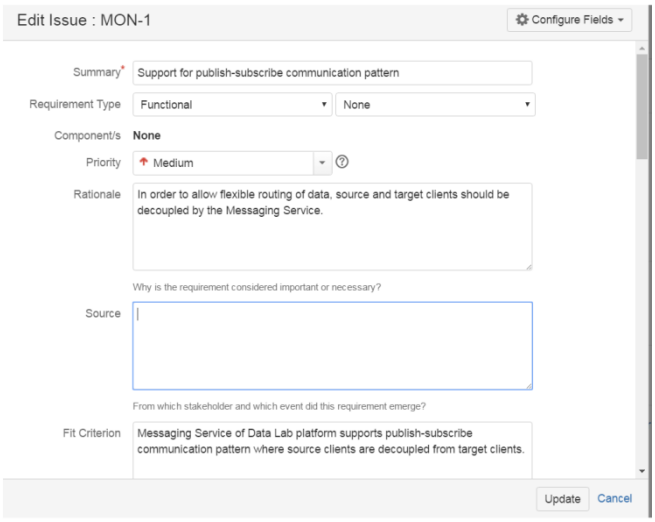

Figure 2 - Screenshot of JIRA with a list of requirements

3.3.5 Requirements Workflow

Two different user groups are involved in the requirements process:

- Reporters: This group contains all project members.
- Assignees: Each newly reported requirement is assigned to a single person – the assignee. The assignee is responsible for passing the requirement through the quality check.

Figure 3 displays a requirement's possible status and the possible transitions between the states.

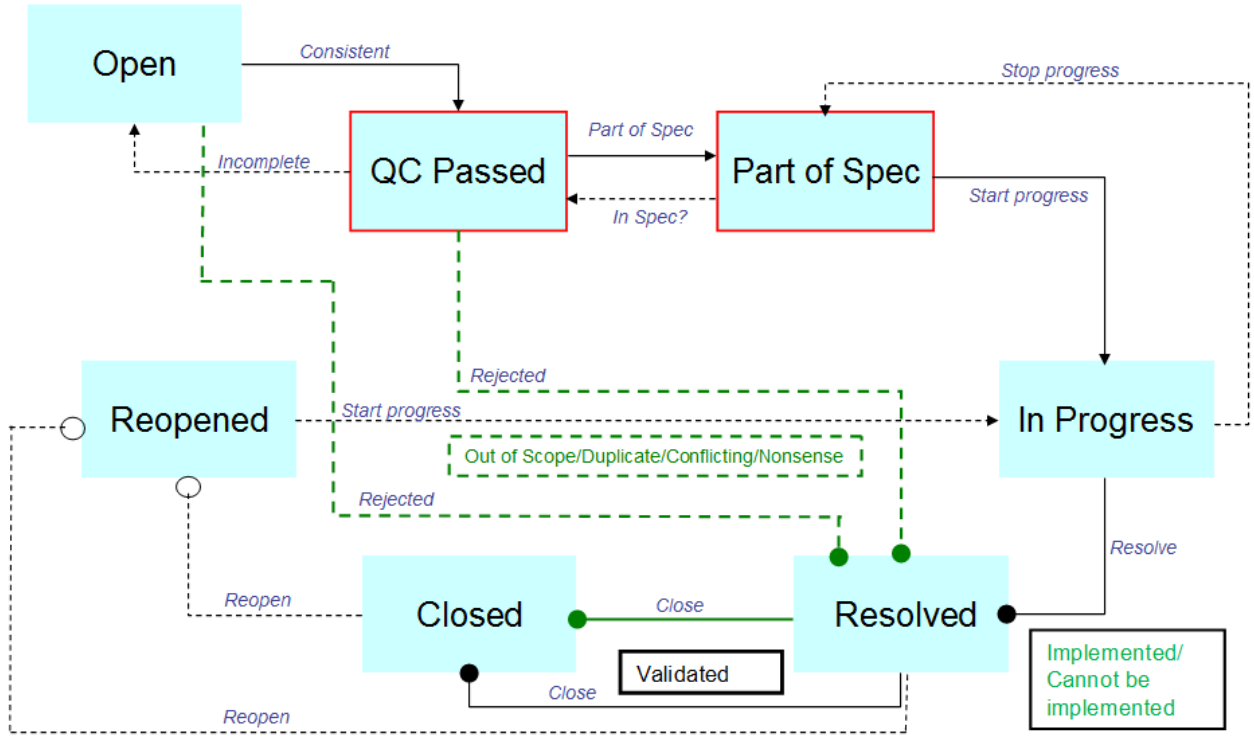

Figure 3- Structure of the Requirements Workflow

When a requirement is entered by a reporter, it gets assigned the status *open*. A project member assigns the requirement to an assignee. If it is complete and unambiguous, the assignee changes the requirement's status into *quality check passed*. Ideally, the assignee and the reporter are different project members. A quality check passed requirement has its text fields filled in sensibly, with appropriate values chosen from the drop-down lists. The priority must be selected to make it possible to rank requirements in relation to each other.

A requirement can fail to pass the quality gateway for three reasons:

1. A requirement can be incomplete. Some fields may have meaningless entries like '?'
2. A requirement can be ambiguous; certain terms are not clearly specified
3. A requirement is too general or does not make sense at all; this can happen for example when the reporter of the requirement does not include enough detail information in order for another person to understand the reasoning behind it.

If a requirement fails the quality check, it gets rejected for one of three possible reasons: incomplete, ambiguous or does not make sense. Once the requirement is updated properly, its status is changed to *reopened*. This status equals exactly the initial status *open* and the quality check process restarts. The status *reopened* is used to indicate that a requirement went through the quality control at least once. This helps to detect requirements that are yet untouched.

Eventually, all requirements will pass the quality gateway. The next step is to decide whether a requirement becomes part of the specification, or whether it is to be rejected. A requirement can be rejected for two reasons:

1. It is a duplicate of another requirement.
2. It is out of the project's scope.

If a requirement's status is either *rejected*, *part of specification* or *duplicate*, a requirement is said to be resolved. If its status is *part of specification*, it means that it will be implemented and validated.

3.4 Overview of Requirements

This section describesthe status of the initial functional and non-functional requirements. A condensed list (not all fields listed) can be found in the*Appendix*. The aim of this approach is to provide a simple and structured representation of requirements, to be used as a reference for the development of the first iteration of the platform and the applications. The list of requirements will be updated during the project lifetime, as soon as the need for new or modified features is identified. We will apply various methods to improve our understanding of the user needs and to improve the user-perceived qualities of the prototypes. In particular, we will review the requirements during the evaluation of the first application prototypes in order to get the second, improved set of user requirements (see for example [Schmidt-Belz et al., 1999]).

Each requirement listed in the requirements table obtains a unique ID to refer to. The description of a requirement is a synthetic but clear description of the requirement. The rationale gives a reason why this requirement is relevant for the system and thus has been included into the table. The column source gives an indication of where the requirement was generated from. According to the Volere scheme the requirements are divided into non-functional and functional requirements.

Current number of requirements: 32

According to priority:

- Major: 11
- Medium: 17
- Minor: 4

According to requirement type:

- Functional: 28
- Non-Functional maintainability: 4

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 Figure 1.
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."

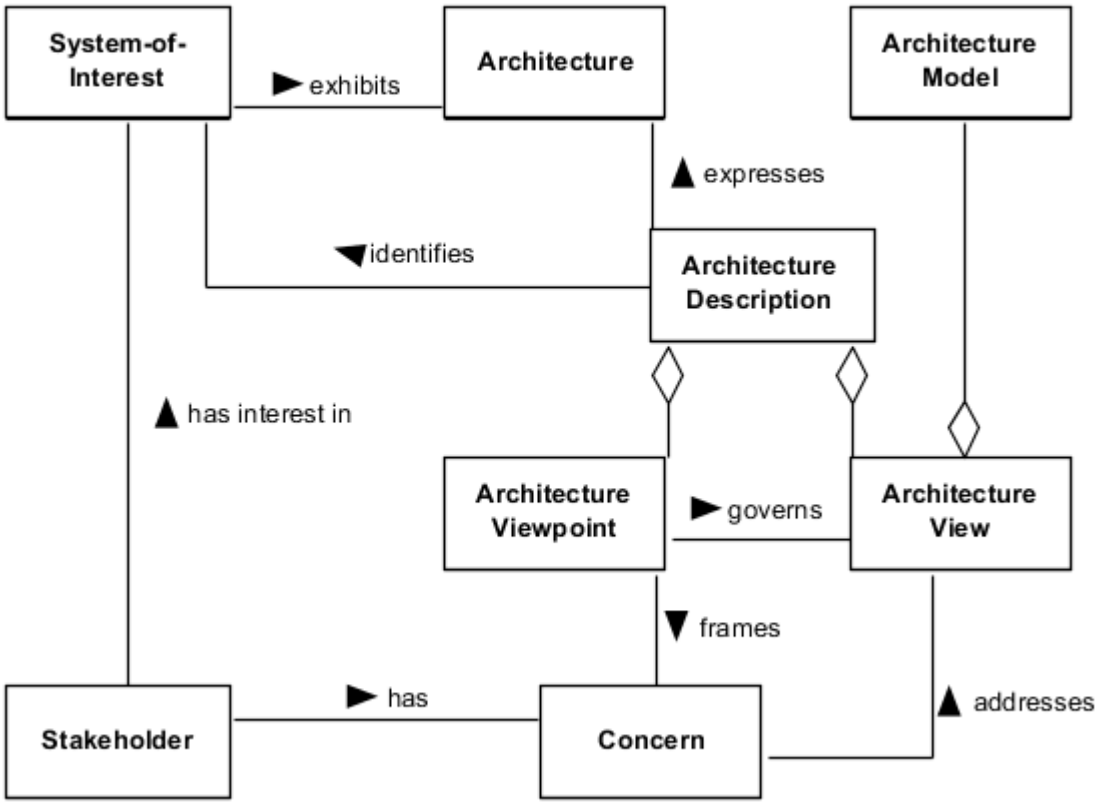

Figure 4 - Architecture description concepts (Adapted from ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description" [IEEE 42010, 2011])

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: Treat 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:

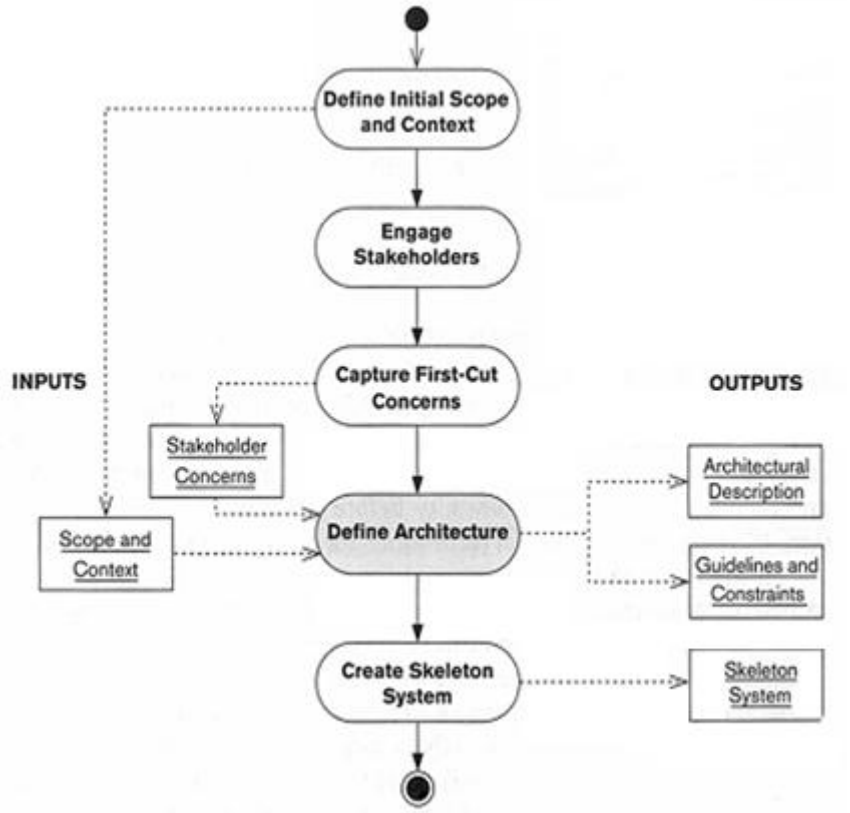

Figure 5 - Activities supporting architecture definition [Rozanski & Woods, 2005]

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 6. 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, such as MESAL. Also companies' and national regulations must be compliedwith and standards should be followed (see D8.5 for more information about standards).

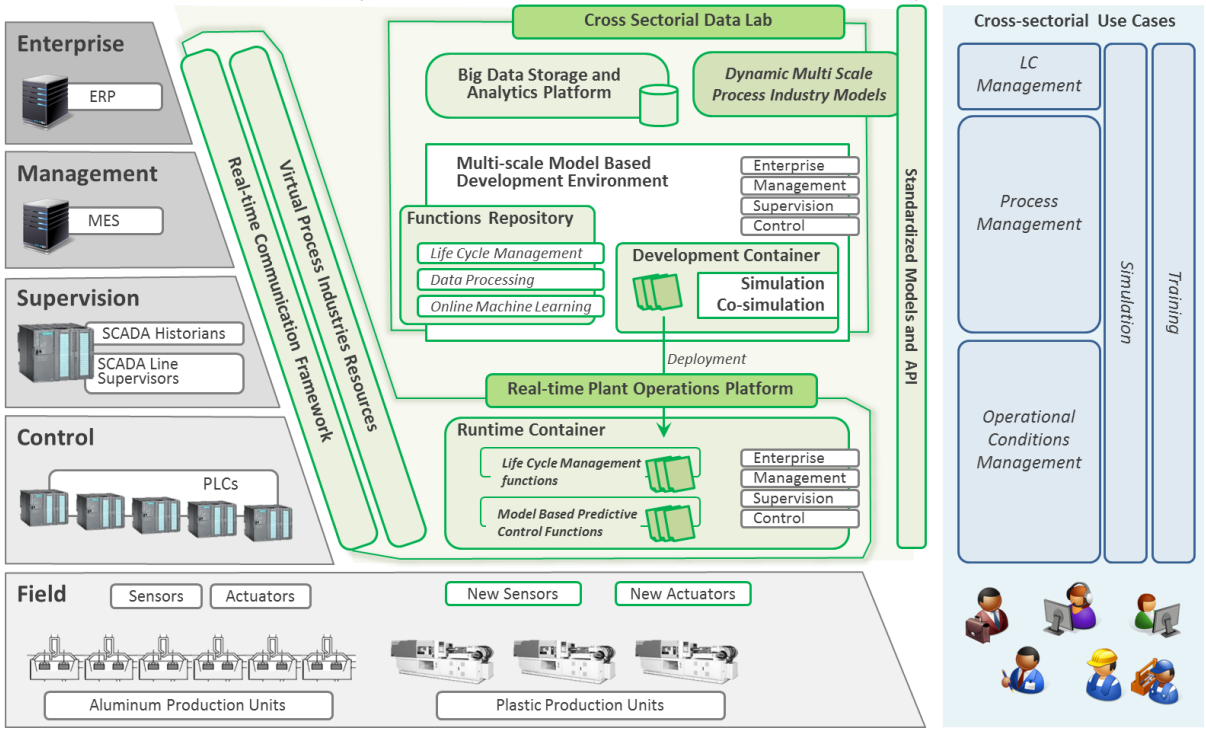

Figure 6 - High level architecture overview

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 6. They each contain several subcomponents and they 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 architecture of the Cross Sectorial Data Lab platform is depicted in Figure 7 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

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.

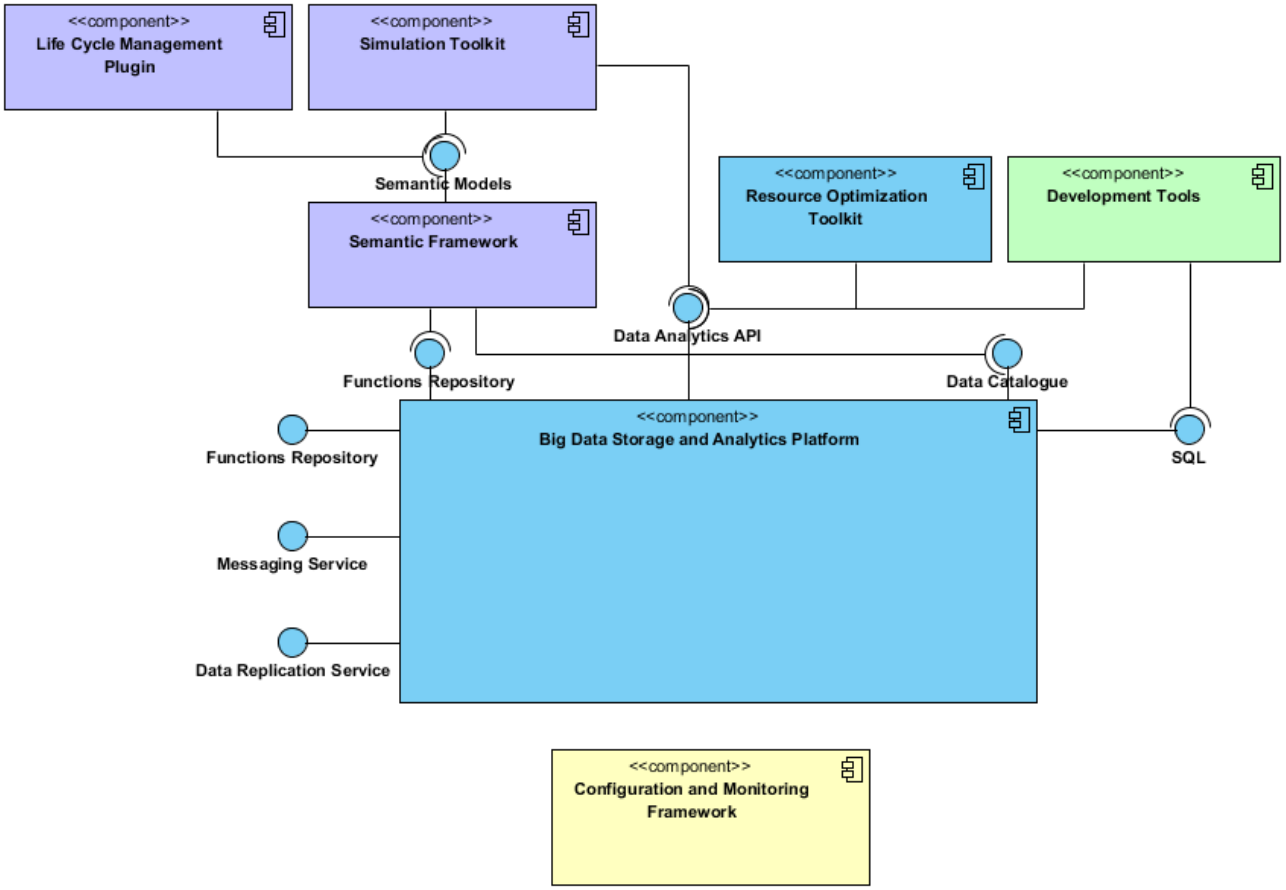

Figure 7 - Cross Sectorial Data Lab Platform

Data Lab platform is connected to the Operations platform using the following interfaces:

- Data Replication Service – This interface allows uploading of large batch historical data collected on the site to the cloud Data Lab storage.
- Messaging Service – This interface allows real-time asynchronous communication between the Data Lab and Operational platform. It is optimized for frequently updated data (e.g. from sensors) with relatively small size of the payload for each update.
- Functions Repository – This interface allows to export predictive functions build on the Data Lab platform and deploy them on the Plan Operations platform for scoring of the operational data.

All interfaces are provided by the Data Lab platform and Plant Operations platform components are acting as the clients.

4.3.2 Big Data Storageand Analytics Platform

The Big Data Storage and Analytics Platform provides resources and functionalities for storage as well as for batch and real-time processing of the Big Data. It provides main integration interfaces between the site Operational Platform and the cloud Data Lab platform and the programming interfaces for implementation of the data mining processes. Internal structure of the Big Data Storage and Analytics Platform is depicted in Figure 8.

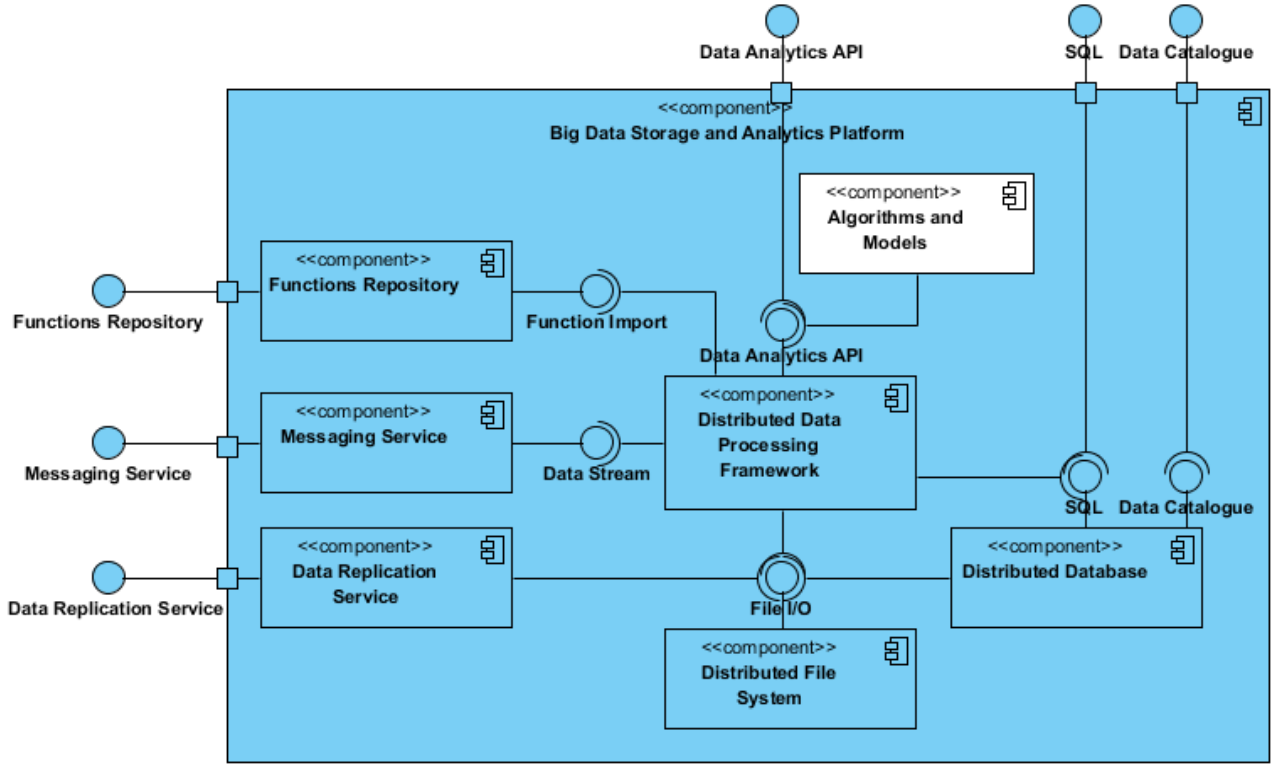

Figure 8 - Internal architecture of the Big Data Storage and Analytics platform

Data is primary stored in the Distributed File System, which is responsible for distribution and replication of large datasets across the multiple servers (data nodes). Unified access to the structured data is provided by the Distributed Database using a standard SQL interface. The main component responsible for data processing is the Distributed Data Processing Framework (development container), which provides high-level application programming interfaces for implementation of the data pre-processing tasks and for the building and validation of the predictive functions. Predictive functions are then stored in the Functions Repository, where they are available for production deployment or for the simulations and overall optimization of the production processes. The rest of the components (Messaging Service and Data Replication Service) provide data communication interfaces and connectthe Operational platform to the Data Lab platform. All components are described in more details in the subsequent chapters.

Distributed File System

Distributed File Systems (DFS) are the distributed variants of local filesystems. The aim of the distributed filesystem is to provide a reliable, scalable filesystem with similar interfaces as local filesystems by providing the same interfaces and semantics as local file systems to access data. Large data files are chunked to multiple parts distributed on multiple servers (data nodes).

The data can either be structured (well organized, such as relational databases) or unstructured (not organized, such as multiple log files and image files). Files are organized in a hierarchical directory structure with one root directory. The file system should support main file operations such as creation, deletion and

writing/reading the content. Modification of the files is an optional feature. Besides the content, the file system maintains for each file or directory a metadata record, such as creation/last modified date or access rights for the users.
The distributed filesystem isrequired to be horizontally scalable to cater growing data needs. Hardware failures are rather norms than exception in distributed environment. Therefore, data can be optionally replicated to many copies in order to achieve data reliability.

Distributed Database
Distributed database provides structured view on data stored in Data Lab platform. It should provide access to the data using the standard SQL language and should provide support for standard RDBMS interfaces such as JDBC for Java or ODBC for .Net platforms.
Data model should be more flexible than traditional relation model, i.e. it should support attributes with nested objects/documents or multivalued attributes. Database engine should implement query federation strategy where multiple files in different formats are directly processed by dedicated "drivers" and engine combines the final results. In this way, it will be possible to evaluate queries without the need to transform data to common format/schema, which can be problematic with very large dataset and require more computational resources.
The distributed databases need to provide real-time or near real-time responses. In addition, the database should provide data security, consistency and integrity

Distributed Data Processing Framework
The distributed processing framework allows execution of application in multiple nodes in order to retrieve, classify or transform the arriving data. This framework provides a low level application programming interface for common algebraic data structures such as vectors, matrices or tensors and standardized library of optimized operations over these structures (i.e. BLAS – basic linear algebra subprograms). This API is intended for implementation of the new data mining algorithms. It should support GPU acceleration.
The framework is expected to provide a data frame API for processing distributed structured data in the form of typed data records. This is the basic API for programming of all data mining phases from data pre-processing to building and validation of the predictive functions. The API should be based on the functional programming paradigm, i.e. the API provides data structures and generic operations such as filter, join, group, split, map and reduce over these structures and the application logic of the data mining process is implemented using closures, i.e. functions passed as the attribute to the generic operations. The API should unify the processing of stream and batch data. The framework provides a library of scalable implementations for common data mining algorithms such as neural networks, decisions trees or linear/logistic regression etc. The latency should be minimal in case of stream processing applications.
Optionally, the frameworks should be capable of handling recovery states so that applications can resume from the failed point after restart.
The Distributed processing framework should contain four main components:

- A component for reading, writing, and managing large datasets coming from the Distributed Database
- A component for providing resource management and delivering consistent operations, security, and data governance tools across the Distributed File System
- An engine for large-scale data processing
- A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services

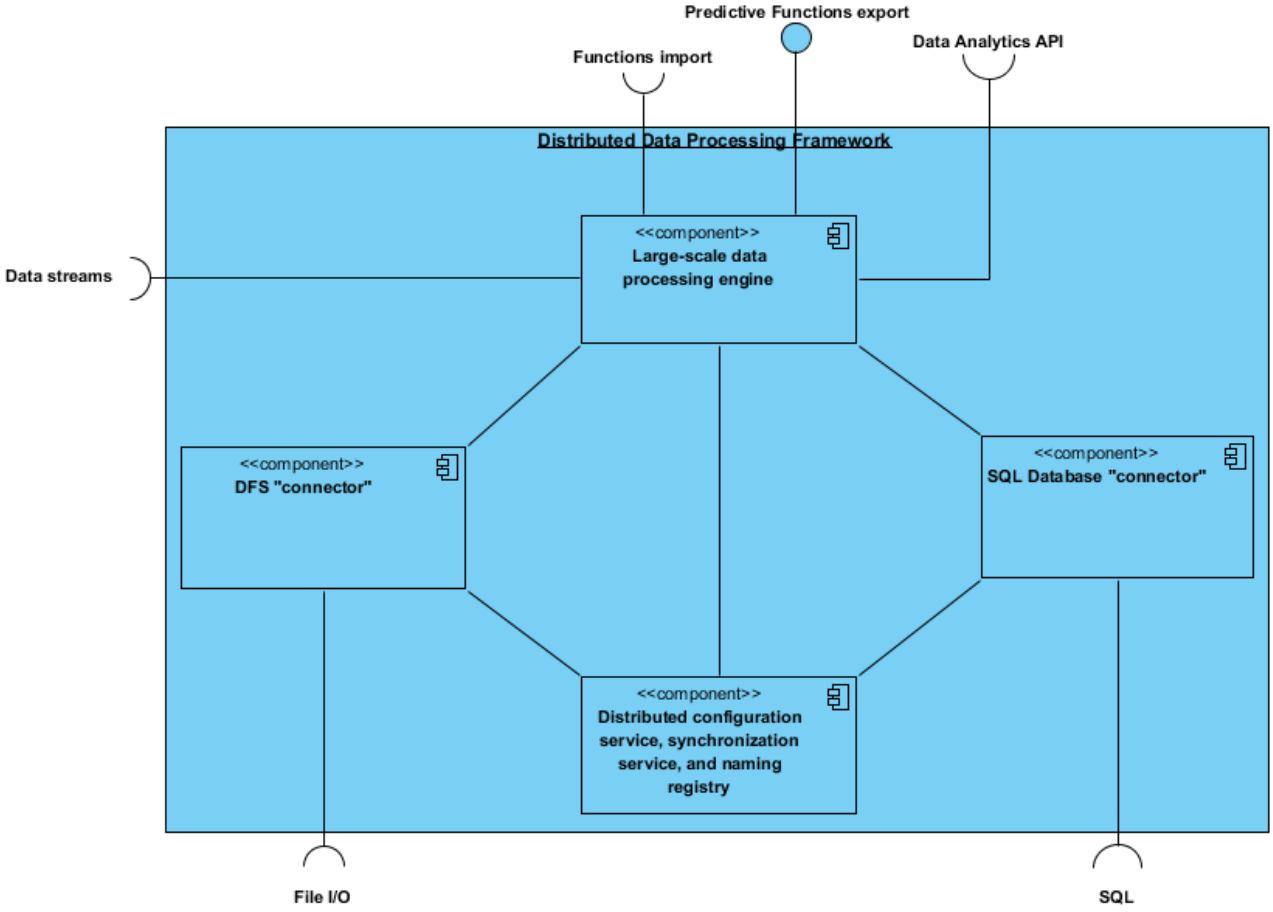

Figure 9 - Distributed Data Processing Framework

Machine learning methods

The Machine Learning methods are algorithms running in the Distributed Data Processing Framework and scripted by Development Tools. They are part of the component "Algorithms and Models" in Figure 8.

The machine learning methods are a key component of the MONSOON platform. A typical goal of these methods is to learn the behaviour of the data, and then make predictions based on this learning. The models are built to bring a solution to a precise use case. Machine learning algorithms are usually divided into two categories:

- Supervised learning, when robust annotation of the data is used to train the model. The model maps the input data to output specified by the annotations. The evaluation of the model can be done "offline", using a set of training and testing data.
- Unsupervised learning, when the model tries to find unlabelled structures in the data. The validation of the model (e.g. the relevance of the retrieved structures) requires inputs from the domain experts.

Note however that what we call a machine learning "model" is not only an algorithm. A model includes the preprocessing of the data, the choice of algorithm, and the adjustment of its parameters.

A development environment from which structured data can be easily accessed, and which includes visualization tools, is necessary for creating the machine learning models. In the Cross Sectorial Data Lab Platform, this corresponds to the "Development Tools". Depending on the use cases, one may need important computation capabilities as well as the capability of handling large volumes of data. The Big Data Storage and Analytics Platform of the MONSOON project fulfil this role.

After the development and evaluation of the machine learning models, the latter must be exported to the Runtime Container. This is done through serialization: the process of exporting a model (including data preprocessing steps, algorithm and parameters) to a file with a specified format, which can then be read in the Runtime Container and used to re-build the model there. Note that the same environment (e.g. dependencies) as the development container is needed for running the de-serialized model in the Runtime Container. Many tools can be used for serialization, such as Openscoring/PMML, Pickle (Python), etc..
In the MONSOON project, the Machine Learning methods put in the Functions Repository are available for the whole consortium. They will be built from collaboration between different partners, and will be re-usable by future users of the MONSOON platform. They should therefore be homogeneously implemented by all partners, for readability purpose.

Functions Repository
The Functions Repository must:

- Provide storage for predictive functions together with all settings required for deployment of the functions. In particular, it includes specifications of all pre-processing operations of raw data.
- Provide web service interface for accessing exported functions and associated metadata (e.g. information about training data, algorithm settings, accuracy on testing data, etc.)

The Functions Repository is in charge of hosting the different machine learning, optimization, life-cycle functionalities which will be developed within the MONSOON project. Machine learning and lifecycle functions are developed using the Development Tools from the Cross Sectorial Data Lab, using standard tools for exploratory data analysis (Apache Zeppelin, Jupyter Notebook, ...).
After being sketched on this platform, functions shall be integrated into the Functions Repository and therefore rolled into the continuous integration system with adequate documentation, testing and dependencies (docker, virtualenv, etc.).

The Functions Repository aims at providing a common set of tools to all the partners of the consortium in order to benefit from the work realized on a given use case to apply it on new ones. As they will be the result of a collaboration between the partners, we emphasize on the need to define a homogeneous standard of implementation.

This repository shall contain three different parts:

- The function library hosts the functions which will be of use for the construction of the different models. There may be a wide variety of such functions (processing, utility, algorithm, etc.). They are the building blocks of the different models and as such, will only be accessible to developers.
- The binary repository hosts scripts which will command the training of models (whole workflow: from the loading of the raw data to data cleaning to feature engineering to training), evaluate the model and serialize the trained model to the export repository.
- The export repository hosts the serialized models. Trained models are serialized (e.g. with OpenScoring/PMML, Pickle, etc.) and dumped in the export repository with an appropriate description of their dependencies (dockerfile, pip requirements). They may then be synchronized with the runtime container through the configuration and monitoring tools for their final use in the plant. We need to keep in mind that these trained models must handle two different connectors (datalab & runtime).

The development of models will require a connection to the database to retrieve structured data. Certain use case may require large storage as well as heavy computation capabilities.

Messaging Service

The Messaging Service implements an interface for real-time communication between the Data Lab and Operations platforms. It provides publish-subscribe messaging system for asynchronous real-time two-way communication which allow to decouple data providers and consumers. On the Operations platform side, this service is connected to the Virtual Process Industries Resources Adapter which publish the data. On the Data Lab platform side, there are multiple clients which subscribe for data for further processing and persistent logging of data into the distributed storage. Messaging service can be also used internally for streaming the data between the components of the Data Lab platform. For this use case, data can be temporarily persistently stored and each message can specify a data retention policy about how long they will be available for clients. Since real-time data can have various semantics and internal structure, data model of messages should not prescribe internal schema of messages and it should be possible to directly send any binary content as the message payload.

Data Replication Service

Similarly to Messaging Service intended for the real-time communication, Data Replication Service provides interface for uploading of the batch data between the Data Lab and Operations platform. Using this interface, plant side components can directly upload historical data to Data Lab distributed storage. This service should provide REST-like interface which allows remote uploading of data. All operations defined by the web service interface are directly delegated to the Distributed File System, i.e. interface should support basic file-oriented operations such as creation of the new file, writing/reading the content, deleting files or listing the files in the folders. Besides the remote protocol for data uploading, Data Replication Service servers also as a security gateway which handle authentication of Operations clients and isolate internal Data Lab environment from internet.

4.3.3 Development Tools

The Development tools are of two kinds:

- Script programming tools – whole data analytics process is implemented as the scripts using the API provided by the Distributed Data Processing Framework.
- SQL analytics tools – provides a user interface for interactive evaluation of the SQL queries and visualization of the results (in the form of tabular reports or graphs).

All tools will be integrated in one web based interface in the form of analytical "notebook" where different part of the analysis (scripts, queries, tables, graphs etc.) are logically grouped and presented in one document.

4.3.4 Semantic Framework

The main goal of semantic modelling for the MONSOON 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 hand side, stakeholders and domain experts need to interpret the results of the analysis. In order to make this communication more effective, Semantic Modelling Framework will define various formalisms and graphical notations. Semantic Framework for Modelling 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.

The semantic models will cover the following main concepts:

- Decomposition of the production process to the phases

- Overall business key performance indicators such as energy and material consumption, product quality or environmental impact and mapping which process phase influence which key performance indicator.
- Logical view of data elements related to the particular 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) value. 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 key performance indicators.
- 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.

4.3.5 **Lifecycle Management plugin**

The Life Cycle plugin is intended to serve as multi-disciplinary, transversal tool to evaluate environmental performance of any processes connected to the MONSOON platform. The scope is to enable the connectors to collect the driver parameters for the investigated process, with special care for material and energy flows; the collection will occur looking at process main characteristics in terms of data availability, quality, and frequency. Once collected, data will be stored in the Big Data Lab, then it will be processed and representative indicators will be calculated; among the whole spectrum of available life-cycle environmental indicators, Global Warming Potential and Total Energy Requirement will represent two reliable and robust global-scale parameters for the evaluation. According to the process nature, additional regional-scale indicators such as acidification potential or water consumption could be computed as well.

The next step consists in the development of the overall life cycle assessment(LCA) component that will be split into two separate nodes:

- KPIs and the overall process are defined in Semantic Framework, developed by TUK according to LCEN specifications
- The user interface and the generated model can be integrated as the plug-in of the Semantic Framework, so the metadata of the Lifecycle management can be used for optimization and simulations

Life cycle components will plug the platform as represented in Figure 10:

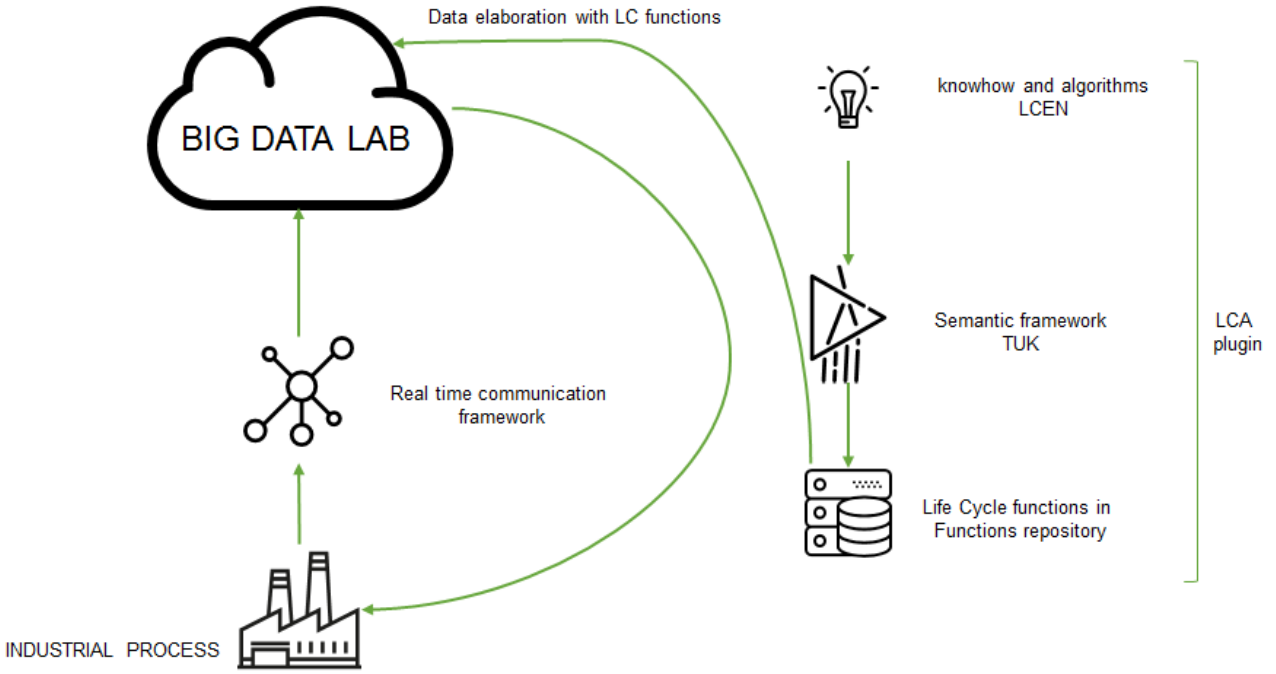

Figure 10– Overview of Life Cycle Plugin in the MONSOON platform

According to data availability from each use case, Life Cycle functions can either refer to theoretical data, coming from professional databases, or to real data, coming directly from management system. Mass and energy balances need to be assessed and validated before to start with the calculations; due to this constraint, references to both time and functional unit shall be carefully addressed. This evaluation should occur during data collection, so that once the platform receives data it can be processed and elaborated effectively.

Functions final output will be a set of indicators, representing the environmental burden of the investigated process; KPIs might be then gathered together and used as an input for the evaluation framework, which is developed in task T7.1.

4.3.6 **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 or for the automatic discrete optimization of the production process by finding the optimal combination of predictive functions for various process phases.
The Simulation Toolkit will leverage semantic models provided by the Semantic Framework including the information about the production process, KPIs and relations between KPIs and predictive function performance. The validation scripts will be implemented using the Data analytics application programming interface and validation tasks will be executed in the Distributed Data Processing Framework. Predictive functions will be instantiated from the Functions repository.
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.

4.3.7 Resource Optimization Toolkit

The main inputs of Resource Optimization Toolkit (Decision Support System) component will be the data provided by data analytics APIs from Distributed Data Processing Framework component, located in Big Data Storage and Analytics Platform component. Resource Optimization Toolkit component will be supported by the knowledge extracted from the development and mainly from the application of Machine Learning Methods component, located also in Big Data Storage and Analytics Platform component, along with knowledge extracted from Simulation Toolkit component, located in Cross Sectorial Data Labe platform. Various indicators will be defined so as to represent the performance of manufacturing process of the plant, in the sense of raw material distribution, energy saving, operational cost, waste reduction etc. Within this component machine learning and deep learning methods will be employed so as to provide resource optimization. The outcome of this component will be an Advanced Monitoring System capable firstly, to cover the needs of better and more accurate information and secondly to automatically screen and access situations that exist in the plant, globally. Below, the functional requirements of the Resource Optimization Toolkit component are given:

Main input(s):

- Cross Sectorial Data Lab (Big Data Storage and Analytics Platform (Distributed Data Processing Framework)
- Cross Sectorial Data Lab (Big Data Storage and Analytics Platform (Functions Repository (Machine learning methods)
- Cross Sectorial Data Lab (Simulation Toolkit)

Main output(s):

- Advanced Monitoring System (Control scenario calibration/fine-tuning)

In Figure 11we present the component diagram of Resource Optimization Toolkit component.

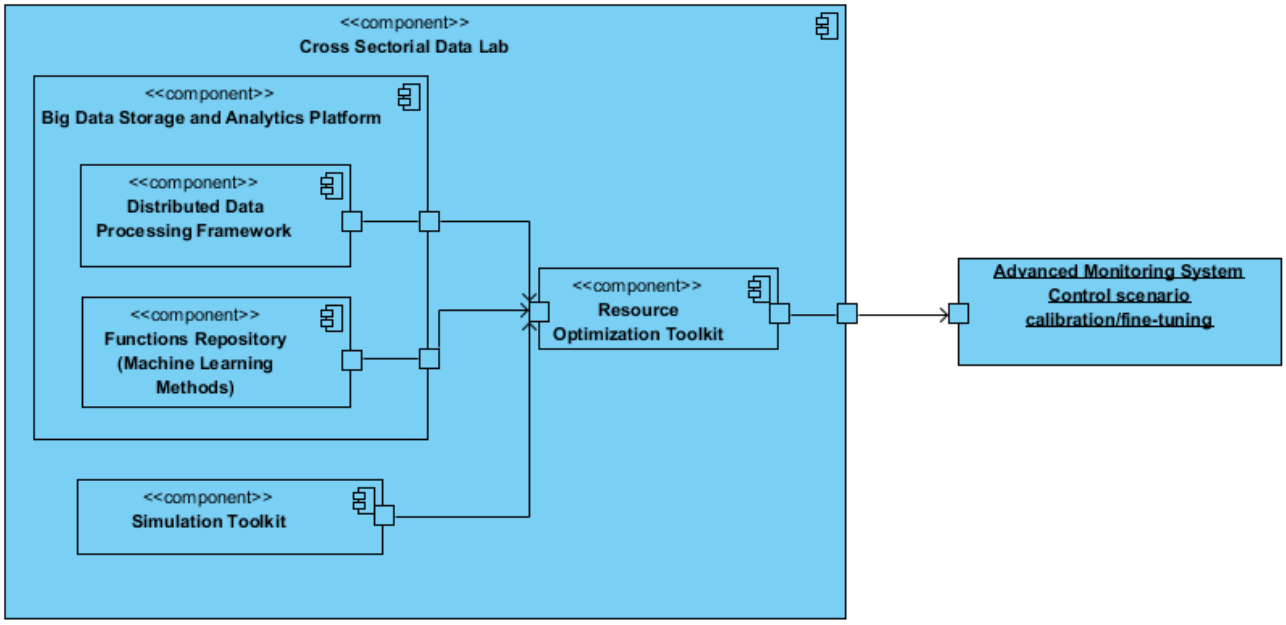

Figure 11– Resource Optimization Toolkit

The development of Resource Optimization Toolkit will be with C++ and Python programming languages.

4.3.8 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 12. It consists of the following main components:

- Real-time Communication Framework
- Virtual Process Industries Resources Adapter
- Runtime Container
- Operational Data Visualization Dashboard
- Trend Analysis Tools

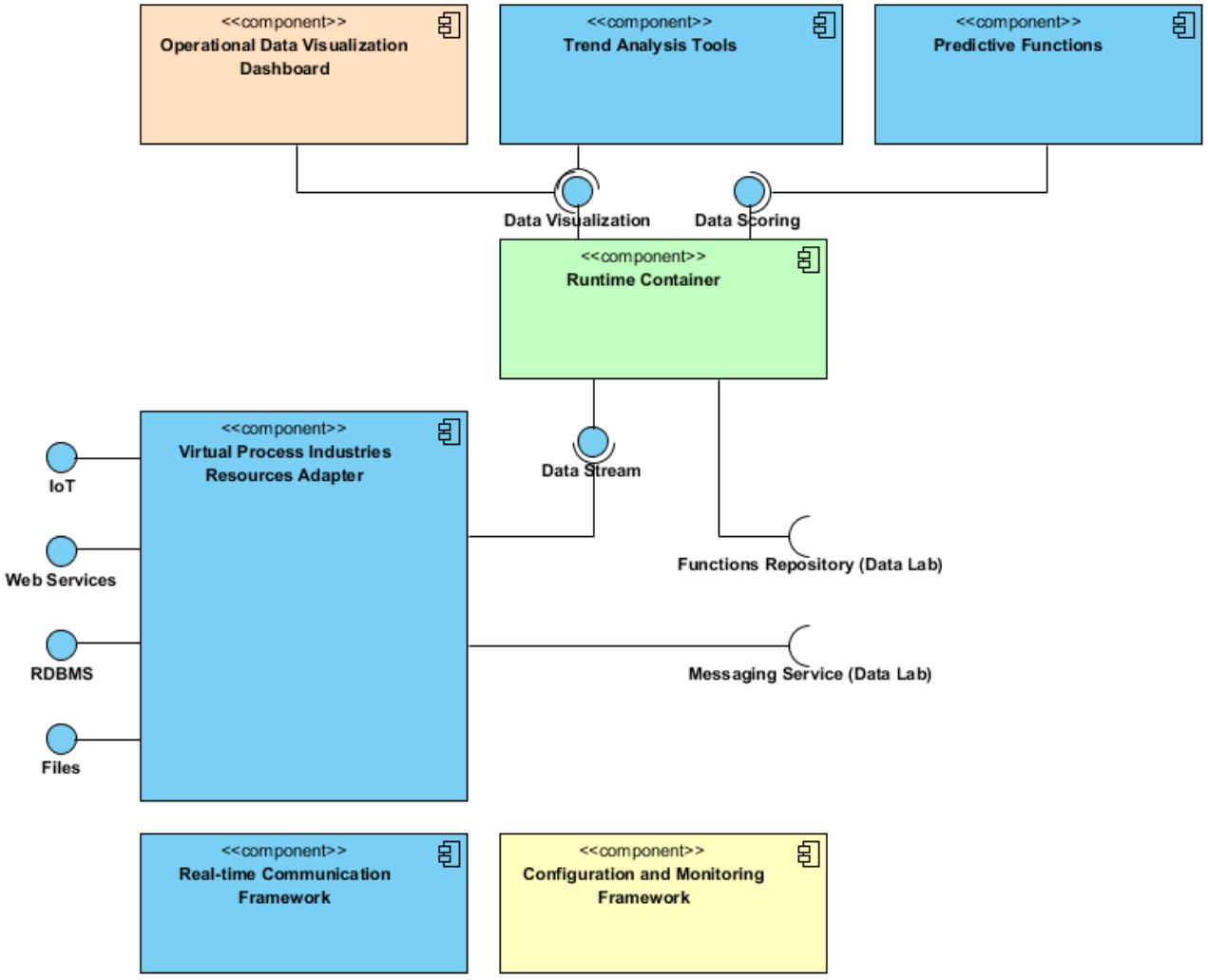

Figure 12– Real-time Plant Operations Platform

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

4.3.9 Real-time Communication Framework

4.3.9.1 Raw data collection: Aluminium domain

The data collection interface in the Aluminium domain is the Historian Pi from ***OSIsoft.*** The Historian Pi is used to archive process data and has two interface levels, namely level 2 (SCADA) and level 1 (PLC) system, where each of the levels can utilize different protocols for its communication. All data from the production units and through the PLCs is stored in the Historian Pi. The information stored in the Historian Pi from the Plant production system includes, but is not limited to events, indicators, sensorvalues and calculated data. The Historian Pi, therefore acts as a data source for the MONSOON Platform in the aluminium domain.

The Connection of the Historian Pi towards the MONSOON Platform is achieved by the MONSOON Pi connector via a gateway server in Capgemini. The MONSOON Pi connector acts as an interface from the Historian Pi to the MONSOON Platform. The MONSOON Pi connector is in fact a real-time web-service, which provides historical data from the Historian Pi; allowing the provision of communication functions, which read "tags" (variables representing different plant production parameters) and save them in flat files to be transferred to the Data Lab. Communication between the Historian Pi and the MONSOON Pi connector uses OLE DB PI provider, based on the TCP/IP communication protocol.

4.3.9.2 Network Monitoring and Downgraded ModeCommunication in the Aluminium Domain

Failure awareness in the IT infrastructure is enabled by monitoring the network communication. Such a process allows monitoring the on-going communication in the network; to evaluate the performance of the network and also to discover anomalies in the network.

This monitoring process can be realized using two methodologies: 1) Active Monitoring and 2) Passive Monitoring. In the **active mode**, network devices can be probed to submit information related to the network traffic (but also for others parameters like CPU, disk usage, etc.) through their interface(s), this is typically done using standardized protocols.While in the **passive mode** a network device only captures relevant data related to the network traffic (without polling or querying other devices), to create overall network traffic scenario.

Failure awareness in the Aluminium domain would be achieved using software with corresponding configurations to allow active/passive monitoring and enabling Failure awareness. Finally in case of error or problems related to the network communication, end users would be notified about the event.

Self-healing in the aluminium domain's Pi Service (in the event of an error, such as the Pi service not responding) is achieved by automatic restart of the service. The service after restart would get and save the exported data since the last reading before it went down. It would also notify the end users about this event.

4.3.9.3 Raw data collection: Plastic domain

Raw data collection in the Plastic domain is realized by utilizing the Euromap63 interface for communication and retrieval of plant system variables from the production plant. The Euromap63 is a standard protocol for communication with the injection moulding machine. Data collected from the plant is stored via the Euromap63 interface to a computer master using the File Transfer Protocol (FTP). The Eurmap63 interface acts as the connection towards the MONSOON Platform.

Similar to the aluminium domain, Failure awareness in the IT infrastructure of the plastic domain is enabled by monitoring the network communications of the network devices. However, since in the plastic domain there is only one production machine, which would connect to the MONSOON Platform, an extensive network monitoring solution is not needed. Nevertheless, the communication with the Euromap63 interface would be achieved using the Active monitoring approach to ensure stable network communication, and allowing the end users to be notified via alerts in case of errors in the network communication.

4.3.10 Virtual Process Industries Resources Adapters

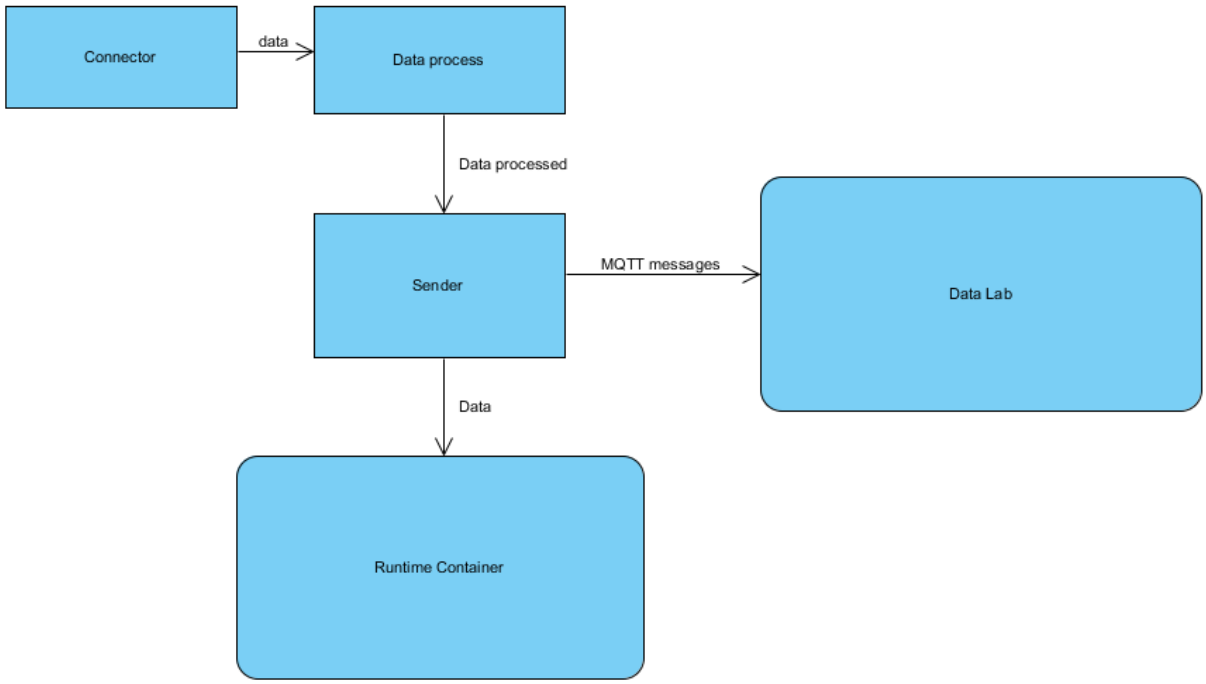

Figure 13– Virtual Process Adapter

Connector: this module aims to retrieve data from the different sources, i.e., the machines in the different plants. In the previous sections, interfaces and communication with these machines in the two domains have been presented, along with the specification of the PI Connector module used to extract data from the Historian PI in the aluminium domain.

Data Process: after the data has been read from the different sources through the connector, a dedicated component is in charge to ensure that raw data can be properly collected, decoded, and enriched with required semantic annotations according to the semantic models defined in the "Cross-sectorial DataLab".

Sender: finally, the last component envelops the data processed and feed predictive control functions real-time container, and the available 'Big Data Storage' systems. At the time of writing this deliverable the 'Runtime Container' component work has not started yet, then the only communication known for the ramp-up phase involves the storage. The protocol envisioned for this communication is MQTT.

4.3.11 Runtime Container

The Runtime container will have the task to execute the predictive functions over the new data collected. We may need some old data and we have to prepare it before the prediction.
We will also need a Data visualization tool and to store the data some were (Data Repository).

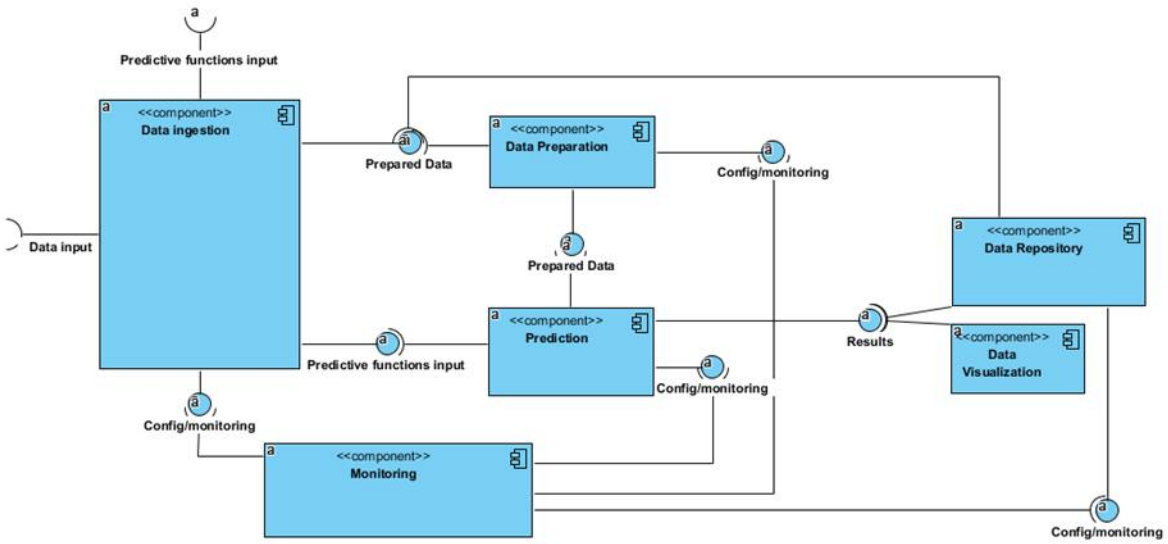

Figure 14– Runtime Container

4.3.12 **Operational Data Visualization Dashboard**

The main inputs of Modula GUI Data Lab/Dashboard component will be operational data from Runtime Container (Real time Plant Operation Platform) component. The outcome of this component will be several visualization aspects, such as graphical charts, bar charts etc.. The development of the dashboard will be based and focused on proper metrics and simplicity. For the rump-up phase of MONSOON project, a few key metrics will be selected initially, so as the managers from aluminium and plastic domains firstly to secure initial successes and usefulness of the selected metrics and secondly to add more strategic elements and complexity. Below, the functional requirements of the Modular GUI Data Lab/Dashboard component are given:

Main input(s):

- Real-Time Plant Operation Platform (Runtime Container (Operational Data))

Main output(s):

- Data Visualization aspects
- Metrics

In Figure 15, we present the component diagram of Modular GUI Data Lab component.

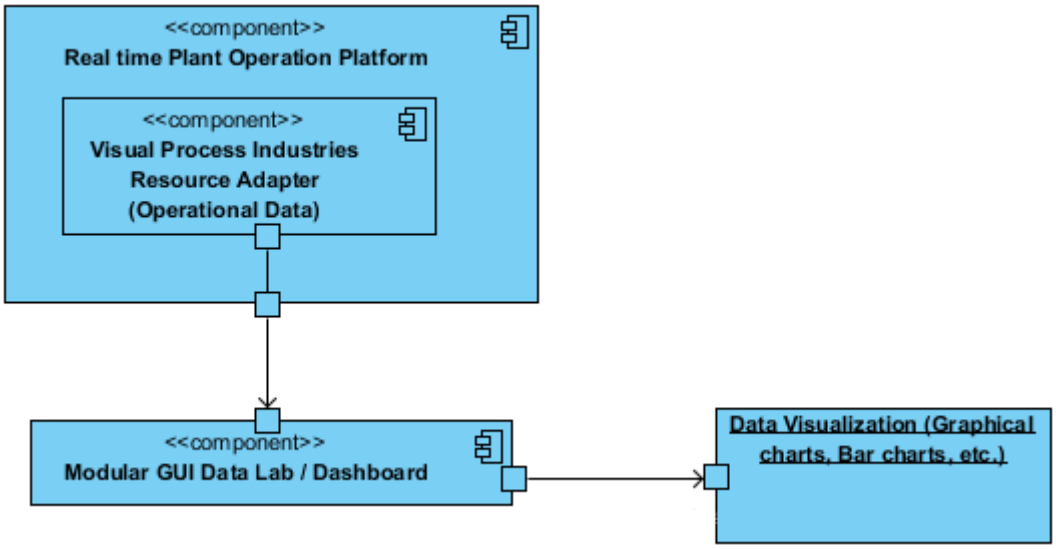

Figure 15– Modular GUI Data Lab

For the needs of MONSOON project, a web based platform will be implemented using HTML5 and Javascript. The implementation of the web platform will also need a range of plugins that are compatible with HTML5. For the implementation of the platform's dashboard, Grafana library will be used. Grafana gives a variety of functionalities such as create new dashboards, or edit or delete a current one. Thus, each user has the ability to customize the dashboard to its own preferences. The library which will be used to create plots is the C3.js which is a parser of D3.js. C3.js provides a variety of charts, such as pie, bar, line, area charts etc.. Its main advantage is that it can easily visualize time series data and also the end user has the ability to zoom in/out on the plot so as to make observation at a particular time slots. The main functionality of the application will be implemented with the utilization of Angular Javascript library. The necessary RESTful services will be invoked by Angular. RESTful services will be implemented using Sitewhere platform.

4.3.13 Trend Analysis Tools

The main inputs of Tool for Trend Analysis component will be operational (real-time) data from Runtime Container (Real-Time Plant Operation Platform) component. The techniques that will be used within this tool provide the opportunity to spot (or detect) structural changes of the linear trend of the examined time series (i.e. machine performance indicators, energy, consumption, quality of raw material, mass pressure and max mass pressure, time of plastification, deformation, time of a cycle of the injection mold process etc.). The trend analysis techniques that will be utilized within this tool will be known (i.e. SSP [Vafeiadis et al. 2011], RTSSP [Vafeiadis et al. 2016], MSSP [Vafeiadis et al. 2016]), properly modified so as to fit to the scenarios of the trend analysis as well as new that will be developed to face new challenges that will probably arise during the pre-processing of the data and requirements of the examined processes.
The knowledge and findings of the Tool for Trend Analysis component will be used as input to the components a) Analytics Platform of the Cross Sectorial Data Lab and b) Dynamic Multi Scale Process Industry Models and c) Data Visualization aspects. Below, the functional requirements of the Tool for Trend Analysis component are given:

Main input(s):

- Real-Time Plant Operation Platform (Virtual Process Industries Resources Adapters (Operational Data))

Main output(s):

- Cross Sectorial Data Lab (Analytics Platform)
- Dynamic Multi Scale Process Industry Models
- Data Visualization aspects

In Figure 16, we present the component diagram of Tool for Trend Analysis component.

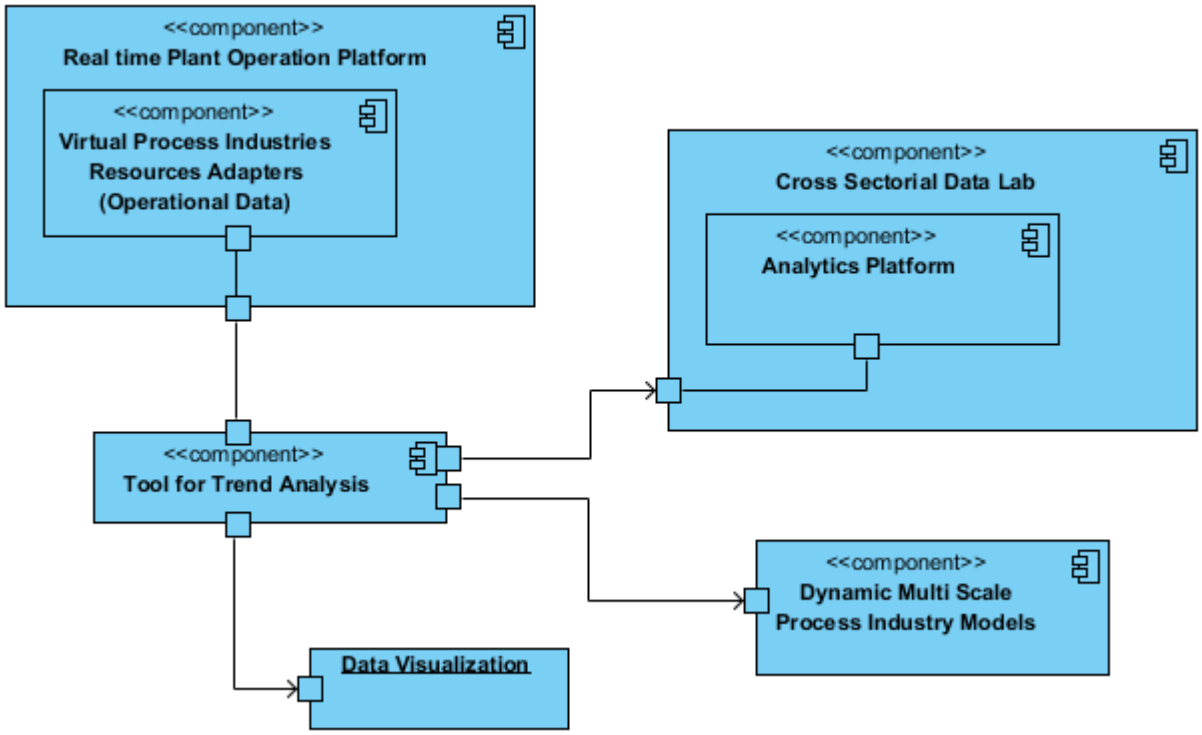

Figure 16– Tool for Trend Analysis

The development of Tool for Trend Analysis will be with Python scripting programming language.

4.3.14 Configuration and Monitoring Framework

Configuration and Monitoring framework provides suite of tools intended for the managing of tasks during the installation and management of all MONSOON platform components. Framework has client-server architecture where the server part is implemented as the service maintaining configuration settings and integrated dashboard user interface where the system administrators can monitor deployment environment, install new platform components or change component's configuration. Client part is implemented as the agent process running on the managed remote server. Agent is responsible for the active monitoring of computation resources (e.g. processors, memory, disks and networks usage) and for the management of installation and configuration tasks. Since the installation of the platform can be complex task, framework should provide support for templates or blueprints for installation of clusters with the servers of various roles and services.

4.4 Information View

Raw data coming from the site plant are transmitted in a homogeneous way to the DataLab where they will be stored, prepared and used to create and simulate Predictive Functions (PF). These PF will be used, in return, by the site Runtime Container in order to produce predictions/analysis on real-time data.
The selection of the needed data in the DataLab is done by the Real-time Communication Framework and, more precisely, by the configurations of the various connectors.
The exact format of all data and flows will be done in future documents of the MONSOON project explaining in details the different use-cases.

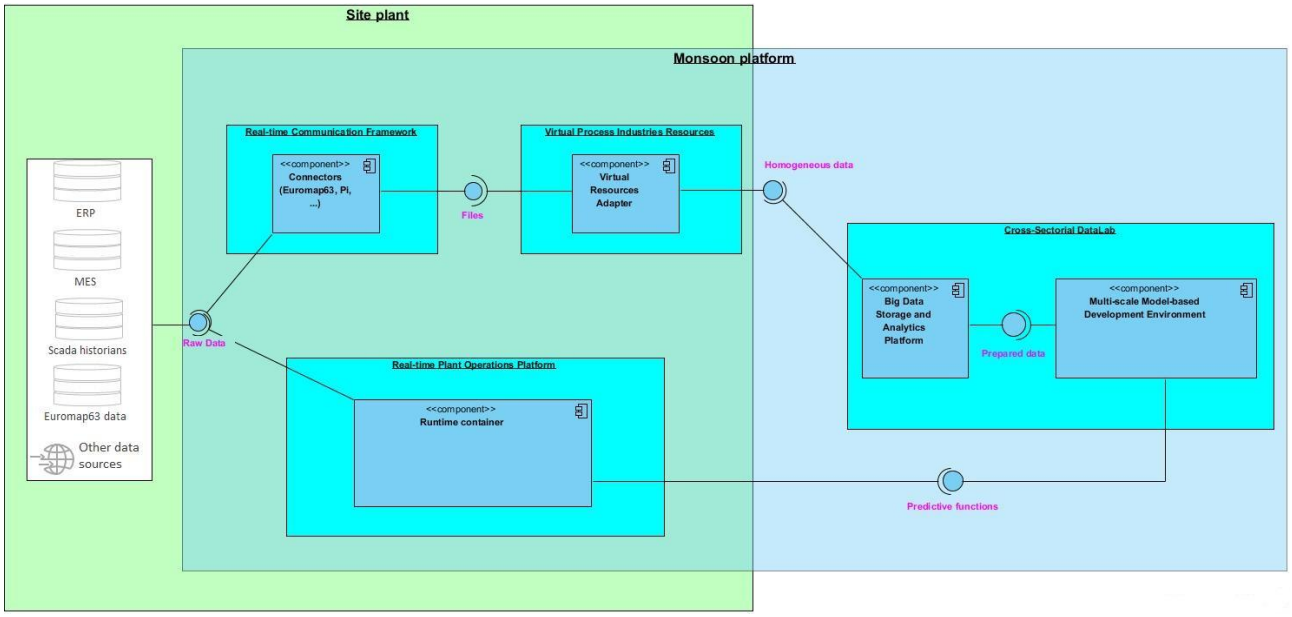

Figure 17– Data processing components

4.5 Deployment View

Deployment view can be divided to various configurations according to the intended usage such as Developing Environment, Testing Environment or Production Environment. Each environment can be further divided to two main parts of the MONSOON platform – Operations Platform and Data Lab platform.

4.5.1 Developing Environment

Since many components will be implemented using the distributed technologies, which require setup of the complex environment and a lot of computation resources, it won't be convenient and efficient to establish complete developing environment with all platform components. Instead of the configuration of the integrated developing environment, each developer should be able to setup local environment for developing the specific component, which will implement only the required interfaces. The implementation of the integration interfaces can be based on the local installation of the implementing technologies or "stub" implementations intended only for testing during the development. This is especially related to the following components:

- Virtual Process Industries Resources Adapter – requires local setup for developing of connectors for new data sources and new data processors.
- Runtime Container – requires local setup for development and testing of new predictive functions and trend analysis methods. This setup should also include local implementation of Data stream interface.
- Distributed Data Processing Framework – requires local set for developing and testing of new data analytics methods and optimization and simulation functions of the Resource Optimization and Simulation Toolkit. This setup should include local implementation of the Distributed File System interface, Distributed Database interface and Messaging Service interface.

4.5.2 Testing Environment

Testing is seen in line with the spirit of continuous integration. One of the basic principles of Continuous Integration is that a build should be verifiable. One has to be able to objectively determine whether a particular build is ready to proceed to the next stage of the build process, and the most convenient way to

do this is to use automated tests. Without proper automated testing, one finds one's self having to retain many build artefacts and test them by hand, which is hardly in the spirit of Continuous Integration.

In the case of MONSOON project, these tests are automated via the Jenkins Continuous Integration server. In Figure 18 a high-level overview of the integration process is given.

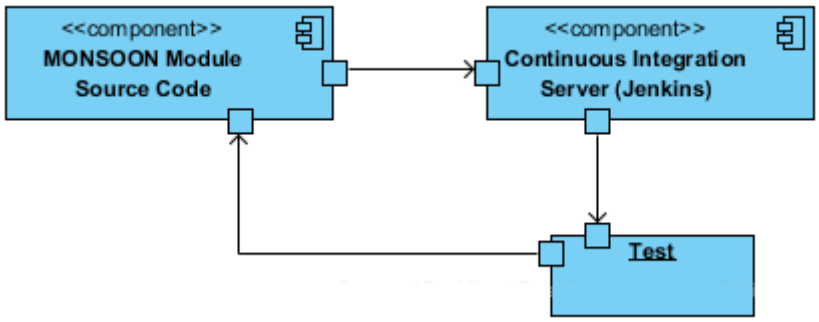

Figure 18– High-level integration process

The flow of the integration process is as follows:

- First, the Continuous Integration server queries the version control repository, where the source code resides (e.g. Git). This is repeated at a specific time interval (e.g. 15 minutes) to avoid launching the process manually.
- Then, the Continuous Integration server attempts to build the module source code. In case of failure the server outputs the error log on the console. If the module builds successfully, the server runs specific tests pertaining to the desired features of the module being tested.

Figure 19 shows the workflow in the Continuous Integration server.

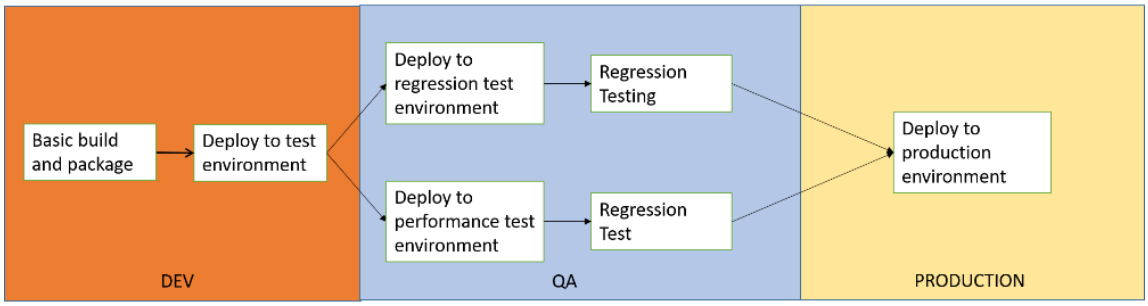

Figure 19– Workflow in the integration server

It is recommended that the Jenkins automation server be used, as it provides a plethora of plugins for automated testing. For more information on Jenkins see section 4.6.1.

4.5.3 Production Environment

The production environment will be deployed on site on each plant. It will manage the MONSOON functions locally and will be used by plant users.

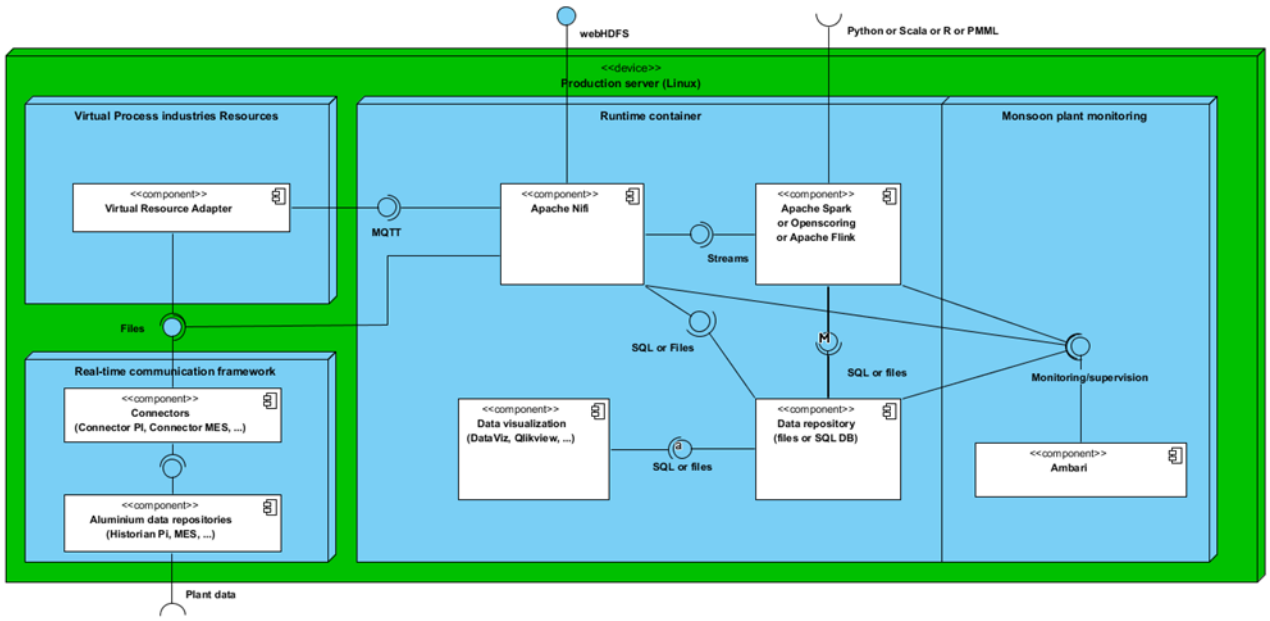

Figure 20– Production environment

4.6 Development View

4.6.1 Tools

As far as source code management is concerned, the platform that has been chosen to host MONSOON's source code is Git.
Git is a version control system that is used for software development and other version control tasks. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows, a perfect fit for the code management requirements of MONSOON. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. It is also free software distributed under the terms of the GNU General Public License.

Git has many desirable features such as:

- Distributed development, with each developer working on a local copy
- Non-linear development, with branching and merging
- Cryptographic authentication of history, change or deletion of a commit is marked
- Pluggable merge strategies, merge auto-completion model is well defined and in case of a conflict the developer can manually merge the files

A Git repository server has been setup on CERTH along with the Redmine environment. The code tracking tool has been integrated with the Redmine web browser interface. All MONSOONpartners have been provided with server access rights for code submission, aiming at the integration of all the modules in one central repository.
Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results etc.

For continuous integration, the platform of choice is Jenkins [Jenkins Automation Server]. Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows for continuous delivery of software by integrating with a large number of testing and deployment technologies.
With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis and much more.
Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.

Advantages of Jenkins include:

- It is an open source tool with great community support
- It is easy to install
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share with the community
- It is free of cost
- It is built with Java and hence, it is portable to all the major platforms

Jenkins has its roots in *Hudson*, a popular open-source, Java-based continuous integration tool developed by Sun Microsystems. After Oracle's acquisition of Sun, Hudson's source code was forked and Jenkins was introduced.
Configuration Management consists of standard processes and techniques often used by organizations to manage the changes introduced to their software products. It helps in identifying individual elements and configurations, tracking changes, and version selection, control, and baselining. Configuration Management tools offer automation in applying configuration states. Below some of the most popular tools are listed.

- **Puppet**
  It is frequently stated that Puppet is a tool that was built with system administrators in mind. The learning curve is less imposing due to Puppet being primarily model driven.

- **CFEngine**
  CFEngine runs on C, as opposed to Puppet's use of Ruby. C is the more low level of the two languages, and one of the main complaints regarding CFEngine is that the learning curve is very steep. It does mean though that CFEngine has a dramatically smaller memory footprint, it runs faster and has far fewer dependencies.

- **Chef**
  Like Puppet, Chef is also written in Ruby, and its CLI also uses a Ruby-based DSL. Chef utilizes a master-agent model, and in addition to a master server, a Chef installation also requires a workstation to control the master. The agents can be installed from the workstation using the 'knife' tool that uses SSH for deployment, easing the installation burden.

- **SaltStack**
  Salt, is developed in Python. It was also developed in response to dissatisfaction with the Puppet/ Chef hegemony, especially their slow speed of deployment and restricting users to Ruby. It supports Python, but also forces users to write all CLI commands in either Python, or the custom DSL called PyDSL. It uses a master server and deployed agents called minions to control and communicate with the target servers, but this is implemented using the ZeroMq messaging lib at the transport layer, which makes it a few orders of magnitude faster than Puppet/ Chef.

- **Docker**
  Docker deploys software applications with all the necessary parts in a container, thereby ensuring it will run on any server, regardless of configuration and/or settings. Containers can be created, configured, and saved as templates for use on other hosts running the Docker engine. These templates can then be used to create more containers with the same OS, configuration, and binaries.

It is useful to note that because of Docker's lightweight and easy containerization technology, it is recommended for use in MONSOON.

4.6.2 Code Organization

As stated above, Git is used as a source code control and management tool to keep track of every part of the code from all components and use it to integrate all modules and components into a final prototype. It is common practice to immediately commit every change to a revision control system, no matter how small it is. The rationale behind is that other developers should always work with the latest version of the code base. The MONSOON Git repository has three top-level directories. The Branches directory is for experimental feature or prototype development; Tags contain a copy of source code for milestones, releases and prototype iterations; and Master is a working source code directory for current development version and is stable development branch. The structure of the source code directory of the Git Master comprises two main subdirectories: Plant-Operations-Platform and Data-Lab. These directories will contain additional sub-directories for their associated sub-components in a flat hierarchy fashion. Names of components in the source repository should be consistent to the latest architecture definition. A description of the sub-components is included in the README.txt file for each component. All public interfaces which need to be shared across components or sub-components are included in API packages of each component. Packages starts with prefix "eu.monsoon" followed by the component name as specified in latest architectural specification. In cases where such prefix is not possible, the other stakeholders and developers should be informed. A source revision repository should contain all files necessary to build the software system, but should not contain any files that can be derived from other files in an automated fashion. Such redundancy only pollutes the repository and is considered bad practice.
To improve the readability of the source code and make its maintenance easier, it is recommended to adhere to coding guidelines and conventions as proposed by programming language environment. For that purpose, it is desirable to use a tool that automates the process of checking code adherence to coding standards. MONSOON components will utilize language-specific tools, such as Checkstyle for java environment and so on.
A "unit" in MONSOON is the smallest testable part of an application, a concise atomic unit of function. Unit tests must be automated and be written using a unit testing framework specific to programming language environment. For Java, this is Junit and for .NET, this is Nunit. For implementations concerning Python language, unittest module is used. For C++ environment, Google's Google test framework is used. Unit tests should be included in the component they belong to, and are placed in a directory called "test". Same pattern is used for configuration and additional resources belonging to the component. Integration tests which can test the compatibility of several components are also part of the component where possible, otherwise, such tests will be placed in a separate dedicated directory.
For each class there should be a test class, which tests the public methods. If a class eu.monsoon.component.Dummy is to be tested, the class name of the test should be eu.monsoon.component.DummyTest. The test class itself and each test method should contain language-specific documentation comments about what is tested and with which parameters. Dependencies to other classes should be substitute by mock objects. Each test case covers exactly one functionality to achieve a quick bug fixing. For each abstract class an abstract test class will be implemented. This abstract test class tests the implementation parts of the abstract class and outlines the correct use of the abstract class and the test classes to implement for the concrete classes. Since they ensure unit's correct behaviour, therefore only

components that passed all unit tests may be committed to a source code repository. Upon completion of an increment of a unit, the following must be considered: code checked into the source code repository must not break the build process; prior to committing new versions of a unit to the source code repository there must be reasonable automated, functional unit tests. There will be no required test coverage criteria, but a test-coverage of 50 percent or higher is advisable. If it is possible, there should be as well tests which provoke exceptions and/or errors. To enhance quality of the software it is recommended to create a new unit test for each detected bug if there was not one already. After fixing the bug the commit statement has to contain the bug number so that an automated tracking of bug fixes can be performed.
The testing approach is preferably "agile" or "lean". This means that MONSOON components should use automatic testing whenever possible. The integration of MONSOON Platform components will be performed continuously and incrementally. The platform's continuous integration approach splits up into a cycle of eight steps described in detail in D6.1.

5 Technical Scenarios and Use Cases

5.1 Use Cases Aluminium domain

5.1.1 Focus on iteration #1

Scope of iteration #1 use cases

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

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

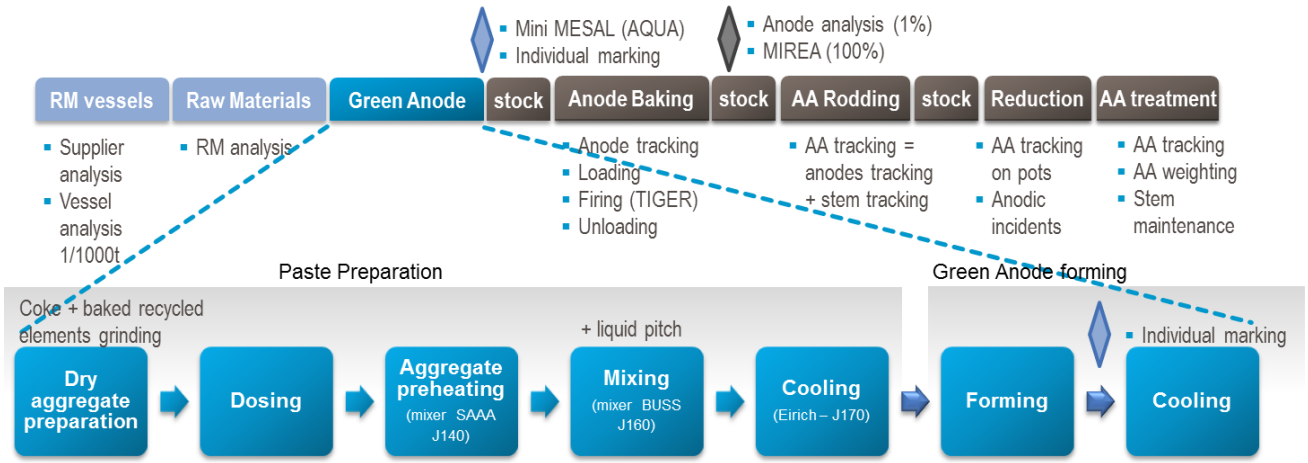

Figure 21–Highlight on the green anode production process (raw material and anode paste plant)

Anodes contribute as the biggest variable impact to the aluminium production cost.
Optimization goals may focus on the anode paste plant to produce cheaper anodes with the same performance in the pots or on the potrooms to use anodes of superior quality to produce metal at reduced cost.

There are numerous optimization goals in the anode plant that can lead to a reduction of the aluminium production cost provided that the anode performance is not negatively affected:

Lower conversion cost:

- Higher throughput
- Less scrap
- Fewer breakdowns
- Lower energy consumption
- Lower maintenance cost

Lower raw material cost:

- Coke / pitch
- Lower pitch content

Pots related optimization goals focus on superior anodes quality:

- 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
- High current efficiency

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.

The first reason of anode density variability at the anode paste plant is the quality of raw materials.
The second reason is the mixing chain and forming equipment stoppage and start-up.

The final objective will be to optimise systematically the process parameters listed here above for example to reduce pitch content, increase paste throughput and ultimately increase anode density.

The business use cases of iteration #1
The 2 first use cases selected for MONSOON project for iteration #1 are:

Predictive Maintenance:
The objective is to anticipate the breakdowns and/or highlight equipment/process deviation that impact green anode final quality (e.g. anode density)

Predictive Anode Quality
The first objective is to qualify the anode quality(green anode for iteration #1):
Predict quality with existing data/sensors and scrap the very bad green anodes at Paste Plant and finally when we will integrate also baked anode process in iteration #2 to manage the potential bad anodes with special operation procedures in potline ("sacrificed" pots receiving systematically the potential bad anodes with particular pot process control settings)
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.

To elaborate these predictive functions, production data will be get from:
- PI system to get data for predictive maintenance
- PI system and MESAL™ (MES system for Aluminium) to get data for predictive anode quality

Data will be transfer to the data Lab platform using MONSOON components:
- PI connector to generate local data in CSV files
- VRA to transfer data to the Data Lab using MQTT

Then predictive functions will be built on data Lab using production data or simulate data. Then a PMML model will be generate per function.
Tools available on data Lab will be used to validate the function (SQL interface, dashboard, visualisation component).
The predictive function developed (PMML model) will be import in runtime container to be able to deploy this function on production environment.

5.1.2 Predictive Maintenance use case

Scope: the whole mixing chain (J) and forming equipment (K)

The MONSOON team has done some early analysis on the available data to better qualify the scope of the predictive maintenance use case.
Our initial goal was to only target the J160 BUSS mixer.
Nevertheless the analysis done and presented hereafter led us to extend the scope to the whole paste mixing chain and anode forming equipment.

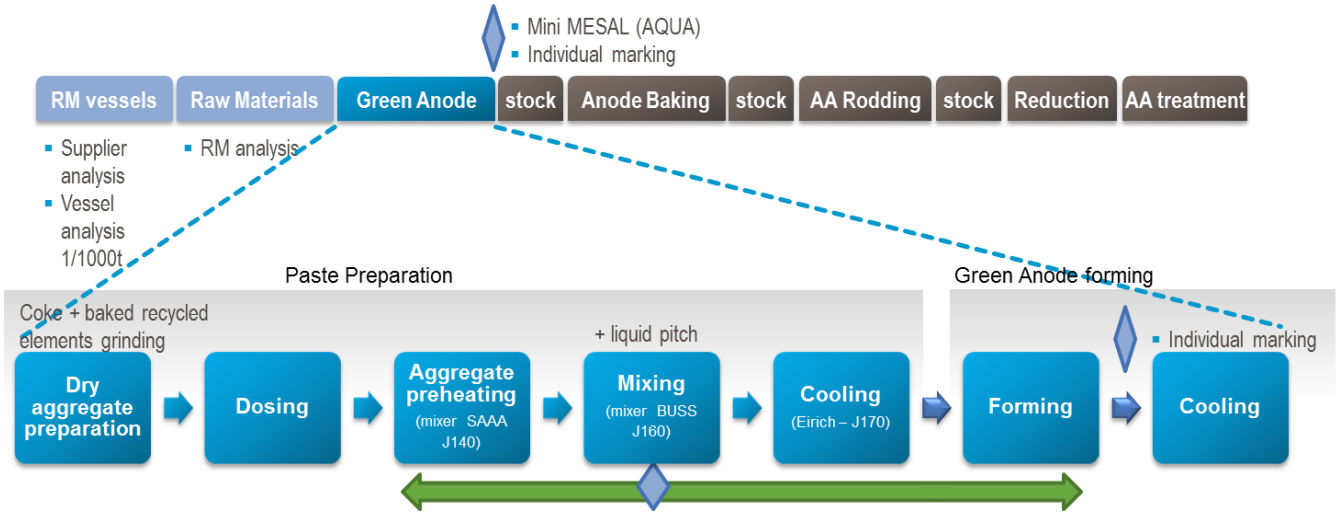

Figure 22–Scope of the predictive maintenance use case

Formalization of the scope of the use case
Available data:

- sensor data extracted from PI
- description of all the stops (faults, conventional, "clean" stops, …)

Scope: we have selected the chains/equipment according to:

- Interruption time
- Correlation between faults (or stop) on the chain and anode quality loss
- Assess if the chain level is sufficient or should we go to equipment level?
- If we want to choose the supervised approach, we will need to precisely define what we call a fault

Based on the interruption time: J & K chains are good targets
The J and K chains account for 73.5% of the time lost to faults:

- J: Mixer (52.2 %)
- K: Vibrocompactor (21.3%)

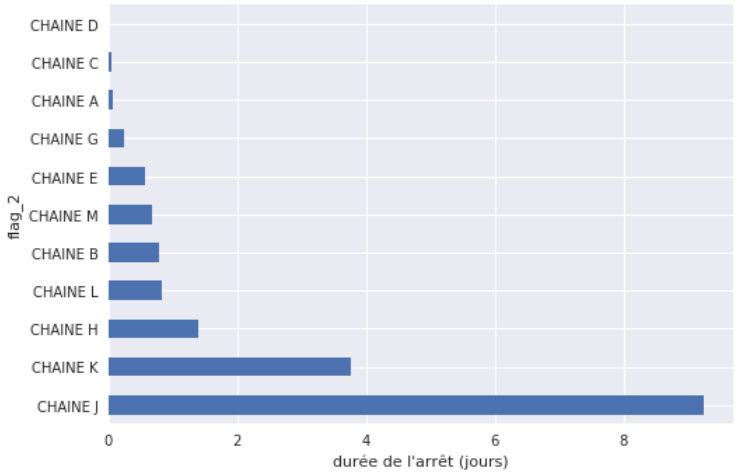

Figure 23– Paste Plant process chains stoppage duration

Since a fault on any part of the process means a full stop on the whole process, we have redefined the scope of this use case to include the whole paste mixing chain (J-chain) and anode forming equipment (K-chain).

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)
- The type of the fault
- Description of the precursors

Approach to address the predictive maintenance use case

Supervised or unsupervised approaches

We have to choose either supervised or unsupervised approaches depending on the annotation and on business goals.

There are two ways of tackling this subject:

1. Predicting certain faults from the equipment chains given the sensors data and the full annotation.
2. Identifying "unexpected behaviours" from the equipment chains given the sensors data and no or only partial annotation

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, …)

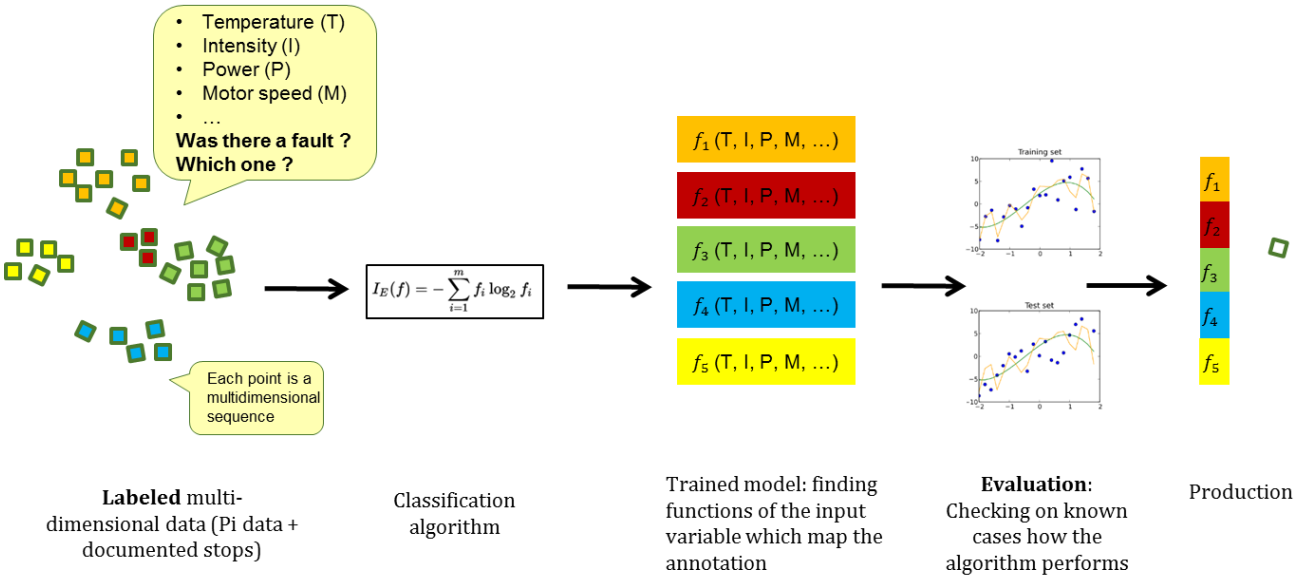

Figure 24– Illustration of supervised learning using full annotation for prediction

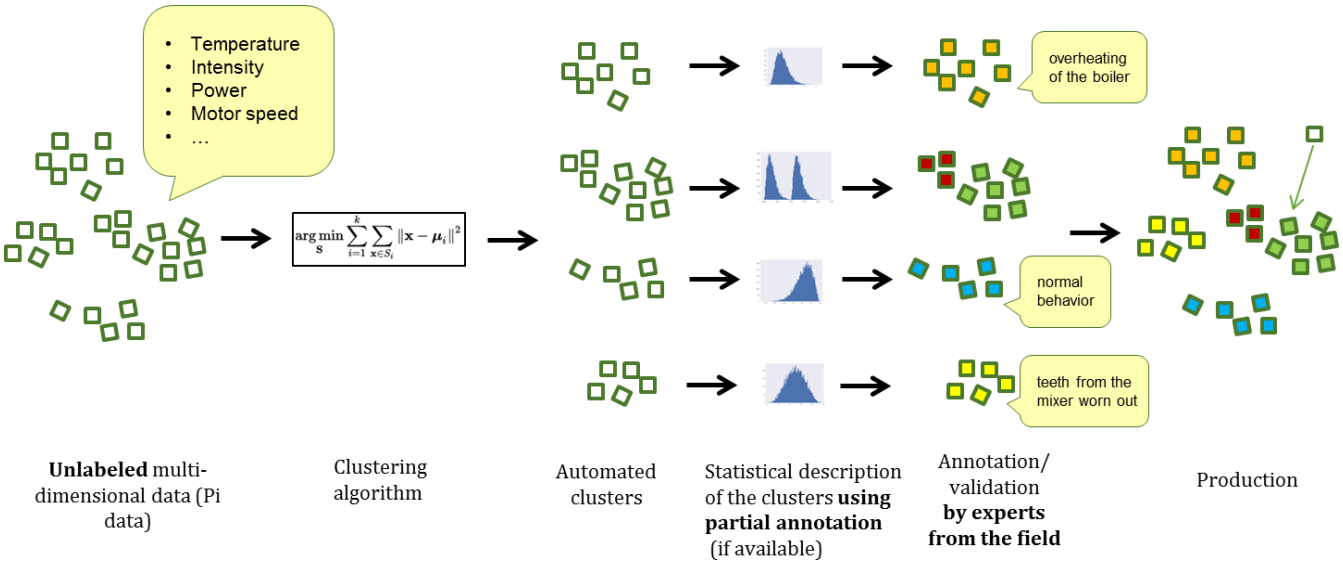

Figure 25– Illustration of unsupervised learning "unexpected behaviours"

The first conclusion of our preliminary analysis is that both supervised & unsupervised approaches are relevant to our situation.

The predictive maintenance use case targets both:

- Predicting identified faults on given equipment (supervised)
- Detecting unexpected and potentially unknown drifts using multivariate analysis (unsupervised)

Both approach should be addressed as they provided us with different yet complementary information.

<u>Proposed workflow:</u>

The proposed workflow to address the predictive maintenance use case is the following:

Scope

- Which are the chains which account for most of the interruption time?

- Which are the chains on which faults are linked to low quality anodes?
- Do we need to work at the equipment level rather than the chain?

Feasibility

- Assessing the availability of the variables related to faults on the chain
- Sufficient number of faults?
- Sufficient associated drops in anode quality?

Annotation

- Assessing the quality of the annotation
- Does it match what we are actually trying to identify?
- Does it cover all cases?
- Is it reliable (timestamp, errors, etc.)?

Modelling

- Bottom-up approach
- Supervised & unsupervised

5.1.3 Anode predictive quality (green process part) use case

Scope of the green anode predictive quality

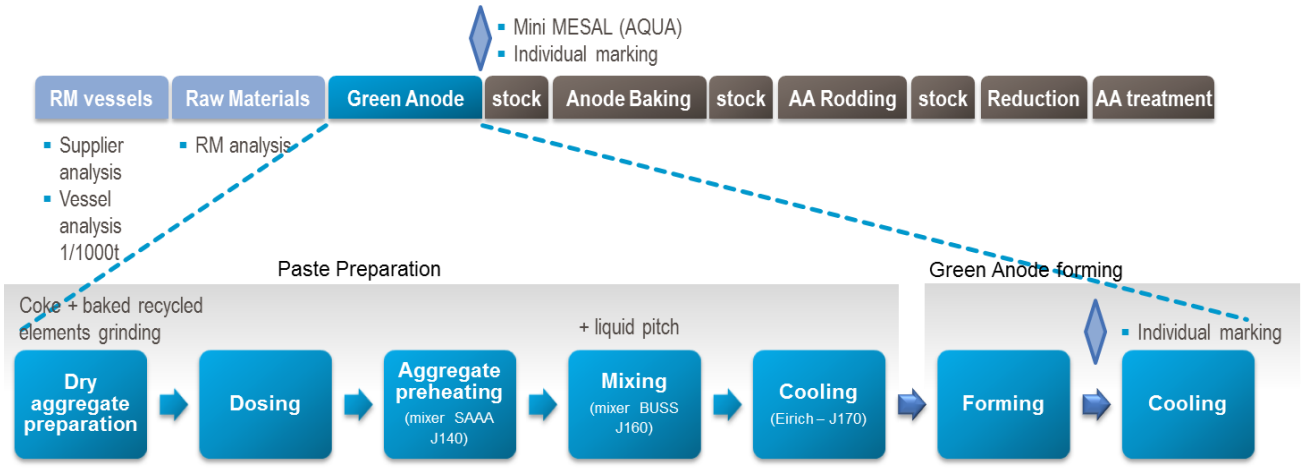

Figure 26–Scope of the anode predictive quality on green process part

Main objectives and approach

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 is manifold. Several definitions were proposed:

1. Predict the density* of the green anode
2. Understand the source of variability in density of a batch of green anodes
3. Predicting which anodes will default

Depending on the definition and final business objectives decided by the Aluminium business domain experts, we will employ a regression or a classification technique. We will use a supervised algorithm.

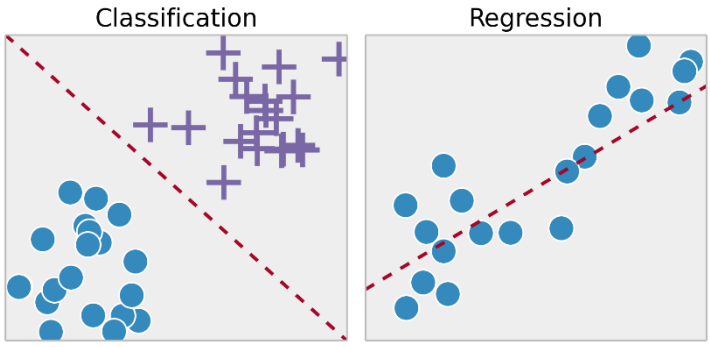

Figure 27– Illustration of classification and regression principles

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

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

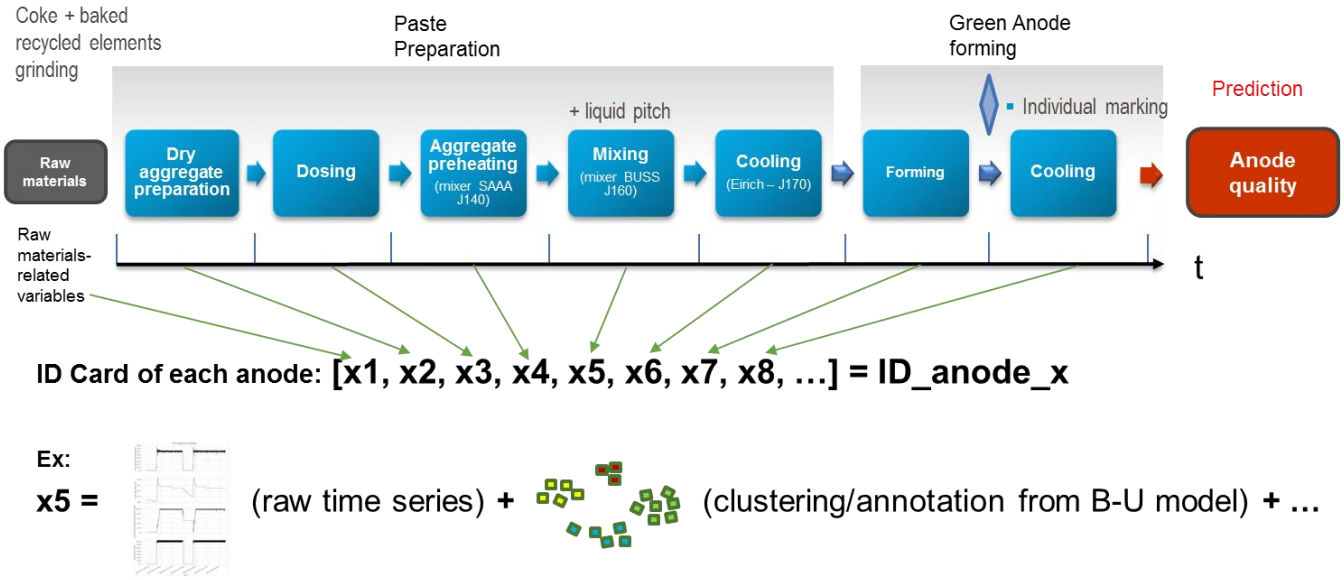

Figure 28–Green anode predictive quality use case approach principle

The first key questions to address with Aluminium business experts:

1. How can we relate each anode to the data corresponding to each step of the production process?
2. What definition should be used for anode quality?
3. Do faults immediately stop the downstream process or is there some kind of buffer (e.g. fault on the mixer, do we keep forming the paste which is already made until we run out or do we fully stop?)
4. Is there a documentation of the duration of each step of the process?
5. Which part of the process is batch? Flow? Are there anode-specific steps (forming)?

<u>Proposed workflow:</u>

The proposed workflow to address the green anode predictive quality use case is the following:

Annotation

- Defining the target variable(s) of our model

- Defining "acceptable" range for this(these) variable(s)
- Check if we have sufficient episodes of anode quality loss

Feature engineering
- Reconstruct the anode "id card": linking all the relevant variables to the anode of interest
- Computing features for relevant representation of the variables

Modelling
- Global approach
- Supervised

5.2 Technical Scenarios and Use Cases Plastic domain

The use cases selected for MONSOON project for period 1 are:
- Use Case 1: Coffee capsules, a mass part with comparable low requirements
- Use case 2: Automotive part (SUBASSY), a high quality part with high requirements

Data will be transfer to data Lab platform using MONSOON components:
- The data concerning use case 1 will directly come from the machine. All needed data will be exported via Euromap63 as csv-files which then can be evaluated by the MONSOON Platform
- The data concerning use case 2 will be fetched with the help of a data acquisition control system. This collects data from various sensors being integrated in the moulding tool as well as predefined data from the injection moulding machine to define the injection moulding cycle and to predict trends concerning the technical parameters from the injection moulding machine.

5.2.1 Scenario 1: The Worker

Steve is a worker in an injection molding company. Part of his job is to keep the quality of the molded parts on a constant high level. Therefore he measures the manufactured parts from each cavity of one complete every hour. He is also responsible to make small changes in the injection molding process if he figures out that the quality of the manufactured parts is decreasing. The manufactured parts are mass parts without high quality requirements.
The company Steve is working for wants to help him by installing a side-wide platform monitoring the injection molding processes Steve is responsible for. The company decided to integrate the MONSOON-platform in the company.
Steve now has a tool where he can store the measured data and where they are evaluated automatically. Steve has the permission to enter these values in a part of the MONSOON platform, called RUNTIME CONTAINER. Furthermore, the values listed inTable 4are monitored directly from the injection molding machine in order to fully describe the specific injection molding cycle and the technical parameter that belong to a specific cycle.

Table 4–Machine parameter for use case 1

Address of the machine parameter	Unit	Machine parameter
ActCntCyc	-	Actual CycleCounter
@010Cyc1Para\CycCntAct.Data	Anz	Cycle
@010CycDataInj1\CycInjTim\CycVal.PdeValue	s	Injection time (cycle value)
@010CycDataInj1\CycScrPosXfr\CycVal.PdeValue	mm	Switchover position to packing pressure (cycle value)
@010CycDataInj1\CycScrPrsXfr\CycVal.PdeValue	bar	Mass pressure at switchover (cycle value)
@010CycDataInj1\CycPlstStr\CycVal.PdeValue	mm	Plastification stroke (cycle value)

Address of the machine parameter	Unit	Machine parameter
@010CycDataInj1\CycHldTim\CycVal.PdeValue	s	Packing pressure time (cycle value)
@010CycDataInj1\CycPlstTim\CycVal.PdeValue	s	Plastification time (cycle value)
@010CycDataInj1\CycInjPrsMax\CycVal.PdeValue	bar	Maximum mass pressure
@010CycDataInj1\CycInjMinStr\CycVal.PdeValue	mm	Mass pad
@010CycDataMld\CycCoolTim\CycVal.PdeValue	s	Cooling time (cycle value)
@010CycDataMld\CycClpClsTim\CycVal.PdeValue	s	Time for closing the mould (cycle value)
@010CycDataMld\CycClpOpnTim\CycVal.PdeValue	s	Time for opening the mould (cycle value)
@010CycDataMld\CycDemold\CycVal.PdeValue	s	Time to eject
@010CycDataOth\CycCycTim\CycVal.PdeValue	s	Cycle time (cycle value)
@010CycDataOth\CycBreakTim\CycVal.PdeValue	s	Break time (cycle value)
@010CycEgConsLCCylHtg\CycVal.PdeValue	Wh	Energy consumption of the Cylinder heating zones during the cycle
@010CycEgConsLCMainDrive\CycVal.PdeValue	Wh	Energy consumption of the main drive during the cycle
@010CycEgConsLCMldHtg\CycVal.PdeValue	Wh	Energy consumption of the Mould heating zones during the cycle
@010CycEgConsLCPeri\CycVal.PdeValue	Wh	Energy consumption of periphery/ automation in whole cycle
@010EJE1.StrAct	mm	Position of the Ejectors
@010EJE1.EjeVelAct	mm/s	Velocity ofthe ejectors
@010Inj1.InjStrAct	mm	Position of the Screw
@010Inj1.ScrPrsAct	bar	Mass pressure
@010Inj1.ScrVelAct	mm/s	Injection velocity
@010Inj1.InjTimAct	s	Injection time
@010Inj1.HldTimAct	s	Packing time
@010Inj1.PlstRpmAct	1/min	Plastification drive
@010MLD.StrAct	mm	Position ofthe mould
@010MLD.ClpFceAct	kN	Clamp force
@020Inj1T1\CycDataTmpZone\CycVal.PdeValue	°C	Cylinder zone 1 (cycle value)
@020Inj1T11\CycDataTmpZone\CycVal.PdeValue	°C	Cylinder zone 11 (cycle value)
@020Inj1T2\CycDataTmpZone\CycVal.PdeValue	°C	Cylinder zone 2 (cycle value)
@020Inj1T3\CycDataTmpZone\CycVal.PdeValue	°C	Cylinder zone 3 (cycle value)
@020Inj1T4\CycDataTmpZone\CycVal.PdeValue	°C	Cylinder zone 4 (cycle value)
@010ActBkPrsSet.Data	bar	Active back pressure

Steve is also responsible for a high quality part for the automotive industry. This part also should be monitored by the MONSOON platform. Several temperature sensors and pressure sensors will soon be integrated in the molding tool by the tool maker department of the company. A data acquisition control system (DACS) will also be ordered soon so that the sensor signals can be evaluated by the MONSOON platform. The DACS will also have the possibility to receive some signals (max. eight signals) from the machine. A possible choice for these is given inTable 5.

Table 5–Machine parameter for use case 2

Address of the machine parameter	Unit	Machine parameter
@010CycEgConsLCCylHtg\CycVal.PdeValue	Wh	Energy consumption of the Cylinder heating zones during the cycle
@010CycEgConsLCMainDrive\CycVal.PdeValue	Wh	Energy consumption of the main drive during the cycle
@010CycEgConsLCMldHtg\CycVal.PdeValue	Wh	Energy consumption of the Mold heating zones during the cycle
@010EJE1.StrAct	mm	Position of the Ejectors
@010Inj1.InjStrAct	mm	Position of the Screw
@010Inj1.PlstRpmAct	1/min	Plastification drive
@020Inj1T11.TmpAct	°C	Cylinder zone 11
@020Inj1T4.TmpAct	°C	Cylinder zone 4

The MONSOON platform is able to evaluate all the monitored data for long-term trends with the help of predictive functions. So Steve now has a powerful tool to further increase the quality of the manufactured parts and to reduce the manufacturing of waste parts.

5.2.2 Scenario 2: The industrial engineer

Robert is the industrial engineer of the company. Amongst other things he is responsible to define parameter belts in which Steve is allowed to optimize the injection moulding process during the manufacturing of plastic parts. As well as Steve he was very happy to hear that the company is going to install a platform monitoring the injection molding processes in the company. After having finished the installation and deep learning phase of the MONSOON platform he now has an instrument to predict tolerance belts for a specific injection molding cycle more precisely. He has a very well understanding of the injection molding cycle and is therefore able to interpret the results from the MONSOON platform in order to define working orders to Steve how he should react if various situations are happening that lead to waste parts.

6 Conclusions

This deliverable describes the initial 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. The initial requirements described here are, as the name suggests, only an initial set, which will likely be changed and expanded during the project as we gain more knowledge about the domain, the process, the intended uses of the MONSOON system and the lessons learned during the project. As we are still in the collection phase for the user requirements, until now there are only technical requirements, which have an influence on the defined architecture. The initial architecture is also defined in this deliverable which is done following 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 used 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.The most important components are described as well as the environments where deployment occurs and tools used for development. This deliverable only gives a brief overview, because for many of these topics separate deliverables will be/were created and/ or development and design start at a later point in time.

For the architecture the same holds as for the requirements: it is the initial version, which will be changed during the course of the project depending on lessons learned and experience and knowledge gained. There will be a final description of the architecture and the requirements: D2.6 Final Requirements and Architecture Specifications which is due in month 21.

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 Markup 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

List of figures

Figure 1 - Screenshot of JIRA with a list of requirements 12
Figure 2 - Screenshot of JIRA with a list of requirements 13
Figure 3 - Structure of the Requirements Workflow 14

Figure 4 - Architecture description concepts (Adapted from ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description" [IEEE 42010, 2011]) 18
Figure 5 - Activities supporting architecture definition [Rozanski & Woods, 2005] 19
Figure 6 - High level architecture overview 21
Figure 7 - Cross Sectorial Data Lab Platform 22
Figure 8 - Internal architecture of the Big Data Storage and Analytics platform 23
Figure 9 - Distributed Data Processing Framework 25
Figure 10 – Overview of Life Cycle Plugin in the MONSOON platform 29
Figure 11 – Resource Optimization Toolkit 30
Figure 12 – Real-time Plant Operations Platform 31
Figure 13 – Virtual Process Adapter 33
Figure 14 – Runtime Container 34
Figure 15 – Modular GUI Data Lab 35
Figure 16 – Tool for Trend Analysis 36
Figure 17 – Data processing components 37
Figure 18 – High-level integration process 38
Figure 19 – Workflow in the integration server 38
Figure 20 – Production environment 39
Figure 21 – Highlight on the green anode production process (raw material and anode paste plant) 43
Figure 22 – Scope of the predictive maintenance use case 45
Figure 23 – Paste Plant process chains stoppage duration 46
Figure 24 – Illustration of supervised learning using full annotation for prediction 47
Figure 25 – Illustration of unsupervised learning "unexpected behaviours" 47
Figure 26 – Scope of the anode predictive quality on green process part 48
Figure 27 – Illustration of classification and regression principles 49
Figure 28 – Green anode predictive quality use case approach principle 49

List of tables

Table 1 - Stakeholders Aluminium Domain 5
Table 2 - Stakeholders Plastics Domain 5
Table 3 - Other Stakeholders 6
Table 4 – Machine parameter for use case 1 50
Table 5 – Machine parameter for use case 2 52

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.

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 forArchitectural 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.

Appendix

Requirements

Issue key	Summary	Requirement Type	Priority
MON-1	Support for publish-subscribe communication pattern	Functional	Medium
MON-2	Streaming real-time data from Messaging Service to Distributed Data Processing framework	Functional	Medium
MON-3	Support for MQTT protocol	Functional	Major
MON-4	REST-like remote interface for uploading of batch data to Data Lab	Functional	Medium
MON-5	Support for Distributed File System operations	Functional	Medium
MON-6	Support for storage of very large files	Functional	Medium
MON-7	The data scientist can use SQL-like queries for structured datasets	Functional	Medium
MON-8	The data scientist can query for data in sql syntax over reach structured data	Functional	Minor
MON-9	Support for queries over BigData file formats such as Parquet or Avro	Functional	Minor
MON-10	Support for JDBC/ODBC standards	Functional	Minor
MON-11	The administrator can manage the platform via an integrated web-user management console	Functional	Medium
MON-12	The administrator can use active monitoring of Data Lab services	Functional	Medium
MON-13	Support for batch processing of large data sets	Functional	Major
MON-14	High-level functional Application Programming Interface for batch and streaming processing	Functional	Major
MON-15	Support for iterative computation and shared variables	Functional	Medium
MON-16	The operator needs to be able to do Network Monitoring in Plastic Domain	Functional	Medium
MON-17	The data scientist has access to a visualization of data in the data lab	Functional	Major
MON-18	The operator is able to monitor the status of MONSOON components on a dashboard	Functional	Medium
MON-19	MONSOON Platform is able to collect and store data	Functional	Major
MON-20	The Data Scientist can also use machine learning algorithms which are not included in Apache Spark	Functional	Medium
MON-21	The data scientist is able to serialize machine learning models	Functional	Major
MON-22	The operator can deploy components in the data lab and the runtime container in a similar fashion	Non-Functional - maintainability	Major
MON-23	The operator can do deployment of applications inside software containers in the Functions Repository	Non-Functional - maintainability	Medium

MON-24	The data scientist must have an access to the data from the Functions Repository	Functional	Major
MON-25	The developer can use data formats in the Data Lab and Runtime Container respectively without additional data transformation	Non-Functional - maintainability	Major
MON-26	The data scientist is able to visualize the data with the Model Development Tools	Functional	Medium
MON-27	The developer must always have access to an up-to-date, working, tested and documented version of the functions located in the Functions Repository.	Functional	Medium
MON-28	The data scientist can execute scripts in the Functions Repository	Functional	Medium
MON-29	The data scientists can store the machine learning models, once ready to be exported, in a dedicated repository in the Functions Repository	Functional	Minor
MON-30	The data scientist needs to have access to data originating in the production sites	Functional	Major
MON-31	The operator can do deployments of applications inside software containers in the Runtime Container	Non-Functional - maintainability	Medium
MON-32	The data scientist has access to a visualization of the results of the predictive functions	Functional	Major