Production harmonizEd Reconfiguration of Flexible Robots and Machinery

Horizon 2020 – Factories of the Future, Project ID: 680435

Deliverable 3.2

Real-time Process Information Exploitation

Lead Author: UNINOVA

Version: 1.0
Date: 26.10.2016
Status: Final
Dissemination level: PUBLIC

Version	Date	Content
0.1	20.06.2016	Table of Contents
0.2	04.10.2016	Draft document including the contributions of the different partners involved in the task.
0.3	17.10.2016	Conclusion of incomplete sections (Abstract, Sections 1.2, 3.4, 6), addition of Section 5 – Tests and Validation, and refining based on the feedback received from the various involved partners. Revision of Acronyms and References.
1.0	26.10.2016	Final proof reading.

Author List:

André Rocha (UNINOVA)
Danielle Sandler (UNINOVA)
Jonas Queiroz (IPB)
José Barbosa (IPB)
Ricardo Peres (UNINOVA)

Abstract

The recent advances in Information and Communications Technology (ICT) brought forward several new concepts such as Cloud Computing, the Industrial Internet of Things, Big Data and Cyber-Physical Systems, allowing for more agile, flexible, highly-scalable, distributed and thus far more complex systems and solutions to be developed, causing a profound change in various applications domains.

In modern manufacturing, higher and higher volumes of data are thus being constantly generated by the manufacturing processes and systems adopting these new paradigms and technologies. However, only a small percentage is actually used in a meaningful way, originating the data "dump" phenomenon due to data volumes and rates becoming unmanageable.

Aligned with PERFoRM's Industry 4.0 vision, this documents details a modular framework for the implementation of a highly flexible, pluggable and distributed data acquisition and analysis system supported by the results of previous successful European projects, which can be used for both assisting in run-time decision making and triggering self-adjustment methods, allowing corrections to be made before failures actually occur, therefore reducing the impact of such events in production. Additionally, a possible implementation is thoroughly described, along with a few preliminary results from tests conducted under simulated conditions.

Table of Contents

List of Figures

List of Tables

I. Acronyms

Abbreviation	Explanation
AB	Alarm Bolt
CMA	Component Monitoring Agent
CSS	Component Simulation Spout
CPS	Cyber-Physical System
DAL	Data Acquisition Layer
DCL	Data Collection Library
DOI	Data Output Library
DPL	Data Processing Layer
DQL	Data Queue Layer
DVL	Data Visualization Layer
DAG	Directed Acyclic Graph
EDL	Event Description Library
FCB	ForeCast Bolt
FIPA	Foundation for Intelligent Physical Agents
HLCMA	Higher-Level Component Monitoring Agent
HMI	Human-Machine Interface
ICT	Information and Communication Technologies
IT	Information Technologies
JADE	Java Agent DEvelopment Framework
KPI	Key Performance Indicator
MDD	Monitoring Data Description
MAS	MultiAgent System
OCA	Output Coordinator Agent
PRIME	Plug and pRoduce Intelligent Multi-agent Environment
PERFoRM	Production harmonizEd Reconfiguration of Flexible Robots and Machinery
RAB	Rolling Average Bolt
SMA	Simple Moving Average
WP	Work Package

1. Introduction

1.1. Objective of the Document

This deliverable contains the outcome of Task 3.2, entitled "Real-time Process Information", which encompasses the development of the means to perform real-time data acquisition and information extraction, the generation of predictive data to assist in run-time decision making, as well as the detection and computation of trends, correlations and forecasts to predict future production parameters and trigger self-adjustment and correction methods.

For this purpose the principles advocated by the Industry 4.0 movement were used as guidelines during the development of the task, ensuring the solution is capable of being integrated into a smart production environment, supporting changeable conditions at the shop-floor level including the plugging and unplugging of components during run-time, possible reconfigurations and unexpected disturbances. Additionally, this task considers the requirements imposed in WP1, WP2.2, WP2.4, WP7.1, WP8.1, WP9.1 and WP10.1 [1-7].

In line with the overall PERFoRM vision, results from previous successful R&D projects in the field manufacturing data acquisition and processing, as well as ambience intelligence were also taken into account, more concretely FP7 PRIME, IN-LIFE and the Self-Learning projects were used as a basis for the work documented hereafter.

1.2. Structure of the Document

The document is divided into six main sections. Excluding the introduction and beginning with Section 2, the overall approach is presented as a multidisciplinary solution, encompassing data acquisition, data pre-processing or preparation, data processing and visualization in its layered architecture. Afterwards, Section 3 describes each of these layers in further detail, more specifically in terms of purpose, the associated requirements, core functionalities and interactions of each one. Section 4 showcases a possible implementation based on the proposed framework, describing the applicability and technical aspects regarding each of the layers encompassed in Section 2. Furthermore, Section 5 presents some tests conducted in a simulated environment, illustrating the implementation's behaviour in normal operation and under two different simulated failure settings. Finally, Section 6 summarizes some conclusions regarding the developments regarded in the document.

2. Real-time Data Acquisition, Processing and Visualization Approach

Being a critical part of the PERFoRM ecosystem, the proposed solution is responsible for not only performing the context-aware data analysis, thus generating predictive data that can be used to trigger the system's self-adjustment mechanisms (e.g. reconfiguration), but also for the acquisition of the data itself at both the manufacturing cell and component levels.

Additionally, a given number of requirements are imposed on the architecture's design. First and foremost, in line with PERFoRM's vision the architecture should be generic enough to be applicable to various different scenarios, being open so as to not depend on the existence of a single communication protocol or standard on the shop floor, thus facilitating its industrial integration and adoption. Moreover, it needs to be capable of adapting to changes to the process or its components in run-time, for instance in terms of both pluggability and changes to the Key Performance Indicators (KPI) to be analysed. Furthermore, data and context representation should follow PERFoRM's common data model in order to enable the seamless interoperability and data exchange between the data analysis architecture and the remaining PERFoRM system elements and tools.

Another point to take into account is the aspect of scalability. In order to ensure that the approach is applicable to a varied number of different use cases, it needs to be capable of scaling according to each use case requirements. However, as a system scales its complexity tends to increase to higher levels as a consequence. Thus, in order to tackle this challenge, a layered architectural structure is proposed. An overview of this approach can be seen in Figure 1.

Figure 1 - Task 3.2 Layered Approach Overview

As depicted, the proposed architecture is divided into several layers in order to decrease the overall complexity, each operating according to a specific purpose on top of the shop floor, which stands as the base layer. Subsection 2.1 will approach the specification of the aforementioned proposed architecture.

2.1. Architecture Specification

As previously mentioned, the architecture consists in a series of stacked layers, dividing the different tasks that are encompassed in the processes of data acquisition and analysis among them in order to decrease overall complexity and achieve a higher degree of scalability and adaptability. This is made possible also due to the modular and generic nature of each of the architectural elements, which can operate independently from the technology used in the remaining intervenient, so long as the communications specifications imposed by the generic interfaces that connect them are respected. An overview is provided in Figure 2. In which the grey layers in between each element represent these generic communication interfaces.

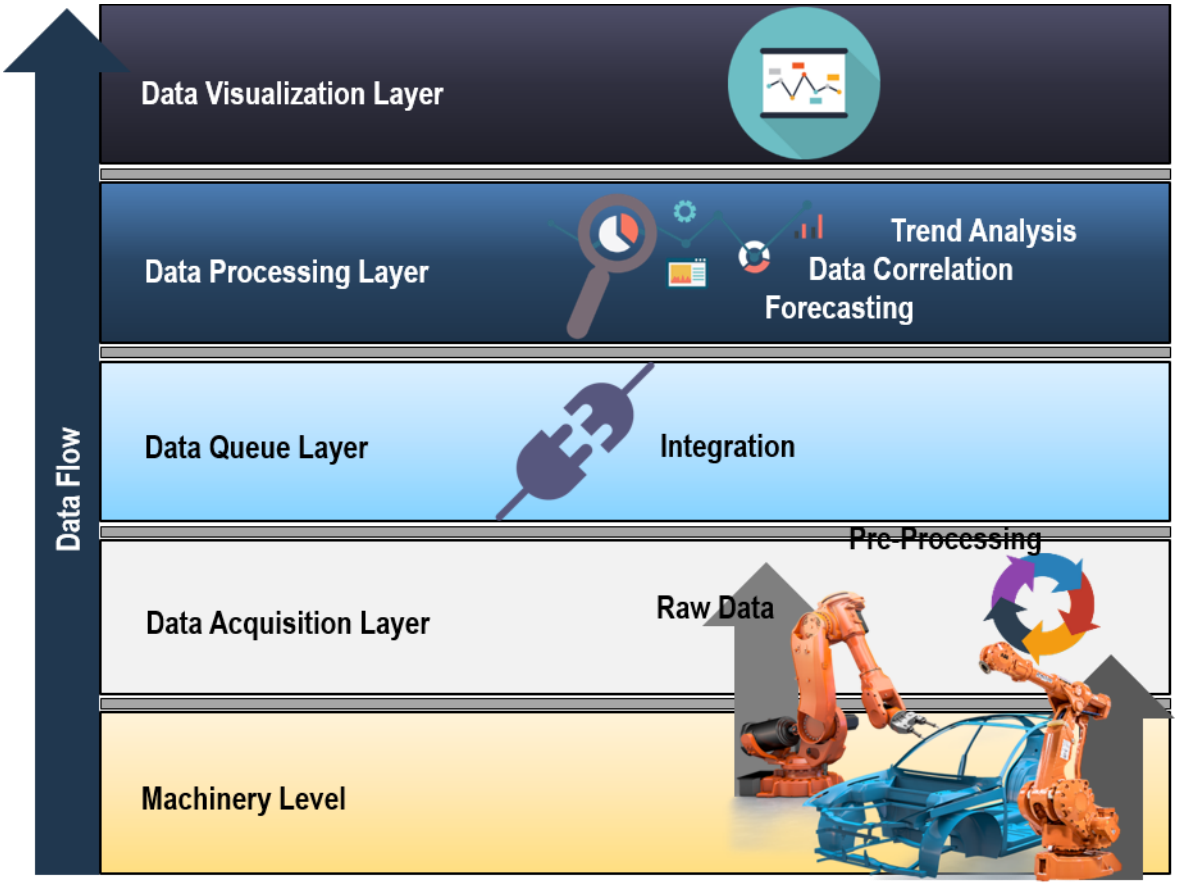

Figure 2 - Architecture Stack Overview

As it can be seen in Figure 2, data flows solely from the bottom to the top layer, meaning that the data flow consists in raw data being collected and pre-processed by the Data Acquisition Layer (DAL), being buffered in the Data Queue Layer (DQL), which in turn prepares it to be consumed by the Data Processing Layer (DPL), which can compute trends, forecasts and correlations to be observed in the Data Visualization Layer (DVL), as well as triggering appropriate corrective actions.

The architecture is thus guided by the main principle that as long as each of the encompassed modules provides its designated services, exposing them through the generic interfaces and

respecting the requirements for scalability, robustness, pluggability and flexibility, there should be no dependencies in terms of underlying technology stacks or communication protocols.

As such, some functionalities should be present in any implementation of the proposed architecture, namely:

- The capacity for handling the plugging and unplugging of components during run-time, acquisition and pre-processing of raw data, which should be handled by the DAL.
- The integration between the DAL and DPL, supporting high throughput of data, handled by the DQL.
- The analysis of incoming (possibly high-volume) streams of data, along with the generation of predictive data, computation of trends, forecasts and correlations, for which the DPL is responsible.
- Finally, the capacity to display near real-time graphic representations of the different steps involved at the various stages, providing a means to better understand and interpret the data as well as to support run-time operations, should be associated to the DVL.

Each of these layers is described in further detail in the Section 3.

3. Architectural Elements

3.1. Data Acquisition Layer

Standing directly above the shop floor layer, the DAL is responsible not only for the acquisition of relevant data but also by its pre-processing in terms of the extraction of context-aware information. In regards to the data acquisition, the DAL needs to be flexible in order to adapt to changes coming directly from its sources in the shop floor, be it in terms of new components being plugged or unplugged, or even changes to the KPIs that need to be collected and analysed. Also, the communication with the shop floor needs to be specified in a generic way, thus allowing the consideration of different requirements from different potential use case. For instance, a specific case might present time constraints in the order of weeks or days, while a different one might require data to be collected and analysed in near real-time, therefore requiring different approaches. To this end, the DAL follows an approach similar to that presented in another successful European project, FP7 PRIME [8-10], in which a Cyber-Physical System (CPS) based approach was used. This approach is centred on a Multiagent System (MAS) architecture which abstracts both components and subsystems (e.g. cells, workstations) alike, as showcased in Figure 3.

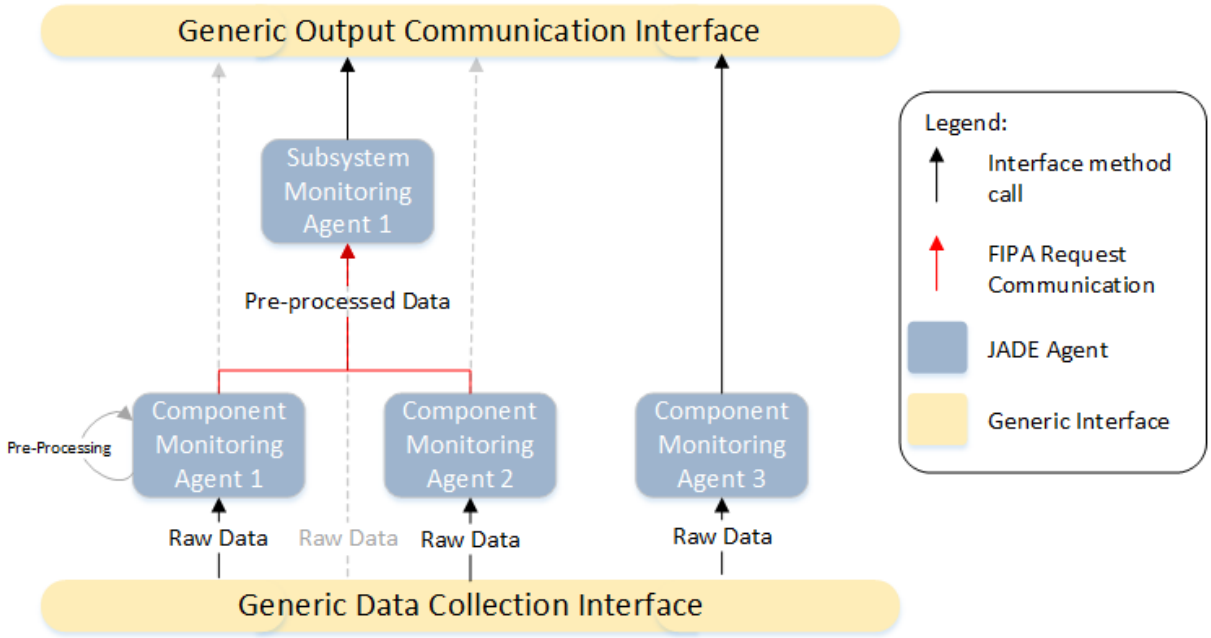

Figure 3 - DAL Multiagent System Overview

The adoption of MAS paradigm confers additional flexibility and robustness to the DAL, allowing it to quickly adapt to changes in the shopfloor. No less important is the existence of generic communication interfaces which allow the agents to interact with the environment in a ”black-box” fashion, regardless of the underlying technology or communication standard. In PERFoRM’s case, this means that the approach can be implemented in a way that the agents can communicate with the hardware via the harmonization middleware, or if required (e.g. specific time constraints), a different

instantiation of these interfaces would allow an approach closer to edge computing. Upon collecting the raw data, the agents can pre-process it in order to extract more meaningful information before passing it on to the upper layers, in this case the Data Queue Layer (DQL), which is described in further detail in Subsection 2.2.

3.2. Data Queue Layer

The DQL's main purpose is to serve as a distributed continuous buffer for the data coming from the DAL. It should add another layer of robustness, allowing for high-volume streams of data to be transported from the DAL in order to be consumed by the data analysis network. As such, it should provide reliability in terms of message delivery, which can be achieved through the sequencing and replication of data messages. More than a simple message queue, the DQL should be capable of not only handling a high throughput of data (in order for it to cope with the aforementioned varied time constraints), but also to enrich and filter or aggregate the buffered data as required in order to facilitate its consumption by the Data Processing Layer (DPL).

3.3. Data Processing Layer

The last core layer is the DPL, responsible for the actual data analysis of the inputs coming from the lower layers. In the context of PERFoRM, this analysis is meant to generate predictive data related to the KPIs relevant for each use case, producing forecasts and identifying trends and correlations between these indicators. As such, this layer enables the early detection of possible disturbances, degradation or KPI deviation from the expected boundaries in the shop floor. Hence, due to this capacity for predictive analysis, the DPL is a key-enabler of condition-based maintenance, allowing manufacturers to schedule maintenance operations before a failure actually occurs, thus diminishing the direct impact on production. Additionally, the DPL is not limited to assisting in run-time decision making (e.g. by interfacing with external data visualization tools, which are however outside the scope of this work), being also capable of triggering self-adjustment methods (e.g. self-reconfiguration) which can promptly perform corrections in order to return the system to a state of normal operation.

3.3.1. Generation of Predictive Data

The generation of predictive data comprises the application of mathematical models (such as those based on statistical techniques) to estimate future values or "guess" unknown events. The development of such models is supported and researched by the field of predictive data analysis [11], which encompasses a class of Data Mining [12-13], also known as predictive analytics or regression analysis. Predictive data analysis is applied to many fields, such as meteorology, financial markets, customer relationship management and others [10]. In this context, it is used to produce an output that can be

directly employed, for example, to control a variable or the whole system, or indirectly to drive and support a decision-making. Recently with the advances in sensor and data acquisition technologies, as well as advanced data analysis frameworks and techniques, more and more organizations and companies have integrated predictive analytics along all their operations and decision-making processes [22]. For instance, at the business levels, it can be used for planning tasks (expenditures, inventory and resource allocation, according with time and investment impacts), management tasks (failures in assets, improve employee allocation and productivity, reduce operational and maintenance costs, drive development and distribution phases. While at the operational level, it can be used for monitoring tasks (identify and diagnose leaks, critical issues and abnormal patterns), and for controlling (take action in real time to prevent fault, reduce handling time, alert operators of problems). The use of predictive analytics allows companies to better extract the value of their data and use it to improve and optimize their operations and processes through a more proactive and informative actions and decision-making [22].

Predictive data analysis comprises the use of several data mining techniques in order to analyse and extract patterns from current and past observations and build models (e.g., statistical inference or based on machine learning) capable to estimate values or predict events, usually for a future period. The predictive analytic models can be a simple univariate moving average model [14], which can estimate the next values based on the trend identified in the previous value, or a more complex neural network model [15], which support nonlinear and multivariate functions.

The moving average model is a common approach to predict future points in univariate time series (time series forecasting) [14]. It assumes that the future values depend linearly on the current and past observed values. It is a quite simple model that can provide accurate outputs for many application cases. However, it only supports stationary time series, i.e. time series which the mean and variance do not change over time. In order to support non-stationary and other features, in the context of time series forecasting there exist more complex models such as the ARIMA (autoregressive integrated moving average) [14].

In predictive analytics, while time series forecasting focuses in the prediction of values of a single time series, the regression analysis [16] comprise an approach employed to predict the value of a dependent variable (not necessarily a time series) based on the values of one or more independent variables. In this sense, a moving average model can be viewed as a simple linear regression model. The process of regression analysis modelling encompasses the analysis of how the value of the dependent variable behaves when the values of the independent variables vary. Then the regression model, usually consisting in a mathematical function, is created to be able to estimate the expected average value of the dependent variable given the values of the independent variables. For example, a health insurance company can use a predictive model that take into consideration many independent variables, such as age, gender and medical records of

past and current clients in order to estimate the most probable costs of a new client, with an acceptable level of reliability, and based on that define the insurance price.

In this context, machine learning [17] comprises another field that provides several approaches used for prediction that has an overlap with regression analysis. Usually machine learning techniques are used in cases where the relationship between input and output variables are very complex and its nature is unknown. In this context, these techniques emulate some human cognition aspects to learn the relationship between variables from training examples.

There are several techniques and algorithms to perform the predictive analysis modelling. The linear regression is one of the simplest, but there are others like logistic regression, which as opposed to previous techniques that estimate numerical variables, focuses in the prediction of a categorical variable. Additionally, there are a set of machine learning algorithms with mix concepts, such as regression trees, or those that are only based on machine learning techniques, such as neural networks and support vector machines [17].

The main focus of such techniques and algorithms is to generalize the patterns found in past and historical data in a predictive model that can be used to estimate values or predict events given the input variables, even if the input values were not presented in the historical dataset. In order to build such models several tasks should be performed (see Figure 4). In the process of data mining the main tasks comprehend data preparation (pre-processing and cleaning data), modelling (split data in training and test sets, and apply the algorithms to build the predictive models), and evaluation (use the model in the test set and evaluate the output accuracy). In this context the accuracy and usability of outputs will depend on the quality and quantity of data available to build the model. Usually, the prediction outputs are not so accurate, however they are accurate enough to generate some trustful values to the application.

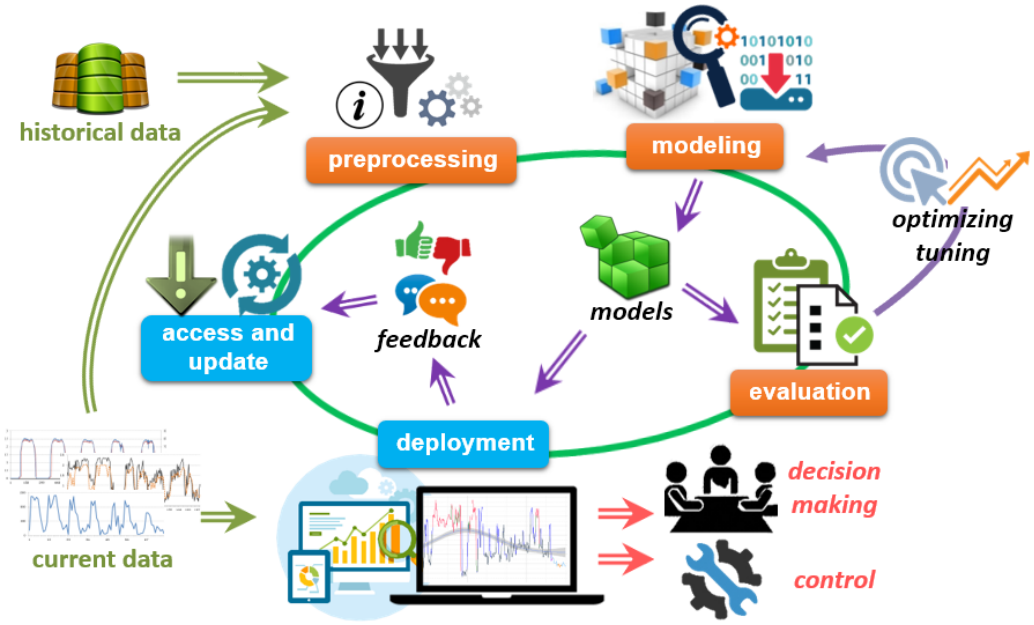

Figure 4 - Predictive Data Analysis Process Overview

In many scenarios, particularly in dynamic environments, the variables can evolve over time, presenting continuous changes, characterizing the drift concept [18]. This imposes some challenges for data analysis techniques, which need to support evolving data streams, requiring some adaptive learning mechanisms for a continuous assessment and update of the predictive models as new data become available and current models become obsolete. For example, predictive models to forecast the weather conditions need to consider the different seasons. Other challenges that can be faced by such techniques encompass the need to deal with missing values (information incompleteness), as well as computational and response time constraints, which can directly affect the accuracy of predictive models as also the selection of the predictive model itself.

3.3.2. Data Correlation

Data correlation comprises the process of analysing the statistical relationship between two continuous variables or sets of data, i.e., measure how and how much they are dependent or linked together [19-20]. For instance, considering two variables if their values increase or decrease together it characterizes a positive correlation, on the other hand a negative correlation indicates that the value of one variable increase while the other decrease. In this sense, the correlation between two datasets is given by a coefficient that ranges from 1 (perfect positive correlation) to -1 (perfect negative correlation). A coefficient equals to 0 means that there is no correlation.

The correlation analysis is widely exploited in practice since it can provide evidences that exist a behavioural pattern among two variables. For instance, if it is known that when the value of a variable increases, the value of the other variable also increases with a given rate, this can be used to understand the causes of an event or generalized in a

mathematical model that can be used to predict future values. In spite of that, the results of the correlation analysis need to be treated carefully, since it is easy and tempting to have early conclusions regarding that the changes in one variable is the cause of the changes in the other variable. However, correlation does not imply causation, i.e., in reality the appearance of a correlation does not mean that one thing causes the other, since there may be others unknown factors that influence the behaviour of the variables [21].

There are many techniques to compute the correlation coefficients. The Pearson correlation coefficient (also known as Pearson product-moment coefficient) [19-20] comprise the simpler and most common, which is only able to indicate that the relationship between two variables can be approximated by a linear function, i.e., the relationship follows a straight line. The Pearson correlation coefficient can be obtained by dividing the covariance of the two variables (i.e., how much they change together) by the product of their standard deviations (i.e., the variation or dispersion of their values). There are also others well-known techniques, based on rank correlation coefficients [19-20], such as Spearman rank correlation coefficients and Kendall rank correlation coefficients which don't require that the variables present a linear relationship.

As previously shown, correlation is a simple and suitable approach to identify and characterize the linear relationship between two variables. Multiple correlation can be computed by fixing a variable and using a linear function of the other variables. Another more specific type of correlation is the autocorrelation [14], which is mainly used in the field of signal processing where a signal is correlated with itself at different points in time in order to find periodic patterns.

On the other hand, there are more sophisticated and robust approaches that can be used to identify other types of relationships, e.g., that follows a curve, or even more complex relationships, e.g., when the behaviour of one variable is correlated with the behaviour produced by the combination of other two or more variables. Most of this approaches are inside the context of data mining [12], where descriptive and predictive analytics provide several techniques and methods that help to identify and understand the different kinds of relationships or correlations that can be found among the datasets, as well as the causes of these relationships.

In this context, regression analysis [16] is a well-known and widely used approach to understand and mathematically modelling the relationship between a dependent variable and a set of independent variables. Unlike correlation analysis, regression analysis can be used to infer and test causal relationships. In both descriptive and predictive analytics regression analysis can be used to understand the patterns and create models capable to describe and predict behaviours and events for different scenarios and applications (Figure 5).

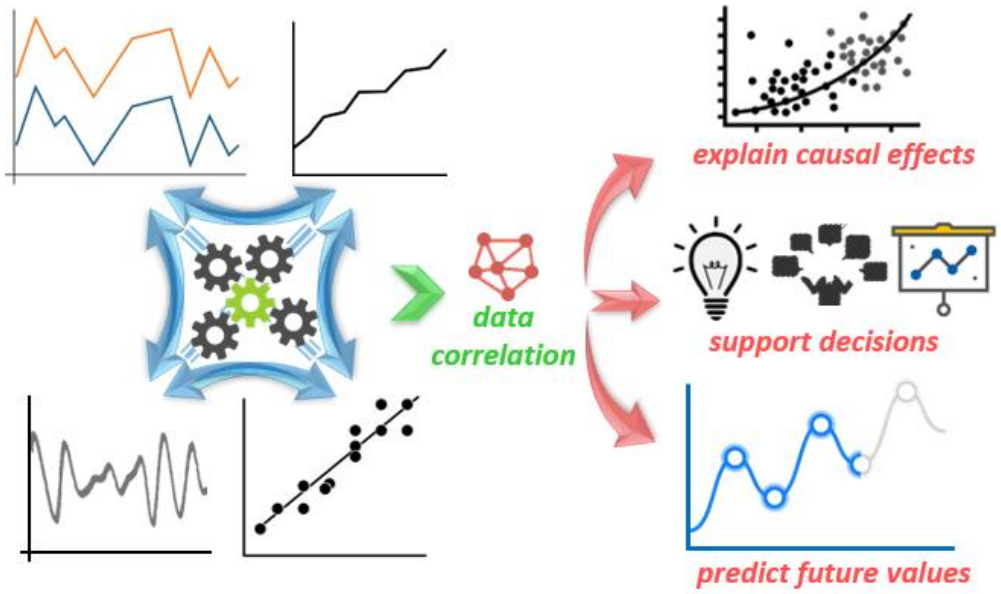

Figure 5 - Data Correlation Application Overview

3.4. Data Visualization Layer

In the past, the different areas involved in the manufacturing process were extremely disconnected, with information such as schedules, job orders and inventory lists being written down and passed along the line in paper format. This could typically lead to several miscommunication issues, as well as making any change required due to a given issue in production an excruciating task, consequently resulting in inefficient and often unpredictable procedures.

With the evolution of Information and Communications Technologies (ICT), along with the emergence of concepts like Cloud Computing, the Internet of Things and the Industry 4.0 movement (see Figure 6) modern shop-floors are generating larger and larger volumes of data. Researchers estimate that every year about 1 Exabyte (1M Terabyte) of data is being generated worldwide, of which a considerably large portion is available in digital form. Due to this, the approaches based on paper forms became unsuitable to deal with such a scenario.

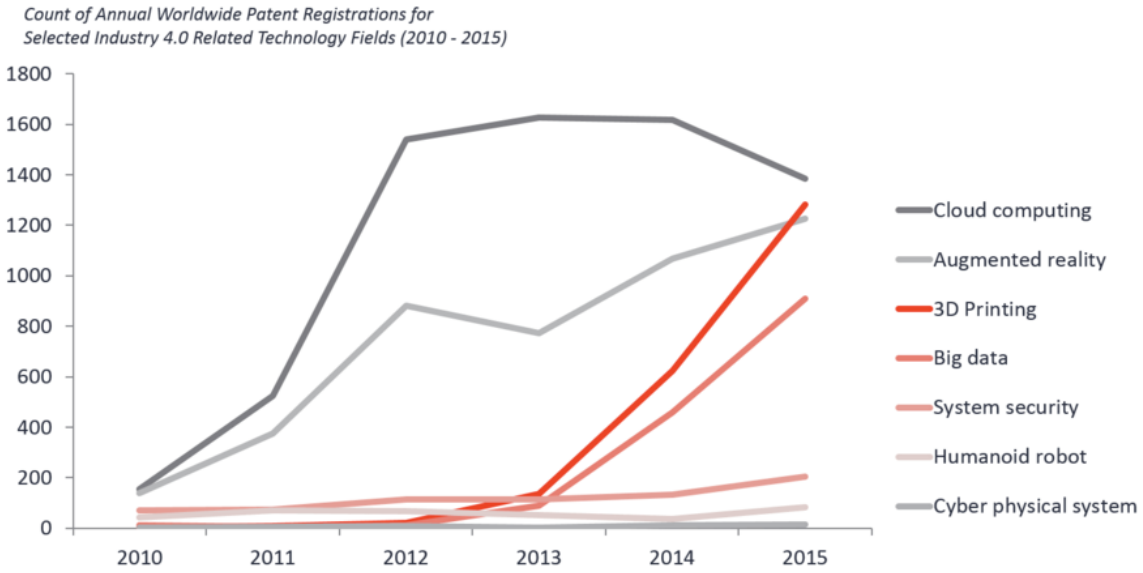

Figure 6 - Evolution of Industry 4.0 Related Technologies

Furthermore, if data is simply presented textually, finding meaningful information that can provide a business advantage in millions of data entries becomes a task similar to finding a needle in a gigantic haystack. Consequently, without a proper way to interpret and explore these large volumes of data, most of them are deemed useless and databases are more likely to act as data 'dumps' instead [23].

To avoid this, and in order for proper data analysis to be effective, it is very important to integrate the human in the loop, combining his or her general knowledge and creativity with the processing and storage capacities of modern IT. Hence, the main goal of data visualization is to present data in such a way that a human can get insight into said data which would have been incomprehensible otherwise, drawing conclusions and directly interacting and acting upon the data.

Bottom line is, data visualization enables or facilitates the comprehension of data, as well as the detection of patterns, trends and relationships contained in large complex data sets, being a largely important tool for data analysis when allied with a human's perception and knowledge.

4. Technical Description and Implementation

This section presents a possible implementation of each of the modular layers described in Section 3. Each of the chosen technologies is detailed along with an overview of their applicability in relation to the requirements already defined in the previous sections. However, due to the modular framework design, it is possible for any of these layers to be switched out for a different implementation, or even accommodate existing technologies (e.g. proprietary data acquisition systems) in a given shop-floor, thus being well aligned with PERFoRM's aim to confer legacy systems with the additional intelligence and flexibility that Cyber-Physical Systems enable.

4.1. Agent-based Data Acquisition Network

4.1.1. The JADE Framework

The Java Agent DEvelopment Framework (JADE) framework facilitates the implementation of agent-oriented approaches, serving as a MAS-oriented distributed middleware that provides a flexible domain-independent infrastructure. This infrastructure facilitates the development of complete agent-based applications by providing a run-time environment implementing the required basic features required by agents, their core logic and various auxiliary graphical tools [24]. JADE is written completely in Java, benefitting from the varied array of language features and third-party libraries widely available.

4.1.2. Component Monitoring Agent

The Component Monitoring Agent (CMA) class acts as the core for both the data extraction and pre-processing tasks. To allow an easier comprehension of its implementation, a class diagram of the CMA's associated data model is provided in Figure 7.

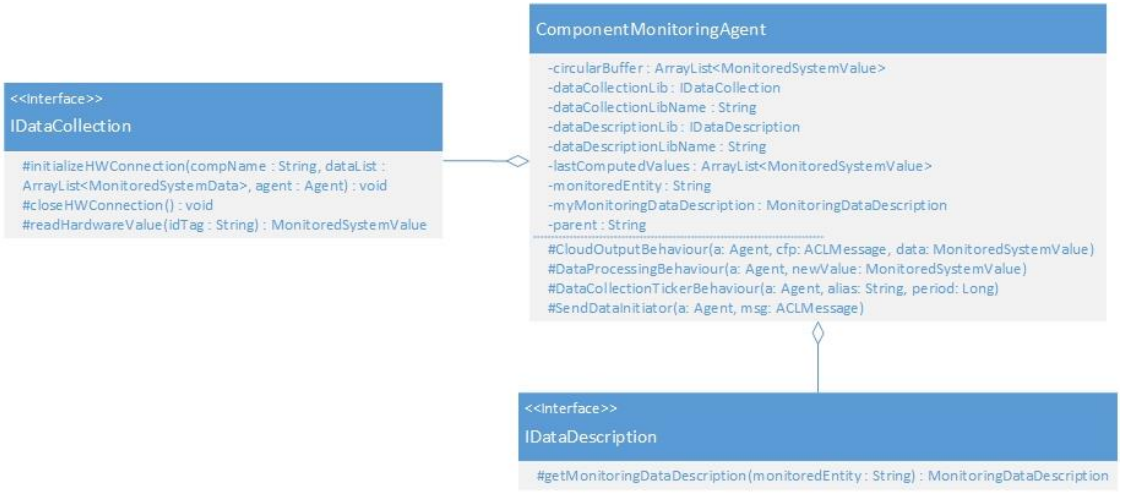

Figure 7 - CMA's Class Diagram

As seen above the CMA class has two different interfaces associated, *IDataCollection* and *IDataDescription*. The former possesses three different methods which enable the establishment of communications between the agent and the hardware, namely *initializeHWConnection* and *closeHWConnection*, and also the ability to read a given value indicated by a certain id tag via the *readHardwareValue* method. The *IDataDescription* interface provides the method which allows the agent to learn from and external source which device and associated data fields it is responsible for monitoring. This is achieved through the *getMonitoringDataDescription* method.

The CMA class also has four different behaviours implemented into its logic, supported by the following data fields:

- *circularBuffer* is the main data buffer where the latest extracted values are stored. As previously mentioned in 3.4.2, each CMA/HLCMA contains its own circular buffer in order for it to be able to compute any required values, taking into account recently extracted data. It is implemented as an *ArrayList* of elements of the *MonitoredSystemValue* class, which can be seen in Figure 4.5. This class contains a series of attributes that fully describe the associated abstracted value.

Figure 8 - MonitoredSystemValue Class Diagram

- *dataDescriptionLib* is an instance of a *IDataDescription's* implementation, required for the agent to learn which data values it should collect and process. Even though its implementation varies depending on the application, it should always provide the agent the method listed in Figure 7.
- *dataCollectionLib* is an instance of a *IDataCollection's* implementation. It enables the agent to communicate with the hardware in order to collect relevant data as learned in the process described in the previous bullet point. It should also always provide the agent the methods listed in Figure 7.
- *lastComputedValues* functions similarly to the *circularBuffer*, however only recently computed values are stored in it. Its existence allows the agent to save precious processing time by making sure it is not propagating values that have already been processed through the system. This could happen due to how the processing behaviour is triggered, which will be explained in further detail in a following section.
- *monitoredEntity* is a simple string containing the name of the component or subsystem being monitored (e.g. Gripper1, SafetyGroup2).

- *myMonitoringDataDescription* is an object containing the lists of *MonitoredSystemValue* elements that the CMA/HLCMA is expected to either collect or compute from extracted values. Both classes are detailed in Figure 9.

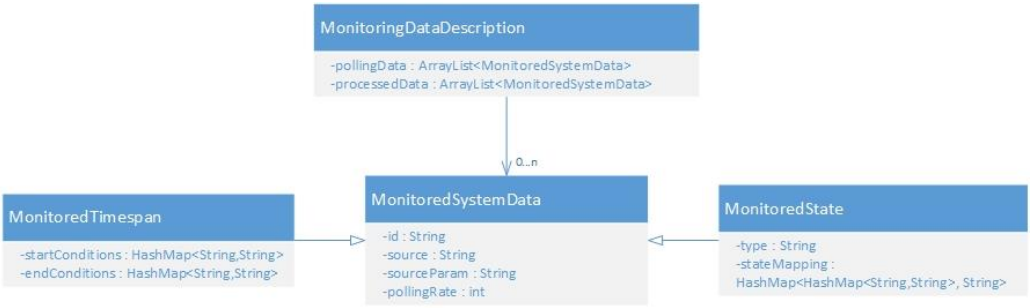

Figure 9 - Monitoring Data Description Class Diagram

- parent is a string that references as the name suggests the agent's parent in the monitoring tree.

4.1.2.1. Acquiring the Monitoring Data Description

In order for a CMA to start collecting and processing data it is mandatory that the Monitoring Data Description (MDD) is loaded before-hand. For this purpose, during the CMA's initialization an instance of the Event Description Library (EDL) is created, allowing the agent to call the *getMonitoringDescription* method which returns an object of the *MonitoredSystemValue* class. As described in Figure 9, this object contains two *ArrayLists*, pollingData and processedData, detailing which data the CMA needs to collect periodically and which values require computation on the agent's side. Figure 10 shows the steps executed by the CMA during this task.

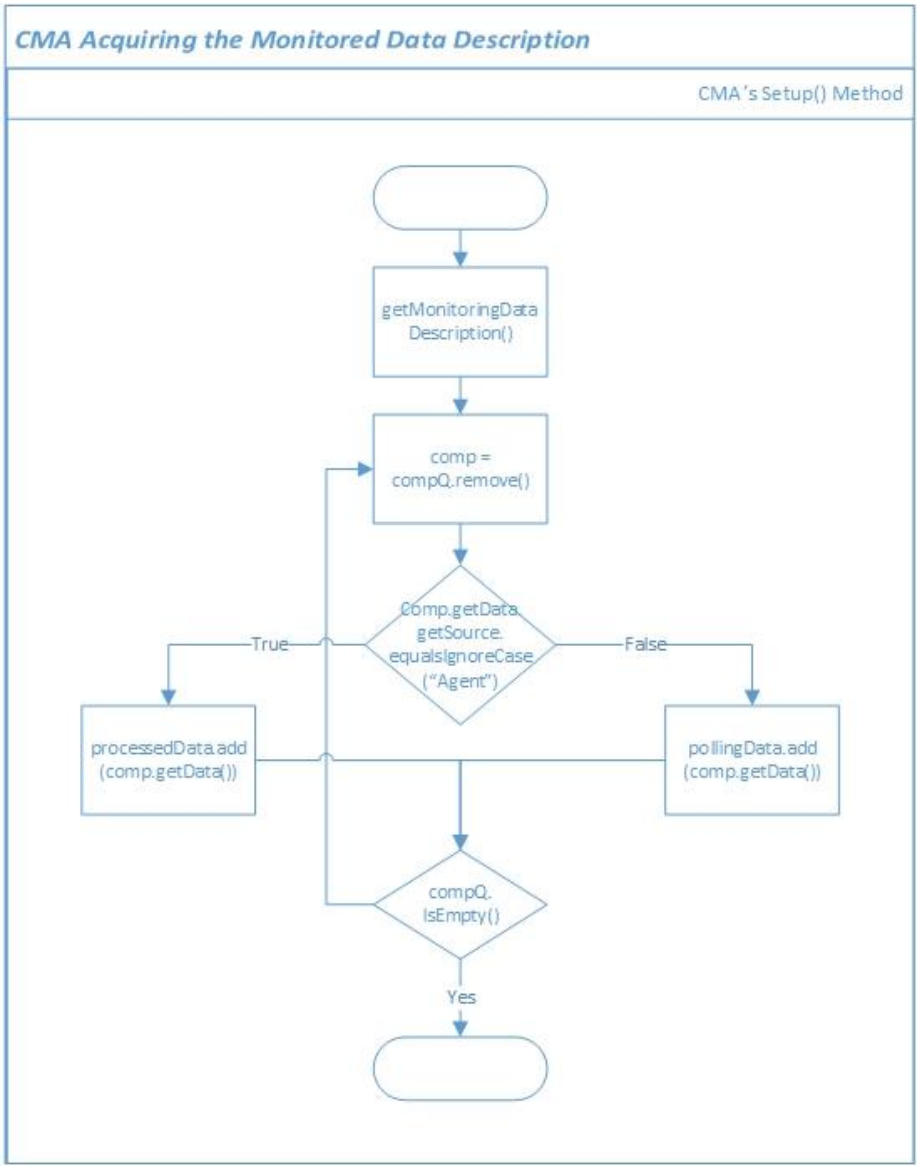

Figure 10 - Acquiring the Monitoring Data Description

As it can be seen in Figure 10, by using the EDL the CMA is able to retrieve a list of all the data related to its monitored component from an external source. It is pre-established that the MDD must provide the source for each data event described, therefore by checking this parameter the CMA can determine if the value needs to be calculated or if it is extracted directly from an external source (in case the parameter refers to anything other than the agent itself), separating the aforementioned *ArrayList* into the two data fields that make up the MDD class, *pollingData* and *processedData*.

4.1.2.2. Data Collection

Data collection can happen in two different ways, either by having the agent periodically extracting data or by having this process triggered by the detection of a change in a given monitored data value.

In the first scenario, considering that this is a repetitive process, a *TickerBehaviour* was chosen for the implementation of the *readHardwareValue* behaviour. This behaviour is repeatedly executed at a fixed rate, defined during the agent's deployment, as illustrated in Figure 11. During its initialization, the CMA iterates over the *pollingData* list contained within the *myMonitoringDataDescription* attribute, launching a *readHardwareValue* behaviour for each iterated element.

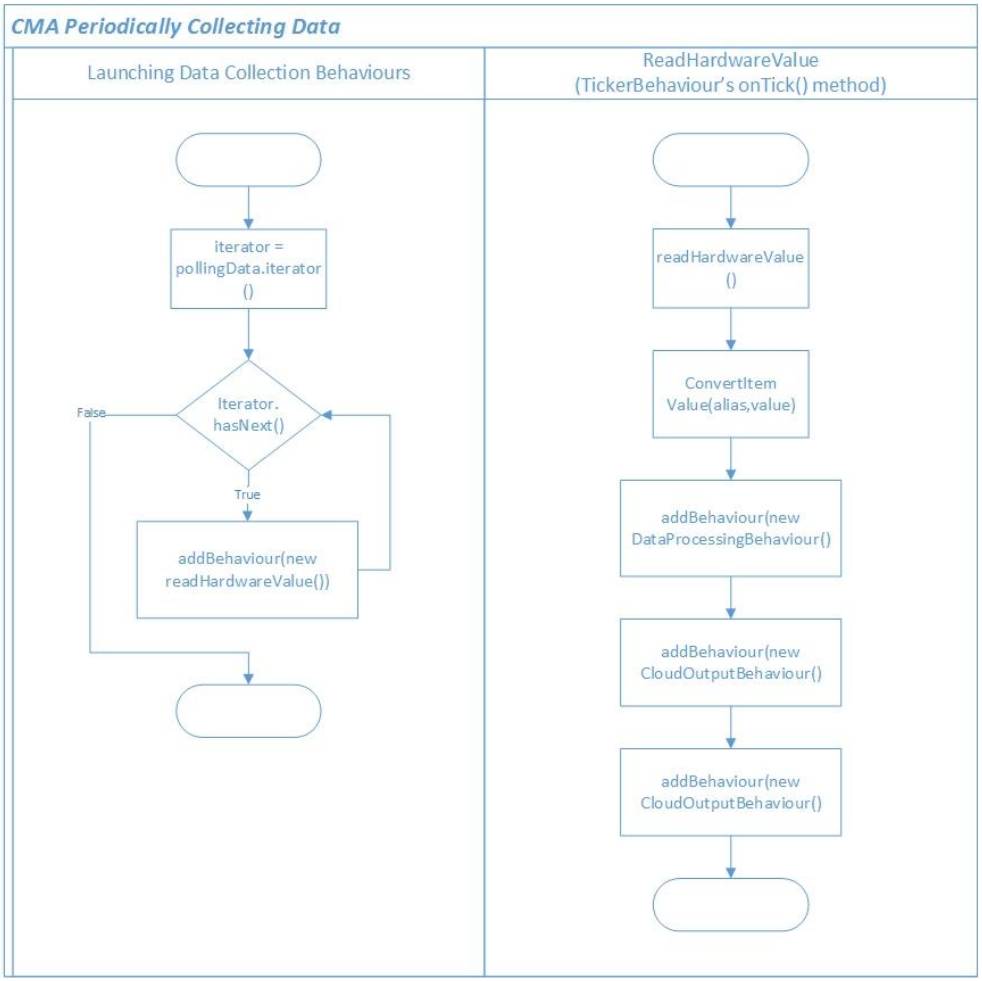

Figure 11 - CMA Periodic Data Collection Implementation

For the sake of enabling the agent-device interaction a communication interface is also required. For this purpose the Data Collection Library (DCL) provides three different methods, *initializeHWConnection*, *closeHWConnection* and *readHardwareValue*. The first two respectively establish and close the connection to the source device (this connection is maintained while the CMA is running), while the latter provides a means to extract a specific data value from it.

In the second case, the extraction is triggered by the DCL itself. In situations where this is supported by the underlying technology, with the agent reference passed as an argument in the *initializeHWConnection* (see Figure 7), the DCL is able to launch behaviours in the associated CMA. As such, using a simple event listener,

upon detecting a change in a monitored data value the DCL can extract it and using the agent reference it can directly deploy the behaviours responsible for sending the extracted data up the monitoring tree.

4.1.2.3. Data Pre-Processing

The *DataProcessingBehaviour* detailed in Figure 12 is responsible for handling all the computations required to calculate new values from the extracted data. Since this behaviour is simply launched as a consequence of new data having been collected, ending after it finishes its task, a *OneShotBehaviour* was used for its implementation.

The CMA starts off by iterating over the MDD *ArrayList* elements that describe which values it is expected to calculate. For each one of these it checks if the values stored in the *circularBuffer* at that given point in time meet all the associated sets of conditions, and if so it creates a new instance of the *MonitoredSystemValue* class to store the new computed data. Afterwards it stores the new data in the *newProcessedValues ArrayList* and moves on to test whether the remaining *processedData* elements can be calculated.

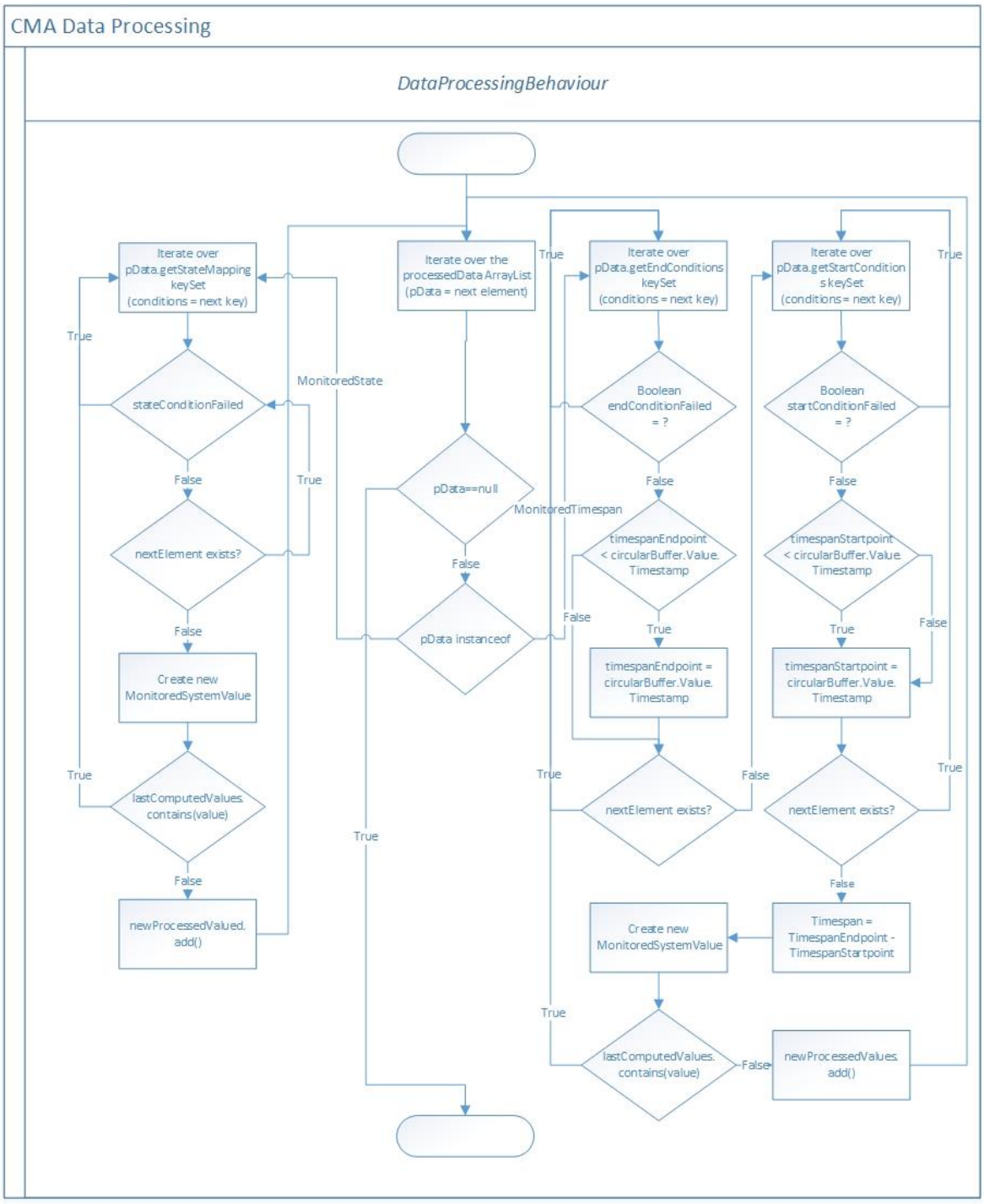

Figure 12 - Data Pre-Processing Behaviour Implementation

As previously shown in Figure 9, the conditions that define *MonitoredTimespan* and *MonitoredState* objects were implemented resorting to a *HashMap*. In the former's case two different *HashMaps* were used, *startConditions* and *endConditions*. Both use the correspondent state's ID as the key, while the pair is the actual state's value that satisfies the condition. For the latter only one set of conditions exist, more specifically the stateMapping, in which another *HashMap* is used as the key, and

the corresponding state designation is used as the value. A simplified example of a *stateMapping* for a gripper's current state can be visualized in Table 1.

Table 1 - Example of a gripper's stateMapping

Hardware I/O		Current State
Open	True	Open
Closed	False	
Close_Signal	False	
Open_Signal	False	
Open	False	Closed
Closed	True	
Close_Signal	False	
Open_Signal	False	
Open	False	Opening
Closed	False	
Close_Signal	False	
Open_Signal	True	
Open	False	Closing
Closed	False	
Close_Signal	True	
Open_Signal	False	

After all possible values have been calculated and stored in the *newProcessedValues* list, for each element contained in it the CMA initiates a new CFP through the *CloudOutputBehaviour* in order to select a suitable OCA to export the new data to external entities. In similar fashion, for each CFP initiated a Request is also sent to the CMA's parent through the *SendDataInitiator* behaviour.

4.1.2.4. Transmitting Monitored Data

The last task performed by the CMA is the transmission of both collected and processed data. This transmission occurs when either of the behaviours described in the previous two sections terminates its execution, and consists in two different behaviours responsible for sending said data to both the CMA's parent HLCMA and an available OCA from the Output Cloud.

The SendDataInitiator behaviour consists in a simple point-to-point communication between a CMA and its parent HLCMA. When the CMA finishes the execution of a *readHardwareBehaviour* or receives new data via the event listener in the DCL, it serializes the new data value and sends a request to its parent containing the serialized data. This can be seen in Figure 13.

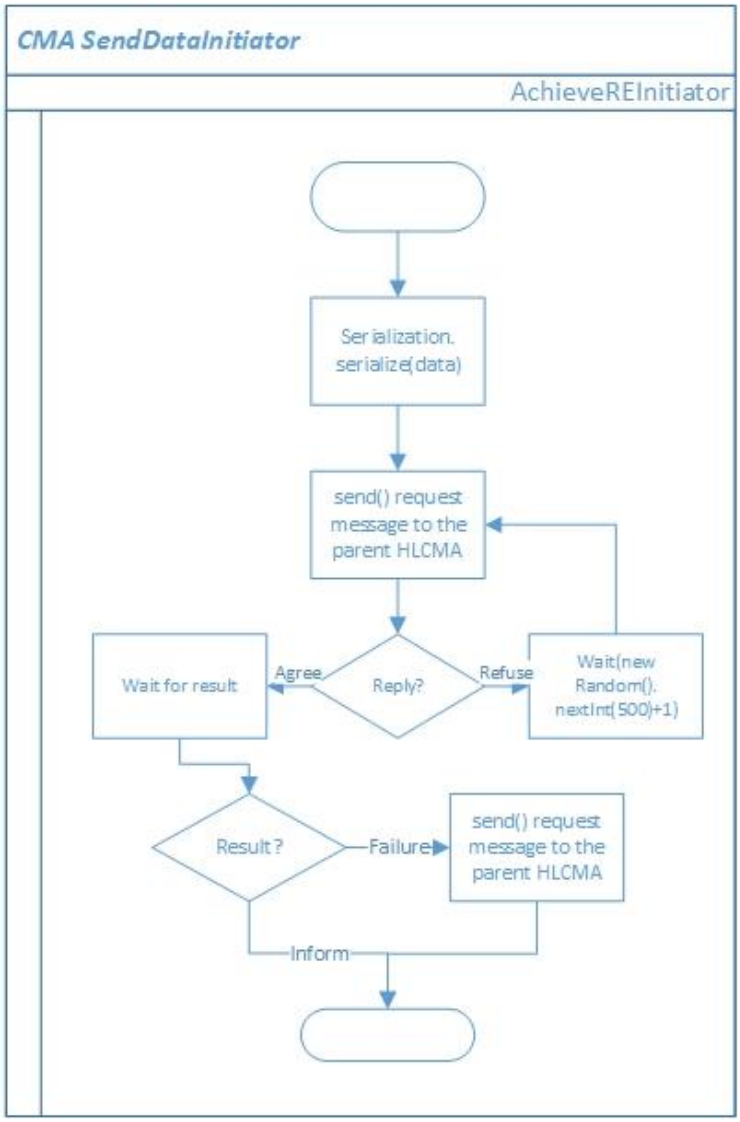

Figure 13 - SendDataInitiator Behaviour

The *CloudOutputBehaviour*, observed in Figure 14, differs mainly in the fact that whilst the communication between a CMA and its parent HLCMA is point-to-point through and through, in this case the interaction starts out between the CMA and possibly many different OCAs.

Firstly the CMA diffuses CFPs to the OCA cloud to find an available agent to process the data exportation. Upon receiving the proposals from possible OCAs, it evaluates them according to the established metric, in this case the response time, and selects only one to process its request, sending refusal messages to the rest. At this point the CMA serializes the fresh monitoring data and sends it in the *Accept-Proposal* message to the selected OCA, awaiting its response, finalizing the CMA's role in the process.

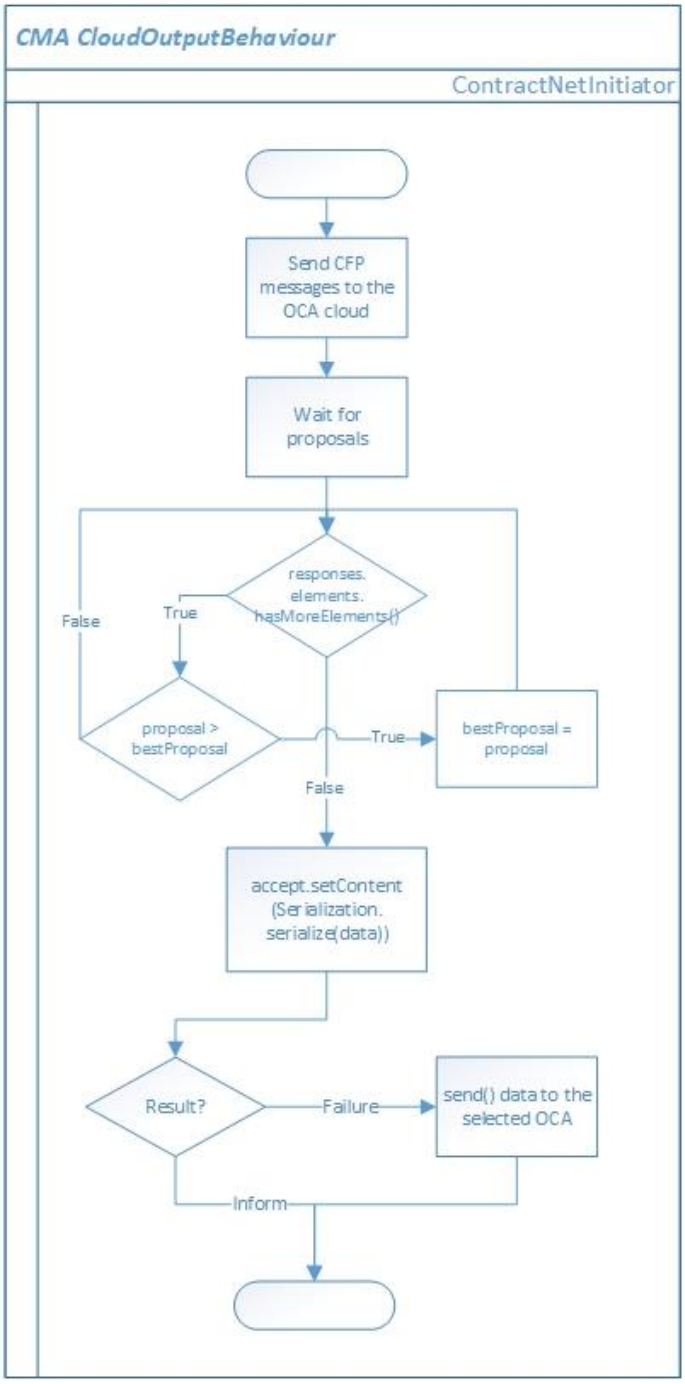

Figure 14 - CloudOutputBehaviour Implementation

4.1.3. Higher-Level Component Monitoring Agent

The CMA and HLCMA classes are very similar from an implementation standpoint, being that the latter is simply an extension of the former. As such, the HLCMA displays all the behaviours and attributes previously described for the CMA, having just the added capacity to receive and process data computed by lower-level CMAs or HLCMA s. The data model representing this extension, along with the relationship between the CMA and HLCMA classes is presented in Figure 15.

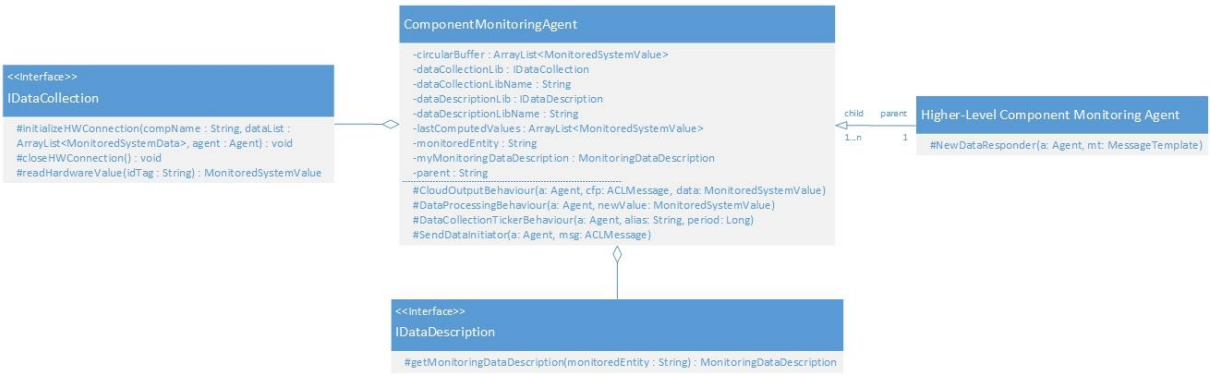

Figure 15 - HLCMA's Data Model - Class Diagram

As illustrated, the HLCMA class inherits the CMA's data fields and its behaviours, presenting a similar functionality. Also, as indicated, a HLCMA can have multiple CMAs associated to it, however, each CMA can only have exactly one parent HLCMA, or none.

The point where the CMA and HLCMA differ is in the existence of the *NewDataResponder* behaviour, which will be described later. Being a higher-level entity, the HLCMA can only gather data from the subsystem it abstracts but also from the agents in the lower layers of the monitoring tree.

4.1.3.1. Receiving Pre-Processed Data

The *NewDataResponder* behaviour is based on JADE's *AchieveREResponder* class, allowing the HLCMA to process a CMA/ HLCMA's Request message containing data processed at a lower-level of the monitoring tree. This behaviour is detailed in Figure 16.

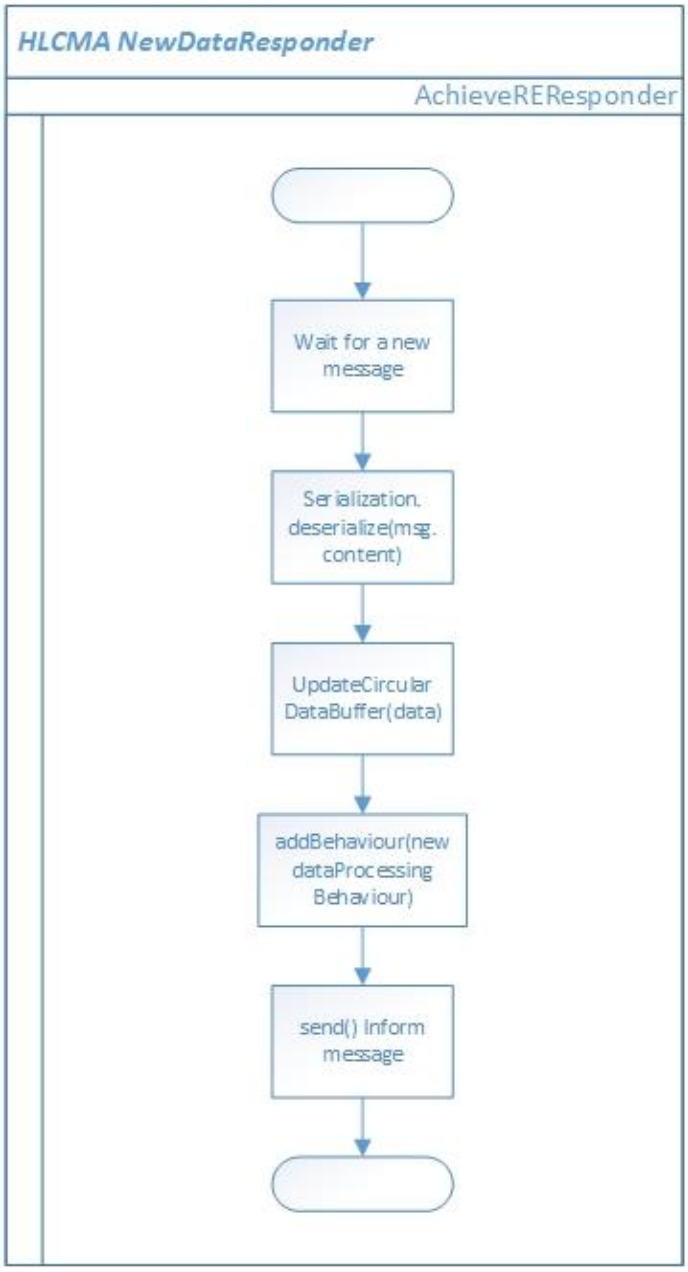

Figure 16 - NewDataResponder Behaviour

As seen in Figure 16, after receiving a Request message from an associated lower-level CMA/ HLCMA, the parent HLCMA deserializes its content in order to obtain the newly processed data, storing it in its local *circularBuffer*, initiating the *DataProcessingBehaviour* afterwards.

4.1.4. Output Coordinator Agent

The OCA acts as the gateway that allows data to flow from the monitoring architecture to external entities, relaying that information through the use of a *Data Output Interface* (DOI). This agent can existing as a singleton entity, or as a cloud of distributed OCAs in order to avoid a single point of failure in the output communications. The class

diagram depicting the data model for both the OCA and the DOI can be observed in Figure 17.

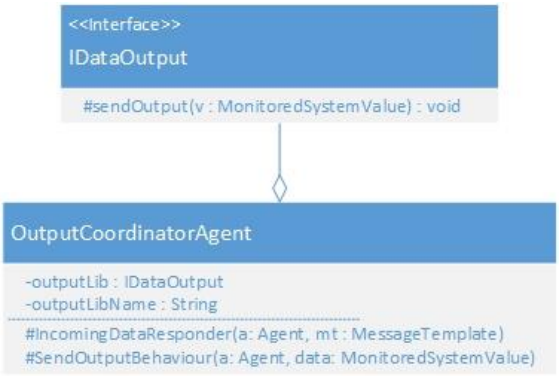

Figure 17 - OCA's Data Model - Class Diagram

Any implementation of the interface must provide a sendOutput method that allows the OCA to send objects of the *MonitoredSystemValue* class to relevant external entities such as a remote historical DB or a large processing network.

4.1.4.1. Exporting Data

The OCA's behavioural logic comprises two different behaviours, the first of which is the *IncomingDataResponder*, described in Figure 18. Having been implemented based on a *ContractNetResponder* behaviour, it gives the agent the capability of handling negotiation requests (in the form of a CFP, as later described in Figure 21) from a CMA/HLCMA, replying with proposals and handling their response.

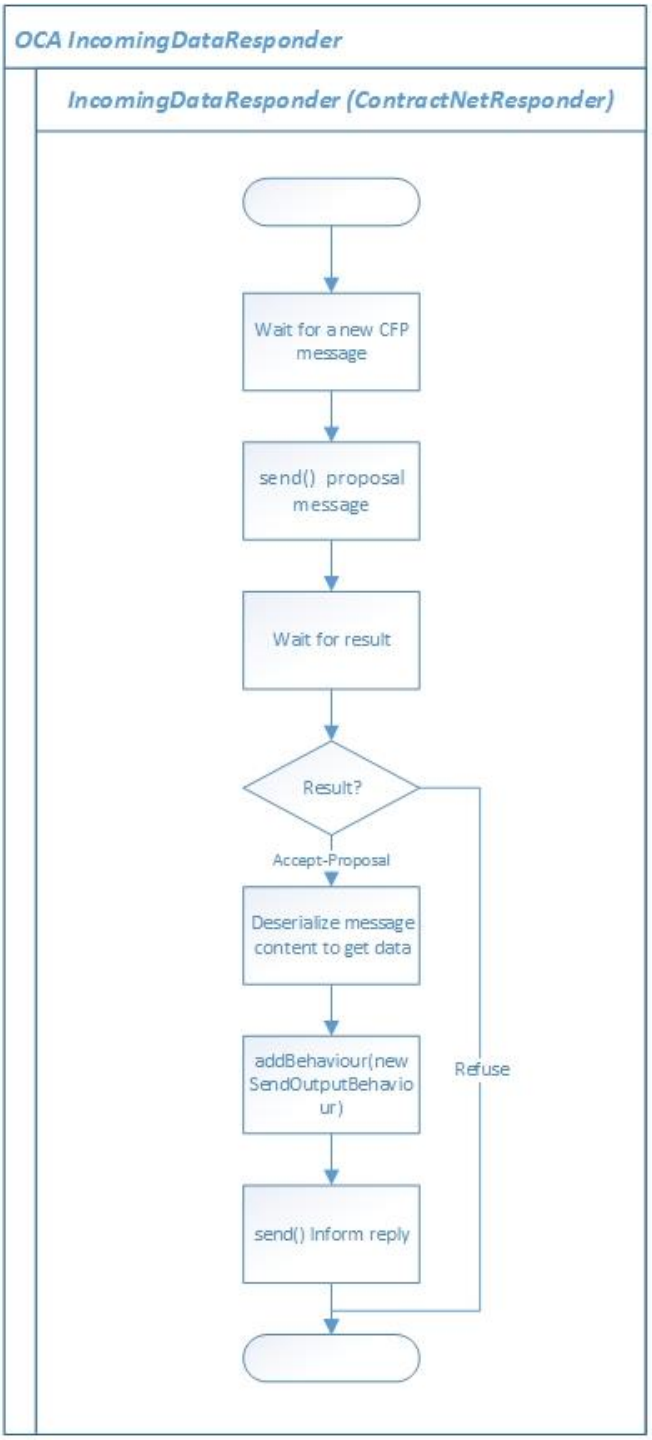

Figure 18 - OCA's IncomingDataResponder

Upon receiving an Accept-Proposal message, the OCA's IncomingDataResponder launches the *SendOutputBehaviour*, an extension of the *OneShotBehaviour* class, which in turn calls the DOI's corresponding method, *sendOutput*, which interfaces with the external entities in order to relay the monitored data to them. Since the DOI is a generic interface, the agent isn't concerned with neither its implementation nor

the type of external entity it interfaces with, meaning that the agent's logic is unchanged regardless of the technology used in the other end.

4.1.5. Agents Communication

Since all JADE agents are FIPA compliant [25], the communications established between were implemented according to the specifications of two different FIPA protocols, FIPA Request and FIPA Contract Net. Both protocols are analysed in the sub-sections ahead.

4.1.5.1. FIPA Request Protocol

The FIPA Request Protocol [26] allows agents to perform point-to-point communications, being therefore able to request another agent to perform a certain action. As illustrated in Figure 19, this protocol specifies that the communication starts when the Initiator agent sends a request to the Participant agent.

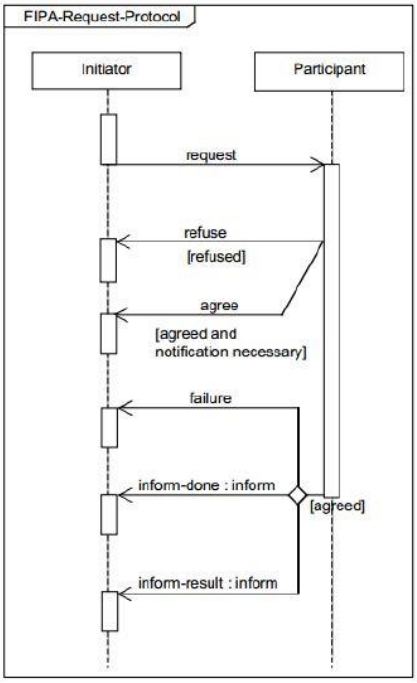

Figure 19 - FIPA Request Protocol

Upon receiving a request, the Participant can either accept it, sending an agree as a reply, or refuse it sending a refuse message back. After it finishes performing the requested action, in case it was completed successfully the Participant sends an inform message to the Initiator instructing it of its completion. Otherwise it sends a failure message.

4.1.5.2. FIPA Contract Net Protocol

The Contract Net Protocol (FIPA, 2000) promotes a negotiation between an Initiator and several Participant agents, allowing the former to evaluate which Participant agent or agents are more suitable to perform a certain task.

As it can be seen in Figure 20, the protocol dictates that communication starts with an Initiator sending a call for proposals to *m* Participant agents, where *m* is the given number of agents, which can in turn refuse the communication if for some reason it cannot perform the requested task, or reply with a proposal otherwise. After the Initiator has received all the responses, regardless of them being refusals or actual proposals, it can start evaluating them.

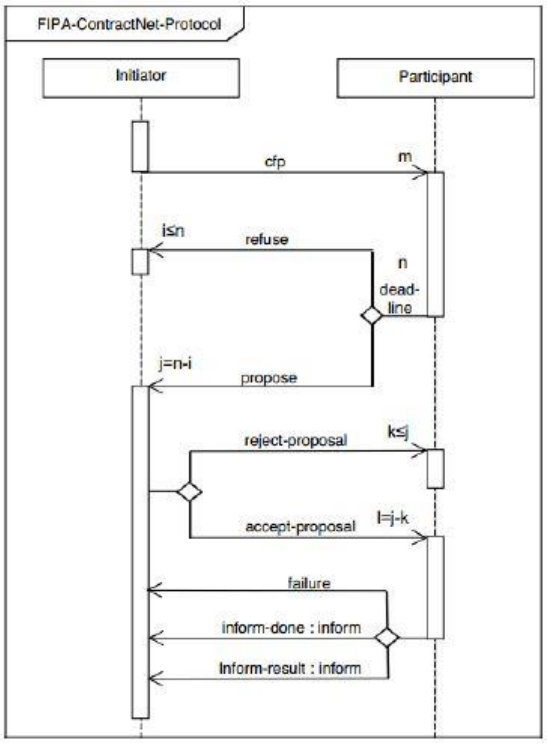

Figure 20 - FIPA Contract Net Protocol

For each proposal that the Initiator accepts it will send an accept-proposal message to the associated Participant, who starts processing the requested task. Upon the completion of the requested task, each Participant replies with an inform message indicating success, or ultimately if the task was unsuccessful a failure message is sent instead.

Proposals that do not pass the evaluation process also receive a reply, in this case a reject-proposal message is sent to the associated Participants, terminating the interaction between them.

4.1.5.3. Agents' Interactions

These interactions are summarized in Table 2, where each row corresponds to a different communication initiator (INIT) and each column represents the different responders (RESP).

Table 2 - Summary of the Agent Interaction

RESP / INIT	CMA	HLCMA	OCA
CMA		FIPA Request	FIPA Contract Net
HLCMA		FIPA Request	FIPA Contract Net
OCA			

As suggested by [27], in a distributed scenario the use of JADE's standard message transfer protocol conjugated with potential network delays takes its toll in the system's performance as a whole. For this reason the interactions between the agents that constitute the monitoring infrastructure were kept to a bare minimum to improve the system's overall performance.

A more complete view of the entire process can be seen in the sequence diagram depicted in Figure 21, which showcases the data flow from the collection at the machinery level up until it is transported to the higher-level layers via the OCA.

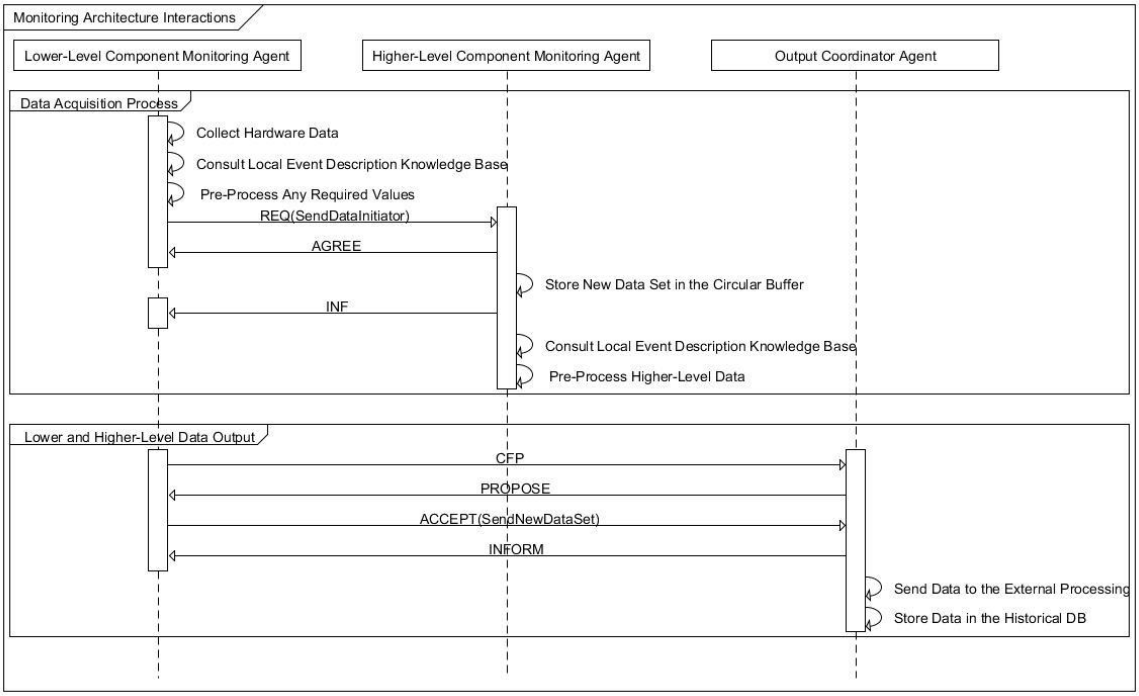

Figure 21 - Monitoring Agents' Interactions

4.2. Data Message Queue Layer

4.2.1. Apache Kafka

Any implementation of the DQL needs to take into consideration the requirements specified in Subsection 3.2 (fundamental architectural principles and the requirements for the DQL, respectively), more specifically in terms of scalability, capacity to handle high volumes of data, low latency and reliability. With this in mind, Apache Kafka [28, 29] is proposed as a possible framework to implement such a data queue.

Kafka is a fault-tolerant, highly scalable, distributed messaging system. In essence, it functions with a publish-subscribe approach, allowing producers (data sources, in this case the agents in the DAL) to publish data messages which are maintained in categories called topics, as observed in Figure 22.

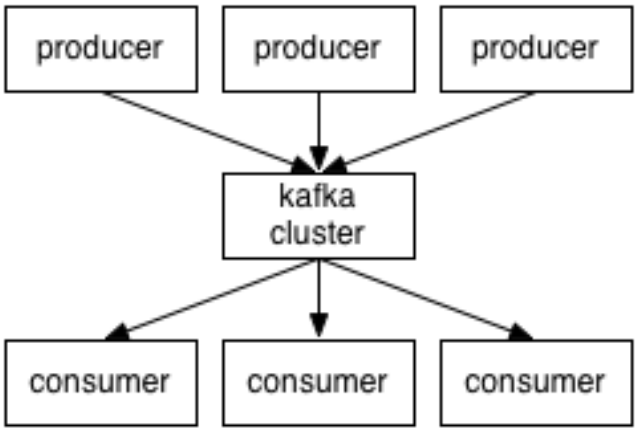

Figure 22 - Apache Kafka Overview

These can be subscribed by consumers (represented by the DPL nodes), being divided into ordered partitions supporting message persistence and replication. Kafka's message management is optimized for low latency and high throughput, with documented uses for even real-time applications [30].

Thus, for the purpose of this implementation, Apache Kafka acts as the link between the MAS and Storm's entry point (the spout element, detailed in Section 4.3), as illustrated in Figure 23.

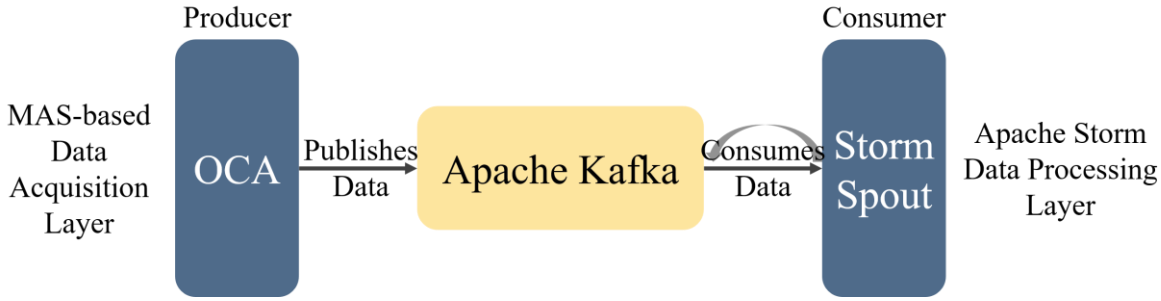

Figure 23 - Kafka's Implementation as an integration layer

4.3. Data Analysis Layer

4.3.1. Apache Storm

Finally, for the DPL Apache Storm is considered, being a distributed stream processing system which easily integrates with both databases and queuing technologies (such as the aforementioned Apache Kafka).

Storm is a distributed real-time computation system, working somewhat similarly to how Hadoop provides a set of general primitives for doing batch processing, it provides a set of general primitives for doing real-time processing.

Storm's processing runs in topologies, which wire data through a series of nodes in a directed acyclic graph (DAG), each containing certain processing logic, with the associated links specifying the data flow. An overview of this behaviour can be seen in Figure 24.

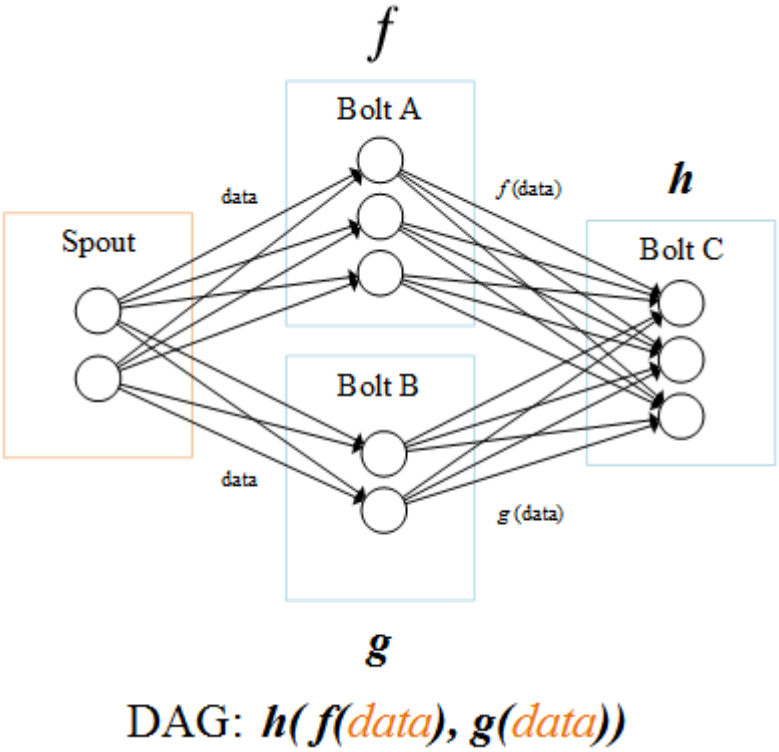

Figure 24 - Apache Storm Overview

Furthermore, Apache Storm is an extremely versatile tool, supporting the development of spouts and bolts in several other languages other than Java, through the inclusion of a "Multi-Language" (or "Multilang") protocol. As such, bolts with complex data analysis functionality that would be extremely difficult and elaborate to write in Java can be written for instance in R, being later on easily integrated into the topology like a "regular" bolt.

Taking this characteristics into account, and focusing on the PERFoRM requirements and goals, a possible topology was implemented to tackle the project's particular needs in terms of real-time data analytics. This topology is represented in Figure 25.

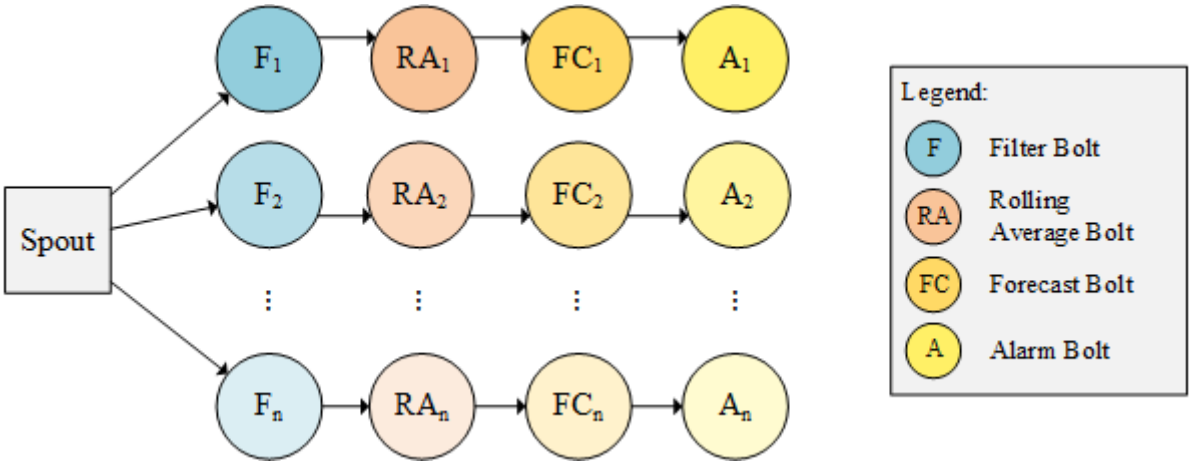

Figure 25 - Real-time Data Forecasting Topology Overview

Each of the individual components comprised in the proposed Storm topology will be approached in further detail in the coming subsections.

4.3.1.1. ComponentSimulationSpout

The *ComponentSimulationSpout* (CSS) was implemented to serve as a data source to be used for testing the Storm Real-time analysis topology.

In essence it has three main operation modes, which are:

- **Regular operation**: Normal behaviour simply consists in a random integer between a given minimum and maximum value being generated and emitted at a specific data rate, simulating the acquisition of simple data from a component under normal conditions.
- **Spike Failure**: This mode of operation simulates an equipment failure which causes a sudden spike in the absolute value of the extracted data (maintaining an increased value thereafter), allowing for instance to test the topology's behaviour when data suddenly spikes above the threshold boundaries defined as regular operation conditions.
- **Incremental Failure**: Contrastingly, this mode prompts the absolute value of the simulated data to be increased steadily over a given period of time, enabling the testing of trend detection and prediction of future values based on said trend.

With these three modes of operation, different situations can be simulated in order to test the topology's performance under a controlled setting. However, once the topology is required to be applied in a production environment, this spout can be simply replaced by a Kafka spout which consumes data from a Kafka topic, or a different kind as deemed necessary based on the associated technologies.

4.3.1.2. FilterBolt

As the source spout (regardless of it being the aforementioned simulation spout, or a production ready one) propagates the source data downstream (see Figure 25), a bolt responsible for filtering component specific data is necessary. This functionality can be seen in Figure 26.

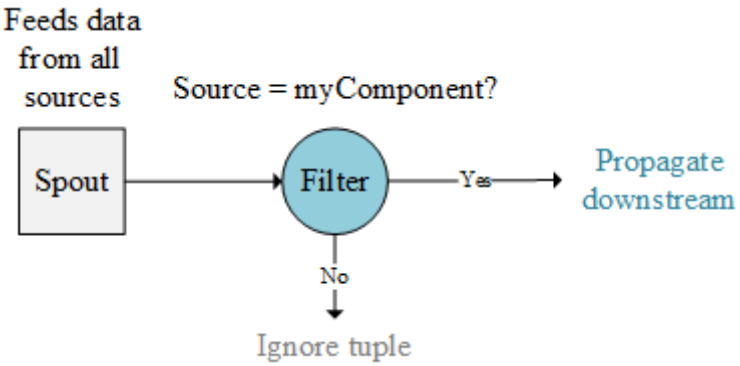

Figure 26 - *FilterBolt* Functionality

Simply put, the data coming from the source spout is checked to see whether or not it pertains to the associated component. If it is, the bolt propagates it further downstream, otherwise it is ignored so that the appropriate sub-topology can process it instead.

4.3.1.3. RollingAverageBolt

A simple rolling average (also referred to as simple moving average, or SMA), is an unweighted mean of the last *n* values of a given data set, as dictated in the formula below:

$$SMA = \frac{\sum_{i=1}^{n} Vi}{n}$$

The calculation of a rolling average serves to smooth the data, aiding in the detection of emerging trends and patterns that could be otherwise hard to observe. Being based on the x most recent values, it can provide a more realistic basis for forecasting future values and take corrective action as needed.

The *RollingAverageBolt* (RAB) employs such a formula, storing incoming raw values in a sliding window with a fixed size, emitting the *SMA* value on each iteration (every time new data arrives), as illustrated in Figure 27.

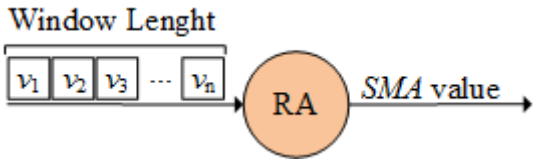

Figure 27 - *RollingAverageBolt* Functionality

The RAB was implemented by extending the functionality of a *BaseWindowedBolt*, essentially maintaining a window of a fixed sized passed as an argument upon initializing the topology, along with a window update rate (e.g. every new value). As such, new data tuples are added to a sum value and kept there for the duration of the window, being removed (subtracted from the sum) upon expiration. On each iteration, the sum is divided by the window length and the resulting *SMA* value is propagated downstream.

4.3.1.4. ForecastBolt

The implemented *ForecastBolt* (FCB) estimates an ordinary least squares regression model with a single independent variable based on the formula below, where *m* and *b* represent the slope and the intercept, respectively.

$$y = mx + b$$

Observations (*x*,*y* pairs) can be added one at the time to the model as they are received by the node in order to update it to enable forecasting future *y* values for a given *x*. These observations are not stored in memory, so theoretically there is no limit to the number of observations that can be added to the model itself.

4.3.1.5. AlarmBolt

The *AlarmBolt* (AB) is responsible for processing the incoming forecasts produced by the FCB in order to ascertain which values surpass a given threshold, triggering an alarm response. An overview of the bolt's behaviour is provided in Figure 28.

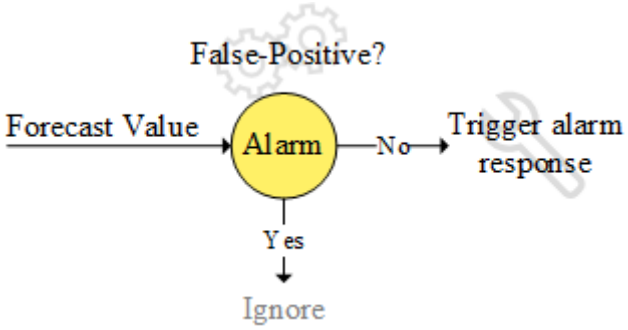

Figure 28 - *AlarmBolt* Functionality

Additionally, the possible existence of false positives is also considered. As seen in Figure 28, which depicts a case where a forecast above the alarm threshold has been detected, each AB evaluates the incoming forecast values according to a given set of rules, triggering an alarm response only when these rules are met, whilst ignoring singleton irregularities in the data, thus acting as a slightly more complex *FilterBolt*. These alarm responses can in turn be used as triggers for reconfiguration mechanisms, predictive maintenance or other such behaviours.

4.4. Data Visualization Layer

The data visualization was implemented as a Java application which updates each chart as new data arrives in near real-time. This makes it easier to visually understand and interpret the data being outputted by the processing network, allowing it to be used for instance at the shop-floor level as a Human-Machine Interface (HMI) to support operators in run-time. An overview of the data visualization application can be seen in Figure 29.

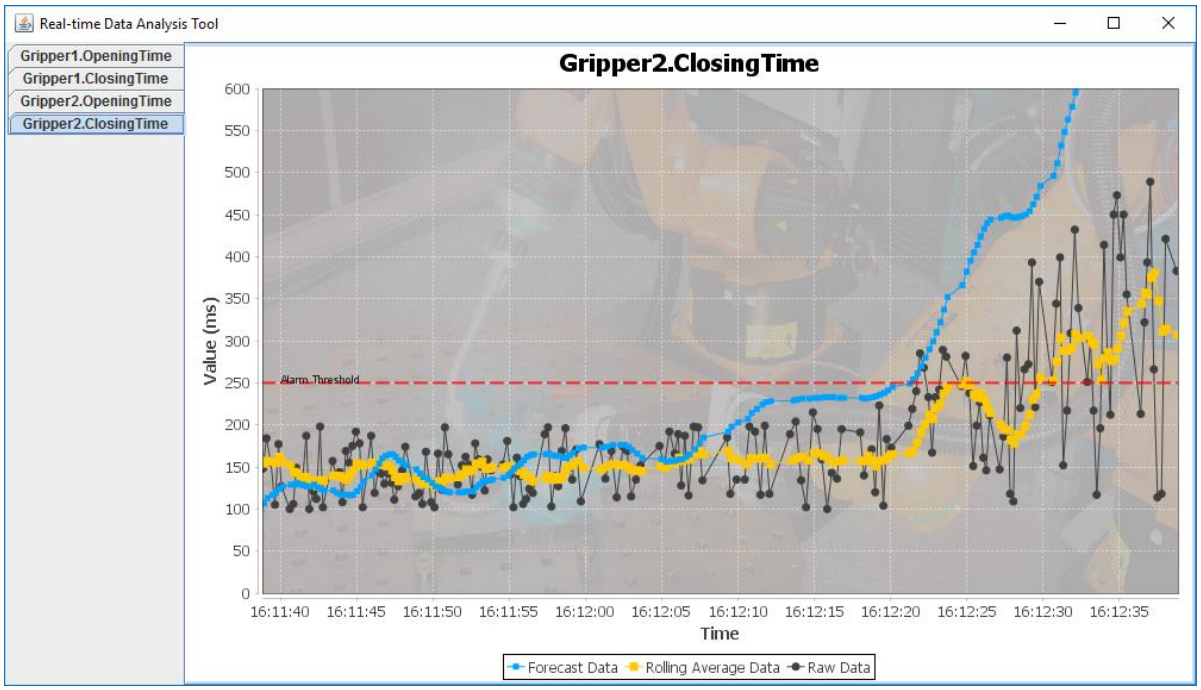

Figure 29 - Data Visualization

As depicted, the application allows the user to navigate to each of the monitored Key Performance Indicators (KPI), displaying relevant information such averages and forecast data in near real-time.

Furthermore, Figure 29 showcases the incremental failure scenario described in Subsection 4.3.1.1, where the grey plot represents the observed raw data which steadily escalates, the yellow values provide the smoothed data calculated by the rolling average and finally the forecast for 500 values into the future, along with the pre-defined alarm threshold which acts as a boundary for normal operation conditions.

5. Preliminary Tests

As indicated in Subsection 4.3.1.1, a simulation spout was implemented in order to emulate the behaviour of a shop-floor component. In essence, the *ComponentSimulationSpout* is configured according to the following settings:

- ***Data Rate**:* As the name suggests, specifies the intervals at which a new observation (*x*,*y* pairs) is generated.
- ***Data Range**:* Indicates the minimum and maximum values between which the new data point is generated.
- ***Forecast**:* The *forecast* parameter defines the number of *x* entries into the future for which the corresponding *y* pair value should be forecast.
- ***Failure**:* Acts as a control variable in order to define the spout's operation mode, being either normal or failure.
- ***Failure Type**:* If failure mode is enabled, this parameter defines the type of failure to be simulated, more specifically spike or incremental.
- ***Failure Duration**:* Lastly, the *Failure Duration* indicates how long the failure condition will last.

The tests documented in this section consisted in running the topology in the three different modes, namely normal operation conditions, simulated spike failure and incremental failure. Using the data visualization component described in Subsection 4.4, the behaviour illustrated in Figure 30 was obtained for normal operation conditions.

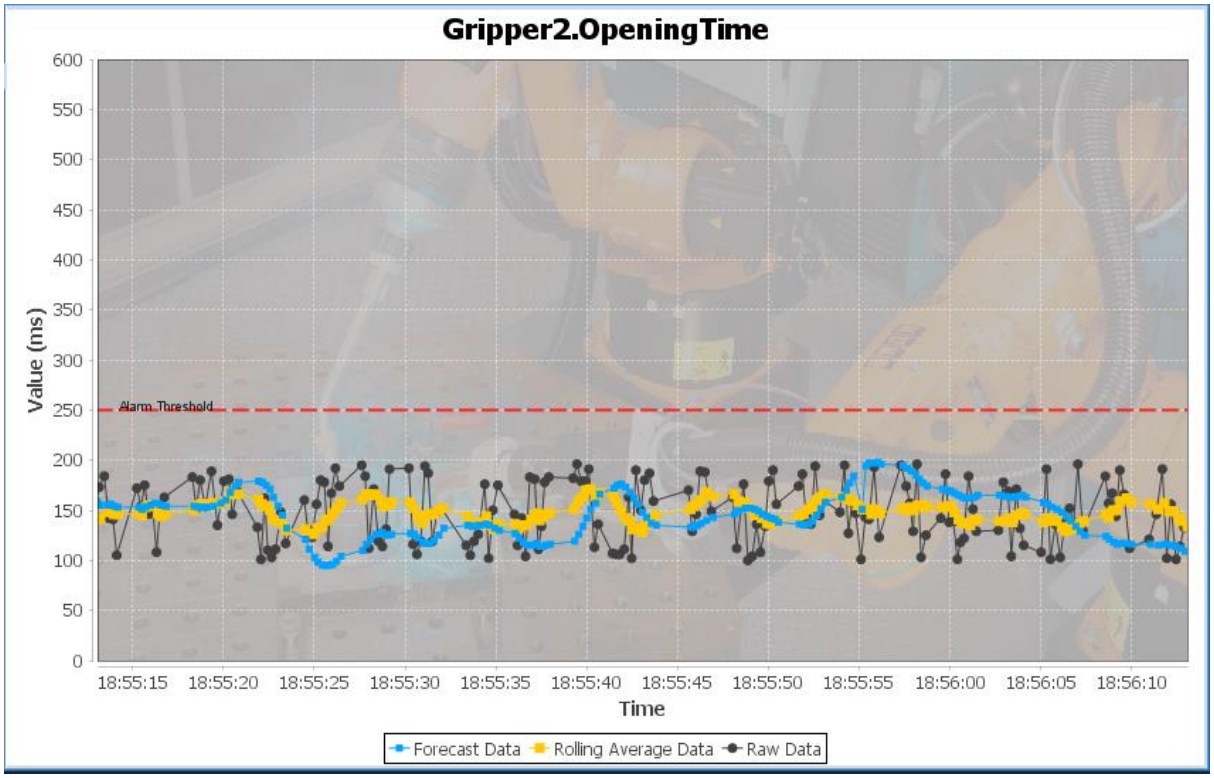

Figure 30 - Simulation Scenario - Normal Conditions

For these tests, the settings were defined as shown in Table 3:

Table 3 - Simulation Settings

Data Rate	100 (ms)
Data Range	[100 , 200] (ms)
Forecast	500
Failure	TRUE/FALSE
Failure Type	SPIKE/INCREMENTAL
Failure Start Delay	600 * *Data Rate* = 60000 (ms)
Failure End Delay	1200 * *Data Rate* = 120000 (ms)

As seen in Figure 30, three different data types are charted. The dark grey line represents the raw values coming directly from the source spout, with the values ranging from 100 to 200 milliseconds as specified in the *ComponentSimulationSpout* settings (see Table 3). The rolling average can be seen in yellow, representing and smoothed dataset, eliminating outliers and preparing the incoming data points to be processed. Finally the forecast is shown in blue, depicting the predicted value of the *y* variable 500 data point entries into the future.

Upon inducing a spike failure, the corresponding reaction can be seen in Figure 31.

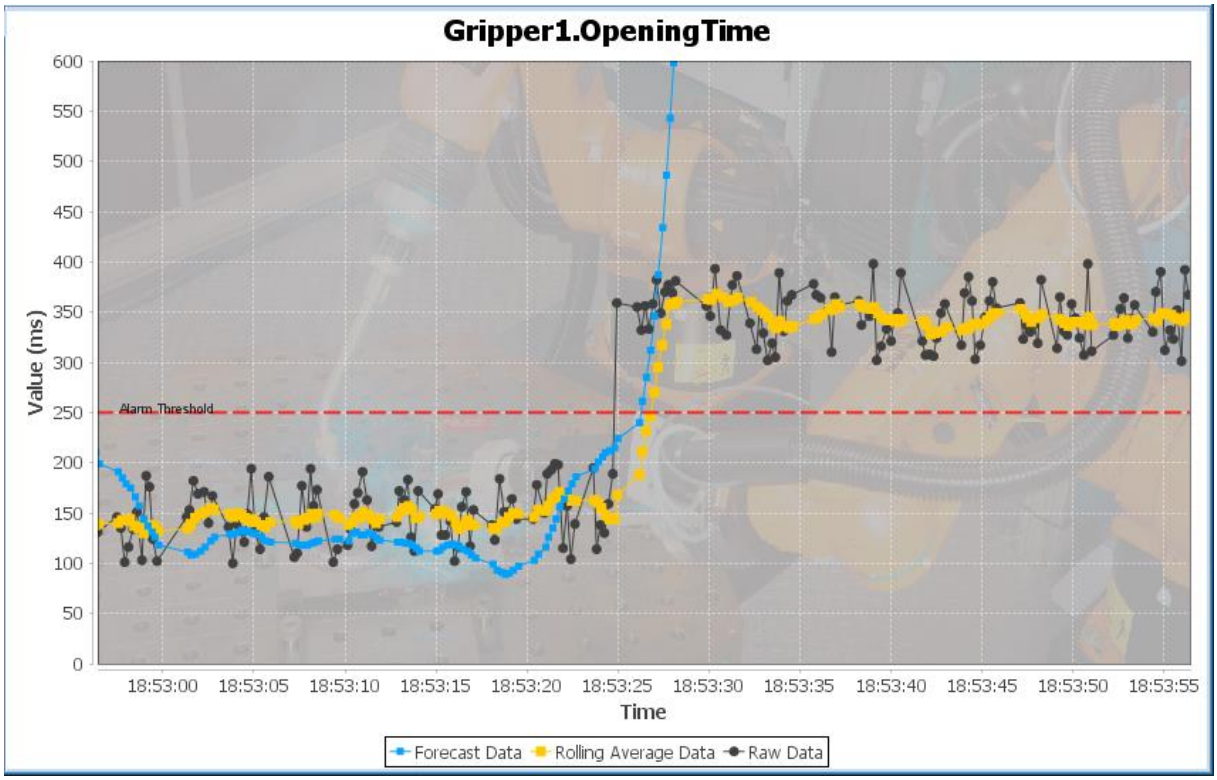

Figure 31 - Simulation Scenario - Spike Failure

As it can be observed, after the time specified in the *Failure Start Delay* parameter has elapsed (600*100ms = 60000ms = 60s) a spike increase in the raw data values is detected, causing the rolling average values to suddenly increase as well.

This in turn causes the forecast values to skyrocket due to the way the least squares model is estimated. Clearly this is not an ideal application scenario since there was no trend in the data that would allow the failure to be predicted. Still, this occurrence can be detected due to the forecast exceeding the alarm threshold, causing an alarm event to be emitted, triggering a possible self-adjustment or maintenance response.

A better suited scenario would be the case where there is an underlying trend in the data being analysed which can be used to anticipate such a failure or machine breakdown event. In order to test this, the incremental failure mode was implemented, as it can be seen in Figure 32.

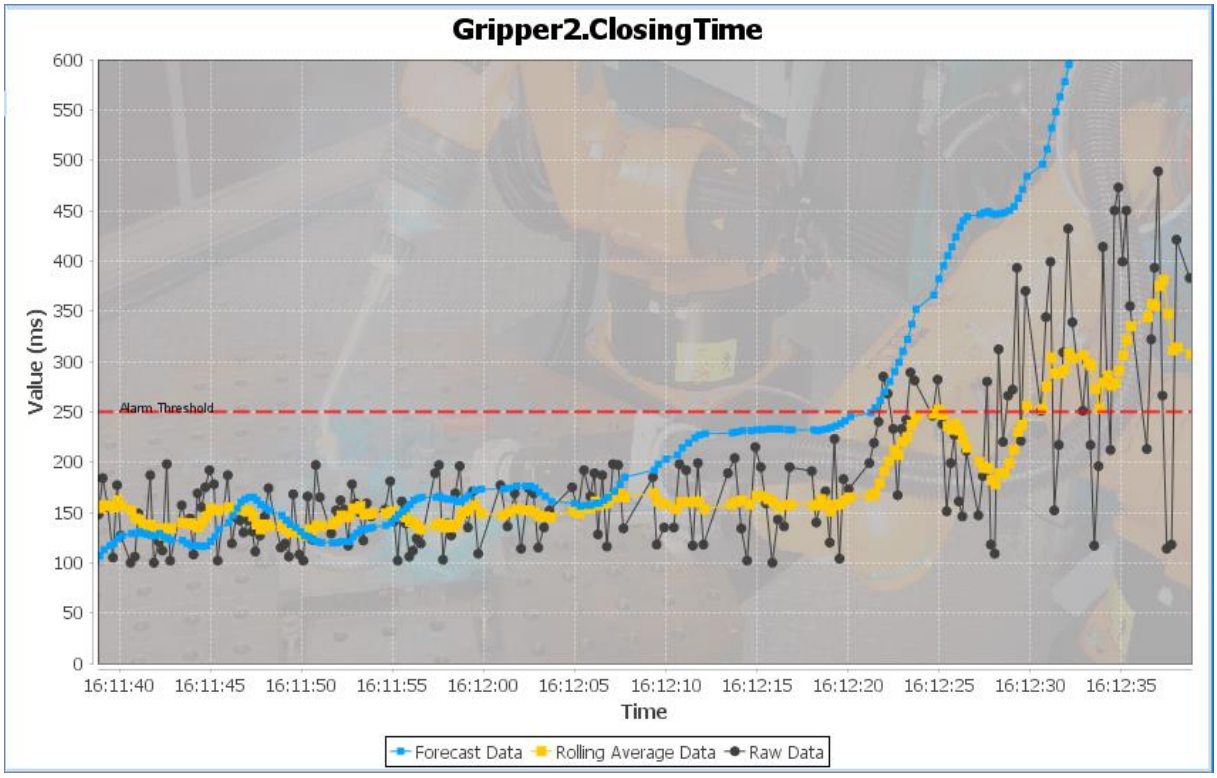

Figure 32 - Simulation Scenario – Incremental Failure

In this case, instead of a sudden spike in the raw data values generated by the *ComponentSimulationSpout*, a gradual increase in the observed value can be detected, thus presenting a rising trend in the data. With this, the *ForecastBolt* can anticipate this behaviour and allow an intervention before things get too out of hand. This bolt can also be adjusted and further refined, both in terms of varying its configuration parameters or even the algorithm itself, allowing for predictions to be either faster or slower to respond to the changes in the raw data values, controlling issues such as false positives or late detections.

6. Conclusion

This document entailed the description of both the architectural design as well as a technical implementation of a real-time data analysis framework developed under the H2020 PERFoRM project, capable of extracting raw real-time data and compute it in order to translate said data into a business advantage for manufacturers. This tackles one of the main challenges being faced nowadays in the industry concerning the large amount of data being generated but unutilized, and is well aligned with the Industry 4.0 vision of factory interconnectivity and the big data concept.

To this end, the document starts by proposing a layered architecture, decomposing the overall complexity of this multidisciplinary challenge into several different levels, each with its own characteristics, requirements and goals, namely:

- The Data Acquisition Layer, which is responsible for extracting raw data from the shop-floor and needs to be able to cope with possible real-time constraints, different communication protocols, scalability and pluggability concerns.
- The Data Queue Layer, charged with the integration of the data acquisition and processing modules, and required to support high throughput and reliability.
- The Data Processing Layer, responsible for detecting trends and correlations in the data, as well forecasting future values, all the while contemplating the issues of performance and scalability.
- The Data Visualization Layer, which facilitates the interpretation and understanding of the data, enabling the integration of the human in the process.

Additionally, a possible implementation of each of the proposed architecture's layers was presented, contemplating several different technologies and the applicability of each of them regarding the requirements imposed by each of the different modules. This implementation was subjected to some preliminary testing under a simulated environment, in which promising results were shown in terms of the early detection and prevention of possible machine failures or breakdowns in different situations, serving as an input to trigger self-adjustment mechanism, thus successfully supporting PERFoRM's aims for seamless reconfigurability and visualization of manufacturing processes.

References

1. Deliverable D1.1, “Report on Decentralized Control & Distributed Manufacturing Operation Systems for Flexible and Reconfigurable Production Environments”, PERFoRM project, 30th March 2016.
2. Deliverable D1.2, “Requirements for Innovative Production System Functional Requirement Analysis and Definition of Strategic Objectives and KPIs”, PERFoRM project, 29th March 2016.
3. Deliverable D2.2, “Definition of the System Architecture”, PERFoRM project, 28th September 2016.
4. Deliverable D7.1, “Siemens Description and Requirements of Architectures for Retrofitting Production Equipment”, PERFoRM project, March 2016.
5. Deliverable D8.1, “Micro Electric Vehicles Description and Requirements of Architectures in View of Flexible Manufacturing”, PERFoRM project, 29th March 2016.
6. Deliverable D9.1, “Description of Requirements and Architecture Design”, PERFoRM project, May 2016.
7. Deliverable D10.1, “Use Case goals/KPIs and Requirements Defined”, PERFoRM project, September 2016.
8. Andre Dionisio Rocha, Ricardo Peres, and Jose Barata. An agent based monitoring architecture for plug and produce based manufacturing systems. In 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pages 1318-1323. IEEE, 2015.
9. Andre Dionisio Rocha, Diogo Barata, Giovanni Di Orio, Tiago Santos, and Jose Barata. Prime as a generic agent based framework to support pluggability and reconfigurability using different technologies. In Doctoral Conference on Computing, Electrical and Industrial Systems, pages 101-110. Springer International Publishing, 2015.
10. Andre Dionisio Rocha, Ricardo Silva Peres, Luis Flores, and Jose Barata. A multiagent based knowledge extraction framework to support plug and produce capabilities in manufacturing monitoring systems. In Mechatronics and its Applications (ISMA), 2015 10th International Symposium on, pages 1-5. IEEE, 2015.
11. S. Finlay. Predictive Analytics, Data Mining and Big Data. Myths, Misconceptions and Methods. Palgrave Macmillan, 1st ed., 2014.
12. J. Han, M. Kamber, J. Pei. Data Mining: Concepts and Techniques. Morgan Kaufmann, 3rd ed., 2011.
13. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. “From Data Mining to Knowledge Discovery in Databases,” AI Magazine, vol.17, n.3, pp. 37-54, 1996.
14. G.E.P. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung. Time series analysis: forecasting and control. John Wiley & Sons, 5th ed., 2015.
15. S. Haykin. Neural Networks and Learning Machines, Pearson, 3rd ed., 2009.
16. N.R. Draper, H. Smith. Applied regression analysis. John Wiley & Sons, 3rd ed., 1998.

17. C. Bishop. Pattern recognition and Machine Learning. Springer, 2006.
18. J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia. "A survey on concept drift adaptation." ACM Computing Surveys (CSUR), vol.46, n.4, p.44, 2014.
19. P. Bobko. Correlation and Regression: Applications for Industrial Organizational Psychology and Management. SAGE Publications, 1st ed., 2001.
20. J. Cohen, P. Cohen, S.G. West, L.S. Aiken. Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences. Routledge, 3rd ed., 2003.
21. J. Aldrich. "Correlations Genuine and Spurious in Pearson and Yule". Statistical Science, vol.10, n.4, pp. 364-376, 1995.
22. J. Lee, B. Bagheri, H.A Kao. "Recent advances and trends of cyber-physical systems and big data analytics in industrial informatics." In International Conference on Industrial Informatics (INDIN). 2014.
23. Keim, Daniel A. "Information visualization and visual data mining." IEEE transactions on Visualization and Computer Graphics 8.1 (2002): 1-8.
24. Bellifemine, F., Caire, G., & Greenwood, D. (2007). Developing Multi-Agent Systems with JADE. Developing Multi-Agent Systems with JADE (pp. 1–286). doi:10.1002/9780470058411
25. Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE–A FIPA-compliant agent framework. Proceedings of PAAM, 97–108. doi:10.1145/375735.376120
26. FIPA. (2002). FIPA Request Interaction Protocol Specification. Retrieved from http://www.fipa.org/specs/fipa00026/SC00026H.pdf
27. Ribeiro, L., Rocha, A., & Barata, J. (2013). A study of JADE's messaging RTT performance using distinct message exchange patterns. IECON Proceedings (Industrial Electronics Conference), 7410–7415. doi:10.1109/IECON.2013.6700366
28. Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB, pages 1-7, 2011.
29. Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. Proceedings of the VLDB Endowment, 8(12):1654-1655, 2015.
30. Ken Goodhope, Joel Koshy, Jay Kreps, Neha Narkhede, Richard Park, Jun Rao, and Victor Yang Ye. Building linkedin's real-time activity data pipeline. IEEE Data Eng. Bull., 35(2):33{45, 2012.