Project Number 723634

D4.5 Final Specification of Situational Awareness Services

Version 1.0
16 October 2018
Final

EC Distribution

ATB

Project Partners: **ATB, Electrolux, IKERLAN, OAS, ONA, The Open Group, University of York**

PROJECT PARTNER CONTACT INFORMATION

ATB Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany Tel: +49 421 22092 0 E-mail: scholze@atb-bremen.de	**Electrolux Italia** Claudio Cenedese Corso Lino Zanussi 30 33080 Porcia Italy Tel: +39 0434 394907 E-mail: claudio.cenedese@electrolux.it
IKERLAN Trujillo Salvador P Jose Maria Arizmendiarrieta 20500 Mondragon Spain Tel: +34 943 712 400 E-mail: strujillo@ikerlan.es	**OAS** Karl Krone Caroline Herschel Strasse 1 28359 Bremen Germany Tel: +49 421 2206 0 E-mail: kkrone@oas.de
ONA Electroerosión Jose M. Ramos Eguzkitza, 1. Apdo 64 48200 Durango Spain Tel: +34 94 620 08 00 jramos@onaedm.com	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of York Leandro Soares Indrusiak Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325 570 E-mail: leandro.indrusiak@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial Document Structure	3 September 2018
0.4	Intermediate version with updated specifications	19 September 2018
0.8	Further refinement of specifications after internal reviews	8 October 2018
1.0	Final version	16 October 2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE SNIPPETS

EXECUTIVE SUMMARY

The SAFIRE project provides technology and infrastructure to enable reconfiguration as a service for dynamic smart factory systems and manufactured smart products. To improve the efficiency, security and context-based adaptation of these systems and products, manufacturers need more information about affects in their lifecycles during their use, and how product design affects the production processes. SAFIRE addresses these challenges through developing, combine and verify big data analytics approaches, situational awareness services, security-policy, as well as optimisation and reconfiguration approaches.

This document provides the final specification of situational awareness services composed in the situation determination module as part of the SAFRIRE solution. This module processes data coming from connected systems / devices / products (data producers) but also from data analytics result providers to extract the current situation of these connected systems / devices / products. The document describes next to the specification also (1st) the novelties of the situational awareness services in section 2, (2nd) the business case specific customisation and configuration of the situational determination module in section 6, and (3rd) the requirement coverage of the situational awareness services in the early and full prototype in section 7.

1. INTRODUCTION

This deliverable represents the final specification of Situational Awareness Services of the SAFIRE project and summarises the work realised under the task T4.2 Specification of Modelling of Correlation between Information Sources, Products and Situations.

The objective of the present document is to provide an overview of the requirements and the corresponding specification of the Situational Awareness Services functionalities. Furthermore, the SAFIRE infrastructure is specified and the approach for the implementation is describe within this document.

1.1 OVERVIEW

Based on the results from T1.1, Business Cases analysis, and the results from T1.4, the SAFIRE Concept definition, services and algorithms for situational-aware-based monitoring and situation determination are specified.

The overall Situation Determination process follows the structure of sensory systems or CPS to collect & deliver situation relevant data about processes / equipment / products to reason for this situation based on an Ontology, which is further evolved and enhanced through similarity measures.

Figure 1 shows the conceptual architecture for the Situational Awareness Architecture the is specified in this document.

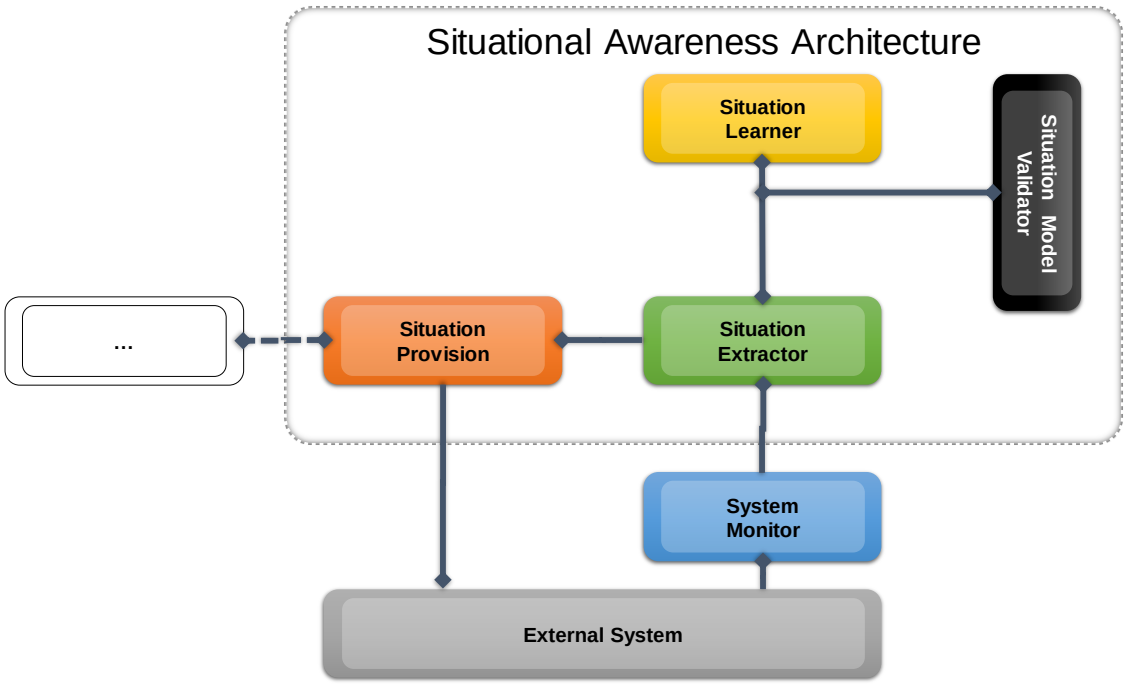

Figure 1: Conceptual Situational Awareness Architecture

1.2 PROGRESS BEYOND D4.2 EARLY SPECIFICATION OF SITUATIONAL AWARENESS SERVICES

The progress beyond the early prototype specification (D4.2), documented in this deliverable, is introduced in the following.

- **Situational Model** - the situation models were reviewed to model more accurately the environment of operation of the SAFIRE solution so that it allows for situational awareness. Towards this direction, the entities of the generic situation model was updated with more suitable terminology and adequate definitions, which allow non-domain experts to understand the scope of situational awareness within the SAFIRE solution. Additionally, there was a redefinition of the relations between the model entities, to describe the interaction between the different aspects of the SAFIRE situations.
- **Integration of Security** – the full prototype specification refines the integration of the SAFIRE Security, Privacy and Trust (SPT) framework for the situation monitoring and situation determination services.
- **Integration of the Optimisation Engine and Predictive Analytics Module** – the full prototype specification refines integration and the communication between the Situational Awareness Services and the Optimisation Engine according to a Metrics API defined by the Optimisation Engine (using HTTP POST request). Additionally is specified the integration with the Predictive Analytics using Kafka as communication channel to send Predictive Analytics results to Situational Awareness Services via defined Kafka topic strings and a serialised xml-based format of analytics results.

1.3 DOCUMENT STRUCTURE

The document consists of:

- Section 1, which provides an overview of the topics addressed in this document and the document structure.
- Section 2, which provides an overview of situational awareness module in SAFIRE, and the innovation beyond the state of the art.
- Section 3, which provides the specification of the situational awareness services and the situation model in SAFIRE.
- Section 4, which provides the background technologies used for the situational awareness services in SAFIRE.
- Section 5 specifies the final SAFIRE Infrastructure.
- Section 6, which describes how the generic situational awareness services have to be customised and configured for application in different use cases.
- Section 7, which provides an overview of the covered requirements with respect to situational awareness and infrastructure.

2. NEW TECHNOLOGIES / INNOVATIONS

This deliverable specifies the situational awareness services composed in the situation determination module as part of the SAFRIRE solution. This module is able to process data coming from connected systems / devices / products (data producers) but also from data analytics result providers to extract the current situation of these connected systems / devices / products.

The situation determination module is divided into three parts. The situation model (specified in section 3.1), the situation monitoring component (specified in section 3.2) and the situation determination component (specified in section 3.3).

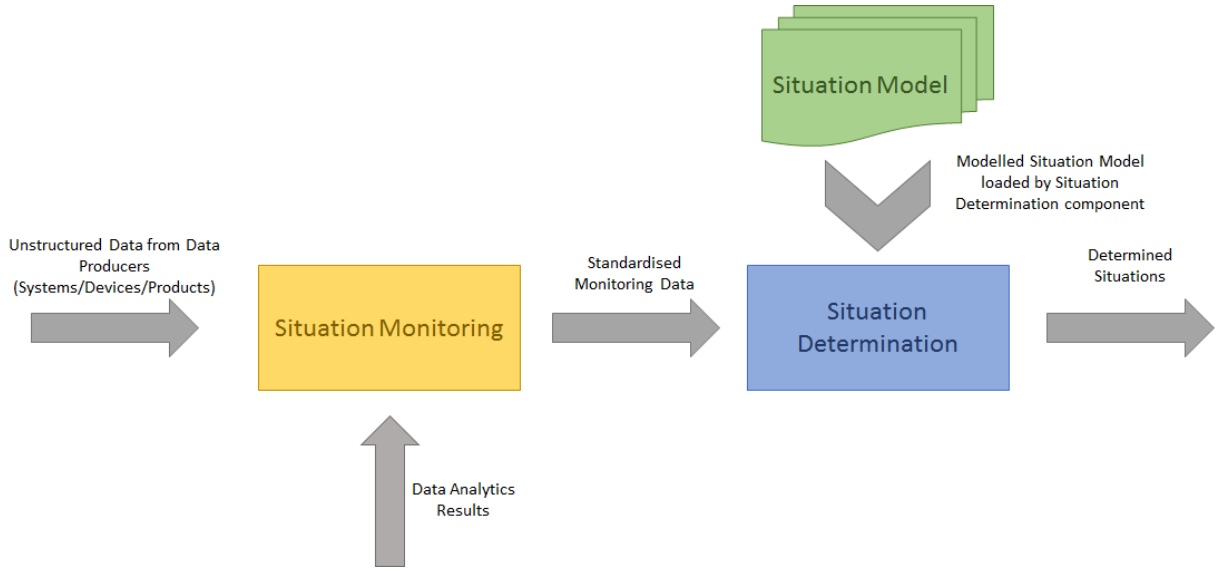

Figure 2: Overview Situation Determination Module

Figure 2 shows an overview of the components that comprise the situation determination module and shows the data processing flow. The Situation Monitoring component gets data from monitored data producers and amalgamate the data. Therefore it

- wraps raw data coming from data producers (different communication protocols and data structures) and data analytics results into a standardised format
- filters data as needed for situation determination
- pre-processes data (aggregations, calculations, combinations)
- enriches data with meta-data (e.g. ID for identification, specifications for limits, data types, etc.)
- and amalgamates data using extended data models for a data mapping.

Standardised monitored data are being forwarded to the situation determination component which loads a situation model ontology based on the data model used for data standardisation in the situation monitoring component. The situation model

provides extended information about data associations and is enriched with provided situation-influencing parameters to enable a situation determination of connected data producers. Instances of the situation model are used for the mapping of standardised monitored data. Based on these instantiated situational models, the component is able to determine and provide the situation. The support of reasoning rules executed by reasoning engines is even able to enhance the situation determination by calculating hidden situation influencing parameters to define situations beyond the situational model. The determined situations are provided to users as the other modules in the SAFIRE solution.

2.1 INNOVATION

The SAFIRE Situation Determination component builds on the State-of-the-Art (SotA) elaborated in EU-FP7 ProSEco project which addressed tools for building context sensitive AmI based Production Engineering Systems, and FP7-NMP Self-Learning project which addressed context sensitive embedded services in the manufacturing industry.

The determination module in SAFIRE goes beyond the SotA (as described in 1.4.2.3 in the DoA) and combines information sources, products, production processes and situations, as common models for reconfiguration & optimisation and provides enhanced reasoning features for situation determination. Another innovation within SAFIRE is the loosely coupled integration in a data analytics solution, in which the situation determination module cooperates with the SAFIRE predictive analytics platform and uses data analytics results for an enhanced situation determination.

3. SITUATION DETERMINATION MODULE SPECIFICATIONS

The Situation Determination Services (SD) includes three components, namely the situation model, the situation monitoring component and the situation determination component. The requirements and the respective specifications for each component are described in the following chapters.

3.1 SITUATION MODEL SPECIFICATION

This chapter introduces the requirements for the Situation Model (SAFIRE ontology) which is used from the Situation Determination Services to identify the situation under which the SAFIRE solution is being used. The Situation Model is developed as an ontology (in the more technical approach towards a semantic model) showing the relations between all the concepts that are relevant for the configuration and operation of the SAFIRE solution. Ontologies provide a representation flexible enough to support common modelling of situation information in a structured way, as well as domain specific extension to the model, thus it's chosen for representation purposes. Especially with the advances of the semantic web technologies, this approach ensures that the SAFIRE solution uses the new technologies to provide a flexible solution for more innovation in manufacturing.

Figure 3: Generic Situation Model structure in a "is-a" hierarchy (excerpt)

The purpose of the SAFIRE ontology is to define a fundamental data model for situation determination. This data model describes the parameters of the SAFIRE

environment which affects its operation, and are related to the products/machines, processes or users that interact with it. However, the identification and representation of all parameters that affect (or might affect) the operation of SAFIRE requires deep analysis and available resources, which will increase the time and cost of the modelling process. Therefore, for the final prototype of the SAFIRE solution the situation model was specified to model specific environment parameters selected to serve the purpose of SAFIRE and in particular, the targeted business case demonstrators.

3.1.1 Requirements

In order to model the situation of SAFIRE it is important to define the limits for the situational awareness of SAFIRE. Since the SAFIRE solution will be used for the optimisation and reconfiguration of products, machines and processes of factories, the situation model should define which information related to products, machines and processes will need to be observed and adjusted. The following table describes the requirements that the situation model should cover. The requirements were extracted based on the generic SAFIRE solution requirements, as well as on those of the respective business cases.

Table 1: Situation Model – Use Case Requirements

Requirement No.	Description	Relevant BC
U68	Shall be able to model situation under which a set of machines is operating	OAS
U69	Shall be able to model situation under which a production process is operating	OAS / ELX37
U76	Shall be able to model situation under which proNTo is operating	OAS

As seen from the aforementioned requirements, the situation model should be able to cover the particularities of the different business cases. At the same time, since the SAFIRE solution is intended to be generic and flexible, easily adjustable to the particular purposes of use in different factories, the situation model should also serve towards this purpose. Furthermore, as the purpose of ontologies suggests, the SAFIRE ontology should be developed in such a way as to be understandable by non-experts who would like to adjust the operation of the SAFIRE solution to the needs of their factory for situational awareness.

The following table presents additional requirements derived from a more thorough examination of the purpose of the SAFIRE situation model, and will allow the solution to be flexible enough for easy adaptation to other systems.

Table 2: Situation Model – Additional Requirements

Requirement No.	Description	Relevant BC
REQ-SD-1	Shall be easy to maintain / extend	ALL
REQ-SD-2	Shall be generic to cover the needs of more business cases than those for which it was created	ALL
REQ-SD-3	Shall enhance information / knowledge representation and sharing	ALL

3.1.1.1 Functional Specification

Protégé is used as a basis for the ontology modelling tool as it offers all functionalities needed to meet the requirements.

The main functions of the ontology modelling tool are:

- Create situation models (Table 3),
- Select and import situation models (Table 4), and
- Export situation model (Table 5).

Table 3: Create situation models.

Create Situation Models	
Classification	Function of the ontology modelling tool
Definition	This function will allow the selection and import of existing situation models.
Users involved	System administrator
Trigger/Event	The administrator wants to create a new or continue working on an existent situation model
Pre-Condition	The company and the partners which will use the SAFIRE solution are defined, and some situation models are already defined
Interaction/Users involved	The administrator can select and import existing situation models
Post-Condition	The situation model is imported and updated as needed
Data/Interface/Export	The situation model in OWL format

Table 4: Select and import situation models.

Select and import Situation Models	
Classification	Function of the ontology modelling tool
Definition	This function will allow the selection and import of existing situation models.
Trigger/Event	The system administrator wants to start or continue working on an existent situation model
Pre-Condition	The company and the partners which will use the SAFIRE solution are defined and there are already some situation models defined
Post-Condition	The situation model is imported and updated as needed
Data/Interface/Export	The situation model in OWL format

Table 5: Export situation model.

Export Situation Model	
Classification	Function of the ontology modelling tool
Definition	Possibility to export a situation model
Pre-Condition	The situation model is created
Data/Interface/Export	The situation model in OWL format

3.1.2 Approach to Situation Modelling

To implement situational awareness, the situation model has to be defined and modelled. We shall use an OWL-based ontology as situation model to represent the extracted information, as explicit machine interpretable knowledge. These ontologies serve as a base for situation determination.

As situation models may vary from company to company, this means that in the setup phase of the SAFIRE environment, the situation model(s) for the specific company and their partners have to be defined and modelled.

For each application scenario, one has to define which concepts are relevant for the description of the situations (context). For example, for a company x it may be important where the user is - in the company or at the customer site, as he may need different information depending on his location.

The project will develop situation models for the three business cases that may serve as a basis for the definition of situation models for other companies in the machine/equipment sector.

The process for defining the situation model(s) for a company when it starts to use the SAFIRE solution, is depicted in Figure 4, and is described below:

- Start from the existing situation model(s) – ontologies.
- Identify which concepts in the existing models (ontologies) are relevant to describe situations by the specific company and their partners.
- Identify which additional concepts may be needed to describe the situations.
- Analyse if there are available sufficient "sensors" to automatically extract the state of each concept, i.e. if there are row information available to extract the current situation, without asking the user to provide data (which in most cases would not be an effective solution).
- If there are not sufficient raw data for automatic situation determination, then one should analyse if it pays back to introduce additional "sensors", or not.

The process is iterative, i.e. based on analysis of the 'needs' of each tool, the initial model can be updated and the process is being repeated.

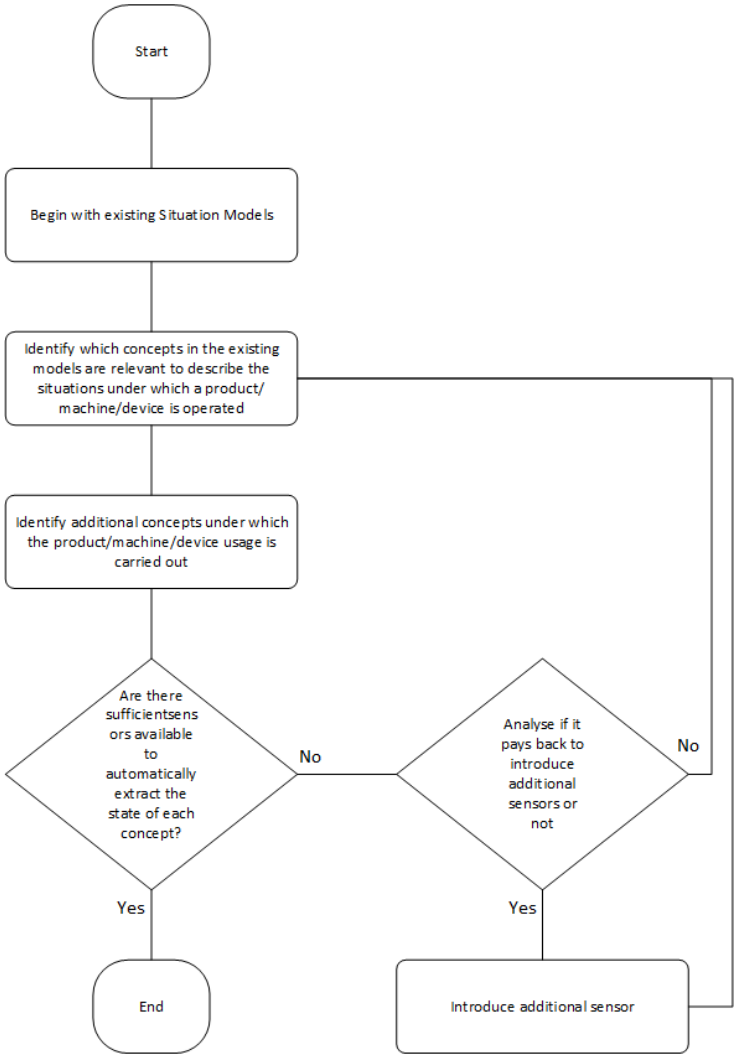

Figure 4: Situation modelling for a specific company in the set-up phase.

3.1.3 Generic Situation Model

The situation model defines a fundamental data model for situation determination. A generic situation model will be created to cover the needs for situational awareness of the SAFIRE solution, and BC-specific situation models will be used to provide additional details on the situation of products, machines and processes, aiming to fulfil specific business case particularities.

3.1.3.1 Model Entities

The following tables describe in detail the categories of situation information (entities).

Name	Activity
Description	Activity is an concept that defines an action performed by someone (Actor or Generic Device/Software) in order to achieve a specific goal.
Has Subclass	Configuration, Maintenance, Optimisation, Prediction
Subclass Of	Thing

Name	Configuration
Description	Configuration is the Activity of adjusting the (operation) parameters of a system, to specific values.
Has Subclass	Reconfiguration
Subclass Of	Activity

Name	Reconfiguration
Description	Reconfiguration is the Activity aiming to change the operation of a system, after its initial operation, to work on a different way towards its purposes or goals.
Has Subclass	-
Subclass Of	Configuration

Name	Maintenance
Description	Maintenance is the Activity of preserving the operation of a system.
Has Subclass	-

Subclass Of	Activity

Name	**Optimisation**
Description	Optimisation is an Activity of improving the operation of a system, by adjusting its (operation) parameters to more effective values.
Has Subclass	-
Subclass Of	Activity

Name	**Prediction**
Description	Prediction is an Activity of foreseen attitudes in the operation of a system (machine, product or process) in order to (re)act accordingly in its future operation.
Has Subclass	-
Subclass Of	Activity

Name	**Actor**
Description	Actor is an entity that describes the humans involved in the situations observed by the SAFIRE solution.
Has Subclass	End User, Industry Expert, Safire Expert
Subclass Of	Thing

Name	**End User**
Description	End User is an Actor that uses the product or machine which is observed by the SAFIRE solution.
Has Subclass	-
Subclass Of	Actor

Name	**Industry Expert**
Description	Industry Expert is an Actor that provides the business expertise to the SAFIRE solution or is involved in the operation of the factory (process/machine) which is observed by the SAFIRE solution.
Has Subclass	Business Expert, Factory Operator, Machine Supplier

Subclass Of	Actor

Name	**Business Expert**
Description	Business Expert is an Industry Expert that has knowledge on the specific business needs and goals, and could give information on the target group of the company and their needs.
Has Subclass	-
Subclass Of	Industry Expert

Name	**Factory Operator**
Description	Factory Operator is an Industry Expert, reflecting to the employee of the respective industrial unit, which is responsible for operating the SW/HW systems.
Has Subclass	-
Subclass Of	Industry Expert

Name	**Machine Supplier**
Description	Machine Expert is an Industry Expert, reflecting to a supplier of the SW/HW infrastructure of the factory.
Has Subclass	-
Subclass Of	Industry Expert

Name	**Safire Expert**
Description	SAFIRE Expert is an Actor that provides the scientific expertise to the SAFIRE solution or is involved in its design/development.
Has Subclass	-
Subclass Of	Actor

Name	**Information**
Description	Information is an concept that defines any received, stored or produced data which describe a situation or part of it.
Has Subclass	Metric
Subclass Of	Thing

Name	Metric
Description	Metric is a measurement to describe a specific condition.
Has Subclass	Control Metric, Key Objective Metric, Observable Metric
Subclass Of	**Information**

Name	Control Metric
Description	Control Metric quantifies the features of a production process which can be changed or tuned during operation.
Has Subclass	-
Subclass Of	**Information**

Name	Key Objective Metric
Description	Key Objective Metric quantifies what feature of a production process should be optimised. .
Has Subclass	-
Subclass Of	**Information**

Name	Observable Metric
Description	Observable Metric quantifies the features of a production process that can be observed during operation.
Has Subclass	-
Subclass Of	**Information**

Name	Product
Description	Product is a Resource that describes the (material) good that is being manufactured as an outcome of the factory. In SAFIRE the product can be individual product or machine.
Has Subclass	Product Part
Subclass Of	Thing

Name	Product Part
Description	Product Part is a piece of a Product (main product), which can be itself a complete/independent product, and its existence affects the operation or functionality of the main product.
Has Subclass	-
Subclass Of	Product

Name	Production Process
Description	Production Process is a series of actions to create a material object (product or part).
Has Subclass	Subprocess
Subclass Of	Thing

Name	Subprocess
Description	Subprocess is a process of limited duration or scope, or which includes less actions than a process.
Has Subclass	-
Subclass Of	Process

Name	Resource
Description	Resource is a Situation model concept that describes any material (e.g. product or person) or entity (e.g. information or process) that is part of the observable situation or its ambient environment, and it could influence its nature, performance or attitude.
Has Subclass	Actor, Generic Device, Generic Software, Product
Subclass Of	Thing

Name	Generic Device
Description	Generic Device is a hardware component within a product/machine, for data acquisition/processing/sharing and communication.
Has Subclass	Network Device, Processing Device, Sensorial Device

Subclass Of	Resource

Name	**Network Device**
Description	Network Device is a Generic Device which provides interconnection between other generic devices or gives access to remote resources.
Has Subclass	-
Subclass Of	Generic Device

Name	**Processing Device**
Description	Processing Device is a Generic Device used to perform calculations for a process.
Has Subclass	Processing Unit
Subclass Of	Generic Device

Name	**Processing Unit**
Description	Processing Unit is a Processing Device of a specific and limited purpose which could operate independently, but in the scope of the SAFIRE solution is considered part of a set (respective Processing Devise).
Has Subclass	-
Subclass Of	Processing Device

Name	**Sensorial Device**
Description	Sensorial Device is a Generic Device that allows for data collection of the environment of the observed situation, and acts as an interface between users, products, machines or processes, and their environment.
Has Subclass	Pressure Sensor, Temperature Sensor, Time Sensor
Subclass Of	Generic Device

Name	**Pressure Sensor**
Description	Pressure Sensor is a Sensorial Device that collects pressure information from the area of operation.
Has Subclass	-
Subclass Of	Sensorial Device

Name	Temperature Sensor
Description	Temperature Sensor is a Sensorial Device that collects information related to the area of operation.
Has Subclass	-
Subclass Of	Sensorial Device

Name	Time Sensor
Description	Time Sensor is a Sensorial Device that collects time information from the area of operation.
Has Subclass	-
Subclass Of	Sensorial Device

Name	Generic Software
Description	Generic Software is a Resource that refers to the non-material aspects necessary for the operation of the Generic Device. It includes also the control software (firmware) integrated on products or machines.
Has Subclass	Enterprise Resource Planning Software
Subclass Of	Resource

Name	Enterprise Resource Planning Software
Description	Enterprise Resource Planning Software is a Generic Software that manages, stores and processes the business data of a company.
Has Subclass	-
Subclass Of	Generic Software

The entities included in the company-specific and in the BC-specific situation models can found described in detailed in the documents for the early and full prototype.

3.1.3.2 Model Relations

The entities have been chosen to fulfil initially the basic relations in form of a hierarchy (see Table 6). Additional relations are being described in the following table, showing also examples of the domains and ranges used.

Table 6: Situation Model Relations (excerpt)

Domain (example)	Relation	Range (example)
	hasDatumUnit	
Production Process	*has*	Metrics
Reconfiguration	*isCausedBy*	GenericDatum
Factory Operator	*isOperatorOf*	Generic Device
Product Part	*isPartOf*	Product
Maintenance	*isPerformedBy*	Factory Operator
Generic Datum	*isProcessedBy*	Processing Device
Product	*isProducedBy*	Production Process
Actor	*isUserOf*	Product

3.2 SITUATION MONITORING SPECIFICATION

The objective of the situational monitoring component is to receive raw data and provide aggregated situational data. It is a generic solution for the monitoring data sources and customisable for different communication protocols and data structures. It enables also data pre-processing or data aggregation. Figure 5 shows the conceptual architecture of the Situation Monitoring module.

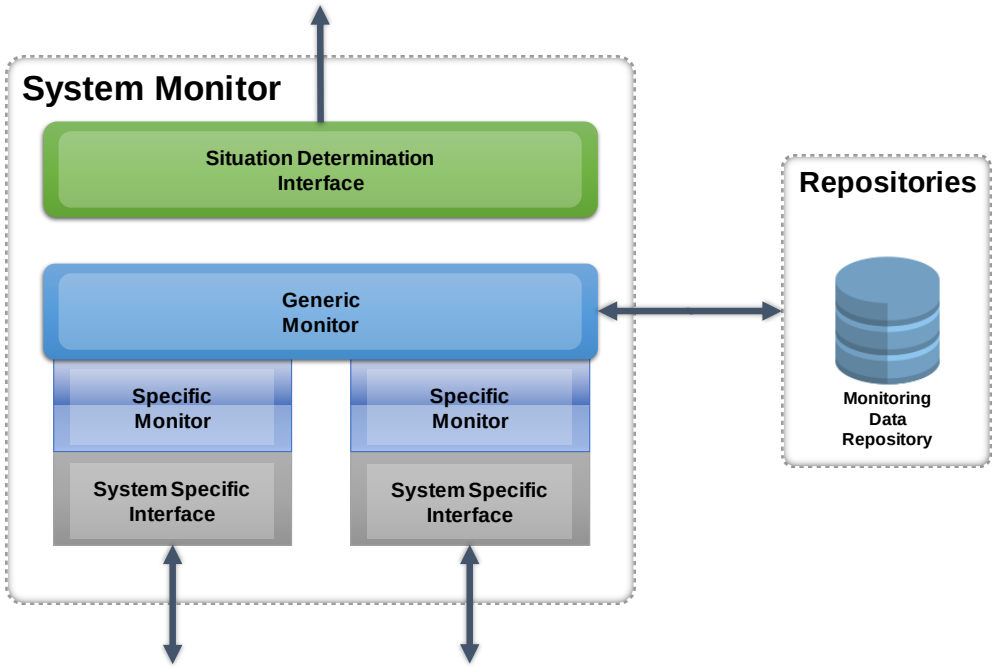

Figure 5: Conceptual Situation Monitoring Architecture

3.2.1 Requirements

In this section the requirements for the situational monitoring component are defined. Table 7 shows again the requirements identified for the SAFIRE project based on the envisaged Business Cases elaborated in T1.1 (Business Case Analysis) and documented in D1.1 (Application Scenarios Requirements Analysis) for the situational monitoring component.

Table 7: Use Case Requirements for Situational Monitoring

Requirement No.	Description
U54	Able to change existing or adding new monitoring sources with minimal effort
U55	Able to support collection of environmental data to identify current situation
U56	Able to support collection of operators' behaviours to identify current situation
U57	Able to monitor machine current status data to identify situation
U58	Able to monitor machine health status to identify current situation
U59	Able to monitor overall equipment effectiveness (OEE) to identify current situation
U60	Able to monitor production status to identify current situation
U61	Able to support collection of data from proNTo behaviours to identify current situation
U62	Able to monitor Hob Temperature status to identify current situation
U63	Able to monitor Pot Boiling status to identify current situation

Based on the requirements defined in Table 7 technical requirement are derived related to:

- User Classes and Characteristics
- Functional requirements
- Non-functional requirements

These derived requirements are described in the following.

3.2.1.1 Functional requirements

Based on D.1.1 Application Scenarios Analysis functional and non-functional requirements for the situational monitoring module were derived. The relevant requirements for this module are shown in Table 8.

Table 8: Functional requirements for situational monitoring derived from Table 7

REQ_ID	Requirement	Derived from requirement(s)
r_fun_mon_01	Situational monitoring shall be easily customi-zable for different kinds of data sources	U54, U55, U56, U57, U58, U59, U60, U61, U62, U63
r_fun_mon_02	Situational monitoring shall support monitoring of data sources.	U54, U55, U56, U57, U58, U59, U60, U61, U62, U63
r_fun_mon_03	Situational monitoring shall support pre-processing of monitored data	U55, U56
r_fun_mon_04	Situational monitoring shall support storing of monitored data	U55, U56

3.2.1.2 Non-functional requirements

The identified general non-functional requirements are:

Table 9: No-Functional requirements for situational monitoring derived from Table 7

REQ_ID	Requirement	Derived from requirement(s)
r_non-fun_mon_01	Situational monitoring shall be a generic, customizable component.	U54

3.2.2 Functional Specification

The objective of this component is to receive raw sensor data and provide aggregated situational data. To achieve this, the situational monitoring component allows monitoring of legacy systems in enterprises and products via different interfaces from the Data Access Layer. It is therefore able to standardise and correlate the data from distinct systems (e.g. map actions from file systems and Web-Services) which later serves as a basis for identification and extraction of situations.

The main component of the Situational Monitoring is the modular monitoring process, used for all monitoring features with an extendable and configurable standardized process (see Figure 6). It is the process description for all features how to attach and monitor external legacy systems in an enterprise environment, as well as how to process the captured information. The process is three-parted and contains the:

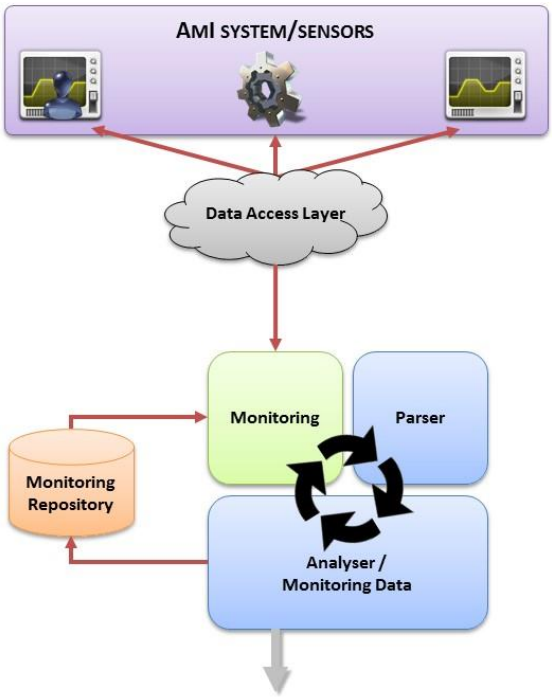

Figure 6: Recurrent Monitoring Process

- *Monitoring Situational system/sensor/behaviours module*, which contains all features to monitor legacy systems and devices in enterprises via the Data Access Layer. The distributed monitoring services also call back to this module with their gathered information. The monitoring features can be extended and configured for different systems and do not need to comply with other modules.
- *Parser module*, which contains content parser for the different possible data captured by the monitoring module. The parser offers access to the diverse data possible interacted and therefore monitored with. It provides the access for the analyser and may parse available environmental properties.
- *Analyser / Monitoring Data builder module*, which correlates the monitored content (and maybe environmental properties) and constructs the standardized Monitoring Data to be stored and handed over to the Situation Monitoring / Determination service or any other service that needs this information.

For each data-source (that is about to be monitored) an index can be specified, that will hold the environmental properties of the resources monitored. Depending on the actual resources to be parsed and analysed (like files or XML data from Web-Services etc.) several resource specific plugins can used in for the generic situation monitoring framework. Each of these allows inclusion of parsers and analysers which are specified for additional external and legacy systems.

The objective of the situation monitoring component is to receive raw sensor data and provide aggregated situational data. To achieve this, the component allows monitoring of different kinds of data sources (e.g. legacy systems in enterprises via different interfaces from the Data Access Layer). It is therefore able to standardize and correlate

the data from distinct systems (e.g. map actions from file systems and Web-Services) which later serves as a basis for identification and extraction of situations.

The main feature of the Situational Monitoring component is the modular monitoring process, used for all monitoring modules with an extendable and configurable standardized process (see Figure 6). It is the process description for all services how to attach and monitor external legacy systems in an enterprise environment, as well as how to process the captured information. The process is three-parted and is supported by external interfaces as described in the following sections.

3.2.2.1 Monitoring

This situational monitoring module performs a permanent loop of monitoring for changes or creation of resources, which indicated a change in content.

Table 10: Situational Monitoring system/sensors

Monitoring of situational-aware system/sensor	
Description	Monitoring of external systems, checking for usage, handing content information to the further processes.
Trigger Event	The process is a monitoring service which performs a permanent loop that checks for changes (or creation) of resources indicated by a change in the content.
Parameters	The parameter to the analyser is the parsed content and Meta-Data from the Parser.
Process/Stages	The process performed is as follows: monitoring is started with the situational configuration parame-ters it starts monitoring the observed system /sensor the information about its origin and all gathered data are given to the parser processes
Input/Output Data/Interfaces	Situational Monitoring provides (pre-processed) situational data coming from monitored data sources.

3.2.2.2 Parser

The backend process of monitoring, parsing, analysing and monitoring data creation is the cohesive process to store a monitored resource in the monitoring repository, ready to be compared with other similar resources and be prepared and conditioned as Monitoring Data for the situation determination.

Description

The Parsing process is connected to the monitoring input and the backend process of analysing and aligning the information. It is important to state that the parsing process itself does not analyse the data but offer access to it and allows as a connected upstream of the analysing to parse and therefore sort the information in a specific manner. For

example, may plain text files be accessible by the whole system and therefore the Analyser without the need of a Parser, but the Parser may utilize a schema or other utilities to arrange the information in Content and Meta-Data in a specific way. This will further be the foundation for the Analyser process which expects only this information and analyses on that without specific rearrangement of the resource. Another example may be the access of an external Web-Service where the Parsers controls and manages calling the Web-Service and prepares the data received from the service in order to hand it over to the Analyser.

Table 11: Parsing monitoring data

Parsing of monitoring data	
Description	The parser offers access to data by sorting the information in a specific manner. Parsing data coming from the monitoring system process. It selects based on the content and resource type the specific parser from a set of type specific computations and parses the Content and the special type Meta-Data and hands the data to the Analyser.
Trigger Event	Parsing process is triggered by the monitoring process. If the monitoring does not identify a change in the monitored resource, it is not triggered.
Parameters	The parameter handed over to the parser is the data monitored, containing environmental properties like the location of the resource to be parsed.
Process/Stages	The process of the Parser is as follows: • the parser gets the monitored data • the data is checked for their type • the appropriate specific parser according to the configuration is selected • the parser prepares the monitored data and possible meta-data • the data is handed to the appropriate Analyser
Input/Output Data/Interfaces	The parser hands a set of content and Meta-Data to the according Analyser.

3.2.2.3 Analyser

The module for transferring the "raw" monitoring data into the standardized monitoring data is the Analyser. This function builds the monitored and parsed data into an RDF-based XMP representation of the recently monitored device's status.

The correlated Monitoring Data representing the last monitored status is needed to be stored in the monitoring repository as comparable source for other components.

As all data of all monitoring service modules will be based on the POJO-based principles, the serialization of the information provided from the Parser will be easily

converted through self-describing classes and functions. The analyser therefore invokes the handed over data and comprehends all single information in a single representation, which will be stored in the monitoring repository. The constructed data is based on a RDF-based XMP description.

Table 12: Analysing monitoring data

Analysis of monitoring data	
Description	Constructing a RDF-based XMP monitoring data based on the data handed over.
Trigger Event	The Analyser is invoked by the Parser.
Parameters	The parameter to the Analyser is the parsed content and Meta-Data from the Parser
Process/Stages	The process of the Analyser is as follows: • receives instances of parsed content and meta data • constructs the Standardised Monitoring Data based on analysed data
Input/Output Data/Interfaces	The outcome of this function is a string-representation of a RDF-based XMP file, comprehending one standardized Monitoring Data, which is stored in the Monitoring Repository.

3.2.2.4 External Interfaces

To acquire the information from legacy systems docking onto the solution, several external interfaces are required, which can be refined and extended furthermore in their special categories. On the one side the SAFIRE process needs all possible data from different systems to parse and analyse and process for data aggregation. Therefore, different communicational interfaces to legacy systems and Situational systems/ sensors must be part of the integration of the solutions process onto other systems.

Table 13: Inputs/outputs for the Situational aware based Monitoring

Service for Situational aware based Monitoring			
Type	Name	Format	Connection to ontology entity
Pre-conditions	Monitored situational systems/sensors are identified and selected service for Situational Monitoring is edited and configured for the SAFIRE solution		Situational Configuration object
Input	All situational system / sensor data available	Raw sensor data	none

Service for Situational aware based Monitoring			
	for the scope of the modules in question (low level data)		
Output	Standardised and aggregated sensory data (high level data)	RDF	Updated situational configuration object that contains information on how and where to access the monitored data in the Situational Monitoring Repository.
Post conditions	Systems/sensors data is monitored continuously. After each monitoring cycle the data is stored in the Monitoring Repository.		

3.2.3 Security issues for Monitoring of User-centred Information

The monitoring of user interactions requires a focussed and controlled insurance of Information Security and Privacy (e.g. based on ISO2700x). As the monitoring services rely on the same standards as the calling sub-system they are distributed from, the monitoring services secures data based on this first layer. For example, the monitoring services are only able to access information available for the software system such as freely available websites, but no restricted pages if no credentials are available.

3.2.3.1 Information Security & Privacy

In the presented methodology of focussing on the user's interactions, the process of monitoring especially this information (users, actions and the content) is critical. The Methodology of Monitoring Knowledge-based Activities allows a high degree of configuration to prevent the gathering of critical data of persons and systems. By that any business specific adaptation can be configured and adjusted to fit the most comprehensive data integrity requirements and ensures the safety of any personal information.

The gathered data depends on the used actions onto interfaces and/or legacy systems. As the service-oriented approach gives a highly configurable and divisible monitoring action container, each action is separated from the other to gather one kind of information. The outcome is that by simply composing the actions in different ways critical data is not transported and therefore secured. The combination of a highly diverse Service-oriented Architecture that supports differentiated orchestration of services in different environments with the overall architectural security of services

and/or data by utilizing an according security framework means the protection and privacy of data can be assured.

This option of configurable services (monitoring process chain of services) offers the best possibility to cover all cases of business/legitimate information security and monitoring system adaptation likewise. All security options are to ensure the Information Security and Privacy according to ISO 2700x.

3.2.4 Technical Specification

The following section describes the specific technical specification related to the component for situation monitoring.

3.2.4.1 Service-oriented Monitoring Architecture

To support the premise of integrating and monitoring existing systems in a collaborative working environment to unobtrusively enhance the existing information with the user's interaction knowledge, it is required to easily integrate and compose the needed monitoring services applicable to existing systems. Furthermore, the services must comply with a common networked enterprise and therefore should base on a loosely coupled but orchestration capable architecture.

A service-oriented approach has been chosen to comply with all these requirements of monitoring in a networked enterprise, as well as the need of a configurable and adaptable service composition. Hence the overall structure to comprehend the described solution is a Service-oriented Monitoring Architecture (SoMA) (see Figure 7). The modularity of the SoMA allows easy adaptation to other external legacy systems through the Access Layer and the Generic Monitoring Services that can be extended by additional services, according to specific business cases. The same applies to the Parser and Analyser which allow to be extended accordingly and in future works broadening the action matrix. All services rely on interchangeable interfaces in-between to comply with the architecture and to apprehend the standardized Monitoring Data generation. Because of the loosely coupled structure of the services inside the layered architecture, all parts rely on an external XML configuration orchestrating the Monitoring, Parser and Analyser process and configure the action and monitored data matrix. This offers a high compatibility and adaptability to different collaborative solutions and business cases.

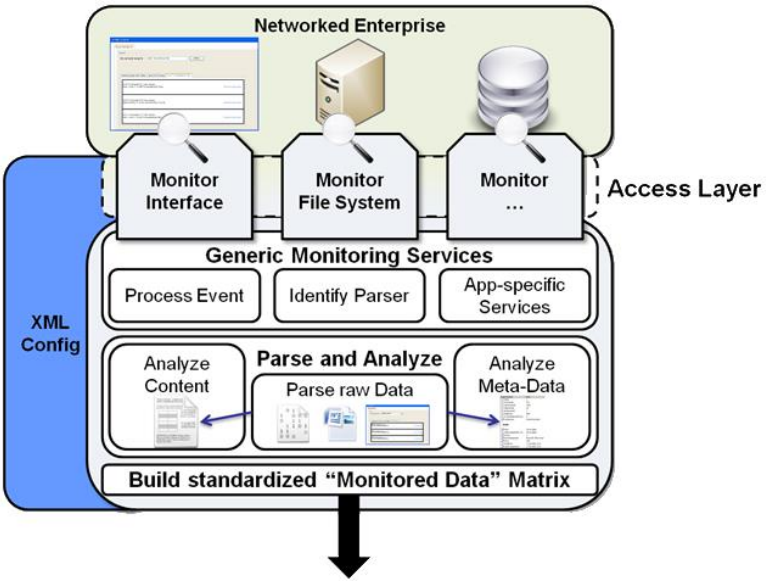

Figure 7: Service-oriented Monitoring Architecture (SoMA)

From the whole implemented SoMA, the most important focus lies on the Monitoring Services and according Analysing Services developed. The Monitoring Services comprehend general solutions to monitor basic systems and parts of these like e.g. the DOM of a webpage or classic file systems as both are most often part of or a mediator foundation for collaborative environments. These basic services can then be further extended hence adapted to Application-specific services, which monitor either special system not accessible by common means or specify a generic service by e.g. templates referencing a specific structure. An example could be a generic website, specified through a template defining what content can be found in which location. These (specific) services are the foundation for the monitoring process and further operated by the Parser and Analyser. Especially the Analyser serves as an element directly correlated to the monitored data. The Monitoring Services provide the access/interface to the systems and react to actions whereas the Analyser gives "sense" to the monitored data in the overall action. It analyses the action itself and puts it into context to the content it refers to (e.g. a whole webpage or a file). Furthermore, it enriches the information with specific Metadata which can be extracted from the monitored source. This process is applicable to several other systems in a collaborative working environment like e.g. a file system. The change of a file can be contextually interpreted by the location, name, date, time and several other configurable properties, besides the information about the interacting user.

This modular and configurable combination of services which can be easily orchestrated to appropriate compositions is the main foundation of the SoMA and allows a high reusability between different application scenarios. Besides the modular composition to comply to different scenarios especially the Monitoring Services underlie a strict configuration to comprise organisation as well as national laws for Information Security and Privacy. By monitoring the user's interaction, trying to elaborate his/her situation, a set of critical knowledge is gathered. Therefore, the configuration is accessible to every organisation and user to individually depict what information may be gathered.

3.2.4.2 Monitoring of Data Sources

To (dynamically) capture changes of Web Services via triggered events (or by polling) there are several libraries used to access the components through the different transport protocols and data bindings. The libraries that offer this accessibility and additional support to deliver data (and hereby results) in a fast and asynchronous way are for example Apache CXF[1]. The monitoring of file systems is accomplished by the build-in Java functionality which supports all needed actions if the foundation system relies on the system attributes.

To parse and analyse the several different types out of the file systems additional libraries are used that offer access and data encryption. These are extendable depending on the change of the type specification.

3.2.4.3 Situation Monitoring Service

The main classes of the System Monitor are:

- *IAmIMonitoringService*: The interface defining the System Monitor web-service and its public classes. Via Java annotations the type of the web-service is defined (i.e. SOAP).
- *AmIMonitoringService*: Main class of the System Monitor. Each configured Monitor is started in a separate thread to allow parallel continuous monitoring of configured data sources. Monitored information are stored in the monitoring data repository that is made accessible via this service for each monitor.
- *IAmIMonitoringDataRepositoryService*: Interface defining the monitoring repository service. A reference to this service is hold in the System Monitor to allow access to the concrete implementation of the monitoring repository.
- *MonitoringConfiguration*: Holds the configuration of the system monitor during run-time. This configuration object reads the XML configuration file that configures the system monitor.
- *ThreadedMonitor*: The ThreadedMonitor object is responsible for starting and stopping all configured monitoring plugins (Monitors). During run-time, it holds a list of all defined monitors and manages their states.

[1] http://cxf.apache.org/

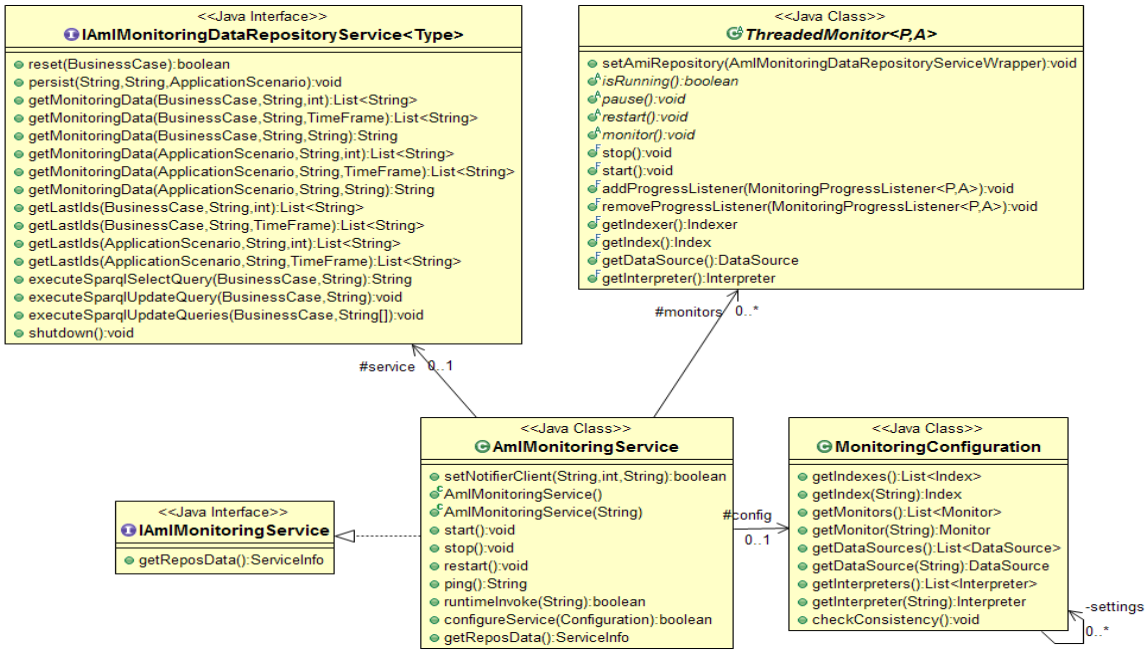

Figure 8: Class Diagram – Situation Monitoring Service

3.2.4.4 Monitoring Repository Service

The main classes of the System Monitor are:

- *IAmIMonitoringDataRepositoryService*: Interface defining the monitoring repository service. Via Java annotations the type of the web-service is defined (i.e. SOAP).
- *AmIMonitoringDataRepositoryService*: Main class of the Monitoring Repository service. Each configured Monitor is started in a separate thread to allow parallel continuous monitoring of configured data sources. Monitored information are stored in the monitoring data repository that is made accessible via this service for each monitor.
- *PersistenceUnitService*: Abstract class, where functionalities related to persistence are implemented.
- *MonitoringDataRepository*: This class implements all required functionalities to persist and load data to and from the monitoring repository. The data is stored in an SDB repository.

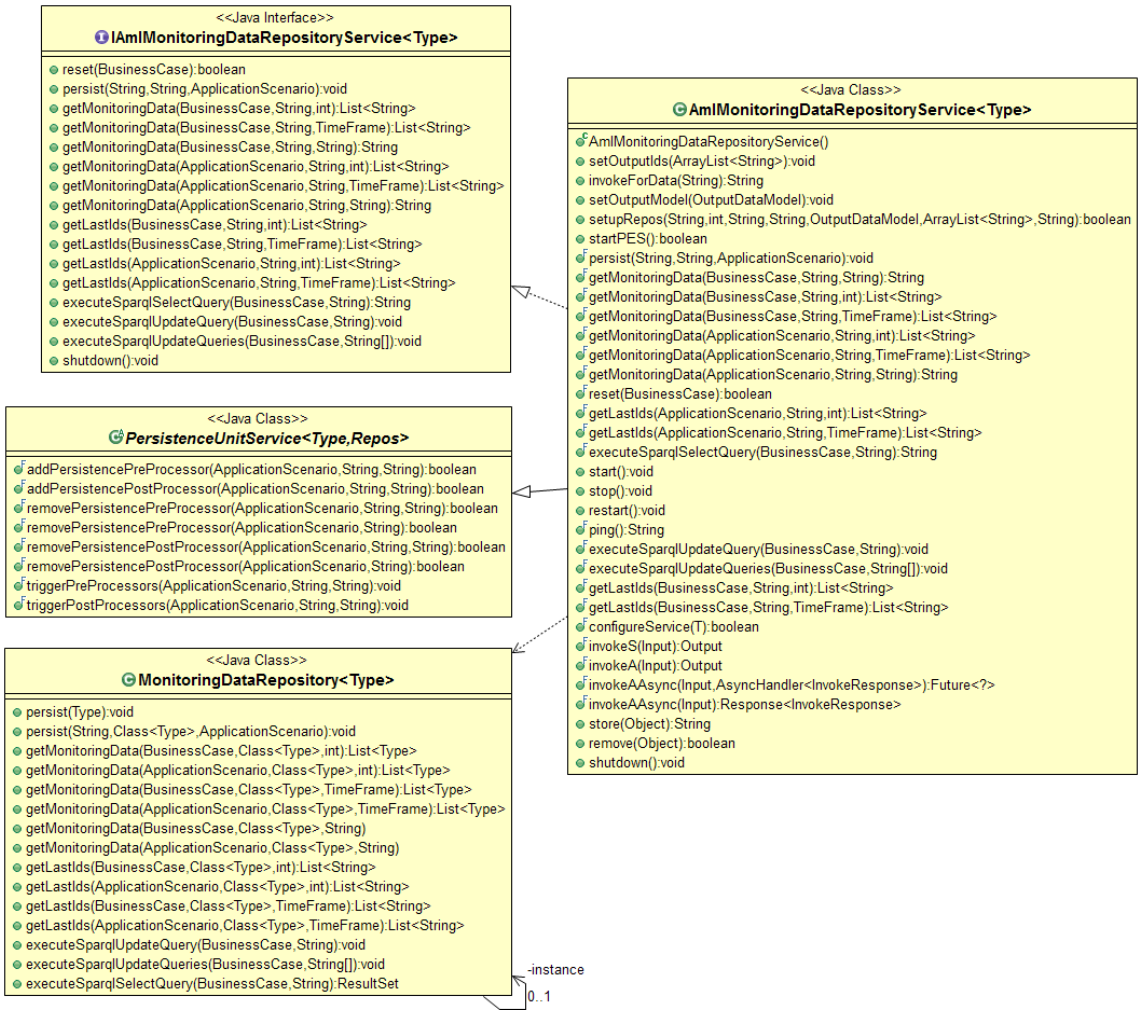

Figure 9: Class Diagram – Monitoring Repository Service

3.2.4.5 Monitors

The System Monitor has several generic monitors that were implemented in the current prototype. Each of the monitors is implemented as a software service. These monitors are:

- a Monitoring Service for SOAP based Web-services,
- a Monitoring Service for legacy File Systems and
- a Monitoring Service for relational databases;

These monitors were implemented to be able to execute the test cases as well as the experiments for the selected validation scenarios:

- *ThreadedMonitor*: The ThreadedMonitor object is responsible for starting and stopping all configured monitoring plugins (Monitors). During run-time, it holds a list of all defined monitors and manages their states.
- *FileSystemMonitor*: This monitor checks files in a specific folder of a filesystem for changes. For example, a production system stores state changes and sensory information in periodic intervals in these files.

- *WebServiceMonitor*: This monitor retrieves data from a (production) system that makes observable data available via a web-service.
- *DatabaseMonitor*: This monitor allows to observe a database for changes in the schemata of the database. This monitor was not used in a validation scenario, but for some unit test cases during the development.

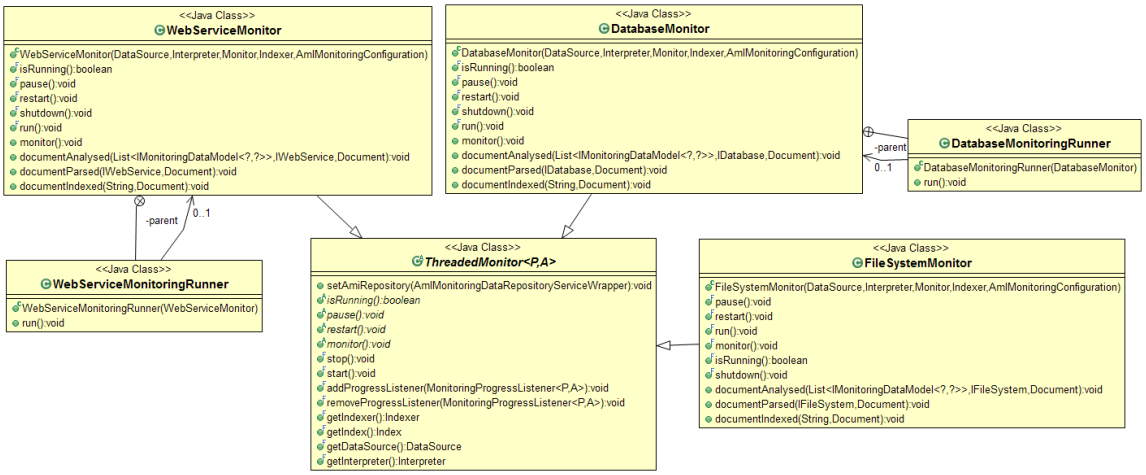

Figure 10: Class Diagram – System Monitors

3.2.4.6 Parser

The main classes for all parsers is the *IndexingParser*. The following briefly describes the implemented parsers:

- *IndexingParser*: This is an abstract monitoring parser. The corresponding analyser for each parser is created during initialisation. The IndexingParser holds a reference to the corresponding analyser (i.e. IndexingAnalyser, which is executed after successful parsing of the data to be monitored.
- *FileParser*: Abstract parser that deals with handling files from a file system. The parsing process itself has to be implemented by each concrete implementation (e.g. VlvParser, TrfParser, CpfParser).
- *DatabaseParser*: Abstract parser that deals with data from a database. The parsing process itself has to be implemented by each concrete implementation (e.g. RGEParser).
- *WebServiceParser*: Abstract parser that deals with handling data from a web service. The parsing process itself has to be implemented by each concrete implementation.

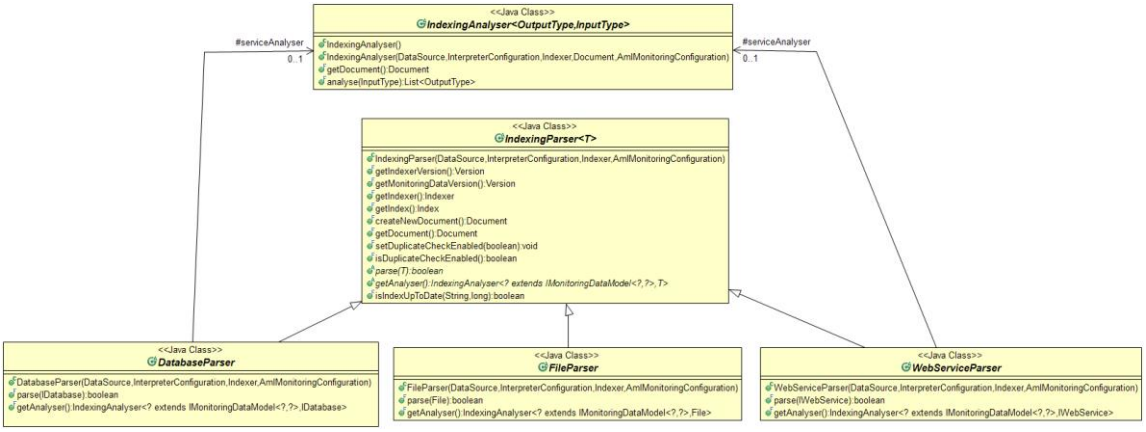

Figure 11: Class Diagram – Parser

3.2.4.7 Analyser

The main classes for all parsers is the IndexingAnalyser. The following briefly describes the implemented analysers:

- *IndexingAnalyser*: This is an abstract monitoring analyser.
- *FileAnalyser*: Abstract analyser that deals with analysing files from a file system. The analysing process itself has to be implemented by each concrete implementation (e.g. ValveAnalyser).
- *DatabaseAnalyser*: Abstract analyser that deals with analysing data from a database. The analysing process itself has to be implemented by each concrete implementation (e.g. PRONTODatabaseAnalyser).
- *WebServiceAnalyser*: Abstract analyser that deals with analysing data from a web service. The analysing process itself has to be implemented by each concrete implementation.

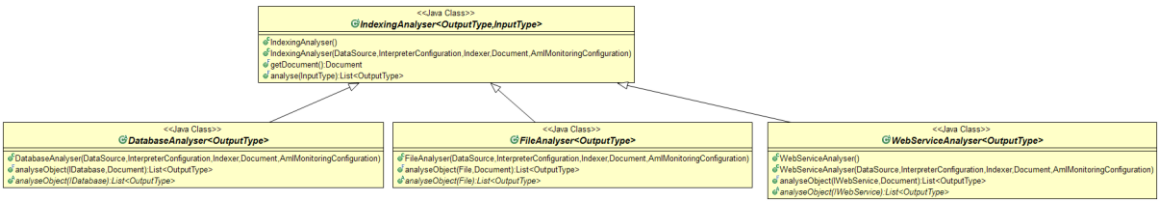

Figure 12: Class Diagram – Analyser

3.2.4.8 Monitoring Data Models

The main classes for all monitoring data models is the MonitoringDataModel. The following briefly describes the monitoring data models:

- *IMonitoringDataModel*: Interface defining the data model for monitored data. The monitoring data model will be created by the analyser and persisted into the monitoring repository.
- *IMonitoringData*: Interface defining which methods have to be implemented for a concrete implementation of monitoring data.

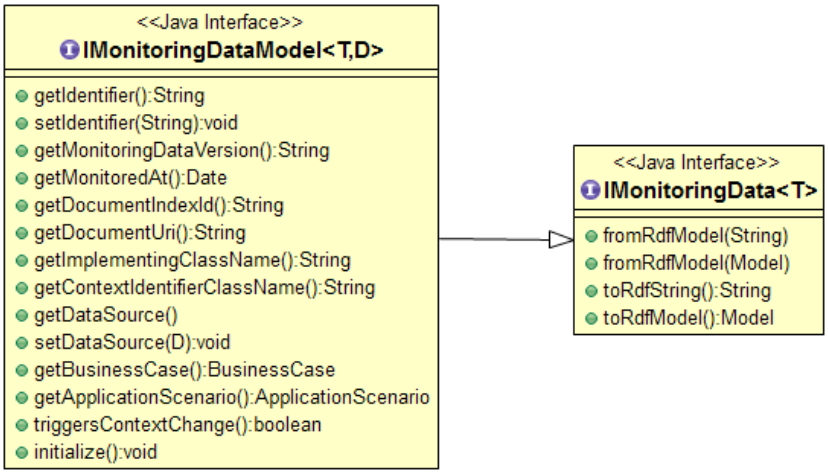

Figure 13: Class Diagram – Monitoring Data Model

3.2.4.9 Kafka Monitoring Interface

The Situation Awareness Services use an interface to the distributed messaging system *Kafka* to retrieve information from the data ingestion module. Since the main communication channel is a *Kafka* node/cluster every service which wants to use this channel has to subscribe to a specific topic in order to get the relevant information. There are following classes related to this interface:

- *KAFKAManager:* The purpose of this class is to manage the consumers of one group (sharing same subscriptions and group id) as a whole. With the initialisation of the class one initial consumer is created (but not started). The class methods allow adding more consumers to the group, starting all the consumers, retrieving the number of consumers. Optional a producer can be added to the class as well, in this case one Manager object can be used for the full data flow control for one node: creating and starting consumers, reading from topic(s) with one or more consumers and sending data to the target topic after internal processing in the node. If the node (Java module) is dealing with different data types, multiple Manager objects might be required.

- *KAFKASender:* sends a serialized message to a specific topic into the cluster

- *KAFKAConsumer:* retrieves serialized messages by subscribing to a specific Kafka topic

- *KafkaLoopConsumer:* specialized consumer class, allows for concurrent consuming of serialized messages from different Kafka topics

As a consequence every service which wants to retrieve information from a specific Kafka topic has to instantiate a *KAFKAConsumer*, resp. a *KafkaLoopConsumer* in order to do so. Find below the class diagram for the interface to *Kafka:*

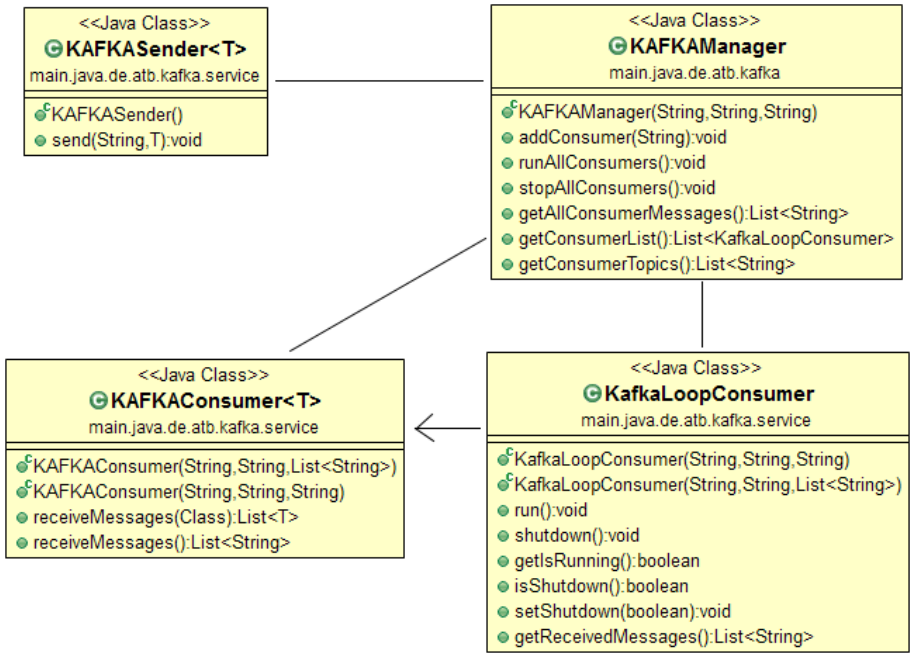

Figure 14: Class Diagram – Interface to Kafka

In particular, in order to fulfil the communication between the Predictive Analytics module and Situation Monitoring, the results from Predictive Analytics is fed back into the Kafka node/cluster such that it can be retrieved by the Situation monitoring via a *KAFKAConsumer* – using a priori specified topic.

3.3 SITUATION DETERMINATION SPECIFICATION

The objective of the situation determination component is to identify the situation of products/machines, production processes or specified systems (e.g. legacy systems as proNTo in the OAS BC) and to provide it for further use within the SAFIRE solution to other modules or external systems. Figure 15 shows the conceptual architecture for the Situation Determination module.

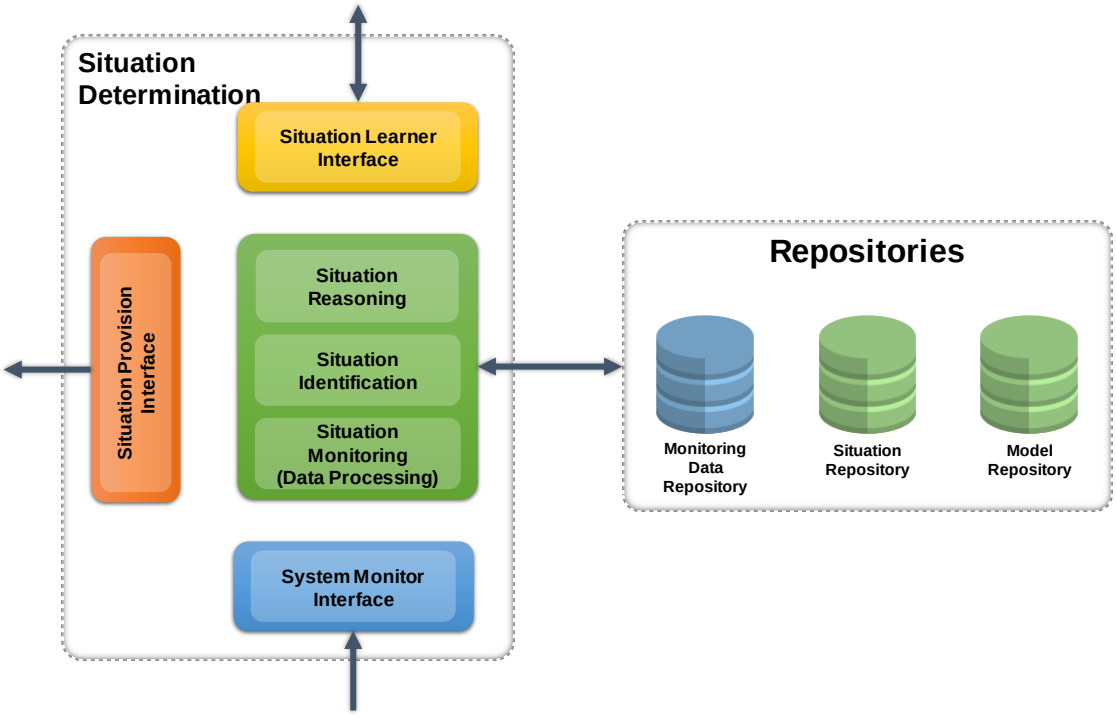

Figure 15: Conceptual Situation Determination Architecture

3.3.1 Requirements

In this section the requirements for the situational determination component are defined. Table 14 shows again the requirements identified for the SAFIRE project based on the envisaged Business Cases elaborated in T1.1 (Business Case Analysis) and documented in D1.1 (Application Scenarios Requirements Analysis) for the situational determination component.

Table 14: Requirements for Situational Determination component elaborated in T1.1

Requirement No.	Description
U64	Able to provide situational information based on raw and monitored data.
U65	Able to extract situational information from monitored machines.
U66	Able to dynamically extract situational information from sensor data.
U67	Able to change existing or add new situations with minimal effort.
U70	Able to extract situational information from sets of related machines.
U71	Able to extract situational information from operator actions.
U72	Able to evaluate situation with respect to capacity, performance, availability (OEE) of monitored machines.

Requirement No.	Description
U73	Able to evaluate situation with respect to capacity, performance, availability (OEE) from sets of related machines.
U77	Able to extract situational information from proNTo and from other systems.

Based on the requirements defined in Table 14 technical requirement are derived related to:

- User Classes and Characteristics
- Functional requirements
- Non-functional requirements

These derived requirements are described in the following.

3.3.1.1 Functional requirements

The analysis done for the development of the SAFIRE solution (presented in D1.1 Application Scenarios Requirements Analysis, chapter 5.5 Situation Analysis) lead to the extraction of the following requirements which show the necessity for situational awareness. Some of these requirements appear also as requirements in the other components of the situation determination module, since all components in the module are complementary, i.e. one creates the situation model and uses the situation determination part to provide the current situation for the data gathered in the situation monitoring part.

Table 15: Functional requirements for situational determination derived from Table 14

REQ_ID	Requirement	Derived from requirement(s)
r_fun_det_01	Situational determination shall be easily customizable for different kinds of monitored standardized data.	U64, U67, U68, U69, U71
r_fun_det_02	Situational determination shall support the determination of situations based on provided standardised monitoring data and data analytics results.	U64-U66, U70-U71-73, U77
r_fun_det_03	Situational determination shall support situation determination based on reasoning rules.	U64-U66, U70-U71-73, U77
r_fun_det_04	Situational determination shall support storing of historical situations	

3.3.1.2 Non-functional requirements

The identified general non-functional requirements are:

Table 16: Non-Functional requirements for situational monitoring derived from Table 14

REQ_ID	Requirement	Derived from requirement(s)
r_non-fun_det_01	Situational determination shall be a generic, customizable component.	U67

3.3.2 Functional Specification

This section provides the specification of the parts composed within the situation determination component. The service uses a situation model for an integrated representation of the observed environment (e.g. products, systems, machines, processes or users).

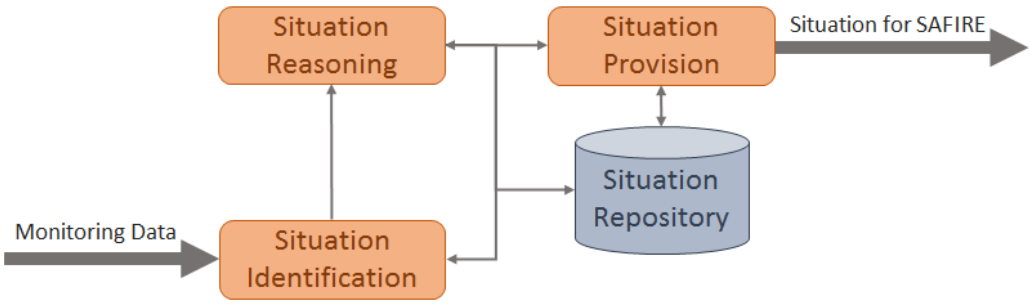

Figure 16: Situation Determination Process

The situation determination component consists of the following services:

- **Situation identification:** determine the current situation of products, machines and production processes using monitored raw data from the situation monitoring component and historic situation information stored in a situation repository, based on the SAFIRE situation model (ontology).
- **Situation reasoning:** uses reasoning rules on the situation provided by the situation identification part, and determines more accurate situations, which cannot be directly identified during the process in the situation identification part
- **Situation provision:** provides detected situations to the other SAFIRE.

As shown in the figure above, the situation determination component identifies situations based on monitoring data, provided by the monitoring module (see section 3.2), enhances it through different types of reasoning techniques (situation reasoning), and provides the refined situations for further exploitation to SAFIRE modules (or external systems).

The situation determination module is based on the services developed within the projects Self-Learning and ProSEco. The module will be extended to address the specific needs of the generic SAFIRE solution, as well as the particularities of the three business cases. The collection of input data will be done by the monitoring module (see Section 3.2), and the situation determination part will analyse structured and unstructured data in order to specify the current situation and identify the parameters of

the product/machine or process environment could affect its performance. The identification of the respective situation is being done based on the situation models which define each time what information is relevant to the observed situation. The identified situation information is being stored in the situation repository as annotation to the content that is used in the observed situation.

The situation determination component is composed by the following parts:

- The situation model (ontology, see Section 3.1), tailored either to cover the generic needs of the SAFIRE solution (generic situation model) or to support the specific needs of the business cases or companies (BC-specific and company-specific situation model). The model is defined as high-level-structured representations of the product/machine, the sub-processes and the resources involved in the production process or the product use situations. All methods included in the situation determination module have as basis for their functionality, this input situation model.
- The situation identification component, which analyses the monitoring data handed over by the monitoring module (see Section 3.2) and extracts a description of a situation.
- The situation reasoning component that reasons on the situation provided by the situation identification module and generates more precise situation, which cannot be directly identified from the situation identification module.
- The situation provisioning component that compares the similarity between the current on-going situations and historical situations in the repository and provides the results to other modules in the situation determinator.

The aforementioned parts are described in more detail in the following sections.

3.3.2.1 Situation Identification

The situation identification component analyses the monitoring data handed over by the monitoring component, and extracts knowledge such as information about the involved products or machines, the users of the observed situations, details on the resources used or consumed, and information on the activities performed and the environment characteristics occur during the current on-going situation

The situation identification is the first function to be executed in the situation determination process. It is triggered by the monitoring services upon receiving the monitoring data or on a certain period according to the customisation made. The data are being analysed and mapped onto the given ontology by the situation identification part in order to recognise as much as possible detailed situational information. The identified dummy situation itself is sent to the situation reasoning function for further processing and refinement (see Figure 17).

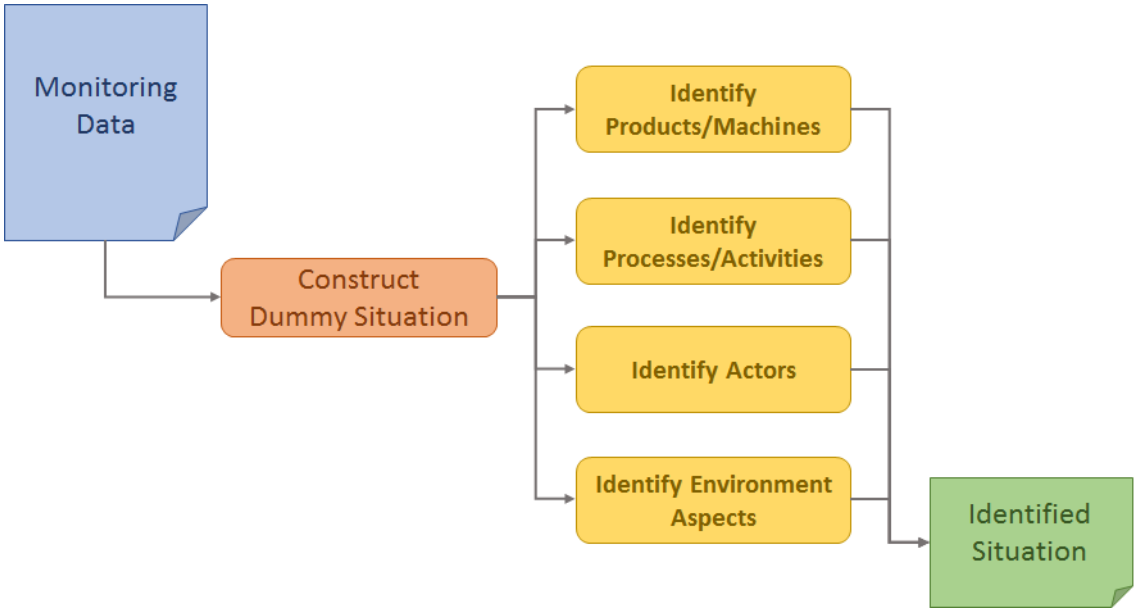

Figure 17: Situation Identification

The situation identification process needs to map the delivered data onto the defined situation model. Figure 17 shows the conceptual situation identification process, and the table (Table 17) later describes its specification in detail.

Table 17: Situation Identification

Situation Identification	
Description	Analyses the monitoring data provided by the monitoring module to identify situations.
Trigger Event	The monitoring module provides monitoring data or is done on a periodic basis based on the component configuration.
Parameters	The monitoring data from the monitoring module in XMP/RDF format.
Process/Stages	The process works as follows: • Interpret the monitoring data as RDF model. • Create a dummy situation to represent the current situation, since it may not be determined yet. • Analyse the monitoring data RDF model to retrieve finer grained situations. Add the identified situations elements to the dummy situation.
Input/Output Data/Interfaces	An RDF model representing the identified situation (a dummy situation and other relevant situations and situational elements attached to it).

3.3.2.2 Situation Reasoning

This part reasons on the situation provided by the situation identification part, and generates more accurate situations, which cannot be directly identified during the process in the situation identification part.

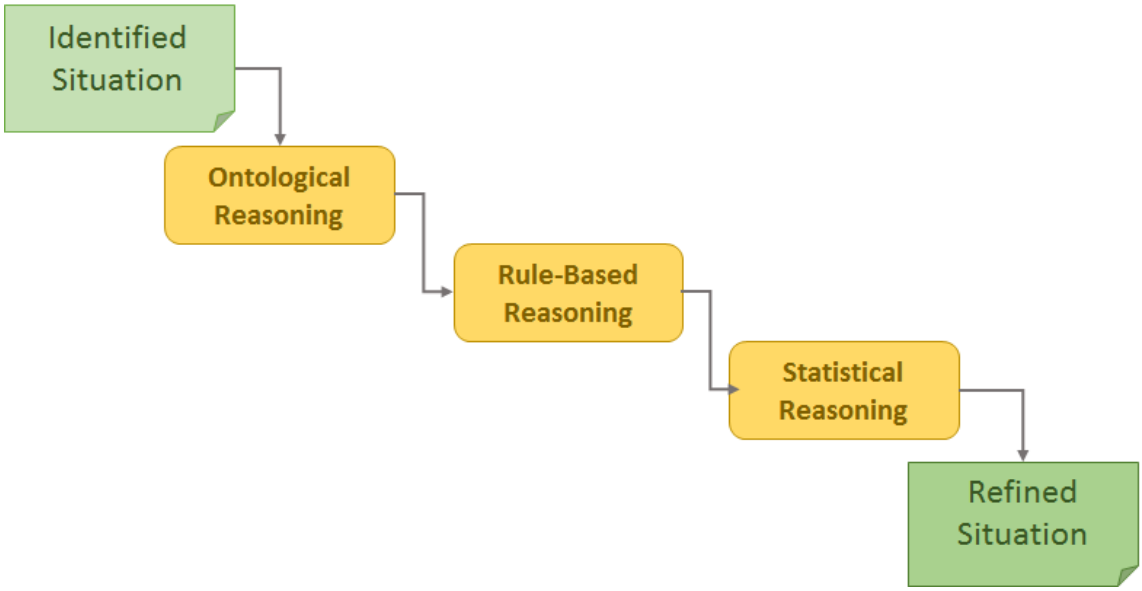

Figure 18: Situation Reasoning

Three types of situation reasoning will be providing by the situation reasoning part, namely ontological situation reasoning, rule-based situation reasoning and statistical situation reasoning. They are performed one after the other, step-by-step to refine the identified situation, as pictured in Figure 18.

Ontological Situation Reasoning

Ontological situation reasoning (Table 18) explores the semantics of the OWL ontology language and the definitions in the SAFIRE situation model, to infer deductive results out of the identified knowledge situation.

At the ontological level, deductive reasoning is based on the semantics of the OWL ontology language and the definitions in the SAFIRE situation model. By performing ontological situation reasoning, much implicit information can be inferred out of the explicit information. For example, assuming that a processing machine is defined as a subclass of a generic device in the ontology. If a rotary table is defined to be a specialisation of a processing machine, the ontological situation reasoning can infer that it is also a generic device.

As SAFIRE uses RDF/OWL to model the situations, deductive reasoning, such as transitive, or sub-class hierarchy reasoning is supported. After reasoning, each instance (identified with its URI) of the situation element is clearly positioned in the situation

model. This is called "subsumption" and serves as basis for the next step rule-based situation reasoning, as it requires all conditions in a rule to be explicitly stated.

Table 18: Ontological Situation Reasoning

Ontological Situation Reasoning	
Description	Situation reasoning based on the semantics of the OWL ontology language and the definitions in the SAFIRE situation model.
Trigger Event	Called by situation identification.
Parameters	The identified knowledge situations in RDF format. The SAFIRE situation model definition in RDF/OWL format.
Process/Stages	Use the OWL ontology language rules to infer implicit RDF statements from the explicit statements in the SAFIRE situation model and the identified situation elements (i.e. "*subsumption*" as a first step).
Input/Output Data/Interfaces	Refined identified situation.

Rule-Based Situation Reasoning

Rule-Based situation reasoning (Table 19) applies user defined domain specific rules to infer new situational knowledge from the existing situational knowledge.

Rule-Based situation reasoning uses the same deductive techniques as in the ontological situation reasoning, but with application-specific rules. During configuration phase, domain specific rules can be defined to help the situation determination services generate more useful and finer grained situational knowledge.

The rules will be defined in rule files and can be updated any time. The Jena rule engine will be used to implement the rule-based situation reasoning, so the rules will be defined in Jena rule syntax.

Code 1 shows an example, which can be explained as "if a user is located in an external location and the maintenance phase is concluded then the machine is in production mode".

```
[rule1:  (?a user:location ?b)
                 (?b rdf:type safire:location)
                 (?c safire:companyLocation !?b)
        -> (?a rdf:type safire:maintenance:production)
]
```

Code 1: Example rule for rule-based situation reasoning

Other types of rule engines can also be used in general, as the service is modular. The W3C has a group working on standards for rule interchange format [RIFWG] which could provide promising outcomes useful for the SAFIRE solution.

Table 19: Rule-Based Situation Reasoning

Rule-Based Situation Reasoning	
Description	Situation reasoning based on the user defined domain specific rules.
Trigger Event	Called by ontological situation reasoning.
Parameters	The refined knowledge situations after ontological situation reasoning. The domain specific rules defined by the user.
Process/Stages	Use the user-defined rules to infer more accurate situations from the available identified, ontologically refined situation.
Input/Output Data/Interfaces	Refined identified and reasoned situation.

Statistical Situation Reasoning

The purpose of statistical situation reasoning (Table 20) is to determine the current situation based on the available situation information so far and historical situations.

Statistic situation reasoning does not rely on strict logical rules, but instead tries to correlate information into possible relations, as suggested by the empirical data. The situation determination uses statistical situation reasoning to rank on-going situations according to the current available knowledge situation, so as to determine the activity of the current systems and the corresponding situation.

This reasoning is performed as the final step, as its purpose is to determine a current activity or processing step based on currently available situation information, which can only be provided after the previous ontological situation reasoning and rule-based situation reasoning have been performed. Normally, a system might have several on-going activities, or processing steps, under certain situations, but, which is the currently active one (defined as the current situation), needs to be decided. The statistical situation reasoning compares the similarities between the situations after the rule-based situation reasoning (which already contains a dummy situation created in the situation identification) with the current on-going situations and ranks them. If the highest similarity value is above a given threshold (between 0 and 1, e.g. 0.95, which is configurable), the according on-going situation is selected as the current one. Otherwise the dummy situation itself is used as the current one, which means a new knowledge-based situation is created. This is one of the ways how situations are populated in the situation determination module.

Table 20: Statistical Situation Reasoning

Statistical Situation Reasoning	
Description	Determines the most possible current situation based on available knowledge situation.
Trigger Event	Called by rule-based situation reasoning.
Parameters	The refined situations after rule-based situation reasoning. A list of on-going situations with their related historical situations from the situation repository.
Process/Stages	The process works as follows: Call situation similarity measure to compute the similarity between the dummy situation and the on-going situations. Sort the on-going situations according to their similarity values. If the similarity value of the first on-going situation is above a given threshold, select it as the current situation. Otherwise use the dummy situation created in situation identification as the current situation.
Input/Output Data/Interfaces	A determined current situation.

The situation similarity measure computes the similarity value between two given situations, which plays a very important role in the situation determination. It supports the statistical situation reasoning function and situation-aware adaptations and selections. Basically, it compares two given situations, by using the situation hierarchy tree defined in the situation model, to tell how similar they are. The actual output of the situation similarity measure is a value between 0 and 1.

A situation is characterised by a situation instance and its related situation resources such as devices, units, processes etc. Or formally, a situation C can be defined as a set of elements, which include one and only one situation element (E_1) and a set of resource elements (situation resources) (E_2). Other environment elements can be added when needed), as shown in Eq. (1):

$$C = \begin{Bmatrix} E_1: & \text{Situation instance} \\ E_2: & [\text{Situation resources}] \end{Bmatrix} \quad \text{Eq. (1)}$$

Then the similarity between two situations C_1 and C_2 can be divided into similarity measures of these two elements, as shown in Eq. (2):

$$sim(C_1, C_2) = \sum_{1}^{2} w_i sim_{E_i}(E_i(C_1), E_i(C_2)) \quad \text{Eq. (2)}$$

Where (C_1, C_2) is the similarity between C_1 and C_2, w_i is the weight of the element set $\sum_1^2 w_i = 1$, $sim_{E_i}(E_i(C_1), E_i(C_2))$ is the similarity between two element sets of C_1 and C_2 (the value is a real number between 0 and 1).

The weight of each situation element can be the same in a simple mode, or a bigger weight can be applied to the more important element in a more complex mode.

3.3.2.3 Situation Provisioning

The purpose of the situation provisioning part is to provide the current knowledge-based situation and a list of other similar knowledge-based situations to other systems or modules.

In the situation provisioning part, is responsible for fulfilling the following two tasks:

- to perform a situation similarity check: the situation similarity measure compares the current on-going situations with preciously acquired historical situations in the repository, and provides the results to other modules in the situation determinator.
- to wrap the results into an RDF-based file format: this part wraps the extracted situation to an RDF model which is a more flexible format and readable by other modules/service which can further use the SAFIRE situation.

Situation Provisioning uses the *Kafka* interface described in section 3.2.4.9 by instantiating a *KafkaSender* which sends the determined situation to a specified Kafka topic into the node/cluster (in serialized RDF-based file format). Other services which want to use the Situation Determination results have to implement an appropriate *KafkaConsumer* and subscribe to the same topic in order to receive the serialized RDF model. Predictive Analytic results will be fed back into the Situation Awareness modules by using an a priori specified Kafka topic.

The communication between the Situation Determination module and Optimisation Engine will be implemented by a HTTP interface (via Post request) using the Metrics API template.

3.3.3 Security issues for Monitoring of User-centred Information

See Section 3.4.3.

3.3.4 Technical Specification

For the implementation of the specified services, several different development tools and IDE[2] will be used. For the overall development and orchestration of all system modules and components the Eclipse IDE will be used. The tested and widely accepted open source development environment for Java offers through a modular system a large plug-in community. Through all these techniques selected for the implementation of the systems architecture and services can be summed up in one environment. The

[2] Integrated Development Environment

repositories will be based on a model- and object-oriented POJO[3] approach. These are the foundation of the data flow and storage and can be easily extended or altered according to the changing requirements and embedded into the system again. Relations and models will also be used in correlation with the serialization for sending and hibernation. The storage layer will, as described, based on a Java JPA solution. The JPA specification is referenced by the Hibernate project which gives the ability to create and use storage systems based on models, JPA-Specifications and POJOs. The implemented services, based on Java as described, will run in the SAFIRE engineering environment.

3.3.4.1 Situation Determination Service

The main classes of the Situation Determination are:

- *IContextExtractionService*: The interface defining the Situation Determinator web-service and its public classes. Via Java annotations the type of the web-service is defined (i.e. SOAP).
- *ContextExtractionService*: Main class of the Situation Determination.
- *IContextRepositoryService*: Interface defining the situation repository service. A reference to this service is hold in the Situation Determinator to allow access to the concrete implementation of the situation repository.
- *ContextExtractionConfiguration*: Holds the configuration of the Situation Determination service during run-time. This configuration object reads the XML configuration file that configures the Situation Determinator.
- *IMonitoringDataModel*: Reference to the monitoring data model, which contains the current monitoring data to be used for situation determination.
- *ContextContainerWrapper*: Wrapper class, which contains the object containing the current identified situation to be provided to other classes.

[3] Plain old Java object

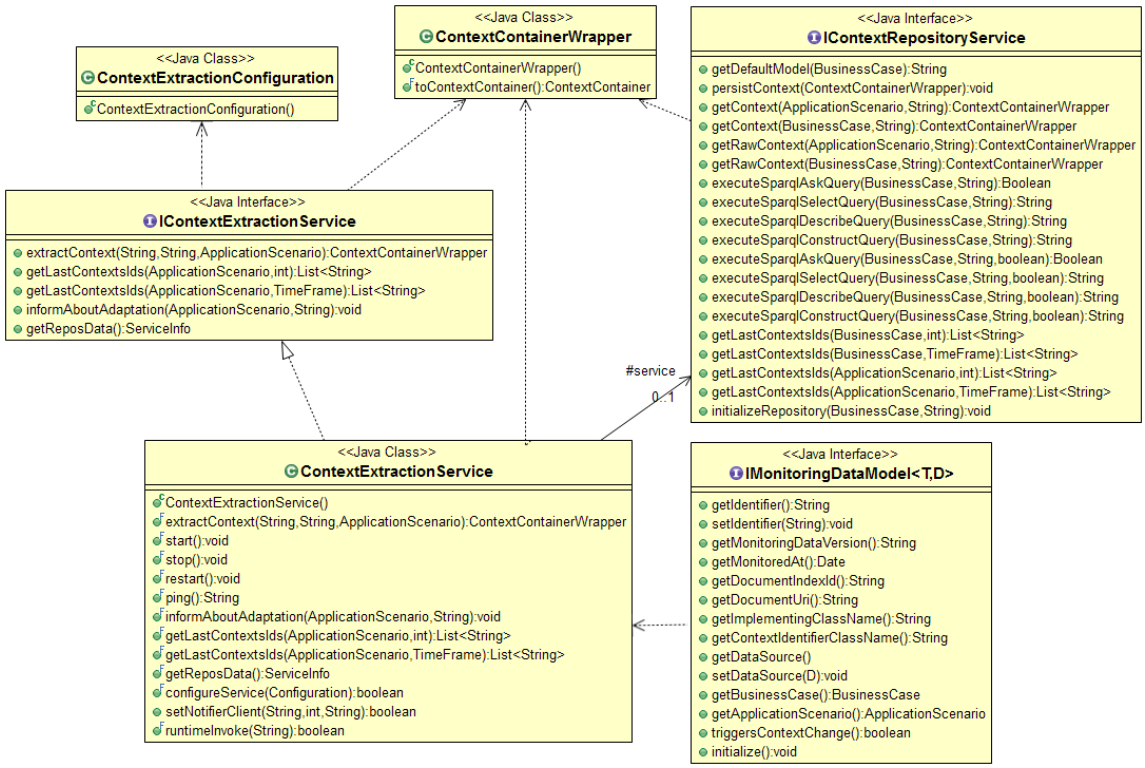

Figure 19: Class Diagram – Situation Determination Service

3.3.4.2 Situation Repository Service

The main classes of the Situation Repository Services are:

- *IContextRepositoryService*: Interface defining the situation repository service. Via Java annotations the type of the web-service is defined (i.e. SOAP).
- *ContextRepositoryService*: Main class of the situation repository service. Identified situations are stored in the situation repository. Via the configuration of the situation determination service the usage of a reasoner can be configured. Reasoning functionality is included in this object, because it is responsible to query the situation repository. In the prototype implementation, the Pellet reasoner will be used.
- *PersistenceUnitService*: Abstract class, where functionalities related to persistence are implemented.

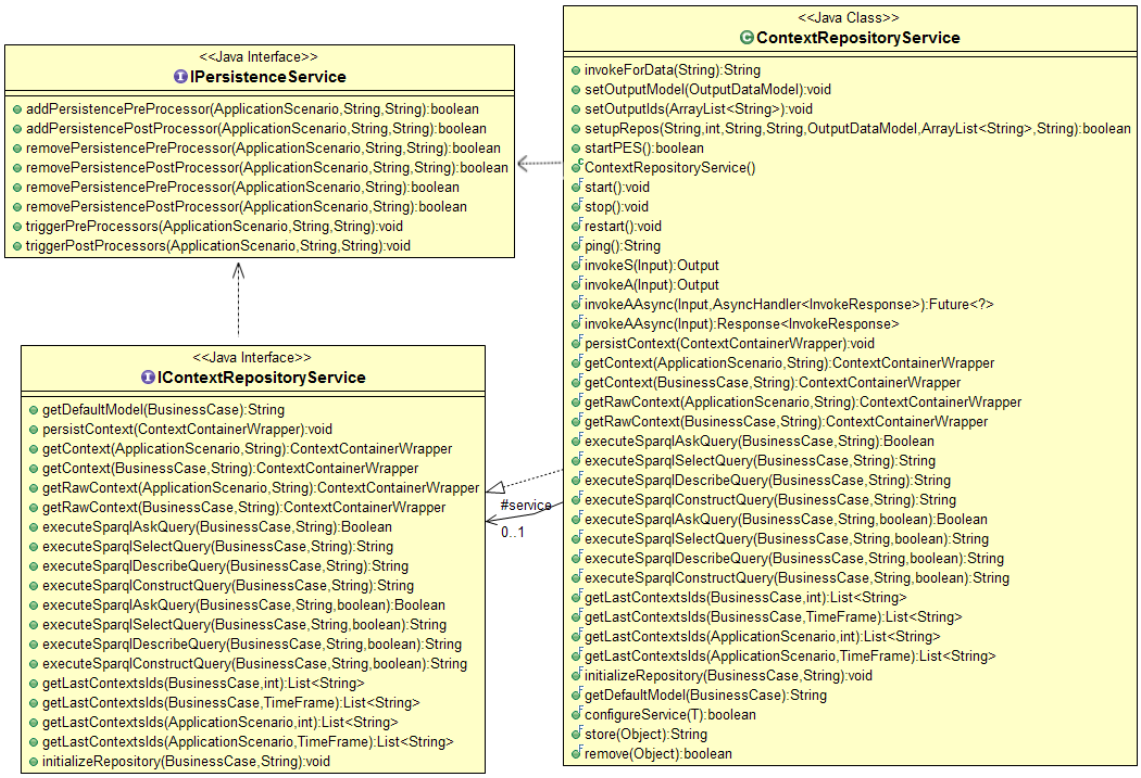

Figure 20: Class Diagram – Situation Repository Service

3.3.4.3 Situation Identifiers

The main classes of the Situation Identifiers are:

- *IContextIdentifier*: The interface defining a situation identifier. A situation identifier is a wrapper that is used to identify a situation based on monitored data and the situation model. In each concrete implementation of a situation identifier the usage of a reasoner can be defined.
- *ContextContainer*: This class is a wrapper object that holds an identified situation during run-time.

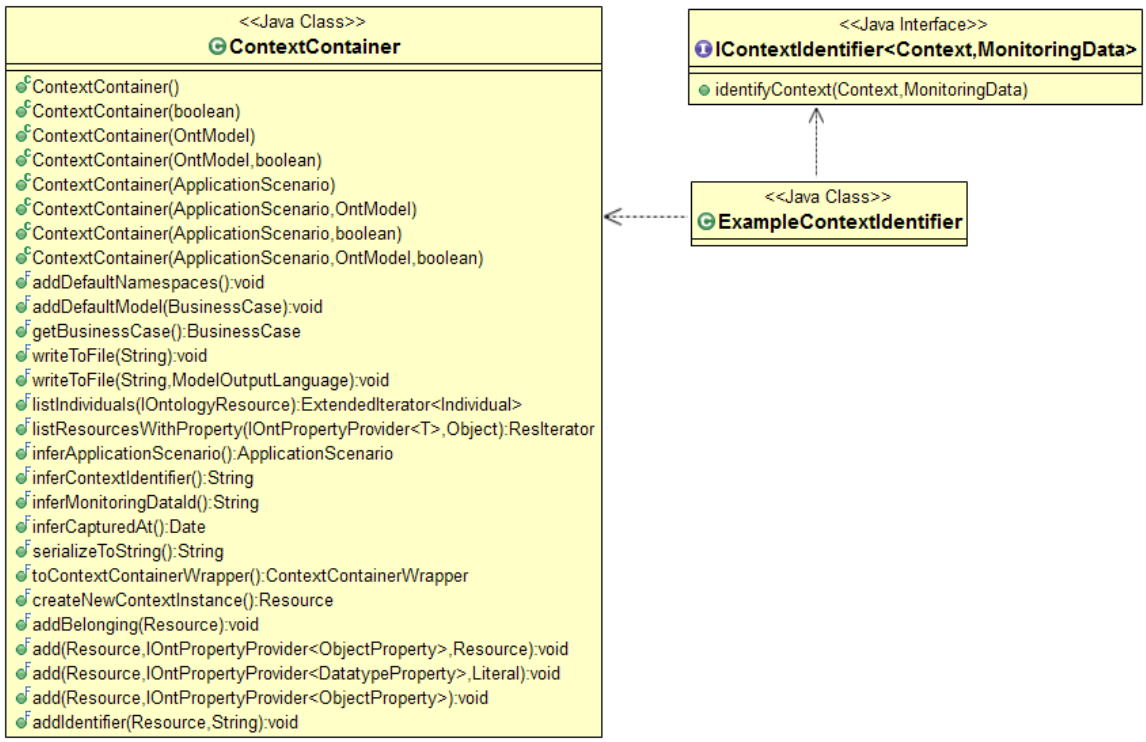

Figure 21: Class Diagram – Situation Identifiers

3.4 INTEGRATION WITH OTHER SAFIRE MODULES

3.4.1 Optimisation Engine

In case a new situation is detected the Situation Determination (SD) will call the Optimisation Engine (OE). In particular, SD will pass the identified situation on to the Objective Function (OF) via the OE. The OF module computes the fitness value for the solutions found during the search-based optimisation process performed by OE. As shown in Figure 22 the SD calls the OE by using a HTTP POST request. The request includes a configuration based on the Metrics API in JSON form (refer to D3.5 for more detailed specification of the configuration).

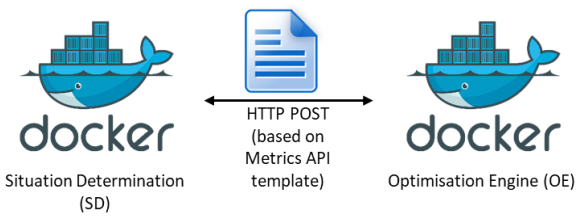

Figure 22: Cooperation between Situation Determination and Optimisation Engine

3.4.2 Predictive Analytics

In case the Predictive Analytics module (PA) has calculated a new predictive analytical result for a monitored system the Situation Determination (SD) will be informed about the recent PA outcome. PA will pass the result to SD using a specified Kafka topic on which SD can subscribe to in order to get the result. The PA result will be passed over in a serialised xml-based format.

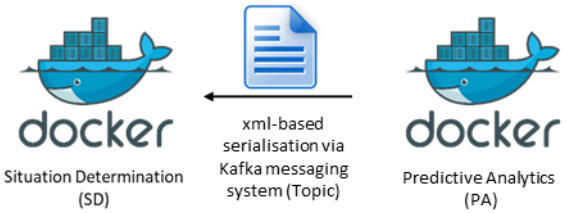

Figure 23: Cooperation between Situation Determination and Predictive Analytics

3.4.3 Implementation of the SAFIRE security framework

The implementation of the situational awareness services shall provide basic functionality to integrate the NGAC of the SAFIRE SPT framework into the situation monitoring and determination services. For more information about the SAFIRE SPT framework, refer to 5.5.

The general functional architecture that is applied when using the situational awareness services with NGAC of the SAFIRE SPT framework is depicted in Figure 24.

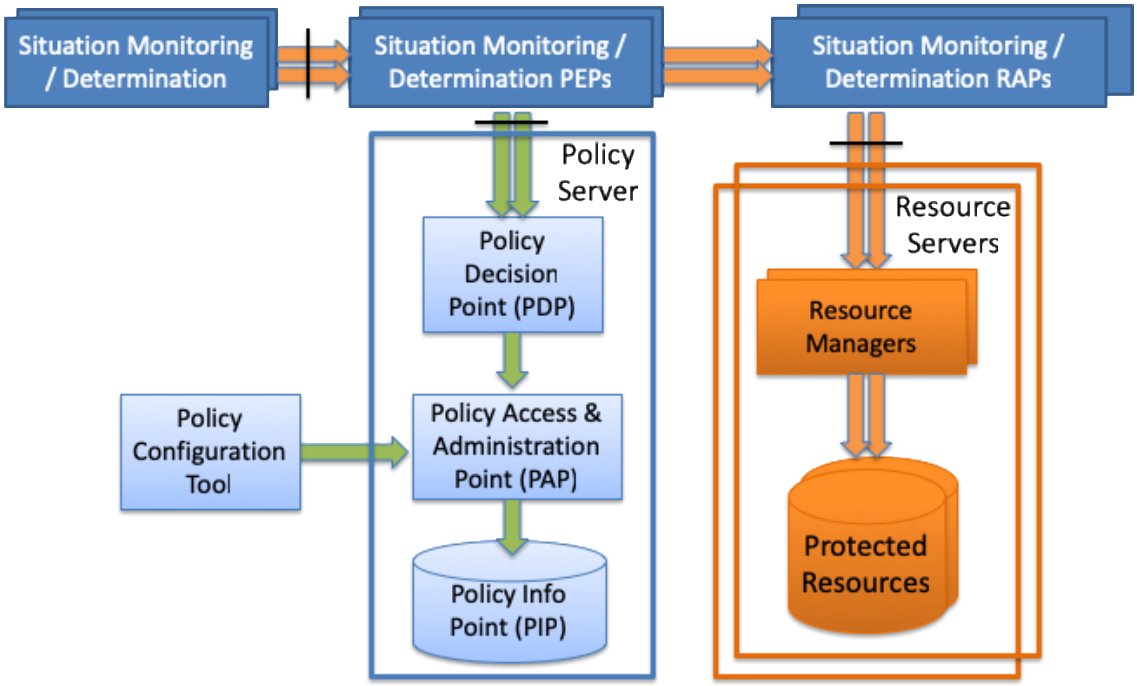

Figure 24: Functional Architecture of Situational Awareness Services using NGAC

The situational awareness services provides functionalities to handle for example configuration files, log-files. Furthermore, the situational awareness services provides abstract functionalities to access the Policy Server of the SAFIRE SPT framework. This abstract functionality can be extended by real implementations inside business case specific customisations. Thereby, all customisations built on top of the general situational awareness services are automatically NGAC aware.

4. FINAL SAFIRE INFRASTRUCTURE SPECIFICATION

SAFIRE platform consists of different modules such as:

- Predictive Analytics Engine
- Optimisation and Reconfiguration Engine
- Situation Model and Determination Services

Each module (related to a different WP) will provide different services for achieving the primary objective of SAFIRE (Cloud-based Situational Analysis for Factories providing Real-time Reconfiguration Services) Apart from the specific services provided by the different modules, there is a common Data Ingestion Layer and a Distributed Messaging System being used for the ingestion and communication between the different modules. The Data Ingestion Layer will be in charge, gathering and sending data from both Interconnected Systems of Production Systems (SoPS) and Connected Product Networks (CPNs), and SAFIRE. An overview of the envisioned platform can be seen in the next figure.

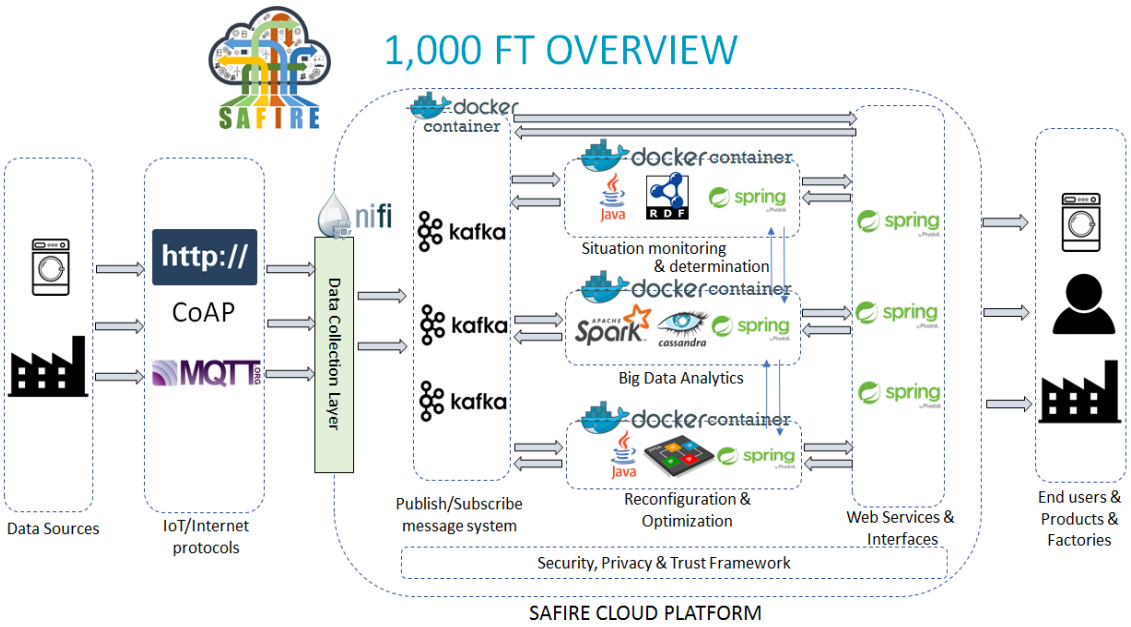

Figure 25: SAFIRE Software Technology Architecture

The main idea is that the services provided by the different modules will be containerised using Docker[4]. Docker is a tool designed to make it easier to create, deploy and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can be assured that the application will run on any other Linux machine regardless of any customised settings that machine might have which could differ from

[4] https://www.docker.com/

the machine used for writing and testing the code. A graphical comparison of the traditional approach of using Virtual Machines vs Docker containers can be seen in the next figure.

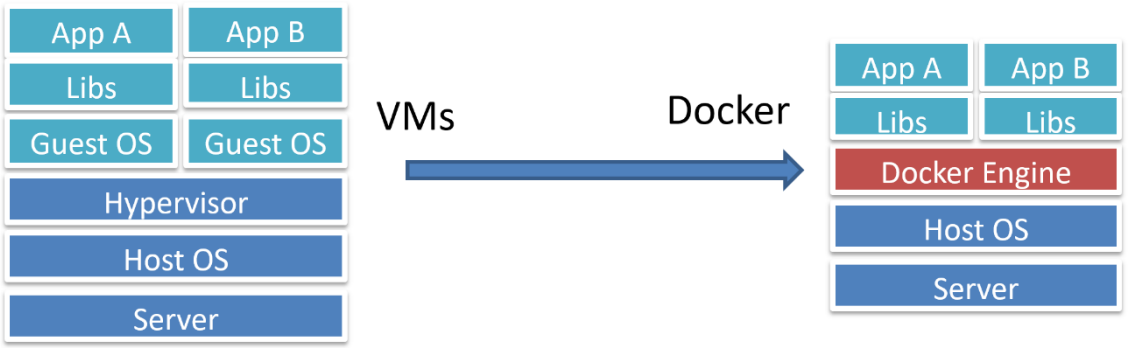

Figure 26: SAFIRE Software Technology Architecture II

The use of Docker in SAFIRE allows the rapid deployment of the different services offered by the different SAFIRE modules. For the early prototypes, the tool selected was Docker Compose, a tool that allows deploying, in a coordinated way, different Docker containers, both the common part (data ingestion and the distributed message system), as well as the specific services of each module. In this way, the development of each module could be done independently.

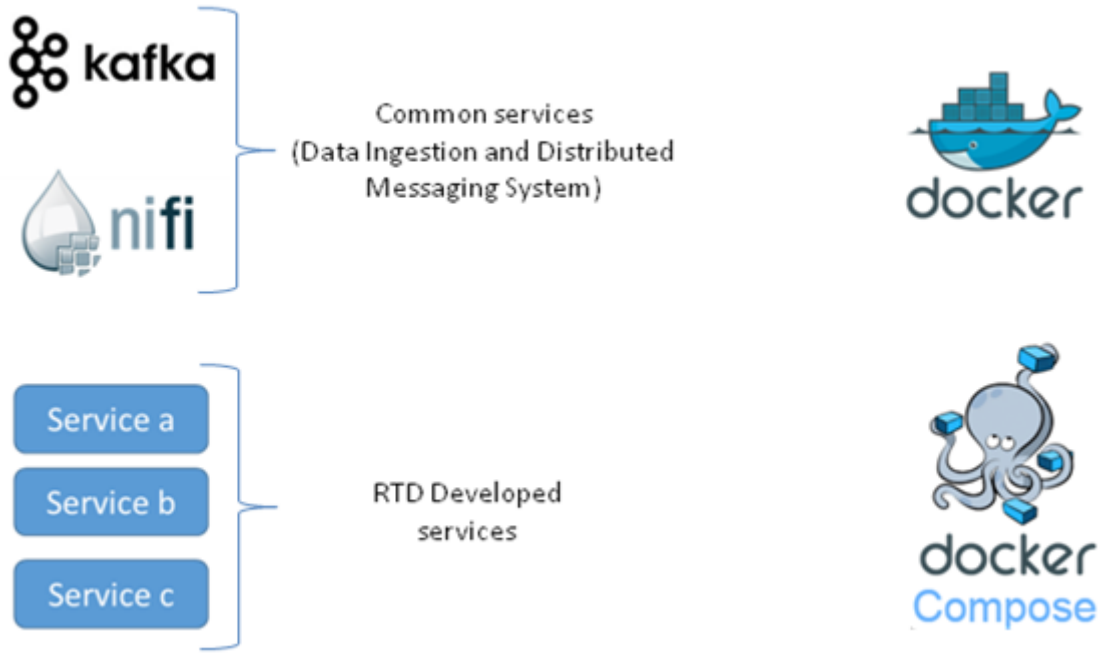

Figure 27: SAFIRE Early prototype infrastructure

For the early prototype, the different modular services were deployed by each RTD partner using Docker Compose. Communications between them in the early prototype was based on Web services.

The final prototypes, once they are all deployed under the same infrastructure they are able to communicate via the distributed messaging system. The distribution of the modules, with respect to the responsible RTD partner, along with the data communication flow, can be seen in the following figure.

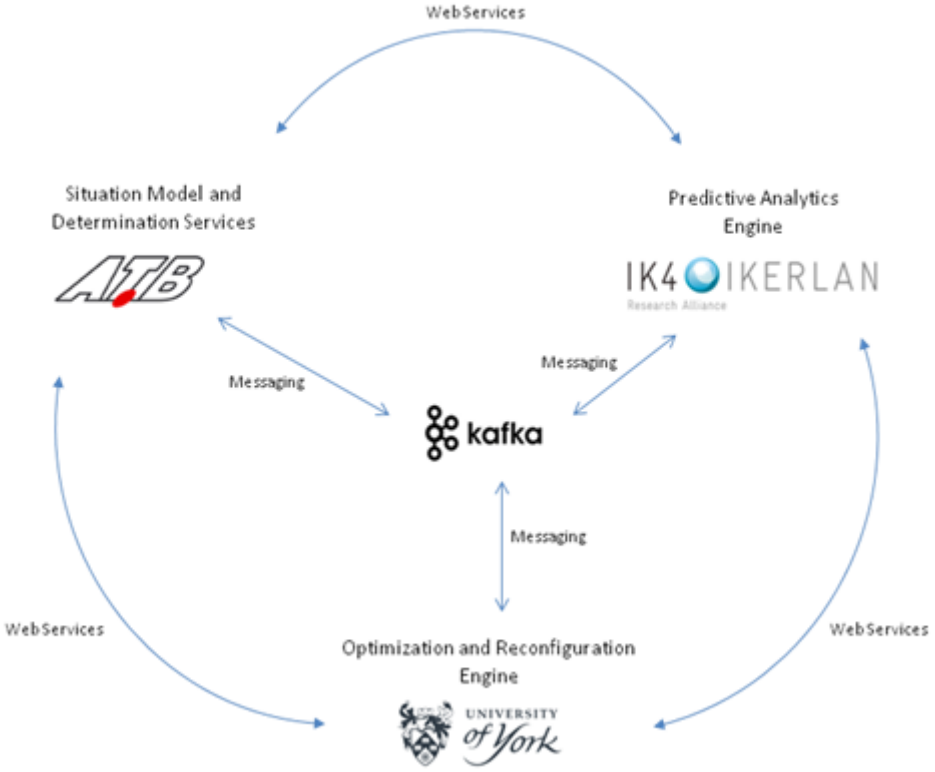

Figure 28: SAFIRE Final infrastructure development environment

Distributed messaging is realised with an Apache Kafka based communication infrastructure (see Figure 29). Apache Kafka is a distributed streaming platform enabling to publish and subscribe to streams of records, similar to a message queue or enterprise messaging system. It stores streams of records in a fault-tolerant durable way and process streams of records as they occur.

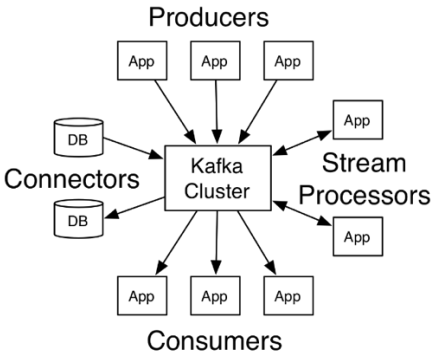

Figure 29: Communication partners in a KAFKA communication infrastructure

Therefore, full prototype components of the Situational Determination module have to provide kafka compliant input and output interfaces which support the topic based kafka messaging method. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.

All these services have to run in the same environment in public or private clouds. Therefore, an orchestrator is needed to manage how, when, where and with what computing resources each service is launched. Orchestrators are very useful, as they allow to manage a cluster of computers in a unified view and improve the resource efficiency as can be seen in the next figure.

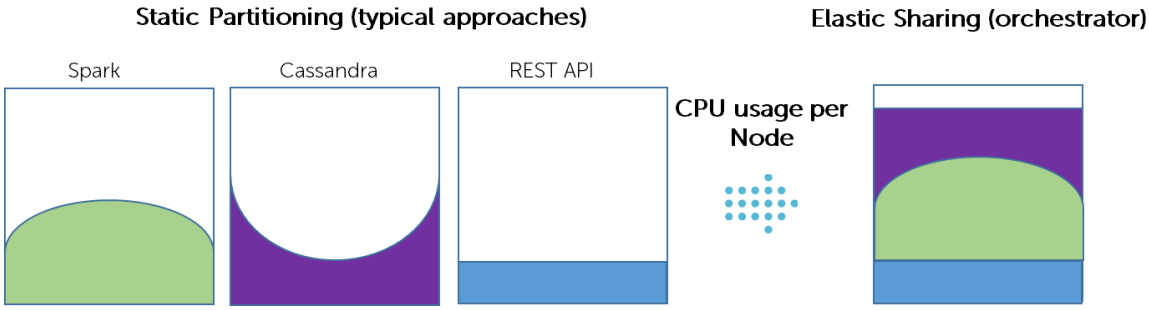

Figure 30: Static Partitioning of Clusters vs Elastic Sharing via an Orchestrator

In SAFIRE, the advantages of using an orchestrator can be seen clearly in the Predictive Analytics module. For example, in case an analysis that requires a large number of computing resources is required, the orchestrator could be used to fine-tuning the computing resources needed for the analysis while letting enough computing resources for the other modules. In addition, since SAFIRE is going to run in the cloud, we can use the cloud's elasticity features to be able to use more or fewer resources as needed to save costs or speed up certain critical reconfiguration processes.

In the literature, there are different orchestrators such as Nomad, Yarn, Mesos, Swarm or Kubernetes. In order to choose the orchestrator to be used in SAFIRE, several things were considered:

- Preferably free and open source.
- Docker support.
- Good automated deployment, scaling and management.
- Well tested.
- Large community
- Good support in several platforms

After reviewing several possibilities, Kubernetes was chosen. However, changing the orchestrator should not imply hard work as we have previously containerised all the different services, and all the orchestrators support Docker containers nowadays.

Kubernetes is a very popular and well supported container orchestration platform. It is very capable of managing the deployment and scaling of containerized applications of any size and complexity. It was originally designed by Google and is now maintained by the CNCF (Cloud Native Computing Foundation).

Using Kubernetes as an orchestrator, the infrastructure would be as depicted in the next figure.

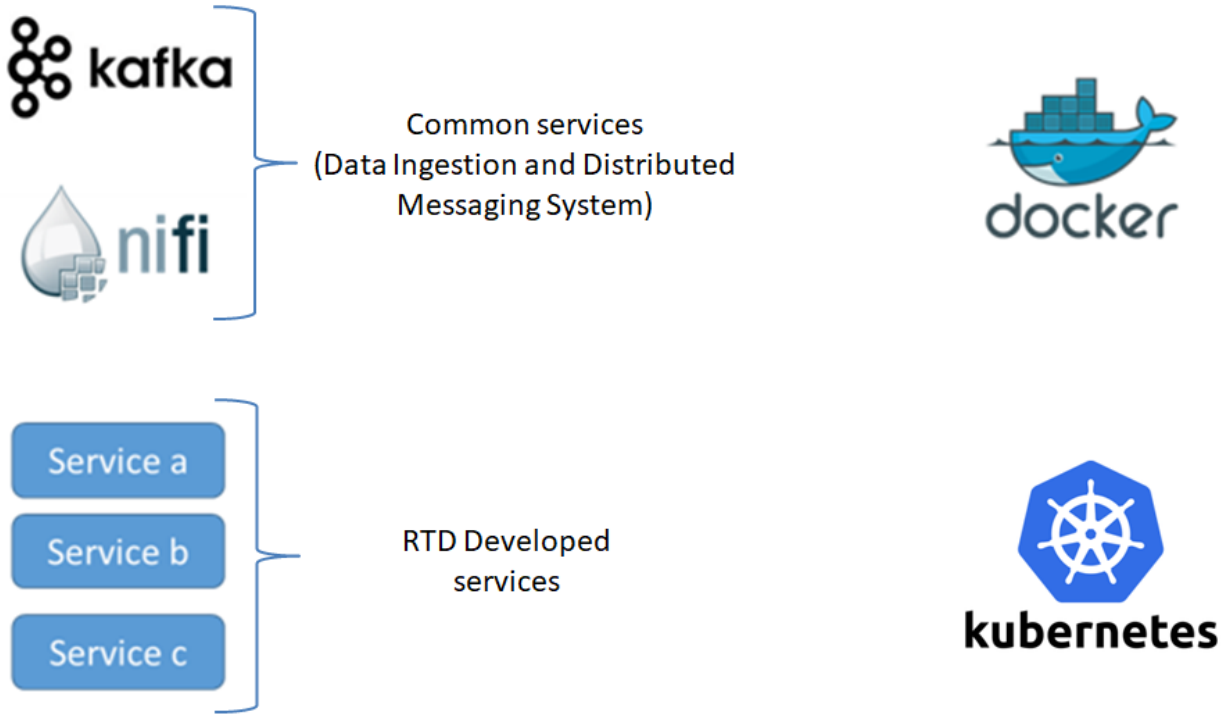

Figure 31: Final infrastructure

5. BACKGROUND TECHNOLOGIES

To create the described solution based on the functional specifications, several technologies and tools are needed. These tools support the creation of specific services describe out of the requirements or offer the ability to sum up all parts of the system.

All services specified in this deliverable that will be implemented from scratch will be implemented utilizing the Java programming language. For developing and maintaining the situation model (ontology), Protégé will be used. Protégé is a widely accepted open-source platform offers the ability to develop and visualise complex models, including properties and relations, and allows for export of the model in different formats (e.g. rdf, owl, turtle, etc.). To monitor the information out of files and Web Services several additional Java compliant libraries will be used.

The repositories are based on a POJO-based approach, which is serialized into an abstraction layer that covers many legacy database systems to work with (e.g. MySQL or H2). The specified engineering tools and core services rely on the abstraction layer as persistence enterprise system and does not require a mandatory database but only a compliant one as technical foundation.

5.1 SOFTWARE TOOLS

For the implementation of the specified engineering tools and core services, several different development tools and IDE will be used. For the overall development and orchestration of all system modules and components the Eclipse and IntelliJ IDE will be used. The tested and widely accepted Open Source development environment for Java offers through a modular system a large plug-in community. Through all these techniques selected for the implementation of the systems architecture and services can be summed up in one environment.

The repositories will be based on a model- and object-oriented POJO approach. These are the foundation of the data flow and storage and can be easily extended or altered according to the changing requirements and embedded into the system again. Relations and models will also be used in correlation with the serialization for sending and hibernation. The storage layer will, as described, based on a Java JPA solution. The JPA specification is referenced by the Hibernate project which gives the ability to create and use storage systems based on models, JPA-Specifications and POJOs.

The software tools, together with their version, link and name of the task they are planned for, are listed in Table 21. These tools will be used to develop and run the specified situation determination module against the systems concept and hereby specified functionality. It resembles state-of-the-art tools and software to provides a modular, extendable and expandable service-oriented approach.

Table 21: Overview of used software tools

Functionality	Software	Version	Link
Programming Language	Java	>= 1.8.0_xx	http://www.java.com
IDE	Eclipse IntelliJ	>= 4.7 >= 2018.2	http://www.eclipse.org https://www.jetbrains.com/idea/
Version Control	GIT	>= 2.19.0	http://www.git-scm.com/
Build Automation	Apache Maven	>= 3.5.4	https://maven.apache.org/
Issue Management	Jira	>= 7.5	https://www.atlassian.com/software/jira
XML Configuration Wrapper	Simple XML	>= 2.7	http://simple.sourceforge.net/
Web Application Framework	Spring	>= 4.1	http://www.springsource.org/
Runtime Environment / Application Server	Apache Tomcat	>= 9.0	http://tomcat.apache.org/
JPA-based persistence	Hibernate	>= 5.2	http://hibernate.org/
Database	H2 Database Oracle Database PostgreSQL	>= 1.4 >= 12cR2 >= 10.5	http://www.h2database.com/ https://www.oracle.com https://www.postgresql.org/
RDF / OWL API	Jena	>= 3.1	http://jena.apache.org/
RDF Storage	SDB / TDB	>= 1.3 / 1.1	http://jena.apache.org/
	Joseki	>= 3.4	http://jena.apache.org/
Indexing	Apache Lucene	>= 7.1	http://lucene.apache.org/
Container Virtualisation	Docker	>= 18.06.1	https://www.docker.com/
Flow Automation	Apache Nifi	>= 1.7.1	https://nifi.apache.org/
Distributed Streaming Platform	Apache Kafka	>= 2.0.0	https://kafka.apache.org/
Ontology modelling	Protégé	>= 5.2.0	https://protege.stanford.edu/

5.2 IMPLEMENTATION/INTEGRATION CONTROL SOFTWARE

The ensure are coherent development of the Situational Awareness Services specified in this deliverable as well as with the other services of the SAFIRE solution, the consortium uses a set of "control" software systems to manage their interconnected development. Besides the continuously communications between the partners, three software tools will be used as basic implementation controlling foundation:

- A Version Control system (i.e. GIT) to manage all source developed from all partners in one system.
- An Issue Tracking System (i.e. Atlassian Jira) to make testing public and accessible between all partners to fix specific as well as inter-connected errors.
- A continuous integration system (i.e. Atlassian Bamboo) to integrate all software components on a permanent basis.

6. BUSINESS CASE SPECIFIC CUSTOMISATION AND CONFIGURATION

The specified components are generic parts, customisable and configurable for different application domains. The following sections describe steps needed for the situation model, situation monitoring and situation determination for an application in an application domain.

6.1 SITUATION MODEL

In order to provide a custom solution for the different business cases and the company specific scenarios, a different situation model should be created for each purpose. The custom situation models will include additional entities that extent the generic ones, and respective relations as needed. As it is foreseen, the generic entities that will be mainly extended to the customised situation models, are the Generic Datum and the Metric, since the specific use-case scenarios will require additional input data and evaluation measurements. The entities and the respective additional relations are being described in detail in the early and full prototype documents.

6.2 SITUATION MONITORING

The generic situation monitoring component is customisable for different kinds of interfaces and data structures. Therefore, it is possible to individually configure indexes and to develop data parsers and data analysers based on requirements for monitored data sources (communication protocols, data structures).

6.2.1 Customisation

The monitoring follows the following process:

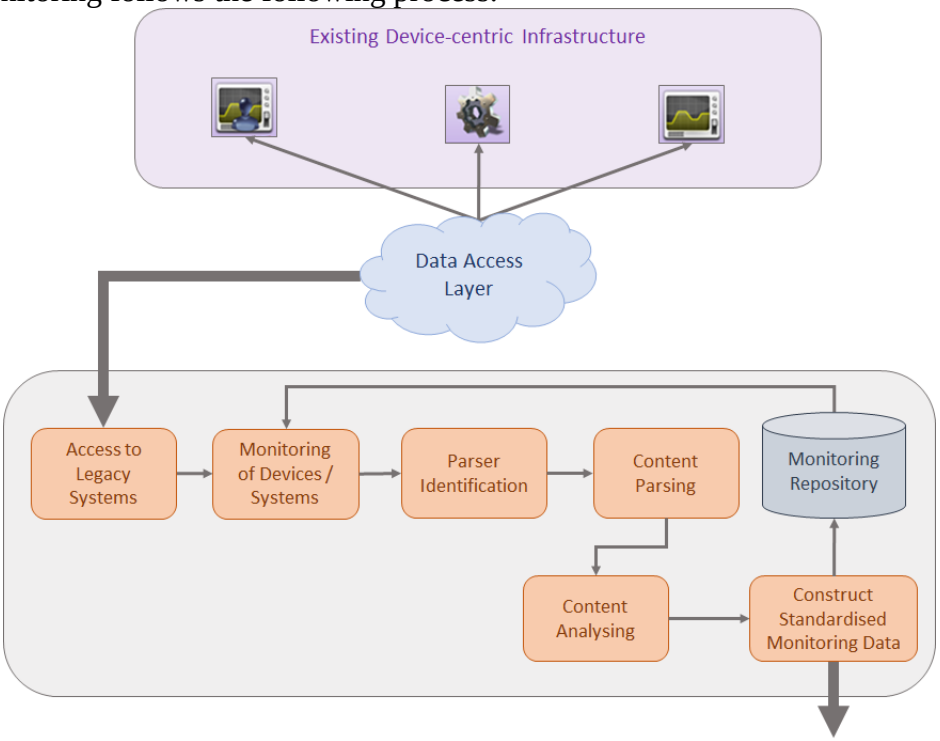

Figure 32: Situation Monitoring Process

For customising the Situation Monitoring for a specific application scenario, one has to implement the following business case specific “plugins”, which will be included in the Situation Monitoring Service via its configuration (see 6.2.2):

System Monitors:

In order to be able to monitor a specific system, one has to implement a Monitor to ingest data into the Situational Monitoring. For implementation, templates are available to allow for an easy integration into the Situation Monitoring Service (e.g. FileSystemMonitor, WebServiceMonitor, DatabaseMonitor as described in Section 3.2.4.5).

Parsers:

For each specific monitor that was implemented, a corresponding parser has to be implemented to be able to parse the content and prepare it for analysing. For implementation, templates are available to allow for an easy integration into the Situation Monitoring Service (e.g. FileParser, WebServiceParser, DatabaseParser as described in Section 3.2.4.6).

Analysers:

For each specific monitor that was implemented, a corresponding analyser has to be implemented to be able to analyse the monitored content. The analyser is responsible for filling the specific monitoring data model. For implementation, templates are available to allow for an easy integration into the Situation Monitoring Service (e.g. FileAnalyser, WebServiceAnalyser, DatabaseAnalyser as described in Section 3.2.4.7).

Monitoring Data Models:

In order to be able to monitor a specific system, one has to implement a Monitor to ingest data into the Situational Monitoring. For implementation, templates are available to allow for an easy integration into the Situation Monitoring Service (as described in Section 3.2.4.8).

6.2.2 Configuration

The individual implementations have to be addressed in a monitoring configuration to define bundles of classes which are responsible for the monitoring of a specified data source.

In the following the xml schema (monitoring-config.xsd) for the situational monitoring configuration is listed. The configuration elements are described in detail.

```
monitoring-config.xsd
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://www.atb-bremen.de" >
  <xs:element name="config">
    <xs:complexType>
      <xs:sequence>
```

```
      <xs:element ref="indexes"/>
      <xs:element ref="datasources"/>
      <xs:element ref="interpreters"/>
      <xs:element ref="monitors"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="indexes">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" ref="index"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="index">
  <xs:complexType>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="location" use="required" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
<xs:element name="monitors">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" ref="monitor"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="monitor">
  <xs:complexType>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="datasource" use="required" type="xs:IDREF"/>
    <xs:attribute name="index" use="required" type="xs:IDREF"/>
    <xs:attribute name="interpreter" use="required" type="xs:IDREF"/>
  </xs:complexType>
</xs:element>
<xs:element name="datasources">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" ref="datasource"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="datasource">
  <xs:complexType>
    <xs:attribute name="class" use="required"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="monitor" use="required"/>
    <xs:attribute name="options" use="required"/>
    <xs:attribute name="type" use="required" type="xs:NCName"/>
    <xs:attribute name="uri" use="required" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
<xs:element name="interpreters">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" ref="interpreter"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="interpreter">
  <xs:complexType>
    <xs:sequence>
```

```
        <xs:element maxOccurs="unbounded" minOccurs="1" ref="configuration"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:ID"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="configuration">
    <xs:complexType>
      <xs:attribute name="analyser" use="required"/>
      <xs:attribute name="parser" use="required"/>
      <xs:attribute name="type" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Code 2: Monitoring Config Schema

indexes - Each index entry has the following mandatory attribute

- id: The unique name of the index
- location: The URI of the location the index is stored

datasources - Each datasource entry has the following mandatory attributes

- id: The unique name of the datasource
- type: The type of the datasource. Possible values are: file, webservice, database
- monitor: The class of the monitor to be used. Possible values are:

```
package de.atb.context.monitoring.monitors.database.DatabaseMonitor
package de.atb.context.monitoring.monitors.file.FileSystemMonitor
package de.atb.context.monitoring.monitors.file.FilePairSystemMonitor
package de.atb.context.monitoring.monitors.file.FileTripletSystemMonitor
package de.atb.context.monitoring.monitors.webservice.WebServiceMonitor
```

- options: Options for the datasource can be entered using this value. The options are dependent on the datasource to be used
- uri: The uri of the data source to be monitored
- class: The following datasource implementations are available

```
package de.atb.context.monitoring.config.models.datasources.DatabaseDataSource
package de.atb.context.monitoring.config.models.datasources.FilePairSystemDataSource
package de.atb.context.monitoring.config.models.datasources.FileSystemDataSource
package
de.atb.context.monitoring.config.models.datasources.FileTripletSystemDataSource
package de.atb.context.monitoring.config.models.datasources.WebServiceDataSource
```

Interpreters:

Each interpreter entry has the following mandatory attributes

- id: The unique name of the interpreter
- configuration

– analyser: The analyser class to be used. The following implementations are available:

```
package de.atb.context.monitoring.analyser.database.DatabaseAnalyser
package de.atb.context.monitoring.analyser.file.FileAnalyser
package de.atb.context.monitoring.analyser.file.FilePairAnalyser
package de.atb.context.monitoring.analyser.file.FileTripletAnalyser
package de.atb.context.monitoring.analyser.webservice.WebServiceAnalyser
```

– parser: The parser class to be used. The following implementations are available:

```
package de.atb.context.monitoring.parser.database.DatabaseParser
package de.atb.context.monitoring.parser.file.FileParser
package de.atb.context.monitoring.parser.file.FilePairParser
package de.atb.context.monitoring.parser.file.FileTripletParser
package de.atb.context.monitoring.parser.webservice.WebServiceParser
```

– type: Currently only used for File analyser and parser. Defines the file extensions to be used.

monitors – monitors are grouping datasources, interpreters and indexes for the monitoring of a specific datasoruce (a monitor for a data source). Each monitor entry has the following mandatory attributes

- id: The unique name of the monitor
- datasource: Id of one datasource which was configured.
- interpreter: Id of one interpreter which was configured.
- Index: Id of one index which was configured.

An example of a whole configuration is provided in the following:

```
monitoring-config.xml
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="http://www.atb-bremen.de"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.atb-bremen.de monitoring-config.xsd">

   <indexes>
      <index id="index-dummy" location="indexes/dummy"></index>
      <index id="index-safire" location="indexes/safire"></index>
   </indexes>

   <datasources>
      <datasource
id="datasource-dummy"
type="file"
monitor="de.atb.context.monitoring.monitors.file.FilePairSystemMonitor"
uri="target/test-classes/filepairmonitor"
options="extensionOne=1&amp;extensionTwo=2"
class="de.atb.context.monitoring.config.models.datasources.FilePairSystemDataSource"
      />
   </datasources>

   <interpreters>
```

```
      <interpreter id="interpreter-dummy">
         <configuration
type="*"
parser="de.atb.context.monitoring.parser.file.DummyFilePairParser"
analyser="de.atb.context.monitoring.analyser.file.DummyFilePairAnalyser" />
      </interpreter>
   </interpreters>

   <monitors>
      <monitor
         id="monitor-dummy"
         datasource="datasource-dummy"
         interpreter="interpreter-dummy"
         index="index-dummy" />
   </monitors>
</config>
```

Code 3: monitoring-config.xml

Finally, a service configuration has to be created which defines the services to be deployed enriched with additional network information (as host, location, name, server, proxy).

```
services-config.xml
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="http://www.atb-bremen.de" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <services>
      <service id="AmIMonitoring">
         <host>localhost</host>
         <location>http://localhost:19001</location>
         <name>AmIMonitoringService</name>
         <server>de.atb.context.services.AmIMonitoringService</server>
         <proxy>de.atb.context.services.IAmIMonitoringService</proxy>
      </service>
      <service id="AmI-repository">
         <host>localhost</host>
         <location>http://localhost:19002</location>
         <name>AmIMonitoringDataRepositoryService</name>
         <server>de.atb.context.services.AmIMonitoringDataRepositoryService</server>
         <proxy>de.atb.context.services.IAmIMonitoringDataRepositoryService</proxy>
      </service>
      <service id="PersistenceUnitService">
         <host>localhost</host>
         <location>http://localhost:19004</location>
         <name>PersistenceUnitService</name>
         <server>de.atb.context.services.PersistenceUnitService</server>
         <proxy>de.atb.core.services.IPersistenceUnitService</proxy>
      </service>
   </services>
</config>
```

Code 4: Services Config for Situation Monitoring

6.3 SITUATION DETERMINATION

The generic situational determination component is customisable for different observed situations. It receives standardised monitoring data from the situational monitoring component and is able to identify and provide the situation of observed data sources.

6.3.1 Customisation

The situation determination follows the following process:

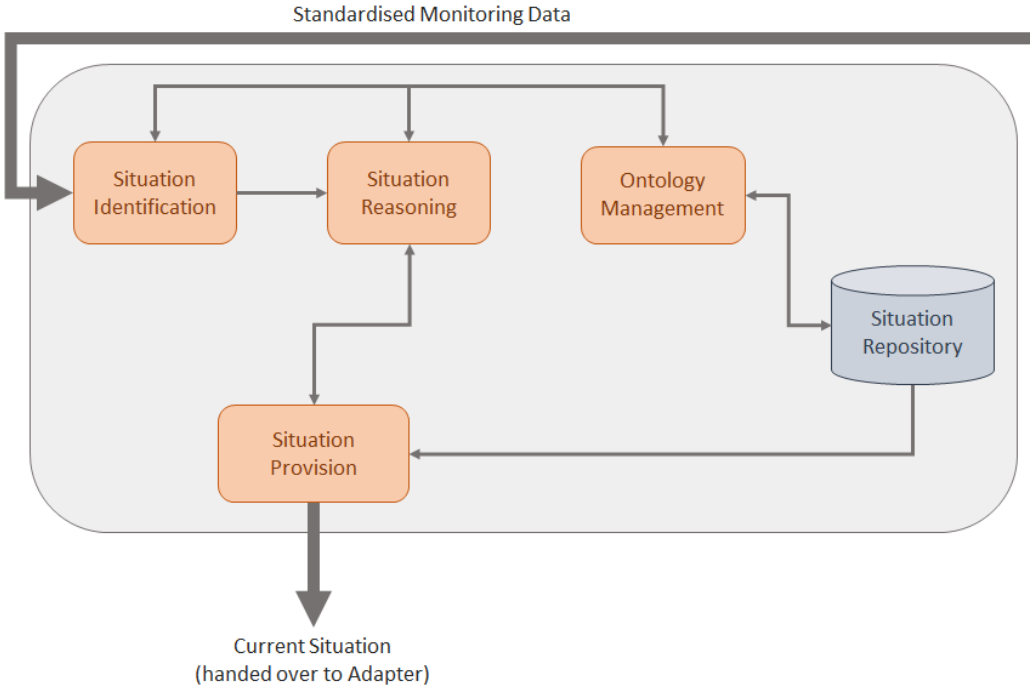

Figure 33: Situation Determination Process

Situation Identifiers

The main classes of the Situation Identifiers are:

- *IContextIdentifier*: The interface defining a situation identifier. A situation identifier is a wrapper that is used to identify a situation based on monitored data and the situation model. In each concrete implementation of a situation identifier the usage of a reasoner can be defined.
- *ContextContainer*: This class is a wrapper object that holds an identified situation during run-time.

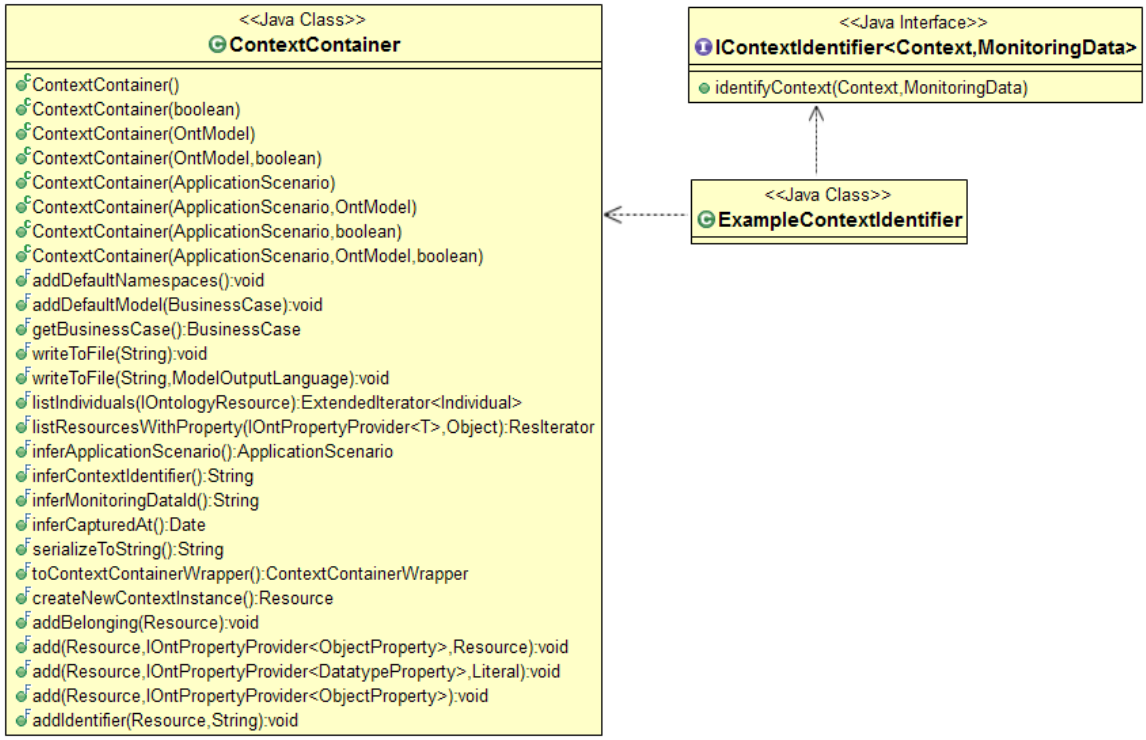

Figure 34: Class Diagram – Situation Identifiers

6.3.2 Configuration

Finally, in the service configuration has to be to be added the situational determination related services to be deployed, enriched with additional network information (as host, location, name, server, proxy).

```
services-config.xml
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="http://www.atb-bremen.de" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <services>
      <service id="ContextExtractionService">
         <host>localhost</host>
         <location>http://localhost:19001</location>
         <name>ContextExtractionService</name>
         <server>de.atb.context.services.ContextExtractionService</server>
         <proxy>de.atb.context.services.IContextExtractionService</proxy>
      </service>
      <service id="ContextRepositoryService">
         <host>localhost</host>
         <location>http://localhost:19002</location>
         <name>ContextRepositoryService</name>
         <server>de.atb.context.services.ContextRepositoryService</server>
         <proxy>de.atb.context.services.IContextRepositoryService</proxy>
      </service>
   </services>
</config>
```

Code 5: Services Config for Situation Determination

7. REQUIREMENTS COVERAGE

This section presents the coverage of the related requirements by the functionalities of the situational awareness components, as well as the SAFIRE infrastructure.

7.1 SITUATIONAL AWARENESS

Table 22: Requirements Coverage for the category Situational Awareness

No.	Requirement	Overall Priority	EP	FP
U54	Able to change existing or adding new monitoring sources with minimal effort	SHALL	+	+
U55	Able to support collection of environmental data to identify current situation	SHALL	+	+
U56	Able to support collection of operator's behaviours to identify current situation	SHALL		+
U57	Able to monitor machine current status data to identify situation	SHALL	+	+
U58	Able to monitor machine health status to identify current situation	SHALL	+	+
U59	Able to monitor overall equipment effectiveness (OEE) to identify current situation	SHALL	+-	+
U60	Able to monitor production status to identify current situation	SHALL	+	+
U61	Able to support collection of data from proNTo behaviours to identify current situation	SHALL	+-	+
U62	Able to monitor Hob Temperature status to identify current situation	SHALL	+-	+
U63	Able to monitor Pot Boiling status to identify current situation	SHALL	+-	+
U64	Able to provide situational information based on raw and monitored data	SHALL	+	+

No.	Requirement	Overall Priority	EP	FP
U65	Able to extract situational information from monitored machines	SHALL	+-	+
U66	Able to dynamically extract situational information from sensor data	SHALL	+-	+
U67	Able to change existing or add new situations with minimal effort	SHALL	+	+
U68	Able to model situations under which a set of machines is operating	SHALL	+-	+
U69	Able to model situations under which a production process is operating	SHALL	+-	+
U70	Able to extract situational information from sets of related machines	SHOULD	+-	+
U71	Able to extract situational information from operator actions	SHOULD		+
U72	Able to evaluate situation with respect to capacity, performance, availability (OEE) of monitored machines	SHALL		+
U73	Able to evaluate situation with respect to capacity, performance, availability (OEE) from sets of related machines	SHALL		+
U74	Able to anticipate alarms before they occur based on current situation	SHALL		+
U75	Able to evaluate status of machine job queues (if available)	SHOULD		+
U76	Able to model situation under which proNTo is operating	SHOULD	+	+
U77	Able to extract situational information from proNTo and from other systems	SHALL	+-	+

7.2 PERFORMANCE

Table 23: Requirements Coverage for the category Performance

No.	Requirement	Overall Priority	EP	FP
U115	Does not negatively affect the usual production processes	SHALL	+-	+
U116	Support for scalability in the size of cloud and computing resources	SHALL	+-	+
U117	Support for horizontal scalability to many machines	SHALL	+-	+
U118	Capable of real time data ingestion (registering data)	SHALL	+-	+
U119	Capable of batch processing of data (offline analysis)	SHALL	+-	+
U120	Capable of real time data processing	SHALL	+-	+
U122	Able to analyse relevant data within a given timeframe	SHALL	+-	+
U123	Capable of storing up to 5 TB/year/machine with resource recycling facilities	SHALL	+-	+

7.3 INTERFACE

Table 24: Requirements Coverage for the category Interface

No.	Requirement	Overall Priority	EP	FP
U126	Able to be integrated with PLC	MAY		+-
U127	Provides a web-based user interface	SHALL	+-	+
U128	Implemented as a set of web services / web-based solution	SHALL	+	+
U129	Able to customise the interfaces	SHALL	+	+
U130	Able to access data stored in a relational database	SHALL	+	+
U131	Able to receive and send data from/to a remote location	SHALL	+-	+
U132	Supports human-machine interface on multiple devices	SHALL	+	+

No.	Requirement	Overall Priority	EP	FP
U133	Able to interface with third-party reporting systems	SHOULD		+[5]
U134	Able to interface with third-party dashboards	SHALL		+[6]
U135	Provides interface for knowledge base access	SHALL		+
U137	Able to be integrated with proNTo	SHALL	+-	+
U138	Supports the ONA Machine Protocols	SHALL	+-	+

7.4 INTEGRATION WITH PRODUCTS/PROCESSES

Table 25: Requirements Coverage for the category Integration with Products/Processes

No.	Requirement	Overall Priority	EP	FP
U139	Supports integration with schedulers / workshop managers	SHOULD	+-	+
U140	Supports integration with communication devices (Cisco, any other, …)	SHOULD	+	+
U141	Supports integration with rapid prototyping tools	SHOULD		+

7.5 COMMUNICATIONS

Table 26: Requirements Coverage for the category Communications

No.	Requirement	Overall Priority	EP	FP
U142	Support for internet/ethernet communications	SHALL	+	+
U143	Support for VPN connectivity	SHOULD	+	+
U144	Support for machine / cloud protocols	SHALL		+
U145	Support for machine / fog computing	SHOULD		+
U146	Support for communications between operators and platform	SHOULD	+	+

[5] Can be added in future if business partners specify the third-party reporting systems to be used and the scope of the interfacing.
[6] Can be added in future if business partners specify the third-party dashboard to be used and the scope of the interfacing.

U147	Support for communications with business analytics	MAY		+

7.6 HARDWARE / PLATFORM / DEVICES

Table 27: Requirements Coverage for the category Hardware/Platform/Devices

No.	Requirement	Overall Priority	EP	FP
U148	Supports continuous operation (24h per day, 7 days per week)	SHALL		+
U149	As platform independent as possible	SHALL	+	+
U150	Supports multiple deployment scenarios including Cloud / On Premise / Hybrid / Third-party Managed	SHOULD		+
U151	Provided connector with proNTo system operates under Windows	SHALL	+	+