SAFIRE

Cloud-based Situational Analysis for Factories providing Real-time Reconfiguration Services

Project Number 723634

D5.6 Integrated Methodology

Version 1.0
25 October 2019
Final

Public Distribution

ATB, IKERLAN, The Open Group, University of York

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

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SAFIRE Project Partners accept no liability for any error or omission in the same.

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	Document Template	12 September 2019
0.5	Full draft with contributions from RTD partners	30 September 2019
0.8	Updates for harmonisation and BC assessments	11 October 2019
0.9	Internal review version	21 October 2019
1.0	Final version	25 October 2019

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliverable presents the Integrated Methodology of the SAFIRE platform and comprises the integrated methodological approach as well as practical guidelines for customising the SAFIRE technologies and platform.

The methodology for the SAFIRE platform can be arranged in four groups of steps, focusing on BC analysis and scenario definition, BC customisation, platform integration, and platform testing and release. As first step, the business-specific requirements are being defined, and in the second step are being applied to the technical part of the platform, namely the SAFIRE services and modules. In the third step, all the modules configured for the business case are being connected to work together as an integrated system which is being tested and accordingly optimised in the business sites for the last step. The observation of the results from the platform released version is being observed using the SAFIRE monitoring dashboard.

Additionally, the SAFIRE methodology describes in detail the individual steps to adapt the four modules, namely the Predictive Analytics (PA), the Situation Determination (SD), the Optimisation Engine (OE) and the Security Framework (SPT). Some of the main aspects of the customisation needed on the modules are the data sources and samples for the PA, the situation model for SD, the optimisation metrics and fitness functions for OE, and the privacy rules for the SPT.

The integrated methodology aims to support the experts from a given industrial company (industry experts), to employ technical stuff (SAFIRE experts) able to customise the SAFIRE services for the selected business needs in order to interact with the industrial legacy systems. It aims to provide an easy to follow, stepwise, workflow to accompany all the interest-to-SAFIRE-solution-parties from the business conceptualisation and specification, to the implementation, testing and release of a business tailor SAFIRE platform.

1. INTRODUCTION

1.1 OVERVIEW AND DOCUMENT STRUCTURE

This document describes the methodology that needs to be followed to adapt, extend or configure the SAFIRE solution for a new installation, which requires a specific customisation for different business cases and scenarios. The steps needed for the setup of the four SAFIRE modules, namely the Situation Determination (SD), the Predictive Analytics (PA), the Optimisation Engine (OE) and the Security Framework (SPT), and their operation within the SAFIRE solution will be described in detail. Furthermore, the methodology to integrate the SAFIRE solution in the business case scenarios, as well information on the how the integrated solution interacts with the industrial environment is being presented.

The structure of the document is as follows:

- *Section 1:* Introductory description of the document purpose and the targeted audience.
- *Section 2:* Definition of the methodology for the SAFIRE platform.
- *Sections 3-7:* Description of the methodology followed for the adaptation and extension of the different SAFIRE modules, as well as of the algorithms applied to setup and customise the platform and its technologies to a business case.
- *Section 8:* Conclusion of the document and summary of the main points discussed on the previous sections.

1.2 DEFINITION AND SCOPE OF THE METHODOLOGY

1.2.1 Objectives and Scope

The objective of the SAFIRE methodology is to demonstrate the easiness of adaptability of the platform to different industrial environments, products, machines and processes, to achieve higher quality and precision with reduced costs and effort for the different stakeholders. Following this methodology, the SAFIRE platform can be adjusted, or extended, to support specific business cases and scenarios, and connect the different and complex legacy systems.

The SAFIRE solution consists of four main services, namely the Predictive Analytics (PA), the Situation Determination (SD), the Optimisation Engine (OE) and the Security Framework (SPT), and is being monitored by a dashboard, to allow the stakeholders in the industrial environment to observe its operation. The methodology starts with the common steps followed for integrating all services together, and continues with specific details for the individual services, and how those can be customised to a business case.

SAFIRE allows for observation of changes in circumstances in which a product/machine is used, and, accordingly for the processes, circumstances under which they are put into practice, which in turn allows for a dynamic adaptation of the involved aspects to these varying conditions, using history analytics and predictive algorithms, as well as, identifying and performing optimisations when needed. SD monitors the environment of operation of the solution and, as required, calls the OE service to calculate an optimised schedule based on the new environment information that were registered. In parallel, PA analyses the data that are being produced in the industrial environment, as well history data that were fed to the system, and identifies patterns of events that occur and might affect the operation of the system. Its output is then fed to the SD which compares and analyses again the situation data and suggests further actions or calls the OE. During the whole operation of the SAFIRE solution the SPT ensures the safety of communication between the modules, as well as acts as a shield for the SAFIRE and its external environment (i.e. the legacy systems).

1.2.2 Targeted Audience

The targeted audience of the present methodology is:

- **SAFIRE experts** who are responsible for the setup and interaction with the SAFIRE solution. SAFIRE experts will use the methodology to apply the guidelines for the adaptation of the data models and the customisation of the different services, to the selected business scenarios.
- **Industry experts** who will use the methodology to understand how to assist SAFIRE experts, with their business knowledge and expertise, towards the selection, or the fine tuning of the business scenarios, and the successful configuration and integration of the SAFIRE solution.
- **Technology providers** who will utilise the open source project technologies in manufacturing and smart product solutions to achieve improvements and optimisations in their commercial offerings.
- **Researchers / Academics** who will utilise the open source technologies to further the state-of-the-art in manufacturing and potentially contribute to the evolution of the open source technologies published by the project.

The information provided in this deliverable provides the practical information for each of these different audience based on the experienced gained from implementing the industrial Business Case (BC) demonstrators.

2. METHODOLOGY FOR SAFIRE PLATFORM

In the complex domain of Manufacturing, information about products, machines and processes can be crucial for the future improvement and achievement of innovation, covering the ever-increasing needs of the customers while maintaining low costs and time to market. SAFIRE aims to enhance the process of information management and (re)use and offer a medium for advanced big data analysis and situation-based process optimisation, easy to adapt in the manufacturing environment, and suitable for different business concepts. Towards this direction, the SAFIRE platform is composed by four service-modules, namely the Predictive Analytics (PA), the Situation Determination (SD), the Optimisation Engine (OE) and the Security Framework (SPT). An informational dashboard is also part of the SAFIRE solution, allowing the users to observe its operation inside the industrial environment.

The challenge in creating the SAFIRE platform lies in the difficulty on integrating the different functionalities into one platform able to be used as a plugin in any industrial environment, performing only simple configuration for individual scenarios. The following Figure 1 illustrates the structure of the methodology followed to adapt and integrate the SAFIRE services into one platform in the business cases.

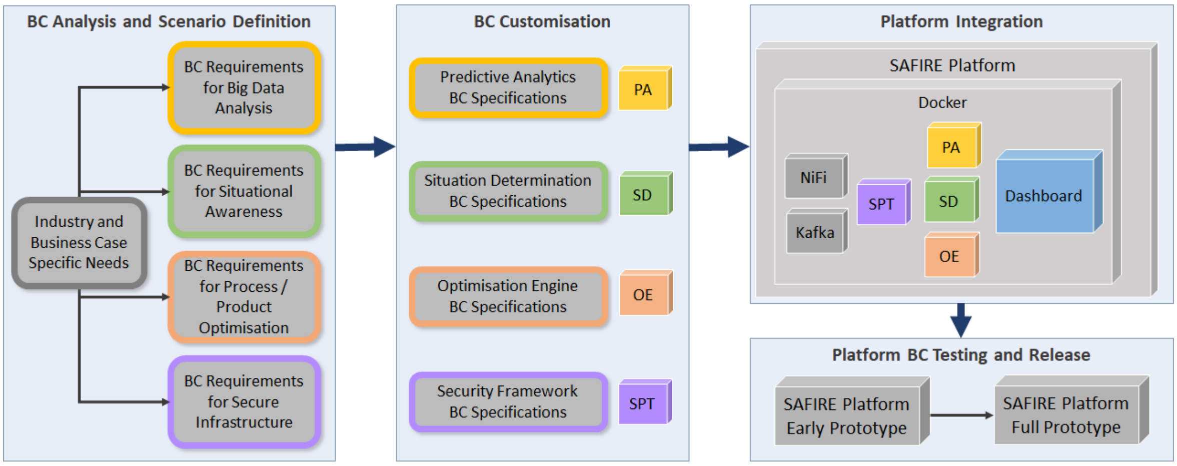

BC Analysis and Scenario Definition
Industry and Business Case Specific Needs
BC Requirements for Big Data Analysis
BC Requirements for Situational Awareness
BC Requirements for Process / Product Optimisation
BC Requirements for Secure Infrastructure
BC Customisation
Predictive Analytics BC Specifications
PA
Situation Determination BC Specifications
SD
Optimisation Engine BC Specifications
OE
Security Framework BC Specifications
SPT
Platform Integration
SAFIRE Platform
Docker
NiFi
Kafka
SPT
PA
SD
OE
Dashboard
Platform BC Testing and Release
SAFIRE Platform Early Prototype
SAFIRE Platform Full Prototype

Figure 1: Structure of the SAFIRE Platform Methodology

As shown in the figure above, the methodology followed for the SAFIRE platform can be structured under four steps:

- BC Analysis and Scenario Definition

  The first step on the SAFIRE methodology includes the research and analysis of the requirements that should be covered by the SAFIRE platform. Industrial and business goals, customer expectations and needs, as well as particularities of the selected business cases have to be collected, analysed and structured so as to lead to specifications for the platform to be developed. During this step, potential actors of the system to be enhanced by applying SAFIRE technologies could be revealed, leading likely to more clear or specialised requirements. Additionally, the specific requirements for the main features promised for the project, namely big data analysis, situational awareness, process optimisation and security in operation, should be identified. Those requirements should be compared and combined with those collected for the industry/business, to lead later on in the shaping of specifications, as well as to a more detailed scenario concept.

This step requires close cooperation of the industry experts with the SAFIRE experts and potential end users. As a practical example of an operation belonging to this step, for the development of the SAFIRE solution, the definition of the use case scenarios (e.g. concept, actors, workflows, etc.) from the business case experts (within the SAFIRE project BC experts are from the companies: Electrolux, OAS and ONA) in cooperation with the research partners, could be mentioned.

- BC Customisation

  As a second step to the SAFIRE methodology, the customisation of the different services based on the selected BC scenarios follows. The specifications, which have been already derived from the process in the previous step are being reviewed, if necessary, and decisions for the adjustment of the data models, the data exchange formats/files and the legacy connection modules will be made.

  Regarding the current development of the SAFIRE solution, an operation of this step is the definition of the situation models, and the data exchange formats to be communicated between the SAFIRE modules and the legacy systems (e.g. xls files, string json formats, etc.).

- Platform Integration

  As part of this step, the individual services are being connected to work together (full prototype of the services and early prototype of the integrated platform), exchanging data and pushing their results to the common communication channel (e.g. Kafka). Additionally, the integrated services are being adjusted (e.g. change input data source) to interact with the business case systems. Modifications can be made to the individual modules, as well as to the integrated part, in case any business case, or scenario, parameter should be adjusted (e.g. data exchange rate or amount of data traffic allowed). Part of this step is also the description of the integration process to documents that can also be used as user manuals for further development of the project or adjustment to different business cases.

  This step might reveal the necessity of additional services or modules to assist in the legacy system integration or in the observation of the platform operation. For this development of the platform, the SAFIRE dashboard was introduced to monitor the SAFIRE service-status.

  The result of this step is fully functional packaged (e.g. using Docker) modules (i.e. early prototypes) that are able to accept and process the selected data (including those coming directly from the business infrastructure).

- Platform Testing and Release

  This final step of the SAFIRE methodology describes the testing of the integrated platform (early prototype of the platform), as well the release of the final version. The first release of the platform is being validated in the industrial environment, and the operation of the whole system is being observed using the SAFIRE dashboard. The tests might reveal conflicts with the legacy systems (e.g. discrepancy in the security policies, or connection issues due to the configured data exchange options), or with the internal communication of the modules in real environment (e.g. delay in result production of a module due to hardware limitations). Those conflicts will be adjusted to the final version of the platform (in case those require platform adjustments), or during the installation process (in case those are due to configuration options).

The steps mentioned above, described above present the common methodology to create the generic SAFIRE solution. The next sections describe the individual methodologies for the customisation of each SAFIRE services (Predictive Analytics, Situation Determination, Optimisation Engine and Security Framework), and give a more specialised insight in their structure.

3. METHODOLOGY FOR PREDICTIVE ANALYTICS

The Predictive Analytics Platform allows doing advanced analytics in real time, storing huge amounts of data and web visualization tools to easily query and visualize the stored data. Moreover, the Predictive Analytics Platform offers different web services for interacting with different modules.

The Predictive Analytics module is related to the following steps. Every step on the list must be taken into account for the system design and at the time of adding additional business cases.

- Define goals: define the goals in the business case, datasets involved, performance, etc.
- Data collection: select mechanisms to collect needed data. Define storage needs.
- Data Analysis: cleaning, filtering, transformation of collected data. Several statistical techniques can be applied to explore relations, correlations, etc.
- Predictive Analytics: functions to create, train and test models that can be used to predict values from past sets of data.
- Visualization: visualization types needed to present results to the user.

Depending on the needed data processing power, the platform can consist of one or several computing nodes automatically instantiated and configured. The predictive analytics module is composed of three general sub-modules (Processing Engine, Storage, Visualization), with well separated responsibilities. As the computing power needs of each sub-component can vary greatly, every sub-module can be executed in different machines and scale independently.

The proposed approach to predictive analytics in the SAFIRE solution is summarised in Figure 2.

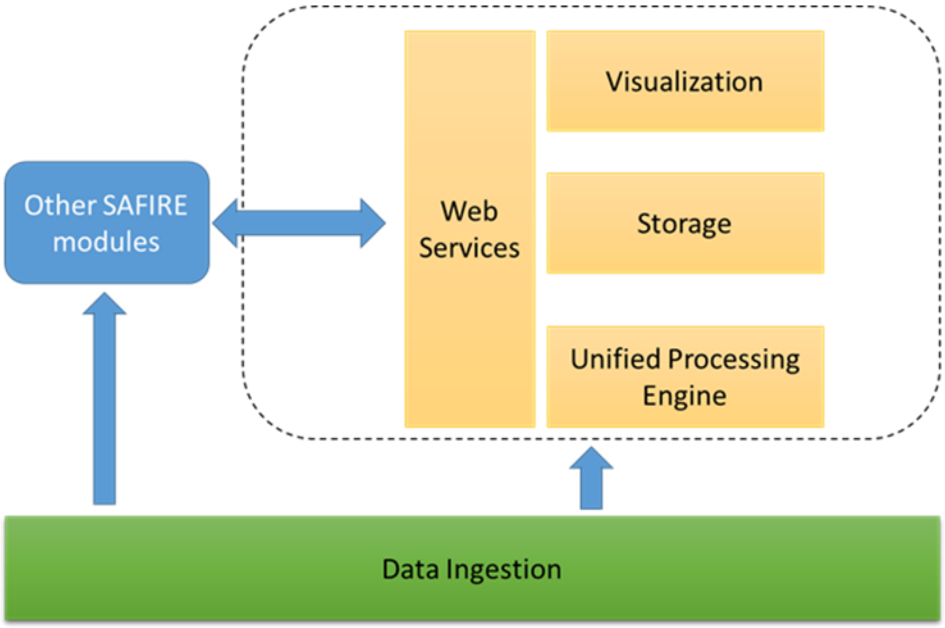


Figure 2: Approach for Predictive Analytics within SAFIRE

The following sections present the general and adaptable platform supporting the Predictive Analytics module, the decomposition in sub-modules and some guidelines useful to adapt them to the different business cases. Additionally, the algorithms, technologies and tools used are described. The backbone of the sub-modules is composed of open source components, carefully selected by being some of the current components with good quality and well-known performance. Their configuration capabilities are in line with the need for easy adaptability to new business cases. The technical detail of every software component used is defined in deliverable D2.3.

3.1 PLATFORM SUPPORT

There are several software components in the solution that can be combined and can profit of several machines, scalability, etc. The solution proposed uses those capabilities, trying to isolate the end user from the technical complexities.

Depending on the use case, more or less computers can be used, configurations can be changed, or the deployment can be on-cloud or on-premises. In the general case, manual configuration and deployment would be a daunting task. The chosen approach is the use of Infrastructure as Code (IaC). The configuration of the platform is defined in an easy and well-defined syntax, parsed, validated and executed by tools.

In the proposed solution, the deployment is done using Terraform. The number and characteristics of the machines deployed in Amazon Web Services (AWS) is defined via some parameters. The specific networking and permissions are also defined and configured by Terraform.

When the machine instantiation is finished, every machine must be provisioned (software installed and configured). To do that, a similar method is used: a text file can be written to define the configuration tasks to be executed in every machine. This source file is parsed and executed by Ansible, a tool that checks the configuration in every machine and executes the tasks defined to arrive to the desired state.

With this IaC approach, deployment and configuration of several machines can be performed automatically in a reproducible way, avoiding the common mistakes in manual processes. Changes in configuration can be tested and deployed easily. Depending on the new business case's needs scripts can be adjusted accordingly to increase the size of the computer cluster.

3.2 Methodology for the Unified Process Engine subcomponent

3.2.1 Overview

The Unified Processing Engine provides support for doing advanced analytics on both real-time and batch approaches.

In order to enable predictive analytics in SAFIRE, the Unified Process Engine subcomponent should have access to relevant business case data. Any analytics processing with data that is incorrect or unrelated to the business case goals would be useless.

The subcomponent should be prepared to receive different kinds of data, from different sources and a wide range of ingestion rates. In order to incorporate a new business case, an initial study of the business case needs should be performed. This study about the goals and data characteristics of the business case is essential for the success of the whole process and enough time and resources should be allocated to it.

3.2.2 Guidelines

Before the customisation of the Unified Process Engine, and its configuration for a business case, some steps should be considered:

1. **Assessment of available data and definition of expected result.** The first step when applying the predictive analytics platform to a new use case is to assess all the data that could be made available for the predictive analytics module, not just the data that will actually be used. With the available data in mind, a clear business goal or goals must be set. This process is iterative in nature until a realistic goal is found based on the available data. SAFIRE experts and business case experts should be involved in the process.

2. **Data selection.** Select the data related to the defined goals. Some of the previously considered data could be irrelevant for the business case aims.

3. **Pre-processing**. Some preliminary treatment of data needs to be performed in order to clean the data (remove unnecessary data, treat noisy data, standardize values, handle missing values…).

4. **Data transformation and analytics**. Depending on the predictive goals and the available date, some algorithms are better suited than others for a specific analytics problem. Once a SAFIRE expert has a good understanding of the problem at hand, the expert will be able to present a subset of algorithms to test individually and compare results among the different possibilities. Most predictive algorithms impose restrictions on the characteristics of the data can handle, and such restrictions are particular to each algorithm. In order to find the best suited algorithm for the problem in hand this step has to be performed iteratively, adjusting the data to the next algorithm to be tested and then carrying out a benchmark experiment to assess the performance of the algorithm to the problem at hand and the selected data.

This process is iterative. The results of some step could suggest changes in the decisions from previous steps. Once the process has been set, it is ready to be deployed into the Unified Process Engine.

The platform will require some minor changes to adjust to the particularity of the new business case. Figure 3 shows the internal subcomponents of the Unified Process Engine. The main driving component is the Predictive Agent which remains unchanged from business case to business case. The other subcomponents, however, need to be adjusted to the selected data and analytics.

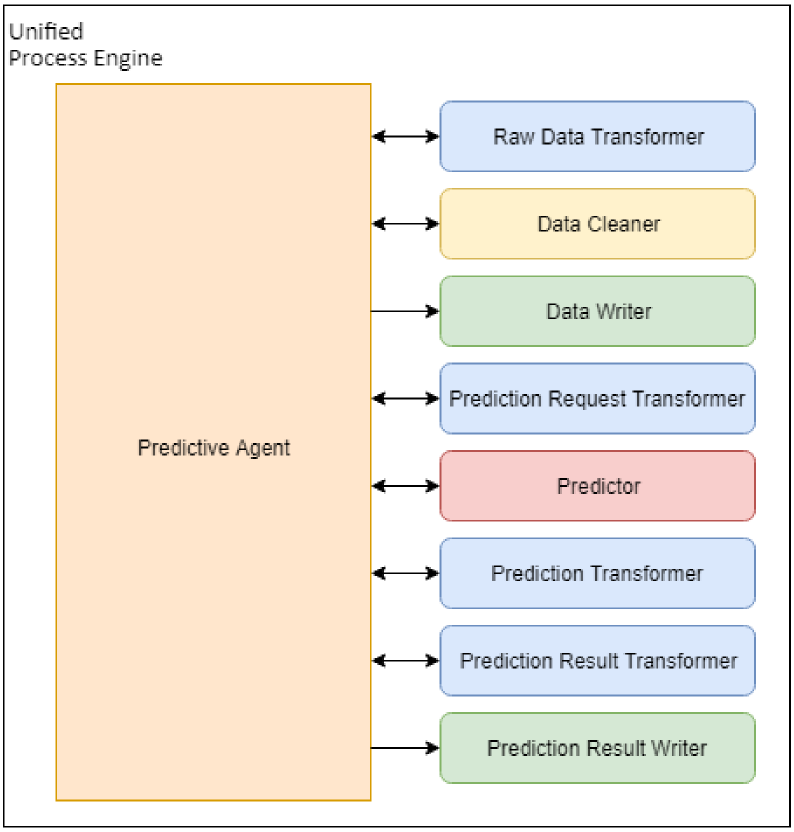


Figure 3: Sub-components of the Unified Process Engine

- The transformers, marked in blue, transform the messages received from and sent to outside the Unified Process Engine. The communication is performed using publisher/subscriber technologies and sending serialized JSON objects from publisher to subscriber. Some elements of these serialized messages are common and known, however most of them are dependent on data selected for the business case. These transformers make transformations between the JSON objects of the Kafka messages and datasets.

  - **Raw Data Transformer** receives the historic and current streaming data that will be used to train and update the predictive models. For a new business class this needs to be adjusted to be able to handle different structures of data and create datasets with different numbers and types of columns. The data handled by this transformer needs to be tagged with the known value for the variable to be predicted.

  - **Prediction Request Transformer** receives new prediction requests. For a new business case this needs to be adjusted to be able to handle different structures of data and create rows with different numbers and types of columns. The data handled by this transformer will not be tagged with the predicted variable since that is the expected goal of the request. However, these messages will need to have a traceable variable that, while not used to make predictions, is needed to trace prediction requests and variables. This traceable value does not need to follow a specific format, but its value should be unique, so a UUID-like string value is recommended.

  - **Prediction Transformer** transforms the results of the prediction so that it can be sent back to the module that made the request. For new business cases this needs to be adjusted for potentially different result types (a numeric value for regression problems, a class and optionally class probabilities for classification problems).

  - **Prediction Result Transformer** handles responses to whether a previously made prediction was correct or not. As with the rest of transformers it will need to be adjusted to the specific result type of the prediction. This transformer is optional because it is not needed for the correct functioning of the module. However, receiving feedback about the accuracy of made prediction helps with keeping models accurate and up to date.

- The **Data Cleaner** subcomponent, marked in yellow, is in charge of cleaning the data to first, get rid of any unnecessary data, and adjust the remaining data so that it conforms the restrictions of the algorithm to be used. It is in charge of performing the data cleaning and adjusting tasks of steps 3 and 4 defined at the beginning of this section, and such, for a new business case it will need to be adjusted to perform the data cleaning and transformation tasks defined in those steps.

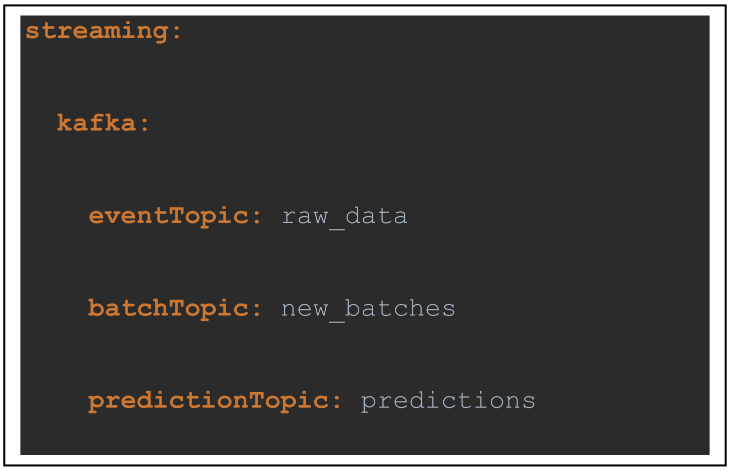
- The **Predictor** sub-component is in charge of making the predictions. It is a very simple component where the only adjustment needed is to implement the use of the predictive algorithm selected for the problem at hand.

- The writer components take the data transformed by the different sub-components of the Unified Processing Engine and persist in the Storage sub-module of the predictive analytics module. For a new business-case adjustments need to be made for the specific storage solutions and data structures.

  - **Data Writer** receives the raw data to be used to train the predictive models.

  - **Prediction Result Writer** is optional like Prediction Result Transformer and works in conjunction with it. It stores prediction results to further improve the predictive models. The changes needed to this writer are minimal compared to the Data Writer.

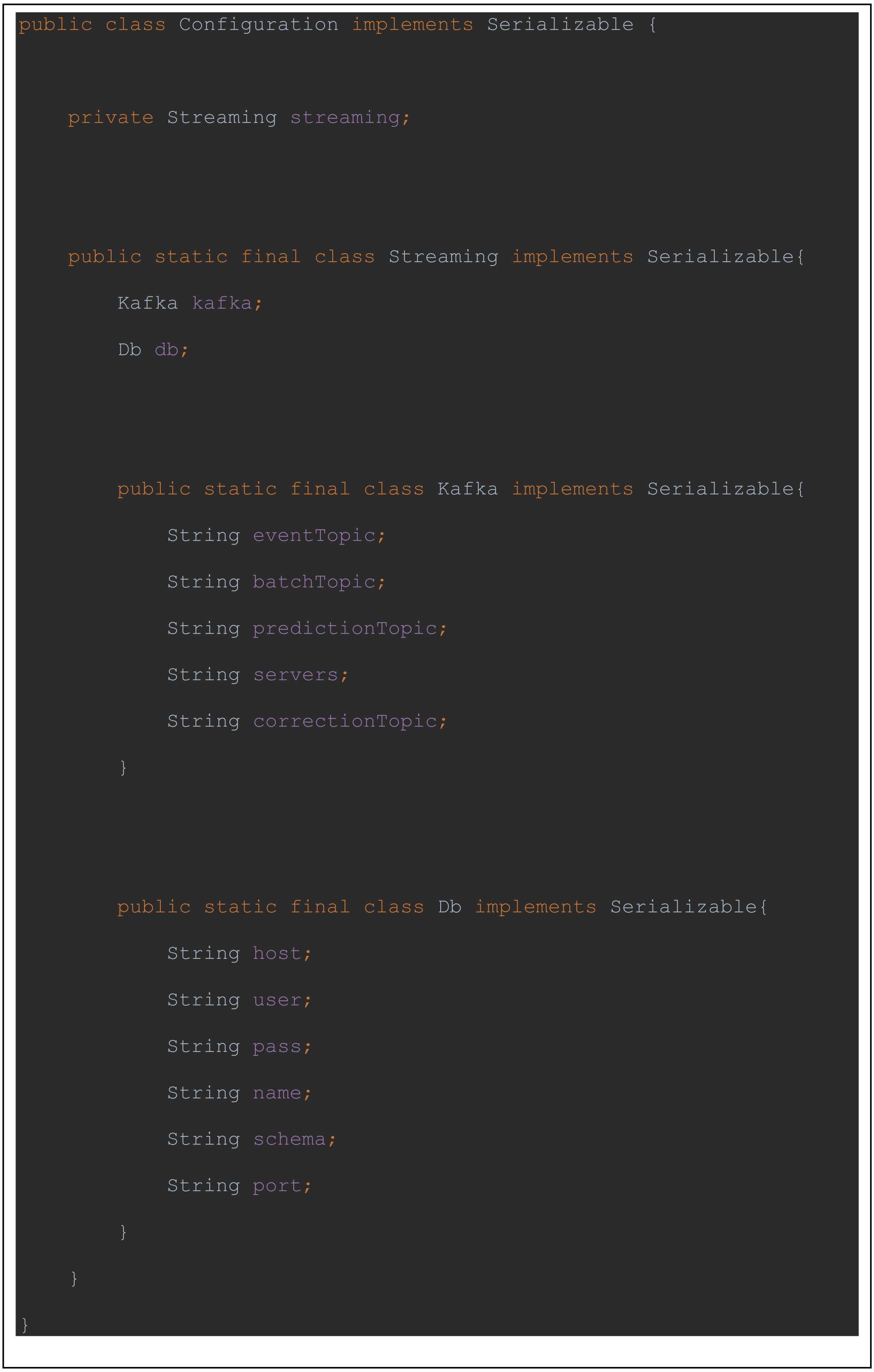
Aside from the code modifications described in this section, configuration changed will need to be made as well to adjust to the new use case. The following portion of code shows the configuration file in YAML format (config.yaml) of the OAS use case of the SAFIRE project. YAML files contain a set of key-value entries, similar to JSON files, structured hierarchically using indentation to set the level of each entry. Due to their simple viewing style and low overhead, for configuration purposes this file type is preferred for configuration files compared to others like JSON or XML.



```
streaming:
  kafka:
    eventTopic: raw_data
    batchTopic: new_batches
    predictionTopic: predictions
```

Figure 4: Configuration file example

The structure of the configuration file for the Unified Process Engine is not fixed and will need to be adjusted to the use case in particular.



```java
public class Configuration implements Serializable {

    private Streaming streaming;


    public static final class Streaming implements Serializable{
        Kafka kafka;
        Db db;


        public static final class Kafka implements Serializable{
            String eventTopic;
            String batchTopic;
            String predictionTopic;
            String servers;
            String correctionTopic;
        }


        public static final class Db implements Serializable{
            String host;
            String user;
            String pass;
            String name;
            String schema;
            String port;
        }
    }
}
```

Figure 5: Configuration code example

This configuration YAML file can then be easily mapped to an object with the exact same structure as the file by using a few lines of code.
The config .yaml file in Figure 4 and the Java class in Figure 5 have the exact same structure. Thus, the values from the file will be mapped into this object at runtime providing easy to use access for all of the entries.
For a new use case, both the configuration file and the containing class will need to be adjusted to reflect the particularities of the use case. The new Java file should match the new configuration file exactly and vice versa.

3.2.3 Algorithms, Technologies and Tools

Apache Zeppelin is a web notebook maintained by the Apache Foundation. Web notebooks work in a similar manner to console shells, interactively running the commands provided by the user, instead of having to write all of the code and compile the project beforehand. Running in a web browser and being able to make use of rich web components instead of a simple command line interface adds great value to this technology. Web notebooks also have the inherent advantage of being able to be used remotely from the user's local machine.

Zeppelin is specially oriented to data analytics and visualizations, which makes it a perfect candidate to be used in SAFIRE implementations.

In order to adjust the Unified Process Engine to new cases, the analytics aspect of Zeppelin is of great use. Once set-up, this tool will allow SAFIRE experts to quickly test different analytic and predictive algorithms over the data of the new business case, without the need of building a complex framework even before the true work begins.

Once the algorithms and pre-processing to be used have been defined, the production process should be implemented in Apache Spark. The main technology behind the process engine is Apache Spark. This framework allows seamlessly running applications across multiple computers networked together. It works on top of a distributed file type system by which all computers running together share a common file system even some files are physically located in as little as one of the computers. From a coding point of view, it does not matter how many computers are networked together, the code is the same. It works by applying the MapReduce methodology where a central node works as master, coordinating the rest of nodes or workers, dividing the work among them and collecting their results.

Apache Spark is able to handle processes that work in batches where a defined chunk of data is processed, but also streaming processes where the flow of data is continuous, and data is handled as it comes.

SAFIRE implementations will make heavy use of two subcomponents of Apache Spark: SparkSQL and MLlib, Spark's Machine Learning library. SparkSQL provides a SQL-like language and set of implemented methods to handle and manipulate data. Thanks to the latest developments within Spark, nowadays it is possible to use virtually the same code to run batch and streaming works. The MLlib library provides a set of data processing and machine learning algorithms to perform predictive analytics on the data.

However, Apache Spark is still a growing technology. Each new version adds new technologies and algorithms, but it is possible some of the necessities might not be covered by the framework. One such case in SAFIRE implementations is the use of neural networks for regression tasks. Apache Spark currently only allows using neural networks for classification, not regression. In this case we can use the Tensorflow library developed by Google, through the Keras library.

Apache Zeppelin combines perfectly with the technologies used in the Unified Process Engine as it is able to connect to Apache Spark and Keras using Scala and Python code.

3.3 METHODOLOGY FOR THE STORAGE SUBCOMPONENT

3.3.1 Overview

The Storage subcomponent is the responsible for the data storage (raw data, processed data, predicted values), in different places. Data can be fed directly from the data ingestion system (Apache NiFi) or, depending on the needed data pre-processing and cleaning, can be published to the Apache Kafka broker so it can go through the Unified Process Engine. Figure 6 shows the different data flows that lead to the storage sub-component.

Different business cases have different data storage needs. The data storage subcomponent can store data in relational databases or NoSQL databases.

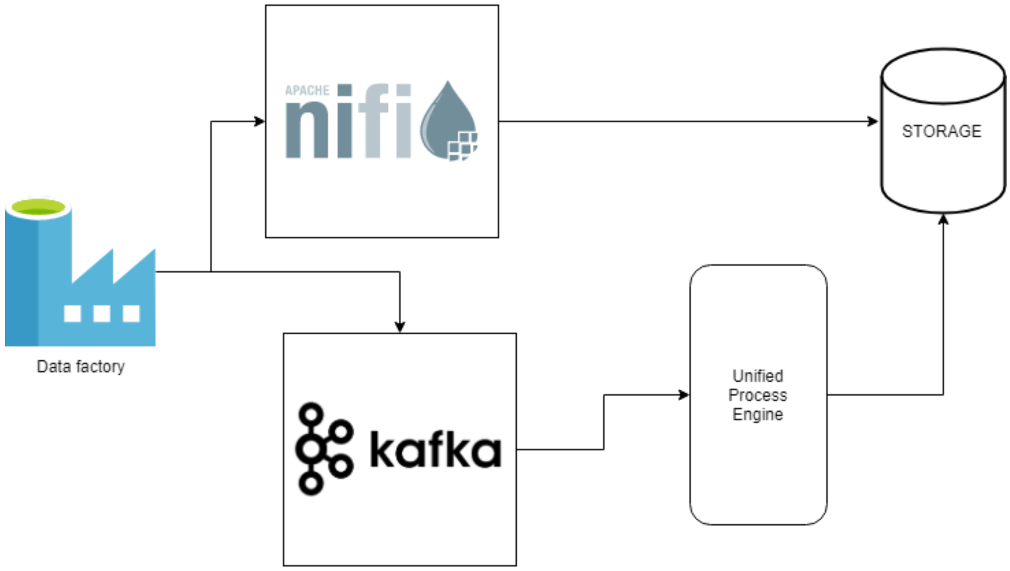


Figure 6: Data flow to Storage sub-component

3.3.2 Guidelines

When applying the SAFIRE methodology to a new business case, the storage subcomponent of the predictive analytics platform will also need to be adjusted. The guidelines in this section are not as specific as those for the Unified Process Engine, since there is some subjectivity in the decisions and adaptations in the storage sub-module are much more heavily influenced by the business case's particular needs and the people involved in the process.

The first decision that needs to be made is how the data will be stored. This decision is actually made up of several smaller decisions, but the most relevant is if data will be stored in relational format or non-relational format.

Traditionally, the most-widely used databases have been relational databases. In relational databases, each entity of data is stored in a table. The structure of these tables, or in other words, the schema of the database, needs to be defined before data can be entered into it. Tables are made of multiple fields referred to as columns, and some columns, act as relationships between different tables. Relational databases aim to reduce the redundancy of data by containing big pieces of data (portions of text, for example) In only the most relevant table, and then linking this big piece of data to any needed entity by using smaller, more storage-wise efficient, numeric fields for cross-reference. These numeric references can then be used to create single queries that can span across different entities (tables) and create a single return value of all the needed information. The relationships between tables can also be used to define integrity restrictions the data has to conform to. For example restricting the values a field can have, or ensuring that relationships between entities are enforced when modifying the data. Relational datasets are also referred to SQL-databases because all of them use the standard query language (SQL) to create, modify and access data. Figure 7 shows the example of a simple relational database.

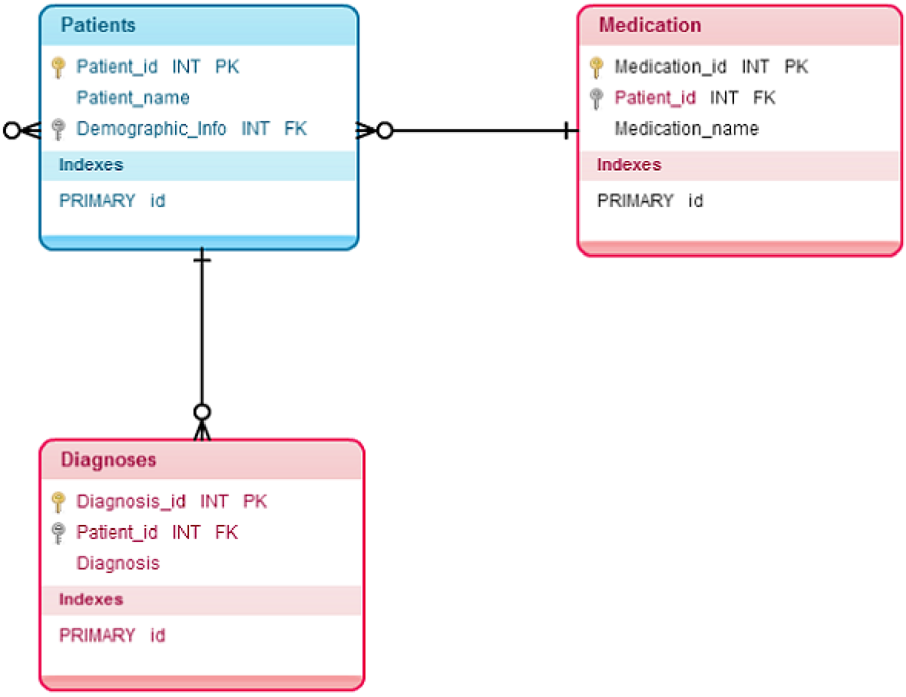


Figure 7: Example of relational database

As of late, new database paradigms have emerged under the "NoSQL" umbrella. Their common thread has nothing do with the SQL language. In reality, their commonality is that data is not stored in entities related by fields, they do not have relations between entities, and so, are not relational. Two of the most typical types of NoSQL database systems are document-oriented databases and key-value entry type datasets. Document oriented databases store entries in JSON-like documents, where each document contains several fields where the value can be a number, a piece of text, blobs of data or even other sub-objects. The first most obvious benefit of this type of databases is that there is no need to define a prior structure for the data. In fact, each entry of the data can have a new and different structure and the database system will be able to work around that characteristic. This results in a much smaller set-up time and freedom to add new information to each entry later on.

Throughout this section stored data will be considered of one of three different types:

- raw data
- processed data
- and result data

It should be noted that these three types do not need to be stored using the same database type. For starters each organization will have a preferred way to store the data. Also, each new business case will have different needs of how the data is produced and how the data should be stored.

For the initial raw data, taken straight from the source with little or no pre-processing, even if it is not a rule set in stone, at least as a first step, raw unfiltered data should be stored in a similar format to how it is created in order to avoid spending computing power on data that could be discarded on a later step.

As long as the hardware-software infrastructure allows it, as much data as possible should be stored. This allows both opening new exploration avenues later on with data that was not considered earlier, and making potential corrections to mistakes done during the analytics process by avoiding to having to discard data because of a discovery of needing more data than considered at the beginning. Redundant data and pieces of information to little or no value can be discarded, as long as both the domain and SAFIRE experts agree on the uselessness of the information. If runtime performance is a concern, raw data can be kept on a separate server to avoid interference with runtime data-related operations.

The next type of data is processed data. Periodically, the raw data is analysed, treated and converted into a format that can readily be used by the predictive algorithm of the unified process engine. The format and storage solution of this data will be heavily influenced by the predictive analytics selected for the process engine.

Retention policies need to be defined for the processed data. A big, always increasing database system can suffer from poor performance. In a traditional batch processing context this could simply mean waiting a little longer for the processing to be finished, but in a streaming context, this can result in processing taking an unacceptable waiting time and ever-increasing delays in the analytics process. Retention policies define how much data and for how long it is stored. The size of the retained data will depend heavily on the particular needs of the business case and the computing capabilities available. In some cases, due to the analytics process, discarding data could be unaffordable because as much data as possible is needed. In these cases, the possibility of creating different predicative models should be considered. For example, for a photovoltaic plant context, the season the data is taken is of key importance and different models can be created for each season. Processed data for each season and their predictive models could be kept and trained in different servers. Discarded data could be transferred to another independent system where it does not interfere with the analytics for potential new uses of the processed data. This backup of the processed data is optional since, if the raw data is always kept, it will always be possible to recreate the processed data.

The final type of data, result data, is mostly kept for internal purposes. Each time a prediction is made, the parameters that led to this prediction and the prediction itself should be stored. Later, the user of the prediction could update the predictive analytics module on the usefulness of the prediction, by telling whether the prediction was accurate or not. This information can be used for reporting purposes and even as feedback for the predictive model, to increase its accuracy in further predictions.

The intuitive way of storing this information is in a document format with a document per prediction however depending on the expertise of the individuals working on the reporting system other possibilities could be considered, such as using relational database tables as document holders, since in this case, the structure of each entry should be the same.

3.3.3 Algorithms, Technologies and Tools

Each new use case will have certain needs that will be covered by some storage solutions and not others. NoSQL databases are quite diverse and the tools and technologies to manage them vary greatly. Relational database systems on the other hand, while offering a wide set of different tools and possibilities, do have a degree of consistency on how they store and handle data, as in the end they all conform to the SQL standard.

Usually, businesses use enterprise-grade relational database systems such as Oracle Database and Microsoft SQL Server, which are proprietary products developed by corporations. On the other hand, there also are free and open source solutions that are fully secure and trustworthy solutions, thanks to the vast communities developing and auditing their source code. Open source solutions can cater to the needs of a wide range of organizations. PostgreSQL is a very robust and advanced solution. Other solutions such as MariaDB (formerly MySQL) and sqlite provide more lightweight solutions that can be set-up with a much lower effort if the requirements are lesser.

While some relational database systems include their own development and data management tools, there are also multiplatform SQL client tools, such as Dbeaver and Toad, that offer a common interface for different types of database technologies. These tools make the behind-the-scenes SQL technology transparent for most functionalities.

When interfacing SQL databases with the Unified Process Engine, code libraries can be used to ease the manipulation of the data by the engine. ORM (Object-Relational Mapping) technologies, such as ORMlite and JPA, bridge relational databases and objects. By means of these tools, each row of the database is represented by an object (such as a Java object) that has the same structure as the database table the entry belongs to. These objects can be manipulated as if they were simple objects, unrelated to a database, and changes can then be mapped from the object to the database. Some of these technologies even offer the ability of making changes to the database in real-time. If a value in an object changes, the related entry of the database is automatically modified to reflect the change.

NoSQL databases offer a wider variety of features and possibilities because NoSQL is simply an umbrella term that covers all non-relational technologies. NoSQL databases can store data oriented to documents (MongoDB, Couchbase, CouchDB, DynamoDB), key-value entries (Redis, Casandra) or even multi-nature data.

3.4 METHODOLOGY FOR THE VISUALIZATION SUBCOMPONENT

3.4.1 Overview

The Visualization subcomponent is the responsible for the display of data to the user (raw data, processed data, statistics, predicted values, etc.) in a variety of visualization styles (tables, graphs, etc.).

The Visualization subcomponent is defined so that the user can select one or several datasets and choose some aspects to visualize (raw data, filtered data, some statistical values, predicted values...). The selection of datasets and aspects to visualize will be performed in an easy way, selecting from a list of options offered by the system.

3.4.2 Guidelines

The visualization of the stored data is fully dependent on the business case thus there is no silver bullet that can be used in every new use case. In this section a few guidelines will be provided to configure and use Apache Superset, a highly configurable tool for data visualization, where almost everything is configured via web interface.

Superset Dashboards are collections of Charts. Charts need a Table to draw data from and Tables need databases or datasources. Thus, once Apache Superset has been setup, the first step is to define datasources. Apache Superset is specially oriented to OLAP-like data storage solutions. However, it can work perfectly with the simplest relational database systems and supports SQL queries to access the data.

Datasources are created by first clicking the *Sources* option at the top menu and then clicking *Databases*. This leads the use to the existing database list, which should be empty at first. New databases are created by clicking the green round button at the top right corner of the database list. Figure 8 shows the web interface used to create new datasources. The mandatory fields are the name which is the identifier used throughout Superset to refer to this database and the connection URI conforming to the following format:

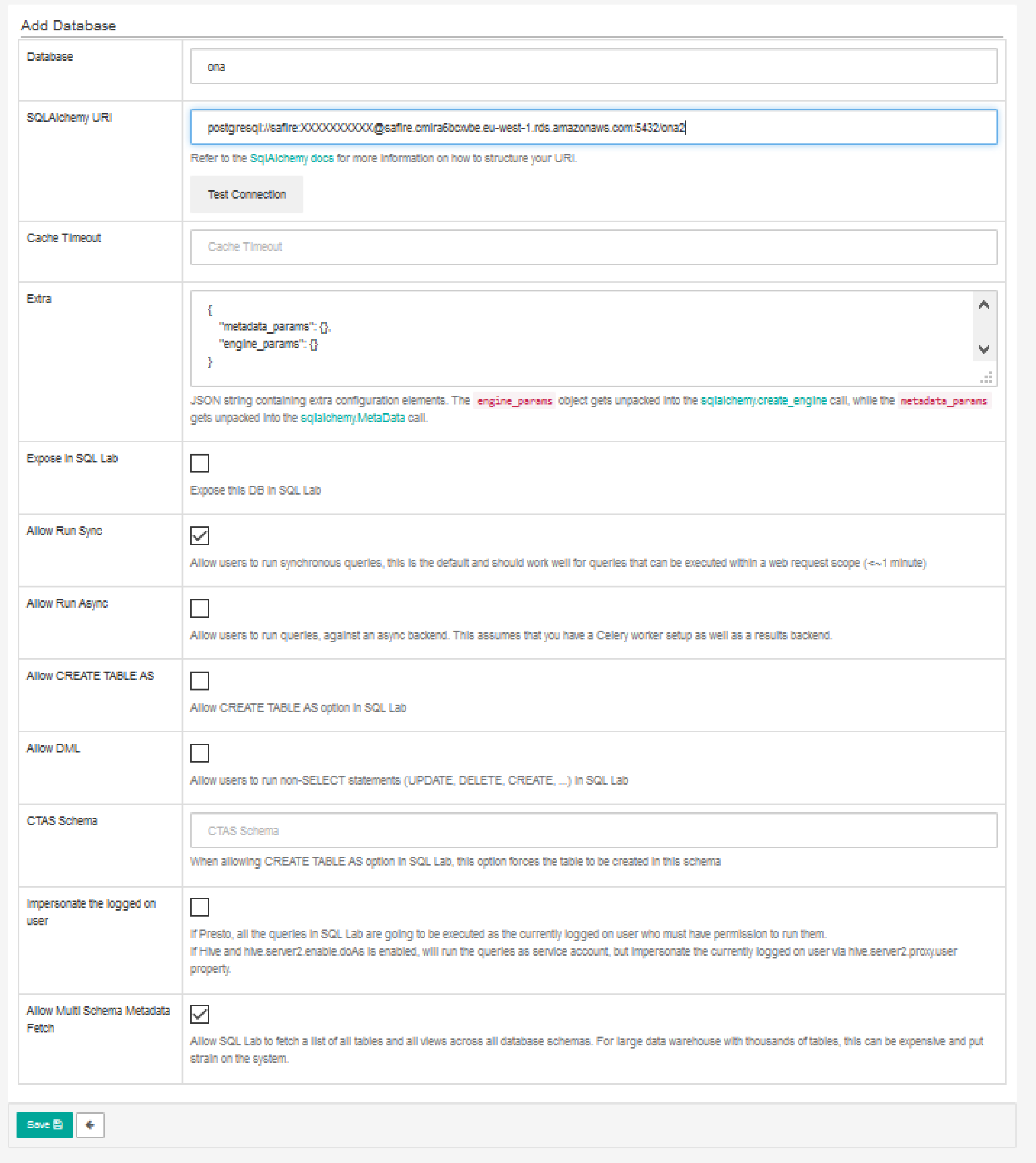
```
dbtype://username:password@databasehost:databaseport/databasename
```

The connection can be tested clicking the button just below the URI to test that all parameters are correct, and the server is reachable with the provided credentials.

With datasources defined, tables can be created. Tables can either be literal database tables already present in the datasource or can be newly defined tables based on a query to the real database, akin to views in relational database systems. Tables are defined using the *Tables* option of the *Sources* menu item at the top. The interface shown on Figure 9 is used to define tables. If the table already exists in the database, the only needed fields are the name (written exactly as in the database) and the database name which is a dropdown menu with all available options from the previously seen database list. If the table is defined, as if it were a view, the name can be made up but then a SQL sentence capable of creating the table must be provided.

After tables are created, the next step is to create charts using them. Charts are created by clicking the *Charts* option in the top menu and then clicking the green round button at the top right corner of the list. This will present the interface shown on Figure 10. This interface is mostly a web-based point-and-click interface that most of the time will not need any SQL knowledge.

The left-side menu allows the use to first select the data source (one of the Tables defined earlier) and chart type. Not all chart types are available to every type of data source, so for each new business case the available chart types will vary. Figure 11 shows an example of available chart types in Apache Superset. If, for the new use case, OLAP-oriented visualizations are used, it is highly recommended that the Storage-subcomponent either have an OLAP-based data warehouse as main storage, or create a basic data warehouse to store the processed data and act as middle man between the main storage and the visualization subcomponent, to avoid performing heavy queries periodically each time the visualization interface changes or is updated.



Add Database

Database	ona
SQLAlchemy URI	postgresql://safire:XXXXXXXXXX@safire.cmlra6bcwbe.eu-west-1.rds.amazonaws.com:5432/ona2 Refer to the SqlAlchemy docs for more information on how to structure your URI. Test Connection
Cache Timeout	Cache Timeout

Extra	{
	"metadata_params": {},
	"engine_params": {}
	}
	JSON string containing extra configuration elements. The `engine_params` object gets unpacked into the sqlalchemy.create_engine call, while the `metadata_params` gets unpacked into the sqlalchemy.MetaData call.

Expose in SQL Lab	□ Expose this DB in SQL Lab
Allow Run Sync	☑ Allow users to run synchronous queries, this is the default and should work well for queries that can be executed within a web request scope (<~1 minute)
Allow Run Async	□ Allow users to run queries, against an async backend. This assumes that you have a Celery worker setup as well as a results backend.
Allow CREATE TABLE AS	□ Allow CREATE TABLE AS option in SQL Lab
Allow DML	□ Allow users to run non-SELECT statements (UPDATE, DELETE, CREATE, ...) in SQL Lab
CTAS Schema	CTAS Schema When allowing CREATE TABLE AS option in SQL Lab, this option forces the table to be created in this schema
Impersonate the logged on user	□ If Presto, all the queries in SQL Lab are going to be executed as the currently logged on user who must have permission to run them. If Hive and hive.server2.enable.doAs is enabled, will run the queries as service account, but impersonate the currently logged on user via hive.server2.proxy.user property.
Allow Multi Schema Metadata Fetch	☑ Allow SQL Lab to fetch a list of all tables and all views across all database schemas. For large data warehouse with thousands of tables, this can be expensive and put strain on the system.

Figure 8: Database creation interface of Superset

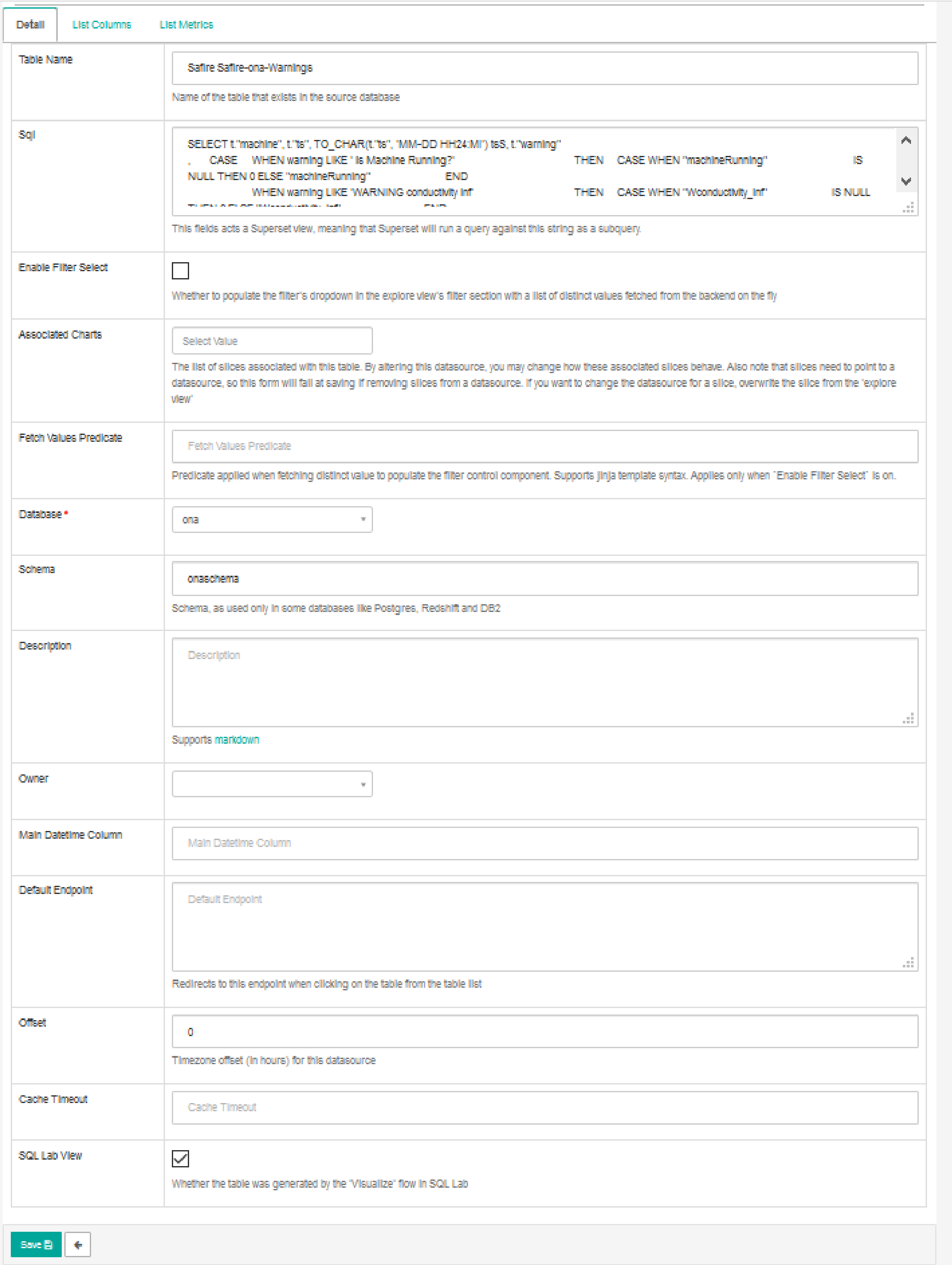


Table Name: Safire Safire-ona-Warnings

Name of the table that exists in the source database

Sql:

```
SELECT t."machine", t."ts", TO_CHAR(t."ts", 'MM-DD HH24:MI') tsS, t."warning"
,     CASE     WHEN warning LIKE ' Is Machine Running?'                 THEN    CASE WHEN "machineRunning"              IS
NULL THEN 0 ELSE "machineRunning"                  END
               WHEN warning LIKE 'WARNING conductivity Inf'             THEN    CASE WHEN "Wconductivity_Inf"           IS NULL
```

This fields acts a Superset view, meaning that Superset will run a query against this string as a subquery.

Enable Filter Select: □

Whether to populate the filter's dropdown in the explore view's filter section with a list of distinct values fetched from the backend on the fly

Associated Charts: Select Value

The list of slices associated with this table. By altering this datasource, you may change how these associated slices behave. Also note that slices need to point to a datasource, so this form will fail at saving if removing slices from a datasource. If you want to change the datasource for a slice, overwrite the slice from the 'explore view'

Fetch Values Predicate: Fetch Values Predicate

Predicate applied when fetching distinct value to populate the filter control component. Supports jinja template syntax. Applies only when `Enable Filter Select` is on.

Database *: ona

Schema: onaschema

Schema, as used only in some databases like Postgres, Redshift and DB2

Description: Description

Supports markdown

Owner:

Main Datetime Column: Main Datetime Column

Default Endpoint: Default Endpoint

Redirects to this endpoint when clicking on the table from the table list

Offset: 0

Timezone offset (in hours) for this datasource

Cache Timeout: Cache Timeout

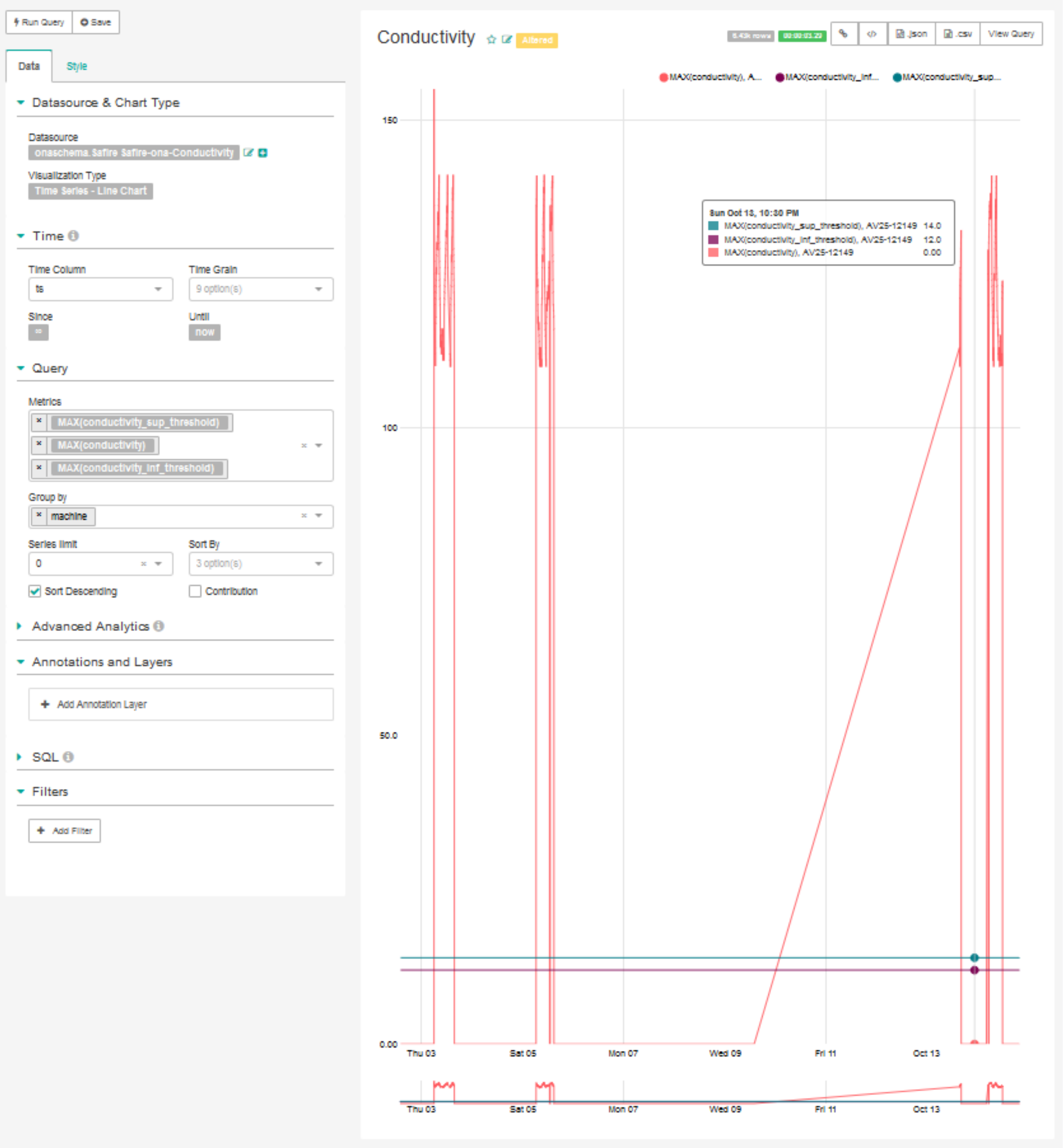
SQL Lab View: ☑

Whether the table was generated by the 'Visualize' flow in SQL Lab

Figure 9: Table creation interface of Superset

After defining the datasource and chart type, the table's fields have to be handled. The following options are dependent on the chart type. The charts shown in these figures are of time-series nature, so one of the dimensions is the timestamp of the entry. Due to the nature of the chart, this timestamp and the time range can be selected in the next portion of the interface. The next portion of the chart is common for all types and allows selecting the metrics that will be shown in the chart. Each of this metrics is formed by a column, and if necessary, an aggregator for this column (max, min, count, avg…). Finally, columns to group by to can be selected (necessary if aggregations are used) and the limit of the query response can be set.

The next portion of the menu is optional and allows to hardcode a specific WHERE clause using SQL in case a fine-grained query is needed that cannot be set using the interface options (e.g. subselects).



Datasource & Chart Type

Datasource: onaschema.Safire Safire-ona-Conductivity

Visualization Type: Time Series - Line Chart

Time

Time Column: ts

Time Grain: 9 option(s)

Since: ∞

Until: now

Query

Metrics: MAX(conductivity_sup_threshold), MAX(conductivity), MAX(conductivity_inf_threshold)

Group by: machine

Series limit: 0

Sort By: 3 option(s)

☑ Sort Descending

□ Contribution

Sun Oct 13, 10:30 PM
MAX(conductivity_sup_threshold), AV25-12149 14.0
MAX(conductivity_inf_threshold), AV25-12149 12.0
MAX(conductivity), AV25-12149 0.00

Figure 10: Chart creation interface of Supertset

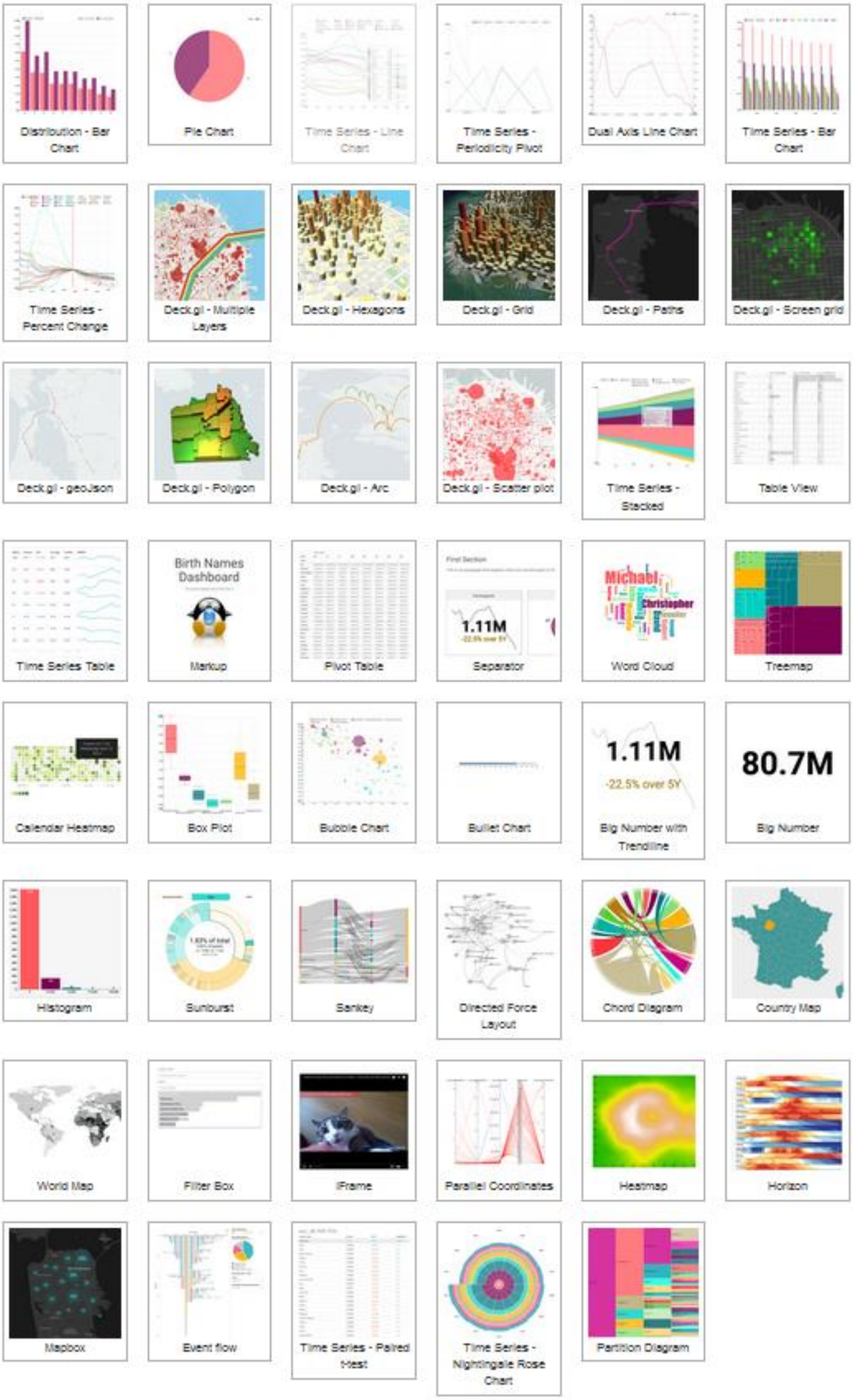
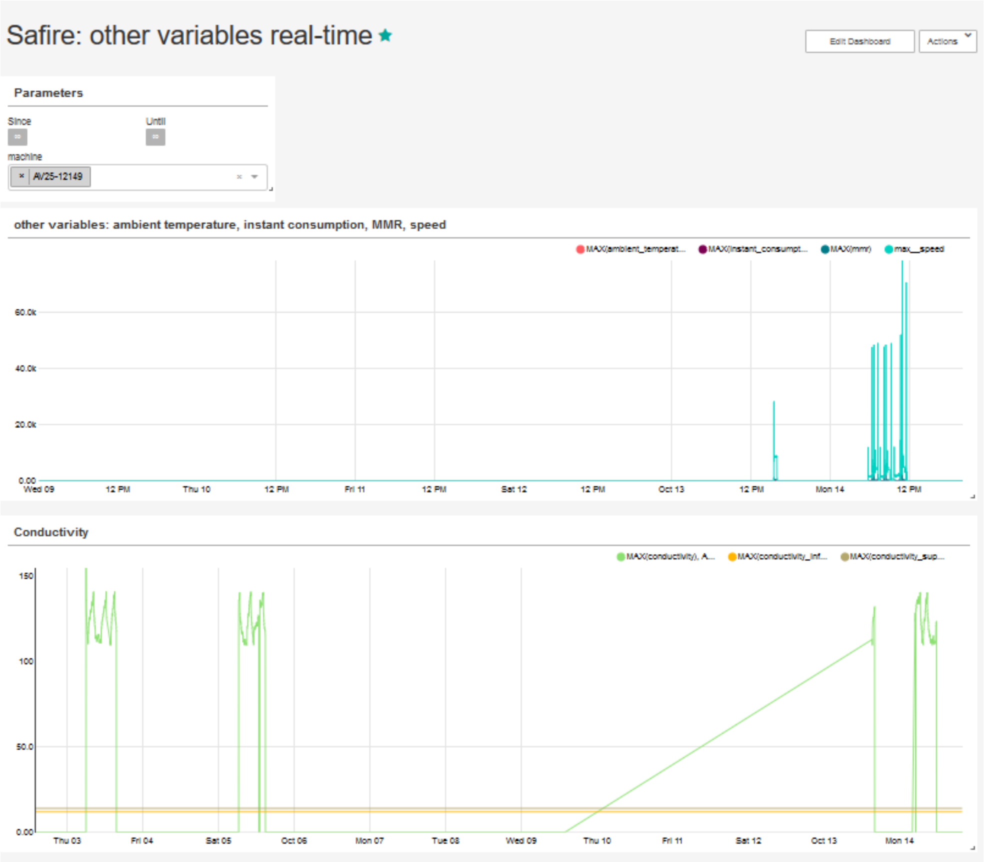


Figure 11: Available chart types in Apache Superset

Finally, after defining one or multiple charts, a dashboard can be created. For this the user needs to click the dashboards option at the top and once the new page is loaded, click on the round button at the top right corner. This new interface allows defining the charts that will comprise the dashboard, and the filtering parameters and chart positions using a not very straightforward JSON definition. It is not advisable to use this interface for anything other than giving the dashboard a name. All other actions can be done visually on the dashboard itself by clicking Edit Dashboard at the top right corner of the interface as shown on Figure 12 and then using the different available actions to perform such tasks.



Safire: other variables real-time ★

Parameters

Since

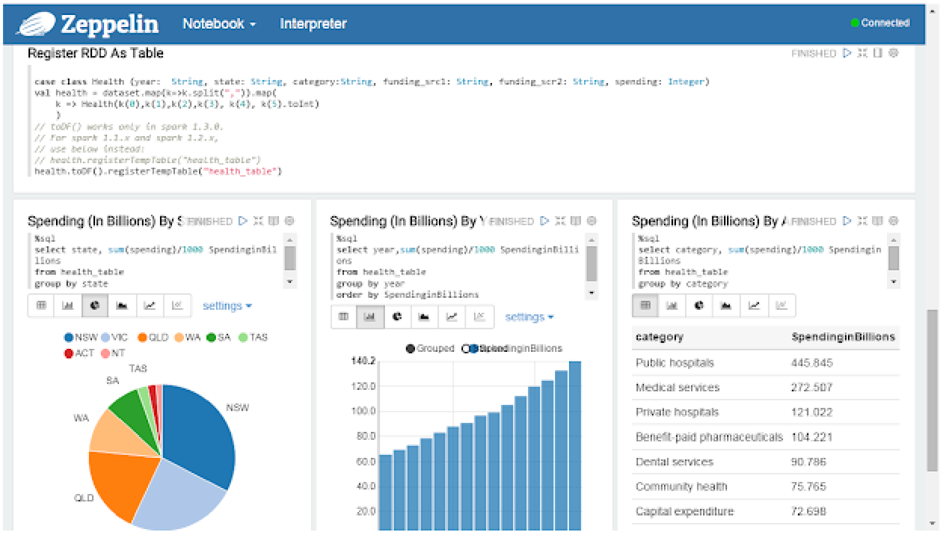
Until

machine: AV25-12149

Figure 12: Example Dashboard

3.4.3 Algorithms, Technologies and Tools

As mentioned in Section 3.2.1 when discussing the tools for the Unified Process Engine, Apache Zeppelin can play a significant role in visualization. Traditional shell applications were limited by the command line interface they run on. Web notebooks can pretty much offer everything a website can show.



```
case class Health (year:  String, state: String, category:String, funding_src1: String, funding_scr2: String, spending: Integer)
val health = dataset.map(k=>k.split(",")).map(
    k => Health(k(0),k(1),k(2),k(3), k(4), k(5).toInt)
    )
// toDF() works only in spark 1.3.0.
// For spark 1.1.x and spark 1.2.x,
// use below instead:
// health.registerTempTable("health_table")
health.toDF().registerTempTable("health_table")
```

```
%sql
select state, sum(spending)/1000 SpendinginBil
lions
from health_table
group by state
```

```
%sql
select year,sum(spending)/1000 SpendinginBilli
ons
from health_table
group by year
order by SpendinginBillions
```

settings
Grouped
Stacked SpendinginBillions
140.2
120.0
100.0
80.0
60.0
40.0
20.0
Spending (In Billions) By A FINISHED

```
%sql
select category, sum(spending)/1000 Spendingin
Billions
from health_table
group by category
```

category	SpendinginBillions
Public hospitals	445.845
Medical services	272.507
Private hospitals	121.022
Benefit-paid pharmaceuticals	104.221
Dental services	90.786
Community health	75.765
Capital expenditure	72.698

Figure 13: Data visualization with Apache Zeppelin

As with the data processing and predictive algorithm testing, thanks to its interactive nature, Zeppelin can be used to test quick ideas around which visualization types to use, which will be different for each potential business goal presented to SAFIRE experts. Figure 13 shows some possible data visualizations that can be created using Apache Zeppelin.

Once the visualization types have been defined, however, the main load of visualization will be handled by the main visualization tool, Apache Superset.

Superset is a business intelligence web visualization tool originally developed by AirBNB and incubated by the Apache Foundation. Its main utility is to create rich and interactive dashboards using a wide array of readily available visualization chart types. It can connect to classical relational databases and current data warehouses, data lakes and NoSQL storage solutions. Most of the configuration of Superset dashboards is done using the web interface without actually needing to use any code to select the data to be shown. However, it is possible to use custom code to further extend the possibilities natively provided by Apache Superset.

4. METHODOLOGY FOR SITUATIONAL AWARENESS

4.1 METHODOLOGY FOR THE CREATION/ADAPTATION OF THE SITUATION MODEL

4.1.1 Overview

In order to enable situational awareness in SAFIRE, first it is necessary to describe the situations to be observed under the production process or product use. This is necessary, since the SD Module is not able to identify any situation unless its structure and related data configuration is predefined. The situations to be observed are defined in the situation model. The situation model defines the limits of the required situations, as well as the framework of the data that will be analysed and the relations between them. In order to more accurately identify situations based on the needs of a specific scenario, different situation models should be created. Within the SAFIRE solution, a generic situation model is created, and one BC-specific model for each business case. For covering a wider range of BC-specific needs, a company specific model can be developed per business case (or an intermediate sector-specific model). The situation model will include information on product/machine usage, process operation information, infrastructure usage and actor information when those affect or interfere with the observed situations.

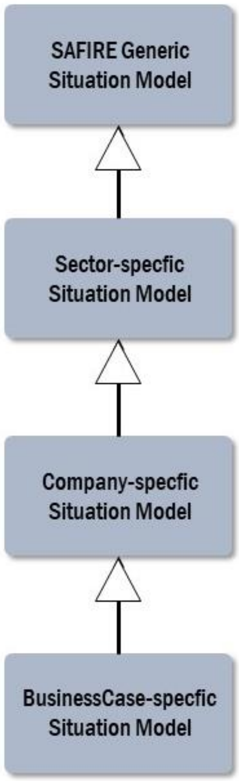

SAFIRE Generic Situation Model
Sector-specfic Situation Model
Company-specfic Situation Model
BusinessCase-specfic Situation Model

Figure 15: To be named

The key task for the correct operation of the SD module is the definition of a 'holistic' and dynamic situation model/ontology, taking into account several factors:

- the situation of products/machines and processes, in which the product/machine is used,
- the situation of the user,
- the process-operation information.

The ontology-based situation modelling is promising to be applicable to the wide scope of SAFIRE, asking for minimal adjustments or developments.

The following sections present the guidelines for creating a generic situation model is being described, as well as how to extend it to more BC-specific ones. Additionally, the algorithms, technologies and tools used are described could be found below.

4.1.2 Guidelines

The approach selected to model the situation is ontology based. Due to their flexibility, expressiveness and extensibility, ontologies can be considered as the most suitable candidates for situation representation. They ensure that different entities that use the situation data have a common semantic understanding of that data. They also come with reasoning mechanisms over the available situation data, making it possible to extract inferred knowledge out of the implicitly stated situations. The key task is the definition of a ‘holistic’ and dynamic situation model / ontology, taking into account the situation of products/machines and processes, in which the product/machine is used, situation of the user, or process operation etc.

For defining a situation model (per observed situation), the following steps are to be followed:

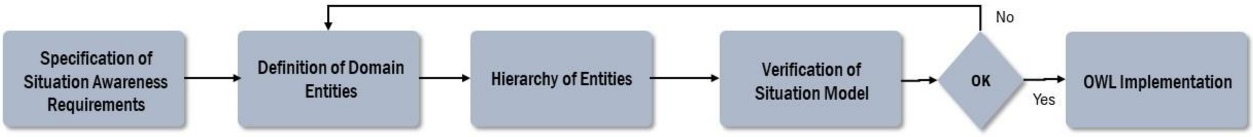

Specification of Situation Awareness Requirements
Definition of Domain Entities
Hierarchy of Entities
Verification of Situation Model
OK
No
Yes
OWL Implementation

Figure 16: Tbn

1. **Identification of the requirements for situational awareness.**
The objective of this stage is to obtain a detailed description of the different scenarios than can be monitored by the SD module. This description includes:

    – the identification of the different sources of data and the type of data available,

    – the correlations between the different data sources and the observed scenarios,

    – the identification of which of the available data are relevant to the monitoring.

    The starting point for the specification for situational awareness is the performance of a workshop where all the stakeholders take part: SAFIRE Experts and industry experts such as plant managers, process managers, machine operators, quality control managers and operators, Marketing Staff, Product Design, etc. It is recommended that the workshop follows Zwicky’s General Morphological Analysis (GMA) [Zwicky, 1969] for problem identification/ solving, that has been extensively applied in the study of the structural relationships between the different parts of an element/system of study facilitating the identification of the inputs/outputs of production systems and the study of the different interactions between their elements/components [Ostertagová, 2012]. The outcomes of the workshop and the methodology used for the effective performance of the workshop is detailed in the *Appendix 1* of the present document.

2. **Definition of the domain entities.**
Out of the outcomes generated in the first step, an SAFIRE expert identifies the general entities that aggregate all the relevant data together, and describes the basic relations between them. In order to obtain an easy to use and clear situation model, understandable from non-experts and reusable for different scenarios or

business cases, the following rules shall be used for the definition of the entities/classes and their relationships:

– Use of simple and singular terms for the concepts.

– Use of terms that are well-known in the domain of application of the observed scenario that are confirmed with domain experts. For the SAFIRE solution, the industry experts could give valuable information and approve the selected terms. Therefore, their support should be used constantly during the process of definition and reviews of the situation model(s).

– Provide a clear description of the entities and the relations chosen.

3. **Creation of the hierarchy of entities.**
Structure the selected entities in a hierarchy based on the *"is_a" relation*. This way a rational schema of the situations to be observed will be created and will enable the easy review and refinement of the model.

4. **Verification of the situation model.**
Review the situation model created so far and refine the entities and the relations. This step is important since it will allow an overview which might reveal redundant information inclusion, or missing categories that are important for the accurate evaluation of a situation. This step, as well as the previous two, are executed iteratively until the result matches the best the required situation to be observed.

5. **Design, verification and validation of the situation model in RDF/OWL.**
Create a situation model in .owl format, so that the SD module can read it, including the specified relations between the entities. The tool used to implement the ontology is Protégé, widely used in semantic web development and very user friendly. Protégé includes tools to verify the correct syntax of the model and also includes an inference engine and the SPRQL Language for inference resolution that can be used to validate the model before its integration in the "Situation Model and Determination Services".

6. **Proceed on creating more specialised situation models** (BC and company specific situation models) by repeating the steps mentioned before. For the BC-specific situation models (and respectively for the company specific) the first step of the situation modelling methodology should be focus more targeted to the BC specific particularities. The process will start having the generic situation model as basis and will continue until the necessary business-case related information are added under the generic concept classes. For this step, close cooperation of the modelling experts with the industry experts is necessary.

Challenges

The analysis of the scenarios to be observed and the definition of the situation model(s) is not an easy process. Depending of the scope of the scenario the situation model can be very complex and the identification of, explicitly, the relevant data to be used might be difficult to proceed.

The key problems in defining a situation model are:

- *Where is the border between the basic operation of the SAFIRE solution and the extended operation using situational awareness?*
  The SAFIRE approach to this challenge is initially to define its **basic operation**. A basic operation of the solution is the one that takes only into consideration aspects that remain static within the given scenario of observation. For instance, if the scenario concerns the operation of a factory process, then the related static aspects could be information on the specialised hardware used (e.g. number of processors, power resources needed, etc), or information on the personnel who operates the specialised hardware. Information that could change over time (e.g, availability of the needed hardware infrastructure, or presence of the personnel needed), and is observed during this period cannot be taken into consideration for the basic operation of the SAFIRE solution, since this information is not automatically identifiable from the system. In this case, the difference from the extended operation is foreseen. The operation of the SAFIRE solution using situational awareness will consider all the information that is relevant for the observed scenarios and could change over time, allowing the adjustment of the functionality to the current needs of operation or usage of processes, products and machines.
- *Which is the relevant information to be associated with the situation model?*
  For SAFIRE, situation can be considered any information available (represented by a concept class) about the circumstances under which products/machines and processes operate or are being used. Any of this information could be included in the situation model to define the scope of situational awareness. However, considering all the available data and processing them will increase the costs (e.g. for sensors, processing algorithms and computers), therefore, it must be carefully studied which information has to be included in the situation model. The decision on which information to include, and which not, depends (exclusively) on the purpose of the situational awareness, i.e. to which changes in the circumstances we want to adapt the operation of SAFIRE and to which circumstances the SAFIRE modules need to perform an action. For instance, if we need to adapt a process or the functionality of a product/machine when the temperature in the room where the process is running, or the product/machines is being used, then the information “room temperature” should be included in the situation model, as it is a necessary aspect to drive the SAFIRE operation. On the other hand, if the processes or the products/machines do not operate in a different way on a change in the room temperature, then the SAFIRE modules do not need to consider as situation the related data, therefore, the related information can be omitted from the situation model. On the same way, information that do not need to be explicitly used for adjusting the operation of SAFIRE to occurring circumstances, should not be included in the situation model.

BC-specific Situation Model(s)

Following the approach described above, for each scenario which the SAFIRE solution will observe, a situation model should be created. Depending on where the focus, or the purpose, for situational awareness of the scenario is, the situation model will include only the related part of information. The following figure presents an example of an extended generic situation model to match the BC specific requirements related to the “GenericDatum” concept class.

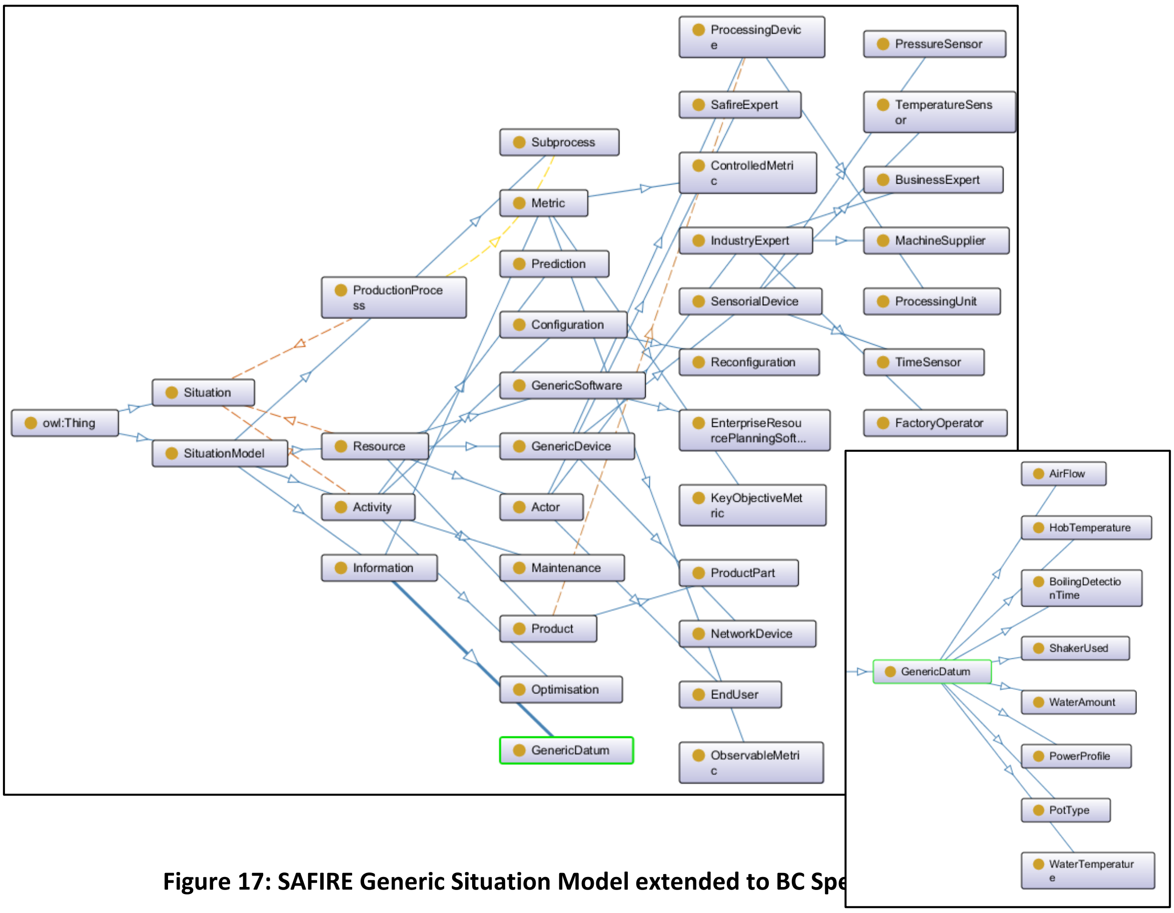


Figure 17: SAFIRE Generic Situation Model extended to BC Spe

For cases where the focus of the scenario is wider, and in order to avoid high costs of processing, more situation models can be created to cover all the necessary aspects of the same scenario. For example, in the case of requiring a production process specific model, the “ProductionProcess” Class can be extended accordingly, as shown in the picture below.

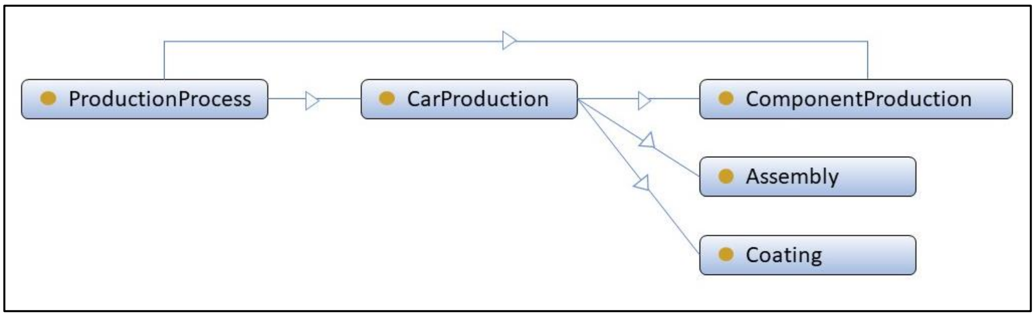


Figure 18: New SAFIRE Situation Model for a specific production process

In the illustration it can be observed how “ComponentProduction” can be further extended, ideally with models provided by a third-party like the component manufacturer. Before attempting the design of the new model it would be necessary to research about the existence of previous ontologies that might already exist (e.g. by searching the W3C community and business groups[1] related to ontologies) and could be integrated within the model.

4.1.3 Algorithms, Technologies and Tools

The situation modelling includes the identification of a set of features that determine the situation under which products/machines and processes are being used or operate, as well as their usage related concepts. Consequently, it includes the identification of the set of parameters to be monitored depending on the use-case scenarios to be covered. Since ontologies allow for flexible representation of information in a structured way, this approach was chosen for the situation modelling. Additionally, an OWL-based situation model provides an explicit machine interpretable knowledge representation, and (re)use. However, modelling using this approach, as already mentioned, could be costly, therefore, the following principles were identified to enhance this process:

PRINCIPLES FOR SITUATION MODELLING

1 Initial consideration of a main situation with basic data needs.

This principle will allow for an initial simple and implementable solution to be created. The initially simple model will require more limited resources for processing than a more detailed one, and at the same time will provide a quick overview of the highlevel necessary information. After the completion of some first test operations of the SAFIRE solution, this simple model can be extended and more deep level information can be added.

2 Initial modelling of easy acquirable situation.

[1] https://www.w3.org/community/

Based on the provided (by the respective experts) industrial means (e.g. sensors, data files, user feedback, etc.) for acquiring situation information, easy acquirable situation is the one which could be identified within the available data inputs. Therefore, it is important to focus initially on modelling only what is currently available to be monitored.

3 **Consideration of the trade-off between investment of situation modelling/determination and the effects of situation sensitive adoption.**

Intuitively, if we could model as much situation factors in as much details, the accuracy of situation will be higher. However, this comes often with a high cost. On the one hand, more time and efforts are needed on situation modelling, and on the other hand, more computing recourses are necessary to handle the situation, which will bring deficiency to the adoption process. Therefore, a balance should be kept on the complexity of modelled information.

For the situation modelling process, it is recommended to use the Protégé, since it is a free open source platform, widely used in ontology development. It provides the necessary functionality to implement all the requirements of the SAFIRE situation model(s). Additionally, the platform fully supports the latest OWL 2 Web Ontology Language and RDF specifications from the World Wide Web Consortium, which are being used within the situation determination part of the SD module. Its desktop application is a powerful tool since its interface can be customised to the specific needs.

Figure 19 shows an example of the SAFIRE generic situation model visualised in Protégé.

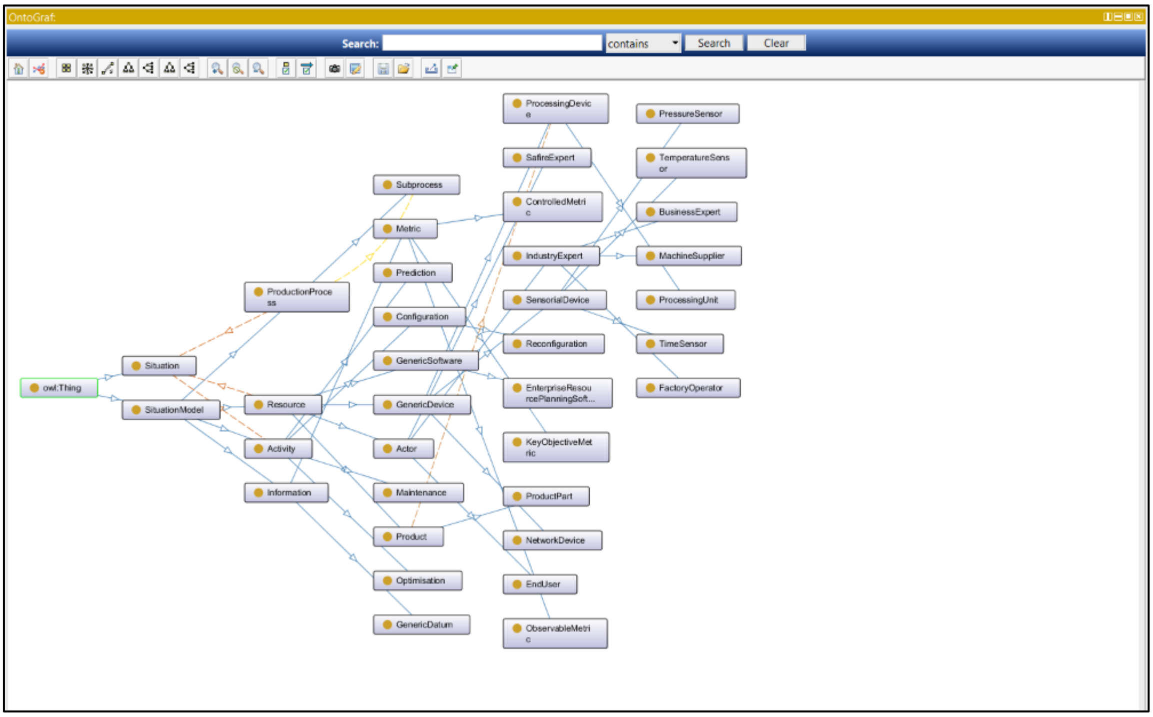


Figure 19: SAFIRE Generic Situation Model Created and Visualised in Protégé

4.2 Methodology for the Customisation of the Situation Monitoring Service

4.2.1 Overview

The monitoring services provide an implicit, passive system to capture the input data that will be used for situational awareness of the SAFIRE solution. It allows for different sources of data to connect, such as file systems, web-service interfaces provided by external systems (like such as .NET based applications) or kafka messaging systems (like those developed for the Data Ingestion and Predictive Analytics modules).

The following figure describes the process the Situation Monitoring component uses internally, from the gathering of the data from external sources, through the processing and the persistence of the structured data for further use.

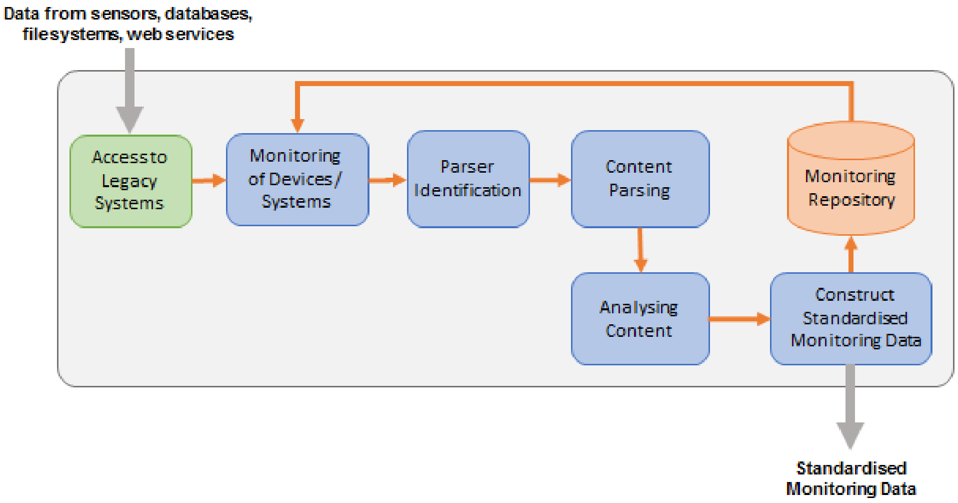

Data from sensors, databases, filesystems, web services
Access to Legacy Systems
Monitoring of Devices / Systems
Parser Identification
Content Parsing
Monitoring Repository
Analysing Content
Construct Standardised Monitoring Data
Standardised Monitoring Data

Figure 20: Monitoring Process

Within the monitoring process, the following parts can be identified:

- **System Monitor**: defines how the data from external sources will be ingested to the monitoring components and receives the incoming data.
- **Monitoring Data Model:** defines the structure of the monitoring data should have in order to be persisted and used from the determination component.
- **Parser**: prepares (structures, transforms, etc.) the incoming data for use within the monitoring component.
- **Analyser**: performs the structuring of the data based on the monitoring data model and forwards the result for persistence.
- **Monitoring Repository**: storage where the structured monitored data are being persisted for further use.

4.2.2 Guidelines

In order to implement the situation monitoring component, and to configure it for supporting different business cases and scenarios, the following steps can be followed:

1. **Identify the requirements for monitoring data.** Experts on the SAFIRE solution and the industry, based on the situation model which defines the needs for situational awareness, should initially define which data scope the monitoring part will observe. These experts will take the results of the workshop for the specification of the situational awareness requirements as an input for this task.
2. **Define the monitoring data model.** Based on the analysis of the previous step and the selected situation model, the monitoring data model should be created to cover the needs of the respective use case scenario. This includes the extension or adjustment of the following classes in the monitoring part of the module:
   - ***IMonitoringDataModel***: Interface defining the data model for the 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 the monitoring data.
     For covering the particularities of the situational awareness respective scenario, a concrete implementation of the IMonitoringData class should be created. Such a class is the following:

3. **Implement the application specific system monitors.** The monitoring part has several generic monitors that were implemented in the current prototype. Each of the monitors is implemented as a software service. In order to cover the particularities of the selected scenario, specific monitors should be implemented. Examples of such implementations are the following:
   - **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.

   - **KafkaMonitor**: This monitor allows to observe topics in a kafka cluster. In SAFIRE these topics are primarily the ones published by the Data-Ingestion Modules.

4. **Implement the necessary monitoring parsers.** The following generic parser is implemented:

   - ***IndexingParser***: This is an abstract implementation of a 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.

   For covering the respective needs of the scenario to monitor, specific parsers should be developed. The following exemplary implementations of parsers can be used or adjusted to specific needs:

   - ***FileParser***: Abstract parser implementation that deals with handling files from a file system. The parsing process itself has to be implemented by each concrete implementation.

   - ***DatabaseParser***: Abstract parser implementation that deals with data from a database. The parsing process itself has to be implemented by each concrete implementation.

   - ***WebServiceParser***: Abstract parser implementation that deals with handling data from a web service. The parsing process itself has to be implemented by each concrete implementation.

   - **ELECParser**: Concrete implementation of an application specific parser. This parser is specialised to parse the data coming from the ELECTROLUX factory, machines, products and users.

   - **ONAcloudParser**: Concrete implementation of an application specific parser. This parser is used to parse the data input for the ONA business cases scenarios.

   - **ProntoParser**: Concrete implementation of an application specific parser. This parser is dedicated to parse the data coming from the proNTo platform used in the OAS business case.

5. **Implement the monitoring analysers**. The following generic analyser is implemented:

   - ***IndexingAnalyser***: This is an abstract implementation of a monitoring analyser.

   The following exemplary implementations of specialised analysers to cover the particularities of the situational awareness scenario could be used or adjusted:

   - ***FileAnalyser***: Abstract analyser implementation that deals with analysing files from a file system. The analysing process itself has to be implemented by each concrete implementation.

- ***DatabaseAnalyser***: Abstract analyser implementation that deals with analysing data from a database. The analysing process itself has to be implemented by each concrete implementation.

- ***WebServiceAnalyser***: Abstract analyser implementation that deals with analysing data from a web service. The analysing process itself has to be implemented by each concrete implementation.

- **ELECAnalyser**: Concrete implementation of an application specific analyser. This analyser is specialised to structure the data to the selected structure for the ELECTROLUX factory, machine, product and user.

- **ONAcloudAnalyser**: Concrete implementation of an application specific analyser. This analyser is used to structure the data for the ONA business cases scenarios.

- **ProntoAnalyser**: Concrete implementation of an application specific analyser. This analyser is dedicated to structure the data to fit the requirements of the OAS business case.

6. **Configure the monitoring sources paths.** In order to define the particularities of the monitoring part, as for example which of the implemented parsers and analysers will be used, a configuration file should be adjusted. The .xml file is named “monitoring-config.xml” and it is placed in the “resources” folder of the monitoring part.

```
<?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-diversity" location="indexes/diversity"></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>
```

6.1 **Configure the monitoring services.** The details of where the monitoring services will run or where the repository for the monitored data should be, should be defined in an xml configuration file. The file is named “services-config.xml” and it is placed in the “resources” folder of the monitoring part. The following example can be used as reference:

```
<?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>
  </services>
</config>
```

6.2 **Administrate the situation monitoring service.** To configure, deploy and administrate the situation monitoring service, an administrative user interface for situational awareness is used, and is described in detail in appendix 4.

4.2.3 Algorithms, technologies and Tools

The Situation Monitoring services serves the purpose of capturing information from sensors / systems. This information will be obtained from Kafka topics, a file system or a Web-Service interface provided by external systems. For example, a system / sensor stores current sensor information in files on a file system. This information is gathered from the threaded monitoring services. Another possibility: data interactions made with web-services can be monitored through the specific services that are able to poll for updates or receive trigger events upon data extraction will occur. Furthermore, the monitoring can gather data pushed into a kafka topic.

4.2.3.1 *Monitoring Framework*

The Situation Monitoring services are included into a modular container. It provides and is extendable for several different monitoring services and offers the ability to configure these for external legacy systems or to be called by web-service request and inject in external sources. It is the interface for all monitoring processes and comprehends the whole monitoring activity chain (see Figure 21).

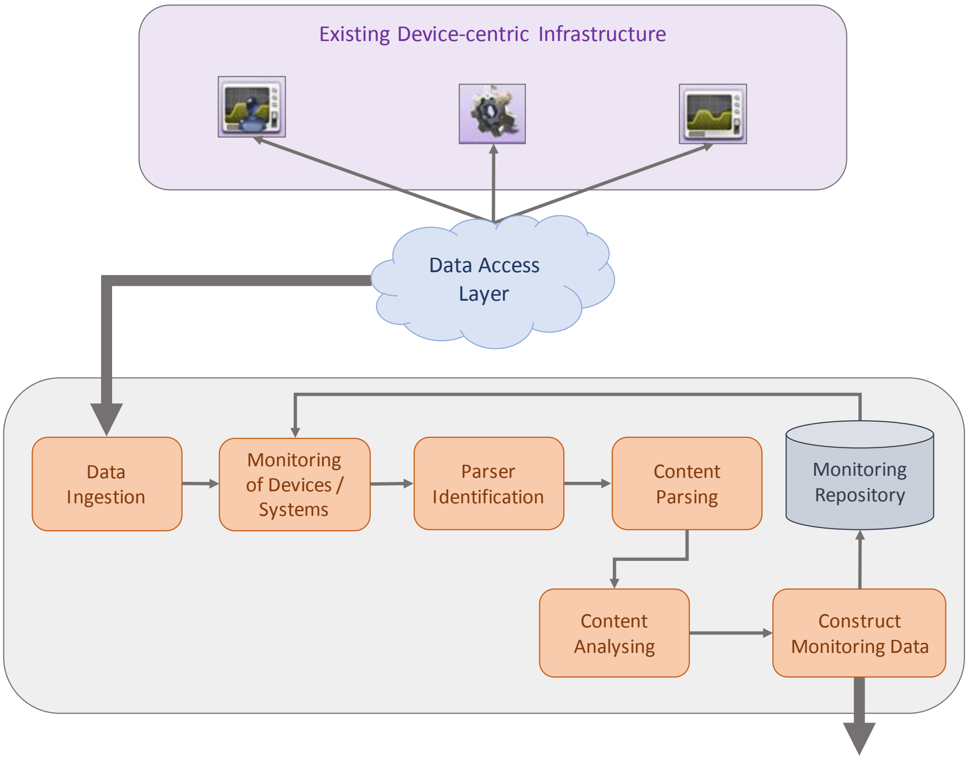


Figure 21: Monitoring Service Platform Process

Therefore, the main functionality of the Situation Monitoring component is:

- To orchestrate and direct requested and configured monitoring services to the depending system
- To operate as interface for all Situation Monitoring Services to send their captured data
- To collect, parse and analyse the gathered information and comprehend and correlate the content and environmental properties
- To construct the respective Situation Monitoring Data and send them to the Situation Determination component

To achieve this functionality the component combines a set of monitoring objects:

- The Situation Monitoring Services, encompasses the possibility to capture general and specific information, and

- The Parser and Analyser, accessing different information formats such as CSV Files, XML documents etc. to collect content and meta-information.

The Early Prototype includes generalised Situation Monitoring Services, which are described in the following chapter.

4.2.3.2 *Monitoring Service for legacy systems*

The process of monitoring different forms of legacy systems is part of the main Monitoring Service framework. It relies on the Data Access Layer and implemented system interfaces, which for the Early Prototype can either be a kafka topic, a file or Web-Service (SOAP) based. However, the data gathered is independent from specific legacy system peculiarities.

The monitoring of external systems is separated into two parts:

- A permanent loop of monitoring of external resources for changes which indicate content change (i.e. new sensory information). For file-based systems this includes monitoring of files and directories and for web-services (SOAP)-based systems this relies on polling the provided interfaces and comparing freshly retrieved data with previously gathered one. For the kafka topic it gets notified (publish-subscribe) if a new topic is available for the monitoring.
- The gathering of all possible information from an external system, besides the content itself. Other possible information refers to environmental properties.

For the permanent loop of watching for changes, the monitoring service requires to look for specialties that offer information about a revision or gives away a date of creation signalling changes. Those so-called “Environment properties” are system specific attributes that define the situation of a system (e.g. the path of a file, the timestamp of a last access of a web-service interface).

4.3 Methodology for the Customisation of the Situation Determination Service

4.3.1 Overview

The Situation Determination is the main component responsible for the identification of any situational changes in the observed environment of the SAFIRE solution. Based on the monitoring of the environment, including external systems, devices and users, the on-going situation for these systems is being identified. This situation will then be used for determination of more specific situational knowledge, using reasoning techniques.

As shown in Figure 22, the Situation Determination identifies situations from the standardised monitoring data provided by the Monitoring Service (Situation Identification), manipulates it through different types of reasoning techniques (Situation Reasoning), and provides the refined situations (the current on-going and a list of similar ones) through a service (Situation Provisioning) to other modules (using kafka messages).

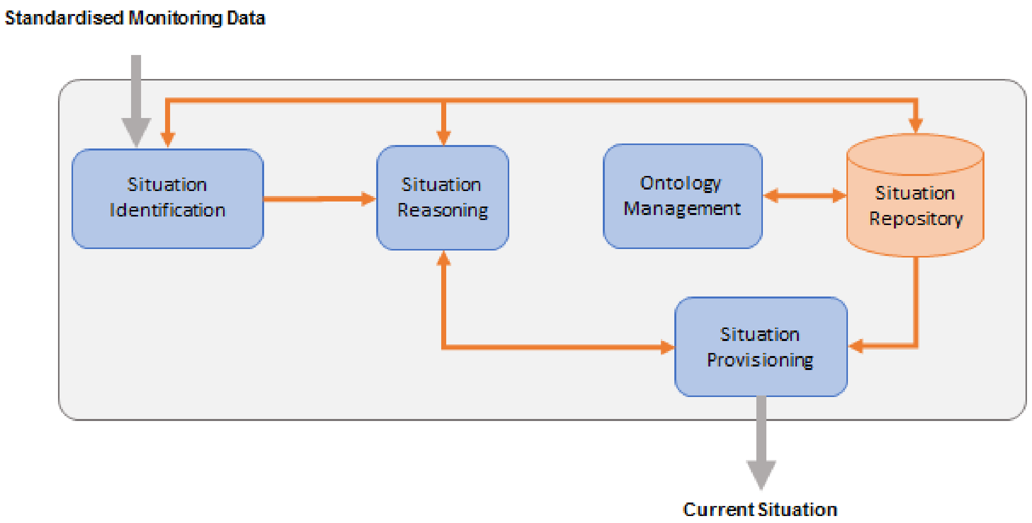


Figure 22: Determination Process

The determination service is based on the Situation Model (semantic model) for an integrated representation of knowledge about products, machines, manufacturing processes and usage information, and provides a method for computing situation similarity measures in order to detect similarities.

The methodology to develop, configure and validate the determination process, which includes the following parts:

- **Situation Identification:** is used to identify possible meaningful situation information among the monitored data, to map information on the given situation model or query situation related information,
- **Situation Reasoning:** is used to perform further analysis of the identified situation in order to extract more accurate and detailed situations.
- **Situation Provisioning:** provides the current situation after comparing it with previous ones in order to identify similarities and assist in statistical reasoning.
- **Situation Repository:** storage of the identified situations, both current and historical.

Each of the above mentioned steps is being described in the following section.

4.3.2 Guidelines

For the customisation of the Situation Determination part, and its configuration to cover the particular needs of a factory or specific use case scenario the following steps can be followed:

1. **Add the situation model .owl file in the resources folder of the module.** The Situation Determination module is based on the information defined in the situation model in order to identify accurately the necessary information within the monitoring data. Therefore, an important step to the configuration of the

module is the situation model input. The situation model should be included in an .owl file under the name “safire-context.owl”. It should be saved in the resources folder of the module so that it can be used during the situation identification process.

2. **Set up the situation repository.** The situation determination part of the module, stores the extracted situations into its own repository.

3. **Configure the determination services.** The details of where the determination service and the respective repository is running can be defined in an .xml file. The file is named “services-config.xml” and it is placed in the “resources” folder of the determination part. The following shows an example of Situation Determination service configuration:

```
<?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>
```

4. **Implement the situation identifiers.** The situation identification process searches through the structured monitoring data for information which reflect to the requested situation. The monitoring data do not contain directly situational information that reflect to the situation model. Therefore, the situation identifiers are wrappers, used to gather all the situation relevant information among the monitored data and map them to the situation model (ontology) format. The identifiers can be adjusted using the following classes:

   – ***IContextIdentifier***: The interface that defines a situation identifier. In each concrete implementation of a situation identifier the usage of a reasoner can be defined. In all current implementations Pellet is/will be used as reasoner.

- ***ContextContainer***: This class is a wrapper object that holds an identified situation during run-time.
  For covering the particularities of the situational awareness respective scenario, a concrete implementation of the IContextIdentifier class should be created. Such a class is for example the following:

- ***MixerRefillingContextIdentifier****:* Concrete implementation of a IContextIdentifier. The identified situation is constructed in this object.

5. **Administrate the situation determination service.** To configure, deploy and administrate the situation determination service, a user interface for situational awareness is used, and is described in detail in appendix 4.

4.3.3 Algorithms, technologies and Tools

The Situation Determination process is separated in two modules – Situation Identification and Situation Reasoning. The Early Prototype implementation if these two modules is explained in the following sections.

4.3.3.1 Situation Identification

The monitored data from the Situation Monitoring service will be sent to the Situation Identification module. It will then be analysed and the situation will be identified, such as what system/sensor is involved, what knowledge items are produced or used, time, location, etc.

Since the monitored data is a list of RDF statements, the identification process is implemented mainly through SPARQL queries and ontological mapping algorithms. For example, assume that there are such RDF statements in the standardised monitoring data:

```
 ...
<rdf:Description rdf:about="#Mixer/28360136">
 <rdf:type rdf:resource=" http://safire-factories.org/bc-oas/Mixer"/>
 <oas:sensoricalMixerInformation rdf:nodeID="A0"/>
 <oas:name>MixerA1</oas:name>
</rdf:Description>
...
<rdf:Description rdf:about="#SensoricalMixerInformation/8440521">
 <rdf:type rdf:resource="http://safire-factories.org/bc-
oas/SensoricalMixerInformation"/>
 <oas:active>1</oas:active>
 <oas:processingTime>0.28</oas:processingTime>
 <oas:cycleNo>9</oas:cycleNo>
 <oas:componentTemperature>45.3</oas:componentTemperature>
 <oas:component>white</oas:component>
</rdf:Description>
...
<rdf:Description rdf:nodeID="A0">
 <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
 <rdf:_1 rdf:resource="#SensoricalMixerInformation/8440521"/>
 <rdf:_2 rdf:resource="#SensoricalMixerInformation/27187756"/>
 <rdf:_3 rdf:resource="#SensoricalMixerInformation/3403998"/>
 ...
</rdf:Description>
```

Code 1: Example abridged RDF Monitoring Data

This can be interpreted as: there is different sensory information for a valve, like the pressure in the valve, the pressure of the component etc. Of course, all information monitored by the Monitoring component is passed over to the Situation Identification module.

By doing a SPARQL query on this data:

```
Select ?mixer ?mixerActive ?mixerCycle
where
{
 ?mixer rdf:type oas:Mixer.
 ?mixer oas:sensoricalMixerInformation ?mixerInfo.
 ?mixerInfo oas:active ?mixerActive.
 ?mixerInfo oas:cycleNo ?mixerCycle.
};
```

Code 2: Example of a SPARQL query

We can retrieve the instance URI and pressure of for example a mixer at a certain point in time (mixer cycle no). Then based on the returned information the situation identification module will match the results on the appropriate ontology classes and create instances of a valve, of pressure sensors and sensory information, valve synchronisation step, the valve synchronisation process etc. In a next step all the required properties are handled and set up accordingly.

The identified situation will be used in Situation Reasoning and a more detailed explanation is presented in the following sections.

4.3.3.2 *Situation Reasoning*

The purpose of Situation Reasoning is to generate more accurate situations from the identified situation. This includes ontology reasoning, rule-based reasoning, as well as statistic reasoning.

Ontology reasoning

Ontology reasoning is implemented through Jena Inference Engine and other external libraries that provide reasoning functionality.

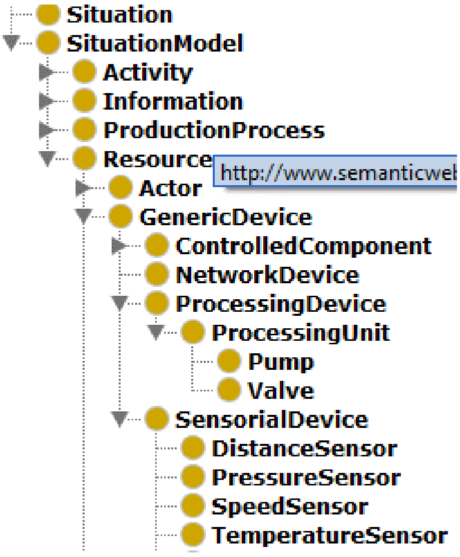


Figure 23: Subsumption via Reasoning

For example, in Figure 23, it can be seen that `TemperatureSensor` is a subclass of `SensoricalDevice`, `SensoricalDevice` is a subclass of `GenericDevice`, `GenericDevice` a subclass of `Resource` (and so on) and all these concepts are subclasses of `SituationModel`. In order to tell that Millimetre is also a sub class of `SituationModel`, one has to use at least a `TransitiveReasoner` (provided by Jena) while creating / loading the ontology:

```
TransitiveReasoner reasoner = new TransitiveReasoner();
InfModel infM = ModelFactory.createInfModel(reasoner, schemaModel, rawModel);
```

Code 3: Creating a TransitiveReasoner

Where the `rawModel` includes statements of an existing situation ontology instance (or even the complete situation ontology), `schemaModel` is the SAFIRE ontology definition, the `infM` is the reasoning result in form of another ontology model.

RDFS reasoning and OWL reasoning are also provided in Jena. In the early prototype, transitive reasoning for exploring the situation hierarchy, which is required to calculate situation similarity, will be used. For the full prototype more high-performance reasoners like Pellet[2] will be integrated in the final solution.

Rule based reasoning

Rule based reasoning is implemented through Jena Inference Engine:

```
Reasoner ruleReasoner = new GenericRuleReasoner(Rule.rulesFromURL(ruleURL));
InfModel infM = ModelFactory.createInfModel(ruleReasoner, rawModel);
```

Where the user defined rules are stored in the `ruleURL`. For example, by applying the following rule to the identified situation:

```
@prefix rdfs:           <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:            <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix safire:   <http://safire-factories.org/base.owl#> .
[rule1:  (?a safire:hasDevicePart ?b)
                 (?b rdf:type safire:Mixer)
                 (?b safire:isObservedBy ?c)
                 (?c rdf:type safire:PressureSensor)
                 (?c safire:providesPressure ?d)
                 (?d rdf:type safire:Bar)
        -> (?a rdf:type safire:MixingHead)
]
```

Code 4 – Example Rule for Rule-Based Situation Reasoning

Code 4 shows an example, which can be explained as “if a processing device has a mixer attached to it, and this mixer is observed by pressure sensor which provides a resource identified as pressure in Bar this processing device is of type MixingHead”.

[2] http://clarkparsia.com/pellet/

Statistic reasoning

Statistic reasoning does not rely on strict logical rules but instead tries to correlate information into possible relations, as suggested by the empirical data. In the early prototype no statistical reasoning was implemented, but for the full prototype, it will be further explored and elaborated how on-going (i.e. currently identified) situations can be compared with historical (i.e. previously identified) situations.

For the full prototype, statistic reasoning to sort on-going situations based on a so-called PCS (Possible Current Situation) possibility criteria will be implemented. PCS possibility will be calculated based on the identified situation and the historical situation.

PCS possibility is used to help set up current, most relevant situation. If there is no on-going situation at all, a new current situation will be created for the system, and the recently identified situation will be glued to it. Otherwise the situation with the highest PCS possibility value will be used as the current relevant one.

4.3.3.3 Situation Provision

The purpose of Situation Provision is to provide the identified situations to other modules or downstream services. In SAFIRE, the Situation Provision, will publish the identified situations in the kafka cluster of the SAFIRE infrastructue, which means it is acting as a kafka topic producer.

As data format for the topics, the situation provision is making use of the Metric API (see *D3.1 Methodology for dynamic and predictable Optimisation and Reconfiguration Engine* for a detailed description of the Metrics API and its usage).

5. METHODOLOGY FOR OPTIMISATION

5.1 OVERVIEW

We describe in this section the main methodology followed in SAFIRE for the optimisation of manufacturing processes. Firstly, we describe a powerful analytical model that can be applied to evaluate the quality of a variety of process planning approaches. We then describe the way such analytical models can be used as fitness functions within SAFIRE's optimisation engine (OE), which follows an evolutionary approach that enabled us to explore large multi-objective optimisation problems. We provide details on the methodologies we developed to accelerate and increase performance-predictability of the OE, by exploring large scale parallelism in a cloud environment, and using techniques such as evolutionary islands. Finally, we describe additional tools and methodologies we developed to facilitate the customisation and development of fitness functions for different business cases, including semi-automatic generation that also relies on the SAFIRE evolutionary infrastructure. With the provided support, we could improve the fitness functions developed for the SAFIRE BC, and we provide enough detail and tool support to enable the creation of fitness functions that allow other BCs beyond SAFIRE to reuse the SAFIRE optimisation approach.

5.2 ANALYTICAL PLATFORM MODELS USED IN SAFIRE

In this section, Interval Algebra, the algebraic model that is used in SAFIRE to explicitly describe objective functions is presented. The rationale behind selecting the Interval Algebra rather than Max-Plus algebra, both described in deliverable D3.1 (1), is the complexity of the considered business cases that requiries using specific features in the Interval Algebra, absent in the Max-Plus Algebra, for example mutual exclusiveness of resources or task affinities. An example of applying Max-Plus algebra to a simpler use case has been described in paper (2).

5.2.1 Interval algebra

One of the optimisation problems that are required to be solved by the SAFIRE optimisation engine is the batch scheduling problem. In the project, this problem is exemplified with the use cases from one of the project industrial partner, OAS. As described later in SAFIRE, we propose to use Interval Algebra (IA) as an analytical model that can describe a related fitness function. As IA was invented in the course of the DreamCloud project (funded under FP7-ICT, project ID: 611411) for task scheduling in real-time computer-based systems, some extensions of it would be required to cover some characteristics of batch scheduling problems. For the sake of self-consistency of this document, the most important features of IA are described in this section, followed by the required new features.

In the SAFIRE project, a manufacturing process is viewed as a set of *tasks*, a taskset $\Gamma = \{\tau_1, \tau_2, \tau_3, ...\}$. The tasks appearing exactly once during a manufacturing are often referred to as *singletons* and are composed of a single *job*. A periodic or sporadic task can be treated as an infinite series of jobs that are released periodically or less often than the provided inter-release time, respectively. The *j*-th occurrence (*j*-th job) of a periodic or sporadic task τ_i is denoted with $\tau_{i,j}$. Tasks are mapped to plant resources, such as conveyors or machines using a selected *scheduler*.

The proposed IA can be used for the analysis of different resource composition alternatives. It is possible to check influence on temporal characteristics (OEE, makespan, slack, etc.) and estimate the total energy dissipated by the modelled solution while processing the provided taskset. To evaluate plant characteristics with IA, system description is to be provided including plant architectures (machines, storage units etc., together with the possible connections between them), different scheduling policies (pre-emptive and non-pre-emptive), or different priorities of tasks.

Let us consider a simple example. A given manufacturing process is composed of three singleton tasks: τ_1, τ_2 and τ_3, to be processed with one of the two available machines of the same type with the first-in-first-out (FIFO) scheduling. Each of the tasks can be represented by an interval that denotes the time it is processed using one of the machines: τ_1 = [0,30), τ_2 = [0,45), τ_3 = [0,20) (assuming in this example that τ_1, τ_2, τ_3 are all independent and ready to run at time t = 0). By using simple interval algebra operations, a resource allocation heuristic can estimate the makespan R of the three tasks under different allocation schemes (e.g. $R_{\tau 1}$ = 30, $R_{\tau 2}$ = 45 and $R_{\tau 3}$ = 50 if τ_1 and τ_3 are allocated, in that order, to one of the machines and τ_2 is allocated to another), and thus can dynamically decide whether it is likely to meet the temporal constraints when using a given allocation. While trivial, such example can be made arbitrarily complex by allowing different resource scheduling disciplines, a larger number of tasks and machines. For IA, however, the analysis of the makespan under a specific allocation would still involve the application of the same interval manipulation rules.

Formally, an algebra is a definition of symbols and the rules for manipulating those symbols. IA therefore establishes rules for the manipulation of intervals. IA defines different types of intervals, which represent the amount of time a particular manufacturing process requires from a notional plant resource. It also defines rules for manipulations of such intervals: what happens when an interval is allocated to a specific type of resource, what if two intervals are allocated to the same resource, etc. Two basic algebraic operations are needed: *time displacement* and *partition*. Time displacement changes the endpoints of an interval by an arbitrary value t, and denotes that the application component had to wait for its allocated resource (i.e. its starting and ending times were moved t time units to the future). Partition simply breaks one interval in two, and denotes that an application component was pre-empted from a resource (and the second interval produced by the partition is likely to be time-displaced). All other interval-algebraic operations of IA, which can represent an arbitrarily large set of allocation and scheduling mechanisms, can be expressed as compositions of those two. By applying those operations, it is possible to investigate the impact of different resource allocation and scheduling mechanisms on the endpoints of the intervals, which in turn denote the completion times of each manufacturing processes.

5.2.1.1 *Modelling application architecture*

Using IA, application jobs are represented as intervals. For example, a singleton task can be represented by the time interval it requires from a notional resource. It can be denoted with the notation exemplified below:

$\#\tau_1\#0\#40,$ (5.2.1.1)

where the first element of the tuple is a unique job identifier, the second is a non-negative real number representing the release time of the job and the third is a positive real number representing the load of the job, i.e. the actual length of the time interval. In the example above, job τ_1 is released at time 0 and requires 40-time units of a resource. The same concept can also be represented using the mathematical notation for a left-closed right-open bounded interval [0,40).

Such interval-based representation of a job is sufficient to express a singleton, and by using a set of intervals, independent jobs can be also represented. To denote a dependency between two tasks τ_1 and τ_2, the notation can be extended to include a job identifier instead of the release time of a job:

$\#\tau_2\#\tau_1\#50$ (5.2.1.2)

This notation is capable of denoting single dependency jobs, and conveys that the start-point of the interval τ_2 depends on interval τ_1. Multiple dependencies can also be specified as a dependency set, and thus multi-dependency jobs can be covered:

$\#\tau_3\#\{\tau_1,\tau_2\}\#260$ (5.2.1.3)

This notation assumes that whenever an interval has dependencies, its start-point lies exactly at the highest endpoint among all the intervals it depends on. In this example, assuming that tasks τ_1 and τ_2 are defined as in formulas (5.2.1.1) and (5.2.1.2), this leads to: $\tau_1 = [0,40), \tau_2 = [40,90), \tau_3 = [90,350)$.

5.2.1.2 *Modelling manufacturing process temporal behaviour*

The intervals described in the previous subsection are single-appearance and have a fixed release time, therefore express singleton tasks. A strictly periodic series of jobs can be characterised by its release time, the period after which a new job is released, and the time interval each job requires from a notional resource. We denote such job series with the notation exemplified below, which is exactly the same as the notation of a singleton task followed by the period:

$\#\tau_4\#0\#40\#100.$ (5.2.1.4)

Mathematically, it represents an infinite series of intervals, such as: τ_4 = [0,40), [100,140), [200,240),.... This extension is expressive enough to represent strictly periodic tasks.

The release time of sporadic tasks is not deterministic but has well defined bounds. In case of aperiodic tasks, those bounds do not exist. To model those cases, we can represent release times with so-called *aleatory variables*. Those variables are associated with probability distributions that can constrain assumed values.

5.2.1.3 Modelling resourcing constraints

A resource can be represented by an algebraic operation over all the jobs mapped onto it, each represented by its respective interval. The algebraic operation determines how the resource is shared between the jobs mapped to it, and how the sharing affects their timings. We denote a resource with the notation exemplified below:

$$+\pi_1(\#\tau_1\#0\#40), \tag{5.2.1.5}$$

where the algebraic operation π_1 is applied to the set of intervals surrounded by brackets (only τ_1 in the example above). The example below shows the same resource, but this time with two distinct jobs mapped to it:

$$+\pi_1(\#\tau_1\#0\#40,\#\tau_2\#0\#50) = +\pi_1(\#\tau_1\&40,\#\tau_2\&90) = +\pi_1([0,90)) \tag{5.2.1.6}$$

In this example, we introduce two different ways to evaluate the operator π_1 (which we can intuitively understand as a resource serving jobs under a FIFO schedule). The first evaluation of the operator preserves the identities of the mapped jobs, and it indicates the completion times of each one of them after the symbol ”&”. We will refer to this type of evaluation as *information-preserving* (or simply *preserving*). The second way to evaluate the operator is equivalent to the first, but it does not preserve any information about the individual operands. It simply determines the busy period(s) of the resource with one or more intervals. We refer to this type of evaluation as *information-collapsing* (or simply *collapsing*).

A slightly different example is shown below, using the same jobs but this time mapped onto resource π_2 that uses a time division multiplexing (TDM) scheduler with a quantum of 8-time units:

$$+\pi_2(\#\tau_1\#0\#40,\#\tau_2\#0\#50) =+\pi_2(\#\tau_1\&72,\#\tau_2\&90) = +\pi_2([0,90)) \tag{5.2.1.7}$$

It is worth noticing that only the intermediate expression (i.e. after the preserving evaluation) differs, and the final result after the collapsing evaluation is the same. This is always the case if the operand denotes a work-preserving scheduler, when no resource is idle as long as there are tasks ready to be processed on this resource.

The two following examples show jobs mapped onto a resource that is shared under a priority-pre-emptive scheduler, assigning priorities in the same order the jobs are passed to the operator (higher to lower):

$$+\pi_3(\#\tau_3\#15\#40,\#\tau_4\#10\#50,\#\tau_5\#0\#50) = +\pi_3(\#\tau_3\&55,\#\tau_4\&100,\#\tau_5\&140) = +\Pi_3([0,140)) \tag{5.2.1.8}$$

$$+\pi_4(\#\tau_6\#10\#4,\#\tau_7\#0\#18,\#\tau_8\#26\#5,\#\tau_9\#24\#8) = +\pi_4(\#\tau_6\&14,\#\tau_7\&22,\#\tau_8\&31,\#\tau_9\&37) = +\pi_4([0,22),[24,37)) \tag{5.2.1.9}$$

In both cases, the algebraic operations abstract away the specific interleaving patterns of the processing of every job. Each of the evaluation types focuses solely on, respectively, the finish times of each job or the idleness of the resource. For example, formula (5.2.1.9) represents the following: task τ_7 starts to be processed at time zero, but after 10-time units it is pre-empted by task τ_6 which runs to completion for 10 time units; then τ_7 resumes and runs for its remaining processing time until time equals 22 units; resource π_4 becomes idle until task τ_9 is released at 24 time units, which in turn executes until time equals 37 units.

Just like single appearance jobs, periodic jobs can be mapped to resources:

$$+\pi_1(\#\tau_1\#0\#40\#100, \#\tau_2\#0\#50) = \quad (5.2.1.10)$$
$$+\pi_1(\#\tau_1\&40, \#\tau_2\&90, \#\tau_1\#100\#40\#100) =$$
$$+\pi_1([0,90), \#\tau_1\#100\#40\#100)$$

It is important to notice that a periodic job series always remains as a distinct interval in the result of both preserving and collapsing evaluations of an operator. This reflects the infinite nature of the series.

One of crucial properties of each task is its affinity, which means that it can be processed only on the designated resources. The task that can be executed on any resource available in a system is referred to as *untyped task*. If a task can be executed on a single type of resources only, it is a *single-typed* task. A *multi-typed* task can be executed on a few (enumerated) resource types, possibly with different processing time. In all the earlier examples, untyped tasks have been presented only. To describe a single-typed or multi-typed task, the notation should support the definition of different types of resources and different types of resource affinity. This can be expressed as follows, where each scalar in pointy brackets denotes a different type and the absence of type constraints implies untyped jobs or resources (as in examples above):

$$+\pi_1 < 2 > (\#\tau_{10} < 2 > \#0\#15, \#\tau_{11} < 2,3,8 > \#0\#20, \#\tau_{12}\#0\#14) \quad (5.2.1.11)$$

By allowing the definition of resource types and resource requirements, it is also possible to present transport jobs between two machines (e.g. using a conveyor) by modelling the job as two fully dependent intervals with distinct resource requirements, one for processing and one for transporting (i.e. the job can only be connected over resource 2 once it has finished being processed by resource 1):

$$\begin{array}{l} \#\tau_{13} < 1 > \#0\#14 \\ \#\tau_{14} < 2 > \#\tau_{13}\#340 \end{array} \quad (5.2.1.12)$$

5.2.1.4 Modelling taskset load characterisation

The representation of load as the interval length, denoted by a positive real number, is already capable of representing a fixed load.

To represent a typed fixed load, we allow the specification of different interval lengths for different resource types using a similar notation as the one introduced earlier:

$$\#\tau_{15} < 2{,}4{,}6 > \#0\# < 10{,}20{,}20 > \qquad (5.2.1.13)$$

To represent a probabilistic load or typed probabilistic load, we have to rely again on aleatory variables to represent the load. This can be done for both typed and untyped jobs.

5.2.1.5 IA extensions in SAFIRE

The optimisation engine in SAFIRE aims to be capable of solving batch scheduling problems with different process topology, including both single and multiple stages. In the latter, multiproduct (aka flow shop) topology should be covered. According to (3), when using this topology, products can be processed on a range of alternative resources using different routes and in any batch size. This implies that between certain resources additional relations have to be defined, denoting their affinity or anti-affinity. Another requirement is related to changeovers which can be sequence-dependent. It means that a new feature of sequence-dependent setup should be introduced. Another required feature stems from the material transfer requirements, as in a plant it is possible that certain tasks have to be executed immediately one after another. As a result, immediate precedence relationship is to be added to IA, which originally allowed only general precedence relationships. Finally, inventory storage policies should be possible to be imposed. To fulfil this requirement, certain tags should be associable with tasks that would inform about the produced commodities and their amount. The new features added to the original IA in order to satisfy all these requirements are described below.

As OE in SAFIRE can be related to the optimisation of batch scheduling problems, some extension to the IA described earlier has to be introduced. In batch scheduling, tasks belonging to a manufacturing job are scheduled to the resources present in a certain plant. A *manufacturing job* is then defined as a set of dependent tasks (i.e., task in an immediate or general precedence relationship, explained below) that have to be processed in order to produce a certain amount of certain commodity. All tasks belonging to a single manufacturing job correspond to a recipe describing a certain manufacturing process. Consequently, there is no possibility of allocating only a subset of such tasks, as the corresponding recipe would not be followed with such partial coverage. The tasks belonging to a single manufacturing job should be then allocated according to their topological order and in case any of its tasks cannot be allocated, the whole job should be rejected.

IA should be capable to solve batch scheduling problems with different process topology, including both single and multiple stages. In the latter, multiproduct (aka flow shop) topology should be covered. According to (3), when using this topology, products can be processed on a range of alternative resources using different routes and in any batch size. This implies that between certain resources additional relations have to be defined, denoting their affinity or anti-affinity. Another requirement is related to changeovers which can be sequence-dependent. It means that a new feature of sequence-dependent setup should be introduced. Another required feature stems from the material transfer requirements, as in a plant it is possible that certain tasks have to be executed immediately one after another. As a result, immediate precedence relationship is to be added to IA, which originally allowed only general precedence relationships. Finally, inventory storage policies should be possible to be imposed. To fulfil this requirement, certain tags should be associable with tasks that would inform about the produced commodities and their amount. The new features added to the original IA in order to satisfy all these requirements are described below.

5.2.1.6 *Mutual exclusiveness of resources*

In plants, certain resources cannot be used at the same time. For example, in the considered OAS use case, an example of such relation can be two conveyors that transport raw material from different silos to the same weighting scale. To prevent such resources to be active at the same time, it is necessary to define a mutex relation between a pair of resources R1 and R2. This binary relation holds if two R1 and R2 cannot be active at the same time. The mutex relation is symmetric and transitive.

5.2.1.7 *Different routes*

Since in the multiproduct topology different routes between processing machines are possible, but the equipment connectivity can be limited (partial), for example defined by existing conveyors or pipes, an additional relation between resources than can be used sequentially by tasks belonging to a single manufacturing job is necessary. Two relations are introduced:

- an affinity relation between resource R_1 and resource R_2 means that for two different tasks t_1 and t_2 belonging to the same manufacturing job and $t_1 < t_2$ with respect to the topological order of the manufacturing job, where R_1 is compatible with task t_1 and R_2 is compatible with resource R_2, if t_1 is allocated to R_1 then t_2 can be allocated to R_2.

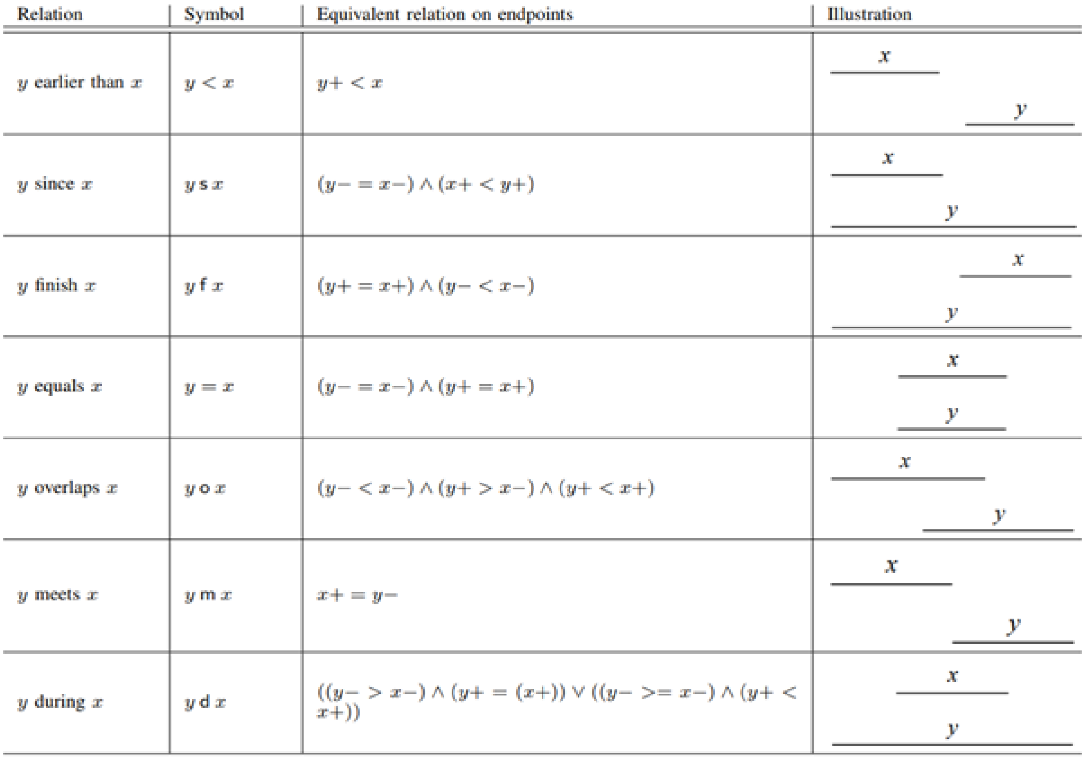
- an anti-affinity relation between resource R_1 and resource R_2 means that for two different tasks t_1 and t_2 belonging to the same manufacturing job and $t_1 < t_2$ with respect to the topological order of the manufacturing job, where R_1 is compatible with task t_1 and R_2 is compatible with resource R_2, if t_1 is allocated to R_1 then t_2 cannot be allocated to R_2.

5.2.1.8 *Sequence-dependent setup*

In batch scheduling problems, changeovers can be sequence (3). An example of such changeover in the considered OAS use case is the situation when in the same resource R_1 (e.g. mixer) two tasks t_1 and t_2 are to be processed one after another, each belonging to a different manufacturing job producing different commodity (e.g. different colour paint). In such situation, an additional task t_3 has to be processed by R_1 between t_1 and t_2 (e.g. cleaning of the mixer). In the IA, a new function has to be introduced that would take as parameters: a resource, the task currently processed by that resource, the task subsequently to be processed by that resource and return the task that has to be processed by R_1 between t_1 and t_2. This function returns empty task (i.e. task of processing time 0) if the sequence-dependent setup is not defined for given parameters.

5.2.1.9 *Intra-task relations*

In the original IA, only one relation between tasks was defined, namely a general precedence relationship. In order to apply IA to SAFIRE problems, more intra-tasks relations have to be considered. Allen in (4) introduced seven relations between intervals, as summarised in Figure 24. One of these relations, *y* meets *z*, can be used to describe immediate precedence relationships between tasks belonging to the same manufacturing job. To increase the genericity of the fitness function evaluation block, all relations from Allen's interval algebra are considered to be implemented in the prototype.



Relation	Symbol	Equivalent relation on endpoints	Illustration
y earlier than x	$y < x$	$y+ < x$	x y
y since x	y s x	$(y- = x-) \wedge (x+ < y+)$	x y
y finish x	y f x	$(y+ = x+) \wedge (y- < x-)$	x y
y equals x	$y = x$	$(y- = x-) \wedge (y+ = x+)$	x y
y overlaps x	y o x	$(y- < x-) \wedge (y+ > x-) \wedge (y+ < x+)$	x y
y meets x	y m x	$x+ = y-$	x y
y during x	y d x	$((y- > x-) \wedge (y+ = (x+)) \vee ((y- >= x-) \wedge (y+ < x+))$	x y

Figure 24: Binary relations in Allen's interval algebra

5.2.1.10 Simultaneous allocation to several resources

It is possible that a task has to be processed by more than one resource at the same time. It means that a simultaneous allocation to a few resources has to be added. The processing time of the task by all these resources is equal.

5.3 Manufacturing Process Planning and Scheduling in SAFIRE

Genetic algorithms have been selected to perform optimisation and reconfiguration in the SAFIRE project. The motivation behind this choice has been presented in deliverable D3.1 (1). Below, a brief problem formulation followed by a description of a general genetic algorithm (with unbounded execution time) and its custom modification (with bounded execution time) are provided.

5.3.1 Problem outline

The class of optimisation problems analysed in SAFIRE concerns mainly manufacturing plants, comprised of *k* resources (e.g. machines), possibly organised in some production lines. Some of these resources cannot be used simultaneously. In the considered plant, *m* recipes for manufacturing commodities have to be allocated to resources and scheduled. The value gained by an end-user from the optimisation depends on both solution quality and the time taken by the optimisation process itself. Since the optimisation process is performed in a cloud, the system architecture covers the problem domain model and the cloud configuration.

The schedulers used in SAFIRE are assumed to be *work-conserving*, which means that the scheduled resources cannot be idle when any recipe operation is ready to be scheduled at that resource. As a consequence of this assumption, there is no need to encode time offsets between recipe operation release time and starting time, which reduces the amount of information that needs to be provided to and from OE. Instead, each task's starting time can be inferred from its dependencies and priority.

Controlled metrics include one metric of the nominal type for each *recipe* to be allocated to resources. Each *recipe* is composed of one or more *recipe operations*. (Both recipe and recipe operations are introduced more formally later in this section.) If all *recipe operations* of a considered *recipe* are to be processed on the same resource, this resource may be provided explicitly to the OE and thus the *recipe* can be allocated to a concrete resource directly by OE. However, if *recipe* is to be processed by a set of devices, OE allocates recipe to a so-called *abstract resource*, which can be viewed as a set of resources unequivocally defining the underlying resources. Such *recipe-resource* mapping is then forwarded to the Fitness Function module (FF). In this module, *recipe operations* are identified for the *recipes* together with their resource affinity both stemming from the recipe and being compatible with the abstract resource assigned by OE. *Recipe operations* are allocated respectively. The difference of mapping between OE and FF is visualised in Figure 25.

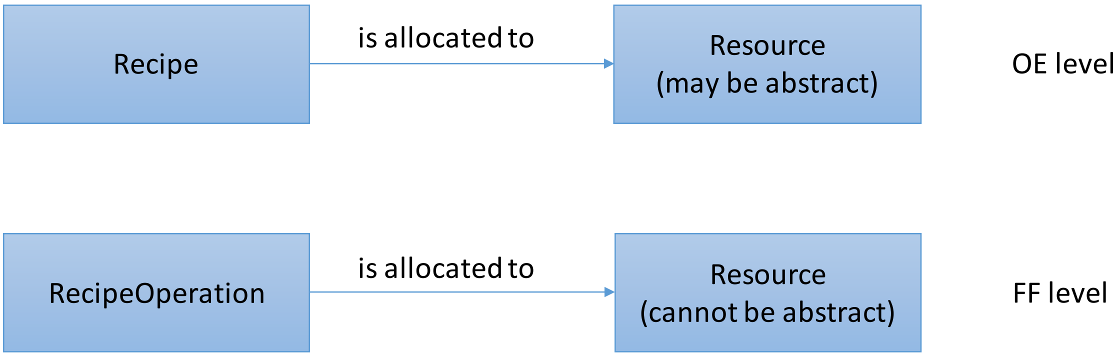


Figure 25: Resource mapping at OE and FF levels

5.3.2 Optimisation engine input definition

As stated in Deliverable D3.2 (5), ‘Metrics API’ has been developed in order to describe configuration of a considered optimisation problem, following metrics division into 3 categories: ‘Observable Metrics’; ‘Control Metrics’ and ‘Key Objective Metrics’, introduced in D1.2 (6). For self-sufficiency of this deliverable, the class diagram of ‘Metrics API’ is shown in Figure 26. The names shown in this diagram will be referred to in the following subsections.

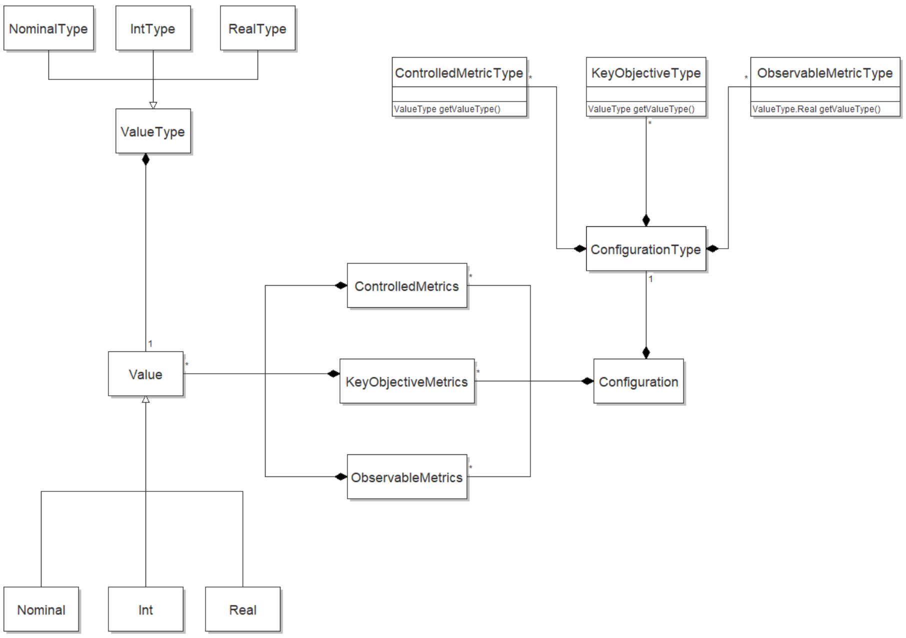


Figure 26: Class diagram of Metrics API

5.3.3 System model

Each taskset Γ includes a set of independent recipes γ_j, $j = 1, ..., n$. Recipe γ_j produces u_j units of certain commodity δ_l, $l = 1, ..., r$. Each j-th recipe is comprised of one or more recipe operations $\tau_{j,k}$, $k = 1, ..., \kappa_j$. Recipe operation $\tau_{j,k}$ can be executed by one of resources defined by a set $\Lambda_{j,k}$, including at least one resource $\pi_i \in \Pi$, , $i = 1, ..., m$. A recipe operation $\tau_{j,k}$ needs $t_{i,j,k}$ time units while executed on resource π_i, consuming $e_{i,j,k}$ units of energy and costing $c_{i,j,k}$ monetary units. A recipe operation $\tau_{j,k}$ has priority $p_{j,k}$ $\in \mathbf{N}_0$, ordered decreasingly (i.e. lower values denote a higher importance).

The plant is supposed to satisfy order O, comprised of o_l units of commodities $\delta_{l.}$ The difference between the actually produced amount of commodity $\delta_{l,}$ θ_l and the ordered amount of commodity δ_l , $o_{l,}$ is referred to as surplus and denoted with $\sigma_l = \theta_l - o_l$.

In the considered business case, a factory can produce r types of paints by executing a multisubset (i.e. a combination with repetitions) of Γ. Each recipe γ_j includes only one recipe operation $\tau_{j,1}$.

5.3.4 Problem formulation

Given a set of recipes Γ, a set of resources Π and an order O, the problem is to assign resources and priorities to a multisubset of recipe operations of recipes Γ so that the total processing time (makespan) is minimised and the amount of each manufactured commodity is higher or equal to the order, $\theta_l \geq o_l$, but the surpluses of each commodity, $\sigma_{l,}$ are minimised.

5.3.5 Proposed approach

Let us consider recipe γ_j producing u_j units of a certain commodity δ_l. To determine the upperbound on the number of this recipe in the recipe multisubset to be allocated to resources, the lowest number of the recipe execution leading to producing sufficient units of ordered commodity $\theta_l \geq o_l$ needs to be determined. This value can be determined with equation

$$\mu_j = \left\lceil \frac{o_l}{u_j} \right\rceil. \qquad (5.3.5.1)$$

Consequently, the cardinality of the multisubset is upperbounded with

$$\eta = \sum\nolimits_{j=1}^{n} \mu_j. \qquad (5.3.5.2)$$

The solution to the problem can be then described with a chromosome of length 2η, following the encoding proposed in Subsection 5.3.6.2. As the considered business case includes several objectives aiming at minimising the makespan and the surplus of each commodity, the multi-objective genetic algorithm techniques briefly described in Subsection 5.3.9, needs to be applied.

5.3.6 Genetic representation of metrics

In genetic algorithms, candidate solutions are treated as individuals. During the optimisation process, these individuals are evolved using a set of bio-inspired operations, described briefly in Subsection 5.3.7. In this section, individuals' encodings that facilitate the manufacturing process optimisation and reconfiguration are proposed.

5.3.6.1 Representation without Alternative Recipes

Since in the considered problems each metric assumes a value from a certain, predefined domain, so-called *value encoding* of chromosomes needs to be applied. This encoding, in contrast to e.g. the traditional binary encoding, allows each gene to directly correspond with a certain value of one variable of the optimisation problem and assume values from the domain of that variable only.

In case without alternative recipes, there is one to one correspondence between *orders* and *recipes* that has to be applied in order to manufacture a requested amount of *commodities*. The role of the OE module is to allocate the *recipes* to (possibly abstract) resources and schedule them in time. The encoding has hence to embrace both the spatial and temporal scheduling. Consequently, in the proposed encoding a chromosome contains genes of two types, as shown in Figure 27. For *n recipes* that need to be scheduled, the number of genes is thus equal to $2n$. The odd n genes indicate the target resource for *n recipes*, $G_{2x+1} \in \{\pi_1, \ldots, \pi_m\}$, whereas the remaining n genes specify the priorities of the *recipes*, $G_{2x} \in \mathbf{N}$, where $x=1, \ldots, n$. The priorities are ordered decreasingly, i.e. priority 0 is the highest. Such chromosome is then forwarded to FF module, where *recipe operations* are mapped to *resources* according to the chromosome values. *Recipe operations* belonging to a certain *recipe* inherit its priority. The aim of introducing priorities is to determine the processing orders of several *recipe operations* belonging to different *recipes*, but being allocated to the same resource and thus to determine the temporal scheduling. This type of encoding can be used for one of SAFIRE BC, specified by ONA.

R_1	ξ_1	R_2	ξ_2	…	R_n	ξ_n

Figure 27: Genes in a chromosome for manufacturing processes without alternative recipes

5.3.6.2 *Representation with alternative recipes*

The difference between the situations described in this and the previous subsections lies in the fact that in a representation with alternative recipes a certain multisubset (i.e. a combination with repetitions) of recipe set Γ needs to be applied rather than all recipes from this set in order to produce the required amount of *Commodities*. The maximal number of recipes that needs to be considered is upperbounded to a certain value η, as explained later in this document. At the encoding level, the novelty is that the odd η genes can indicate either the target resource for η *recipes* or the **rejection** of it, $G_{2x+1} \in \{\emptyset, \pi_1, \ldots, \pi_m\}$, where symbol $\emptyset$ is used for denoting the situation that certain *Recipe* has not been scheduled for execution. This encoding is presented inFigure 28.

R_1	ξ_1	R_2	ξ_2	…	R_η	ξ_η

Figure 28: Genes in a chromosome for manufacturing processes with alternative recipes

For simplicity, when alternative recipes are not allowed, symbols η and n are used interchangeable in this document. This encoding is applicable to two SAFIRE BCs, specified by business partners Electrolux and OAS.

5.3.7 Evolution-inspired operators

In a typical GA, selection, crossover and mutation operators are repetitively applied to a set of individuals. A selection operator is used for choosing a set of individuals from the current population to create individuals of the following population. The probability of an individual to be selected is proportional to its fitting, which is measured with a fitness function.

The individuals can be selected for recombination, and thus to generate further generations, using *fitness proportionate selection*, also known as roulette wheel selection. This operator associates fitness value of an x-th individual, $x = 1, \ldots, \eta$, denoted with g_x, with its probability of selection ρ_x according to equation

$$\rho_x = \frac{g_x}{\sum_{y=1}^{\eta} g_y}. \qquad (5.3.7.1)$$

Another popular individual selection operator is *deterministic binary tournament selection*. It involves performing several comparisons between two randomly selected individuals. The winner of such comparison, i.e. the individual with the highest fitness value, is selected to form future generations of individuals.

Crossover is a genetic operator that produces a new individual based on more than one individuals from a current generation. The *single-point crossover* combines two individuals, usually referred to as parents. A certain integer number $\boldsymbol{\nu}$ between 1 and η is randomly selected, where η is a number of genes in a chromosome. In the newly generated individual, genes from 1 to ν are taken from the first parent individual, whereas the remaining genes are taken from the second parent individual.

The role of mutation operator is to maintain genetic diversity of the population by performing relatively rare random changes of gene values. The mutation operator depends on the applied encoding. As chromosome used in SAFIRE introduces two value types: nominal for resources and integer for priority, two different mutation operators are needed. For metrics of nominal type, mutation overwrites the current value of the metric with another value from the domain of this metrics, selected randomly. This modification changes the allocation of the selected recipe to another compatible resource. In scenarios with alternative recipes, mutation can also overwrite a resource with rejection or vice versa. The integer values represent recipe priorities (and hence their relative ordering) and as such replace the value of the randomly selected gene with a chosen uniform random value from an application-specific range.

5.3.8 Single-objective genetic algorithm with unbounded execution time

The applied genetic algorithm (GA) uses the operators introduced in Subsection 5.3.7 according to the pseudo-code given in Figure 29. In the algorithm, the following two main steps can be singled out.

Step 1. Initial population generation (line 1). A predefined number of random *recipe* mappings (including the possibility of each recipe rejection in case of alternative recipe variant) together with their priorities are created.

Step 2. Creating a new population (lines 3-6). For each individual in the current population, values of the key objectives (line 3) are computed. Then the next generation is created by choosing the best individuals using a roulette-based selection. For each individual in the new generation, two parents are chosen with probabilities proportional to their key objective values (line 4). The selected parent individuals are then combined using a typical one-point crossover operation and mutated (lines 5-7). After these operations, a new population is formed from the child individuals (line 6). Step 2 is repeated in a loop as long as a termination condition is not fulfilled, which can be a maximal number of generated populations or lack of a significant improvement in a number of subsequent generations.

```
inputs:
            Resource set $\Pi$;
            Chromosome size $2\eta$;
            Population size $N$;
outputs:    Recipe mapping;
            Recipe priorities;

1   Generate an initial random population of N individuals with resource mappings (or recipe rejection) and priorities
2   while not termination condition do
3           Evaluate the key objective values of each individual;
4           Perform a fitness proportionate selection of parent individuals according to equation (5.3.7.1);
5           Generate N child individuals using single-point crossover;
6           Perform mutation of the odd $\eta$ genes (i.e. with resources/rejection) using values from the domain of this metrics (and $\emptyset$ in case of alternative recipes);
7           Perform mutation of the even $\eta$ genes (i.e. with priorities) overwriting value of the randomly selected genes with a chosen uniform random value from an application-specific range;
8           Create a new population with the child individuals;
end
```

Figure 29: Pseudo-code of single-objective genetic algorithm with unbounded execution time

5.3.9 Multi-objective genetic algorithm with unbounded execution time

SAFIRE BCs are characterised with multi-objective criteria. For example, not only resource utilisation needs to be maximised, but also the monetary cost of the applied manufacturing process has to be minimised, and in the case of process manufacturing with the presence of alternative recipes, the amount of manufactured commodities should be as close to the ordered amounts as possible to minimise the storage costs. The diversity of these criteria makes it difficult to convert such multi-objective optimisation problem into a single-objective weighted sum of these objective values. Depending on the current situation, some solutions with low weighted sum of objectives may not be acceptable due to, e.g., insufficient energy budget or storage space for a certain commodity. An end-user should be then informed about a wide set of Pareto-optimal solutions to select the final solution based on his/her knowledge of the problem. The set of the alternative solution presented to the end-user should be then diverse and, favourably, distributed over the entire Pareto-optimal region. This expectation is in line with the properties of the NSGA-II proposed by Deb et al in (7) as well as MOEA/D introduced in (8). In deliverable D3.1, NSGA-II, outlined in Figure 30, was selected to be used in SAFIRE.

```
inputs:
        Resource set $\Pi$;
        Population size $N$;
        Chromosome size $2\eta$;
outputs:
        Recipe mapping;
        Recipe priorities;

1  Generate an initial random population of individuals with resource mappings (or recipe rejection) and priorities
2  Evaluate the key objective values of each individual;
3  Assign ranks to the individuals;
4  Compute crowding distances for individuals of the same rank;
5  Sort the initial population based on rank non-domination criteria and crowding distance;

6  while not termination condition do
7        Select individuals using a deterministic binary tournament based on rank dominance and crowding-distance value;
8              Generate child individuals using single-point crossover;
9        Perform mutation of the odd $\eta$ genes (i.e. with resources/rejection) using values from the domain of this metrics (and $\emptyset$ in case of alternative recipes);
10       Perform mutation of the even $\eta$ genes (i.e. with priorities) overwriting value of the randomly selected genes with a chosen uniform random value from an application-specific range;
11       Evaluate the key objective values of each child individual;
12       Combine the child and the parent individuals;
13       Assign ranks to the individuals;
14       Compute crowding distances for individuals of the same rank;
15       Sort the combined individuals based on rank non-domination criteria and crowding-distance;
16       Create a new population by adding each front subsequently until the population size exceeds $N$;
end
```

Figure 30: Pseudo-code of multi-objective genetic algorithm with unbounded execution time

The multi-objective optimisation algorithm used in SAFIRE treats differently the initial generation (lines 1-5) and the remaining generations (lines 6-16), as detailed below. The algorithm performs the same crossover and mutation operators (lines 8-10) as the algorithm for the single-objective optimisation problems, presented in Subsection 5.3.8. Before the (tournament) selection (line 7), the population is ranked based on the individuals' non-domination (line 3 and 13). The non-dominant solutions are assigned with rank 1, which favours these individual in the selection process. The solutions that are dominated only by individuals of rank 1 are assigned with rank 2. The process of assigning the subsequent ranks is continued until all individuals are labelled. To spread the solutions on the whole Pareto-optimal range, so-called crowded distance of each individual is computed (line 4 and 14). The crowding-distance of an individual is based on the distance to the closest solution considering each objective separately and is computed for individuals with each rank separately. Then both the rank and the crowding distance are used as the sorting criteria (lines 5 and 15). The parent and child populations are merged to provide elitism of the algorithm (line 12). The new population is created out of the merged parent and child population by adding all the individuals with the same rank, starting from rank 1, as long as the number of the individuals in the population being generated plus the individuals of the subsequent rank is lower than the assumed population size. In this situation, the number of the individuals added to the new population is equal to the difference between the assumed population size and the number of the individuals already added to the new population, whereas the added individuals have the highest crowding distance among the individuals with the considered rank (line 16).

The above described algorithm is suitable for multi-objective optimisation but not for many-objective, that exist for example in the OAS BC. Hence, this algorithm has been replaced with MOEA/D as a default algorithm, yet it is still available as an alternative in the OE implementation. A pseudo-code of MOEA/D is presented in Figure 31. This algorithm is explained in details in (9).

```
inputs:
            Resource set $\Pi$;
            Population size $N$;
            Chromosome size $2\eta$;
            Neighbourhood size $T$;
            Uniform spread of $N$ weight vectors $\lambda^1,\lambda^2,...,\lambda^N$;
output:
            $EP$ (a set of recipe instance allocation and scheduling solutions);

1           Set EP = Ø
2           Generate $N$ random individuals with recipe instance allocations (or recipe instance
            rejection) and priorities as the initial population;
3           Evaluate the key objective values of each individual in the initial population;
4           Compute the Euclidean distances between any two weight vectors and find $T$ closest weight
            vectors to each weight vector. For each $i$= 1,...,$N$, set B(i) = $\{i_1,...,i_T\}$;
5           Initialise ideal points $z = (z_1,z_2,...,z_m)$ based on the objective values obtained
            from all individuals of the initial population;
6           while not termination condition do
7                   for $i$=1,...,$N$ do
8                           Randomly select two neighbours from $B(i)$, generate a new individual $y$
                            via genetic operators proposed in Subsection 5.3.7 to
                            the selected neighbours
9                           Evaluate the key objective values of $y$;
10                          For each $j$= 1,...,$m$, if $z_j$> $f_j(y)$, then set $z_j = f_j(y)$;
11                          For each $j \in B(i)$, set $x_j = y$ if $g^{te}(y|\lambda_j,z) \leq g^{te}(x_j|\lambda_j,z)$;
12                          Remove all individuals in $EP$ that are dominated by $y$
                            and add $y$ to $EP$ if no individuals dominate $y$.
13                  end
14                  Generate an elite individual employing the operator described
                    in Subsection 5.3.7, evaluate its objectives' values, and add it
                    to the current population (if eligible), update $z$ and $EP$.
15          end
16          return $EP$;
```

Figure 31: Pseudo-code of multi-objective genetic algorithm with unbounded execution time

5.3.10 Genetic algorithm with bounded execution time

The execution time of both single- and multi-objective variants of GA can be bounded, benefiting from the fact that in GA solutions are iteratively improved in subsequent generations and there is no universally agreed criteria for stropping this iterative process. Consequently, the optimisation can be finished after any iteration if timely execution is more important than possible further improvement of the solution quality.

In this section, each iteration of a GA is referred to as *stage*. In the *i*-th stage $i \in \{1, ..., \chi\}$, GA iterations are executed in parallel. Then the results are gathered and a stopping condition is checked, based on the prediction of the total value improvement in the subsequent stage, as illustrated in Figure 32.

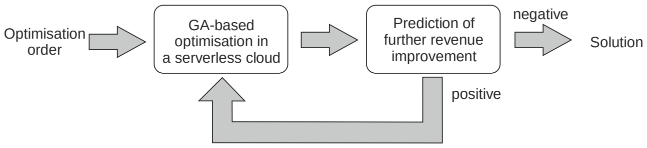

Optimisation order
GA-based optimisation in a serverless cloud
Prediction of further revenue improvement
negative
Solution
positive

Figure 32: Two major stages for genetic algorithm with bounded execution time

We now describe the parameterisation of the case study considered in this document.

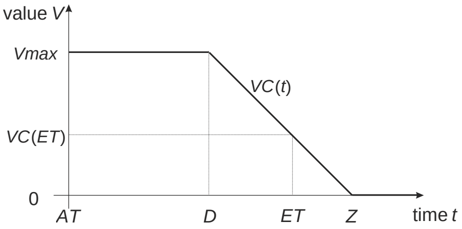


Figure 33: An example value curve of manufacturing order O

Each problem instance is parameterised as follows:

Input: manufacturing order *O* including: the plant given in the form of Activity-on-Arrows graph, its value curve *VC* and arrival time *AT*, a potentially unbounded number of slave processing nodes with (monetary) execution cost per time unit β and the number of individuals sent to each processing node.

Objective: Maximise the profit obtained from the manufacturing order.

The profit from the manufacturing order depends on the following factors:

- the fitness value returned by the GA,
- total processing time allocated to the optimisers, • cloud processing cost (per container invocation).

5.3.10.1 Value curve

The value curve models the value of a process to its end-user as a function of time, $VC(t)$. It may assume various shapes, as discussed in Burkimsher (10). In SAFIRE, the shape shown in Figure 28 has been chosen, which models generating the maximum value (e.g. as agreed in a contract) up to a certain deadline, after which a certain penalty is imposed every time unit. This shape can be intuitively explained as up to the deadline, the factory is occupied with other, previously configured manufacturing orders. So the deadline is the earliest time the factory can start manufacturing new products. Thus, there is no extra benefit in computing a new configuration well before the deadline, but after the deadline the factory becomes idle until a new configuration is found. As during this idle interval both relative overhead cost (ROC) and relative direct labour cost (RDLC) are incurred proportional to the idle time, the value of the solution decreases (11). Without any further modification of the proposed approach, this shape can be exchanged with any other non-increasing function if a curve better describing a certain process is identified.

The chosen value curve assumes positive values starting from the time of the manufacturing order arrival, AT. As in this section we consider only a single order scenario, without any loss of generality it may be assumed that AT = 0. The maximum value of $VC(t)$ is equal to V_{max} and is observed from AT to a certain deadline, $D > AT$. Finally, $VC(t)$ assumes zero value from zero value time, $Z > D$. This shape of the value curve can be described with the following equation

$$VC(t) = \begin{cases} V_{max} & \text{for } AT < t \leq D, \\ \dfrac{-V_{max}}{Z - D}(t - D) + V_{max} & \text{for } D < t \leq Z, \\ 0 & \text{for } t > Z. \end{cases} \tag{5.3.10.1}$$

5.3.10.2 Time and cost of stage execution

The optimisation is performed in stages until the applied stopping condition is satisfied. The stage index is denoted with i, $i \in \mathbf{N}$. During the i-th stage, the optimisation is performed on s_i slave nodes. As these nodes are possible to be executed in the FaaS manner, the monetary cost of using them is given by value β per second for each instance (for example, in IBM Cloud it was $0.000017 per second of execution, per GB of memory allocated on 21.01.2018). The maximal slave execution time in the i-th stage is equal to t_i. Thus the upperbound on cost of the execution of this stage for container c_i is given by:

$$c_i = \beta \cdot t_i \cdot s_i. \tag{5.3.10.2}$$

The cumulative cost of computing the first i iterations, C_i, is equal to:

$$C_i = \sum_{j=1}^{i} c_j \tag{5.3.10.3}$$

The predicted execution time of a stage, $\hat{t}_i$ is determined via the extrapolation mechanism described in Section 4.3. The manufacturing income yielded after the *i*-th iteration is a difference between the income given by value curve *VT* at the moment of completion the *i*-th stage and the manufacturing cost, described by fitness value f_i, i.e.

$$I_i = VC(T_i) - f_i \tag{5.3.10.4}$$

The profit generated after execution of the *i*-th stage is expressed as a difference between the income and the cumulative cost of the optimisation:

$$P_i = I_i - C_i \tag{5.3.10.5}$$

5.3.10.3 Value prediction

The values of t_i and f_i can be predicted via extrapolation. The extrapolation method used is the Bluirsch and Stoer algorithm (12), an extension of the well-known Neville interpolation/extrapolation algorithm to *diagonal* rational functions $p(x)/q(x)$ for polynomials *p*, *q* where *p* is of degree *r* (the length of the history vector from which to extrapolate) and the diagonal property requires that *q* is of degree *r* or *r* + 1, accordingly as *r* is even. In many cases, this method can be analytically shown to provide superior accuracy to more traditional methods of polynomial extrapolation (12). For history lengths of 3 or less, such extrapolation is either undefined or else the result was empirically determined to be inaccurate: the predicted value of f_i is then given by the best fitness found so far and that of t_i by the last (actual) processing time. After predicting the values $\hat{f}_i, \hat{t}_i$, they are used to predict the profit generated after the subsequent, (*i* + 1)-th stage as follows:

$$\hat{P}_{i+1} = VC(T_i + \hat{t}_{i+1}) - \hat{f}_{i+1} - C_i - \hat{c}_{i+1}. \tag{5.3.10.6}$$

This value can be used in a value-based stopping criterion, as described in the subsection below.

5.3.10.4 Stopping criteria

The stopping criteria are evaluated for a container at each stage *i*. We first apply an *absolute* criterion (ensuring that the process will eventually terminate) by comparing the *i* to a fixed upper bound on the number of stages (here, a value of 100 was empirically chosen). The *phenotypic convergence* criterion compares the Standard Deviation sd_i of the GA population against a threshold value (here, 0.02), similarly to e.g. Yin et al (13). The *predicted profit* criterion uses the method of diagonal rational extrapolation described above to predict whether the execution of the subsequent stage will not decrease the profit generated by the optimised process or not:

$$P_i > \hat{P}_{i+1}. \qquad (5.3.10.7)$$

The benefits of these stopping criteria have been evaluated in Deliverable D3.1 (1). More details about this approach can be found in (2).

5.3.11 Parallel execution of genetic algorithm

5.3.11.1 Introduction

The typical parallelisation of GAs can be performed either at the fitness evaluation or the population level (the island model), performed synchronously following the master-slave architecture (14). In clouds, these approaches are beneficial only under certain conditions, since the nodes are heterogeneous and connected with links characterised with different latencies. The fitness-evaluation level parallelism is beneficial only for expensive fitness functions (15), whereas the barrier applied in the island model is detrimental when the slave nodes are unreliable or have assorted response times (16). In SAFIRE, the approaches similar to evolutionary Peer-to-Peer (P2P) computing have been applied, described for example in (17). Following the principles of such approaches has been caused with the similarity between P2P and clouds with respect to the varied response time and nodes' unreliability. In SAFIRE, a custom multi-objective GA has been containerised using Docker to be deployed the containers in a Kubernetes[3] cluster. The islands communicate each other using a NoSQL database.

5.3.11.2 Asynchronous Island-based GA with Migrations

In the island model of GA, the evolution is performed independently on a number of subpopulations by GA instances named ”islands”. Aperiodically, the islands exchange individuals, so-called”migrants”. The traditional island model follows a fully synchronous master-slave architecture: the iterations on all islands begin at the same time, triggered by the master node, and the iteration completion is synchronised with a barrier. However, this approach can be modified to be fully distributed. In this section, the asynchronous island-mode GA is provided in Figure 34 with several migration strategies suggested.

Each island in the island mode of GA maintains its own subpopulation. It searches towards the optimal solution within a given number of execution stages, where each execution stage contains a fixed number of iterations. The optimisation engines run in each island are executed asynchronously and do not communicate directly with each other. Instead, they communicate using a light-weight database, pushing their selected solutions at certain time points. At other time points, the solutions pushed by other islands are popped and applied by an island to modify its current Pareto Front approximation. Similar to (18), a complete migration is performed by a selection and a replacement operator. The former selects the migrants to be pushed to a database and possibly later imported (popped) by other islands, whereas the latter operator selects the individuals in the Pareto Front approximation in an island that will be replaced by the migrants popped from a database so that the same population size is maintained during

[3] https://kubernetes.io/

the entire execution. In each island, the optimisation process stops after evolving a predefined number of generations.

Four strategies for the selection operator have been implemented, as enumerated below:

- *Generic selection* does not perform the actual selection from the current Pareto Front approximation but, instead, it randomly generates a new solution. This strategy serves as the performance baseline for the remaining selection operators.
- *Random selection* randomly selects a solution from the current Pareto Front approximation.
- *Best selection* selects the best solution from the current Pareto Front approximation. The solution quality is evaluated with the Generational Distance (GD) performance indicator from (19), which quantifies the proximity of a given solution to the ideal point.
- *Diversity selection* selects the solution with the highest diversity based on the Crowding Distance (CD) value (20), which measures the average distance between the solution and its two closest neighbours in the current Pareto Front approximation.

To maintain a fixed size of each island's population, a certain replacement operator is required to be applied during the migration. In SAFIRE, two replacement strategies have been implemented:

- *Random replacement* removes a randomly selected solution in the population of the target island.
- *Worst replacement* removes the worst solution in terms of the solution quality based on a certain quality indicator.

With the above selection and replacement operators combined, we provide, in total, eight migration strategies that can be pre-configured before the optimisation process.

```
inputs:
            I: number of iterations;
            P: number of individuals per island;
            S: number of stages;
            R: number of maximum stuck iterations in a row;
            M: number of solutions to migrate;
            CI: quality indicator

outputs:    PF: a Pareto Front (PF) approximation;

PF = ∅, s = 0, c = 0;
create a GA island with P randomly generated solutions;
for s=1,...,S do
        execute the GA islands for I iterations
        add non-dominated solutions returned into PF
        if CI value of PF obtained after stage s is not higher than that of stage (s-1) then
                increment c;
                if c==R then
                        c=0;
                        push the PF approximation to database;
                end
```

```
                        for m=1,...,M do
                                pull a PF approximation from a database;
                                migrate one solution from the remote set to the current population;
                        end
                end
        end
```

Figure 34: Pseudo-code of asynchronous island-based genetic algorithm

The algorithm starts with *P* randomly generated solutions and then executes for *S* stages, where each stage contains *I* iterations. After the GA island is executed in each stage, an approximation of Pareto Front, *PF* is updated with new non-dominated solutions (if there exist any). Then, a quality indicator is applied to check the quality of the current Pareto Front approximation and is compared to that of the approximation in the previous execution stage. (The choice of quality indicator applied in the algorithm is arbitrary, but it is assumed that a higher quality value indicates a higher quality of the optimisation result.) If the quality is not improved continuously over the prior *R* iterations (i.e., stuck in a local optimum), the Pareto Front approximation is *pushed* to the database by overriding the previous approximation set of this island (if it exists). In addition, after each execution stage that does not improve the Pareto Front approximation, a *pull* operation is performed to get solutions from a Pareto Front approximation from other islands, randomly selected, in the database (if there exits any). Then migrations are performed to migrate *M* solutions from the selected front to the current population based on a certain selection and replacement operators described previously. Lastly, the *PF* approximation is pushed to the database as the final optimisation result obtained by this GA island.

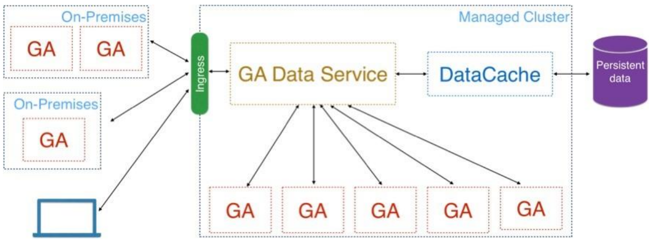


Figure 35: The architecture of the distributed island-based GA optimisation algorithm

5.3.11.3 Cloud Deployment

To deploy the algorithm presented in the previous subsection in a cloud environment, the architecture depicted in Figure 36 has been applied. It contains the following components:

- GA Data Service (data tier) is responsible for the data communication between islands and storing the data in a persistent data storage.

- Data Cache is used to reduce the response time when the data service reads/writes data from/to the persistent storage.

- GA Island executes the proposed GA; it can run either on a managed cluster or on-premise.

GA Data Service is highly available and automatically scale out/in according to the load of the requests from GA islands. Additionally, the data cache is designed to use a distributed key/value storage, such as a Redis[4] cluster or Cassandra[5], to support both high availability and fast data exchange.

The micro-service architecture employed by the proposed solution decouples the components so that the whole solution can be easily deployed to any distributed system. This enables this solution to be provided as a cloud service by cloud providers and requires the minimum possible maintains.

The deployment of the proposed architecture is based on the following assumptions:

- The number of islands that are running at the same time is up to hundreds.
- These islands issue requests to data-tier servers in a sporadic fashion, i.e., the requests (both sending data to or requesting data from the data-tier) arrive with a minimum interval, longer that the data-tier servers' response time.
- The amount of data exchange between the islands and the data-tier is relatively low, up to a few MBs in a single push/pop operation.

In the past several years, Docker and Kubernetes are two popular techniques for containerisation and container orchestration, respectively. Docker allows applications to be shipped to any popular operating systems by creating a Docker image that is similar to a virtual file system so that the application and its dependencies are encapsulated together. A Docker image is instantiated as a running container by the underlying execution-engine, such as *Docker Engine* or *containterd*. Kubernetes is a platform running on a computer cluster, and provide container orchestration functionalities, such as component abstraction (e.g., Pod, Service), DNS service, software-defined network, resource allocation, load balancing etc. Additionally, Kubernetes also provides Horizontal Pod Autoscaler (HPA) to dynamically auto-scale out/in the replicas of a service component based on several metrics, for example, the CPU or memory utilisation. The Cluster Autoscaler (CA) is used to dynamically adjust the number of computing nodes in a Kubernetes cluster. Lastly, Kubernetes allows different plugins to be installed. In the proposed deployment, we employ an ingress controller to allow users/applications to communicate with the data-tier service outside of the cluster.

[4] https://redis.io/

[5] http://cassandra.apache.org/

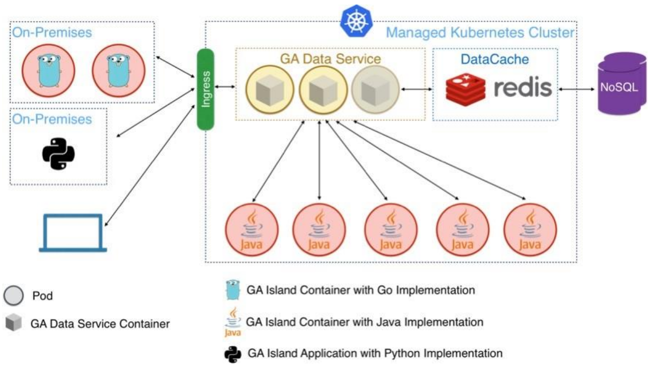

On-Premises
On-Premises
Managed Kubernetes Cluster
Ingress
GA Data Service
DataCache
redis
NoSQL
Java
Pod
GA Data Service Container
GA Island Container with Go Implementation
GA Island Container with Java Implementation
GA Island Application with Python Implementation

Figure 36: Architecture of the cloud-base manufacturing planning and scheduling optimisation system

Docker and Kubernetes have been adopted by many providers such as Amazon AWS, Microsoft Azure, Google Cloud, and IBM Cloud. This enables us to leverage the managed Kubernetes services from these cloud providers, rather than installed locally on premises. The Kubernetes HPA and CA enable autoscaling the components (such as the data-tier service) in our system based on the load. By creating multiple instances of service components, Kubernetes automatically handles the load balancing and re-starts a faulty container once detected. This approach enables the proposed system to be highly available during the operation.

Figure 36 depicts the deployment of the system described above. The core component is GA Data Service, which is responsible for data exchange between islands and also generating reports to users. It has a minimum number of instances by default to provide service high availability and scaling out/in according to the load of the requests. The islands can be implemented using any programming language, and communicate with the GA Data Service via REST API from within the cluster or outside of the cluster through the ingress. The GA Data Service stores all the data into an external NoSQL cluster and uses a Redis cluster as a cache layer.

The benefits of the above sketched algorithm have been evaluated in (21).

5.3.12 Specification of fitness function using Factory Description Language

As written in deliverable D3.5 (1), the primary way of specifying fitness function of a plant or smart thing is by describing its architecture, recipes available and tasks to be processed using an XML-based factory modelling language named Factory Description Language (FDL) and then using a software tool developed in the project to automatically generate the corresponding fitness function evaluator. Below, we discuss the most important components of the FDL language.

Element *objectives* includes a set of elements named *objective*, where each *objective* represents one objective of the optimisation problem. These objectives will be passed directly into the multi-criteria optimisation engine as the optimisation objectives. Below, the template for describing the objectives is presented.

```
<objectives>
    <objective name=”objective1” />
    <objective name=”objective2” />
    <objective
name=”objective3” />
</objectives>
```

Element *processingDevices* includes a set of elements named *processingDevice*, representing all processing resources (e.g. machines) in a plant. A *processingDevice* element requires the name attribute. As a resource can operate in a number of various operation modes, a *processingDevice* element includes the nested *modes* element, which in turn includes a set of *mode* elements with the mandatory *name argument*. Each resource has to include at least one mode.

```
<processingDevices>
    <processingDevice name= ”device1”>
      <modes>
        <mode name= ”mode1”/>
        <mode name= ”mode2”/>
        <mode name= ”mode3”/>
      </modes>
    </processingDevice>
</processingDevices>
```

The *productionLines* element describes all production lines in a factory, introduced as nested *productionLine* elements.

The name attribute in the *productionLine* element is mandatory. The *productionLine* element includes a nested *production-*

LineProcessingDevices element, which in turn includes nested *productionLineProcessingDevice* elements. Each *productionLineProcessingDevice* start-tag includes two attributes, *order* and *name*. The former attribute values are consecutive numbers that identify the resource order in a production line, whereas the latter attribute values have to be equal to the resource names introduced in element *processingDevice*. Each production line is linear and thus each possible split of processing results in creating a new production line, from the production line source to its sink. In the example below, the two production lines starts with the same resource (*Scale*), but as two routes are possible starting from *Converyor1* or *Conveyor2*, two *productionLine* elements starting from the *Scale* resource are generated.

```
<productionLines>
  <productionLine name=”ProductionLine1”>
    <productionLineProcessingDevices>
      <productionLineProcessingDevice order=”1” name=”device1”/>
      <productionLineProcessingDevice order=”2” name=”device2”/>
      <productionLineProcessingDevice order=”3” name=”device3”/>
      </productionLineProcessingDevices>
    </productionLines>
```

Element *productionProcesses* includes a set of production processes that need to be scheduled in the considered plant. Each *productionProcess* element, nested in *productionProcesses*, includes the mandatory *name* attribute and one or more alternative sets of *subprocesses* leading to manufacturing a certain commodity. Each *subprocess* element requires the *name* attribute and a set of nested *subprocessProcessingDevice* elements. Unique names of subprocesses are required to refer to them unambiguously from other elements, e.g. *sequenceDependentSetup* (explained later). If more than one *subprocessProcessingDevice* elements are provided, they are treated as alternative ones and being capable of producing the same commodity.

In the *subprocessProcessingDevices* element, all processing devices that have to be allocated simultaneously to execute the given subprocess are listed with elements *subprocessProcessingDevice*. The mandatory argument of this tag is *processingDeviceName*, whose value shall be found in the *processingDevice* element described earlier. Then *subprocessProcessingDevicesMode* elements follow with the mandatory *modeName* attribute whose value shall be listed into the corresponding *processingDevice* element, as described earlier. The *subprocessProcessingDevicesMode* element includes at least one of the three elements: *processingTime*, *energyConsumption* and *monetaryCost*. These three elements specify the corresponding numeric costs of using the particular processing device in the particular mode and as such can be later used to define a fitness function of a factory scheduling. The usage of these elements is demonstrated in the following example.

```
<productionProcesses>
  <productionProcess name=”production1”>
    <subprocesses>
      <subprocess name=”production1Task1”>
        <subprocessProcessingDevices>
          <subprocessProcessingDevice processingDeviceName=”device1”>
            <subprocessProcessingDeviceMode modeName=”mode1”>
              <processingTime>x1</processingTime>
              <energyConsumption>y1</energyConsumption>
              <monetaryCost>z1</monetaryCost>
            </subprocessProcessingDeviceMode>
          </subprocessProcessingDevice>
          <subprocessProcessingDevice processingDeviceName =”device1”>
            <subprocessProcessingDeviceMode modeName=”mode2”>
              <processingTime>x2</processingTime>
              <energyConsumption>y2</energyConsumption>
              <monetaryCost>z2</monetaryCost>
            </subprocessProcessingDeviceMode>
          </subprocessProcessingDevice>
        </subprocessProcessingDevices>
      </subprocess>
    </subprocesses>
  </productionProcess>
</productionProcesses>
```

Another element that is mandatory in a *productionProcess* element, as long as that element includes more than one *subprocess* element, is *subprocessRelations*, using *subprocessRelation* to describe relations between subprocesses in the considered *productionProcess*. Three arguments are mandatory: *source* and *destination* require a proper name of subprocess introduced in the considered *productionProcess*, whereas *allensOperator* requires any relation from the interval Allen’s algebra that describes the temporal relation between the source and the destination. The following *allensOperator* values are possible: *LT* for source earlier than destination, *S* for source since destination, *F* for finish destination, *EQ* for source equal to destination, *O* for source overlapping destination, *M* for source meeting destination and *D* for source during destination. Below, an FDL template for describing subprocess relations is presented.

```
<subprocessRelations>
  <subprocessRelation source=”Task1” destination =”Task2” allensOperator=”M”/>
  <subprocessRelation source=”Task2” destination =”Task3” allensOperator=”M”/>
  <subprocessRelation source=”Task3” destination =”Task4” allensOperator=”M”/>
```

```
   <subprocessRelation source=”Task4” destination =”Task5” allensOperator=”M”/>
</subprocessRelations>
```

Element *sequenceDependentSetup* determines extra costs when two certain subprocesses, specified with attributes *source* and *destination*, are performed subsequently using the same processing device, specified with attribute *processingDevice*. This extra cost can refer to time, energy or monetary cost, so three elements are provided: *extraProcessingTime*, *extraEnergyConsumption* and *extraMonetaryCost*, as shown in the following example.

```
<sequenceDependentSetups>
   <sequenceDependentSetup source=”commodity1Task1” destination=”commodity2Task1” processingDevice=”device1”>
      <extraProcessingTime>x1</extraProcessingTime>
      <extraEnergyConsumption>y1</extraEnergyConsumption>
      <extraMonetaryCost>z1</extraMonetaryCost>
   </sequenceDependentSetup>
</sequenceDependentSetups>
```

After specifying a scenario with FDL, the tool named Optimisation Engine Configurator (OEC) is used (OEC) is used to generate both the configuration template and the fitness function evaluator in the evaluator in the following way. Depending on the input factory parameter, OEC locates to the correct to the correct XML factory modelling file and reads the corresponding optimisation parameters and parameters and factory descriptions, which include optimisation objectives, factory resources with resources with their availability, production processes with their subprocesses, subprocess relations subprocess relations of subprocesses and dependent setups for production processes. This flow is This flow is illustrated in

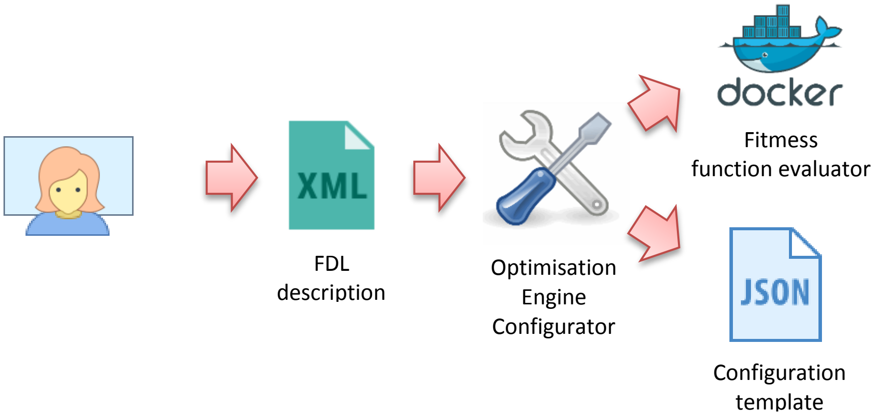


Figure 37. Examples of the FDL-based factory models are provided in deliverable D5.8 (22). More details regarding specifying fitness function using FDL can be found in (23).

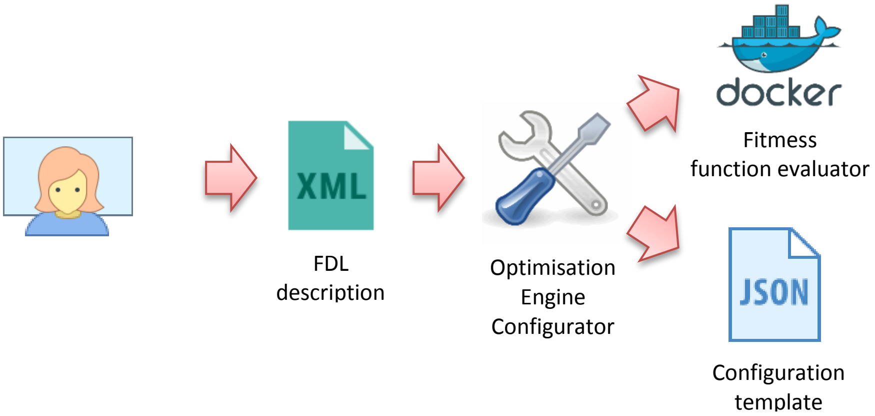


Figure 37: Optimisation Engine Configurator use flow

5.3.13 Evolving fitness functions using genetic programming

During the course of the SAFIRE project, some problems with formulating appropriate fitness functions by business partners have been observed. Finally, the fitness functions for all scenarios in SAFIRE have been defined using the technique described in subsection 5.3.12, but to facilitate this task, another possibility has been explored: semi-automatic evolving of fitness functions using grammatical evolution which is a popular genetic programing technique introduced in (24). The details of the grammatical evolution implementation and application in SAFIRE have been presented in (25). Utilising the proposed semi-automated formulation assistance, the overall work flow for a particular scheduling problem can be summarised as follows. First, end-users consider the system model and scheduling policies used, and determine a set of symbols and operators forming a context-free grammar that can be used to express analytical models that could potentially serve as fitness functions to evaluate solutions to the problem. Second, they obtain a set of verification vectors. Each verification vector represents a concrete system, and provides the parameter values for all of the entities that are scheduled in that system, as well as their indicative fitness values. The indicative fitness values are typically obtained via measurements taken from: (i) a real system, (ii) a cycle-accurate simulation of the system, or (iii) a simulation using an appropriate high level model. The grammar and the verification vectors are used as inputs into the formulation assistant. The formulation assistant uses an evolutionary algorithm to create *populations* of *candidate* analytical models that comply with the grammar. Each candidate analytical model is evaluated against the data for every entity in the set of verification vectors, resulting in a set of *computed* fitness. The *suitability* of the candidate analytical model is then determined by comparing the set of computed fitness that it produces with the set indicative fitness values. High suitability implies that the computed fitness values provide a tight upper bound on the indicative fitness values. The evolutionary algorithm creates subsequent *generations* of candidate analytical models by recombining and mutating candidates from the previous generation that are selected with a probability depending on their suitability. This selection pressure ensures that the overall suitability of the population increases over a number of generations, and the algorithm is able to find individual candidates with high suitability. The best candidate analytical models are returned as the output of the formulation assistant.

The aim of using a formulation assistant to provide suggestions for fitness functions is not to supplant end-users in this area, but rather to help them in finding effective fitness functions that can be explored in more detail. The overall processes is illustrated in Figure 38, which depicts end-users taking a system model and using it to create a grammar and a set of verification vectors that form the inputs to the formulation assistant. The formulation assistant produces candidate equations that can be checked and refined by the end-users.

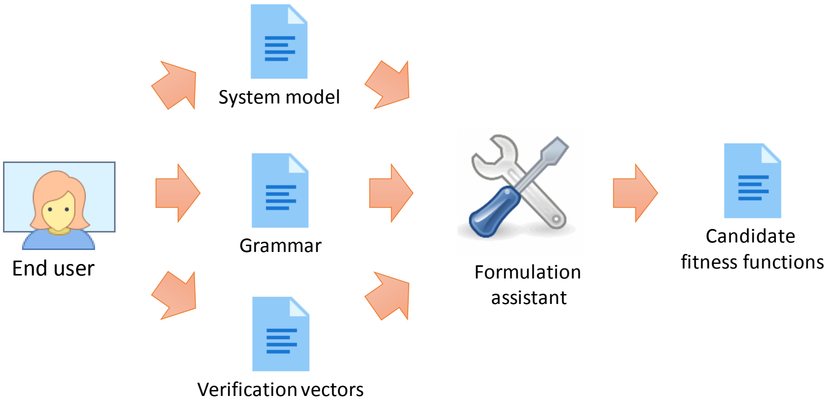


Figure 38: Overall process of deriving candidate fitness functions

6. METHODOLOGY FOR SECURITY, PRIVACY AND TRUST

We describe several aspects of methodology concerning the security, privacy and trust framework, moving from specific methods and tools to general considerations. Since the focus of our effort for SAFIRE has been the development of an implementation of the Next Generation Access Control (NGAC) standard (26) (27) (28) by The Open Group (TOG-NGAC), we devote most of our discussion of methodology to how it may be integrated with SAFIRE augmented systems, but also how it may be applied more broadly.

It would be worthwhile to mention a complicating factor for the security, privacy, and trust aspects of SAFIRE. SAFIRE is intended as an augmentation to existing manufacturing systems that should continue to operate normally when SAFIRE is “turned off”. However, SAFIRE’s security functions should be incorporated within the supporting infrastructure, requiring changes to the configuration of that infrastructure. In particular, existing manufacturing systems should be assumed to already incorporate security systems that have been selected or designed and implemented by ICT staff using an integration of commercial and bespoke components in a way consistent with each enterprise’s own ICT framework. On the one hand, the adoption of SAFIRE by an existing manufacturing system would be hampered by SAFIRE’s security features if it were to demand drastic changes to, or bypassing of, already implemented security functions, replacing them with prototype functions that would render the host system degraded or inoperable with the removal or deactivation of SAFIRE and its security mechanisms. On the other hand, the demonstration of novel security approaches cannot be accomplished in the context of an existing platform without modification to its “business as usual” configuration. Consequently, the adoption of SAFIRE’s security mechanisms does require modification and reconfiguration of existing enterprise and platform security features.

To mitigate the impact of this complication, we bound the application of SAFIRE's security mechanisms to uses where it affects primarily SAFIRE's operation. Though the SAFIRE mechanisms could beneficially be applied more broadly in the enterprise system, it requires commitment by the ICT organisation, and should only be undertaken with a future production-ready version of the SAFIRE implementation. We provide mechanisms that may have broad applicability, but that are initially used within SAFIRE in the least disruptive way to the existing infrastructure, while identifying additional potential future applications of the mechanisms. Initially, our TOG-NGAC implementation is applied only within SAFIRE. If the prospect of extending its use beyond this role is found to be attractive then it can be further integrated into the ICT environment for additional roles. We will discuss such potential applications in Section 6.3.7.

We begin with the methodologies that concern the application of our Next Generation Access Control implementation (TOG-NGAC), the development of attribute-based access control policies for use with its tools and enforcement mechanisms, the integration of TOG-NGAC with SAFIRE architecture and client applications, and the embedding of TOG-NGAC within distributed platforms. In this way we hope to provide a methodology that is applicable beyond the current demonstrations.

After the NGAC-related methodologies we proceed to methodology for broader issues: security problem identification and security requirements specification in a manufacturing system context, and the selection of appropriate security practices and mechanisms to address the identified security problems. At this stage we also describe some of the additional roles to which TOG-NGAC may be extended in an industrial system.

6.1 METHODOLOGY FOR THE DEVELOPMENT OF SECURITY POLICIES

6.1.1 Overview

NGAC is a novel approach to access control that affords unprecedented flexibility and the ability to represent and enforce arbitrary attribute-based access control policies within a unified framework.

According to the NGAC-FA Standard (26):

> *Next Generation Access Control (NGAC) is reinvention of traditional access control into a form that suits the needs of modern, distributed, interconnected enterprise. The NGAC framework is designed to be scalable, to support a wide range of access control policies, to enforce different types of policies simultaneously, to provide access control services for different types of resources, and remain manageable in the face of change.*

The attribute-based access control approach of NGAC subsumes diverse categories of access control policies, such as discretionary access control (DAC), multi-level security (MLS), role-based access control (RBAC), identity-based access control (IBAC), and others. NGAC has not, to our knowledge, been applied previously in industrial manufacturing or any Industrial Internet of Things (IIoT) environment.

The features and descriptions of previous NGAC reference implementations, and the published examples, suggest that NGAC has thus far been applied primarily to enterprise IT office automation systems. We have found the reference implementations to be complex, difficult to port, difficult to modify, and have excessive external dependencies. However, the scalability and flexibility of the NGAC *approach* make it a potent tool for addressing the scale, complexities, and policy challenges of combined IT and OT as found in IIoT systems. This has provided the motivation for us to pursue our own simpler and more portable implementation.

NGAC provides a framework and mechanisms that have the potential to unify the system-wide access control policies, and provide the ability to understand the net effect of the composed policies, and to provide a coherent approach to specifying access control policies, and configuring the underlying protection mechanisms that are relied upon to provide the isolation and architectural integrity to the NGAC implementation.

6.1.2 Guidelines

The following subsections first introduce the conceptual framework for NGAC policy specification, then describe the preliminary steps to the creation of a policy, then the steps to complete a policy, and finally the expression of the policy in the declarative policy language for use with the TOG-NGAC software.

6.1.2.1 NGAC Policy Framework

The fundamental concepts of the access control framework are:

- A set of basic elements – representing entities
- A set of containers of different types – to represent characteristics of basic elements
- A set of relations – to represent relationships among basic elements and containers

There is also a set of commands for the creation, deletion and maintenance of basic elements, containers and relations.

The basic elements comprise:

- Users – unique entities that are either humans, trusted programs, or devices
- Processes – system entities that have a reliable user identity and operate in a distinct memory
- Objects – resources to which access is controlled, e.g. files, messages, database records, etc.
- Operations – denote actions performed on elements of policy (either external protected resources or internal resources)

- access rights – enable actions to be performed on elements of policy (either external protected resources or internal resources)

Containers comprise:

- User attributes – defines membership on the basis of an abstract user capability or property. The members of a user attribute may be users or other user attributes. Membership is transitive.
- Object attributes – defines membership on the basis of an abstract object characteristic or property. Members of an object attribute may be objects or other object attributes. Membership is transitive.
- Policy classes – defines membership related to an access control policy, such as RBAC, MLS. Members of a policy class may be users, user attributes, object, or object attributes. Multiple policy classes may exist simultaneously.

Every user attribute, object attribute, and policy class has a unique identifier. Policies are expressed as configurations of relations of the following four types:

- Assignment – defines membership within containers, involves a pair of policy elements
- Association – defines authorized modes of access, it is a 3-tuple < *userattribute, accessrightset, attribute* >
- Prohibition – specifies a privilege exception
- Obligation – dynamically alters access state, triggered by an event

The version of the access control framework implemented by TOG-NGAC does not include Prohibitions or Obligations at the present time.

From the relation types above, these types of derived relations can be computed for the purpose of making access control decisions:

- Access control entry – derived from association; < user, accessright >
- Capability – derived from association; < accessright, policyelement >
- Privilege – derived from association; < user, accessright, policyelement >

The aforementioned commands for the creation, deletion and maintenance of basic elements, containers and relations are administrative in nature and exist outside of the policy expression framework, but they are central to policy automation and to the activity of incrementally building or modifying a policy. These commands are used internally within the software and exposed for use by administrative tools.

We now describe the methodology for using the NGAC policy framework to develop attribute-based access control policies. These are the technical policies, or policy models, that are created to enable technical protection mechanisms to be applied.

6.1.2.2 Policy Definition Preliminaries

Before a technical policy model can be created several activities should have already occurred. A policy, though declarative, presumes an operational semantics that is provided by the software that interprets and applies the policy. It is necessary to understand this semantics so the behaviour of the software under the control of the policy is predictable. The effect of a policy also depends on the ICT and physical environment to satisfy the assumptions that the policy and its animating software rely upon. These include supporting features of the platform software and hardware, the physical security of the facility that houses the hardware and network connections, and the properties of the communication lines over which networking is done. A high-level security analysis is presumed to have been conducted to identify the collection of resources to be protected (access controlled) and the users who are permitted to interact with the system and who are to be granted or denied access to resources. These topics are discussed in the later sections 6.2 and 6.3.

The necessary information should be compiled in order to complete the following steps to prepare for the development of an attribute-based access control policy:

- Create identifiers for the objects (resources) to be controlled by the policy. These are the assets to be protected that were identified in the security analysis.

- Create identifiers for the users (subjects) to be controlled by the policy. These are the active entities that were identified in the security analysis, which are to be granted or denied access to resources.

- Identify a set of attributes of objects (binary attributes, i.e. that indicate membership in a class named by the attribute) that are important to the organisation and relevant for the purpose of determining how objects should be handled. Create an identifier for each of these “object attributes”.

- Identify a set of attributes of users (again, binary attributes) that are important to the organisation for the purpose of determining the status of a user, such as a user’s responsibilities, that may contribute to determining what privileges should be granted to a user. Create an identifier for each of these “user attributes”.

A user is contained in a user attribute and an object is contained in an object attribute. Since attributes may designate broad or narrow categories, a narrow category may be contained in a broader category. Thus, an attribute may “contain” another attribute. Attributes should be thought of as “containers”. Objects will be assigned to (contained in) object attributes and users will be assigned to user attributes. Object attributes may in turn be contained within other broader object attributes. The same will be true for user attributes. The choice of attributes is arbitrary as far as the framework and the enforcement mechanisms is concerned, but the attributes should capture all of the factors that are relevant to making decisions about protection and access regardless of what the particular objects and users may do.

6.1.2.3 Policy Creation

The next step to create a policy specification is to organize objects and users each within appropriate attribute containers. The *assignment relation*, introduced in the NGAC policy framework Section 6.1.2.1, is used to place a user or a user attribute into a (different) user attribute container, or to place an object or object attribute into another object attribute container.

Since attributes may contain other attributes, this structure forms a (inverted) hierarchy (or non-inverted tree) of object attributes and user attributes respectively, with users appearing as leaves at the top of a user attribute tree and objects appearing as leaves at the top of an object attribute tree.

The concept of attributes as containers, for example, user attributes that contain users, rather than being something that is attached to a user, may be the biggest conceptual adjustment to which a user of the access control framework must adapt. When an entity is contained in another (attribute) entity we say that it is *assigned* to the attribute. We may also say that the former entity is an ascendant of the latter (attribute) entity (since our attribute trees are right-side up, not inverted).

Now that we have created rich trees of attributes for users and for objects, and appropriately allocated the users and objects to those attributes, it is time to create the relations among user and object attributes that will create permissions for users to perform operations on objects. These will be the *association relations* introduced in the NGAC policy framework. Recall that an association is a 3-tuple consisting of a user attribute, a set of access rights, and an object attribute.

To create appropriate associations in our policy graph we may ask the question, “If a given user is to have a certain access right for a given object, what attribute of the user and what attribute of the object are those that best characterise the justification for the permission within the hierarchical scheme of attributes?” The corresponding access right should be added to the set of access rights in the association between the user attribute and the object attribute. If we ask the question again for the same user and object but for a different access right, we may find that the same attributes apply, in which case the new access right would be added to the set of access rights in the existing association, or, if different attributes apply the access right would be added to a new association between the newly identified pair of attributes.

Another approach would be to take user attribute and object attribute pairs and determine what set of access rights (if any) should apply based on the interpretation of the attributes. By a careful and appropriate assignment of users to user attributes and objects to object attributes each user would gain appropriate permissions to each object by virtue of the associations already created. Remember that an association applies to all of the ascendants of the included user attribute and object attribute.

6.1.2.4 Policy Representations

Security policies, as well as the definition of the subjects and objects of the access control system and their attributes, are readily illustrated in the form of a directed graph. In the foregoing section one is invited to think of the relations in graphical terms. This graph is an external rendering of the model constructed and used in the TOG-NGAC Policy Tool and the Policy Server. It may be particularly convenient for developing a policy concept and for illustrating simple policies but the graphical representation may become unwieldy for complex policies. Still it is only a drawing and not something that can be used by the software. A representation of a policy that may readily be processed by logic is what is needed for automation of policy enforcement. Such a representation is specified in the User Manual and discussed in Section 6.1.3.1.

6.1.3 Algorithms, technologies and tools

6.1.3.1 The Open Group's NGAC Implementation

The Open Group has developed its own implementation of NGAC software (TOG-NGAC), including a Policy Tool and Policy Server that use a straightforward declarative language to express policies that comply with the NGAC framework. We first discuss this language and its use. Later we describe the software components of TOG-NGAC.

Our version of the NGAC functional architecture is shown in Figure 39. All of the components coloured in blue are part of the NGAC subsystem. We "unbundle" the PEP and RAP components from the rest of the implementation to place control of these components into the hands of the Client Application developer as we discuss further in Section 6.2, where we describe the methodology for the integration of NGAC mechanisms.

A policy expressed in the declarative policy language can be read by both the Policy Tool and the Policy Server, and both components incorporate the logic to make decisions based on the imported policy. The Policy Tool is used offline to develop a policy configuration to be used with the Policy Server in a production setting.

NGAC declarative policies are contained in ordinary text files. The 'ngac' Policy Tool may be used in conjunction with any text editor to develop and test a policy in the declarative language.

The Policy Server provides a RESTful interface for the Policy Query Interface and the Policy Administration Interface. The administration interface is used to load and manage policies in the server; the query interface is used to obtain policy decisions.

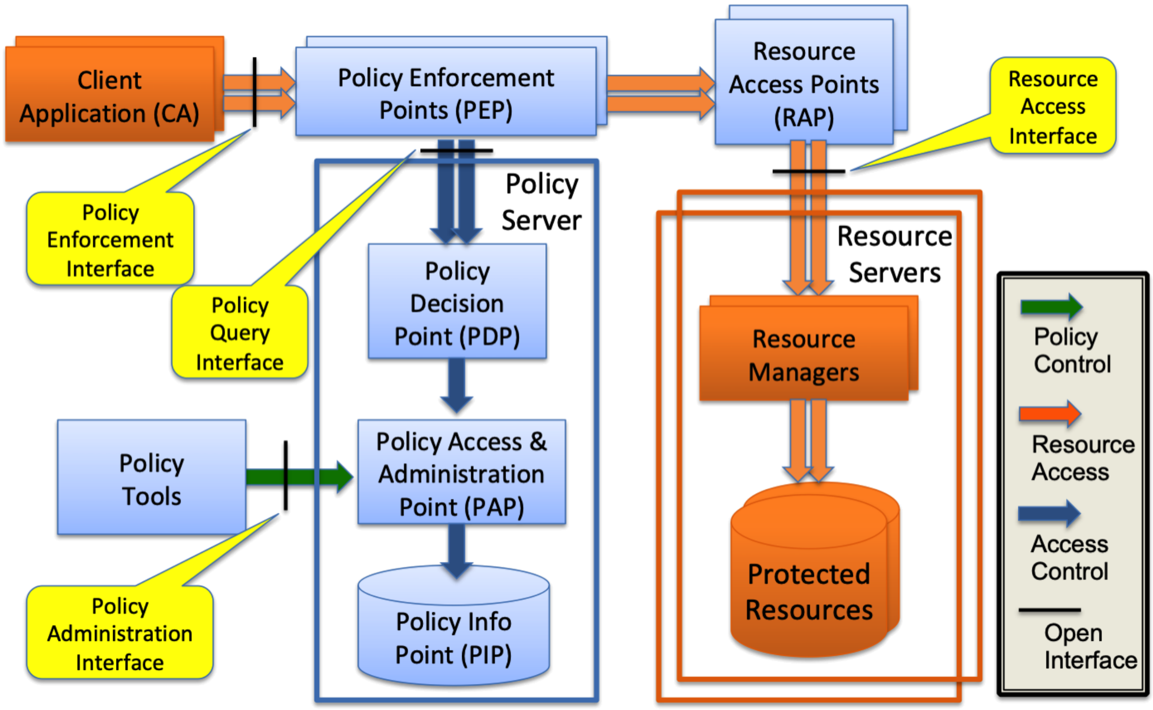

Client Application (CA)
Policy Enforcement Points (PEP)
Resource Access Points (RAP)
Resource Access Interface
Policy Enforcement Interface
Policy Server
Policy Query Interface
Policy Decision Point (PDP)
Resource Servers
Resource Managers
Policy Tools
Policy Access & Administration Point (PAP)
Protected Resources
Policy Administration Interface
Policy Info Point (PIP)
Policy Control
Resource Access
Access Control
Open Interface

Figure 39: NGAC Functional Architecture with "unbundled" PEP/RAP

The declarative language representation of a policy is easily constructed from a graphical representation of the policy, which may be used informally for its conceptualisation and development. The present declarative language for SAFIRE is based on the NGAC policy framework but it does not include prohibitions and obligations, though these may be added incrementally in the future for cases that need them.

The specification and implementation description of TOG-NGAC are given in D5.4 Full Prototype of the SPT Framework, D5.5 Final Specification of the SPT Framework, and D5.8 Final Integrated Cloud Platform. The D5.8 document also contains instructions for using the 'ngac' Policy Tool to develop and test policies. We further discuss the use of the Declarative Policy Language, the Policy Tool and the Policy Server in the following sections.

6.1.3.2 Declarative Policy Specification

The Declarative Policy Language developed by The Open Group is described in the Security Framework section of the User Manual of the SAFIRE Platform in D5.8. The declarative policy language representation is easily constructed from a graphical representation of a policy if one was created as vehicle for the policy design. Every identified user, object, user attribute, and object attribute has an identifier that is first declared by the following elements: `user( <identifier> )`, `object( <identifier> )`, `user_attribute( <identifier> )`, and `object_attribute( <identifier> )`.

Object declarations have two forms, a short form with a single argument and a long form with seven arguments. The short form only declares an identifier as being the object identifier. The long form adds arguments for an object class, which is like an object type, and other arguments that contain information about the location of the actual object. The short form is provided since the object identifier is all that is required to respond to an `access( )` request, and is therefore sufficient for the development and testing of a policy. The long form, provides object location metadata, that is returned in response to a getobjectinfo( ) request. This API may be used by a PEP and/or RAP to obtain the type and location metadata, which the PEP/RAP may need in order to carry out an operation that was approved by an `access( )` request. Furthermore, the location of the object may not be known at the time a policy is written. A short form used in a policy specification for development and testing can later be replaced by the long form of the `object( )` element either by editing the policy file, or by dynamic incremental element replacement after the policy file is loaded, using `add( )` and `delete( )` administrative operations.

Similarly, there are elements to declare identifiers for operations, object classes, policy classes, and a distinguished element called the connector. The object class declaration `object_class( <object class identifier>, <operations> )` supplies a list of operations that may be used on any object of the declared object class. The connector, by convention, always has the identifier 'PM'.

After all the elements have been declared, the assignments may easily be read from the graphical representation; these are the arrows indicating membership of an entity in the container represented by another entity. Assignments are encoded as `assign( <entity identifier>, <entity identifier> ).` We say <entity identifier> here because assignments can be made not only from a user to a user attribute, a user attribute to a user attribute, an object to an object attribute, and an object attribute to an object attribute, but also from a user attribute or an object attribute to a policy class, and from a policy class to the connector. In fact, any element may be assigned to the connector if it is not assigned somewhere else, since the purpose of the connector is to provide connectivity of any element in the graph, so there are no disconnected elements.

Finally, the associations that cross between the user attribute and object attribute trees are read from the graph along with the corresponding set of access rights. These then comprise the elements of the declarative policy. A complete declarative policy is contained in a declaration of the form, `policy( <policy name>, <policy root>, <policy elements> ).` where, policy root is the identifier of a policy class appearing among the policy elements, and policy elements is a square-bracketed, comma-separated list of the policy elements described above that make up the policy.

For the graphical policy in Figure 40, the declarative policy representation is shown in Figure 41. In the graph we've used the convention of placing users in the top left, objects in the top right, and user and object attributes below the users and objects respectively. The arrows represent assignment and the dotted lines associations.

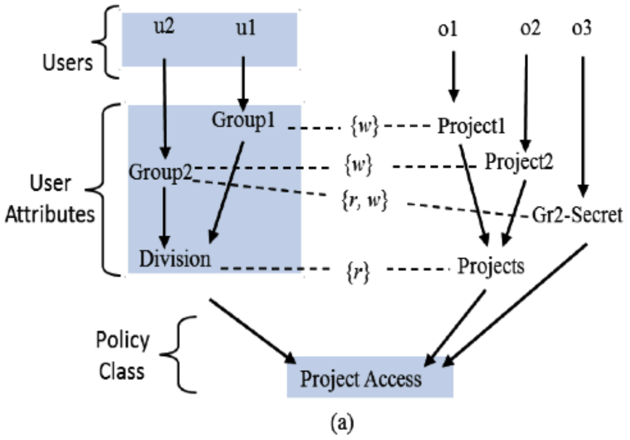


Figure 40: Example Policy (a) in graphical form

```
policy('Policy (a)','Project Access', [
    user('u1'),
    user('u2'),
    user_attribute('Group1'),
    user_attribute('Group2'),
    user_attribute('Division'),
    object('o1'),
    object('o2'),
    object('o3'),
    object_attribute('Project1'),
    object_attribute('Project2'),
    object_attribute('Gr2-Secret'),
    object_attribute('Projects'),
    policy_class('Project Access'),
    connector('PM'),
    assign('u1','Group1'),
    assign('u2','Group2'),
    assign('Group1','Division'),
    assign('Group2','Division'),
    assign('o1','Project1'),
    assign('o2','Project2'),
    assign('o3','Gr2-Secret'),
    assign('Project1','Projects'),
    assign('Project2','Projects'),
    assign('Division','Project Access'),
    assign('Projects','Project Access'),
    assign('Gr2-Secret','Project Access'),
    assign('ProjectAccess','PM'),
    associate('Group1',[w],'Project1'),
    associate('Group2',[w],'Project2'),
    associate('Group2',[r,w],'Gr2-Secret'),
    associate('Division',[r],'Projects') ] ).
```

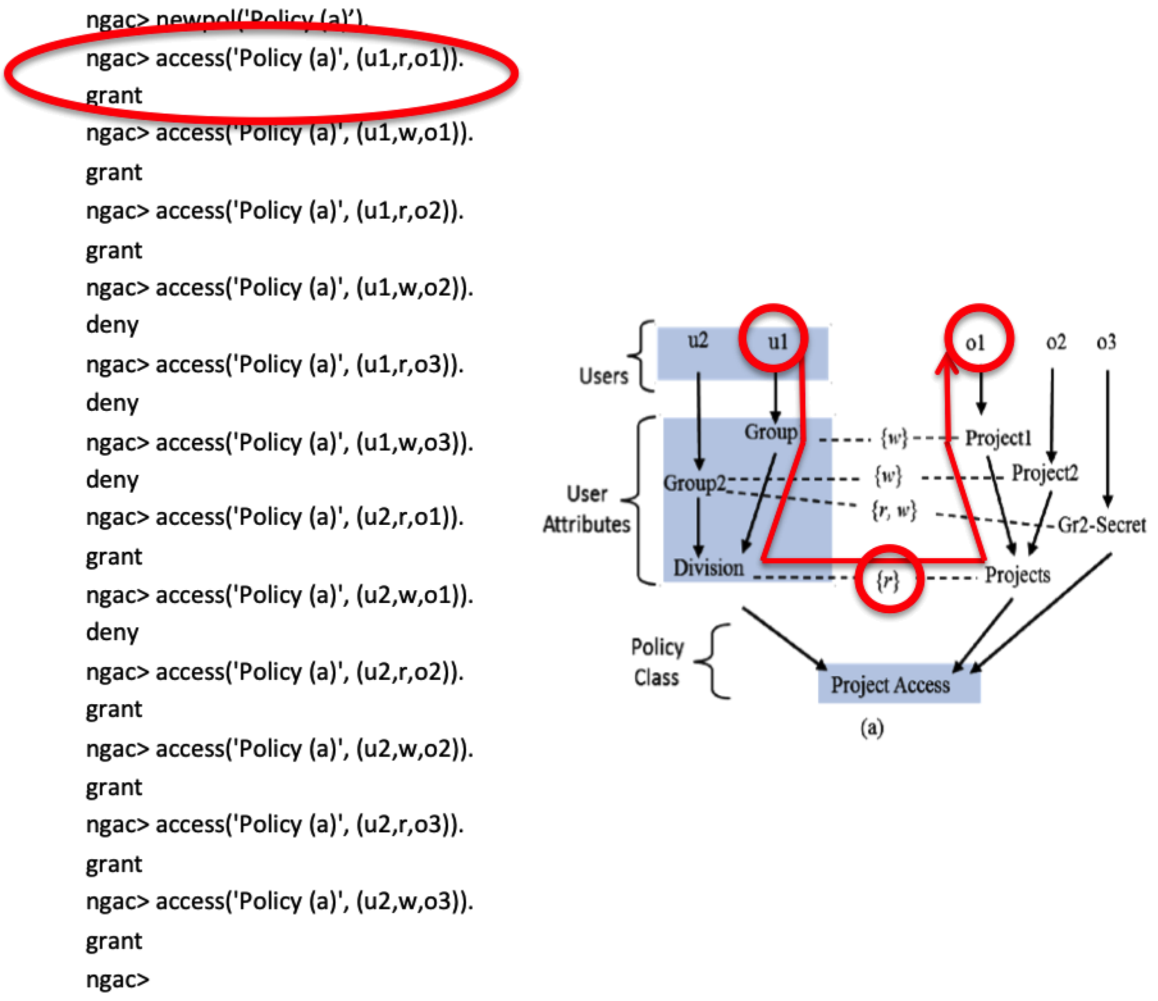
Figure 41: Example Policy (a) in the declarative policy language

6.1.3.3 Policy Development and Testing

The TOG-NGAC implementation provides a desktop command-line Policy Tool called ‘ngac’ that can load policies expressed in the declarative policy language and can answer queries such as `access(‘Policy (a)’,(u1,r,o1))`, the meaning of which is: “under the policy named ‘Policy (a)’, is user ‘u1’ allowed to read object ‘o1’?”. Policies are contained in text files that may be created with any text editor.

While editing a policy in a text editor window the ‘ngac’ tool can be started in another window. The policy file should be saved from the text editor and then read with the `import_policy( )` command of the ‘ngac’ tool. A full list of commands available in the Policy Tool are documented in D5.8. These commands include importing of policy files, setting an imported policy as the current policy, and querying the policy with access commands. By querying the policy the user may ascertain that the interpretation produced by the policy algorithms operating on the internalized policy model correspond to the expected interpretation and that the policy has been correctly coded. The edit-save-import-query cycle may be repeated as necessary until the user is satisfied with the policy.

Figure 41 illustrates the ‘ngac’ Policy Tool being used to query the example policy of Figure 40. The circled query, which returns “grant”, succeeds because of the existence of the red-highlighted path in the accompanying policy graph.



```
ngac> newpol('Policy (a)').
ngac> access('Policy (a)', (u1,r,o1)).
grant
ngac> access('Policy (a)', (u1,w,o1)).
grant
ngac> access('Policy (a)', (u1,r,o2)).
grant
ngac> access('Policy (a)', (u1,w,o2)).
deny
ngac> access('Policy (a)', (u1,r,o3)).
deny
ngac> access('Policy (a)', (u1,w,o3)).
deny
ngac> access('Policy (a)', (u2,r,o1)).
grant
ngac> access('Policy (a)', (u2,w,o1)).
deny
ngac> access('Policy (a)', (u2,r,o2)).
grant
ngac> access('Policy (a)', (u2,w,o2)).
grant
ngac> access('Policy (a)', (u2,r,o3)).
grant
ngac> access('Policy (a)', (u2,w,o3)).
grant
ngac>
```

Figure 42: 'ngac' tool queries against example Policy (a)

When the user is satisfied that the policy is complete and produces the desired behaviour, the last version of the policy file should be retained for loading into the Policy Server for production use.

6.1.3.4 *Policy Enforcement Mechanisms*

The central component of the NGAC policy enforcement scheme is the Policy Server, which contains a Policy Decision Point (PDP) to answer access queries against a loaded policy that has been selected to be the *current* policy. The Policy Server provides a Policy Query Interface to utilise the services of the PDP, and a Policy Administration Interface to manage policies within the server through the Policy Administration Point (PAP).

The effect of NGAC policy enforcement is achieved by a collection of cooperating components that are identified and organised according to an architecture introduced by the NGAC Functional Architecture.

Our version of the NGAC functional architecture, the roles of its components, and the actions needed to integrate NGAC into an application environment are further discussed in Section 6.2.

6.1.3.5 Other NGAC Implementations

There have been several reference implementations (RI) of NGAC, mostly under the name "Policy Machine" (PM), that have been developed over several years by some of the authors of the NGAC standard.

One reference implementation of the NGAC standard consists of components that are implemented in Java and communicate over SSL sockets. It was developed to run on Windows Server, with LDAP, Active Directory, Certificate Authority, and other system and network services. A newer version relaxes the need for Windows Server, LDAP and AD. Considerable support is still required from the IT environment in which the PM operates. In place of LDAP/AD a newer PM reference implementation used a MySQL database for policy storage, and later, as an option, a Neo4j[6] database, which is better suited to policies in the NGAC framework. As more recent prototypes had emerged, they seemed to have focused on narrower aspects of NGAC functionality, rather than a complete system.

Early reference implementations of NGAC had incomplete or out-of-date documentation. Later releases had even less documentation that we had seen with early versions. The troublesome characteristics of the reference implementations caused us to consider whether we should create an implementation with more favourable characteristics that we could control.

By the time the SAFIRE project was underway we had come to the conclusion that we could not depend on other parties to produce future versions of NGAC RIs that would be aligned with our evolving needs. Based on the favourable outcome of a feasibility study conducted on another EC project, we proceeded with our own implementation of NGAC.

Ultimately, our efforts have produced a pleasing result: TOG-NGAC is simple, portable, extensible, and lightweight, having no external dependencies. It is being exploited and extended in other projects.

6.2 Methodology for the Integration of NGAC Mechanisms

6.2.1 Overview

NGAC, as a *Reference Monitor*[7], is inserted as a mediator between resource-using applications and the resources used, enabling the enforcement of an access control policy.

[6] Neo4j is a graph database that is available in community and commercial versions.

[7] Technically, it is a *Reference Validation Mechanism*, an *implementation* of the Reference Monitor *concept*.

An application is often initially developed without consideration of access control, implemented to access resources using an interface provided by the resource itself or by the system infrastructure. This situation is illustrated in Figure 42. In order to introduce policy control over the application's access to the resource, a reference monitor, such as NGAC, is inserted into the resource access path as illustrated in Figure 43.

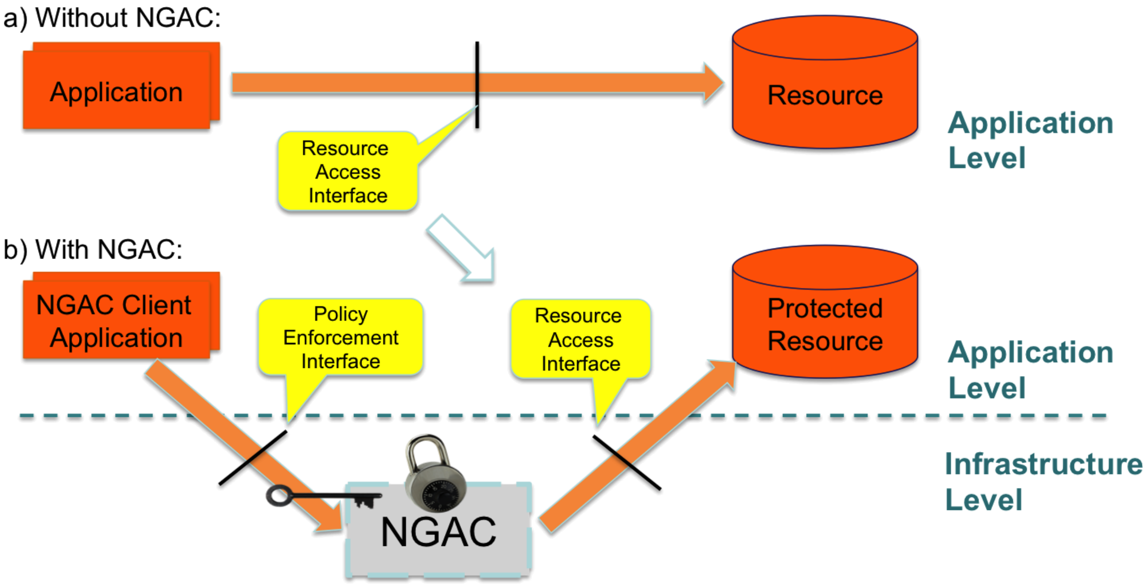


Figure 43: Inserting NGAC into the Resource Access Path

NGAC effectively "locks up" the resource as it mediates the path to the protected resource. NGAC presents to the Client Application (CA) a Policy Enforcement Interface, which is now the entry point to the *only path available* for accessing the protected resource, because the reference monitor is placed within the infrastructure level and given exclusive access to the Resource Access Interface. The CA must now have authorisation in order to perform an operation on the protected resource, indicated in Figure 43 as a key for the lock.

A refinement of this simple illustration of NGAC is presented in Figure 44, showing how the functional components of NGAC are arranged to achieve mediation of the resource access path. The developer of an application that will use resources protected by NGAC needs to perform the following steps to integrate the application with NGAC:

- Modify the application to use the NGAC resource access path by calling a Policy Enforcement Interface instead of directly calling the Resource Access Interface to perform operations on protected resources.

- Develop a Policy Enforcement Point (PEP) and a Resource Access Point (RAP) to complete the Resource Access Path as represented by the orange arrows in Figure 43 and Figure 44, if they do not already exist for the kind of resource involved, The PEP must consult the Policy Server for a policy decision, and, if permission is granted, perform the resource access through the RAP, which in turn uses the original Resource Access Interface.

- Test the function of the PEP and RAP for both “grant” and “deny” responses from the Policy Server. In the “grant” case the resource access path should function just as though NGAC were not present. The Policy Server provides a feature to facilitate such testing.

- Create and test an appropriate access control policy, using the ‘ngac’ Policy Tool, to govern all users and objects,

- Isolate the protected resource and install the PEP and RAP in the infrastructure so that they run with the privilege to access the isolated resource (a system administrator may need to assist).

- Load the policy into the Policy Server for final policy testing and for production use.

Further details of policy development, PEP, RAP, PDP interaction, and integration of the NGAC components into the platform infrastructure are provided in Section 6.2.2.

6.2.2 Guidelines

We now discuss in more detail some of the steps introduced above.

6.2.2.1 Adapting a Client Application to use the NGAC Resource Access Path

The PEP and the RAP are “unbundled” from the TOG-NGAC core implementation. In this way the development efforts and code bases of the Client Applications (CA) can be decoupled from those of the NGAC core. By unbundling the PEP and RAP the details of the Policy Enforcement Interface are left to the discretion of the application developer, and the protected resources can be accessed at a level of abstraction chosen to be appropriate by the developer.

Revisiting Figure 43, observe the differences between the unmediated access scenario and the NGAC-mediated access scenario. Suppose an application has been developed to use resources directly with no access control mediation. The application accesses the resource through an interface provided by the resource server[8]. When NGAC mediation is introduced, the code fragment (or a similar code fragment) is reproduced within the logical perimeter of NGAC (in the RAP) to use the existing resource access interface. The resource-accessing code in the Application (now an NGAC Client Application) is replaced to use an NGAC-provided Policy Enforcement Interface to access the now Protected Resource. If the policy enforcement interface provided by NGAC is not well suited both to the Application and the kind of resource, e.g. if the interface is fixed for all resource kinds, this conversion activity can be disruptive to the application’s development. It may possibly even require restructuring or refactoring of the application (such as when object-oriented constructs are used to encapsulate the implementation of

[8] For simplicity, the resource server is implied in this figure. It is the agent that actually provides the Resource Access Interface and provides the defined operations on the resource and manages the hidden data of the resource. The resource server is an abstract data type manager. In the case of an ordinary file the resource manager is the operating systems file system software that manages a raw storage device and provides an abstraction of files and directories, and provides the file access APIs.

external objects). To avoid such difficulties, the implementation of TOG-NGAC takes an approach whereby the PEP and RAP are factored-out from the rest of the implementation as shown in the simplified functional architecture depicted in Figure 44. The core TOG-NGAC implementation consists of the ‘ngac’ Policy Tool and the ‘ngac-server’ Policy Server.

The Client Application developer must create the PEP and the RAP to complete the resource access path (shown in orange) if an appropriate PEP and RAP do not already exist for the resource. However, instead of the developer having to adapt to a Policy Enforcement Interface that is fixed for all resource kinds, the developer is free to define the Policy Enforcement Interface when implementing the PEP.

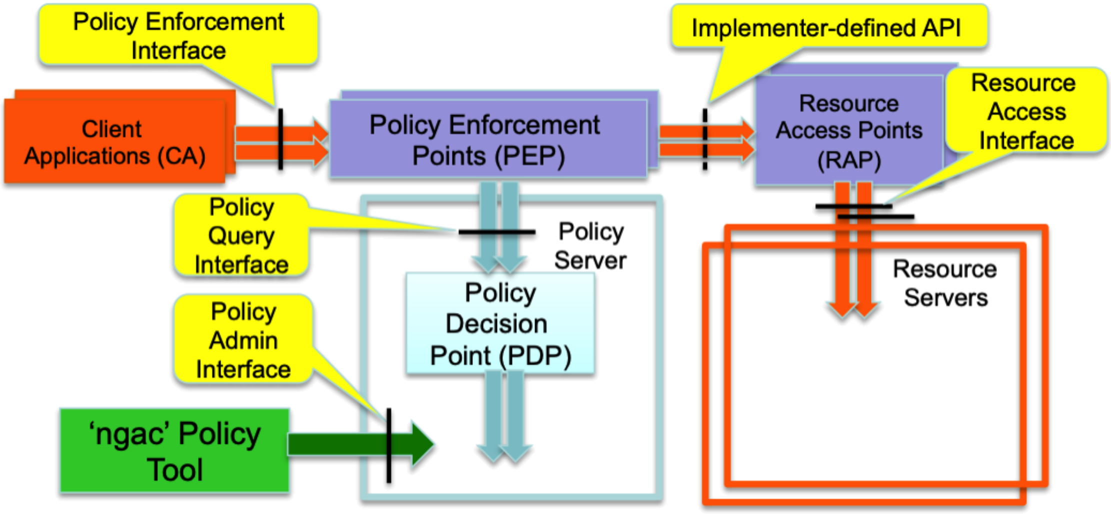

Policy Enforcement Interface
Implementer-defined API
Client Applications (CA)
Policy Enforcement Points (PEP)
Resource Access Points (RAP)
Resource Access Interface
Policy Query Interface
Policy Server
Policy Decision Point (PDP)
Resource Servers
Policy Admin Interface
‘ngac’ Policy Tool

Figure 44: The PEP-RAP Design Pattern for the Resource Access Path

The PEP and the RAP should both be small and logically simple modules, and they may possibly be packaged within the same executable program. The RAP, which uses the resource access interface, essentially includes the code previously appearing in the Client Application to access the Resource Server, which was replaced by calls to the Policy Enforcement Interface.

The PEP is basically a branching statement that invokes the PDP’s `access( )` query function as the condition of the branch. Figure 45 uses high-level pseudo-code to illustrate the conceptual simplicity of the PEP and the RAP. Of course, since both could be implemented as Web services, there would be the necessary additional details of marshalling arguments, invoking the PEP and the RAP, and returning and interpreting results. Nonetheless, following this simple design pattern will help to assure that the PEP and RAP are *trustworthy* components of the NGAC resource access path.

PEP pseudo-code	RAP pseudo-code

`pep( Op, Object, Data )`
`  determine User from the session environment`
`  query_result = pdp:access(User, Op, Object)`
`  if query_result == ‘grant’ then`
`    ObjInfo = pdp:getobjectinfo(Object)`
`    ra_result = rap(Op, Object, Data, ObjInfo)`
	`rap( Operation, Object, Data, ObjInfo )`
	`  identify res_server using ObjInfo`
	`  result = res_server:Operation(Object, Data)`
	`  return to PEP: result`

```
    return to Client App: ra_result
  else
    return to Client App: ‘Op on Object denied’
```

Figure 45: PEP and RAP pseudo-code

The PEP is called by a Client Application (CA) on behalf of some user through a Policy Enforcement Interface to perform an operation on a resource provided by a Resource Server. After marshalling arguments and calling the Policy Server’s Policy Decision Point (PDP), the PEP implements a simple conditional that, based on the “grant”/”deny” response from the PDP, either proceeds to perform the requested resource operation through the RAP, or it returns to the CA with an indication of an access failure.

The `access( )` query, made by the PEP to the PDP, specifies the user, the object, and the operation to be performed on the object. Using the server’s current policy, the PDP computes whether the user is permitted to perform the operation on the object and it returns either “grant” or “deny”. The “grant” branch of the PEP branching statement invokes the RAP to perform the specified operation on the object, while the “deny” branch immediately returns a failure to the CA.

The RAP is simple as well, containing code to access a resource server that looks much like the code that would otherwise be in the resource-using application if it were to directly access the resource without mediation by the access control system. The RAP uses the identity and location of the object, obtained from the server by the PEP, to perform the indicated operation, optionally using the data argument.

The simplicity of the PEP and the RAP, and their isolation as separate and distinct execution units, are a critical foundation of the strategy of unbundling them from the core NGAC implementation. Since they are simple, they can be verified by inspection, thereby establishing their trustworthiness. Since they are separate execution units, they cannot be corrupted by the CA or by other processes. Finally, as distinct execution units (processes) they can be given privileges distinct from those of the CA and those of other processes. Specifically, they should be given *exclusive access* to the protected resources (or resource servers), so that there is no way for the CA or another process to bypass them and directly access the protected resources.

This architectural approach allows a CA implementer to develop and test access to new object kinds, without a need to modify the Policy Server or resource servers.[9]

6.2.2.2 Installing the components and configuring the operating environment

Using the ‘ngac’ Policy Tool and the ‘ngac-server’ Policy Server as an individual user working in a private directory for development and testing, or for demonstrations in a benign environment, is very straightforward. Installing and configuring the Policy

[9] The only required action is to assign the new resources, through administration of the operating environment’s permissions, to the pool of resources that can only be accessed by the RAPs.

Server and PEPs/RAPs for a production environment, especially one with active threat agents, would require additional steps, such as:[10]

1. Create a new user ‘ngac_user’ and a group ‘ngac_group’.
2. Make the executable files for PEPs/RAPs owned by the ‘ngac_user’.
3. Set the permissions to make the executable files executable only by ‘ngac_user’ and ‘ngac_group’.
4. Turn on the set-UID-on-execution bit on the Policy Server and PEP/RAP executable files so that they will run as ‘ngac_user’.
5. Allocate TCP ports for the network services, PEPs, and Server APIs.
6. Set up trusted channels (e.g. SSL sockets) for application and NGAC component interactions. This task includes generation, distribution, and installation of certificates carrying cryptographic keys. (It also entails enabling the Policy Server to use SSL sockets.)
7. Using the operating environment’s security features, set the permissions on ordinary resources that are to be protected, such as files, to be read/write only by ‘ngac_user’ and ‘ngac_group’.
8. Create a master initialisation script that starts the Policy Server, PEPs/RAPs, and Web services with appropriate process ownership, privileges, and ports. Arrange for the script to be run on system startup.

This is an advanced system administration exercise.

6.2.3 Algorithms, technologies and tools

6.2.3.1 TOG-NGAC

We have implemented a Policy Server that contains the Policy Decision Point (PDP), Policy Administration Point (PAP), and Policy Information Point components of the NGAC functional architecture as illustrated in Figure 46. The server offers two external interfaces, the Policy Query Interface and the Policy Administration Interface, both realised as RESTful APIs.

Further, we have implemented an interactive Policy Tool that can be used to test policies under development. It replicates the decision logic of the PDP in a desktop tool and provides features to automate tasks and to observe the internal operational status of the algorithms when necessary.

[10] In fact, the prototype implementation of TOG-NGAC is not intended for use in production environments.

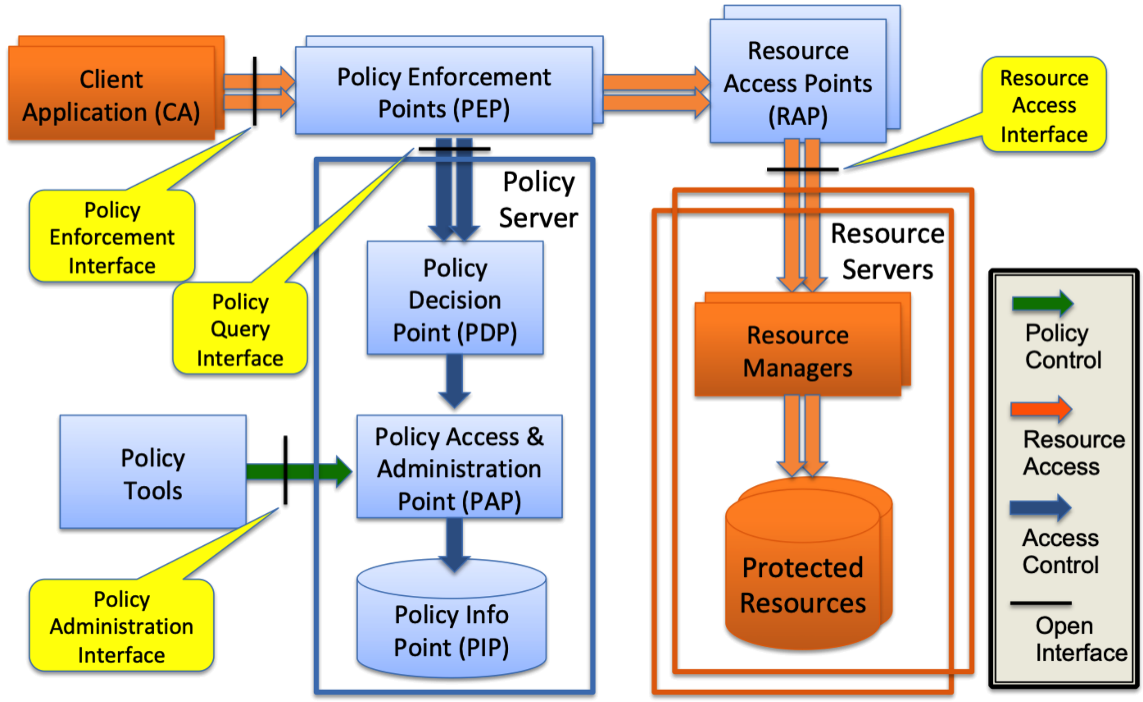

Client Application (CA)
Policy Enforcement Points (PEP)
Resource Access Points (RAP)
Resource Access Interface
Policy Enforcement Interface
Policy Query Interface
Policy Server
Policy Decision Point (PDP)
Resource Servers
Resource Managers
Policy Tools
Policy Access & Administration Point (PAP)
Policy Administration Interface
Policy Info Point (PIP)
Protected Resources
Policy Control
Resource Access
Access Control
Open Interface

Figure 46: NGAC Functional Architecture with "unbundled" PEP and RAP

To place the ability to add new object types into the hands of the developer we have “unbundled” the PEPs and RAPs. The path from the application to the protected resource, indicated in the colour orange, is now owned by the application developer.

The developer is expected to faithfully implement this architecture and to use the indicated standardized interfaces, except as may be necessitated by novel resource kinds. The PEP and the RAP should ideally be distinct processes so that they provide maximum functional isolation and can be run with privileges different to those of the application, and possibly to each other. They remain trusted components of the NGAC functional architecture and should be very small and simple so that their correctness is easily verifiable by inspection. The PEP only needs to call the PDP using an access query that the PDP answers with “grant” or “deny”. Based on this response the PDP must not perform the access to the RAP (if “deny”), or proceed to perform the object access and return the result to the application (if “grant”). The PEP is basically a decision statement conditioned by the PDP’s response that performs the access on one branch, or reports an error on the other branch.

The Policy Administration Interface is protected with a token, the possession of which is granted to administrators and administrative tools. Through this interface the PAP can be commanded to load and unload policies, select a policy to be active, combine policies in various ways, and modify loaded policies incrementally by adding and deleting policy elements. The value of the token is established when the Policy Server is started. If no token is specified at that time the PAP will use a default token from the Server’s build-time parameters. This feature may be convenient for testing and benign environments.

6.2.3.2 NGAC Initialisation and Session Manager

NGAC performs the policy interpretation and access control function but it does not itself perform identification and authentication (I&A), a vital preliminary service. These functions are ordinarily performed by the operating environment. The user id (or something uniquely linked to the user id) of the user running the application must be made available to the PEP and subsequently to the PDP to use when answering access requests. This user identity must be reliable or the access control system can be spoofed[11]. The application itself should not be trusted to provide this information.

One approach is to have the system process that logs the user in and initiates the user's session provide the user or session identifier to a distinct PEP instance that is started for the session. We refer to this system process the *initialisation shell* or *session manager*. The shell could provide the user identity directly to the PEP.

The session manager is a trusted component of the system. It should notify the server when a session is established, who is the associated user, and when the session is terminated. The server is already able to register sessions and accept a session id in lieu of a user id in access queries. The NGAC server could further participate in the I&A scheme by storing the user id and password in the PIP, where the session manager can request it when performing a user login.

Another thing we don't want to trust the user or application to do is to set the policy that will be used for computing access requests. Otherwise an all-permissive policy can temporarily be substituted for a restrictive one. Policy selection should be done by the master initialisation shell.

6.2.3.3 Protecting Web Services

We assume that PEPs will be implemented as a Web service and the RAPs may be as well, for ordinary local resources.

For protected objects that are themselves Web services the PEP will be a proxy for the Web service. The contrast for PEPs as a Web service serving ordinary resources vs PEPs servicing Web service resources is depicted in Figure 47.

[11] If a user or a process can claim an identity without authentication then it can claim an identity that has maximum privileges in the policy.

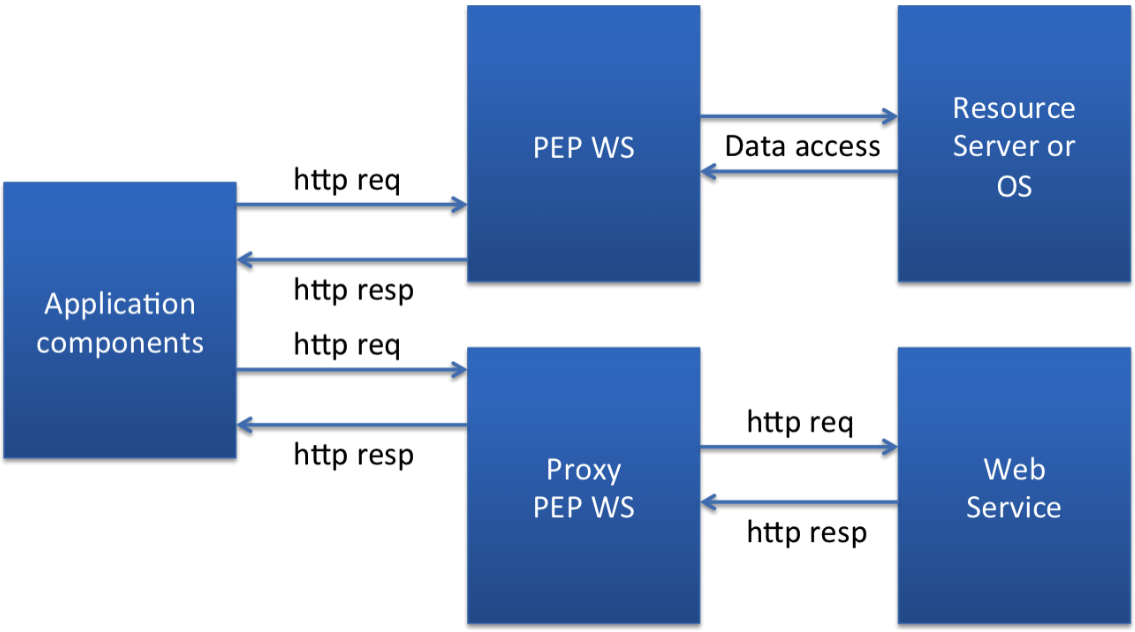


Figure 47: NGAC PEPs for ordinary resources and Web services

The interaction of the components is a bit different in the two cases. We consider this in the next section.

6.2.3.4 *Sequence of NGAC Component Interactions*

An example of a simple NGAC execution sequence with an ordinary PEP/RAP is shown in Figure 48. This example assumes that trusted channels (e.g. SSL sockets) are used for communication among the components so that identity is established by the communication negotiation.

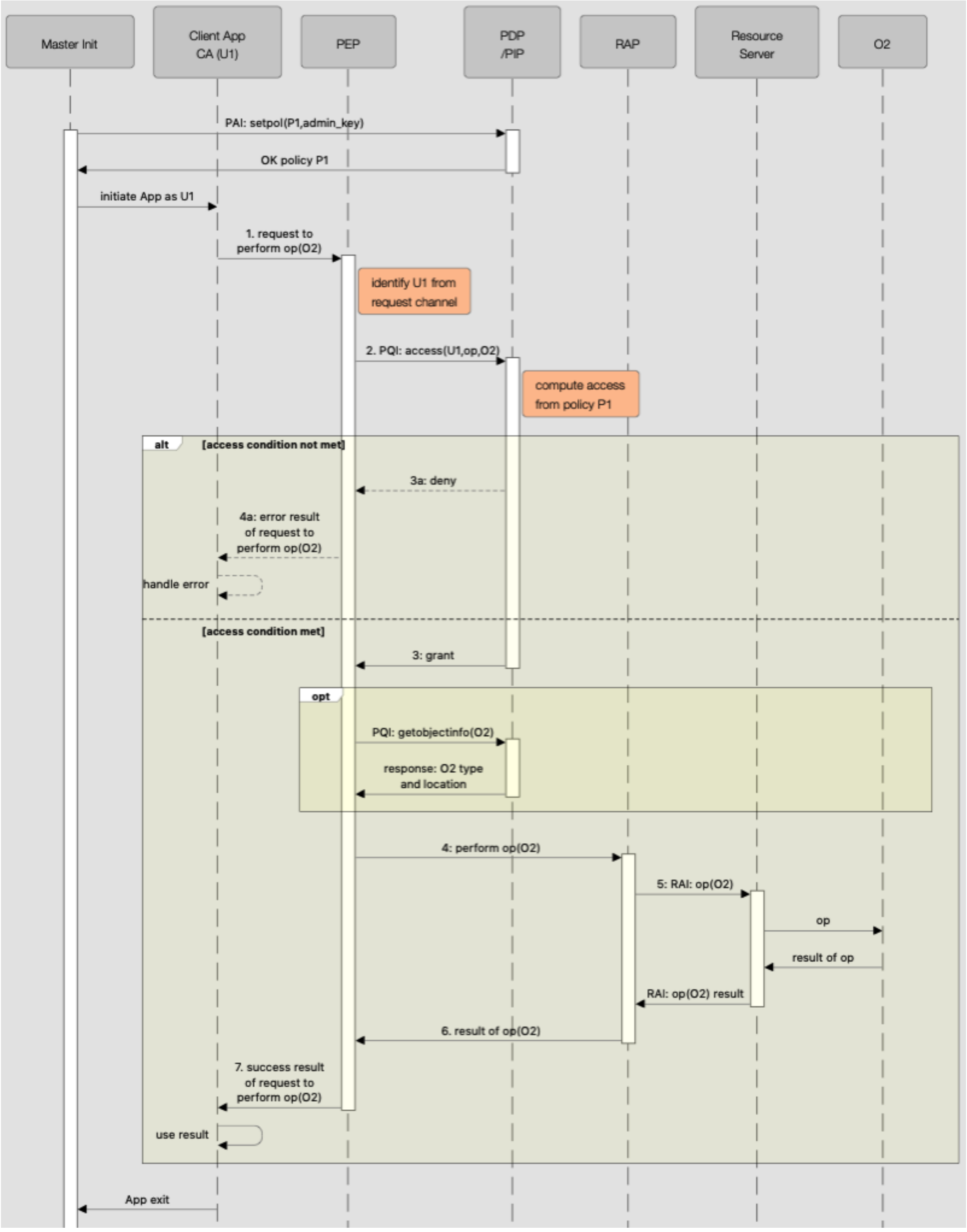


Figure 48: Normal PEP NGAC sequence chart

An example of an NGAC execution sequence with a Web service proxy PEP is shown in Figure 49. This example also assumes that trusted channels (e.g. SSL sockets) are used for communication among the components so that identity is established by the communication negotiation. Another approach would be to package the PEP into the Web service.

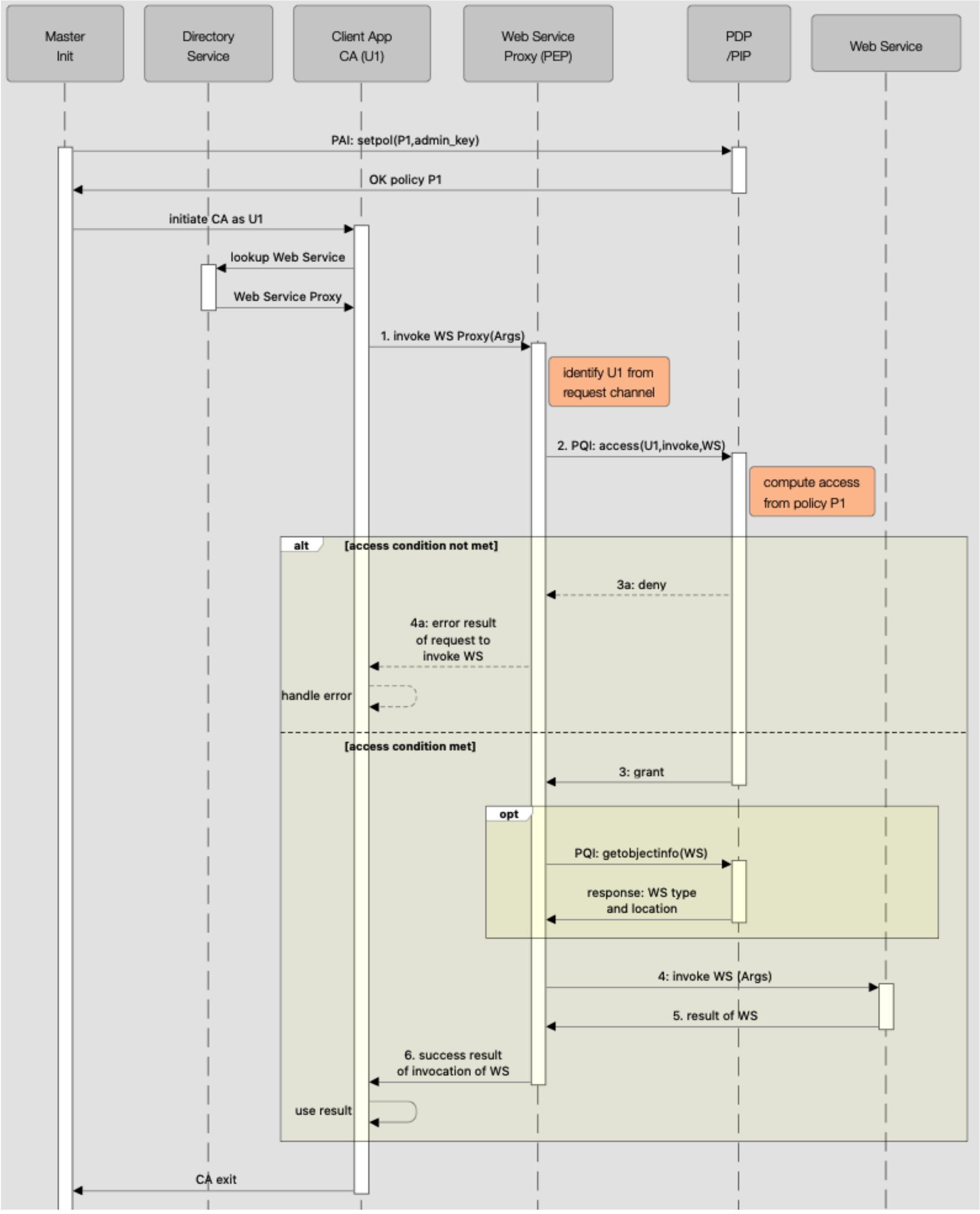


Figure 49: Web service PEP NGAC sequence chart

6.3 METHODOLOGY FOR OVERALL SYSTEM SECURITY, PRIVACY AND TRUST

6.3.1 Overview

The approach taken by this methodology is a simultaneous application of two different but complementary established approaches, ISO 15408 (29) (30) (31) and IIC IISF (32). They are complementary in this application because ISO 15408, otherwise known as the Common Criteria (CC), can be applied to any kind of system that includes security enforcing functions, while the IISF specialises the domain to large complex distributed systems of systems in an industrial environment. SAFIRE adds the further specialization to Systems of Production Systems (SOPS) in a Factory of the Future (FoF) setting.

ISO 15408 has a powerful approach and method for identifying and validating security requirements for a particular system or product, and providing a methodology for evaluating the target for success in implementing identified requirements. It starts from primitive concept of threat and policy and ends with the certification of systems and products.

The IISF has already baked-in some reasoning of the kind prescribed by ISO 15408 leading to the identification of a generalized architecture (represented in the IIC's Reference Architecture, IIRA (33)) and a framework of security functional requirements applied within the architecture. It is useful to understand and apply the principles of ISO 15408 even when some of the work has already been done, because the IISF is still very general and leaves many choices of implementation and deployment to be made. The disciplines of ISO 15408 concepts are advantageous in weighing the many decisions that must be made.

Thus, we have created a methodology that is a blend of these useful approaches.

6.3.2 Approach

Both ISO 15408 and IIC IISF are useful in analysing the security requirements of a system or a component. Both provide detailed catalogues, ISO having SFRs and SARs, and IISF having a collection of hierarchical building blocks of security functionality. At the very least, consulting them helps to assure that nothing is forgotten. Rather than staring at diagrams of your system in search of security weaknesses, one can search the catalogues and consider whether there is a need for such and such an item in one's system.

Using ISO 15408 Concepts:

- **Define the Security Problem.**
  Using concepts and methodology arising from the ISO/IEC 15408 standards for security evaluation criteria, the methodology starts with defining the *Security Problem* that must be solved in the target system. This definition includes:
  - identification of the **threats** the system is reasonably expected to face,
  - identification of the **organisational security policies** that any solution to the security problem must enforce, and
  - identification of the **assumptions concerning the environment** in which the system operates that can be safely made.
- **Identify Security Objectives.**
  The security problem is stated in terms of three diverse kinds of entities: threats, policies, and assumptions. In the present step these are then normalized into a

uniform set of conditions, *viz.* objectives, which if achieved would solve the security problem by assuring that every reasonable identified threat is countered, and every security policy enforced, by a combination of measures that is consistent with the continuous maintenance of the identified environmental assumptions. Specifically, the objectives must not require stronger assumptions than those that can be guaranteed for the environment.

- **Select Security Requirements.**
  A set of security requirements, consisting of security functional requirements (SFRs) and security assurance requirements (SARs) are selected. SFRs define behavioural requirements, while SARs have to do with the manner in which the SFRs are realised. We say the requirements are "selected" because ISO 15408 provides a catalogue of SARs and SFRs that have proven to cover a very high percentage, if not all, of the situations that have been encountered over a long period of experience preceding and since the establishment of the standard. The applicability of requirements from the catalogue are enhanced by the possibility of instantiating the requirements in four ways that permit them to be adapted to a wide variety of situations without sacrifice of precision.

This approach can be used at any scale, at the system level or the component level. In fact, by far the most common application of the CC is to perform product evaluations. Such products are used in large number and diversity in the building of complex systems. The properties of the products become important when it comes to having confidence in the properties of systems constructed of compositions of such components.

Using IIC IISF Concepts:

- **Understand the Framework.**
  Early sections of the IISF provide motivation and explanation of the viewpoint embodied in the Framework. It also presents some threat and risk-based reasoning that provide foundation for the security functional requirement identification. The actionable substance of the Framework is in the later sections, which contain a detailed presentation of the security functional building blocks that are needed for an industrial networked system and shows how these building blocks are applied to achieve various security objectives that are identified within the context of the reference architecture. More on the functional building blocks will be presented shortly.
- **Understand the relation of the IISF to the Reference Architecture (RA).**
  The Security Framework spans the entire architecture from the IT domain to the OT domain and the connections between them in an IIoT system.
- **Understand how the RA relates to your concrete system architecture**.

  The IIRA functional view is sufficiently general that it should not be difficult to find a mapping to a concrete IIoT system.
- **Now map the Security Framework to your concrete system architecture**.

From this one can discover where the functional building blocks of the IISF need to be applied (potentially) depending upon the specific threats and environmental assumptions of your security problem. This exercise enables the system implementer to see a security approach that spans the entire system and to select, integrate and configure functional components. It should be noted that both the RA and the IISF functional building blocks span the entire IIoT system from end-to-end.

Challenges

Relating the IISF to the IIRA to your system can be daunting if you are unfamiliar with the IISF. For this reason it is useful to start with a good understanding of all three. First you must discover whether one of the architectural view alternatives of the IIRA is more readily relatable to your system. It may be necessary to form an interpretation of the IIRA model that conforms to your system, or you may find it better to make your system architecture conform to one of the architectural view alternatives, if you have the freedom to do that. Next you must create an interpretation of the IISF that maps to the architecture to which you have chosen to conform to the elements of the IISF, for example, identifying endpoints, connections, roles, assets needing protection, etc.

6.3.3 Abstract Platform for SAFIRE Security Implementation

Each industrial SAFIRE deployment will have its own security problem[12]. Since SAFIRE is an “add-on” capability that will operate within the context of an existing system that is already built and managed by an IT and OT staff.

We will refer to this existing system as the *abstract platform* upon which SAFIRE operates. It comprises the processing and networking hardware, the system and networking software, the applications, and application support software (such as DBMSs). Many of the components of the abstract platform are not developed by the platform owner and operator, but are commercial third-party products or freely available software. For convenience we will also consider the additional infrastructure components upon which SAFIRE depends to become part of the abstract platform.

There will be a host of available security mechanisms provided by the abstract platform, not all of which are used, some of which are overlapping, or redundant, or potentially conflicting; and each having its own configuration mechanisms and enforcing its own local policy.

We will define an *Assume-Guarantee Contract* between SAFIRE (generally, and the SAFIRE Security Services in particular) and the abstract platform. SAFIRE will *assume* certain features and functions to be provided by the abstract platform, and these things

[12] We follow ISO 15408 in asserting that the *security problem* is defined by the *threats*, organizational security policies, and environmental assumptions that are deemed to apply to a particular system, and the derived security objectives that arise from these. The *solution* to the security problem is a set of security requirements that meet all of the security objectives.

in turn are ostensibly *guaranteed* by the abstract platform. This assume-guarantee contract allows the separation of responsibilities for the abstract platform's realisation from that of functions specifically developed as part of SAFIRE. By making SAFIRE's assumptions explicit, the platform's guarantees are partial requirements on the platform that can serve notify the platform owner/operator of what must be provided, and as a basis for assessing the suitability of a provided platform for hosting SAFIRE.

As will be seen, many of the SAFIRE Security Services corresponding to conventional security functional requirements will be allocated to the abstract platform, while something representing the unique contributions of SAFIRE to the state-of-the-art with respect to security, i.e. the unifying policy of NGAC, will be allocated to the SAFIRE security implementation.

6.3.4 Security Building Blocks

The IISF functional viewpoint includes the identification of the functional building blocks of the security framework, illustrated in Figure 50. The figure depicts three layers, the upper layer representing four core security functions: endpoint protection, communications and connectivity protection, security monitoring and analysis, and security configuration and management. These functions are in turn supported by the data protection functions of the second layer and a security model and policy functions of the third layer.

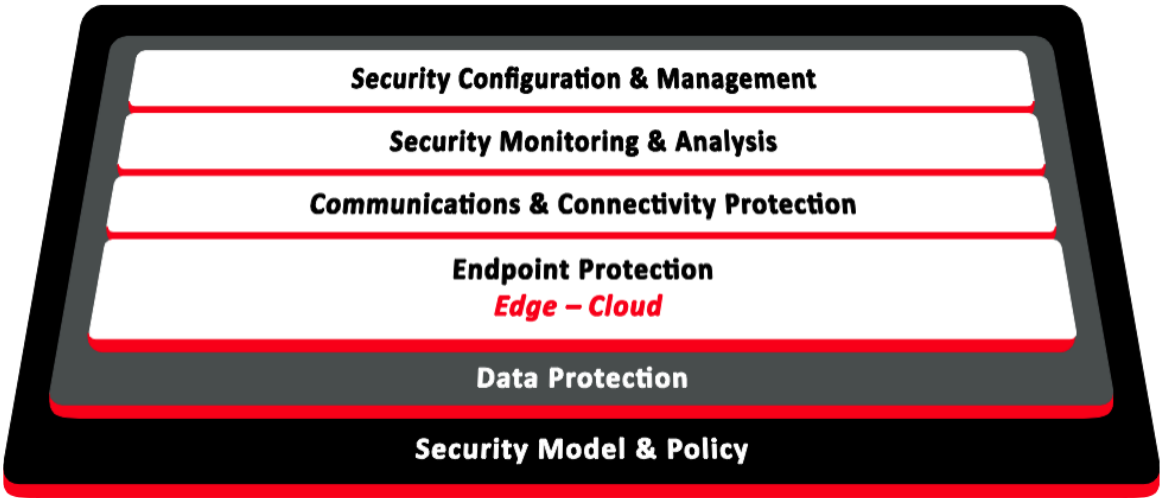

Security Configuration & Management
Security Monitoring & Analysis
Communications & Connectivity Protection
Endpoint Protection
Edge – Cloud
Data Protection
Security Model & Policy

Figure 50: IISF functional building blocks

6.3.5 Mapping IISF to system architecture and its implementation

The IISF establishes first a generic mapping, shown in Figure 51, from the security framework's functional viewpoint, to the IIRA Functional View and then to the IIoT System View, which includes both operational technology (OT) and information technology (IT) and their interconnects.

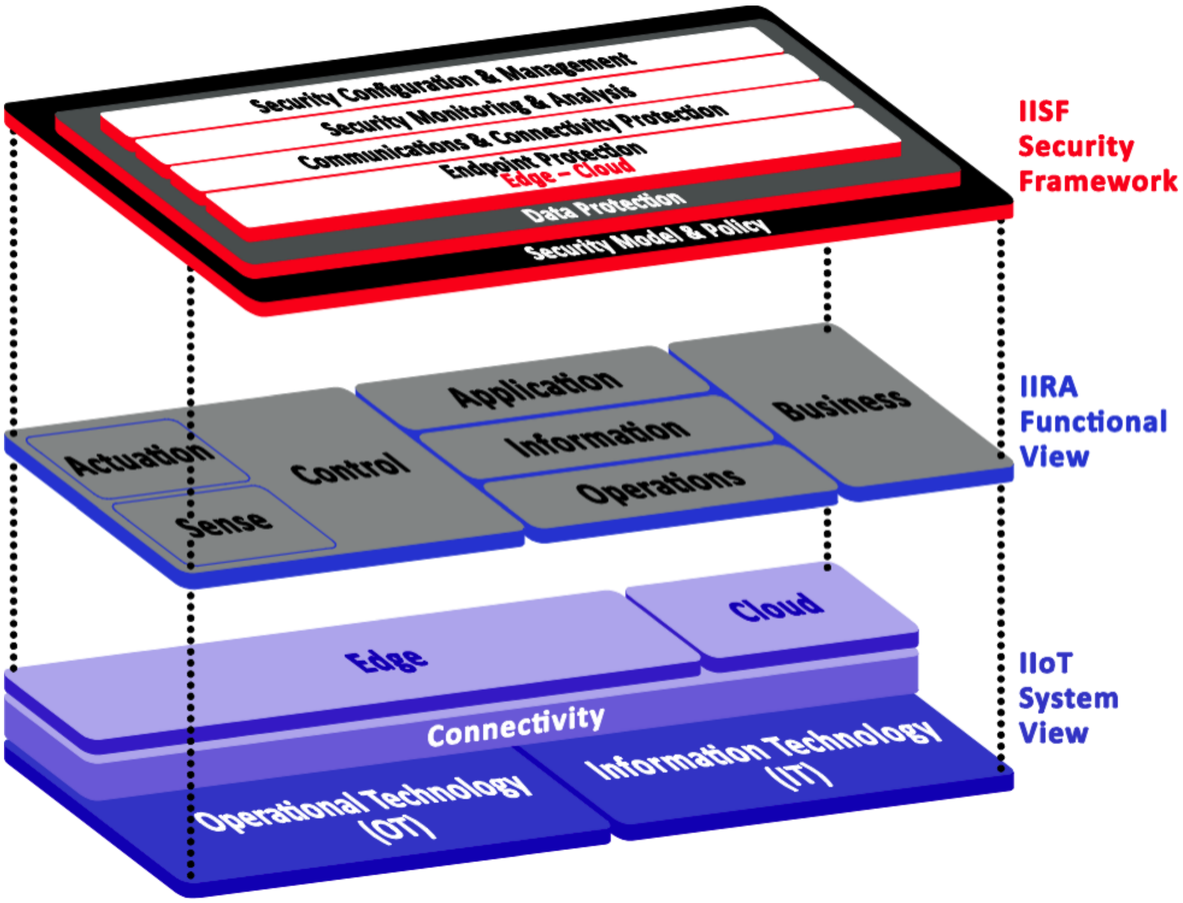


Figure 51: IISF to IIRA to IIoT system mapping

We will employ this mapping approach to demonstrate the adequacy of security functions in the SAFIRE architecture and its deployment in an actual system, identifying the elements of the SAFIRE implementation and the abstract platform that correspond to the IISF functional elements. There will often be numerous options for how the security functions can be provided and we do not intend to unnecessarily prejudice or constrain choices among those options. Our objective is to provide the generic mapping and the implementation principles to be applied in making choices for the realization of security functions in actual systems and to provide guidance to help achieve a consistent and coherent set of choices.

Section 6.3.8 presents a detailed example mapping between NGAC and the IISF security functional building blocks.

6.3.6 Application of the IISF to SAFIRE platform and Security Services

The IISF identifies many different security practices and mechanisms that must be applied to an IIoT system. It provides excellent guidance in the effort to obtain completeness of security coverage.

The IISF functional building blocks are intended to address security across all the functional domains of the Industrial Internet Reference Architecture (IIRA), from end-to-end from edge to cloud.

6.3.6.1 IISF-SAFIRE Security Services and Abstract Platform Correspondence

We intend to develop a correspondence of IISF functional building blocks to the SAFIRE Architecture and to the SAFIRE Abstract Platform. The SAFIRE Architecture was presented in deliverable D1.4. The SAFIRE Abstract Platform is the sum total of operating environment, network, and application components that SAFIRE technology depends upon.

The IISF-SAFIRE Security Services and Abstract Platform Correspondence will provide a mapping from the elements of the IISF functional building blocks to the SAFIRE Security Services and the SAFIRE Abstract Platform.

The Correspondence will identify the needed IISF elements and define them in terms of SAFIRE Security Services and SAFIRE Abstract Platform. During the process of developing the Correspondence, as the use cases and the SAFIRE Architecture are implemented, there will be opportunities and motivation for clarification and refinement of the Correspondence.

In the case of some IISF functional components the mapping will be made to security features of the SAFIRE Abstract Platform. Some portions may also be made to operational security procedures rather than technical mechanisms.

Functions that must be provided by security procedures, by the operating environment(s) of the Abstract Platform, or by third party products integrated as part of a specific use case will be noted as Assumptions on the Abstract Platform. This designation is also used as the default, if it has not yet been determined whether a component of the SAFIRE Security Services will satisfy the requirement.

6.3.7 Further Applications of SAFIRE’s NGAC-based security services

The IISF highlights the foundational role of security model and policy in its functional viewpoint. Critical aspects of this role may be addressed by NGAC.

NGAC has the potential to be applied in several areas identified by the IISF. Its primary application is data protection – access control security policy at endpoint systems. It can also provide integrity protection at the endpoints.

There are several secondary potential areas of application of NGAC within the IISF functional framework: configuration & management data protection, communications data protection, and monitoring data protection. It can also be used to protect its own policy configuration data. See Section 6.3.8 for additional insight into other applications of NGAC.

6.3.8 Mapping of NGAC to IISF Security Functions

The IISF covers virtually all aspects of security, both technical (that is, mechanisms that are implemented as hardware and/or software) and non-technical (that is, policies, procedures, and practices). Figure 52 shows the *functional viewpoint* of the IISF as six interacting building blocks, organized as three layers. The top layer represents the four core security functions, which are in turn supported by a data protection layer and a security model and policy layer.

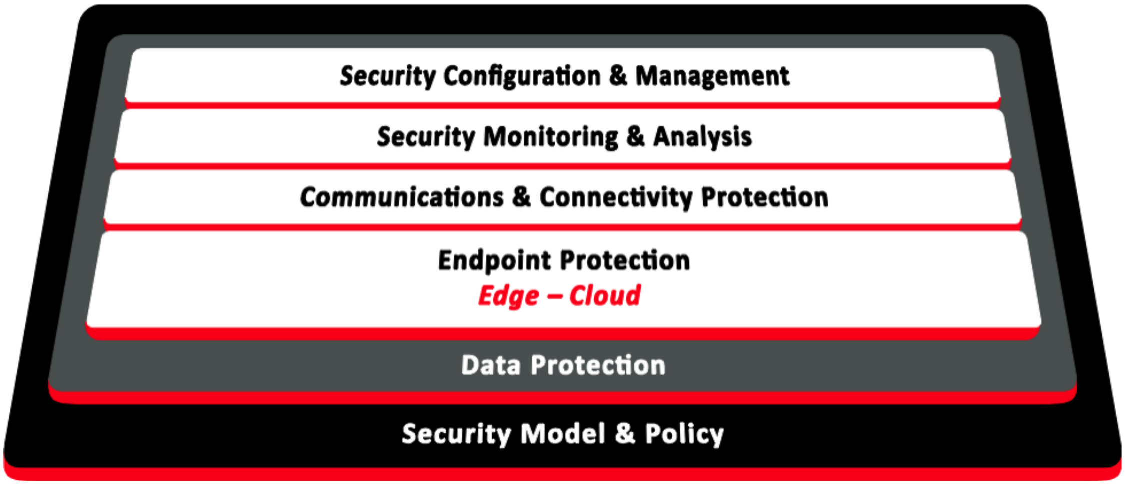

Security Configuration & Management
Security Monitoring & Analysis
Communications & Connectivity Protection
Endpoint Protection
Edge – Cloud
Data Protection
Security Model & Policy

Figure 52: Functional Building Blocks of the IISF

6.3.8.1 Overview of NGAC applicability in the IISF

NGAC is concerned with security model and policy, so in Figure 53 we indicate, is a very coarse way, the role of NGAC relative to the functional building blocks of the IISF. In this and subsequent figures a solid green ellipse indicates a primary role, while a dashed ellipse indicates a secondary or optional role.

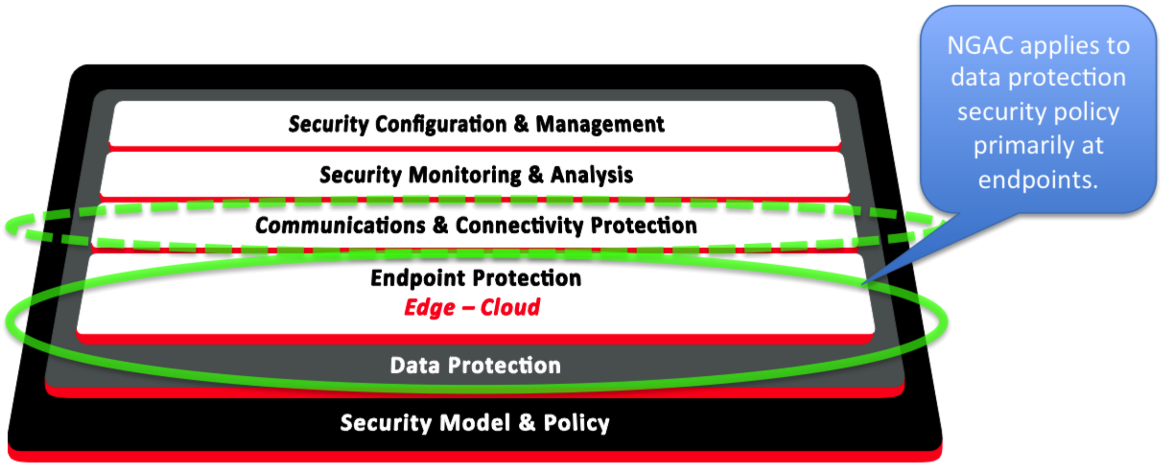

Security Configuration & Management
Security Monitoring & Analysis
Communications & Connectivity Protection
Endpoint Protection
Edge – Cloud
Data Protection
Security Model & Policy
NGAC applies to data protection security policy primarily at endpoints.

Figure 53: Overview of NGAC role in IISF functional building blocks

6.3.8.2 *Security model and policy*

Figure 54 illustrates role of NGAC within the functional breakdown for the security model and policy within IISF. The left side of this figure, Security Policy, is primarily non-technical and refers to organizational security goals and objectives. The right side, Security Model, is the technical aspect. NGAC is a technical measure providing security policy (mode) specification and enforcement. Thus, we show the emphasis of NGAC on the technical side. NGAC provides the capability to express and enforce a security model primarily at the end points. However, since NGAC is distributed, it depends on communications security. However, it is not primarily used to express or enforce communications and connectivity security policy as described by the NGAC standard. The following analysis will present a more refined explanation of the role of NGAC in the IISF.

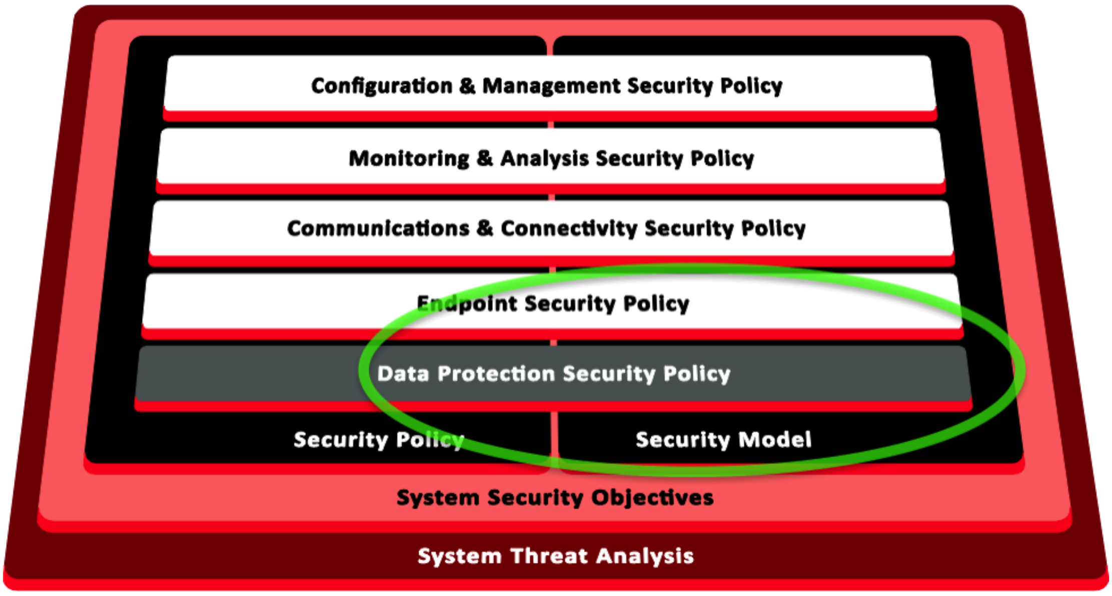

Configuration & Management Security Policy
Monitoring & Analysis Security Policy
Communications & Connectivity Security Policy
Endpoint Security Policy
Data Protection Security Policy
Security Policy
Security Model
System Security Objectives
System Threat Analysis

Figure 54: Overview of NGAC role in IISF security model and policy functional breakdown

6.3.8.3 *Endpoint protection*

Figure 55 shows the IISF functional breakdown for endpoint protection and the role that NGAC can play in endpoint data protection. NGAC is not currently supported natively within ubiquitous operating systems, e.g. through a “pluggable authorization module.” The current feasible deployments of NGAC depend on the endpoint’s operating system to provide the basic security properties of isolation and integrity for the resources that it exports, and which are placed under the NGAC scope of control. Thus, the role of endpoint security model and policy is shared with the operating environment (OE); for this reason a dashed ellipse is used to indicate NGAC’s role in this aspect. NGAC can provide fine-grained access control to OE-exported resources, and can thus provide fine-grained integrity protection, in the sense of “no unauthorized modification”, for objects under its scope of control.

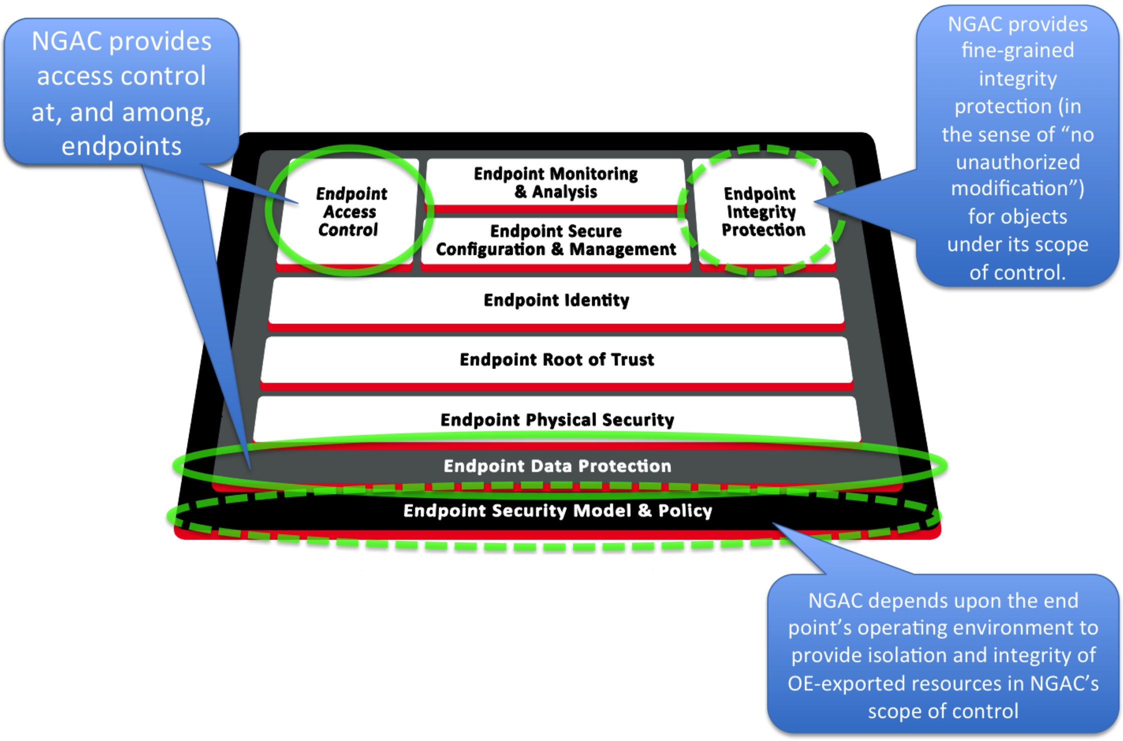

NGAC provides access control at, and among, endpoints
Endpoint Access Control
Endpoint Monitoring & Analysis
Endpoint Secure Configuration & Management
Endpoint Integrity Protection
NGAC provides fine-grained integrity protection (in the sense of "no unauthorized modification") for objects under its scope of control.
Endpoint Identity
Endpoint Root of Trust
Endpoint Physical Security
Endpoint Data Protection
Endpoint Security Model & Policy
NGAC depends upon the end point's operating environment to provide isolation and integrity of OE-exported resources in NGAC's scope of control

Figure 55: NGAC role in endpoint protection

6.3.8.4 *Communication and connectivity protection*

Concerning communication and connectivity protection, the functional breakdown of which is illustrated in Figure 56, NGAC does not have a primary role as it is defined in the NGAC standard. It could be used to model permissions for endpoints to communicate, and we are investigating the use of NGAC's policy modelling capabilities to represent information flow control, but other aspects of communication security are outside the scope of NGAC.

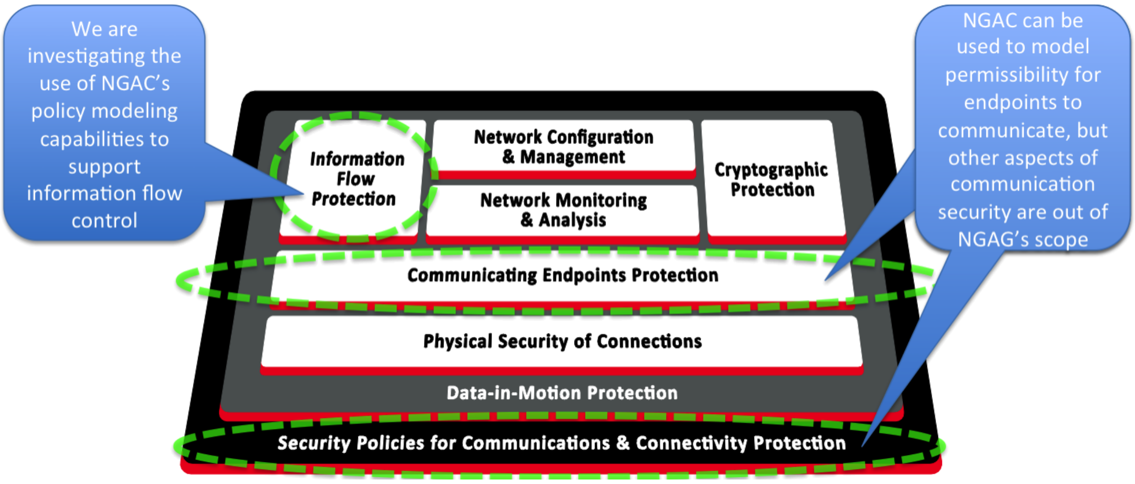
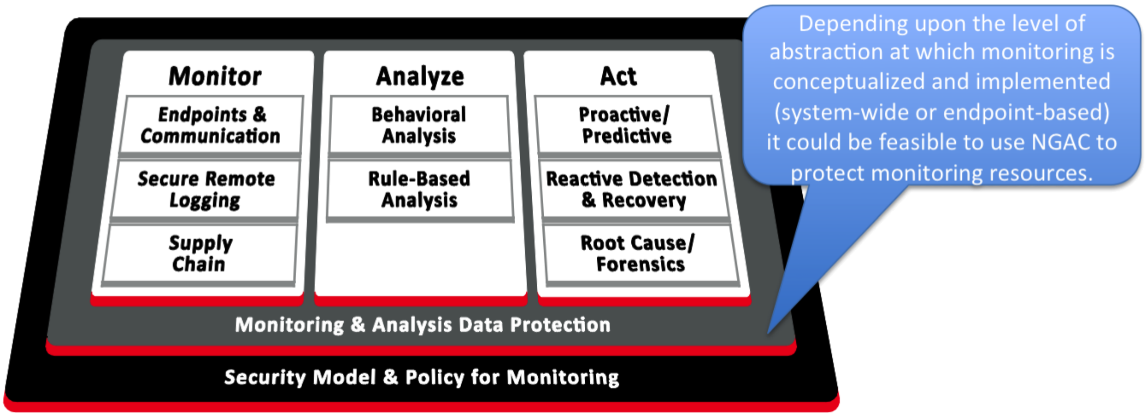

We are investigating the use of NGAC's policy modeling capabilities to support information flow control
Information Flow Protection
Network Configuration & Management
Network Monitoring & Analysis
Cryptographic Protection
NGAC can be used to model permissibility for endpoints to communicate, but other aspects of communication security are out of NGAG's scope
Communicating Endpoints Protection
Physical Security of Connections
Data-in-Motion Protection
Security Policies for Communications & Connectivity Protection

Figure 56: NGAC role in communication and connectivity protection

6.3.8.5 Security monitoring and analysis

Figure 57 illustrated the IISF functional breakdown of security monitoring and analysis. NGAC does not cover this aspect of security. It is conceivable that NGAC could play a supporting role for this aspect, depending on the level of abstraction at which monitoring is conceptualized and implemented, by protecting monitoring resources and granting use of those resources to monitoring agents. In this regard, monitoring is like NGAC itself because of its dependence on the operating environment to provide the underlying allocation of protection of resources used for monitoring. Whether monitoring resources are placed within the NGAC scope of control could be influenced by the consideration whether monitoring and its resources should be part of the overall access control policy, or whether it should be treated as a distinct subsystem from NGAC.

NGAC does not cover the Security Monitoring and Analysis function.



Depending upon the level of abstraction at which monitoring is conceptualized and implemented (system-wide or endpoint-based) it could be feasible to use NGAC to protect monitoring resources.

Figure 57: NGAC role in monitoring and analysis

6.3.8.6 Security configuration and management

Figure 58 shows the application of NGAC in the context of the IISF's functional breakdown for security configuration and management. The primary application of NGAC is its natural role in controlling changes to security policy models as is recommended for NGAC-based enforcement mechanisms. It can also potentially be used for configuration and management data protection if effectively integrated with the operating environment and other management functions. NGAC could also be used for change control of security configuration data used by other security enforcement mechanisms, thus further unifying security management.

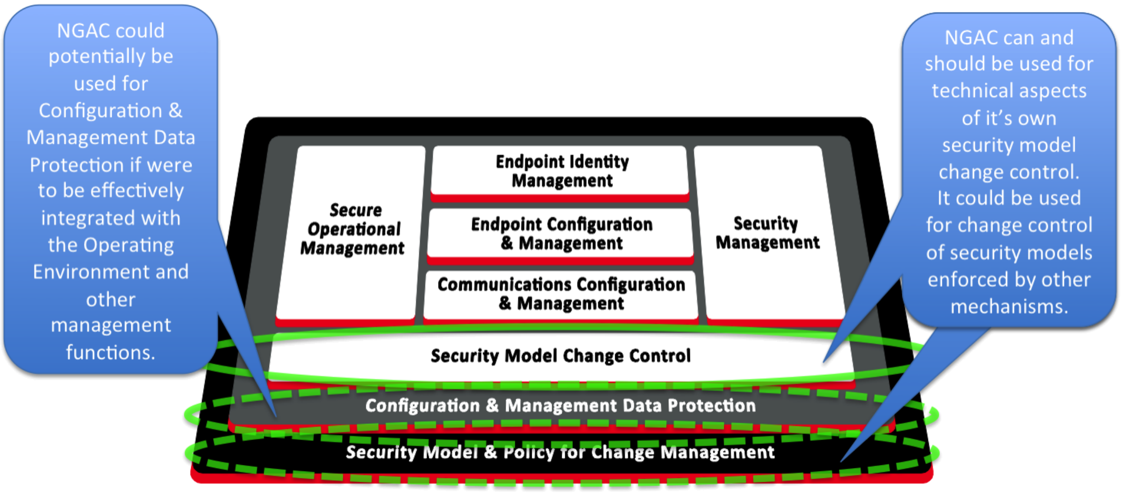

NGAC could potentially be used for Configuration & Management Data Protection if were to be effectively integrated with the Operating Environment and other management functions.
NGAC can and should be used for technical aspects of it's own security model change control. It could be used for change control of security models enforced by other mechanisms.
Secure Operational Management
Endpoint Identity Management
Endpoint Configuration & Management
Communications Configuration & Management
Security Management
Security Model Change Control
Configuration & Management Data Protection
Security Model & Policy for Change Management

Figure 58: NGAC role in security configuration and management

6.3.8.7 *Data protection*

Figure 59 shows the role of NGAC in the functional breakdown of data protection. The primary application of NGAC is to provide security model and policy for endpoint data protection; this is NGAC's *raison d'être*. Because NGAC is not yet integrated into the operating environment, current NGAC implementations leverage the OE to build mechanisms that enforce NGAC policies. As mentioned with respect to security configuration and management, NGAC can be used to protect all manner of operational and security-related data, including communications-related data, configuration data, and monitoring data. With appropriate policies and extensions to enforcement mechanisms, NGAC can address a *slice* of data protection across the system, including data-at-rest (DAR), data-in-use (DIU), and data-in-motion (DIM), providing a higher-level abstraction of protections provided by the operating environment and the network hardware and software.

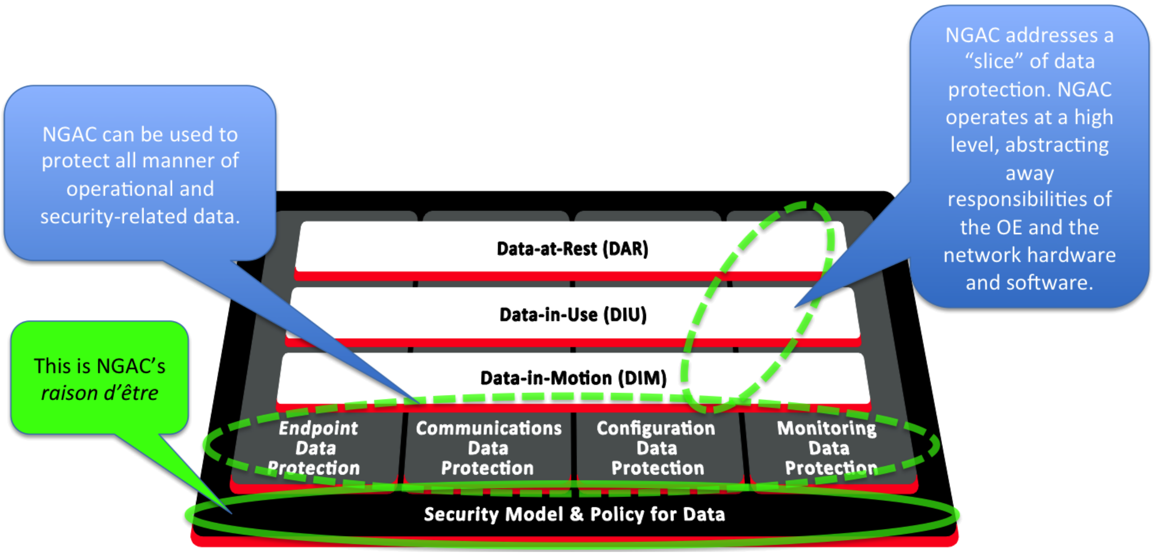

NGAC can be used to protect all manner of operational and security-related data.
NGAC addresses a "slice" of data protection. NGAC operates at a high level, abstracting away responsibilities of the OE and the network hardware and software.
This is NGAC's *raison d'être*
Data-at-Rest (DAR)
Data-in-Use (DIU)
Data-in-Motion (DIM)
Endpoint Data Protection
Communications Data Protection
Configuration Data Protection
Monitoring Data Protection
Security Model & Policy for Data

Figure 59: NGAC role in data protection

6.3.8.8 *Refined role of NGAC in the IISF*

Having examined the role, and potential roles, of Next Generation Access Control in various aspects of security within the Industrial Internet Security Framework, we now present in Figure 60 a refined view of NGAC's roles in the functional breakdown for security model and policy. Previously, in Figure 54 we identified in a broad sense the applicability of NGAC as centering on data protection security policy, endpoint security policy, and security model.

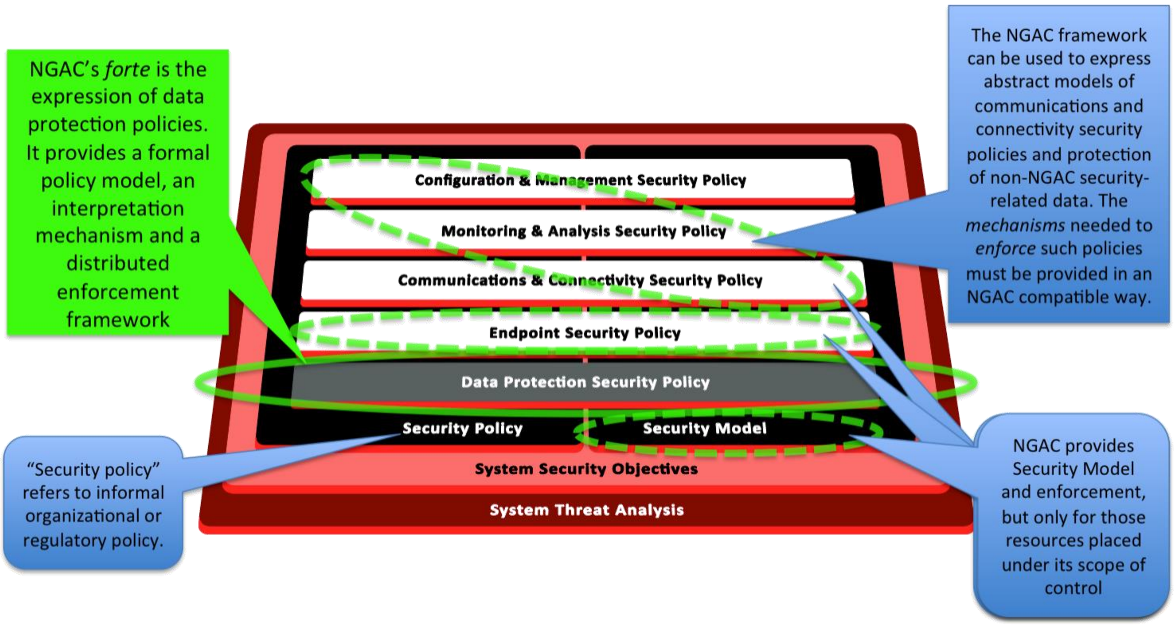

NGAC's *forte* is the expression of data protection policies. It provides a formal policy model, an interpretation mechanism and a distributed enforcement framework
The NGAC framework can be used to express abstract models of communications and connectivity security policies and protection of non-NGAC security-related data. The *mechanisms* needed to *enforce* such policies must be provided in an NGAC compatible way.
Configuration & Management Security Policy
Monitoring & Analysis Security Policy
Communications & Connectivity Security Policy
Endpoint Security Policy
Data Protection Security Policy
Security Policy
Security Model
System Security Objectives
System Threat Analysis
"Security policy" refers to informal organizational or regulatory policy.
NGAC provides Security Model and enforcement, but only for those resources placed under its scope of control

Figure 60: NGAC role in security model and policy refined

Here we assert NGAC's *forte* as the expression of data protection policies, providing a formal framework for policy models, an interpretation mechanism and a distributed enforcement framework in its reference implementations. We call attention to the fact that the language of the IISF is somewhat nuanced in its distinction between security policy and security model, using "security policy" to refer to informal organizational or regulatory policy, and "security model" to refer to the more precise and machine process-able representations that NGAC refers to a "policy specification". In this sense, NGAC provides security model and enforcement, but only for those resources that are placed under its scope of control through an appropriate configuration of the underlying protection mechanisms of the operating environment.

From the standpoint of policy specification, the NGAC framework can be used to express abstract *policy models* of communications and connectivity security policies and protection of non-NGAC security-related data. As we have described in the foregoing presentation, such applications may include configuration & management security policy, monitoring & analysis security policy, communications & connectivity security policy, in addition to endpoint security policy. The *mechanisms* needed to enforce such additional policies must be provided in and NGAC-compatible way by creating policy enforcement points (PEPs) appropriate to the new resources and appropriate resource access points (RAPs) for those resources. These PEPs may call upon the NGAC policy decision point(s) (PDPs) to render access control verdicts based on the policies stored in the policy information point (PIP).

7. CONCLUSIONS

In order to enhance information management in the Manufacturing domain, SAFIRE offers a solution for big data analysis and situation-based process optimisation. To this extend, the steps to adapt and extend the SAFIRE platform to meet the business case requirements where collected and described in the integrated methodology presented in this document.

The methodology for the SAFIRE platform can be arranged in four groups of steps, focusing on BC analysis and scenario definition, BC customisation, platform integration, and platform testing and release. As first step, the business-specific requirements are being defined, and in the second step are being applied to the technical part of the platform, namely the SAFIRE services and modules. In the third step, all the modules configured for the business case are being connected to work together as an integrated system which is being tested and accordingly optimised in the business sites for the last step. The observation of the results from the platform released version is being observed using the SAFIRE monitoring dashboard.

Additionally, the SAFIRE methodology describes in detail the individual steps to adapt the four modules, namely the Predictive Analytics (PA), the Situation Determination (SD), the Optimisation Engine (OE) and the Security Framework (SPT). Some of the main aspects of the customisation needed on the modules are the data sources and samples for the PA, the situation model for SD, the optimisation metrics and fitness functions for OE, and the privacy rules for the SPT.

The integrated methodology described in this document, aims to support the experts from a given industrial company (industry experts), to employ technical stuff (SAFIRE experts) able to customise the SAFIRE services for the selected business needs in order to interact with the industrial legacy systems. It aims to provide an easy to follow, stepwise, workflow to accompany all the interest-to-SAFIRE-solution-parties from the business conceptualisation and specification, to the implementation, testing and release of a business tailor SAFIRE platform.

8. References

1. **University of York, Methodology for Dynamic and Predictable Reconfiguration and Optimisation Engine, SAFIRE project deliverable D3.1.** 2018.
2. **Dziurzanski et al, Piotr Dziurzanski, Jerry Swan, Leandro Soares Indrusiak.** *Value-Based Manu-facturing Optimisation in Serverless Clouds for Industry 4.0.* 2018.
3. **Mendez et al, Carlos A. Méndez, Jaime Cerdá, Ignacio E. Grossmann, Iiro Harjunkoski, Marco Fahl.** *State-of-the-art review of optimization methods for short-term scheduling of batch pro-cesses, Computers & Chemical Engineering, Volume 30.* 2007.
4. **Allen, James F.** *Maintaining knowledge about temporal intervals.* 1983.
5. **University of York, Early Specification of Dynamic and Predictable Reconfiguration and Optimisation Engine, SAFIRE project deliverable D3.2.** 2018.
6. **University of York, Optimisation Metrics and Benchmarking, SAFIRE project delivera-ble D1.2.** 2017.
7. **Deb et al., K. Deb, A. Pratap, S. Agarwal and T. Meyarivan.** *A fast and elitist multiobjective genetic algorithm.* 2002.
8. **Zhang and Li, Li, Q. Zhang and H.** *MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition.* 2007.
9. **Dziurzanski et al, Piotr Dziurzanski, Shuai Zhao, Jerry Swan, Leandro Soares Indrusiak, Sebastian Scholze, Karl Krone.** *Solving the Multi-objective Flexible Job-Shop Scheduling Problem with Alternative Recipes for a Chemical Production Process.* 2019a.
10. **Burkimsher et al, Andrew Burkimsher and Leandro Soares.** *Bidding policies for market-based HPC workflow scheduling.* 2016.
11. **Marcus et al, Darin Marcus and Laron Colbert.** *$EE, the Financial Aspect of OEE.* 2015.
12. **Stoer et al, J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall.** *Introduction to Numerical Analysis.* 2002.
13. **Yin et al, Hao Yin, Huilin Wu, and Jiliu Zhou.** *An improved genetic algorithm with limited iteration for grid scheduling.* 2007.
14. **Di Martino et al, Di Martino, S., Ferruci, F., Maggio, V., Sarro, F.** *Towards migrating genetic algorithms for test data generation to the cloud.* 2012.
15. **Leclerc, G., Auerbach, J.E., Iacca, G., Floreano, D.** *The seamless peer andcloud evolu-tion framework.* 2016.
16. **Ma et al, Ma, N., Liu, X.F., Zhan, Z.H., Zhong, J.H., Zhang, J.** *Load balance awaredistributed differential evolution for computationally expensive optimization problems.* 2017.
17. **Melab et al, Melab, N., Mezmaz, M., Talbi, E.** *Parallel hybrid multi-objective islandmodel in peer-to-peer environment.* 2005.
18. **Nogueras et al, Nogueras, R., Cotta, C.** *An analysis of migration strategies in island-basedmultimemetic algorithms.* 2014.
19. **Ishibuchi et al, Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y.** *Modified distance calculation in generational distance and inverted generational distance.* 2015.
20. **Deb et al., Deb, K., Agrawal, S., S., Pratap, Meyarivan, T.** *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization.* s.l. : Springer, 2000.
21. **Zhao et al, Shuai Zhao, Haitao Mei, Piotr Dziurzanski, Michal Przewozniczek and Leandro Indrusiak.** *Cloud-based Integrated Process Planning and Scheduling Optimisation via Asynchronous Islands.* 2019a.
22. **SAFIRE D5.8, 2019, Institut für angewandte Systemtechnik Bremen GmbH, Final Integrated Cloud Analysis and Reconfiguration Platform, SAFIRE project deliverable D5.8.** 2019.
23. **Zhao et al, Shuai Zhao, Piotr Dziurzanski, and Leandro Indrusiak.** *An XML-based Factory Description Language for Smart Manufacturing Plants in Industry 4.0. In International Workshop on Reconfigurable and Communication-centric Cyber-Physical Systems.* 2019b.
24. **Ryan et al, Ryan C., Collins J., Neill M.O.** *Grammatical evolution: Evolving programs for an arbitrary language.* 1998.
25. **Dziurzanski et al, Piotr Dziurzanski, Robert Davis and Leandro Indrusiak.** *Synthesizing Real-Time Schedulability Tests using Evolutionary Algorithms: A Proof of Concept.* 2019b.
26. ***InterNational Committee for Information Technology Standards, Cyber security technical committee 1. Information technology—Next Generation Access Control—Functional Architecture. ANSI INCITS 499-2018.*** *2018.*
27. ***InterNational Committee for Information Technology Standards, Cyber security technical committee 1. Information technology—Next Generation Access Control—Generic Operations & Abstract Data Structures. ANSI INCITS 526-2016.*** *2016.*

28. ***InterNational Committee for Information Technology Standards, Cyber security technical committee 1. Information technology—Next Generation Access Control— Implementation Requirements, Protocols and API Definitions. ANSI INCITS 525-2018.*** *2018.*

29. ***International Standards Organization, Geneva, Switzerland. International Standard 15408-1: Common Criteria for Information Technology Security Evaluation—Part 1: Introduction and general model, Version 3.1 Revision 4, CCMB-2012-09-001.*** *2012.*

30. ***International Standards Organization, Geneva, Switzerland. International Standard 15408-2: Common Criteria for Information Technology Security Evaluation—Part 2: Security functional components, Version 3.1 Revision 4, CCMB-2012-09-002.*** *2012.*

31. ***International Standards Organization, Geneva, Switzerland. International Standard 15408-3: Common Criteria for Information Technology Security Evaluation—Part 3: Security assurance components, Version 3.1 Revision 4, CCMB-2012-09-003.*** *2012.*

32. ***Industrial Internet Consortium. Industrial Internet of Things, Volume G4: Security Framework, IIC:PUB:G4:V1.0:PB:20160919.*** *2016.*

33. ***Industrial Internet Consortium. Industrial Internet of Things, Volume G1: Reference Architecture, IIC:PUB:G1:V1.80:20170131.*** *2017.*