Project Number 723634

D4.4 Full Prototype of Situational Awareness Services

Version 1.0
20 December 2018
Final

Public Distribution

ATB

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

PROJECT PARTNER CONTACT INFORMATION

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

DOCUMENT CONTROL

Version	Status	Date
0.1	Template creation	2 November 2018
0.2	First Content	13 November 2018
0.3	Additional Content	16 November 2018
0.4	Add OAS specifics	22 November 2018
0.5	Add ONA specifics	30 November 2018
0.6	Update document structure	5 December 2018
0.7	Add Electrolux specifics	11 December 2018
0.8	Internal Review	14 December 2018
1.0	Final Version	20 December 2018

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document provides the full prototype of situational awareness services and the situation model composed in the situation determination module as part of the SAFRIRE solution. This module processes data coming from connected systems / devices / products (data producers) to extract the current situation of these connected systems / devices / products. The document briefly describes the implemented services and functionalities of the FP of situational awareness services. Further, an overview about the additional functionalities compared to the EP is given. The next section gives guidelines on how to install and configure these services is described.

A description of the business case specific customisations for the SAFIRE industrial use cases is given in Section 5. Finally, the deliverable gives an overview about the software tools and frameworks used for implementation is presented.

1. INTRODUCTION

1.1 OVERVIEW

The Situation Determination services were implemented based on:

- the first results from Business Cases Requirements and Analysis (WP1),
- the results from the SAFIRE Concept (WP1),
- the specification of Situational Awareness Services (WP4) and
- the methodology for Situational Awareness (WP4).

1.2 APPROACH APPLIED

For each of the main technologies in SAFIRE the same approach is followed and that is to start by analysing the requirements collected at Business Case requirements and analysis phase, detailing these and from there derive the data model, functional specification, external interfaces, and technical specification.

The general approach followed to write the current document can be seen in Figure 1.

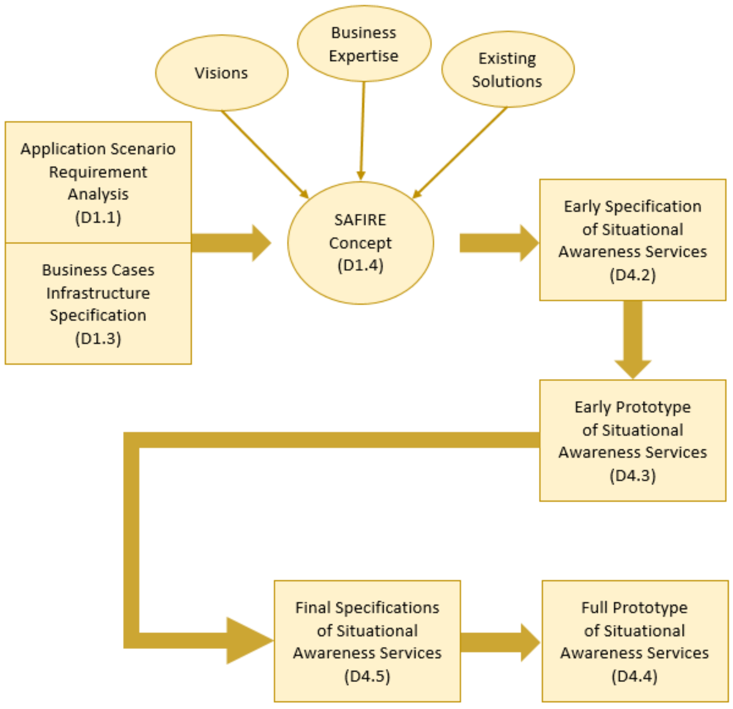

Visions
Business Expertise
Existing Solutions
Application Scenario Requirement Analysis (D1.1)
Business Cases Infrastructure Specification (D1.3)
SAFIRE Concept (D1.4)
Early Specification of Situational Awareness Services (D4.2)
Early Prototype of Situational Awareness Services (D4.3)
Final Specifications of Situational Awareness Services (D4.5)
Full Prototype of Situational Awareness Services (D4.4)

Figure 1: Approach followed for Full Prototype of Situational Awareness Services

1.3 PROGRESS BEYOND D4.3

The progress beyond the early prototype (D4.3), documented in this deliverable, is introduced in the following.

- **Situational Model** - the situation models were reviewed to model more accurately the environment of operation of the SAFIRE solution so that it allows for situational awareness. The BC specific extensions of the Situation Model have been extended to included concepts required for the Factory Description Language (FDL) (see WP3).
- **Situation Monitoring** – The full implementation of the monitoring services was realised according to the requirements and the final specification.
- **Situation Determination services** – are implemented and intgegrated according to the requirements and specification.
- **Integration of Security** – the full prototype specification integrates the SAFIRE Security, Privacy and Trust (SPT) framework into the situation monitoring and situation determination services.
- **Integration of the Optimisation Engine and Predictive Analytics Module** – the full prototype refines integration and the communication between the Situational Awareness Services and the Optimisation Engine according to a Metrics API defined by the Optimisation Engine. The integration with the Predictive Analytics is also realised using Kafka as communication channel to send Predictive Analytics results to Situational Awareness Services via defined Kafka topic strings.

1.4 DOCUMENT STRUCTURE

The document consists of:

- Section 1. Introduction, which describes the purpose of this document, and provides a brief overview of the contents of the document.
- Section 2. Description of the Full Prototype (FP) implementation of the Situation Determination Services including the Situation Model.
- Section 3. Overview about the integration with other modules.
- Section 4. Brief description on how to install and configure the Situation Determination Services.
- Section 5. Describes the specific customisation for the SAFIRE business cases.
- Section 6. Presents the Software tools used for implementation
- Section 7. Conclusions and wrap up of the deliverable

2. SITUATION AWARENESS SERVICES

The Situation Determination allows for identifying changes in the situations of the environment. The current identified situation is used to support the optimisation / reconfiguration.

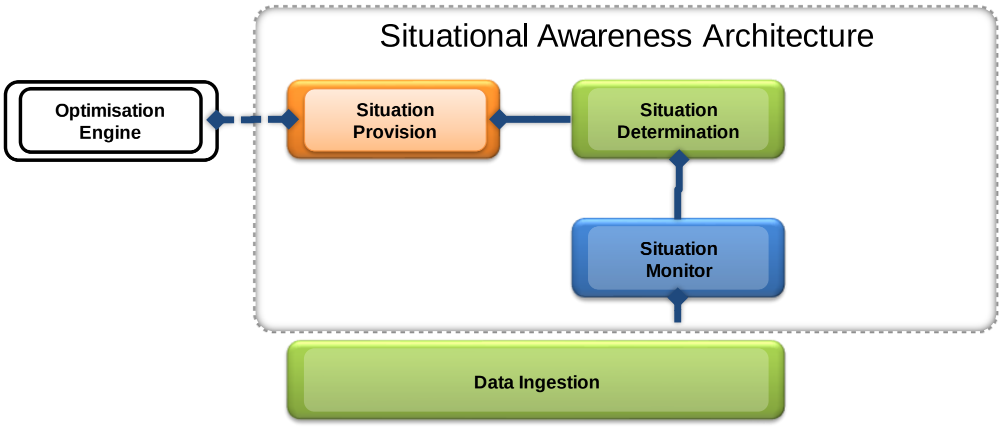


Figure 2: Conceptual Situational Awareness Architecture

The Situation Determination uses monitored "raw data" provided by the SAFIRE data ingestion NiFi templates, which get data directly from the legacy systems, or the predictive analytics for the product and processes, as well as knowledge available in different systems, to derive the product/machine/process current situation. Using the situation model the monitored data are being evaluated and the situation determined.

The workflow of the Situation Determination Services from data acquisition to the publication of the situation to the other modules, is presented in the following figure.

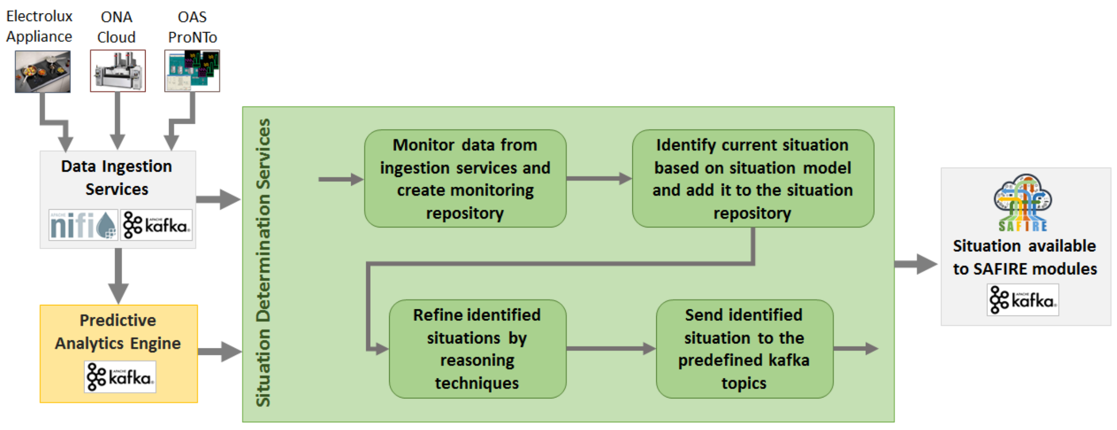

Electrolux Appliance
ONA Cloud
OAS ProNTo
Data Ingestion Services
nifi
kafka
Predictive Analytics Engine
kafka
Situation Determination Services
Monitor data from ingestion services and create monitoring repository
Identify current situation based on situation model and add it to the situation repository
Refine identified situations by reasoning techniques
Send identified situation to the predefined kafka topics
SAFIRE
Situation available to SAFIRE modules
kafka

Figure 3: Situation Determination Workflow

As shown in the picture above, the Situation Determination Services receive the factory or product data through the data ingestion services using NiFi and Kafka. The Situation Monitoring part receives and structures the data accordingly, and forwards them in the appropriate format to the Determination part for situation identification. Using reasoning techniques, the Determination part refines the situation identification and publishes the result to the Kafka messaging system, where it becomes available to the other modules of the SAFIRE solution.

In more detail, the Situation Determination Services workflow is being described in the following sections.

2.1 DATA INGESTION TO SITUATION DETERMINATION SERVICES

Apache NiFi is used as a part of data injection to the SAFIRE modules. The NiFi processors support reading data from different data sources and can be configured to read the data periodically or to read only the new data. With this configurations NiFi can be adjusted to have the optimal reading pattern for each type of data.

As Business Case (BC)-specific configuration (explained in detail in chapter 5), in the case of OAS, NiFi is used to read the contents of the Orders table in the database only once, and the contents of the Mixer Status table periodically to have all the relevant information. In the case of Electrolux, the Microsoft Excel files serve as data source. NiFi processors are used to read the contents of a given folder and to extract the data spreadsheets from the Excel columns. This happens only once to avoid duplicate and irrelevant data, and is updated with the new files as long as those are saved in the given repository. In the ONA case, NiFi is used for the combination of database reading and ONA-cloud API-connection. To prevent duplicates, only the new data are being read.

These examples show that the relevant data are being read from different sources and the reading schemas must be configured differently according to the use case. The read data also must be formatted and sent to respective Kafka topics. NiFi offers good configuration possibilities to ensure the needed reading and transformation.

2.2 SITUATION MONITORING

The Situation Monitoring part of the services is responsible for feeding the module with the SAFIRE data, by using a Kafka for communication. To create a general solution for the communication with Kafka, the Kafka Monitor class was created in the Situation Monitoring part. The monitors (one monitor for each monitored case) continuously check the relevant topics for the new data provided by the data ingestion module(s). The implementation of the Monitor class is generic and can be used for any business case, because it implements the key behaviour of a Kafka consumer: reading from the given topics periodically. If new data are available, the monitored data are being transformed into a data format that is usable by Situation Determination and send to the Situation Determination service.

Docker configuration

The Situation monitoring module is set up to be deployed in a Docker container and run the corresponding jar executable. Therefore a docker-maven-plugin was used[1], allowing an implicit configuration of the Docker container out of the maven environment. At the moment the Situation monitoring container is set up to run on ATB's Docker host machine, to be reached under http://192.168.15.17:2376.

2.3 SITUATION DETERMINATION

The Situation Determination Service continuously listens for monitored data provided by the Situation Monitoring service. If new monitoring data are available, Situation Determination tries to identify the current situation based on the monitored data, the situation model and previously stored identified situations. The current identified situation is stored in the Situation Repository. Furthermore, the current identified situation is posted as a kafka topic into the SAFIRE kafka cluster.

Docker configuration

The Situation determination module is set up to be deployed in a Docker container and run the corresponding jar executable. At the moment the Situation determination container is set up to run on ATB's Docker host machine, to be reached under http://192.168.15.17:2376.

2.4 SITUATION MODEL

The SAFIRE situation model is modelled in OWL, which is an open standard ontology modelling language. This allows the use of tools such as Protégé to develop and manage the ontology, as well as query and manipulate it through RDF compatible methods and tools such as Jena, SPARQL, etc.

The situation model for the full prototype supports the core concepts of Activity, Actor, Information, Product, Production Process and Resource. For the specific purpose of the three business cases, mainly the concept Information is extended to include BC-specific concepts that describe the selected data for the SAFIRE integrated operation of all modules. The BC-specific situation models can be found in the appendix. Enterprises can extend the generic or the BC-specific SAFIRE situation models to better suit their domain, by defining sub-classes for the core concepts (see Section 8 Appendix).

A primary definition of the SAFIRE situation model and more details on its development, is given in D4.2, and the details of the final version have been described in D4.5.

[1] https://github.com/fabric8io/docker-maven-plugin

2.5 Implementation of the Repository

The Situation Repository is implemented in a layered style. In the vey back end, a relational database (e.g. MySQL) is used to provide the storage. Above the relational database, SDB[2] is used to realize RDF storage and query. Other modules of Situation Determination Services manipulate and query the repository through a manipulation layer, which uses Jena API and SPRQL to communicate with SDB. The API provided by the Situation Manipulation Layer is on a higher level than one which simply adds and removes RDF statements from the Jena model, such as: create a new situation instance, delete orphaned information, and update situation resources and so on. This makes it easier to manipulate situational information inside the repository. Besides, it also makes sure the Repository is consistent, as all performed operations are controlled by the manipulation layer.

2.6 Implemented Functionalities

All specified functionality for the Full Prototype of the Situational Awareness Module has been implemented. Some of the already implemented functionality need to be refined within the development of the Full Prototype. An overview of the functionality, implemented is listed in the following table.

Table 1: Overview of implemented functionality

No.	Requirement	Overall Priority	Status
U54	Able to change existing or adding new monitoring sources with min. effort	SHALL	Implemented.
U55	Able to support collection of environmental data to identify situations	SHALL	Implemented.
U56	Able to support collection of operator's behaviours to identify current situation	SHALL	Partially Implemented
U57	Able to monitor machine current status data to identify situation	SHALL	Implemented.
U58	Able to monitor machine health status to identify current situation	SHALL	Implemented.
U59	Able to monitor overall equipment effectiveness (OEE) to identify current situation	SHALL	Implemented.
U60	Able to monitor production status to identify current situation	SHALL	Implemented.
U61	Able to support collection of data from proNTo behaviours to identify current situation	SHALL	Implemented. Situation can be identified for selected behaviours, such as Mixer availability and production orders.

[2] http://openjena.org/SDB/

U62	Able to monitor Hob Temperature status to identify current situation	SHALL	Partially Implemented. Currently supports Electrolux lab environment.
U63	Able to monitor Pot Boiling status to identify current situation	SHALL	Partially Implemented. Currently supports Electrolux lab environment.
U64	Able to provide situational information based on raw and monitored data	SHALL	Implemented.
U65	Able to extract situational information from monitored machines	SHALL	Implemented.
U66	Able to dynamically extract situational information from sensor data	SHALL	Implemented.
U67	Able to change existing or add new situations with minimal effort	SHALL	Implemented.
U68	Able to model situations under which a set of machines is operating	SHALL	Implemented
U69	Able to model situations under which a production process is operating	SHALL	Implemented
U70	Able to extract situational information from sets of related machines	SHOULD	Implemented
U71	Able to extract situational information from operator actions	SHOULD	Partially implemented. Currently supports OAS lab environment.
U72	Able to evaluate situation with respect to capacity, performance, availability (OEE) of monitored machines	SHALL	Not implemented yet
U73	Able to evaluate situation with respect to capacity, performance, availability (OEE) from sets of related machines	SHALL	Not implemented yet
U74	Able to anticipate alarms before they occur based on current situation	SHALL	Partially implemented. Currently implemented in OAS lab environment.
U75	Able to evaluate status of machine job queues (if available)	SHOULD	Implemented.
U76	Able to model situation under which proNTo is operating	SHOULD	Implemented using Protégé as modelling tool
U77	Able to extract situational information from proNTo and from other systems	SHALL	Implemented for proNTo

3. INTEGRATION WITH OTHER MODULES

The Full Prototype of the Situational Awareness services are integrated with the following modules:

- Data-Ingestion: The data ingestion modules are kafka producers, that periodically post information observed from the systems of industrial partners into the SAFIRE kafka cluster. The following data ingestion modules are available:
  - OAS proNTo: The data ingestion module in the OAS case connects to the Oracle database server of the proNTo system (simulated factory) and ingests the data required by SAFIRE modules into the kafka cluster.
  - ONA Cloud: The data ingestion module in the ONA case connects to the Oracle database server of the ONA cloud (real machines) and ingests the data required by SAFIRE modules into the kafka cluster
  - Electrolux: The data ingestion module in the Electrolux case connects to the data provided by the experimental cooker setup. Results are read from Matlab/CSV files and ingests the data required by SAFIRE modules into the kafka cluster.
- **Metrics API**: The main interface for transferring monitored and situational data between the SD services and the optimisation engine, allowing to build a valid *optimisation configuration* which can be parsed by the Optimisation Engine; described in chapter 3.1.
- **Optimisation Engine**: The SD services send an optimisation configuration containing the monitored/situational data via Kafka to the Optimisation Engine which produces an optimisation result. This result will be fed back to the SD services via Kafka.
- **Predictive Analytics**: The SD services receive predictive analytics results from Kafka and enrich it with recent situational data.

3.1 METRICS API

This section describes the Metrics API as Interface between the SD services and the Optimisation Engine. The aim of this module is to provide a common interface for sending monitored and situational data from each BC to the Optimisation Engine. The following section will briefly show the use of the Metrics API taking the OAS BC as an example.

Every time the SD service sends monitoring or situational data to the OE a Metrics API *configuration* will be constructed. Such a configuration consists of different parameters, such as the *Controlled Metrics*, the *Key Objective Metrics* and the *Observable Metrics*. Since the monitored and situational data coming from the legacy systems can be identified as the observable set of data, The *Observable Metrics* will contain exactly these observations. In the OAS BC the observations are the amount of paint to be produced (resp. amount of paint which already has been produced) and the status of a mixer. Additionally the Observable Metrics contains the recipe information, i.e. a mapping of each type of paint to be produced to the mixers, where this specific type of paint can be produced.

The *Key Objective Metrics* refers to the Key Objective function of the OE. It contains the optimisation objective, i.e. in the OAS BC the minimisation of the surplus and makespan for each produced paint.

The *Controlled Metrics* defines the parameter which can be controlled in order to achieve the optimisation objective, i.e. the assignment of mixers to the type of paint to be produced at a specific time.

3.2 INTEGRATION WITH OPTIMISATION ENGINE

The integration between the SD services and the Optimisation Engine is being realised by sending a valid optimisation configuration built accordingly to the Metrics API to a specific Kafka topic.

3.3 INTEGRATION WITH PREDICTIVE ANALYTICS

The integration between the SD services and the Predictive Analytics module is being realised by receiving predictive analytics results in JSON format from a specific Kafka topic, which will then be processed with recent situational data and send back into Kafka for further usage.

4. INSTALLATION, CONFIGURATION AND USAGE

This section describes the installation, configuration and usage of the Full Prototype of the Situational Awareness module. The business case specific customisation is described in Section 5.

4.1 INSTALLATION

The Full Prototype requires Java and Apache Kafka as prerequisites. The installation of Apache Kafka is described in D2.3 *Early Prototype of Predictive Analytics Platform* and therefore not repeated in this deliverable. The Full Prototype can also be deployed as Docker container.

The next sections describe the installation and configuration of both Situational Awareness services.

4.1.1 Standalone installation

The Situation Monitoring and Situation Determination services can be downloaded from https://www.atb-bremen.de/artifactory/ext-releases-local/eu/safire-project/

The Situational Awareness service are available in customised versions for the three SAFIRE business cases, hence the naming of the executables is accordingly. After downloading the jar files for monitoring and determination, the services can be started with the following commands:

```
java -jar situation-monitoring-xxx-1.0.0.jar
java -jar situation-determination-xxx-1.0.0.jar
```

where "xxx" has to be replaced by either "oas", "electrolux" or "ona".

4.1.2 Docker container deployment

The docker containers for Situation Monitoring and Situation Determination can be downloaded from http://gitlab.atb-bremen.de by using the following commands:

```
docker pull gitlab.atb-bremen.de:5555/safire/context-monitoring
docker pull gitlab.atb-bremen.de:5555/safire/context-dtermination
```

After downloading the container image, it can be started with:

```
docker start situation-monitoring
docker start situation-determination
```

In the docker registry two business case specific container images are available, which can be used for the OAS and ONA use case.

4.2 CONFIGURATION

The Situation Monitoring service and the Situation Determination service need to be executed prior to its execution. The following two subsections describe the configuration of the services as well as the configuration of the business case specific Situation Monitoring customisations.

4.2.1 Services Configuration

The Situation Monitoring as well as the Situation Determination service will be configured through an XML configuration file. An example for such a configuration can be seen in the following listing:

```
<?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="Monitoring">
         <host>localhost</host>
         <location>http://localhost:19001</location>
         <name>MonitoringService</name>
         <server>de.atb.context.services.MonitoringService</server>
         <proxy>de.atb.context.services.IMonitoringService</proxy>
      </service>
      <service id="MonitoringRepository">
         <host>localhost</host>
         <location>http://localhost:19002</location>
         <name>MonitoringDataRepositoryService</name>
         <server>de.atb.context.services.MonitoringDataRepositoryService</server>
         <proxy>de.atb.context.services.IMonitoringDataRepositoryService</proxy>
      </service>
      <service id="SituationDeterminationService">
         <host>localhost</host>
         <location>http://localhost:19004</location>
         <name>ContextExtractionService</name>
         <server>de.atb.context.services.ContextExtractionService</server>
         <proxy>de.atb.context.services.IContextExtractionService</proxy>
      </service>
      <service id="SituationDeterminationRepositoryService">
         <host>localhost</host>
         <location>http://localhost:19005</location>
         <name>ContextRepositoryService</name>
         <server>de.atb.context.services.ContextRepositoryService</server>
         <proxy>de.atb.context.services.IContextRepositoryService</proxy>
      </service>
   </services>
</config>
```

Code 1 – Example of Situation Awareness Service Configuration

4.2.2 Monitoring Configuration

As described in the previous sections, the Situation Monitoring need to be customised per Business Case to allow gathering of information, that are specific for each installation. A description on how to customise the Situation Monitoring is presented in D4.2. The following listing, gives an example on how to configure the Situation Monitoring service, so that it uses the business case specific extensions:

```
<?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-ona" location="indexes/ona"></index>
   </indexes>
   <datasources>
      <datasource id="datasource-ona" type="webservice"
         monitor="de.atb.context.monitoring.monitors.webservice.WebServiceMonitor"
         uri=" https://onaedm.savvyds.com/ic/"
         options="username&password"
      class="de.atb.context.monitoring.config.models.datasources.WebServiceDataSource"
   />
   </datasources>
   <interpreters>
      <interpreter id="interpreter-ona">
         <configuration type="*"
            parser="de.atb.context.monitoring.parser.onacloud.ONACloudParser"
            analyser="de.atb.context.monitoring.analyser.onacloud.ONACloudAnalyser" />
      </interpreter>
   </interpreters>
   <monitors>
      <monitor id="monitor-ona" datasource="datasource-ona"
         interpreter="interpreter-ona"
         index="index-ona" />
   </monitors>
</config>
```

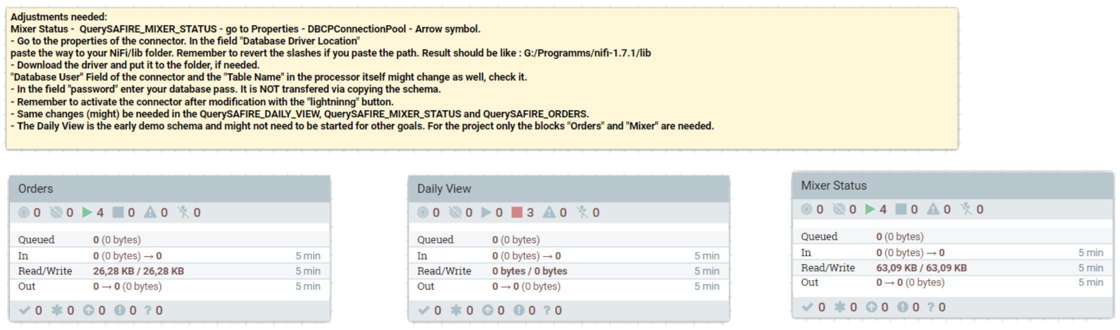
Code 2 – Example Monitoring Plugin Configuration

5. BUSINESS CASE SPECIFIC CUSTOMISATION

5.1 OAS

5.1.1 Data Ingestion

For the data ingestion of the OAS case a specific NiFi template with the necessary processors has been developed (see Figure 4). The goal of the OAS NiFi template is to gather information related to the paint production process, i.e. information about the current status of mixers, recent orders and (historical) batch data. The goal of the OAS template is to connect to the ProNTo database and retrieve data for each required database table. The connection will be established via the Oracle database driver, hence *QueryDatabaseTable* processors are being used to build the connection. At the moment the processors are configured for pulling data each *n* seconds, where *n* can be modified within the *QueryDatabaseTable* processor (default: 10 seconds). The retrieved data will then be further processed, converted into readable JSON format and published into Kafka. See Figure 5 for the Nifi template for gathering the (historical) batch data, Figure 6 for the template to acquire the current mixer status, and Figure 7 for the current orders of paint to be produced.



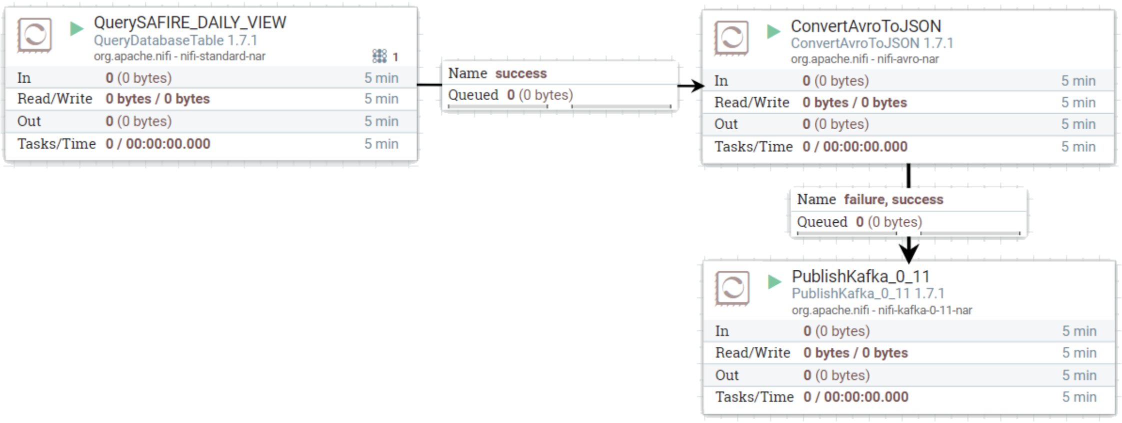
Adjustments needed:
Mixer Status - QuerySAFIRE_MIXER_STATUS - go to Properties - DBCPConnectionPool - Arrow symbol.
- Go to the properties of the connector. In the field "Database Driver Location"
paste the way to your NiFi/lib folder. Remember to revert the slashes if you paste the path. Result should be like : G:/Programms/nifi-1.7.1/lib
- Download the driver and put it to the folder, if needed.
"Database User" Field of the connector and the "Table Name" in the processor itself might change as well, check it.
- In the field "password" enter your database pass. It is NOT transfered via copying the schema.
- Remember to activate the connector after modification with the "lightninng" button.
- Same changes (might) be needed in the QuerySAFIRE_DAILY_VIEW, QuerySAFIRE_MIXER_STATUS and QuerySAFIRE_ORDERS.
- The Daily View is the early demo schema and might not need to be started for other goals. For the project only the blocks "Orders" and "Mixer" are needed.

```
Orders
◎ 0  ⊘ 0  ▶ 4  ■ 0  ⚠ 0  ✗ 0
Queued      0 (0 bytes)
In          0 (0 bytes) → 0               5 min
Read/Write  26,28 KB / 26,28 KB           5 min
Out         0 → 0 (0 bytes)               5 min
✓ 0  ✱ 0  ⊙ 0  ! 0  ? 0

Daily View
◎ 0  ⊘ 0  ▶ 0  ■ 3  ⚠ 0  ✗ 0
Queued      0 (0 bytes)
In          0 (0 bytes) → 0               5 min
Read/Write  0 bytes / 0 bytes             5 min
Out         0 → 0 (0 bytes)               5 min
✓ 0  ✱ 0  ⊙ 0  ! 0  ? 0

Mixer Status
◎ 0  ⊘ 0  ▶ 4  ■ 0  ⚠ 0  ✗ 0
Queued      0 (0 bytes)
In          0 (0 bytes) → 0               5 min
Read/Write  63,09 KB / 63,09 KB           5 min
Out         0 → 0 (0 bytes)               5 min
✓ 0  ✱ 0  ⊙ 0  ! 0  ? 0
```

Figure 4: NiFi Template Groups for the OAS Data Ingestion Module



QuerySAFIRE_DAILY_VIEW
QueryDatabaseTable 1.7.1
org.apache.nifi - nifi-standard-nar
1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

ConvertAvroToJSON
ConvertAvroToJSON 1.7.1
org.apache.nifi - nifi-avro-nar

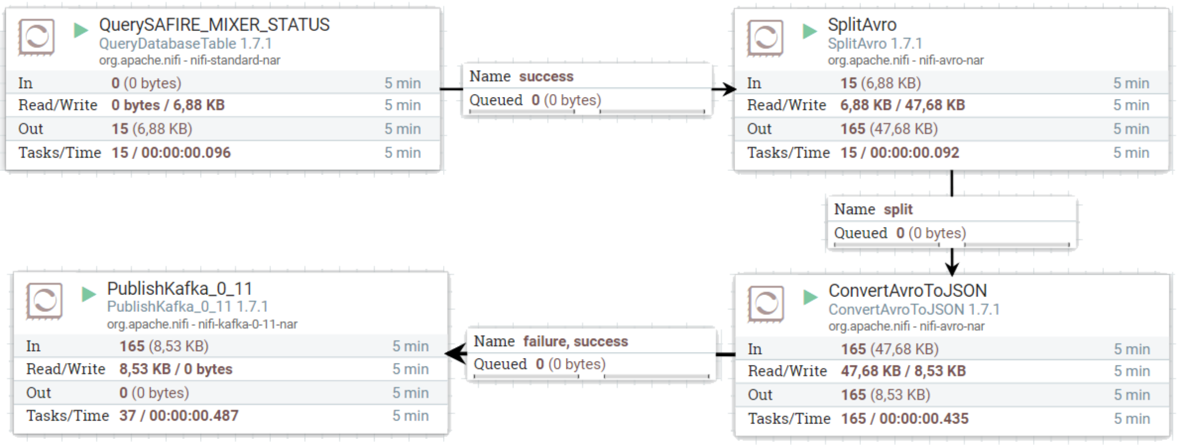
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name failure, success
Queued 0 (0 bytes)

PublishKafka_0_11
PublishKafka_0_11 1.7.1
org.apache.nifi - nifi-kafka-0-11-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Figure 5: OAS NiFi Template for the Daily View Data



QuerySAFIRE_MIXER_STATUS
QueryDatabaseTable 1.7.1
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 6,88 KB	5 min
Out	15 (6,88 KB)	5 min
Tasks/Time	15 / 00:00:00.096	5 min

Name success
Queued 0 (0 bytes)

SplitAvro
SplitAvro 1.7.1
org.apache.nifi - nifi-avro-nar

In	15 (6,88 KB)	5 min
Read/Write	6,88 KB / 47,68 KB	5 min
Out	165 (47,68 KB)	5 min
Tasks/Time	15 / 00:00:00.092	5 min

Name split
Queued 0 (0 bytes)

ConvertAvroToJSON
ConvertAvroToJSON 1.7.1
org.apache.nifi - nifi-avro-nar

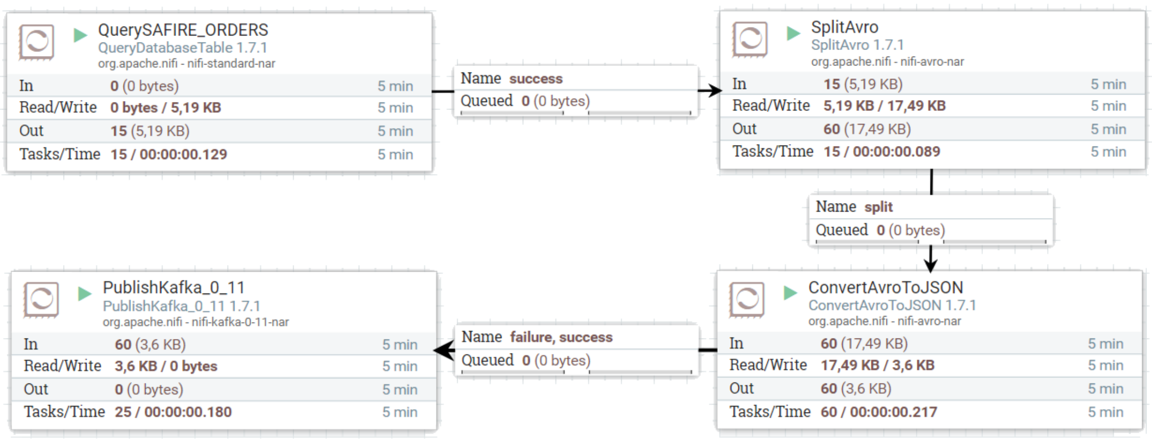
In	165 (47,68 KB)	5 min
Read/Write	47,68 KB / 8,53 KB	5 min
Out	165 (8,53 KB)	5 min
Tasks/Time	165 / 00:00:00.435	5 min

Name failure, success
Queued 0 (0 bytes)

PublishKafka_0_11
PublishKafka_0_11 1.7.1
org.apache.nifi - nifi-kafka-0-11-nar

In	165 (8,53 KB)	5 min
Read/Write	8,53 KB / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	37 / 00:00:00.487	5 min

Figure 6: OAS NiFi Template for the Mixer Status Data



QuerySAFIRE_ORDERS
QueryDatabaseTable 1.7.1
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 5,19 KB	5 min
Out	15 (5,19 KB)	5 min
Tasks/Time	15 / 00:00:00.129	5 min

Name success
Queued 0 (0 bytes)

SplitAvro
SplitAvro 1.7.1
org.apache.nifi - nifi-avro-nar

In	15 (5,19 KB)	5 min
Read/Write	5,19 KB / 17,49 KB	5 min
Out	60 (17,49 KB)	5 min
Tasks/Time	15 / 00:00:00.089	5 min

Name split
Queued 0 (0 bytes)

ConvertAvroToJSON
ConvertAvroToJSON 1.7.1
org.apache.nifi - nifi-avro-nar

In	60 (17,49 KB)	5 min
Read/Write	17,49 KB / 3,6 KB	5 min
Out	60 (3,6 KB)	5 min
Tasks/Time	60 / 00:00:00.217	5 min

Name failure, success
Queued 0 (0 bytes)

PublishKafka_0_11
PublishKafka_0_11 1.7.1
org.apache.nifi - nifi-kafka-0-11-nar

In	60 (3,6 KB)	5 min
Read/Write	3,6 KB / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	25 / 00:00:00.180	5 min

Figure 7: OAS NiFi Template for the Orders Data

5.1.2 Situation Monitoring

5.1.2.1 OAS Monitor

The OAS monitor observes the data in the OAS proNTo system for situational changes. proNTo acts as a manufacturing execution system (MES) but covers also features of supervisory control and data acquisition (SCADA) systems and Enterprise-Resource-Planning (ERP) systems.

The proNTo database stores and manages all kinds of data in the proNTo system (live, master and historical/protocol data). Examples are:

- Batch protocols - Information about (timestamps; executed process steps; comparing actual versus target measurements; used production lines, etc.)
- Product Recipes - Information about how to produce specific products (defined Processes; needed materials; mixing ratio of materials; etc.)
- Status information - Information about the current abilities and status of the factory (e.g. which product recipe is free/approved to use on which lines)

The Full Prototype of the OAS monitor reads data from an Apache kafka node. The data ingestion module establishes the direct connection to the proNTo system using NiFi and sends needed information into the kafka node. The OAS monitor reads the data from the kafka node. See Figure 8 for an overview.

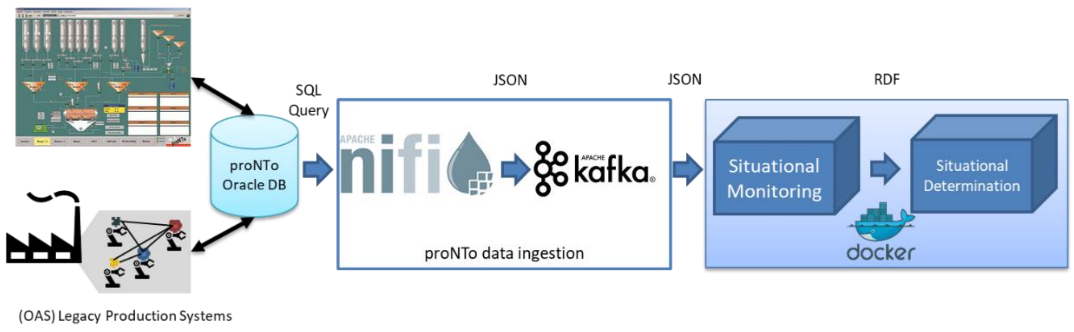


Figure 8: Data transfer from the OAS proNTo system to the OAS monitor

The Full Prototype of the OAS monitor has a permanent loop of watching the current mixers status and the orders to be done for the current day. The following message snippets show some examples for data monitored by the OAS monitor.

<u>**Current mixer status**</u>

Kafka topic: `ResourceAvailability`

Data in kafka:

```
[
MonitoredMixerStatusInformation{ID='1', m_name='Mischer 1', m_status='1'},
MonitoredMixerStatusInformation{ID='3', m_name='Mischer 3', m_status='1'},
MonitoredMixerStatusInformation{ID='5', m_name='Mischer 5', m_status='1'},
MonitoredMixerStatusInformation{ID='2', m_name='Mischer 2', m_status='1'},
MonitoredMixerStatusInformation{ID='4', m_name='Mischer 4', m_status='1'},
MonitoredMixerStatusInformation{ID='6', m_name='Mischer 6', m_status='1'},
MonitoredMixerStatusInformation{ID='7', m_name='Mischer 7', m_status='1'},
MonitoredMixerStatusInformation{ID='8', m_name='Mischer 8', m_status='1'},
]
```

Code 3: kafka data for mixer status

Orders

Kafka topic: `OrderDataTopic`

Data in kafka:

```
[
MonitoredOrdersInformation{ID='1', ord_name='Std Weiss', ord_amount='145000'},
MonitoredOrdersInformation{ID='2', ord_name='Weiss Matt', ord_amount='165000'},
MonitoredOrdersInformation{ID='4', ord_name='Weiss Basis', ord_amount='126000'},
MonitoredOrdersInformation{ID='3', ord_name='W Super Glanz', ord_amount='56000'}
]
```

Code 4: kafka data for orders

Data model

At the moment, two data classes are used, which hold the information about the monitored data from the proNTo system matching the mixer status and orders. Figure 9 shows the relationship between the different data classes: The class *ProntoDataModel* holds the main proNTo data model, which is being specified in detail within the *ProntoInstance,* which on its part holds a relation to the both data classes *MonitoredMixerStatusInformation* and *MonitoredOrderInformation.* The last two classes contain the sensor information from a proNTo machine.

MonitoredMixerStatusInformation: Currently, the ID, the mixer's name and its status is being monitored.

MonitoredOrderInformation: Currently, the Order ID, the name of the order and the amount for this specific order is being monitored.

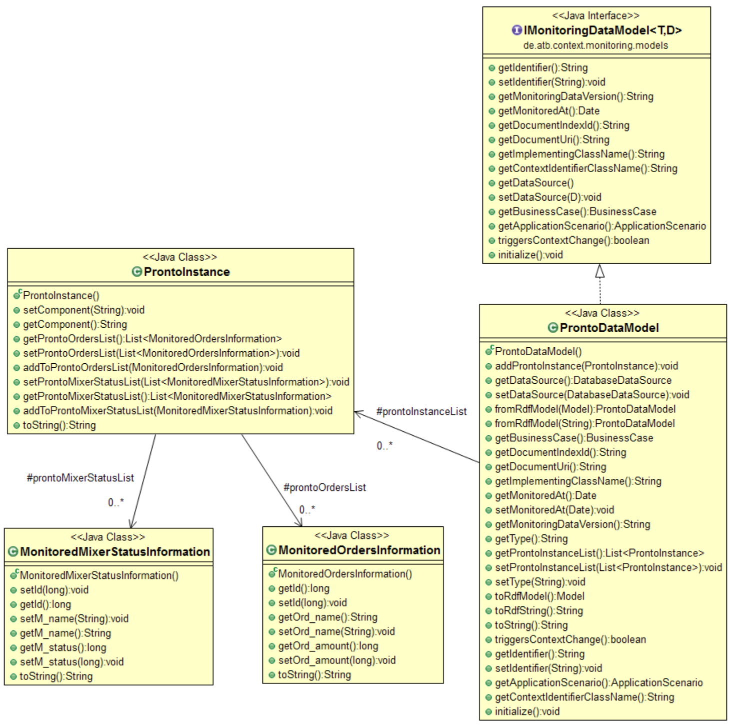

<<Java Interface>>
IMonitoringDataModel<T,D>
de.atb.context.monitoring.models
getIdentifier():String
setIdentifier(String):void
getMonitoringDataVersion():String
getMonitoredAt():Date
getDocumentIndexId():String
getDocumentUri():String
getImplementingClassName():String
getContextIdentifierClassName():String
getDataSource()
setDataSource(D):void
getBusinessCase():BusinessCase
getApplicationScenario():ApplicationScenario
triggersContextChange():boolean
initialize():void
<<Java Class>>
ProntoInstance
ProntoInstance()
setComponent(String):void
getComponent():String
getProntoOrdersList():List<MonitoredOrdersInformation>
setProntoOrdersList(List<MonitoredOrdersInformation>):void
addToProntoOrdersList(MonitoredOrdersInformation):void
setProntoMixerStatusList(List<MonitoredMixerStatusInformation>):void
getProntoMixerStatusList():List<MonitoredMixerStatusInformation>
addToProntoMixerStatusList(MonitoredMixerStatusInformation):void
toString():String
#prontoInstanceList
0..*
#prontoMixerStatusList
0..*
#prontoOrdersList
0..*
<<Java Class>>
ProntoDataModel
ProntoDataModel()
addProntoInstance(ProntoInstance):void
getDataSource():DatabaseDataSource
setDataSource(DatabaseDataSource):void
fromRdfModel(Model):ProntoDataModel
fromRdfModel(String):ProntoDataModel
getBusinessCase():BusinessCase
getDocumentIndexId():String
getDocumentUri():String
getImplementingClassName():String
getMonitoredAt():Date
setMonitoredAt(Date):void
getMonitoringDataVersion():String
getType():String
getProntoInstanceList():List<ProntoInstance>
setProntoInstanceList(List<ProntoInstance>):void
setType(String):void
toRdfModel():Model
toRdfString():String
toString():String
triggersContextChange():boolean
getIdentifier():String
setIdentifier(String):void
getApplicationScenario():ApplicationScenario
getContextIdentifierClassName():String
initialize():void
<<Java Class>>
MonitoredMixerStatusInformation
MonitoredMixerStatusInformation()
setId(long):void
getId():long
setM_name(String):void
getM_name():String
getM_status():long
setM_status(long):void
toString():String
<<Java Class>>
MonitoredOrdersInformation
MonitoredOrdersInformation()
getId():long
setId(long):void
getOrd_name():String
setOrd_name(String):void
getOrd_amount():long
setOrd_amount(long):void
toString():String

Figure 9: Data model for monitored mixer status and orders

<u>Monitor Implementation</u>

For the monitoring process of the proNTo BC a *Database Monitor* is being used, as the sensor data from the proNTo Legacy Systems is being stored within a database. Figure 10 shows the relationship and inheritance between the *ProntoAnalyser* and the more generic *DatabaseAnalyser*. The main task of the *ProntoAnalyser* is to gather information about the mixer status and orders from the database. According to the architecture the data will be fetched from a *Kafka Cluster*. Therefore, *Kafka Consumers* for the different data topics are being instantiated, who are continuously polling data from the cluster. The data are being transferred into the data model shown in Figure 9 and thereafter stored in the *Monitoring Repository*.

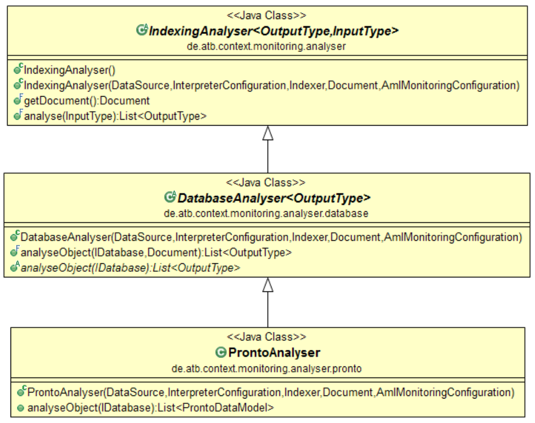

<<Java Class>>
IndexingAnalyser<OutputType,InputType>
de.atb.context.monitoring.analyser
IndexingAnalyser()
IndexingAnalyser(DataSource,InterpreterConfiguration,Indexer,Document,AmIMonitoringConfiguration)
getDocument():Document
analyse(InputType):List<OutputType>
<<Java Class>>
DatabaseAnalyser<OutputType>
de.atb.context.monitoring.analyser.database
DatabaseAnalyser(DataSource,InterpreterConfiguration,Indexer,Document,AmIMonitoringConfiguration)
analyseObject(IDatabase,Document):List<OutputType>
analyseObject(IDatabase):List<OutputType>
<<Java Class>>
ProntoAnalyser
de.atb.context.monitoring.analyser.pronto
ProntoAnalyser(DataSource,InterpreterConfiguration,Indexer,Document,AmIMonitoringConfiguration)
analyseObject(IDatabase):List<ProntoDataModel>

Figure 10: Inheritance structure of the Analyser used for proNTo

Detailed Data Flow

Figure 11 shows a detailed overview of the dataflow between the legacy system, *NiFi*, *Kafka* and the situation awareness, optimisation and visualization modules. All data processing is done via the Kafka cluster. As seen in the figure, the order information and mixer status are being processed by *NiFi* and have their corresponding topic within the *Kafka* node, which will be monitored subsequently by the *Situation Awareness* module. Additionally, historical batch data from the proNTo database will be shown in a graphical calendar representation by a visualisation module. This module will also receive the analysed data from the Optimisation Engine with an optimized Batch scheduling for the monitored orders.

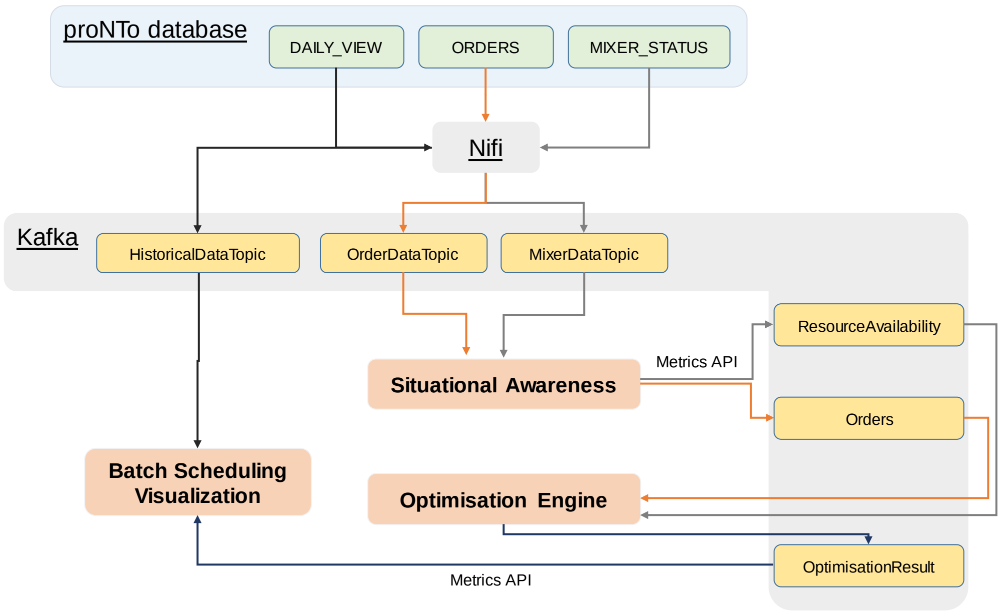


Figure 11: Detailed Data Flow for the OAS scenario

5.1.3 Situation Determination

5.1.3.1 *OAS Situation Identification*

The OAS specific implementation of the ContextIdentifier executes SPARQL queries to identify data from the monitored Situation Monitoring service. Code 5 below shows an example of a monitored data item, that is used to identify Situations.

```
...
  <rdf:Description rdf:about="#MonitoredMixerStatusInformation/275807757">
    <oas:m_status rdf:datatype="#long">-1</oas:m_status>
    <oas:m_name rdf:datatype="#string">Mischer  5</oas:m_name>
    <oas:id rdf:datatype="#long">5</oas:id>
    <rdf:type rdf:resource="#MonitoredMixerStatusInformation"/>
  </rdf:Description>
  <rdf:Description rdf:about="#MonitoredMixerStatusInformation/416448361">
    <oas:m_status rdf:datatype="#long">1</oas:m_status>
    <oas:m_name rdf:datatype="#string">Mischer  10</oas:m_name>
    <oas:id rdf:datatype="#long">210</oas:id>
    <rdf:type rdf:resource="#MonitoredMixerStatusInformation"/>
  </rdf:Description>
  <rdf:Description rdf:about="#MonitoredMixerStatusInformation/59758731">
    <oas:m_status rdf:datatype="#long">1</oas:m_status>
    <oas:m_name rdf:datatype="#string">Mischer  10</oas:m_name>
    <oas:id rdf:datatype="#long">210</oas:id>
    <rdf:type rdf:resource="#MonitoredMixerStatusInformation"/>
  </rdf:Description>
...
  <rdf:Description rdf:nodeID="A1">
    <rdf:_5 rdf:resource="#MonitoredMixerStatusInformation/275807757"/>
    <rdf:_4 rdf:resource="#MonitoredMixerStatusInformation/567702719"/>
    <rdf:_10 rdf:resource="#MonitoredMixerStatusInformation/1718853555"/>
    <rdf:_12 rdf:resource="#MonitoredMixerStatusInformation/2077540885"/>
    <rdf:_20 rdf:resource="#MonitoredMixerStatusInformation/416448361"/>
    <rdf:_8 rdf:resource="#MonitoredMixerStatusInformation/820578646"/>
    <rdf:_22 rdf:resource="#MonitoredMixerStatusInformation/35181829"/>
    <rdf:_19 rdf:resource="#MonitoredMixerStatusInformation/238720618"/>
    <rdf:_3 rdf:resource="#MonitoredMixerStatusInformation/1783590156"/>
    <rdf:_14 rdf:resource="#MonitoredMixerStatusInformation/1994882064"/>
    <rdf:_18 rdf:resource="#MonitoredMixerStatusInformation/144902456"/>
    <rdf:_2 rdf:resource="#MonitoredMixerStatusInformation/244690718"/>
    <rdf:_15 rdf:resource="#MonitoredMixerStatusInformation/198293959"/>
    <rdf:_9 rdf:resource="#MonitoredMixerStatusInformation/595233104"/>
    <rdf:_6 rdf:resource="#MonitoredMixerStatusInformation/859758146"/>
    <rdf:_17 rdf:resource="#MonitoredMixerStatusInformation/1172961897"/>
    <rdf:_21 rdf:resource="#MonitoredMixerStatusInformation/316859560"/>
    <rdf:_7 rdf:resource="#MonitoredMixerStatusInformation/571011620"/>
    <rdf:_11 rdf:resource="#MonitoredMixerStatusInformation/59758731"/>
    <rdf:_16 rdf:resource="#MonitoredMixerStatusInformation/1447564994"/>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
    <rdf:_13 rdf:resource="#MonitoredMixerStatusInformation/211018806"/>
    <rdf:_1 rdf:resource="#MonitoredMixerStatusInformation/1838911889"/>
  </rdf:Description>
...
```

Code 5 – Example of Monitoring Data in RDF representation (excerpt)

Situation Identifier Implementation

The OAS specific implementation of the Situation Determination executes queries, such as the following:

```
Select ?mixer ?mixerId ?mixerName ?mixerStatus
where
{
 ?mixer rdf:type oas:Mixer.
 ?mixer oas:MonitoredMixerStatusInformation ?mixerInfo.
 ?mixerInfo oas:m_id ?mixerId.
 ?mixerInfo oas:m_name ?mixerName.
 ?mixerInfo oas:m_status ?mixerStatus.
};
```

Figure 12 shows the inheritance structure of the OAS specific situation identifier. The data previously observed by the Situation Monitoring is used to identify the situations based on the monitored data.

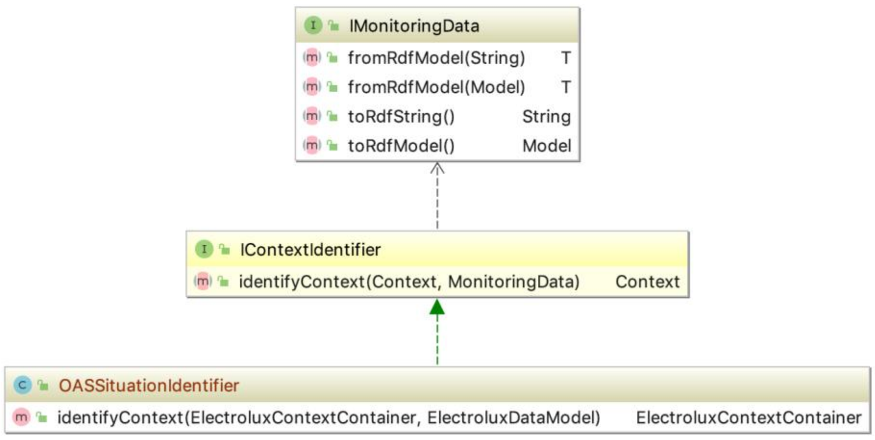

IMonitoringData
fromRdfModel(String) T
fromRdfModel(Model) T
toRdfString() String
toRdfModel() Model
IContextIdentifier
identifyContext(Context, MonitoringData) Context
OASSituationIdentifier
identifyContext(ElectroluxContextContainer, ElectroluxDataModel) ElectroluxContextContainer

Figure 12: Inheritance structure of the OAS Situation Identifier used for proNTo

5.1.3.2 *Rule based reasoning*

The OAS specific rules used in the OAS case for situation reasoning is based on the Jena Inference Engine:

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

Thereby, the business case specific rules are stored in the `ruleURL`. Code 6 – shows an example, which can be explained as “if a production line has a mixer attached to it, and this mixer is observed by volume sensor which provides a resource identified as volume in cm^3, this production line is of type paint production line”.

```
@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://www.safire-factories.org/base.owl#> .
[rule1:          (?a safire:hasDevicePart ?b)
                 (?b rdf:type safire:Mixer)
                 (?b safire:isObservedBy ?c)
                 (?c rdf:type safire:VolumeSensor)
                 (?c safire:providesVolume ?d)
                 (?d rdf:type safire:CM3)
        -> (?a rdf:type safire:PaintProductionLine)
]
```

Code 6 – Example Rule for Rule-Based Context Reasoning

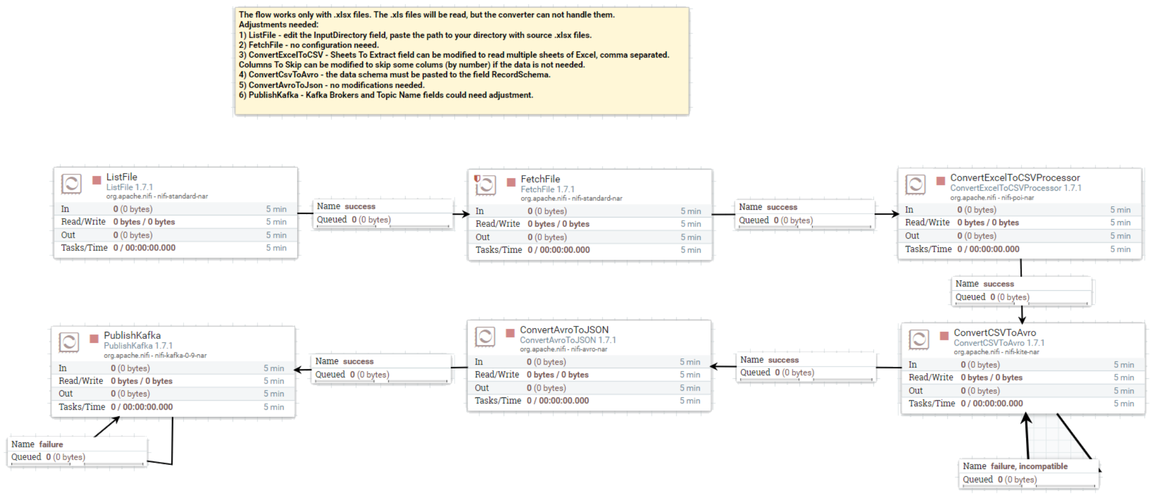
5.1.3.3 Situation Provision

The Full Prototype of Situation Determination module sends the identified situation(s) to the SAFIRE kafka cluster, so that subsequent services can retrieve the situations and use them for their tasks (e.g. Optimisation Engine).

5.2 ELECTROLUX

5.2.1 Data Ingestion

For the data ingestion of the Electrolux case a specific NiFi template with the necessary processors has been developed (Figure 13). The goal of the Electrolux template is to gather information related to cooking processes while using the experimental Electrolux cooker installation. The data are being produced from the installation and stored in .xlsx files. The NiFi template reads the data from the new coming files (ListFile processor) and brings them to a structure ready to convert to the specific data formats (FetchFile processor), NiFi FlowFiles, which are necessary for the next processors (ConvertExcelToCSVProcessor, ConvertCSVToAvro, ConvertAvroToJSON) that promote the data inside the template. The last processor of the template (PublishKafka) receives the data in the FlowFile and publishes them to Kafka in a predefined topic (e.g. elux_data_topic). When published to Kafka, the ingested data are available to all SAFIRE modules to receive for their internal processing.


The flow works only with .xlsx files. The .xls files will be read, but the converter can not handle them.
Adjustments needed:
1) ListFile - edit the InputDirectory field, paste the path to your directory with source .xlsx files.
2) FetchFile - no configuration neeed.
3) ConvertExcelToCSV - Sheets To Extract field can be modified to read multiple sheets of Excel, comma separated. Columns To Skip can be modified to skip some colums (by number) if the data is not needed.
4) ConvertCsvToAvro - the data schema must be pasted to the field RecordSchema.
5) ConvertAvroToJson - no modifications needed.
6) PublishKafka - Kafka Brokers and Topic Name fields could need adjustment.

ListFile
ListFile 1.7.1
org.apache.nifi - nifi-standard-nar
Name success
FetchFile
FetchFile 1.7.1
org.apache.nifi - nifi-standard-nar
Name success
ConvertExcelToCSVProcessor
ConvertExcelToCSVProcessor 1.7.1
org.apache.nifi - nifi-poi-nar
Name success
ConvertCSVToAvro
ConvertCSVToAvro 1.7.1
org.apache.nifi - nifi-kite-nar
Name failure, incompatible
Name success
ConvertAvroToJSON
ConvertAvroToJSON 1.7.1
org.apache.nifi - nifi-avro-nar
Name success
PublishKafka
PublishKafka 1.7.1
org.apache.nifi - nifi-kafka-0-9-nar
Name failure
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Queued 0 (0 bytes)

Figure 13: NiFi Template for the Electrolux Data Ingestion Module

5.2.2 Situation Monitoring

5.2.2.1 Electrolux Monitor

The Electrolux monitor observes the data from cookers. An example for information observed by the Electrolux Monitor is:

- Cooker Status – Information about the status of a cooker. It contains information about the energy, type of pot, amount of water used.

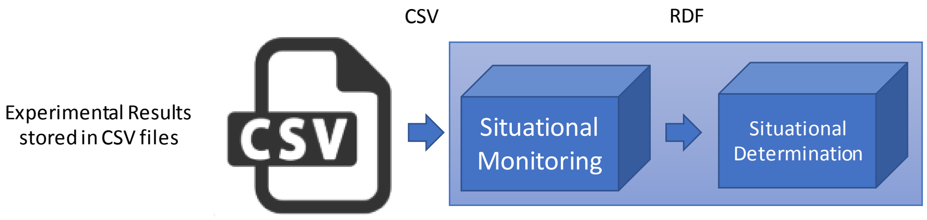

CSV
RDF
Experimental Results stored in CSV files
CSV
Situational Monitoring
Situational Determination

Figure 14: Data transfer to the Electrolux monitor

The Full Prototype of the Electrolux monitor has a permanent loop of watching a folder in a filesystem for new files, which contain experimental results.

<u>Data model</u>

The class *ElecDataModel* holds the main data model for the Electrolux business case. The class *MonitoredCookerInformation* contains the sensor information from the Electrolux cookers.

CookerStatusInformation: Currently, the ID, the cookers name, its status, the pot used and the amount of water is being monitored.

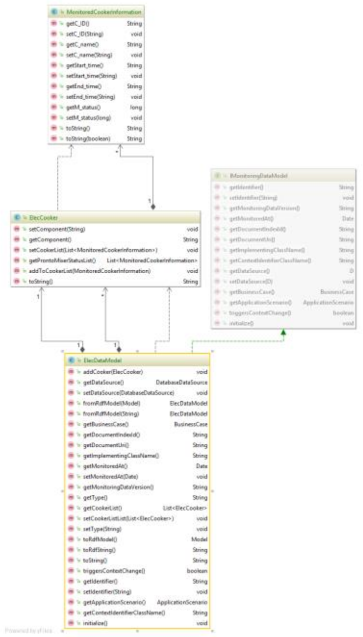

MonitoredCookerInformation
getC_ID() String
setC_ID(String) void
getC_name() String
setC_name(String) void
getStart_time() String
setStart_time(String) void
getEnd_time() String
setEnd_time(String) void
getM_status() long
setM_status(long) void
toString() String
toString(boolean) String
*
1
ElecCooker
setComponent(String) void
getComponent() String
setCookerList(List<MonitoredCookerInformation>) void
getProntoMiserStatusList() List<MonitoredCookerInformation>
addToCookerList(MonitoredCookerInformation) void
toString() String
IMonitoringDataModel
getIdentifier() String
setIdentifier(String) void
getMonitoringDataVersion() String
getMonitoredAt() Date
getDocumentIndexId() String
getDocumentUri() String
getImplementingClassName() String
getContextIdentifierClassName() String
getDataSource() D
setDataSource(D) void
getBusinessCase() BusinessCase
getApplicationScenario() ApplicationScenario
triggersContextChange() boolean
initialize() void
1
*
1
1
ElecDataModel
addCooker(ElecCooker) void
getDataSource() DatabaseDataSource
setDataSource(DatabaseDataSource) void
fromRdfModel(Model) ElecDataModel
fromRdfModel(String) ElecDataModel
getBusinessCase() BusinessCase
getDocumentIndexId() String
getDocumentUri() String
getImplementingClassName() String
getMonitoredAt() Date
setMonitoredAt(Date) void
getMonitoringDataVersion() String
getType() String
getCookerList() List<ElecCooker>
setCookerListList(List<ElecCooker>) void
setType(String) void
toRdfModel() Model
toRdfString() String
toString() String
triggersContextChange() boolean
getIdentifier() String
setIdentifier(String) void
getApplicationScenario() ApplicationScenario
getContextIdentifierClassName() String
initialize() void

Figure 15: Data model for Cooker status

Monitor Implementation

For the monitoring process of the Electrolux BC a *Filesystem Monitor* is being used, as the sensor data from the experimental set-ups is stored in CSV files in a file system. Figure 23 shows the relationship and inheritance between the *ElectroluxCSVAnalyser* and the more generic *FileAnalyser*. The main task of the *ElectroluxCSVAnalyser* is to gather information about the cooker status. The data are being transferred into the data model shown in Figure 23 and thereafter stored in the *Monitoring Repository*.

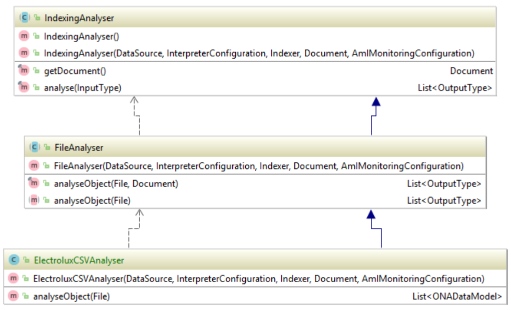

IndexingAnalyser
IndexingAnalyser()
IndexingAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
getDocument() Document
analyse(InputType) List<OutputType>
FileAnalyser
FileAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
analyseObject(File, Document) List<OutputType>
analyseObject(File) List<OutputType>
ElectroluxCSVAnalyser
ElectroluxCSVAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
analyseObject(File) List<ONADataModel>

Figure 16: Inheritance structure of the Analyser used for Electrolux

5.2.3 Situation Determination

5.2.3.1 Electrolux Situation Identification

The Electrolux specific implementation of the ContextIdentifier executes SPARQL queries to identify data from the monitored Situation Monitoring service. Code 5 below shows an example of a monitored data item, that is used to identify Situations.

```
...
  <rdf:Description rdf:about="#ElectroluxMonitoredInformation/705471857">
    <elux:cur_f06 rdf:datatype="#double">0.0</elux:cur_f06>
    <elux:time rdf:datatype="#double">0.17587581125236543</elux:time>
    <elux:cur_f05 rdf:datatype="#double">0.0</elux:cur_f05>
    <elux:cur_f11 rdf:datatype="#double">0.0</elux:cur_f11>
    <elux:t_coil rdf:datatype="#double">0.0</elux:t_coil>
    <elux:cur_f10 rdf:datatype="#double">0.0</elux:cur_f10>
    <elux:cur_f08 rdf:datatype="#double">0.0</elux:cur_f08>
    <elux:cur_f02 rdf:datatype="#double">0.0</elux:cur_f02>
    <elux:cur_f07 rdf:datatype="#double">0.0</elux:cur_f07>
    <elux:cur_f01 rdf:datatype="#double">0.0</elux:cur_f01>
    <elux:t_water rdf:datatype="#double">100.16326904296875</elux:t_water>
    <elux:cur_f13 rdf:datatype="#double">0.0</elux:cur_f13>
    <elux:cur_f12 rdf:datatype="#double">0.0</elux:cur_f12>
    <elux:cur_f04 rdf:datatype="#double">0.0</elux:cur_f04>
    <elux:cur_f09 rdf:datatype="#double">0.0</elux:cur_f09>
    <elux:cur_f03 rdf:datatype="#double">0.0</elux:cur_f03>
    <elux:energy rdf:datatype="#double">1006.7838700061303</elux:energy>
    <rdf:type rdf:resource="#ElectroluxMonitoredInformation"/>
  </rdf:Description>
  <rdf:Description rdf:about="#ElectroluxMonitoredInformation/847308988">
    <elux:cur_f05 rdf:datatype="#double">0.0</elux:cur_f05>
    <elux:time rdf:datatype="#double">0.19321985970376387</elux:time>
    <elux:cur_f10 rdf:datatype="#double">0.0</elux:cur_f10>
    <elux:cur_f09 rdf:datatype="#double">0.0</elux:cur_f09>
    <elux:cur_f13 rdf:datatype="#double">0.0</elux:cur_f13>
    <elux:cur_f06 rdf:datatype="#double">0.0</elux:cur_f06>
    <elux:t_coil rdf:datatype="#double">0.0</elux:t_coil>
    <elux:cur_f12 rdf:datatype="#double">0.0</elux:cur_f12>
    <elux:energy rdf:datatype="#double">684.3312550976387</elux:energy>
    <elux:cur_f03 rdf:datatype="#double">0.0</elux:cur_f03>
    <elux:cur_f01 rdf:datatype="#double">0.0</elux:cur_f01>
    <rdf:type rdf:resource="#ElectroluxMonitoredInformation"/>
    <elux:cur_f07 rdf:datatype="#double">0.0</elux:cur_f07>
    <elux:cur_f11 rdf:datatype="#double">0.0</elux:cur_f11>
    <elux:cur_f04 rdf:datatype="#double">0.0</elux:cur_f04>
    <elux:cur_f02 rdf:datatype="#double">0.0</elux:cur_f02>
    <elux:t_water rdf:datatype="#double">71.4569091796875</elux:t_water>
    <elux:cur_f08 rdf:datatype="#double">0.0</elux:cur_f08>
  </rdf:Description>
  <rdf:Description rdf:about="#ElectroluxMonitoredInformation/1456717480">
    <elux:cur_f03 rdf:datatype="#double">0.0</elux:cur_f03>
    <elux:cur_f01 rdf:datatype="#double">0.0</elux:cur_f01>
    <elux:t_water rdf:datatype="#double">32.39593505859375</elux:t_water>
    <elux:time rdf:datatype="#double">0.03197233499693058</elux:time>
    <elux:energy rdf:datatype="#double">113.13161745422406</elux:energy>
    <rdf:type rdf:resource="#ElectroluxMonitoredInformation"/>
    <elux:cur_f06 rdf:datatype="#double">0.0</elux:cur_f06>
    <elux:cur_f13 rdf:datatype="#double">0.0</elux:cur_f13>
    <elux:cur_f09 rdf:datatype="#double">0.0</elux:cur_f09>
    <elux:cur_f10 rdf:datatype="#double">0.0</elux:cur_f10>
    <elux:cur_f05 rdf:datatype="#double">0.0</elux:cur_f05>
    <elux:t_coil rdf:datatype="#double">0.0</elux:t_coil>
    <elux:cur_f12 rdf:datatype="#double">0.0</elux:cur_f12>
    <elux:cur_f08 rdf:datatype="#double">0.0</elux:cur_f08>
    <elux:cur_f04 rdf:datatype="#double">0.0</elux:cur_f04>
    <elux:cur_f02 rdf:datatype="#double">0.0</elux:cur_f02>
    <elux:cur_f11 rdf:datatype="#double">0.0</elux:cur_f11>
    <elux:cur_f07 rdf:datatype="#double">0.0</elux:cur_f07>
  </rdf:Description>
...
```

Code 7 – Example of Monitoring Data in RDF representation (excerpt)

The Electrolux specific implementation executes queries, such as the following:

```
Select ?cookerInfo?time ?tempWater ?tempCoil
where
{
 ?mixer rdf:type elux:Cooker.
 ?mixer elux: ElectroluxMonitoredInformation ?cookerInfo.
 ?mixerInfo elux:time ?time.
 ?mixerInfo elux:t_water?tempWater.
 ?mixerInfo elux:t_coil?tempCoil.
};
```

Situation Identifier Implementation

Figure 17 shows the inheritance structure of the Electrolux specific situation identifier. The data previously observed by the Situation Monitoring is used to identify the situations based on the monitored data.

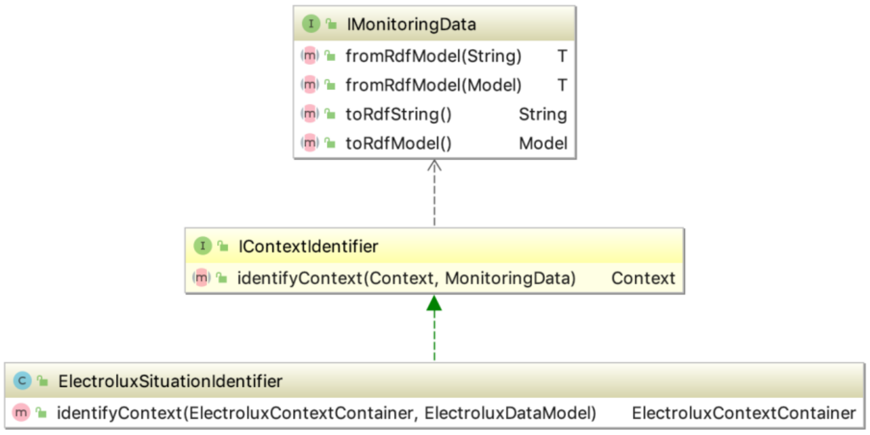

IMonitoringData
fromRdfModel(String) T
fromRdfModel(Model) T
toRdfString() String
toRdfModel() Model
IContextIdentifier
identifyContext(Context, MonitoringData) Context
ElectroluxSituationIdentifier
identifyContext(ElectroluxContextContainer, ElectroluxDataModel) ElectroluxContextContainer

Figure 17: Inheritance structure of the Electrolux Situation Identifier

5.2.3.2 *Rule based reasoning*

The rule-based Situation Reasoning works similar to the OAS case, see Section 5.1.3.2.

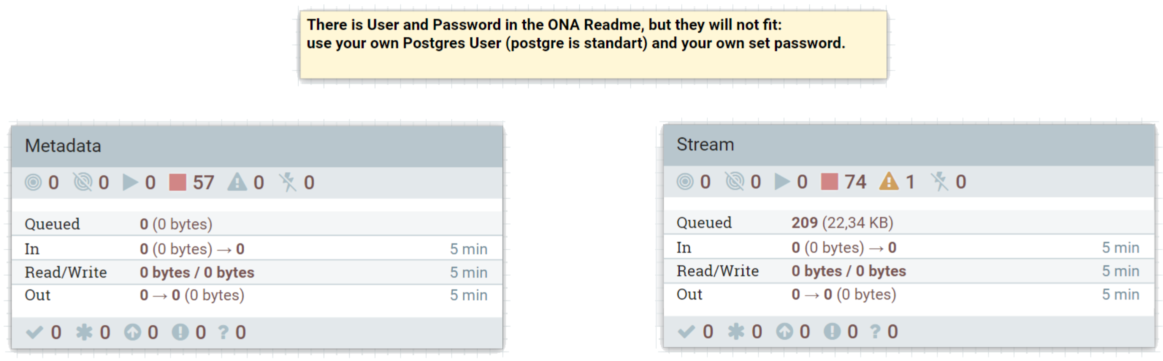
5.2.3.3 Situation Provision

The Situation Provision is working similar to the OAS case, see Section 5.1.3.3.

5.3 ONA

5.3.1 Data Ingestion

The data ingestion module for the ONA case is composed by two different NiFi template groups (figure Figure 18), namely the Metadata (figure Figure 19) and the Stream (figure Figure 20) groups. The goal for the Metadata template is to connect to the ONA cloud API and retrieve the appropriate configuration data that the Stream template will need to know in order to get the data from the ONA machines. The Metadata template populates a PostgreSQL database with information regarding the expected data groups (table "groups"), the required data ids (tables "indicator" and "selectedIndicator"), the available machines for the SAFIRE user (table "machine") and their locations (table "location"), and a lookup table that connects machines and data groups (table "stream"). Using these database records, the Stream template is being configured so to retrieve and publish to Kafka only the data necessary for the selected ONA scenario.



There is User and Password in the ONA Readme, but they will not fit:
use your own Postgres User (postgre is standart) and your own set password.

Metadata

0 0 0 57 0 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

✓ 0 ✱ 0 0 0 ? 0

Stream

0 0 0 74 1 0

Queued	209 (22,34 KB)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

✓ 0 ✱ 0 0 0 ? 0

Figure 18: NiFi Template Groups for the ONA Data Ingestion Module

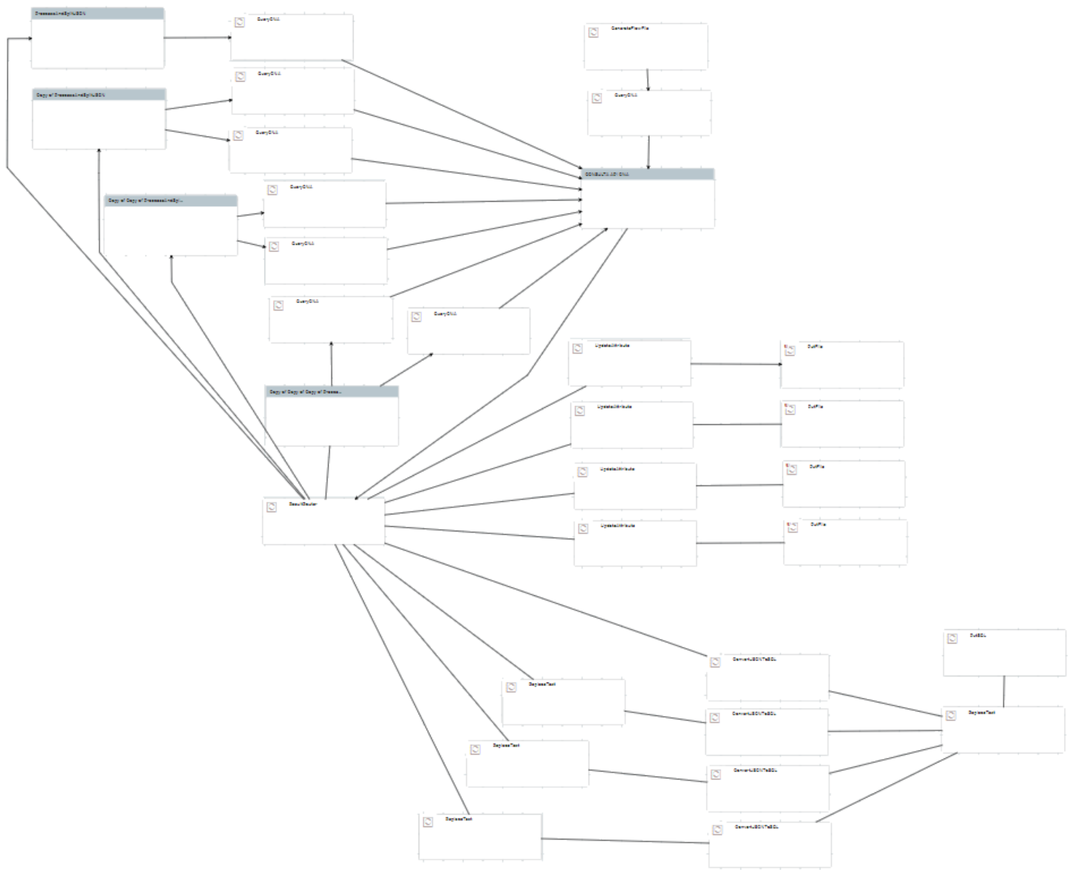


Figure 19: Metadata Group of the ONA NiFi Template

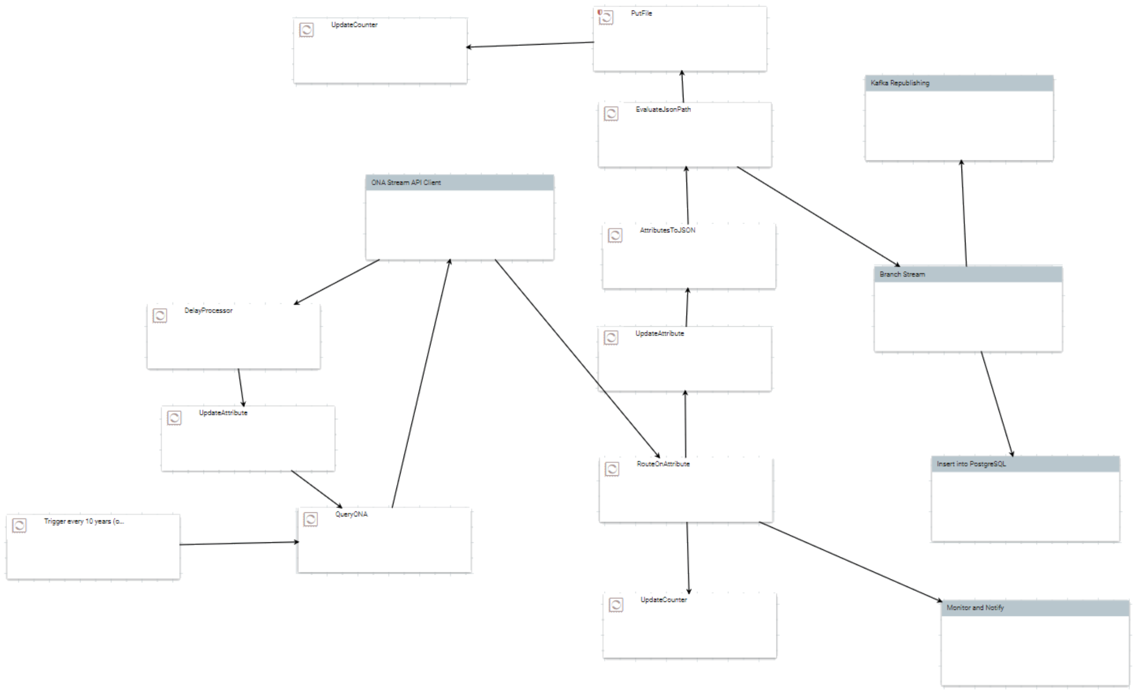


Figure 20: Stream Group of the ONA NiFi Template

5.3.2 Situation Monitoring

5.3.2.1 ONA Monitor

The ONA monitor observes the data in the ONA cloud platform. The ONA cloud platform acts as a solution to allow for example machine monitoring, data analysis and planning of predictive maintenance. Examples are:

- Machine Execution Status – Information about the availability of a machine and in case it is available, it provides information whether the machine is currently in production mode or not.

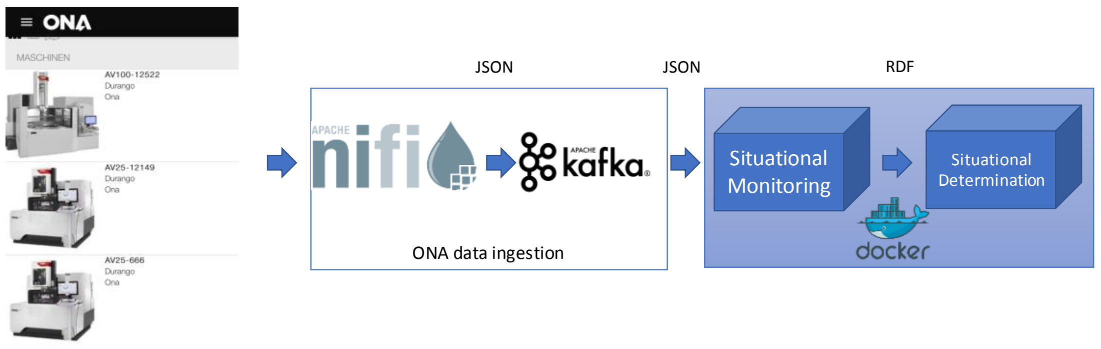


Figure 21: Data transfer from the ONA cloud to the ONA monitor

The Full Prototype of the ONA monitor has a permanent loop of watching the current machine execution status. The following message snippets show some examples for data monitored by the ONA monitor.

Current machine execution status

Kafka topic: `MachineExecutionStatus`

Data in kafka:

```
[
MachineExecutionStatusInformation{ID='1', m_name='AV100-12522', m_status='1',
e_status=’0’},
MachineExecutionStatusInformation {ID='2', m_name='AV25-12149', m_status='1',
e_status=’1’},
MachineExecutionStatusInformation {ID='3', m_name='AV25-666', m_status='1',
e_status=’0’},
]
```

Code 8: kafka data for machine execution status

Data model

Currently, two data classes are used, which hold the information about the monitored data from the ONA cloud platform matching the machine execution status. Figure 22 shows the relationship between the different data classes: The class *ONACloudDataModel* holds the main ONA cloud data model, which is being specified in detail within the *ONAMachine* class, which on its part holds a relation to the data class *MachineExecutionStatusInformation*, which in turn contains the sensor information from ONA machines.

MachineExecutionStatusInformation: Currently, the ID, the machines’s name and its execution status is being monitored.

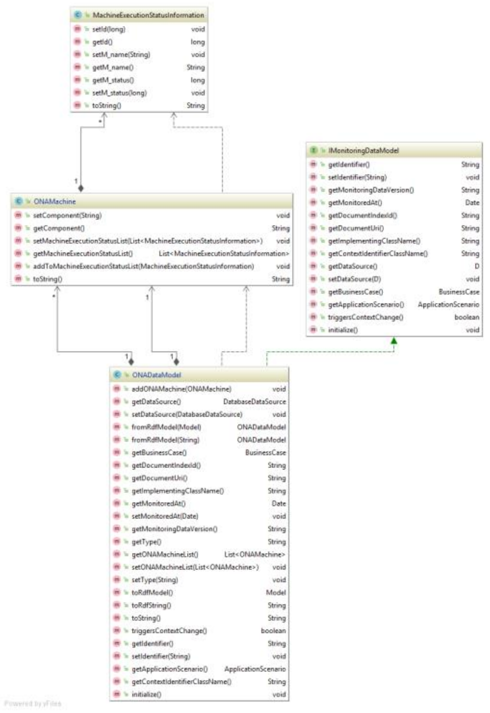

MachineExecutionStatusInformation
setId(long) void
getId() long
setM_name(String) void
getM_name() String
getM_status() long
setM_status(long) void
toString() String
*
1
IMonitoringDataModel
getIdentifier() String
setIdentifier(String) void
getMonitoringDataVersion() String
getMonitoredAt() Date
getDocumentIndexId() String
getDocumentUri() String
getImplementingClassName() String
getContextIdentifierClassName() String
getDataSource() D
setDataSource(D) void
getBusinessCase() BusinessCase
getApplicationScenario() ApplicationScenario
triggersContextChange() boolean
initialize() void
ONAMachine
setComponent(String) void
getComponent() String
setMachineExecutionStatusList(List<MachineExecutionStatusInformation>) void
getMachineExecutionStatusList() List<MachineExecutionStatusInformation>
addToMachineExecutionStatusList(MachineExecutionStatusInformation) void
toString() String
*
1
1
1
ONADataModel
addONAMachine(ONAMachine) void
getDataSource() DatabaseDataSource
setDataSource(DatabaseDataSource) void
fromRdfModel(Model) ONADataModel
fromRdfModel(String) ONADataModel
getBusinessCase() BusinessCase
getDocumentIndexId() String
getDocumentUri() String
getImplementingClassName() String
getMonitoredAt() Date
setMonitoredAt(Date) void
getMonitoringDataVersion() String
getType() String
getONAMachineList() List<ONAMachine>
setONAMachineList(List<ONAMachine>) void
setType(String) void
toRdfModel() Model
toRdfString() String
toString() String
triggersContextChange() boolean
getIdentifier() String
setIdentifier(String) void
getApplicationScenario() ApplicationScenario
getContextIdentifierClassName() String
initialize() void

Figure 22: Data model for machine execution status

Monitor Implementation

For the monitoring process of the ONA BC a *Webservice Monitor* is being used, as the sensor data from the ONA cloud platform is being accessible via a web service-based API. Figure 23 shows the relationship and inheritance between the *ONACloudAnalyser* and the more generic *WebServiceAnalyser*. The main task of the *ONACloudAnalyser* is to gather information about the mixer status and orders from the database. According to the architecture the data will be fetched from a *Kafka Cluster*. Therefore, *Kafka Consumers* for the different data topics are being instantiated, who are continuously polling data from the cluster. The data are being transferred into the data model shown in Figure 23 and thereafter stored in the *Monitoring Repository*.

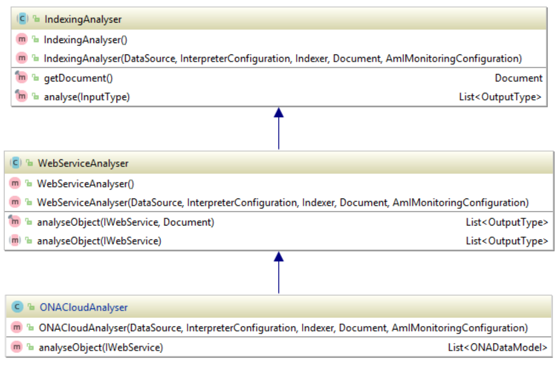

IndexingAnalyser
IndexingAnalyser()
IndexingAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
getDocument() Document
analyse(InputType) List<OutputType>
WebServiceAnalyser
WebServiceAnalyser()
WebServiceAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
analyseObject(IWebService, Document) List<OutputType>
analyseObject(IWebService) List<OutputType>
ONACloudAnalyser
ONACloudAnalyser(DataSource, InterpreterConfiguration, Indexer, Document, AmlMonitoringConfiguration)
analyseObject(IWebService) List<ONADataModel>

Figure 23: Inheritance structure of the Analyser used for ONA cloud platform

Detailed Data Flow

Figure 24 shows a detailed overview of the dataflow between the legacy system, *NiFi*, *Kafka* and the situation awareness modules. All data processing is done via the Kafka cluster. As seen in the figure, the order information and mixer status are being processed by *NiFi* and have their corresponding topic within the *Kafka* node, which will be monitored subsequently by the *Situation Awareness* module.

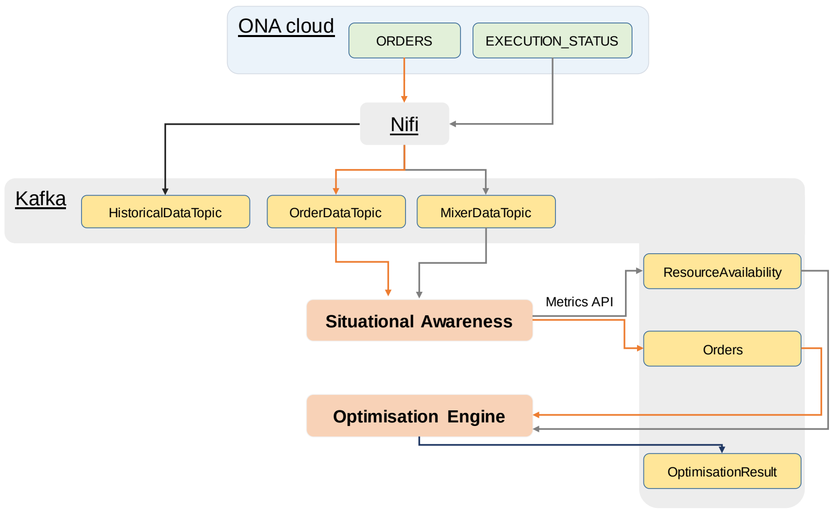


Figure 24: Detailed Data Flow for the ONA scenario

5.3.3 Situation Determination

5.3.3.1 ONA Situation Identification

The ONA specific implementation of the ContextIdentifier executes SPARQL queries to identify data from the monitored Situation Monitoring service. Code 5 below shows an example of a monitored data item, that is used to identify Situations.

```
...
  <rdf:Description rdf:nodeID="A0">
    <rdf:_1 rdf:resource="http://atb-bremen.de/bc-ona/OnaMachine/445895563"/>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://atb-bremen.de/bc-ona/OnaMachine">
    <elux:javaclass>de.atb.context.monitoring.models.ona.OnaMachine</elux:javaclass>
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://atb-bremen.de/bc-ona/OnaMachine/445895563">
    <ona:machineExecutionStatusList rdf:nodeID="A1"/>
    <ona:component rdf:datatype="#string"></ona:component>
    <rdf:type rdf:resource="http://atb-bremen.de/bc-ona/OnaMachine"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://atb-bremen.de/bc-ona/OnaDataModel">

<elux:javaclass>de.atb.context.monitoring.models.ona.OnaDataModel</elux:javaclass>
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://thewebsemantic.com/javaclass">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#AnnotationProperty"/>
  </rdf:Description>
...
```

Code 9 – Example of Monitoring Data in RDF representation (excerpt)

The ONA specific implementation executes queries, such as the following:

```
Select ?mixer ?mixerId ?mixerName ?mixerStatus
where
{
 ?mixer rdf:type oas:Mixer.
 ?mixer oas:MonitoredMixerStatusInformation ?mixerInfo.
 ?mixerInfo oas:m_id ?mixerId.
 ?mixerInfo oas:m_name ?mixerName.
 ?mixerInfo oas:m_status ?mixerStatus.
};
```

Situation Identifier Implementation

Figure 25 shows the inheritance structure of the ONA specific situation identifier. The data previously observed by the Situation Monitoring is used to identify the situations based on the monitored data.

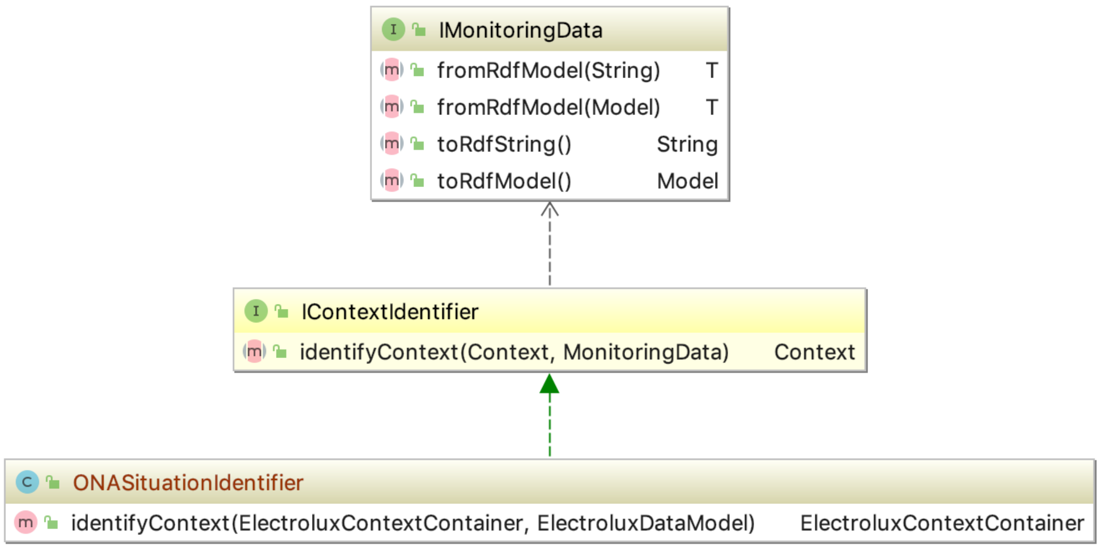

IMonitoringData
fromRdfModel(String) T
fromRdfModel(Model) T
toRdfString() String
toRdfModel() Model
IContextIdentifier
identifyContext(Context, MonitoringData) Context
ONASituationIdentifier
identifyContext(ElectroluxContextContainer, ElectroluxDataModel) ElectroluxContextContainer

Figure 25: Inheritance structure of the ONA Situation Identifier

5.3.3.2 Rule based reasoning

The rule-based Situation Reasoning works similar to the OAS case, see Section 5.1.3.2.

5.3.3.3 Situation Provision

The Situation Provision is working similar to the OAS case, see Section 5.1.3.3.

6. SOFTWARE TOOLS USED FOR IMPLEMENTATION

For the implementation of the Full Prototype several different development tools and IDE[3] have been used. For the overall development and orchestration of all system modules and components the Eclipse IDE has been used. The tested and widely accepted Open Source development environment for Java offers through a modular system a large plug-in community. Through all these techniques selected for the implementation of the systems architecture and services can be summed up in one environment.

The software tools used, together with their version, link and name of the task they are being used for, are listed in the following Table 2. These have been used to develop and run the SAFIRE tools and services against the systems concept and hereby specified functionality. It resembles state-of-the-art tools and software to provide a modular, extendable and expandable service-oriented approach.

Table 2: Overview of used key software tools Table

Functionality	Software	Version	Link
IDE	Eclipse IntelliJ IDEA	>= 4.4 >=2018.1.4	http://www.eclipse.org https://www.jetbrains.com/idea
Build-Management tool	Maven	>=3.5.3	https://maven.apache.org
Version Control	GITlab SVN	>= 2.3	
Issue Management	Jira	>= 6.3	
Programming Language	Java	>= 1.8.0_xx	http://www.java.com
XML Configuration Wrapper	Simple XML	>= 2.7	http://simple.sourceforge.net/
Web Application Framework	Spring	>= 4.1	http://www.springsource.org/
Runtime Environment / Application Server	Apache Tomcat	>= 8.0	http://tomcat.apache.org/
JPA-based persistence	Hibernate	>= 4.3	http://hibernate.org/
Database	H2 Database	1.3	http://www.h2database.com
RDF / OWL API	Jena	>= 2.12	http://jena.apache.org
RDF Storage	SDB / TDB	>= 1.3 / 1.1	http://jena.apache.org
	Joseki	>= 3.4	http://jena.apache.org
Indexing	Apache Lucene	>= 5.0	http://lucene.apache.org
Protege		>=5.2.0	https://protege.stanford.edu
Data processing and distribution	Apache NiFi Apache Kafka	>=1.6.0 >=1.1.0	https://nifi.apache.org https://kafka.apache.org
Container virtualization	Docker		https://www.docker.com

7. CONCLUSIONS

This document presented the work done by SAFIRE in WP4, in particular in T4.3: Early and Full Prototype of Modelling Correlation between Information Sources,

[3] Integrated Development Environment

Products and Situations, specifically it documents the work on Full Prototype implementation.

Following the requirements and specification for SAFIRE Full Prototype defined in accordance with SAFIRE Concept and Business Case requirements and analysis and the following requirements definition, as well as the data model, external interfaces and functional and technical specifications, the Full Prototype was developed. This document serves as brief description of this Full Prototype implementation given that the result of this task is actually the developed Software.

8. APPENDIX

8.1 BUSINESS CASE SPECIFIC SITUATION MODELS

8.1.1.1 Electrolux

The Situation Model for the Electrolux extends the Generic SAFIRE Situation Model in the following concepts under the concept “Information”:

Entity	Description
AirFlow	It describes the presence or absence of air flowing in the hob coil.
Alarm	A possible alarm raised during the cooking process.
BoilingDetectionTime	The predicted from PA water boiling point.
Event	Information that describes the execution of some happening which could raise an alarm.
HobTemperature	The temperature of the hob in the cooker.
PotType	The id of the type of pot currently in use.
PowerProfile	The id of the energy schema used in the cooking process.
ShakerUsed	An indicator that shows whether the user mixes the pot ingredients during the cooking process.
Timestamp	The time of the current data measurement.
WaterAmount	The amount of water (or food) currently contained in the pot.
WaterTemperature	Current temperature of the water (or food) in the pot.

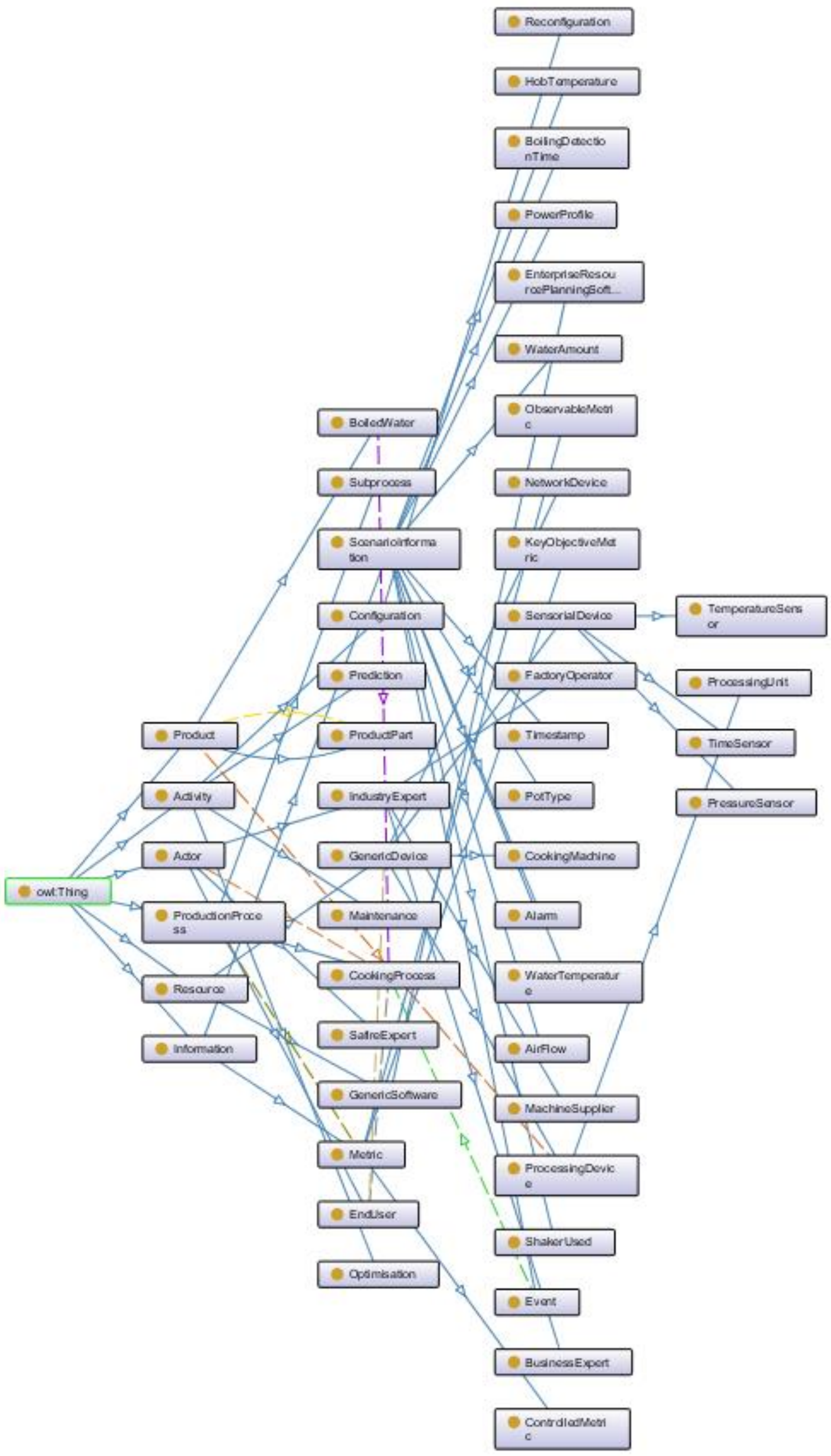


Figure 26: ELECTROLUX specific SAFIRE Situation Model (excerpt)

8.1.1.2 OAS

The Situation Model for OAS extends the Generic SAFIRE Situation Model in the following concepts:

Entity	Description
Alarm	A possible alarm raised during a monitored process.
Event	Information that describes the execution of some happening which could raise an alarm.
Alarm Priority	Defines the priority of an alarm (e.g. INFO, WARN, ERROR).
Batch	Identifies a production batch.
Product	Identifies a product, that can be produce on a production line.
Production Line Name	Name of a production line of the factory.
Product Name	Name of the product that can be produced on a production line.
Recipe	Recipe of the product that can be produced on a production line.
Source Silo	Identifies a silo from which a source material will be taken.
Target Silo	Identifies a silo to which a finished product will be pumped.
Order	Identifies a production order coming from the ERP system.
Production Schedule	
Paint	Identifies the type of paint to be produced
Silo	Identifies a silo. A silo can be used as source for materials or as destination for finished products.
Mixer	Identifies a mixer, that is used for mixing the paint.
Pipeline	Identifies a pipeline. A pipeline can be used for source materials or finished products.
Scale	Identifies the scales, that are used within the production process.
Conveyor	Identifies a conveyor, that is used in the production process to transport dry raw materials.
Pump	Identifies a pump within the production process.
Valve	Identifies a valve within the production process.
Pressure Sensor	Identifies a pressure sensor within the production process (e.g. pressure in the pipelines).
Speed Sensor	Identifies a speed sensor within the production process (e.g. turn speed of a mixer).
Temperature Sensor	Identifies a temperature sensor within the production process (e.g. temperature of paint during mixing process).
Time Sensor	Identifies a time sensor within the production process (e.g. execution time of a mixing process).
ERP	The Enterprise Resource Planning system used in the factory – SAP in this use case.

MES (proNTo)	The MES used in the factory – proNTo in this use case.

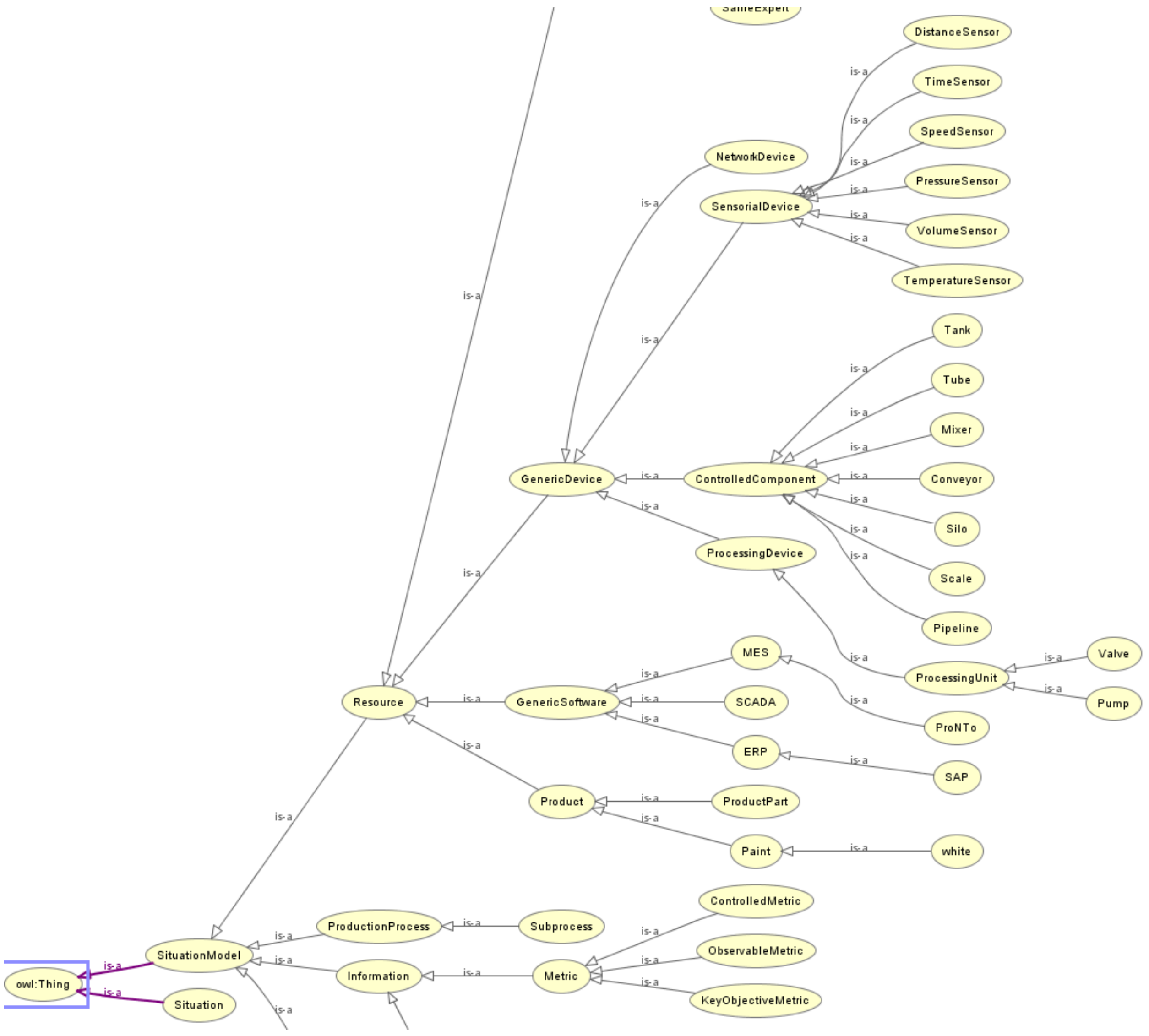


Figure 27: OAS specific SAFIRE Situation Model (excerpt)

8.1.1.3 ONA

The Situation Model for ONA extends the Generic SAFIRE Situation Model in the following concepts:

Entity	Description
AccuracyDeviation	Difference from the real value of precision in the cutting edge.
ActiveMachine	The id of the current monitored machine.
Alarm	A possible alarm raised during the EDM process.
Availability	It describes the status of availability of the monitored machine.
DueTime	Deadline for the completion of a specific process.
Event	Information that describes the execution of some happening which could raise an alarm.
GeneratorPositionX	The x coordination of the cutting edge.
GeneratorPositionY	The y coordination of the cutting edge.
RecastLayerThickness	Thickness of the part which will need to be cut.
TotalEnergy	The total amount of energy currently used.
WireCost	Given current cost for the order of new wire of the current used type.
WireDiameter	The diameter of the wired used in the cutting edge.
WireType	The type of wired used in the cutting edge.

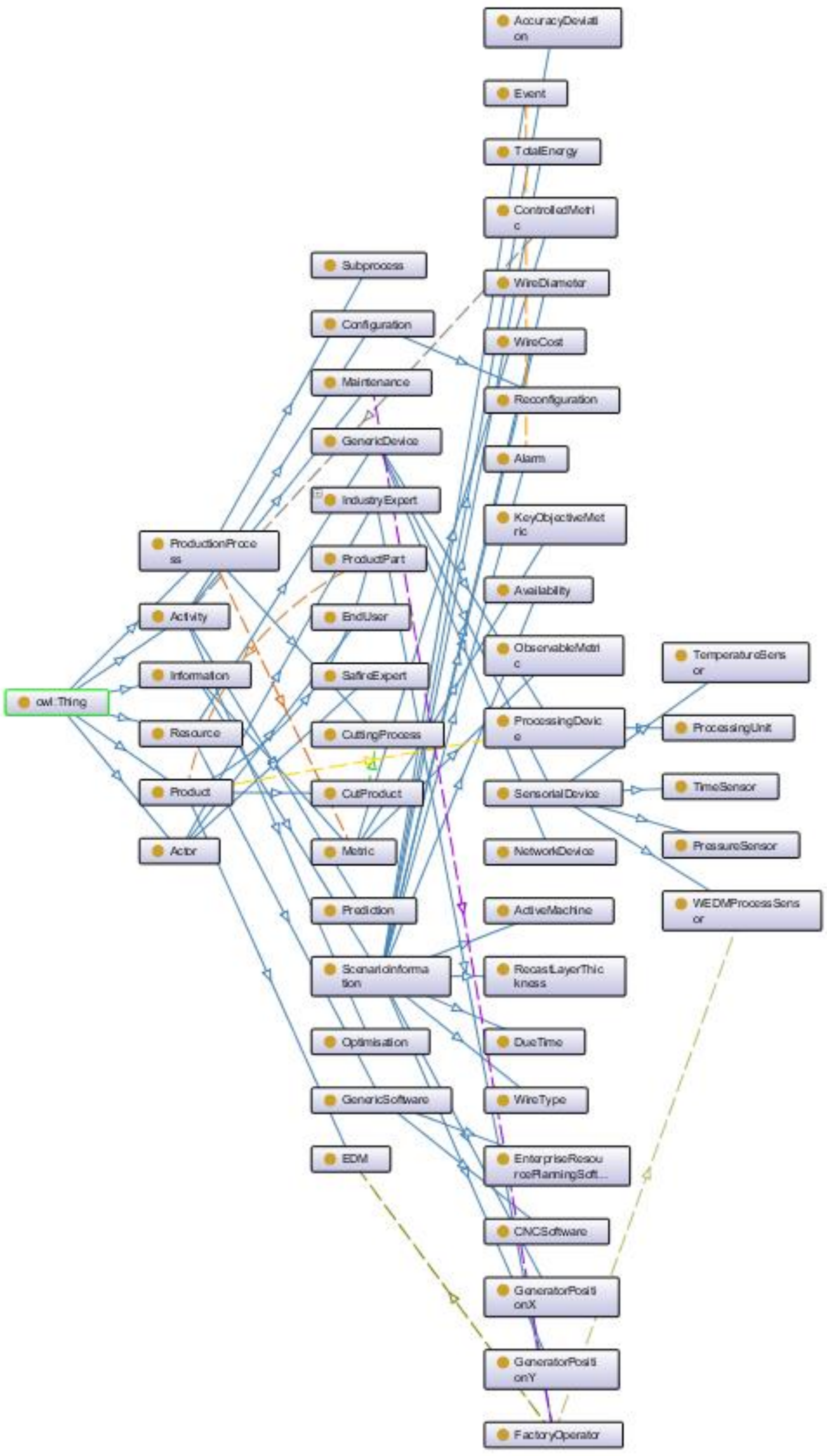


Figure 28: ONA specific SAFIRE Situation Model (excerpt)