**Ricardo Alexandre Fernandes da Silva Peres**

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

# An Agent Based Architecture to Support Monitoring in Plug and Produce Manufacturing Systems using Knowledge Extraction

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: José António Barata de Oliveira, Professor Doutor, FCT-UNL

Co-orientador: André Dionísio Bettencourt da Silva Rocha, Mestre, UNINOVA/CTS

Júri:

Presidente: Prof. Ricardo Jardim Gonçalves, PhD

Arguente: Prof. Paulo Jorge Pinto Leitão, PhD

Vogal: Prof. José António Barata de Oliveira, PhD

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

**Setembro, 2015**

# An Agent Based Architecture to Support Monitoring in Plug and Produce Manufacturing using Knowledge Extraction

*To my family and everyone who helped me get where I am today*

# Acknowledgments

In this section I would like to take the opportunity to express my gratitude towards everyone who helped me through the process of developing and writing this thesis.

First and foremost I would like to thank my supervisor, Prof. José Barata, for the possibility of developing my thesis in my field of choice, as well as for all the opportunities he has provided me over these last couple of years, particularly my integration in the manufacturing R&D team and the participation in the FP7 PRIME project.

To my co-supervisor, André Rocha, I can only say I will forever be thankful for the vote of confidence he demonstrated when we first started working together and for his tutelage during the development of this work. For all I've learned, his patience, guidance and companionship, not only is he a reference, but also a friend. I can sincerely say it was an honour learning and working with him.

I would also like to thank everyone I had the pleasure of meeting and working with during the development of this thesis. To my lab colleagues, Diogo Barata, Mafalda Parreira Rocha, Pedro Monteiro and Tiago Peralta Santos, thank you for providing a fantastic work environment, for your friendship and for your patience enduring my relentless silliness. To all the incredible professionals I met during PRIME's development, thank you for welcoming me with open arms into the project, it was truly a pleasure and a fantastic learning experience.

To my dear friends, Liliana Sequeira, João Pedro Matos, Diogo Alves and Miguel Raposo, thank you for always being there for me, through thick and thin. Without your friendship, your support and motivation over the years everything would have been different. I feel incredibly lucky for having the privilege of calling you my friends.

And last but not least, I would like to express my infinite gratitude to my family, more specifically my mother and stepfather, for all they have done for me during this journey, all the

sacrifices, the unconditional support and invaluable lessons. None of this would have been possible without you.

# Abstract

In recent years a set of production paradigms were proposed in order to capacitate manufacturers to meet the new market requirements, such as the shift in demand for highly customized products resulting in a shorter product life cycle, rather than the traditional mass production standardized consumables.

These new paradigms advocate solutions capable of facing these requirements, empowering manufacturing systems with a high capacity to adapt along with elevated flexibility and robustness in order to deal with disturbances, like unexpected orders or malfunctions.

Evolvable Production Systems propose a solution based on the usage of modularity and self-organization with a fine granularity level, supporting pluggability and in this way allowing companies to add and/or remove components during execution without any extra re-programming effort.

However, current monitoring software was not designed to fully support these characteristics, being commonly based on centralized SCADA systems, incapable of re-adapting during execution to the unexpected plugging/unplugging of devices nor changes in the entire system's topology.

Considering these aspects, the work developed for this thesis encompasses a fully distributed agent-based architecture, capable of performing knowledge extraction at different levels of abstraction without sacrificing the capacity to add and/or remove monitoring entities, responsible for data extraction and analysis, during runtime.

**Keywords:** *self-monitoring; manufacturing system; multiagent system; plug & produce; evolvable production system*

# Resumo

Nos últimos anos foi proposto um conjunto de paradigmas de produção com o intuito de possibilitar aos fabricantes fazer frente às novas necessidades do mercado, tais como o aumento exponencial na demanda por produtos altamente customizados, resultando em tempos de vida mais curtos para os produtos, contrastando com as políticas mais tradicionais de produção em massa de artigos estandardizados.

Estes novos paradigmas promovem soluções capazes de fazer face a estes requisitos, concedendo aos sistemas de manufatura uma forte capacidade de adaptação, bem como uma elevada flexibilidade e robustez de forma a lidarem com possíveis perturbações inesperadas.

O paradigma de Sistemas Evolutivos de Produção, em particular, propõe uma solução baseada na utilização de modularidade e de características de auto-organização com elevado grau de granularidade, suportando *pluggability* e desta forma permitindo às empresas de manufatura adicionar e/ou remover dispositivos durante a execução do sistema, sem no entanto requerer qualquer esforço adicional de reprogramação.

No entanto, muitos dos sistemas de manufatura atuais não foram desenhados de forma a suportar plenamente estas características, sendo normalmente baseados em sistemas SCADA centralizados, incapazes de se readaptar durante a execução à adição ou remoção de dispositivos ou a alterações à topologia do sistema.

Tendo estes aspetos em consideração, o trabalho desenvolvido no âmbito da presente tese de mestrado encapsula uma arquitetura multiagente de suporte à monitorização completamente distribuída, capaz de executar extração de conhecimento a diferentes níveis de abstração sem para isso sacrificar a capacidade de adicionar e/ou remover entidades de monitorização durante a execução, responsáveis pela extração e processamento de dados em *runtime*.

**Palavras-Chave:** *auto-monitorização; sistema de manufatura; sistema multiagente; plug & produce; sistema evolutivo de produção*

# Table of Contents

# Table of Figures

# Table of Tables

# Acronyms

| | |
|---|---|
| **ACT** | Action Time |
| **ANN** | Artificial Neural Networks |
| **CA** | Component Agent |
| **CMA** | Component Monitoring Agent |
| **CSK** | Complex Skill |
| **CST** | Computed State |
| **DA** | Deployment Agent |
| **DAL** | Data Acquisition Library |
| **DB** | Database |
| **DCL** | Data Collection Library |
| **DEP** | Data Extraction Process |
| **DOL** | Data Output Library |
| **DPA** | Data Processing Algorithm |
| **EDL** | Event Description Library |
| **GA** | Genetic Algorithm |
| **HDL** | Hardware Detection Library |

| | |
|---|---|
| **HLCMA** | Higher-Level Component Monitoring Agent |
| **HMI** | Human-Machine Interface |
| **HMIA** | Human Machine Interface Agent |
| **IntRoSys** | Integration for Robotic Systems |
| **KBANN** | Knowledge-Based Artificial Neural Networks |
| **MAS** | MultiAgent System |
| **MDD** | Monitoring Data Description |
| **OCA** | Output Coordinator Agent |
| **P&P** | Plug and Produce |
| **PA** | Product Agent |
| **PD** | Part Detector |
| **PLC** | Programmable Logical Controller |
| **PMA** | Production Management Agent |
| **PRIME** | Plug and produce Intelligent Multiagent Environment |
| **PSA** | PRIME System Agent |
| **RST** | Raw State |
| **SCADA** | Supervisory Control and Data Acquisition |
| **SMA** | Skill Management Agent |
| **SSK** | Simple Skill |
| **TRT** | Transition Time |
| **WG** | Workgroup |

# 1

# Introduction

## 1.1. Background

Over the past few decades manufacturing has undergone several profound changes, with new market trends obligating manufacturers to steadily shift from the popular mass production concept to highly dynamic and flexible production lines.

Due to the increasing market competitiveness and the growing demand for highly customized products, with many variants and fluctuating demand for each one, companies are required to quickly adapt and adjust to new business opportunities in order to survive.

As a direct consequence, manufacturing systems are required to be more and more agile in order for manufacturers to thrive and prosper in such a competitive environment riddled with unpredictable changes, empowering them to rapidly and effectively react to the changing markets driven by the increasing demand for customization.

Motivated by the inadequacy of traditional systems to fully correspond to these new requirements, several manufacturing paradigms were proposed advocating agility, flexibility modularity and reconfigurability, enabling concepts such as Plug & Produce (P&P), which refers to the capacity of modular systems to add and remove components during execution without further significant re-programming effort and without stopping the system.

These emergent paradigms originated several new approaches in the manufacturing field, however, despite their crucial role in preventing unscheduled shutdowns, reducing production loss and decreasing the equipment's maintenance frequency, monitoring systems

did not fully accompany this transition. Many current monitoring applications are still based on centralized, rigid solutions, being unable to support these emergent concepts such as P&P.

Sections 1.2 and 1.3 describe the research problems motivated by this scenario and the proposed solutions to tackle them, respectively.

## 1.2. Research Problems and Hypotheses

Taking into account the challenges that arise in the manufacturing scenario previously described in section 1.1, it should be clear that monitoring is a crucial part of any modern manufacturing system. As such, a few research problems can be posed, namely:

- Would it be possible to perform monitoring in a modular manufacturing system possessing P&P capabilities, whilst supporting those same characteristics by means of a distributed approach?
- How can knowledge extraction be performed so that more complex knowledge can be derived from the combination of raw data obtained in the system's monitoring process?

For the first research problem it is proposed, as a hypothesized solution, the development of a multiagent-based monitoring architecture, due to the agent's innate modular, cooperative and distributed nature. These agents should be capable of extracting data from the entity they are abstracting, whilst supporting the addition/removal of said entity from the system.

Concerning the second research problem, it is proposed that the agents comprising the monitoring architecture should possess a certain degree of reasoning capabilities, allowing them to infer more complex, higher-level knowledge from the data collected in the regular monitoring process, based upon a given set of pre-established rules.

## 1.3. Thesis Outline

Having identified the research problems, it is possible to hypothesize a solution. To this end a distributed multiagent monitoring architecture is proposed, capable of performing knowledge extraction at different levels of abstraction.

The proposed architecture comprises three different types of mechatronic agents, more specifically those responsible for abstracting physical devices, collecting and pre-processing relevant data, those performing a similar task at the subsystem level, which refers

to an agglomerate of devices operating in conjunction with one another, and finally those responsible for acting as a data gateway to external entities. These entities possess only a partial awareness of the whole system, with corresponding different levels of granularity associated. For this reason full system monitoring emerges from the cooperation of all these different entities forming a monitoring tree of different layers of abstraction.

Whenever a monitoring agent is launched it should consult an external data file containing the raw data description pertaining to its associated device or subsystem, as well as a set of rules regarding the composition of higher-level data. Basing itself on these rules, each monitoring agent should be capable of inferring more complex knowledge using previously extracted data as its building blocks.

This approach aims to provide a technology independent middleware for knowledge extraction to support manufacturing systems at the data acquisition and processing stages. To enable this technology independence, several generic interfaces were defined to allow agents to collect and relay monitored data without changing their behavioural logic, maximizing compatibility when applying this solution to already established manufacturing systems without requiring any structural modifications by the manufacturers.

The proposed architecture should also be able to support the addition and removal of monitored devices during the system's execution. For this purpose a fourth entity is responsible for constantly looking for changes in the system's topology, managing the corresponding agents accordingly and assuring that the associative relations between them are maintained congruently. Upon detecting the plugging of a new device, the architecture should enact a self-organized response by having this entity launch a new monitoring agent to abstract the recently connected component, associating it to the correct higher-level entity in the appropriate layer of the monitoring tree. Similarly, once a device is unplugged the system is capable of detecting this occurrence, organizing itself accordingly by removing the corresponding agent from the tree.

Once the architecture's design was finalized it was implemented employing the Java Agent DEvelopment Framework (JADE). This implementation was validated simultaneously in two different industrial robotic cells executing the welding of a car's side-member.

## 1.4. Main Contributions

The architecture designed and implemented as part of this work presents a validated solution to enable the monitoring of systems displaying modular and distributed characteristics, along with P&P capabilities, whilst supporting those same defining factors itself.

As such, this distributed multiagent-based approach consists in a valid solution to enable data extraction and analysis in even real industrial systems based on emergent manufacturing paradigms such as Reconfigurable Manufacturing Systems (RMS) and Evolvable Production Systems (EPS).

The implementation of the proposed architecture was integrated in the FP7 PRIME project, conferring its existing architecture the capacity to perform knowledge extraction without sacrificing its self-reconfiguration and P&P capabilities.

An article was written based on the work documented in this thesis, submitted and presented at the INDIN 2015 IEEE International Conference on Informatics (A. D. Rocha, Peres, & Barata, 2015).

# 2

# State-Of-The-Art

During the past few decades the manufacturing industry has undergone severe changes, mostly due to the shift in the consumers' mentality, demanding up-to-date, more varied and customisable products. This recent change in the customer's mind-set contrasts heavily with the standardized, streamlined mass production assembly lines popularized in the early twentieth century, which allowed companies to produce large amounts of non-diversified products at a relatively low production cost (Ribeiro & Barata, 2011).

This fact forced a certain paradigm shift for the manufacturers, moving away from the standardized mass production concepts to more agile and flexible systems capable of keeping up with the demand for more customised products with shorter life-cycles (Bolwijn & Kumpe, 1990).

Agile manufacturing, as introduced by Nagel and Dove (Nagel & Dove, 1991), refers to the capacity to thrive and prosper in the face of constant and unpredictable changes by being able to quickly adapt to these new trends, whilst flexibility is connected to the ability to produce different products in an efficient manner through the reconfiguration of the manufacturing system.

All this has inspired varied research efforts, mainly focusing in coming up with diverse solutions to meet the new industry requirements, leading to the emergence of different manufacturing paradigms, namely the Flexible Manufacturing Systems (FMS), Holonic Manufacturing Systems (HMS), Bionic Manufacturing Systems (BMS), Reconfigurable Manufacturing Systems (RMS) and Evolvable Production Systems (EPS).

## 2.1. Emergent Manufacturing Paradigms

Traditionally, conventional manufacturing architectures rely entirely on a fully centralized control system, in which a central entity completely governs the decision making process. While hierarchical approaches can potentially introduce optimizations at the control system level at the cost of having massive processing entities, the associated processing time greatly increases as the system's structure and size grow, sacrificing other relevant performance indicators, such as the system's adaptability, responsiveness and agility (Barbosa, 2015).

However, more recently the continued increase in the demand for customized products, often to the extreme (a different product for each customer), which are in turn getting more and more complex and varied in regards to their application domain, has translated into shorter changeover times and product life cycles, moving further and further away from the idea of standardized mass production, towards mass customization instead (Nagorny, Colombo, & Schmidtmann, 2012).

Due to this, the emerging market requirements cannot be met by simply using the conventional manufacturing structures and systems, generally based on hierarchical architectures with centralized decision-making, accompanied by hard-connected, non-interchangeable layouts.

These new business forms, reliant on a desire for a strong collaboration between suppliers and customers, impose further challenges to the shop floor, making older approaches unsuitable for this new reality (Frei, Barata, & Onori, 2007).

All of this has culminated in the emergence of several different manufacturing paradigms, in an attempt to meet these new requirements for flexibility, agility and reconfigurability. These paradigms will be discussed in the following subsections.

### 2.1.1. Flexible Manufacturing Systems

As the name suggests, Flexible Manufacturing Systems (FMS) emerged due to the manufacturers' necessity of being able to cope with the ever-increasing need for product customisation and variety. This paradigm enables the production of different types of products, with changeable volume and mix, on the same system.

However, when compared with dedicated manufacturing lines, which are based on relatively inexpensive fixed automation and produce a company's core products or parts at high volume, FMS usually present a lower throughput capacity which translates into higher costs per part (Koren, 2006).

Since the paradigm's appearance, extensive work has been done in the FMS field, covering varied application fields. In (Jahromi & Tavakkoli-Moghaddam, 2012), regarding the problem of dynamic machine-tool selection and operation allocation in a FMS environment, a programming model is proposed with the objective of determining a machine-tool combination for each operation that minimizes some productions costs, namely machining costs, setup costs, material handling costs and tool movement costs. Some efforts have also focused the monitoring field, such as the work described in (Ouelhadj, Hanachi, & Bouzouia, 2000), which will be discussed further in section 2.3.

Regardless, the current FMS approaches used in the shop floor are not well suited to deal with turbulence. Particularly, one important limitation is their inability to deal with the evolution of the production system life cycle (Ribeiro & Barata, 2011), leading to the emergence of new paradigms.

### 2.1.2. Holonic Manufacturing Systems

Holonic Manufacturing Systems (HMS) are based on the term *holon*, coined by Arthur Koestler circa 1967, which derives from the combination of the Greek word "holos", meaning "whole", and the suffix "on", referring to "part", used to describe both living organisms and social organizations (Koestler, 1967).

A holarchy, a term also defined by Koestler, consists in a hierarchically organized system of self-regulating, autonomous and cooperative holons working towards common goals. These holons can be composed of smaller organisms whilst being simultaneously part of the bigger entity, hence their name. Therefore, a holon acts as the hybrid basic organizational unit of these systems. An HMS results from the encapsulation of the entire manufacturing system in a holarchy (Barbosa, 2015).

When applied to a manufacturing context, the holonic concept results in a distributed system comprising several subsystems, each exhibiting holonic behaviour offering higher degrees of adaptability and scalability when compared to the current more commonly used approaches.

A few solutions following the HMS paradigm have been proposed in the existing literature. An example, the PROSA reference architecture (Van Brussel, Wyns, Valckenaers, Bongaerts, & Peeters, 1998), defines the basic holons that a manufacturing system must have as well as their core interactions. A more recent example is the *ADAptive holonic COntrol aRchitecture* (ADACOR) (Leitão & Restivo, 2006), which addresses the improvement of performance in an industrial setting characterised by the frequent occurrence of unexpected disturbances at the shop floor level. The proposed approach provides an adaptive production control, enabled by the existence of a supervisor holon and the self-organizational capabilities of each ADACOR holon, allowing to balance production control between two different states, stationary, which presents a hierarchical control

structure, and transient, presenting a heterarchical control structure), fusing production optimization and agile reaction to disturbances.

Several documented HMS-based implementations are developed using multi-agent-based approaches, which are discussed in Section 2.2.

### 2.1.3. Bionic Manufacturing Systems

Similarly to the HMS paradigm, Bionic Manufacturing Systems (BMS) are inspired in living organisms, with a greater focus in natural organs (Ueda, 1992), sharing some of the same base concepts, namely complexity encapsulation, self-organizing behaviours and decentralisation. The basic concept is centred in a hierarchy where the whole organism is composed different interacting organs, dynamically exchanging data similar to DNA enable the system as a whole to enact a self-organising response. This information exchange allows the organisms to evolve from generation to generation as time passes, in a similar fashion to the naturally occurring evolution of the species in nature.

When applied to manufacturing, this paradigm aims to provide solutions to deal with unpredictable changes in the production environment based on bio-inspired behaviours such as self-adaptation, self-organisation and the capacity to evolve as previously stated.

### 2.1.4. Reconfigurable Manufacturing Systems

The emergence of the Reconfigurable Manufacturing Systems (RMS) paradigm originated from the need to cope with constant, fast changes in the production environment, where factors such as the capacity to quickly reconfigure and integrate system components are vital. RMSs can be placed somewhere between FMSs and Dedicated Manufacturing Systems (DMS), providing customized flexibility through scalability and reconfiguration as demanded by the market requirements, as opposed to FMSs which provide general flexibility via machines that are able to operate on a random sequence of parts or products of varied types, with little to no time required for changeover (ElMaraghy, 2006).

One of the main purposes of RMSs is to reduce the *ramp-up time* in manufacturing systems. *Ramp-up time* can be defined as the timespan between the planning and development of a new product and the point in time where sustainable production levels are achieved.

The RMS paradigm advocates that reconfigurability and flexibility should be achieved through open reconfigurable control and dedicated inter-modular tools, instead of optimised controls

and multipurpose tools. With this, the defining characteristics of RMSs are modularity, integrability, customization, diagnosability and scalability (Ribeiro & Barata, 2011).

RMSs thereby set the paving stones needed to support dynamic shop-floors capable of dealing with new market demands, quickly adapting to new products through modules that can be easily switched and reconfigured depending on the production requirements, the integration of different technologies and variances in the demand.

### 2.1.5. Evolvable Production Systems

To face the current manufacturing challenges imposed by the increasing demand for product customisation, a new paradigm, Evolvable Production Systems (EPS), has been proposed as a possible solution for manufacturing systems to deal with changes in production and disturbances.

EPSs share many characteristics with the aforementioned HMS, BMS and the RMS paradigms, to the point where they can sometimes be confused. The modularity, self-organization and reconfiguration concepts are all also part of the EPS paradigm, however, the key difference which boosts agility in EPS lies in the granularity level. Granularity defines the functional complexity of the components which compose a manufacturing system. Taking as an example a line composed of several modular cells that can be plugged and unplugged, this refers to coarse granularity. However, if the components themselves (i.e. grippers, sensors, cylinders) can be plugged in or out, then it refers to fine granularity (Onori, Lohse, Barata, & Hanisch, 2012). Modules are abstracted as process specific entities rather than function specific, as it happens in RMS for instance (Ribeiro & Barata, 2011). This characteristic is extremely important when distinguishing the different paradigms, since only EPSs can achieve fine granularity if needed.

The main characteristics that need to be taken into account when designing an EPS are (Barata & Onori, 2006):

- Modularity: One of the characteristics shared with the other previously mentioned paradigms, the existence of independent modules is required in an EPS.
- Pluggability: The plug and produce (P&P) concept is related to the capacity of introducing/removing modules without the need to stop the system and with minimal reprogramming effort.
- Granularity: Different levels of granularity should be allowed, as in a module can represent a robot, while another one might be related to an entire safety group.
- Reconfigurability: The system must be able to cope with physical changes in the production layout or its control requirements.

These characteristics, modularity and reconfigurability in special, are also enablers of the Plug and Produce (P&P) concept. As demonstrated in (A. Rocha et al., 2015), changes in the production environment have an heavy impact in its control and supervision systems regarding reprogramming effort. With this in mind, the P&P concept was introduced as a methodology to easily connect/disconnect a component or device into/from a manufacturing system without the need for further reprogramming tasks and most importantly without stopping the production line (Arai, Aiyama, Maeda, Sugi, & Ota, 2000).

This paradigm follows two fundamental principles. The first one states that the most innovative product design can only be achieved if there are no constraints imposed in the assembly process, allowing the process selection to result in an optimal assembly system methodology. The second principle is related to the system's evolution and states that systems under dynamic condition need to have an inherent capability of evolution to address a new or changing set of requirements (Barata, Santana, & Onori, 2006).

Several projects have been developed involving the EPS paradigm. The FP6 IST EUPASS project focused on the research of affordable, cost effective ultra-precision manufacturing solutions by offering quickly deployable assembly services on demand. From the progress made with EUPASS resulted the FP7 NMP IDEAS project, which focused on implementing mechatronic agent technology into industrial controllers, with the purpose of assessing and validating its performance at the device level (Onori et al., 2012). Both of these projects involved multi-agent-based implementations, which will be discussed in section 2.2.

## 2.2. Multi-Agent Systems

Computer Integrated Manufacturing (CIM) has been in a constant state of evolution, and lately manufacturers have started to move away from the rigid hierarchical control architectures, along with their exponentially increasing complexity (Monostori, Váncza, & Kumara, 2006), towards distributed, scalable and more robust approaches.

It is in this context that Multi-Agent Systems (MAS) can appear as a viable solution to solve complex decision-making problems and provide the versatility, scalability and interoperability required to put the emerging manufacturing paradigms into practice, in which agents can act as cyber-physical entities that can abstract both physical and logical components of a given manufacturing system.

Many different definitions for the concept of *agent* have been proposed over the years (Wooldridge, 2002)(Russel & Norvig, 2003), however a few common characteristics can be found

among them, more specifically the notion that an agent acts autonomously, possessing a partial knowledge of its surrounding dynamic environment and interacting with other agents to fulfil certain common goals.

This concept can also be associated with intelligence, proactiveness, mobility and adaptability, as suggested in (Monostori et al., 2006). An agent should possess intelligence, based on a set of reasoning and even learning rules, in order for it to be able to act and influence its surrounding environment. Proactiveness allows the agent to actively seek out its goals without any interference from external entities, basing its decisions purely on its reasoning, its environment and the interactions with other agents in the same community. Since each agent only has a partial knowledge of the environment that surrounds it, the system's goals can only be achieved with the cooperation between the agents that are comprised by a given MAS.

Adapting MAS technology to the manufacturing industry can be however fairly complicated, as mentioned by (Marik & McFarlane, 2005). There is a wide array of different application levels, each with different associated requirements concerning response times and requiring fairly different MAS approaches. As defined in the ISA-95 standard (Alexakos et al., 2006), these response times can range from tenths of milliseconds at the real-time execution control level, passing by the slightly less critical Supervisory Control and Data Acquisition (SCADA) and Manufacturing Execution System (MES) layers, to the less time sensitive logistics and planning level, where response times could go up to days, or in extreme cases to weeks or months.

Regardless, as mentioned in the previous sections, MAS-based approaches have been widely used to implement solutions based on the previously mentioned emergent paradigms. Due to their distributed and cooperative nature, MAS are well-suited to enable the self-capabilities and flexibility associated with those paradigms.

There are several examples of MAS-based implementations in recent projects and research efforts. ADACOR, mentioned in section 2.1.2, was implemented with the use of MAS technology through the Java Agent Development framework (JADE), taking advantage of the agent's intelligent and cooperative behaviour, along with the modularity and decentralisation associated with software agents (Leitão & Restivo, 2006). As such, ADACOR holons are Java classes that extend the agent class provided by the JADE framework, simply extended with characteristics that represent the specific behaviour of each ADACOR holon class.

Both the EUPASS and the IDEAS projects were also based on MAS architectures. The IDEAS project in particular makes use of the MAS technology to enable P&P at the shop floor level, through mechatronic agents that abstract manufacturing components and can be plugged or

unplugged to create systems, without any extra re-programming effort being required (Onori et al., 2012).

## 2.3. Monitoring in Manufacturing

Manufacturing systems need to cope with ever-changing production environments where sudden failures can significantly influence their performance. An early detection of these failures, or better yet the detection of potential failures using automated monitoring can help preventing unscheduled shutdowns, reducing production loss and decreasing the equipment's maintenance frequency (Bunch et al., 2004).

Monitoring can be viewed as a way to provide systems with the capability to collect, contextualize and process data. This allows them to integrate more advanced e-maintenance technologies, leading to an improvement in product quality and production efficiency (Jianbo Yu, Xi, & Zhou, 2008).

Nowadays, manufacturing has become an increasingly complex domain, where the diversity of systems and applications in the industrial environment has originated the need for a reference model dividing this complexity into four different layers (Alexakos et al., 2006)(Nagorny et al., 2012), the first two being related to the system control, the third with operations and the fourth with the business domain.

The processes of data extraction and monitoring are particularly related to layers two (SCADA) and three (MES). In a manufacturing context SCADA usually refers to a centralized system capable of gathering, analyzing and monitoring real time data, responsible for controlling and monitoring plants or equipment, being present in most nations' critical infrastructures, in turn making SCADA a critical information system (Abbas, 2014). As such, it also faces the same challenges of other information system types such as dynamicity, reliability, robustness, complexity handling and flexibility. The MES layer on the other hand comprises scheduling, quality management and maintenance activities in order to monitor and aid the production process executed at the lower levels.

Despite the fact that SCADA systems have been evolving over the years, moving from monolithic architectural styles to more distributed, networked approaches, it is becoming more and more challenging for SCADA applications to cope with the steady growth in size, complexity and level of uncertainty of the systems they are meant to be applied to. As a direct consequence, MAS have been mentioned as a possible step-forward in the evolution of SCADA systems in order to enable them to cope with this new challenges (Abbas, 2014) and enabling an easier implementation

of modular monitoring, bringing monitoring systems a step closer to the ideas advocated in the emergent manufacturing paradigms.

Still, regardless of the advancements achieved in recent years concerning the new emerging manufacturing paradigms and their respective aforementioned implementations in the control and reconfiguration areas, according to current literature most monitoring systems seem to be lagging behind, being still based on rigid structures, addressing only specific problems and lacking the flexibility and intelligence characteristic of these new paradigms (Merdan, Vallee, Lepuschitz, & Zoitl, 2011).

Nevertheless, various different monitoring approaches have been proposed, based in a wide array of different technologies. In (Hou, Liu, & Lin, 2003) an integrated approach of neural networks and rough sets is proposed, while (Jianbo Yu et al., 2008) suggest the use of an hybrid learning-based model for intelligent monitoring and diagnosis, through the combination of a Knowledge-Based Artificial Neural Network (KBANN), responsible for the monitoring process, and a genetic algorithm (GA) capable of discovering the causal relationship between manufacturing parameters and product quality. However, most algorithms involving ANN-based approaches can quickly become extremely intensive from a computational standpoint (Jian-bo Yu & Xi, 2009), becoming therefore unsuitable for real-world manufacturing applications.

Some MAS-based distributed solutions have also been proposed. In (Ouelhadj et al., 2000) a MAS architecture is proposed for distributed monitoring in FMS. This architecture functions under the principle that each resource in the FMS has an associated computer, where the associated Resource Monitoring Agent is running. Global monitoring is achieved through the cooperation each singular monitoring agent, achieved via the exchange of appropriate messages between them. However, despite being a distributed MAS solution, showcasing the importance of a decentralized monitoring approach done at the component level, it is reliant on each resource having an associated computer and does not display the required pluggability to enable the P&P characteristics existent in some of the emergent paradigms. In addition, no actual results are shown other than a brief mention of some tests performed in a simulated environment, being limited to the theoretical description of the proposed monitoring approach.

## 2.4. Integrated Discussion

As it is possible to conclude from the analysis of the present chapter, the continually increasing demand for highly customizable products, result in shorter life-cycles, has spurred manufacturers into looking at new possible solutions to answer the changing market requirements of

more agile, flexible and robust manufacturing systems, moving away from the standardized mass production, focusing instead on mass customization.

As a result, several new manufacturing paradigms emerged from varied research efforts, namely the FMS, HMS, BMS, RMS and EPS paradigms, which despite presenting promising capabilities, are still met with a certain resistance due to the manufacturer's traditionally reserved mindset in the face of technological innovations that imply a large reformulation of their production lines. Furthermore, these concepts are fairly easy to confuse with one another, considering that several characteristics are shared between them.

These emergent paradigms inspired a considerable amount of work, that once associated with the rising of MAS technologies bore fruit to varied research projects such as the HMS based ADACOR architecture and the EPS focused FP6 EUPASS and FP7 IDEAS, which contributed significantly by showcasing actual implementations of the theoretical principles advocated by these paradigms.

Simultaneously, the increasing complexity of the manufacturing systems and the need to cope with rapidly changing production environments showcased the importance of a good monitoring approach, where an early detection of potential failures using automated monitoring can help preventing unscheduled shutdowns and reducing some production costs.

Although some efforts were made in regards to developing new convincing monitoring approaches, some even following distributed MAS solutions, these pale in comparison to the advancements made in the control and reconfiguration domains, whilst displaying a severe lack of results in the current literature, being generally limited to propositions of theoretical architectures.

For this reason, it can be considered that there is a good research opportunity regarding the development of a truly modular, distributed and robust monitoring solution for data acquisition and knowledge inference, able to display the P&P capabilities, agility and flexibility that characterize the latest advancements made regarding the emergent manufacturing paradigms.

# 3

# Architecture

As previously discussed in Chapter 2, lately there has been an increasing effort to develop MAS based solutions as a way to deal with the emerging industry requirements for agility, flexibility and robustness. However, these efforts have been focused mainly in the control and reconfiguration of the productions systems, leaving the monitoring aspects in the side-lines.

With this in mind, the proposed monitoring architecture described in the present chapter aims to take advantage of some of the main characteristics normally associated with MAS approaches, such as their Plug & Produce (P&P) capabilities and high tolerance in the face of disturbances and unexpected changes, applying them at the Supervisory Control and Data Acquisition (SCADA) and Manufacturing Execution Systems (MES) levels. More specifically, the proposed architecture was designed to function mainly in the data acquisition and pre-processing stages of the monitoring process, acting as a middleware between the hardware layer and the higher-level data processing and storage entities.

The proposed architecture was also integrated in the FP7 Plug and produce Intelligent Multiagent Environment (PRIME) project. As such, even though the work presented in this documented is focused mainly in the aspects related to data monitoring, an overview of PRIME's system as a whole is also provided in this chapter. For that purpose key supporting concepts such as configurations, events and timespans are also presented. These concepts are useful to better understand not only the interactions between agents but also their respective tasks and goals.

## 3.1. PRIME Architecture Overview

As its name suggests, FP7 PRIME is a project which aims to provide a multiagent environment for the creation of new solutions regarding highly adaptive, self-aware, self-monitored, reconfigurable P&P systems.

In its core PRIME is based upon a distributed, highly-scalable multiagent architecture comprised of different generic entities. Each entity plays a distinct role in order for the system to provide all the capabilities described above, such as the capacity to easily reconfigure associated components or to extract and monitor data directly from the shop-floor level.

An overview of the generic agents that support PRIME's architecture can be seen in Figure 3.1.

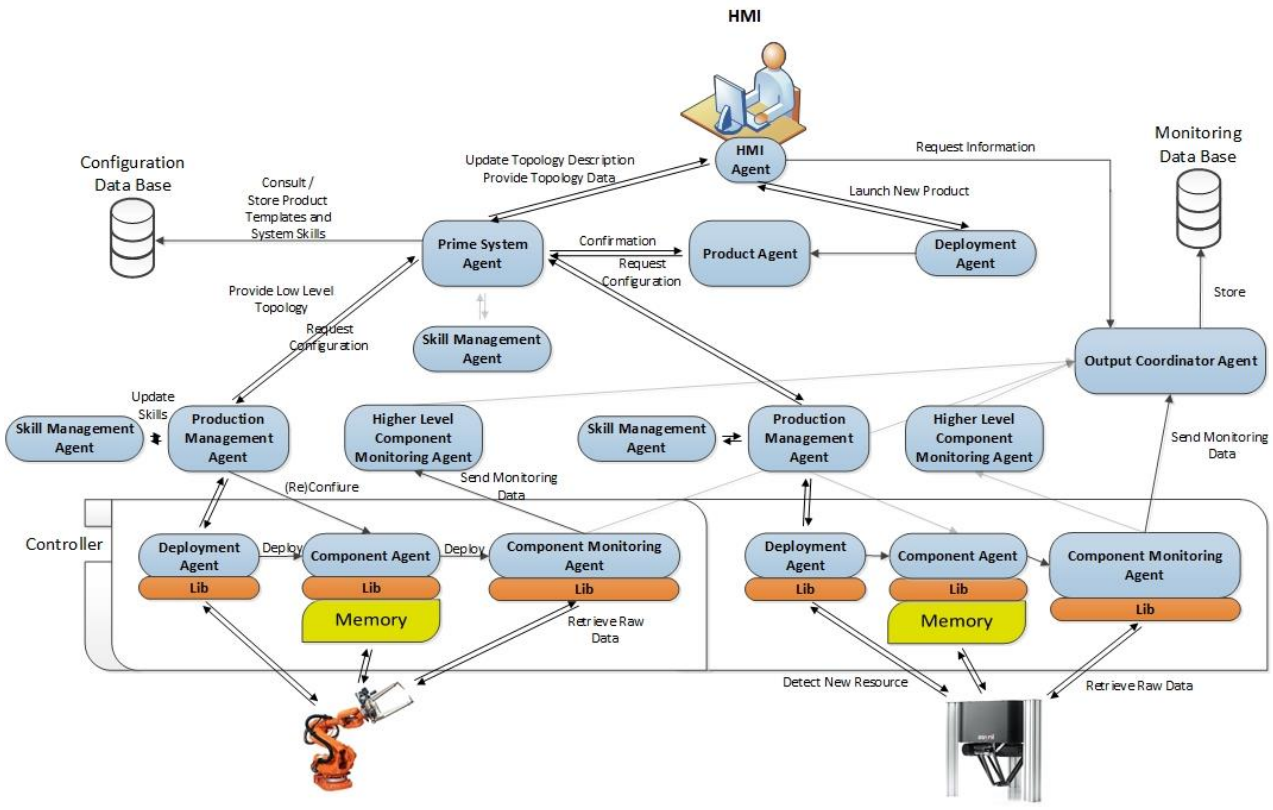


**Figure 3.1 - PRIME Architecture Overview**

As seen above, this architecture is composed by a total of ten different types of agents that support its varied capabilities. For this reason, these agents can be divided into groups according to their respective role in the architecture. A brief description of each agent type can be found in Table 3.1.

**Table 3.1 - PRIME Agents and their Descriptions**

| Role | Name | Description |
|---|---|---|
| Support | Human Machine Interface Agent (HMIA) | The HMIA acts as the bridge between the user and the architecture by communicating directly with the Human Machine Interface (HMI). This agent allows the HMI to acquire and display relevant information, such as the production requirements for a certain product, by providing it with a variety of services that can be called to trigger the necessary behaviours in the remaining agents that will provide said information. It can also initiate the process of launching a new Product Agent, hence triggering the reconfiguration of the system for a new product variant. |
| | Deployment Agent (DA) | The DA is responsible for launching other PRIME agents, therefore playing a vital role in both the reconfiguration and monitoring aspects of the architecture. Consequently it is mandatory that each machine intended to run PRIME agents has one DA running in it prior to their launch. |
| Execution / Reconfiguration | PRIME System Agent (PSA) | This agent is the highest-level entity in the system's hierarchy. It is mainly responsible for managing the semantic model and the repository which contains information pertaining to the entire system, having therefore a large amount of information flowing through it. |
| | Production Management Agent (PMA) | Each PMA is responsible for managing a subsystem, detaining all the information concerning its topology and skills. Its existence is one of the key factors that allows a system to be modulated as a tree of agents with different layers of complexity, improving the system's scalability and modularity. |
| | Skill Management Agent (SMA) | SMAs are the entities in control of the skill generation process which are always linked in a one-to-one relationship to a PMA. Following a pre-determined set of rules, they are capable of verifying the possibility of creating higher-level skills from the pool of already existing ones, informing their respective PMA of any new skills it may be capable of offering. |
| | Component Agent (CA) | A CA is a low-level entity which abstracts a physical resource of the system (e.g. a gripper or a robot), encapsulating all the information related to that device. It is upon the CA that falls the responsibility of reconfiguring its associated component. |
| | Product Agent (PA) | The PA is the agent responsible for abstracting a given product variant, being responsible for initiating the reconfiguration process. As such, it encapsulates all the information required to fully describe the associated product, including its production plan along with all the skills required for its execution. |

| Role | Name | Description |
|---|---|---|
| Monitoring | Component Monitoring Agent (CMA) | Within the monitoring module, the CMA is the entity at the lowest level of the architecture. Each CMA has an associated CA, being therefore responsible for all the monitoring activities related to that agent's associated component. More specifically, a CMA periodically collects raw data from a physical device and pre-processes it according to certain rules, extracting relevant information related to its performance, such as Transition Times (TRT) and Action Times (ACT), finally sending it up the monitoring tree. |
| | Higher-Level Component Monitoring Agent (HLCMA) | Applying the concept of PMA to the monitoring architecture, the HLCMA is one of the key elements that enables the existence of the monitoring tree. As the name indicates, its purpose is similar to the CMA's, simply transposed to a higher level of abstraction. Being linked to a PMA, the HLCMA can receive not only raw data related to the associated subsystem from a computational device, e.g. Programmable Logic Controllers (PLC), but also pre-processed data from other CMAs or HLCMAs monitoring cooperating entities of that subsystem. |
| | Output Coordinator Agent (OCA) | OCAs are the highest-level entities of the monitoring architecture. Working as a cloud of agents sharing the same purpose, they act as a bridge between the monitoring environment and the other external entities, such as historical repositories and data processing networks capable of computing large amounts of data. For this reason, all the data collected and pre-processed by the CMAs and HLCMAs is sent to one of the available OCAs in the cloud in order for it to be sent to relevant external entities. |

As mentioned in the beginning of this chapter, a few key concepts should be defined to allow a better understanding of the presented architecture. These concepts can be divided in two main groups, the first being those that are related to the reconfiguration part of the architecture and the latter consisting on the concepts that belong to the monitoring domain, which is the main focus of this document. Both groups are described below, however, the aspects related to the agents responsible for data extraction and pre-processing will be explained in further detail afterwards.

## 3.2. Supporting Concepts for Reconfiguration

Despite being outside the scope of the work developed for this thesis, it is still important to mention PRIME's Reconfiguration environment, considering that it constitutes one of PRIME's most defining characteristics. The multiagent-based reconfiguration environment enables PRIME to describe and reconfigure manufacturing systems without the need to physically modify already existing structures, making these systems more dynamic and robust. Some concepts supporting the reconfiguration approach are described in this section.

### 3.2.1. Skill

In accordance to the definition provided in (Orio, Rocha, Ribeiro, & Barata, 2015), skill is in essence something that can be executed by a certain component or subsystem (group of components functioning as a whole), encapsulating its capabilities.

Considering the amount of different variations of skills that can exist, it is important for these skills to provide enough information in their specifications for them to be defined regardless of the entity that offers them. Considering this, a skill should have at the very least a certain amount of associated information, namely a unique ID, a name, a brief description and a list of parameters to control its execution.

Skills can be further classified as:

- Simple Skills (SSK) – These are atomic capabilities provided by an entity which may or may not match a process.
- Complex Skills (CSK) – At a higher level of abstraction, whenever an SSK is not enough to execute a certain process by itself, a new CSK should be created (if possible) by combining two or more available skills following a pre-determined set of rules, in order to allow the execution of said process by enabling more complex functionality (Antzoulatos et al., 2015).

### 3.2.2. Configuration

A configuration is fundamentally information that is associated to a skill and allows components to be prepared to execute it. As such, it should provide a list of parameters required for the reconfiguration process along with a collection of components or subsystems to which the configuration is destined for. Within PRIME the notion of priority also arose as a necessity, mainly due to the fact that a certain skill can be executed by different components and these components can require different configurations (Santos, 2015). Adding priority as one of the configuration's

parameters facilitates the reconfiguration process by providing a selection criterion for the possible configurations.

### 3.2.3. Semantic Model

When using a distributed approach built upon autonomous and cooperative entities, it is important to verify that the semantic content is preserved during the communication process (Borgo & Leitão, 2004). In PRIME's specific case, as stated in (Orio et al., 2015), the semantic language developed by UNINOVA, displayed in Figure 3.2 specifies the structure for the knowledge models and system's communication in general. It captures the main characteristics of the physical resources and their aggregations as a system, encapsulating enough information to support the extraction of knowledge required to promote self-awareness, monitoring and the capacity to adapt to changing conditions (such as those introduced by disturbances).

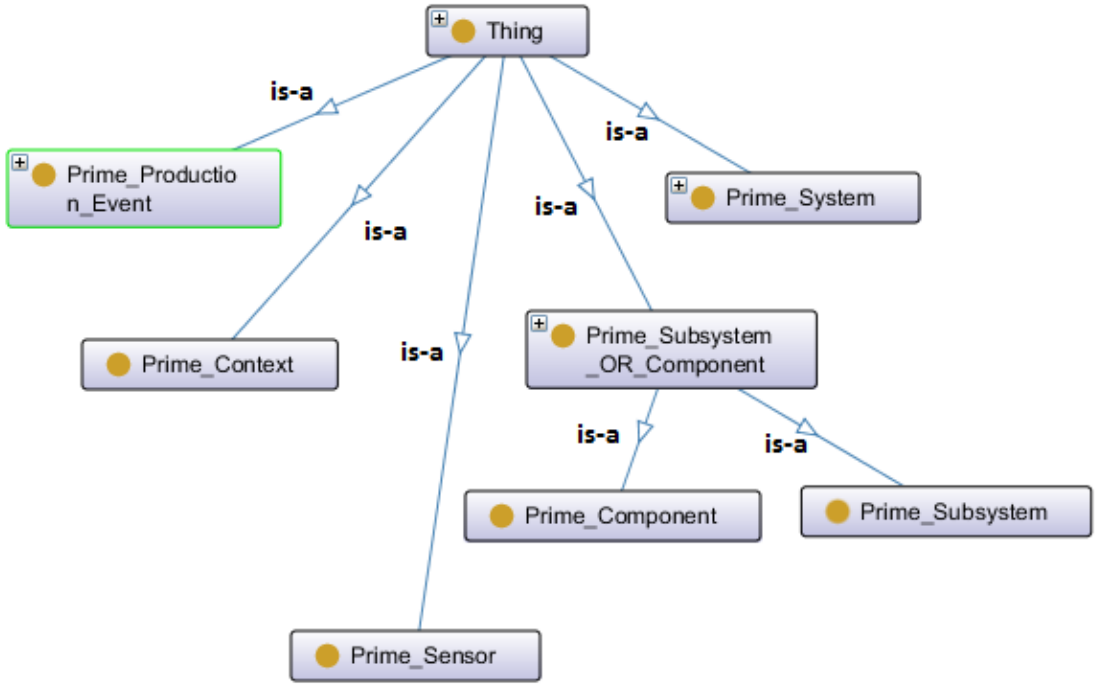


**Figure 3.2 - Core concepts described in the PRIME semantic language - Adapted from (Orio et al., 2015)**

Due to its nature, this model can be used to represent and define important concepts, such as the ones presented in this section, along with the relations between them by means of a device centric ontology. This knowledge representation is the foundation upon which possible implementations should be built.

## 3.3. Supporting Concepts for Monitoring

### 3.3.1. Data Extraction

Data extraction is usually viewed as a problem mainly concerning system integration, consisting on the retrieval of available data provided by a system's data sources (e.g. PLCs and other devices that can provide and store data related to the system's execution)(A. D. Rocha et al., 2015). This process usually leads to the acquisition of raw data which requires further processing in order for it to be useful in regards to the description of the system or its analysis. For instance, the bits stored in a controller's memory which describe a state indicated by sensors and actuators can be a good example of raw data.

Data can also be divided into different types. The definitions of states and timespans are particularly useful and can be found below.

### 3.3.2. State

A state indicates the condition of an entity at a certain point in time according to any given number of variables, in this case the values given by a system's sensors and actuators.

Associating this to the fact that most data extracted directly from a data source is usually raw, unprocessed data, it is possible to further divide states into two groups, namely Raw States (RST), which are given directly by a certain value extracted from the data source, and Computed States (CST), referring to those derived from processing a set of extracted values.

### 3.3.3. Timespan

A broad definition of timespan would be the time elapsed between two relevant states. However, as mentioned in (A. D. Rocha et al., 2015), in an industrial manufacturing setting (and more specifically in the study case in question) there are two categories which are particularly important to distinguish, namely Transition Times (TRT) and Action Times (ACT).

A Transition Time is the time it takes a certain resource to physically move from one position to another, being usually associated with three different states as illustrated in Figure 3.3. In this example the transition is performed by a clamp, indicating the time needed for it to close. The initial state is related to the point right before the beginning of the transition, the intermediate state indicates the transition itself and the final state refers to the end of said transition, as indicated by a sensor that signals that the clamp has reached its "closed" position. From a monitoring standpoint, this timespan is valuable for manufacturers due to the fact that it serves as an indicator of the associated

component's condition. By computing its moving average and trend, possible increases in these values can often be interpreted as a sign that maintenance is required for the associated component.

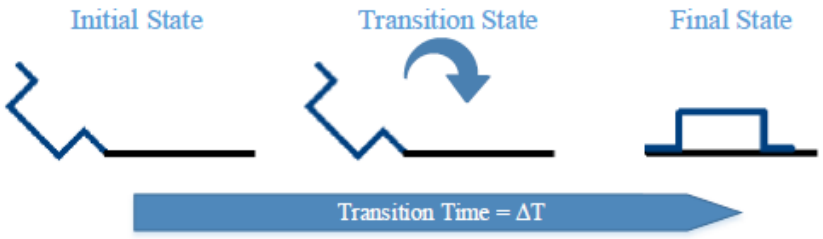


**Figure 3.3 - Transition Time**

Another interesting timespan is the ACT. An ACT represents the delay verified since an input is signalled and the point in time where the corresponding action starts, being therefore directly related to the component's responsiveness.

An example of an ACT can be seen in Figure 3.4, which illustrates a clamp closing. The elapsed time is measured since the input is triggered until the associated component actually starts the corresponding action.

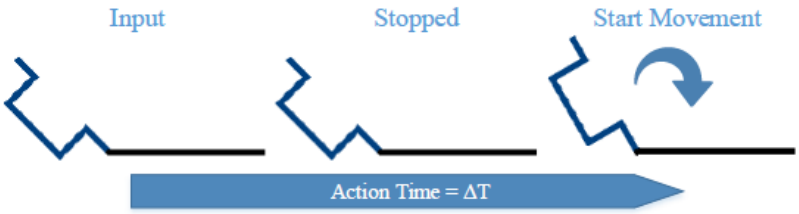


**Figure 3.4 - Action Time**

In the situation described above the initial state, which signals the starting point to measure an ACT, is defined as the moment the input is triggered while the clamp is in the "home" position. This state is followed by a period of time during which the clamp itself does not move, ending only when the component starts the triggered action and leaves the "home" position, signalling the ending point of the ACT measurement.

## 3.4. PRIME Monitoring Architecture

The PRIME Monitoring Architecture presents a possible solution to enable monitoring capabilities in PRIME compliant systems. The architecture employs a generic multiagent approach in order for it to be adaptable to a varied number of manufacturing systems, providing a middleware capable of bridging the gap between the devices on the shop floor and external entities such as remote historical repositories and powerful processing networks with the capacity to support large amounts of data.

An overview of the referenced architecture, composed by three distinct types of agents and their respective interfaces, can be seen in Figure 3.5. This illustration is followed by a detailed explanation of the main procedures that constitute the monitoring process as a whole, namely agent deployment, data extraction, data pre-processing and data exportation.

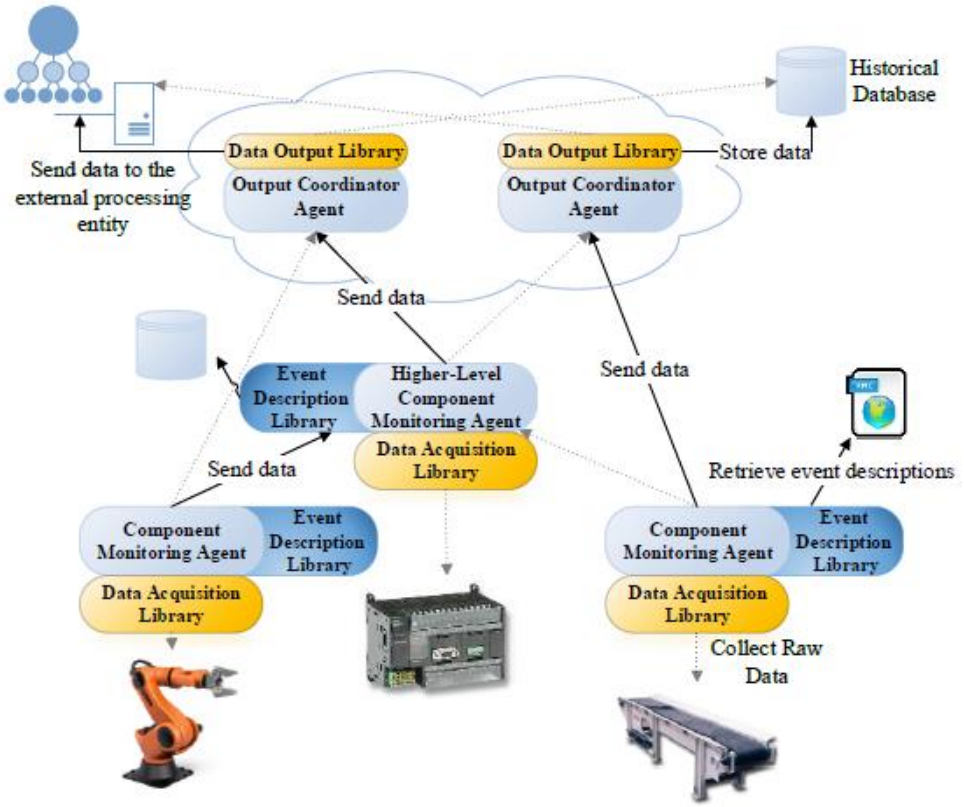


**Figure 3.5 - PRIME Monitoring Architecture**

As Figure 3.5 suggests, each device on the shop-floor is abstracted by a low level agent, the CMA, responsible for the acquisition and pre-processing of data pertaining to said device. This can be achieved due to the existence of two auxiliary libraries which will be further detailed in the subsequent sections of this chapter. Each CMA can in turn be associated with a HLCMA, which

despite having similar functionalities acts at a higher-level of abstraction, enabling the creation of a tree of monitoring entities. Finally both of these types of agents communicate with the OCA cloud with the purpose of finding an available OCA to relay their collected and pre-processed data to the appropriate external entities.

### 3.4.1. Topology Acquisition and Agent Deployment

As described in Table 3.1, the DA is the agent responsible for launching new agents in the system. This operation can happen in two distinct situations, more specifically the moment the DA is initialized and when it detects that a new component has been plugged.

When the DA is initialized, before it launches any other agents on the platform it provides the ability to, if desired by the manufacturer, autonomously discover the topology that describes the system being monitored. With this purpose in mind, using a Hardware Detection Library (HDL) it can consult an external source to acquire this information. This behaviour is depicted in Figure 3.6.

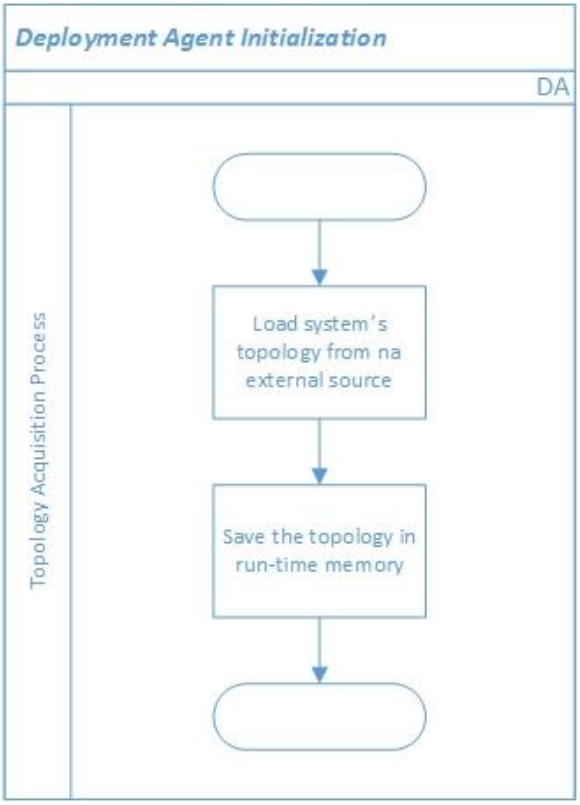


**Figure 3.6 - Deployment Agent Initialization**

As demonstrated, having this information the DA is then able to assert how many components and subsystems are encapsulated in the system, launching CAs and PMAs accordingly which in turn launch their respective CMAs and HLCMAs.

This generic behaviour makes it possible for the architecture to be applicable to a wide array of different manufacturing systems, regardless of the number or type of existing components.

DAs also play a vital role in enabling P&P characteristics for a given system by detecting changes in its topology during runtime, allowing components to be plugged or unplugged without the need to stop the entire line. This behaviour can be seen in Figure 3.7.

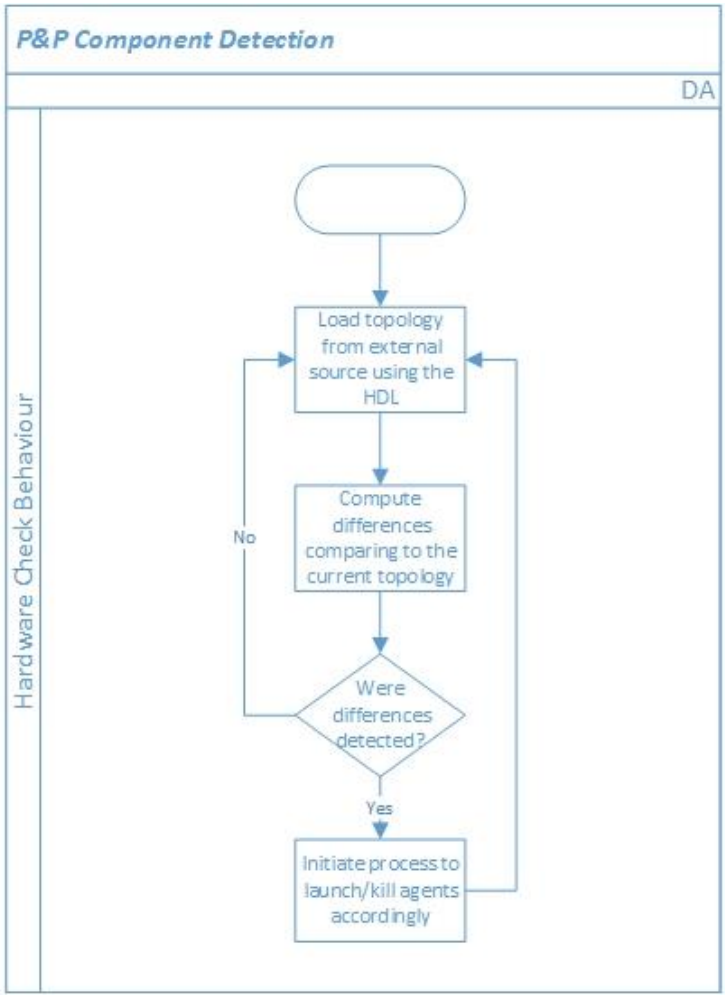


**Figure 3.7 - P&P Component Detection**

This is achieved by periodically checking the external topology source, using the HDL interface, to see if anything has changed when compared to its latest recorded information. As shown in Figure 3.7, in case a modification is detected, the DA acts accordingly either by launching a new CA (in case a new device is plugged) or by initiating the process to remove agents associated with an unplugged device from the platform.

Taking the assembly line represented in Figure 3.8 as an example, the topology description file would contain four different subsystems and twelve components.

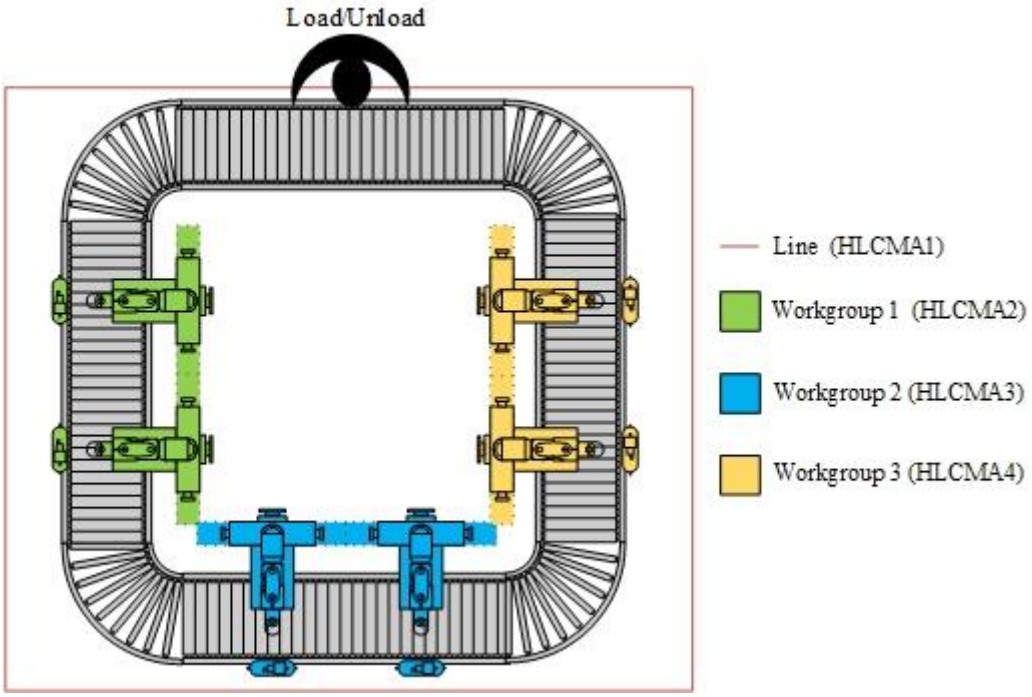


**Figure 3.8 - Example Line for the Topology Acquisition**

The highest level entity is the line itself, which would be abstracted by a PMA and an associated HLCMA. This subsystem comprises three other subsystems, namely Workgroups 1, 2 and 3. Each workgroup (WG) is in turn composed by two robots and two part detectors (PD), each abstracted by a CA and an associated CMA, resulting in the monitoring tree seen in Figure 3.9.

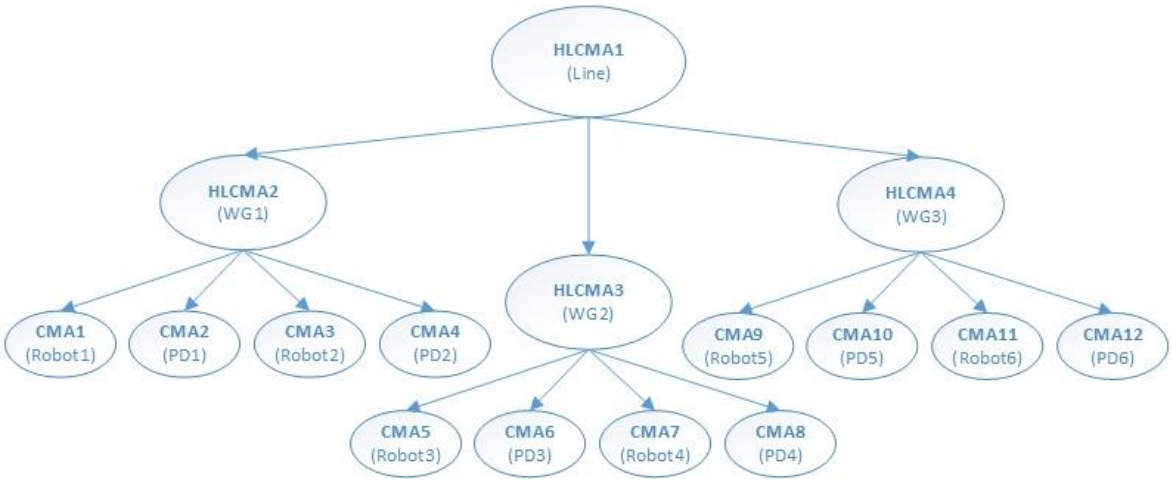


**Figure 3.9 - Example of a Monitoring Tree**

As illustrated, each CMA is responsible for abstracting a specific device, being associated with one and only one parent HLCMA. In turn, each HLCMA abstracts a subsystem, such as a specific workgroup or even the entire line, allowing the association of several lower-level CMAs or HLCMAs, therefore forming the monitoring tree.

### 3.4.2. Data Extraction Procedure (DEP)

In order for data to be pre-processed it is first and foremost necessary to extract it from its sources, making the DEP the first stage of the actual monitoring process itself.

They key players in this procedure are the CMAs and HLCMAs. As it can be verified in Figure 3.5, both types of agents possess two different communication interfaces to external entities, more precisely the Event Description Library (EDL) and the Data Acquisition Library (DAL).

In order for the agents to recognize all the possible events and values related to their abstracted component or subsystem, each agent possesses its own Knowledge Base (KB) capable of storing the rules and descriptions that define their associated monitoring data. For this purpose, the EDL should contain methods to allow the CMAs and HLCMAs to access an external data source (e.g. a DB or an XML file) in order for them to learn information regarding the data they will be monitoring, more specifically which values they should extract, how often they should be extracted (polling rate) and all the rules concerning the conditions that define possible events that need to be computed by the agents themselves (pre-processing).

This learning process takes place during the agent's initialization and can be seen in Figure 3.10. Upon waking up the agents load the monitored data description related to their associated component using the EDL, storing it in their KB to enable the data extraction process.

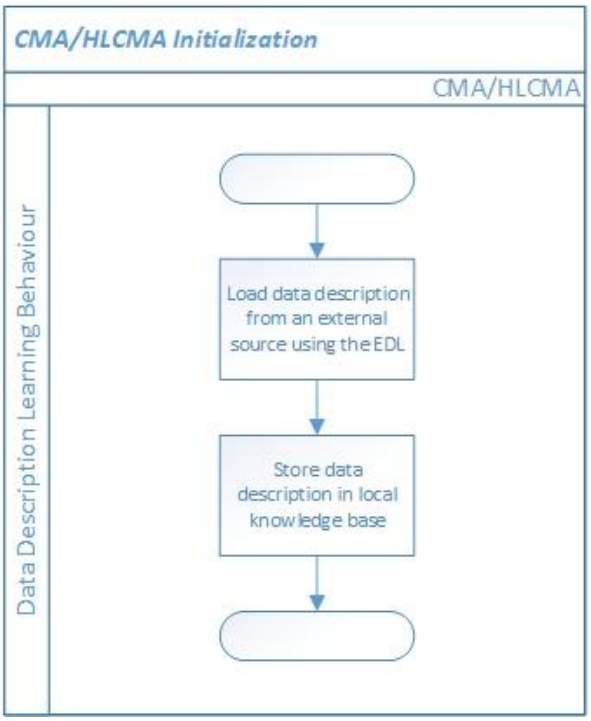


**Figure 3.10 – CMA/HLCMA Initialization**

After acquiring the necessary information in the previous step, the CMAs and HLCMAs can begin extracting data from their associated components and subsystems respectively. This extraction can happen in two different ways, depending on the DAL's implementation. Both methods can be seen in Figure 3.11.

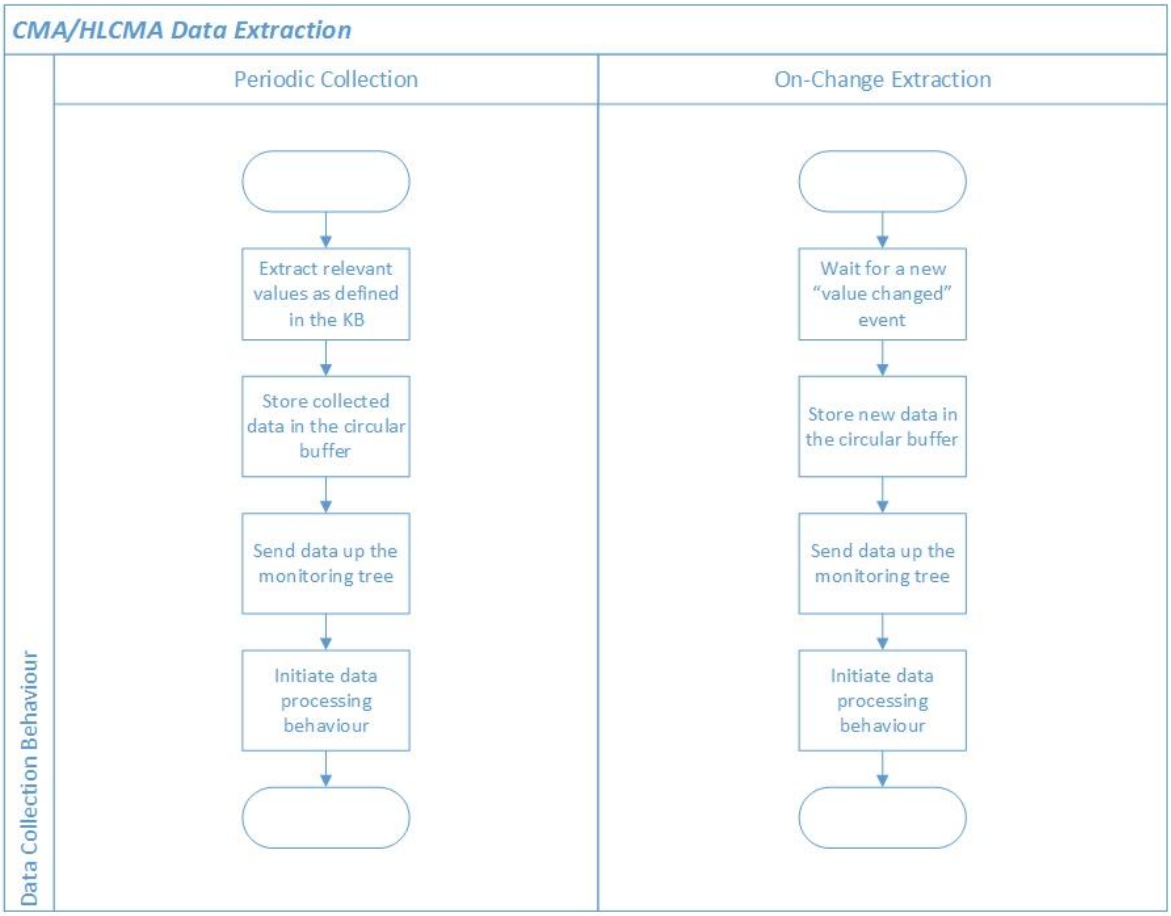


**Figure 3.11 - Data Extraction Process**

As observed in Figure 3.11, data can either be extracted periodically according to a certain polling rate, usually specified in the monitored data description, using methods provided by the DAL, or it can be acquired via events fired by the DAL itself, for instance when the underlying technology fires a "value-changed" event upon detecting that a certain monitored value has changed since its last extraction.

Regardless of the way the values are acquired, each CMA and HLCMA stores all collected data temporarily in its internal memory. This memory should act as a circular buffer, similar to the one presented in Figure 3.12, where only recent states are kept, making it possible for the agents to consult relevant past states in order for them to compute interesting events that may have occurred, as described in the following section.

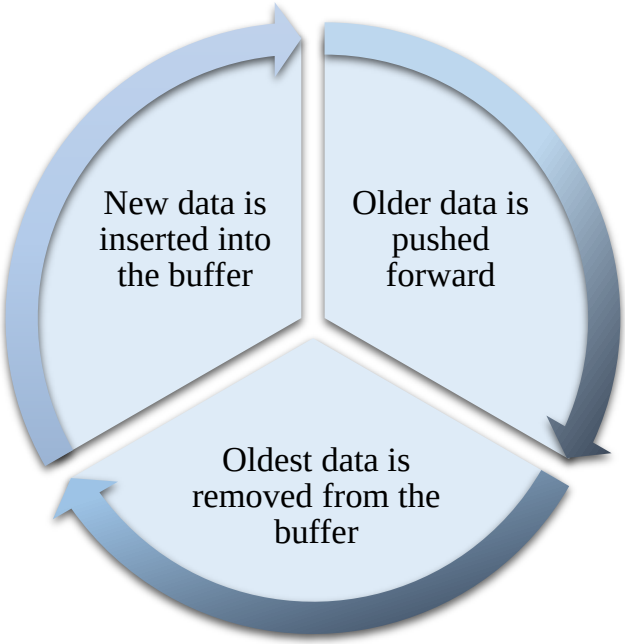


**Figure 3.12 - Circular Buffer's Functionality**

As seen above, as fresh data is inserted into the agent's buffer, all previously stored information is pushed forward, forcing the oldest, outdated data out. This functionality allows the agent to only keep relevant data in its memory, whilst flushing out obsolete information.

All collected data is also sent upwards to an available OCA in order for it to be exported. This behaviour will be approached in further detail in section 3.4.5.

### 3.4.3. Higher-Level Data Propagation

The main difference between the CMA and HLCMA resides in their different levels of abstraction. While the CMA acts at the component level, the HLCMA stands at a higher point in the monitoring tree, being responsible for abstracting groups of components cooperating amongst themselves, which are in essence subsystems.

As such, taking into account that these agents operate at a higher-level of abstraction, it can be useful for them to not only collect data, when possible, directly from the subsystems themselves but also to receive the data acquired and pre-processed by their associated lower-level agents. This process is illustrated in Figure 3.13.

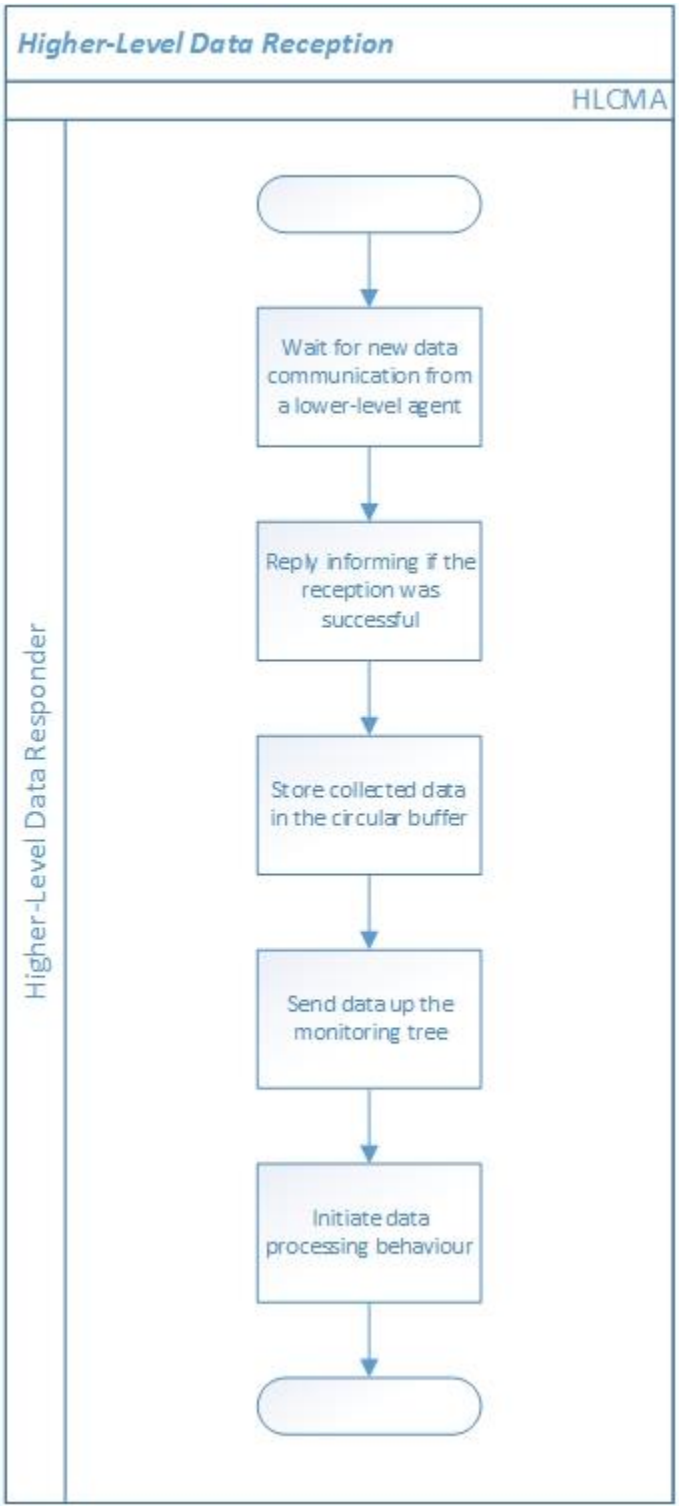


**Figure 3.13 - Higher-Level Data Reception**

Essentially all data that passes through the monitoring agents is also propagated throughout the agents that stand above them in the monitoring tree, allowing more complex data to be derived from the lower-level information gathered.

This is achieved by having every monitoring agent initiate a data propagation process upon either collecting or processing new data. A communication is initiated with its parent HLCMA (if it exists) in order for said data to be transmitted, which in turn replies with the transmission's status. Afterwards the HLCMA stores the new data, propagates it in similar fashion and initiates the processing behaviour, as shown in Figure 3.13 .

### 3.4.4. Data Processing Algorithm (DPA)

The DPA is one of the key points of the monitoring structure, being used by the CMAs and HLCMAs to infer more complex knowledge from the raw data collected through the regular monitoring process.

Based on the raw data extracted from the physical devices, the DPA allows the CMAs/HLCMAs to acquire higher-level data, such as states and timespans, that wouldn't normally be available to be extracted directly from the devices themselves. This process is the cornerstone that enables the proposed architecture to provide more useful data to the external processing entities, permitting them to perform the analysis of relevant trends and tendencies in the extracted data's values.

The inference process consists initially in computing the raw data values according to a certain rule set stored in each agent's respective KB, as seen in Figure 3.14.

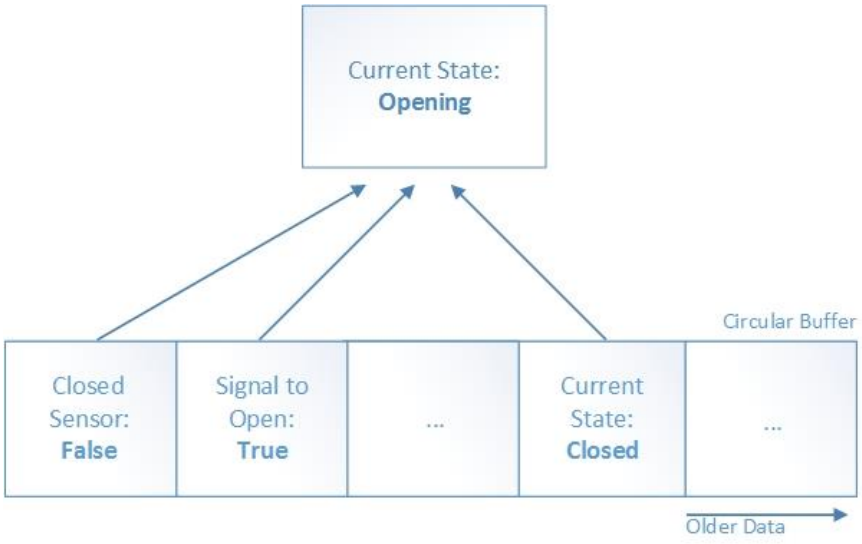


**Figure 3.14 - Knowledge Inference Example - Clamp's Current State**

However, as more complex states are inferred they can also be used to compute new data values, as described above using a clamp's current state as an example. The general workings of the full algorithm are portrayed in Figure 3.15.

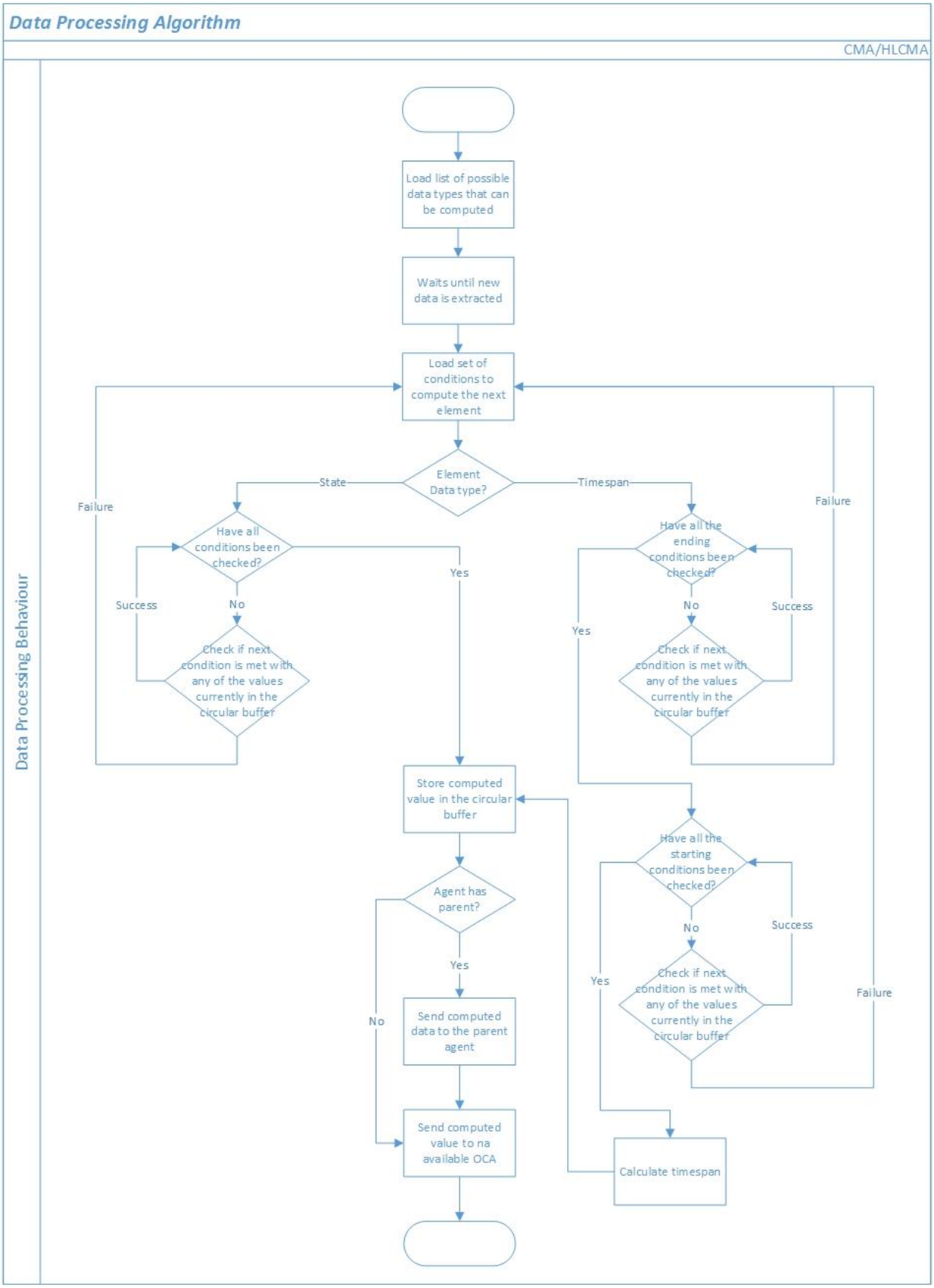


**Figure 3.15 - Data Processing Algorithm**

As already stated in 3.4.2, during the CMA's and HLCMA's initialization process all the information regarding what kind of events can be computed and the rules that define them is loaded onto each agent's KB. The agent then waits until new monitored data is received, regardless of it coming from the agent's own DEP or its children's, starting the main processing cycle upon its arrival.

The processing cycle consists in a series of procedures that for each possible value to be computed, allow the CMA/HLCMA to decide whether the collected data present in its circular buffer in that given moment is enough to compute said value, according to the conditions defined in the monitored data description contained in its KB.

While the processing of some data types, for instance states, is done in a fairly straightforward manner by simply checking if all conditions that define the referenced data type are met, the same does not apply to timespans.

In the former's case, only a single set of conditions must be met in order for a given state to be computed. Therefore all that is required is for the agent to iterate over the entire set of conditions, checking if for each and every one of them there is a value stored in the circular buffer that satisfies it. If all conditions are met, the new inferred state itself is then stored in the circular buffer and propagated to the upper layers of the monitoring tree.

However, in the latter's case, two different sets of conditions must be met in order for a timespan to be calculated, namely those that define the beginning and the end of the relevant period of time. For this reason, the agent starts by checking if all the ending conditions are met, similarly to how a new state is processed. If they are, it stores the latest timestamp among the values that verify that set of conditions, moving on to repeat the process for the starting conditions. However, if at either stage the conditions are not verified the computed value is discarded and the agent starts the cycle anew for the next possible computed value.

If all the conditions are satisfied then the CMA/HLCMA computes the timespan by calculating the difference between the timestamps associated with the starting and ending conditions sets, storing it in its circular buffer and sending it up the monitoring tree.

### 3.4.5. Data Exportation

The last step of the monitoring process is the Data Exportation. As stated in the previous sections of the present chapter, all data (be it raw or processed) is relayed by each CMA/HLCMA to an available agent from the OCA cloud in order for it to be exported to external entities. This OCA

is selected based upon a negotiation process following a specific metric (e.g. response time), which can be seen in Figure 3.16.

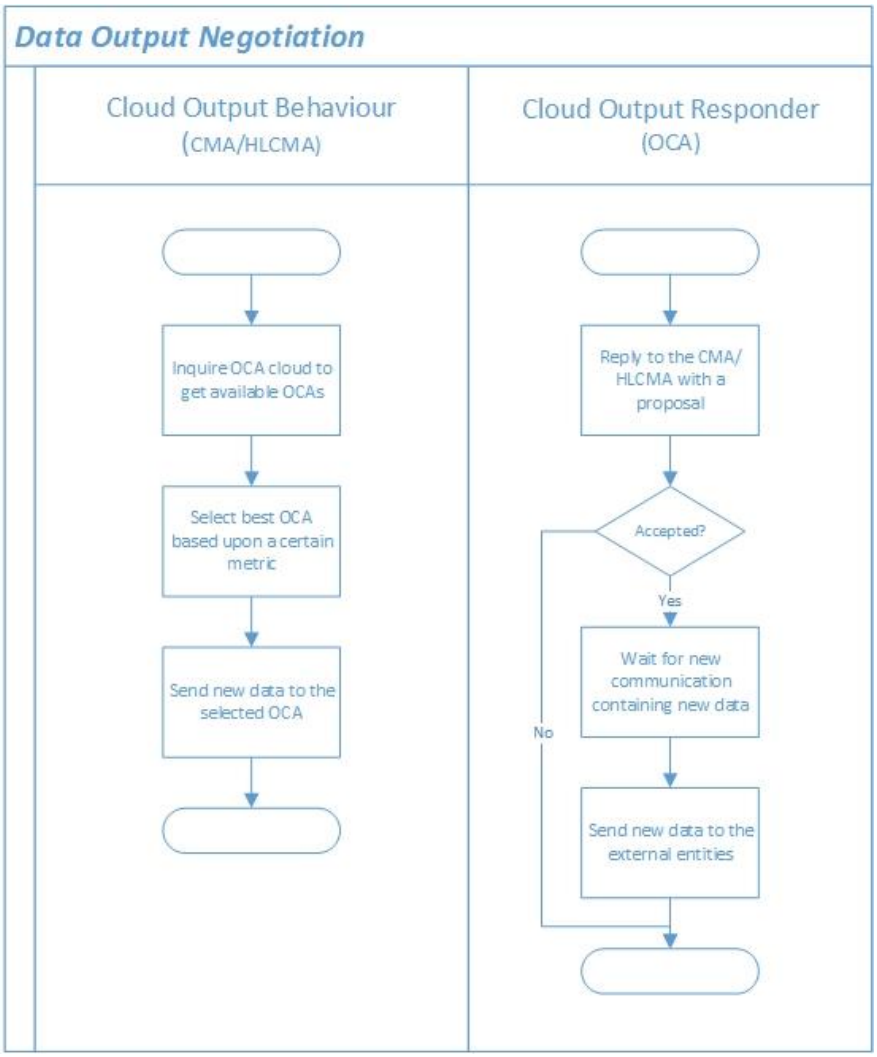


**Figure 3.16 - OCA Negotiation**

As fresh data is received, the OCA begins executing the behaviour responsible for sending it to the aforementioned external entities, using for this purpose the Data Output Library (DOL). This interface should contain all the required methods for the agent to communicate with said entities, acting as an output gateway for the monitoring module.

# 4

# Implementation

This chapter describes the implementation of the monitoring architecture detailed in Section 3.4, developed using Java alongside the Java Agent Development Framework (JADE).

JADE 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 (Bellifemine, Caire, & Greenwood, 2007).

The Java programming language was chosen not only for integration purposes (since the project in which the proposed architecture was integrated for the validation process had been developed in Java) but also due to the fact that JADE is written completely in Java, benefitting from the varied array of language features and third-party libraries widely available.

The present chapter is structured as follows: Section 4.1 starts off by providing a brief explanation of the communication protocols adopted for the agent communication, namely FIPA Request and FIPA Contract Net. Afterwards Sections 4.2, 4.3, 4.4 and 4.5 present a detailed description of each agent's implementation, including its associated data model and respective behaviours.

## 4.1. Agent Communication

Since all JADE agents are FIPA compliant (Bellifemine, Poggi, & Rimassa, 1999), 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.

### 4.1.1. FIPA Request Protocol

The FIPA Request Protocol (FIPA, 2002) allows agents to perform point-to-point communications, being therefore able to request another agent to perform a certain action. As illustrated in Figure 4.1, this protocol specifies that the communication starts when the *Initiator* agent sends a request to the *Participant* agent.

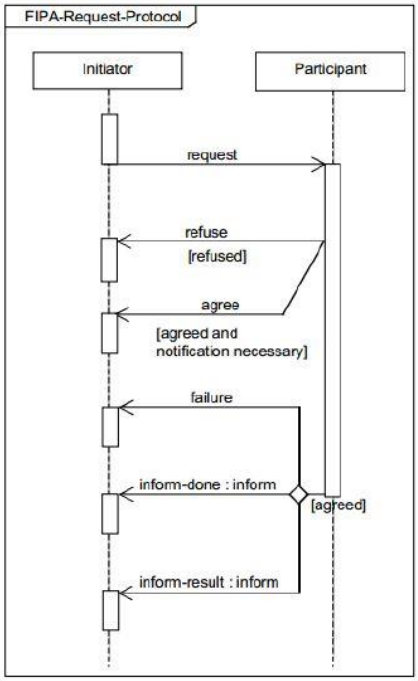


**Figure 4.1 - FIPA Request Protocol**

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.

### 4.1.2. 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.

The protocol dictates that communication starts with an *Initiator* sending a call for proposals to *m* of *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.

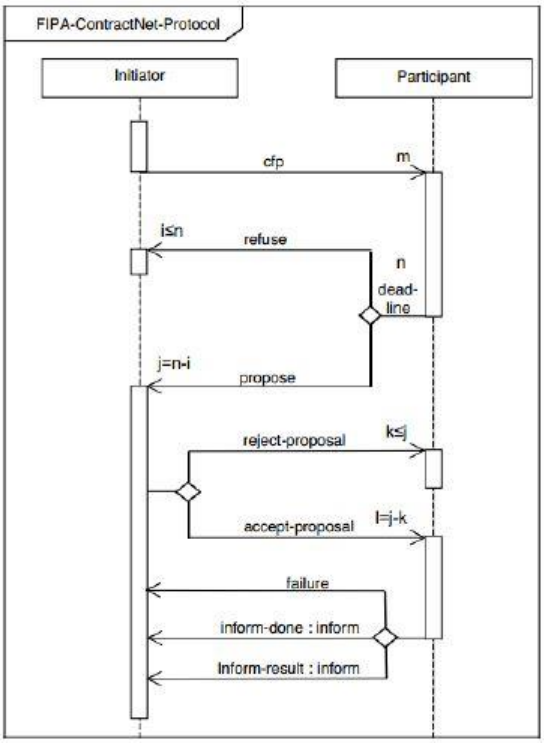


**Figure 4.2 - FIPA Contract Net Protocol**

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.

### 4.1.3. Communication Overview

These interactions are summarized in Table 4.1, where each row corresponds to a different communication initiator (INIT) and each column represents the different responders (RESP).

**Table 4.1 - Agent Interaction Summary**

| RESP / INIT | **CMA** | **HLCMA** | **OCA** |
|---|---|---|---|
| **CMA** | | FIPA Request | FIPA Contract Net |
| **HLCMA** | | FIPA Request | FIPA Contract Net |
| **OCA** | | | |

As suggested by (Ribeiro, Rocha, & Barata, 2013), 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.

## 4.2. Deployment Agent

Even though the DA had already been implemented at an earlier development stage of the project, its capabilities were built upon in regards to the way not only the newly added monitoring agents are deployed, but also the CAs and PMAs themselves.

### 4.2.1. Acquiring the System's Topology

As stated in Chapter 3 Section 4.2.1, before launching the agents that make up the monitoring tree, the DA acquires the system's topology from an external source. This file contains the system's hierarchy of subsystems and associated components, and through the HDL (implemented specifically to process a given file type) the DA is able to parse said file and extract the current system's topology. The specific method in question, *getPluggedResources*, is illustrated in Figure 4.3.

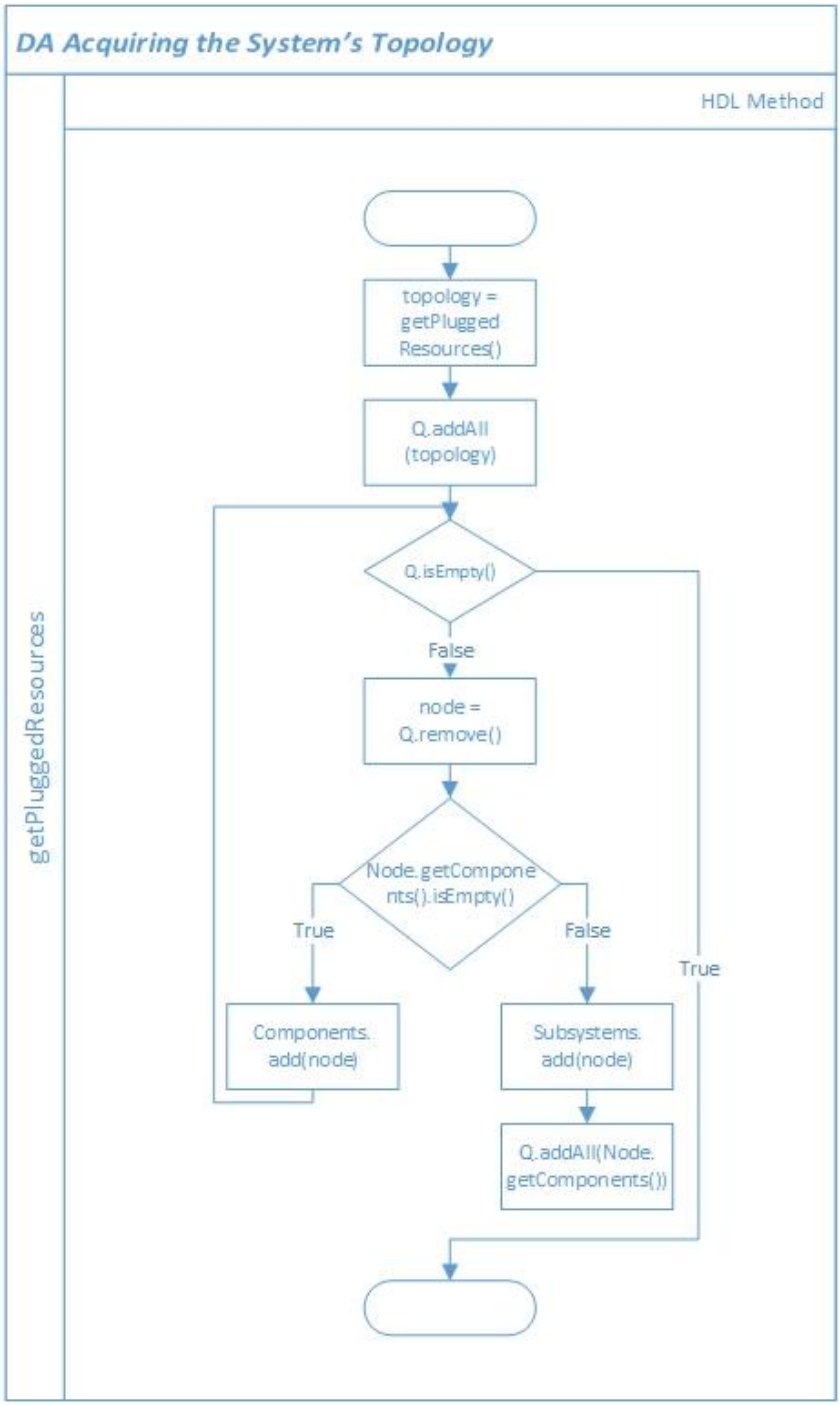


**Figure 4.3 - getPluggedResources (HDL) Implementation**

The agent starts off by parsing the data description file, acquiring the system's topology in an *ArrayList* of objects describing the highest-level entities existent in the system. Afterwards it adds them to a *Queue* in order for them to be processed and initiates the processing cycle, separating the components from the subsystems by checking if each of them has any child elements associated. This can be achieved through the *Node.getComponents()* method. If a certain element is identified as a subsystem, its underlying entities are added to the queue in order for them to be processed, until finally all the entities comprising the monitored system have been identified and processed.

## 4.3. Component Monitoring Agent

As seen in the previous chapter, the 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 4.4.

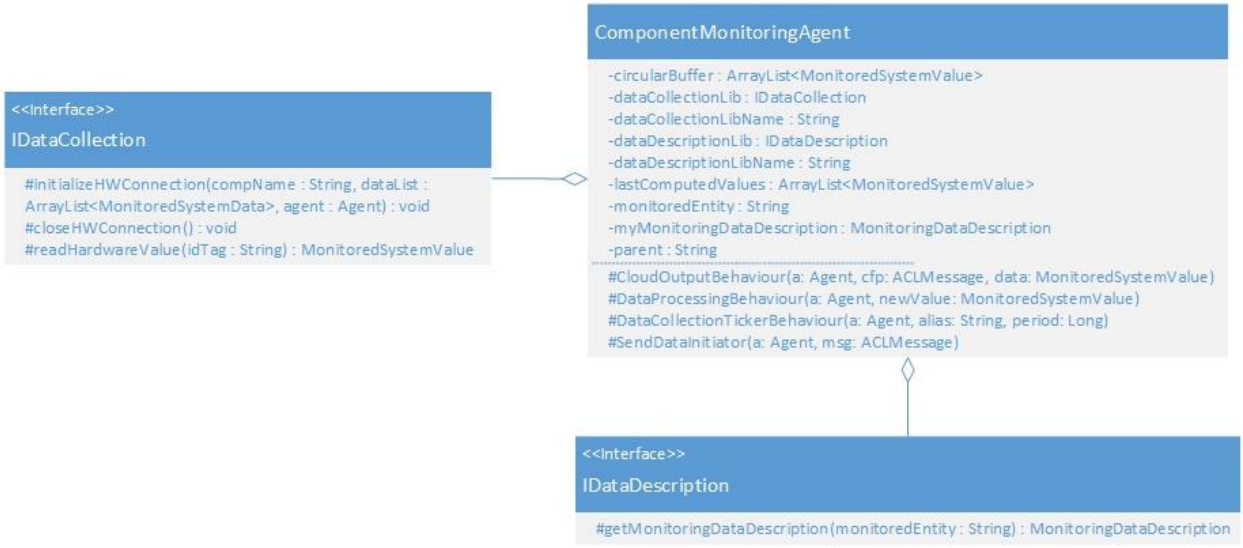


**Figure 4.4 - CMA's Data Model - Class Diagram**

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 4.5 - MonitoredSystemValue Class**

- **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 4.4.
- **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 4.4.
- **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 *MonitoredSystemData* elements that the CMA/HLCMA is expected to either collect or compute from extracted values. Both classes are detailed in Figure 4.6.

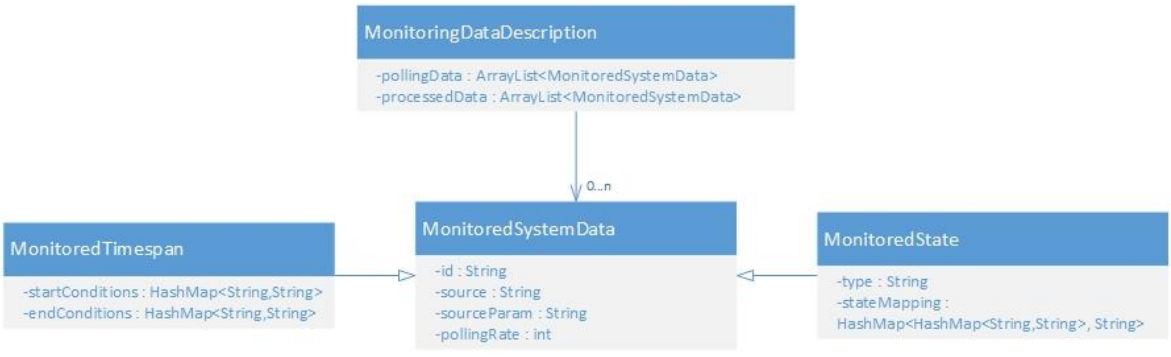


**Figure 4.6 - Monitoring Data Description - Class Diagram**

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

The implementation of each aforementioned behaviour and agent's communication protocols, along with their required interfaces will be described in the following sections.

### 4.3.1. 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 EDL is created, allowing the agent to call the *getMonitoringDescription* method which returns an object of the *MonitoringDataDescription* class. As described in Figure 4.6, 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 4.7 shows the steps executed by the CMA during this task.

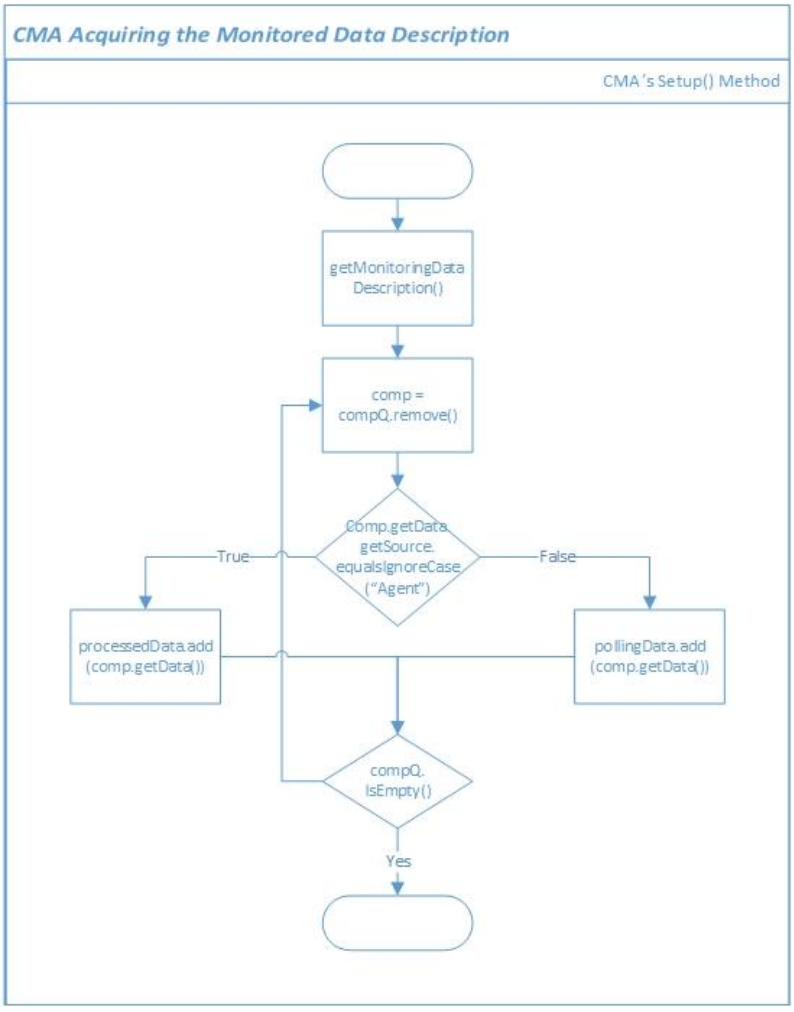


**Figure 4.7 - Acquiring the Monitoring Data Description**

As it can be seen in Figure 4.7, 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 PRIME MDDs 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*.

### 4.3.2. Collecting Data

As discussed in section 3.4.2, 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 4.8. During its initialization, the CMA iterates over the *pollingData* list contained within the *myMonitoringDataDescription* attribute, launching a *readHardwareValue* behaviour for each iterated element.

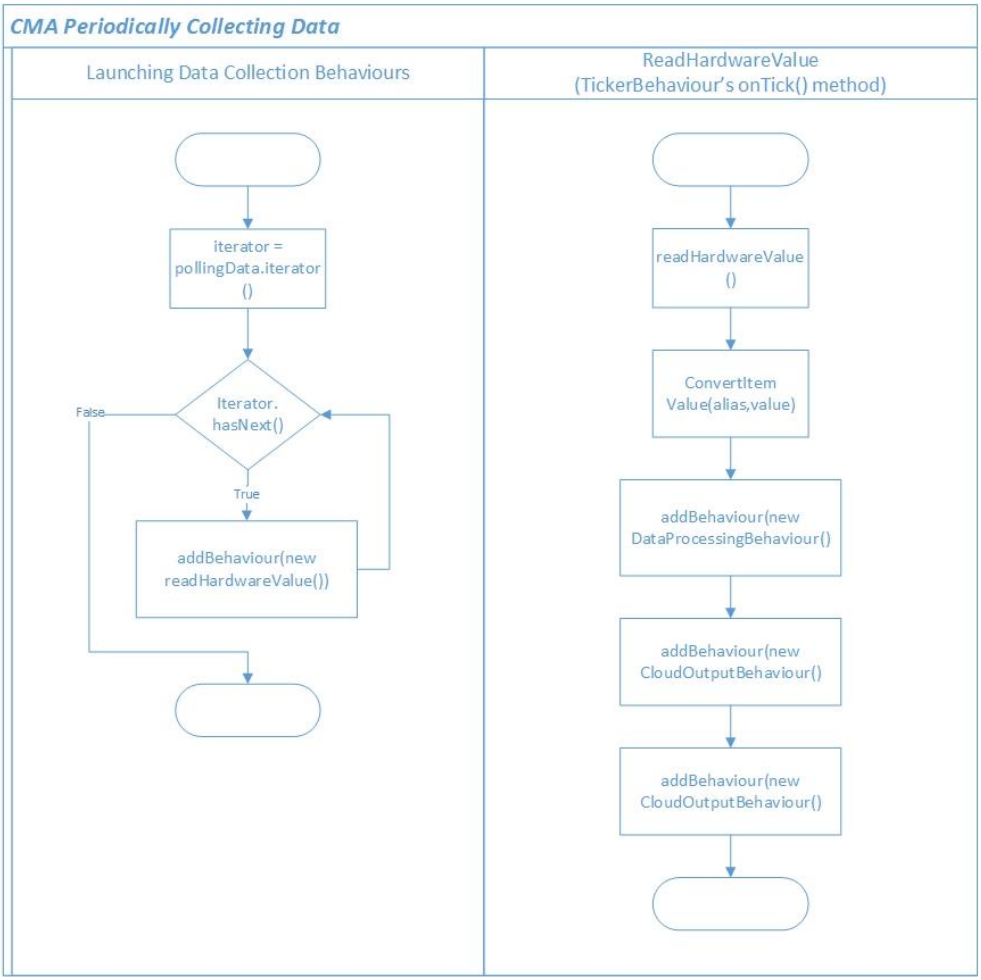


**Figure 4.8 – CMA Periodic Data Collection Implementation**

For the sake of enabling the agent-device interaction a communication interface is also required. For this purpose the 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 (such as in the demonstrator described in Chapter 5), with the agent reference passed as an argument in the *initializeHWConnection* (see Figure 4.4), 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. Section 4.3.4 provides a closer look at these behaviours.

Regardless of the way data is extracted, it is always converted into an object of the *MonitoredSystemValue* class, guaranteeing that the data acquisition process matches the semantics

utilised in PRIME enabled systems, making the monitoring module as independent of the underlying technology as possibly, therefore granting a higher level of interoperability to the system as a whole.

### 4.3.3. Data Processing

The *DataProcessingBehaviour* detailed in Figure 4.9 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.

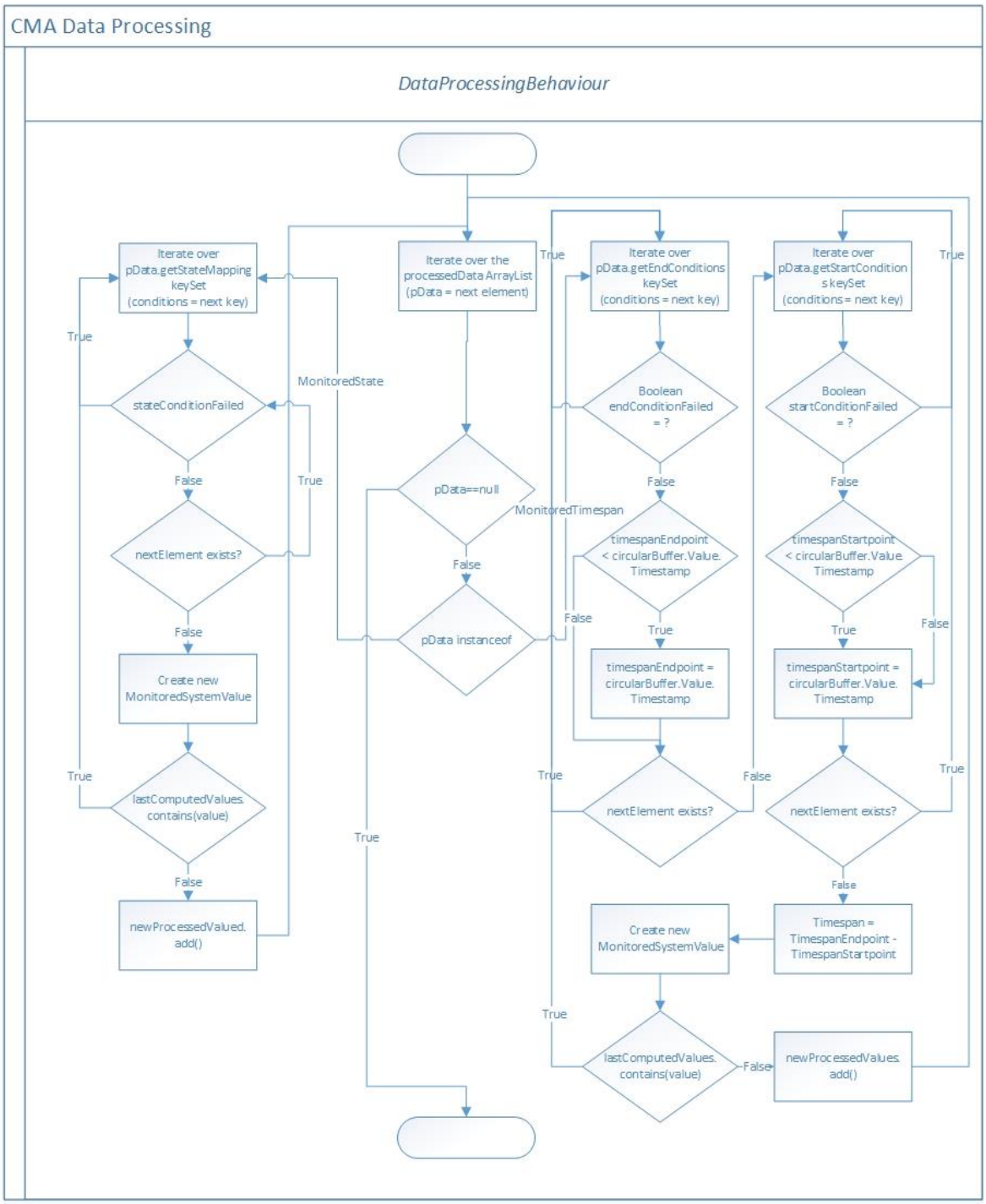


**Figure 4.9 - Data Processing Behaviour Implementation**

As previously shown in Figure 4.6, the conditions that define *MonitoredTimespan* and *MonitoredState* objects were implemented resorting to a *HashMap*. In the former's case two dfferent *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 4.2.

**Table 4.2 - 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. This process is highlighted in Section 4.3.4. In similar fashion, for each CFP initiated a Request is also sent to the CMA's parent through the *SendDataInitiator* behaviour.

### 4.3.4. 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. These interactions can be seen in Figure 4.10.

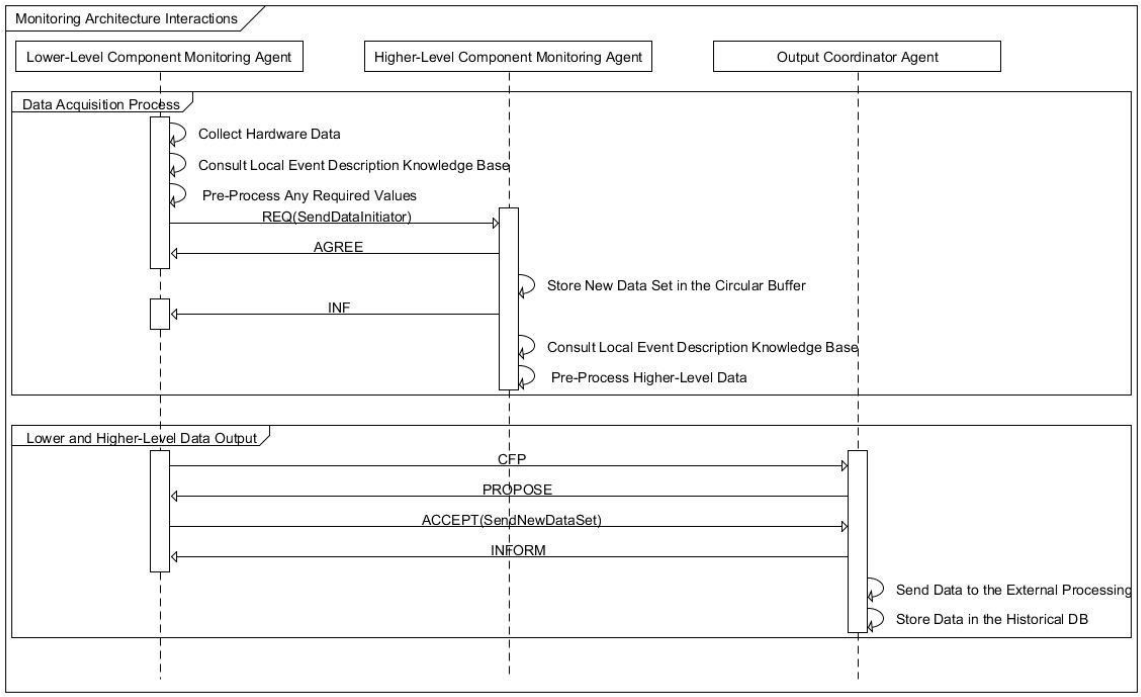


**Figure 4.10 - Monitoring Agents' Interactions**

As illustrated 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 4.11.

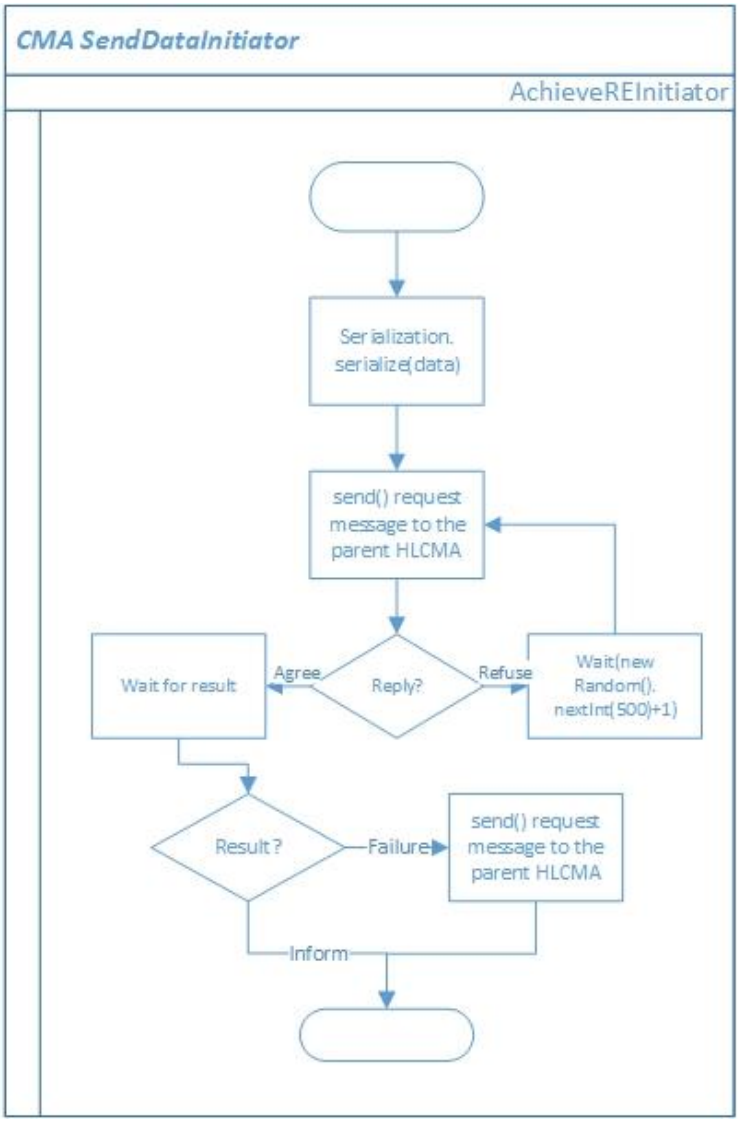


**Figure 4.11 - SendDataInitiator Behaviour**

The *CloudOutputBehaviour*, observed in Figure 4.12, 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 (see Section 4.1.2).

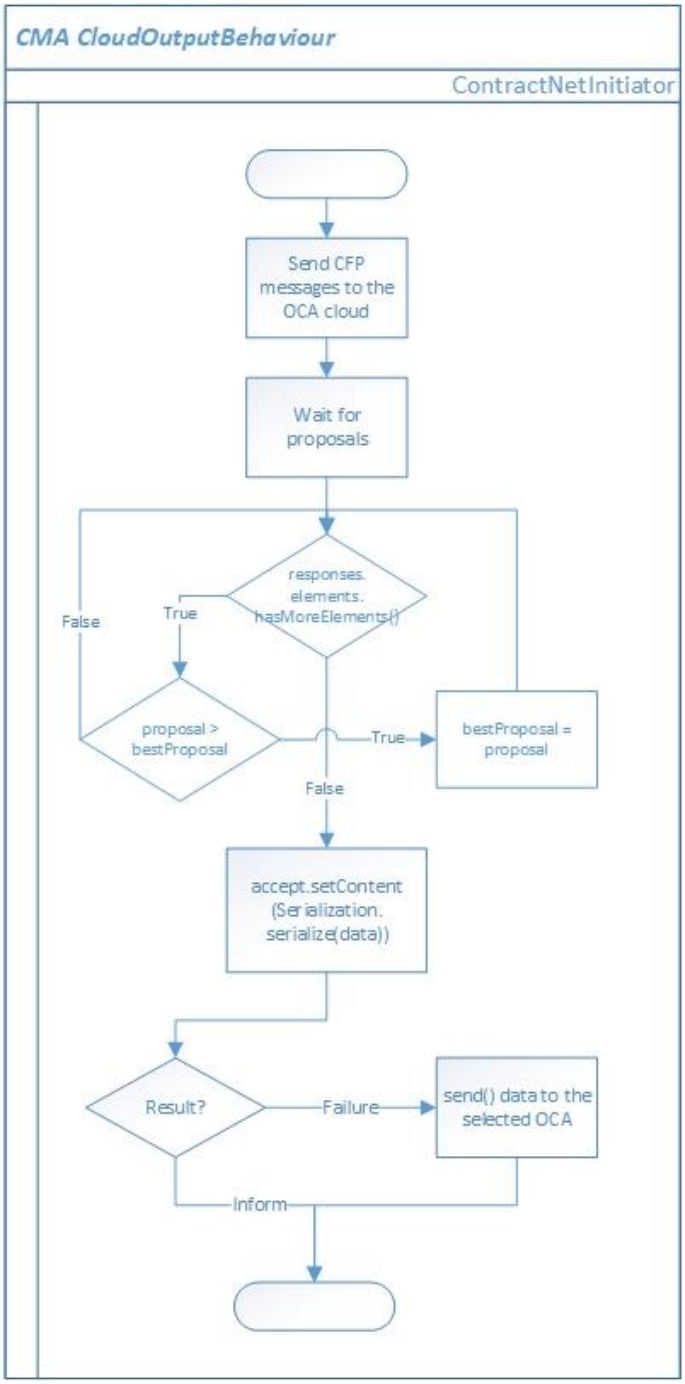


**Figure 4.12 - CloudOutputBehaviour Implementation**

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.

## 4.4. 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, tohe 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 HLCMAs. The data model representing this extension, along with the relationship between the CMA and HLCMA classes is presented in Figure 4.13.

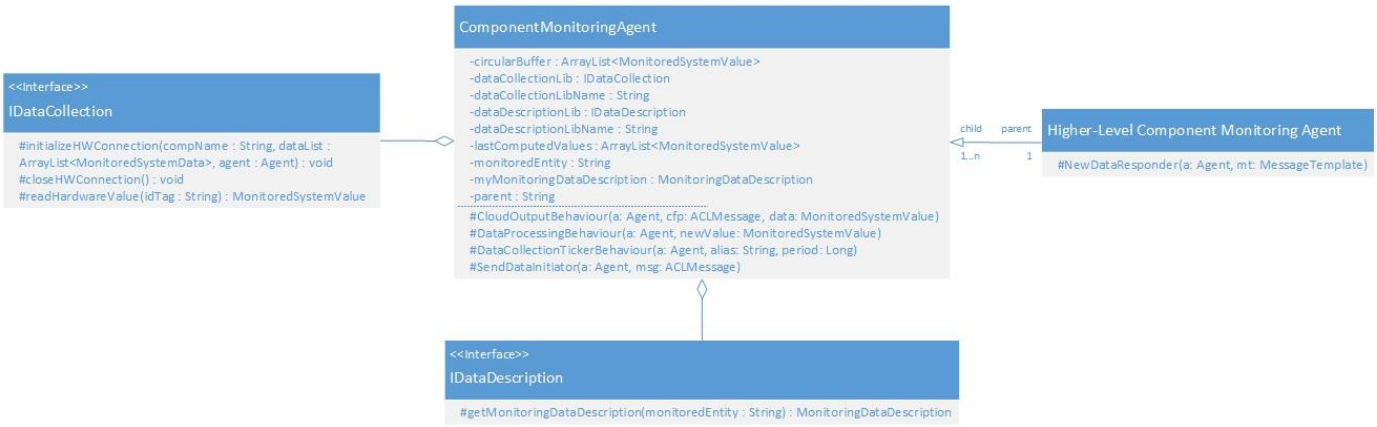


**Figure 4.13 - HLCMA's Data Model - Class Diagram**

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 in Section 4.4.1. 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.

### 4.4.1. Receiving Computed 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 4.14.

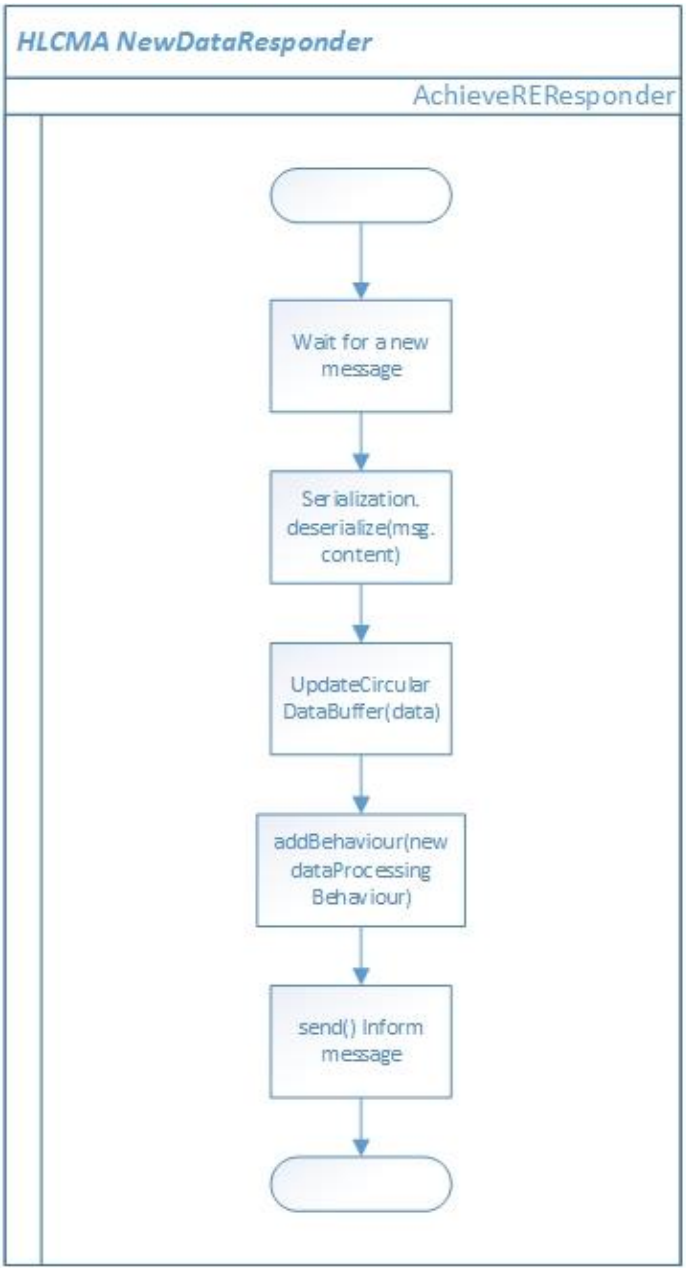


**Figure 4.14 - NewDataResponder Behaviour**

As seen in Figure 4.14, 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.

## 4.5. 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 the DOL The class diagram depicting the data model for both the OCA and the DOL can be observed in Figure 4.15.

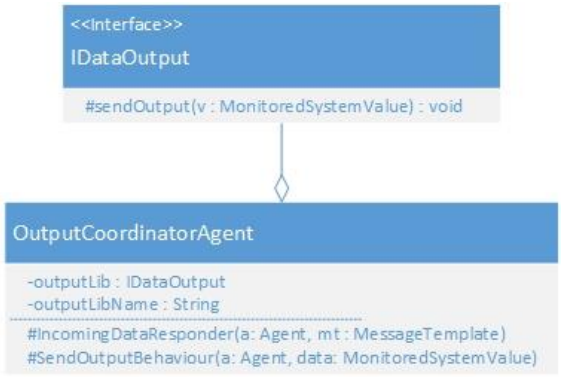


**Figure 4.15 - OCA's Data Model - Class Diagram**

Any implementation of the DOL 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.

### 4.5.1. Exporting Data

The OCA's behavioural logic comprises two different behaviours, the first of which is the *IncomingDataResponder*, described in Figure 4.16. 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 described in Figure 4.10) from a CMA/HLCMA, replying with proposals and handling their response.

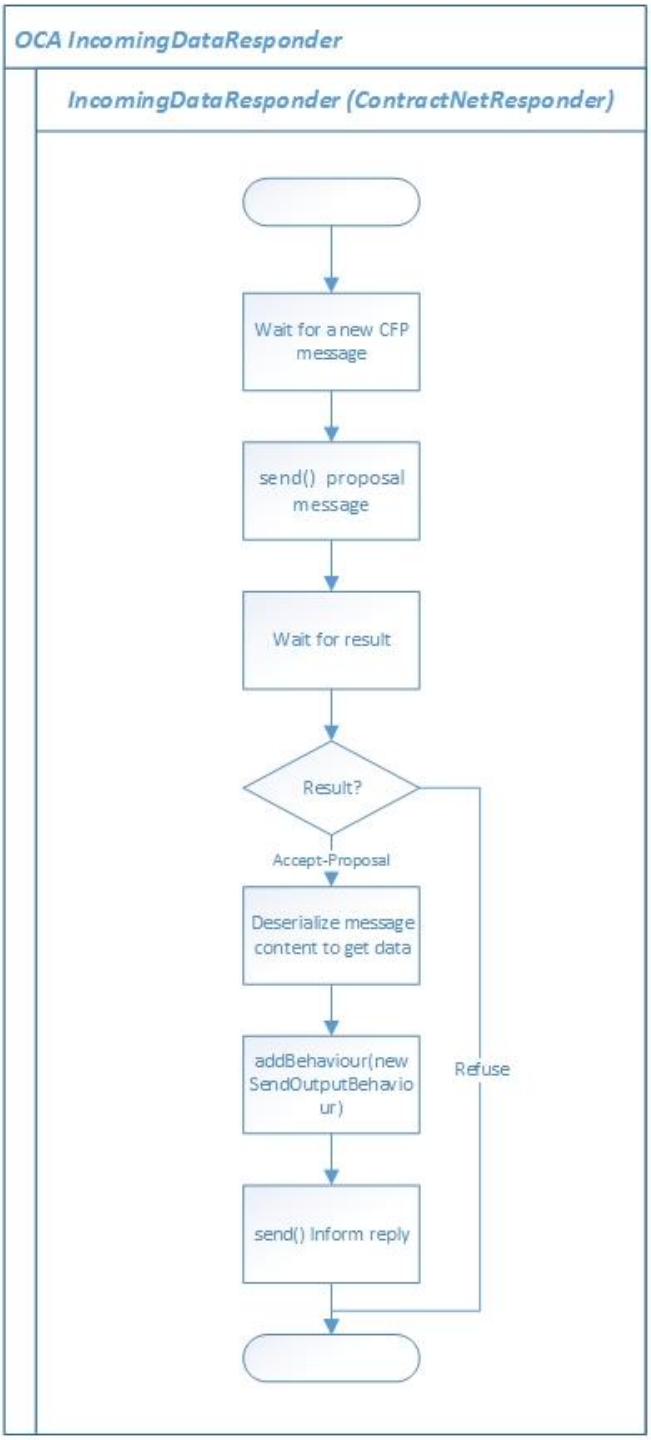


**Figure 4.16 - OCA's IncomingDataResponder**

Upon receiving an *Accept-Proposal* message, the OCA's *IncomingDataResponder* launches the *SendOutputBehaviour*, an extension of the *OneShotBehaviour* class, which in turn calls the DOL's corresponding method, *sendOutput*, which interfaces with the external entities in order to relay the monitored data to them. Since the DOL 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.

# 5

# Results and Validation

The present chapter describes the tests executed to validate the previously presented implementation of the proposed architecture. It mainly showcases the its capacity to extract and pre-process data concerning the system's performance, acting as a technology independent middleware capable of relaying said data to other relevant entities. The chapter is divided into two main sections, both approaching differed aspects of the validation process.

The first subsection covers the description of the industrial demonstrator used to test the developed implementation in a real world manufacturing scenario. It details the agents considered to abstract the whole system, along with the adopted hierarchy.

The last subsection contains a study of the obtained results, providing an analysis of the data collected and processed by the developed monitoring module.

## 5.1. Industrial Demonstrator

The monitoring module was tested in an industrial setting using a demonstrator provided by IntRoSys – Integration for Robotic Systems, a multinational company specialized in industrial automation, focusing mainly in the automotive and aeronautical fields.

### 5.1.1. IntRoSys' Cells Overview

The demonstrator provided by IntRoSys consists in two distinct robotic cells, both capable of transporting and welding a car's side member. In spite of performing the same task, the robots, PLCs and respective standards used differ from one another. An overview of the shop floor in which both cells reside can be seen in Figure 5.1.

**Figure 5.1 – IntRoSys' Cells – FANUC/VW (Left), KUKA/Ford (Right)**

Both cells comprise a robot with an attached gripper, a tool to hold the product before and after the welding process is completed, a stationary welding gun and a series of security devices. Each cell's tool possesses a set of three pneumatic clamps and a centring pin alongside a number of associated sensors. An operator is responsible for loading and unloading the part before and after the process' execution, respectively.

Regarding the structure, the whole assembly line is divided into two workgroups, namely Workgroup 1 which refers to the FANUC/VW cell, and Workgroup 2 which is referent to the KUKA/Ford one. This nomenclature is explained ahead in each cell's individual description. Furthermore, each workgroup has an associated safety group, which in turn comprises a series of components. This hierarchy can be seen in Figure 5.2.

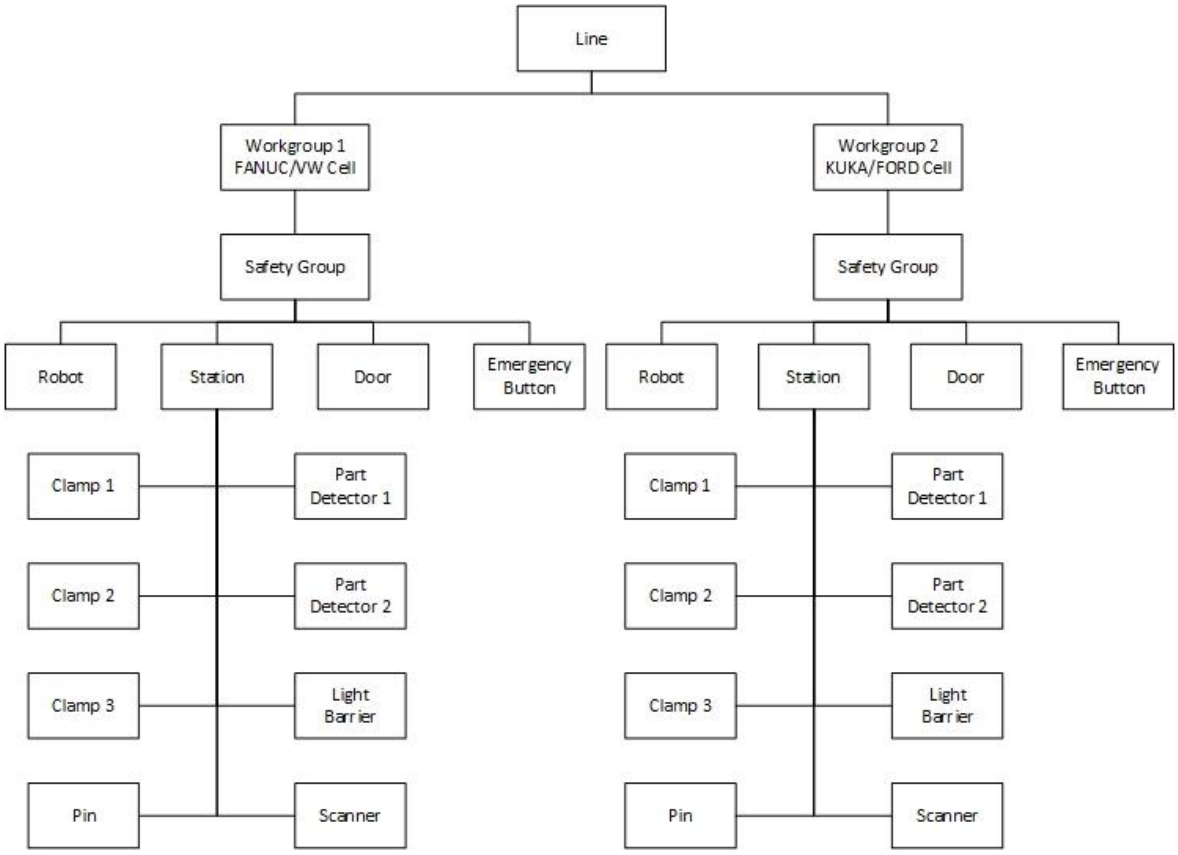


**Figure 5.2 - IntRoSys Demonstrator Structure**

### 5.1.2. FANUC/VW Cell

The leftmost cell depicted in Figure 5.1 encompasses a FANUC robot and a Siemens PLC which controls the process' execution. The VW standard was adopted, hence this cell will hereby be referenced as the FANUC/VW cell.

Its composition is listed in Table 5.1.

**Table 5.1 - FANUC/VW Cell Composition**

| Component | Manufacturer | Type | Communication Interface |
|---|---|---|---|
| R-2000IB/210F | FANUC | Robot | Proprietary/ProfiNet |
| R-30iA | FANUC | Robot controller | Proprietary/ProfiNet |
| Gripper | IntRoSys | Robot gripper | Digital wiring |
| Welding gun | ARO | Joining tool | n.a. |
| Loading Station | IntRoSys | Subassembly | Digital wiring |
| PLC | Siemens | Controller | ProfiNet |
| ET200S | Siemens | I/O | ProfiNet |

| Component | Manufacturer | Type | Communication Interface |
|---|---|---|---|
| IPC677C | Siemens | HMI | ProfiNet |
| Scanner PLS 3000 | SICK | Operation safety | Digital wiring |
| Light Barrier C4000 | SICK | Operation safety | Digital wiring |

### 5.1.3. KUKA/Ford Cell

The remaining cell shown in Figure 5.1 presents a KUKA robot and an Allen Bradley PLC responsible for the process execution control. Since the Ford standard was used, this cell will hereby be referenced as the KUKA/Ford cell.

Its composition is listed in Table 5.2.

**Table 5.2 - KUKA/Ford Cell Composition**

| Component | Manufacturer | Type | Communication Interface |
|---|---|---|---|
| 210 R2700 extra | KUKA | Robot | Proprietary/Ethernet IP |
| KR C4 | KUKA | Robot controller | Proprietary/Ethernet IP |
| Gripper | IntRoSys | Robot gripper | Digital wiring |
| PLC | Allen Bradley | Controller | Ethernet IP |
| AB Safety I/O | Allen Bradley | I/O | Ethernet IP |
| PanelView Plus 1250 | Allen Bradley | HMI | Ethernet IP |
| Loading Station (Tool) | IntRoSys | Subassembly | Digital wiring |
| Welding Gun | ARO | Joining Unit | n.a. |
| Scanner PLS3000 | SICK | Operation safety | Digital wiring |
| Light Barrier C4000 | SICK | Operation safety | Digital wiring |

### 5.1.4. Product Description

The product itself consists in a car’s side member, as illustrated in Figure 5.3.

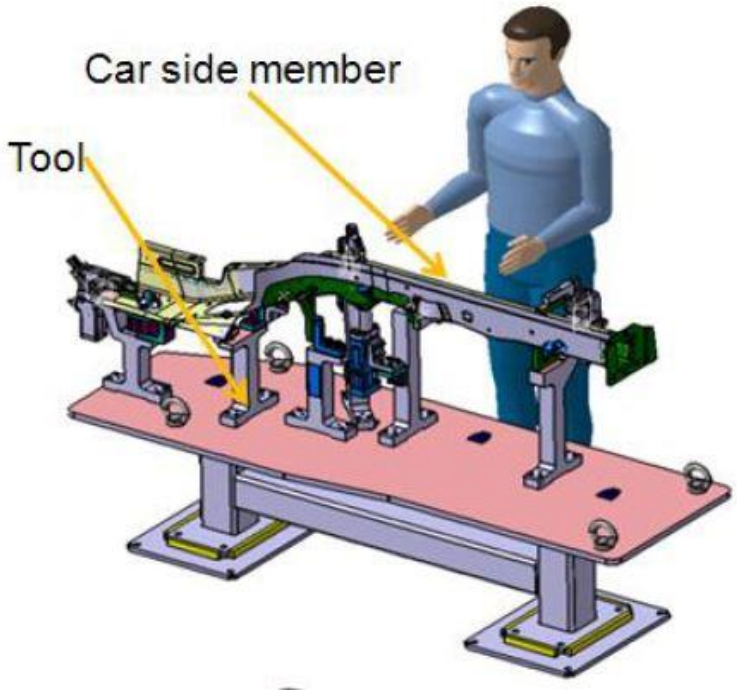


**Figure 5.3 – Product Overview - Car's Side Member**

The product is loaded onto the loading station (tool) by an operator, enabling the clamps to hold it in position in order to allow precise gripping on the robot's part.

The general assembly process is as follows:

- An operator loads the product onto the station
- Station's clamps close
- Robot picks the product
- Station's clamps open, pin goes down, robot unloads the product
- Welding process is executed, station's pin goes up
- Robot loads the part back onto the station
- Station's clamps close
- Robot exits the loading area
- Station's clamps open, pin goes down, operator unloads the welded product
- Station's pin goes up

### 5.1.5. System Execution

In order for it to be possible to successfully extract and process data from the assembly line as a whole, the system was modelled as shown in Figure 5.4. The agent distribution for each cell is

essentially mirrored, since from a device standpoint both cells mirror one another. Hence, the agents pertaining only to the KUKA/Ford cell were omitted from the illustration.

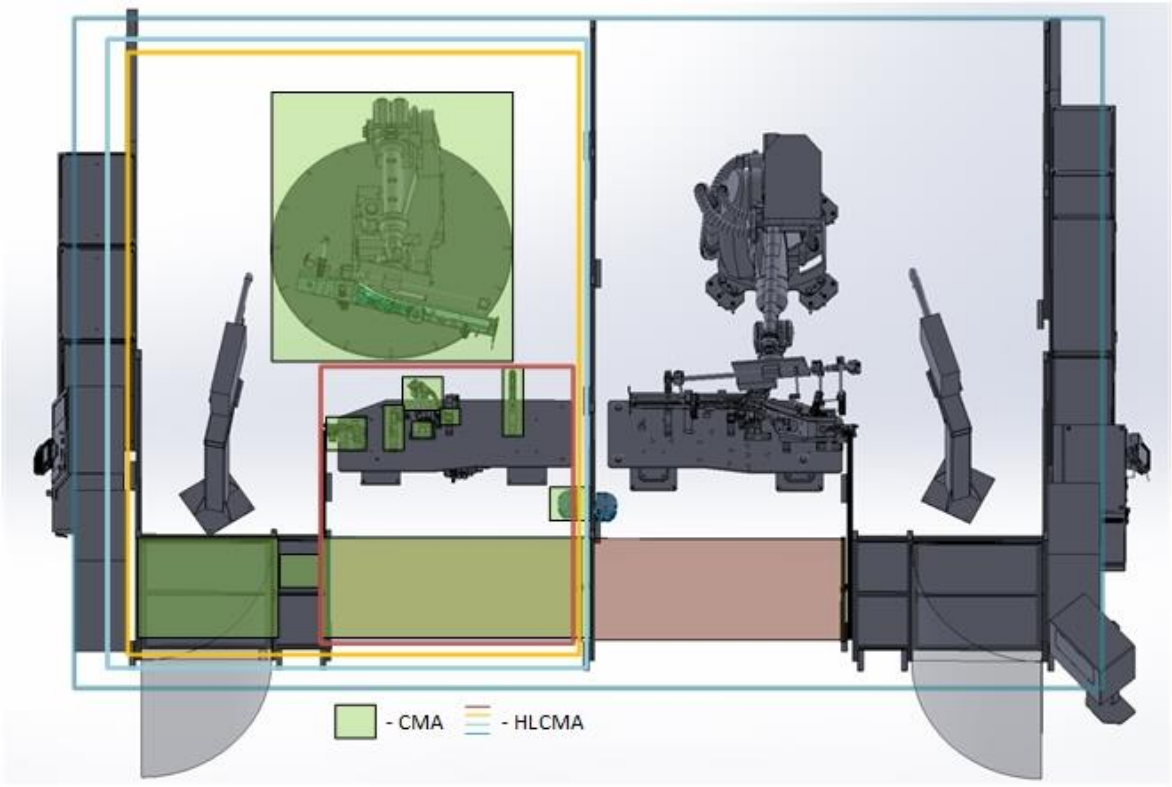


**Figure 5.4 - CMA and HLCMA Distribution**

Essentially each CMA is responsible for abstracting a specific component, being then aggregated to the HLCMA that abstracts the subsystem in which that component operates, such as a safety group or a station. The line's structure is provided by an XML blueprint file, which the DA interprets through the use of an implementation of the HDL, provided specifically for this demonstrator. The use of these interfaces (see Figure 3.5) is one of the main points that allows the monitoring module to function without the need to modify anything in the agents' side regardless of the other technologies present in the system. This grants the system a higher level of interoperability and is showcased in this demonstrator by having PRIME run simultaneously in both cells, despite both using different standards.

As mentioned in the previous chapters, the purpose of PRIME's Monitoring Architecture is to act as a middleware between the physical devices and external processing and storage entities, collecting, pre-processing and relaying data to them.

In this demonstrator's case, three main external entities are worth highlighting. The first of which is a communications platform supporting the OPC UA specifications, KEPServerEX, which

connects to the PLC and enables data extraction. A DCL was implemented which utilises a Generic OPC Connector, allowing the agents to act as a client collecting device data from the OPC server.

One of the other external entities present in this demonstrator consisted in an Apache Cassandra historical DB. Using a Cassandra Connector also provided by SimPlan, a DOL was developed to allow the OCA to export data to a Cassandra queue. This queue would later on be used to feed data to a processing network, which is the last external entity utilised in this demonstrator.

The processing network was developed by TTS – Technology Transfer System S.r.l using ReactiveX/RxJava. Through a Monitoring Daemon, depicted in Figure 5.5, the data inserted into the Cassandra queue by the monitoring agents' module is fed to the processing network, which is in turn capable of computing moving averages and trends based on said data.

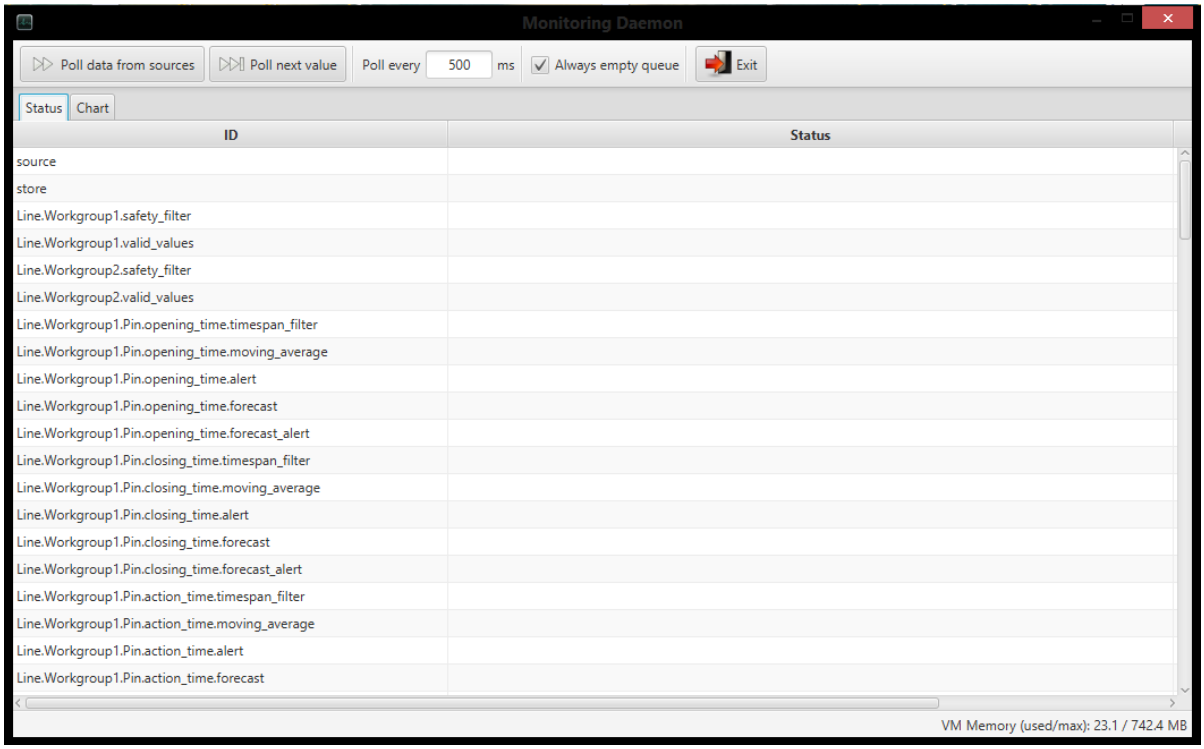


**Figure 5.5 - Monitoring Daemon**

This allows the possibility of preventive maintenance being performed, by predicting possible system failures caused by for instance the degradation of a certain component, ultimately displaying this information on an HMI provided by SimPlan. One of the HMI's views, more specifically the one concerning the assembly line's overview, can be seen in Figure 5.6.

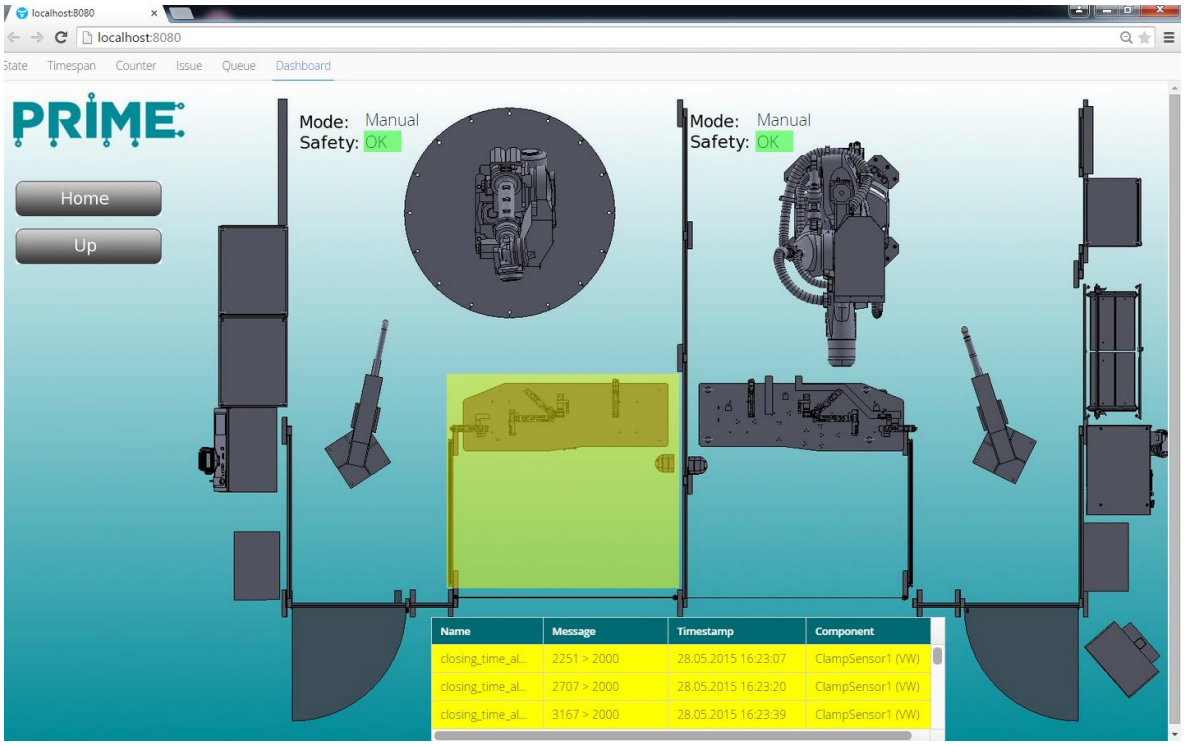


**Figure 5.6 - PRIME's Monitoring HMI Home View**

As illustrated using the information provided by the MAS monitoring module and the processing network, the HMI is capable of displaying potential issues in the assembly line, indicated for instance by an increasing trend in a certain clamp's closing time. This issues are signalled on screen by either a yellow or red highlight of a certain subsystem or component, depending on the issue's severity, as well as by a list of all known issues.

This can be achieved due to the interaction between the different aforementioned entities. First and foremost the monitored data is extracted and pre-processed by the monitoring agents responsible for each component and subsystem. Afterwards, both the collected low-level information and the more complex generated are inserted into the Cassandra queue, making it available for the Monitoring Daemon to access it and pass it through the processing network, eventually inserting it into the DB. Once it has been inserted into the DB the HMI can finally use this processed data to display relevant information, as seen above.

## 5.2. Discussion of Results

As previously mentioned, the tests performed using the IntRoSys' demonstrator were intended to showcase the system's self-awareness and interoperability, via the MAS-based monitoring module's capacity to extract a pre-process data regardless of the underlying technologies and standards, without required further modifications to the agents' coding itself.

These tests were executed on a single machine running the entire PRIME MAS as well as all the aforementioned external entities. A total of 69 agents were launched on the JADE platform, as shown in Figure 5.7.

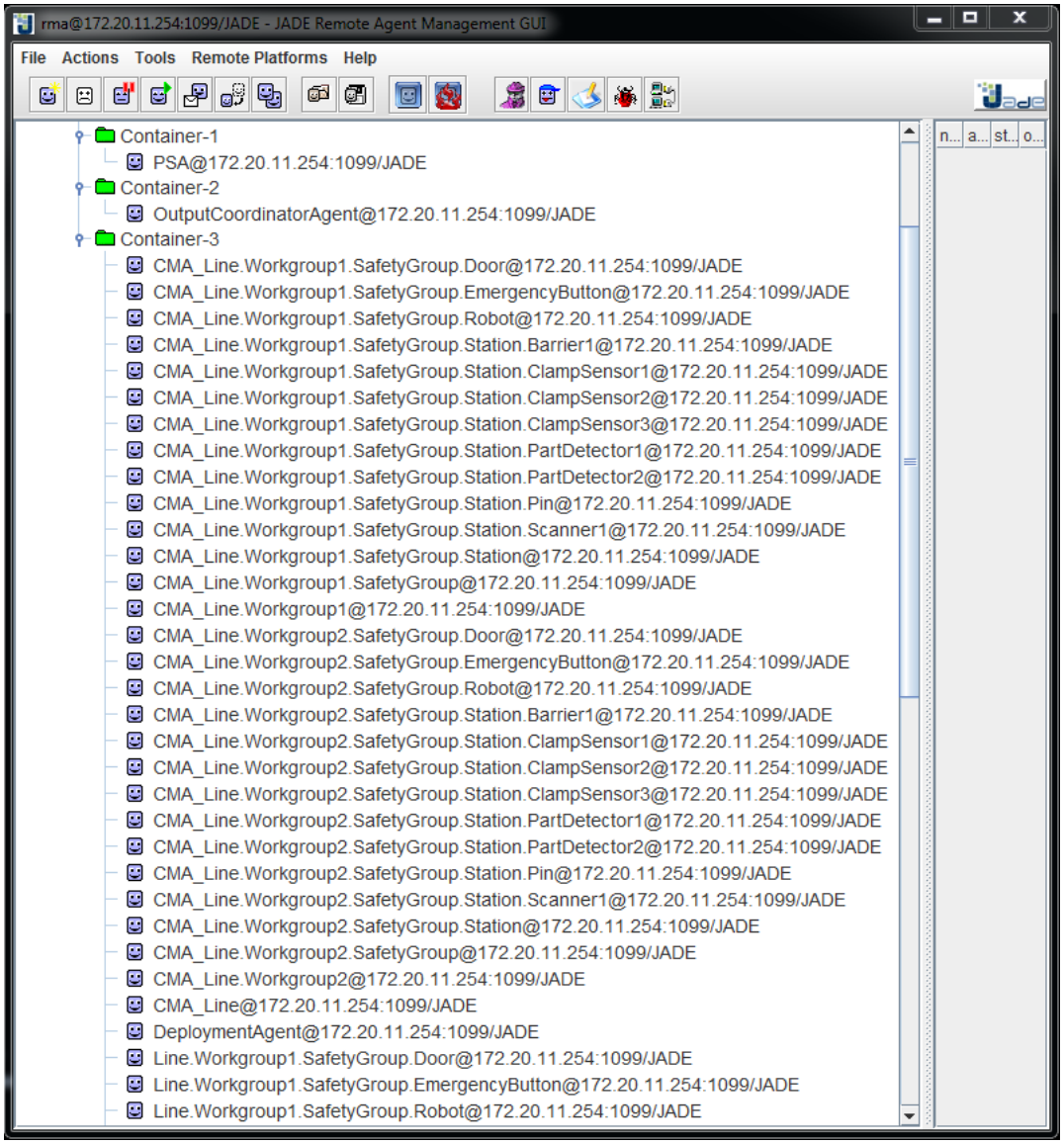


**Figure 5.7 - Monitoring Agents Running on the Platform**

The tests themselves consisted in having the proposed architecture monitoring the system while both cells executed the assembly process described in Section 5.1.4 repeatedly over a period of around forty minutes. During that time a series of disturbances were simulated in order to see if the agents could pick up on these changes in real-time and ultimately have them correctly displayed in the HMI. These disturbances included having an operator trigger the security mechanisms and also limiting the air supplied to a certain clamp or pin, thereby varying its opening and closing times to simulate component degradation.

Due to the high-volume of data collected over the course of these tests, only an illustrative fraction of the values related to the clamps and pins from both cells is listed in Tables 5.3 through 5.10.

**Table 5.3 – Workgroup 1 Clamp1 Monitored Data**

| Raw Data | | | Computed Data | | |
|---|---|---|---|---|---|
| **Monitored Data** | **Value** | **Timestamp** | **Monitored Data** | **Value** | **Timestamp** |
| open | true | 14:53:26 | current_state | open | 14:53:27 |
| closed | false | 14:53:26 | current_state | closing | 14:53:28 |
| signal_change_open | false | 14:53:26 | action_time | 136ms | 14:53:28 |
| signal_change_close | false | 14:53:26 | current_state | closed | 14:53:29 |
| open | false | 14:53:28 | closing_time | 883ms | 14:53:29 |
| closed | true | 14:53:29 | current_state | opening | 14:53:32 |
| signal_change_close | true | 14:53:28 | action_time | 369ms | 14:53:32 |
| signal_change_close | false | 14:53:29 | current_state | open | 14:53:33 |
| signal_change_open | true | 14:53:32 | opening_time | 1003ms | 14:53:33 |
| open | true | 14:53:33 | current_state | closing | 14:53:36 |

**Table 5.4 – Workgroup 1 Clamp 2 Monitored Data**

| Raw Data | | | Computed Data | | |
|---|---|---|---|---|---|
| **Monitored Data** | **Value** | **Timestamp** | **Monitored Data** | **Value** | **Timestamp** |
| open | true | 14:53:27 | current_state | open | 14:53:28 |
| closed | false | 14:53:27 | current_state | closing | 14:53:28 |
| signal_change_open | false | 14:53:27 | action_time | 189ms | 14:53:28 |
| signal_change_close | false | 14:53:27 | current_state | closed | 14:53:29 |
| open | false | 14:53:28 | closing_time | 559ms | 14:53:29 |
| closed | true | 14:53:29 | current_state | opening | 14:53:32 |
| signal_change_close | true | 14:53:28 | action_time | 369ms | 14:53:32 |
| signal_change_close | false | 14:53:29 | current_state | open | 14:53:33 |
| signal_change_open | true | 14:53:32 | opening_time | 1127ms | 14:53:33 |
| open | true | 14:53:33 | current_state | closing | 14:53:36 |

A few aspects are worth noting regarding the data displayed in these tables. Firstly, all data is listed in the order it was processed by the agent, meaning that even if data arrives scrambled and out of order, the CMA/HLCMA is still capable of making sense of it using the associated timestamps and process it accordingly. Also, the agents do not require to be launched before the system's execution starts, meaning that there is no need to stop the assembly line to initiate the monitoring process.

**Table 5.5 - Workgroup 1 Clamp 3 Monitoring Data**

| Raw Data | | | Computed Data | | |
|---|---|---|---|---|---|
| **Monitored Data** | **Value** | **Timestamp** | **Monitored Data** | **Value** | **Timestamp** |
| signal_change_open | false | 14:53:42 | current_state | closed | 14:53:51 |
| signal_change_close | true | 14:53:50 | closing_time | 858ms | 14:53:51 |
| open | false | 14:53:50 | current_state | opening | 14:53:54 |
| closed | true | 14:53:51 | action_time | 369ms | 14:53:54 |
| signal_change_close | false | 14:53:51 | current_state | open | 14:53:55 |
| closed | false | 14:53:54 | opening_time | 1020ms | 14:53:55 |
| signal_change_open | true | 14:53:54 | current_state | closing | 14:53:58 |
| open | true | 14:53:55 | action_time | 228ms | 14:53:57 |
| signal_change_open | false | 14:53:55 | current_state | closed | 14:53:58 |
| open | false | 14:53:58 | closing_time | 882ms | 14:53:58 |

**Table 5.6 - Workgroup 1 Pin Monitored Data**

| Raw Data | | | Computed Data | | |
|---|---|---|---|---|---|
| **Monitored Data** | **Value** | **Timestamp** | **Monitored Data** | **Value** | **Timestamp** |
| open | true | 14:53:20 | current_state | open | 14:53:21 |
| closed | false | 14:53:20 | current_state | closing | 14:53:26 |
| signal_change_open | false | 14:53:20 | action_time | 66ms | 14:53:26 |
| signal_change_close | false | 14:53:20 | current_state | closed | 14:53:26 |
| open | false | 14:53:26 | closing_time | 145ms | 14:53:27 |
| closed | true | 14:53:26 | current_state | opening | 14:53:32 |
| signal_change_close | true | 14:53:26 | action_time | 137ms | 14:53:32 |
| signal_change_close | false | 14:53:27 | current_state | open | 14:53:32 |
| closed | false | 14:53:32 | opening_time | 255ms | 14:53:32 |
| signal_change_open | true | 14:53:32 | current_state | closing | 14:53:35 |

Another aspect worth highlighting is the difference between the values obtained for the pin and those obtained for the clamps. The pin acts at much faster speeds, both for the closing and opening motions, when compared to those of the clamps. However, even with such short timespans (some under one hundred milliseconds), the monitoring agents were still able to extract and process the associated data successfully.

Overall the monitoring data that resulted from the FANUC/VW cell's workgroup revealed itself to be fairly consistent in regards to the tendency that can be expected for the components in question. As an example, from a data set of one hundred and fifty *closing time* samples (without induced disturbances), the distribution of the collected data for Workgroup 1's Clamp 1 can be seen in Figure 5.8, with an associated standard deviation of 28.2501ms.

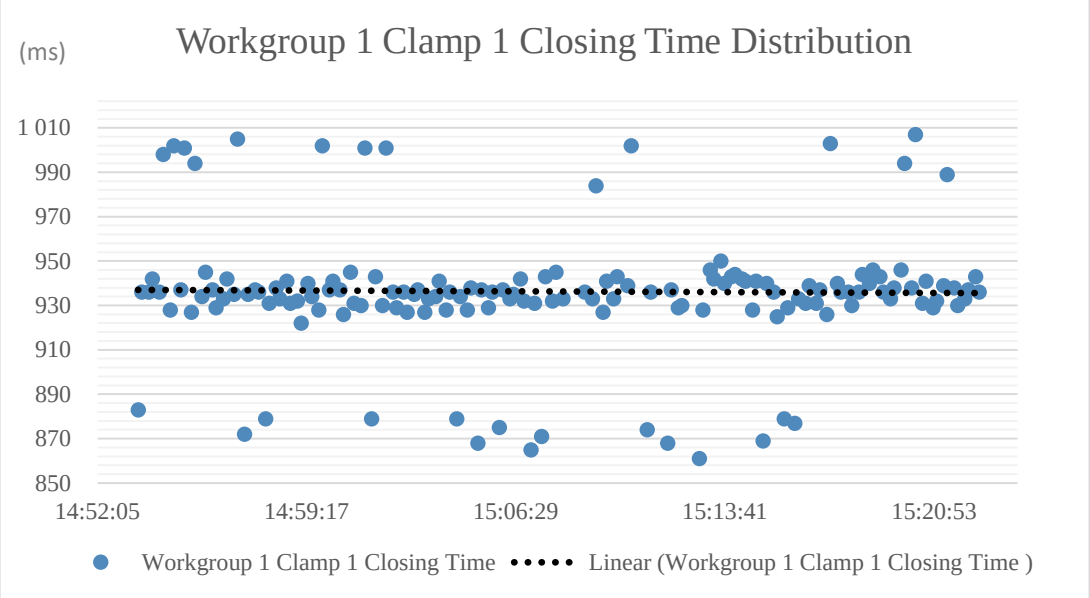


**Figure 5.8 - Workgroup1 Clamp 1 Closing Time Distribution Chart**

The distribution charts for clamp 2 and 3, as well the pin can be observed in Figure 5.9, Figure 5.10 and Figure 5.11, presenting standard deviations of 31.5993ms, 29.4465ms and 8.4027ms respectively.

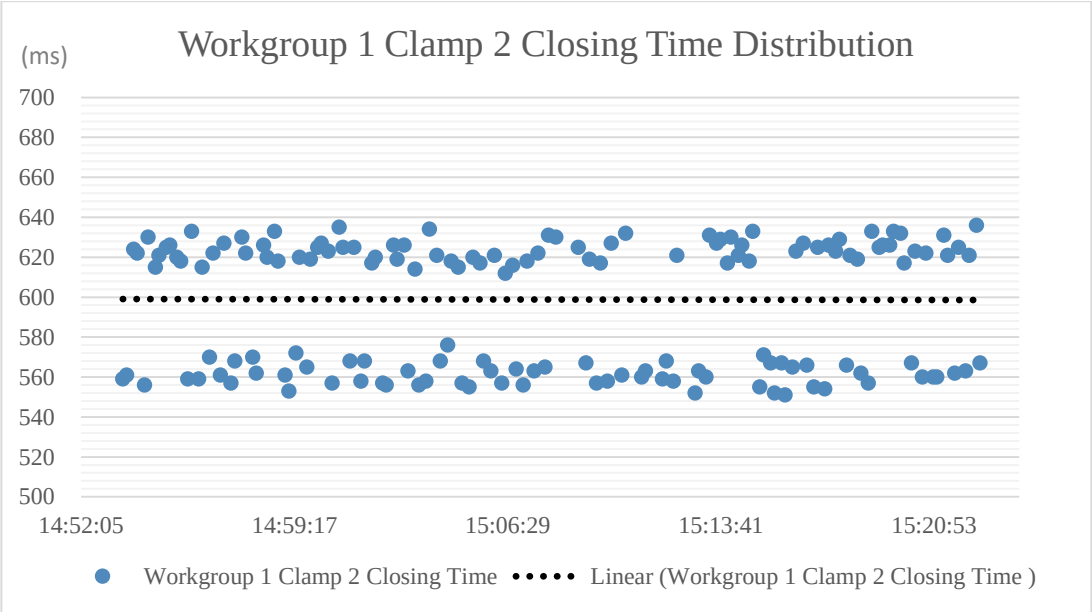


**Figure 5.9 - Workgroup 1 Clamp 2 Closing Time Distribution Chart**

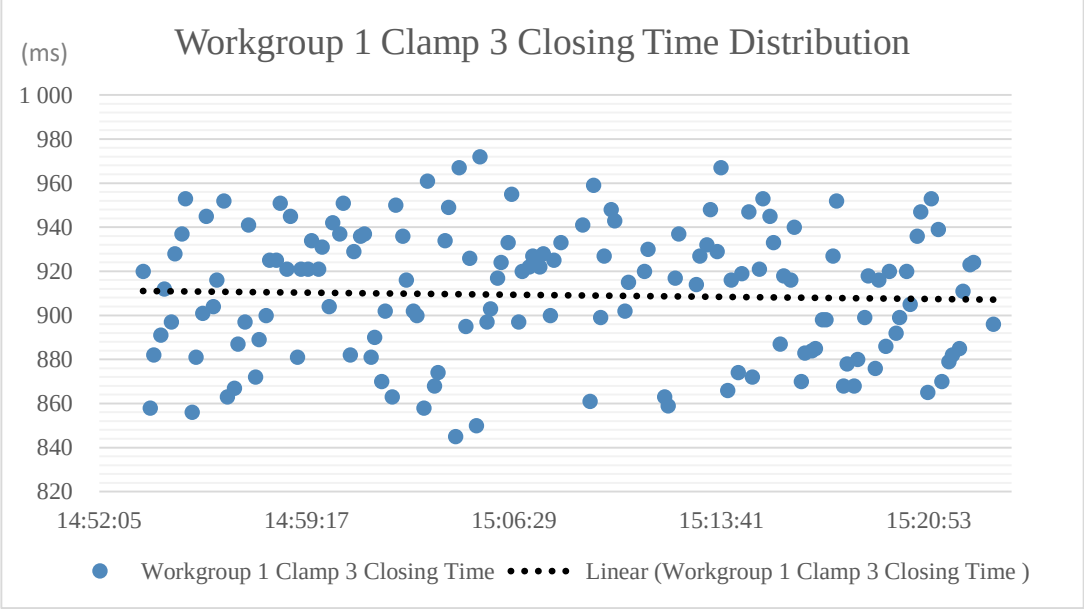


**Figure 5.10 - Workgroup 1 Clamp 3 Data Closing Time Distribution**

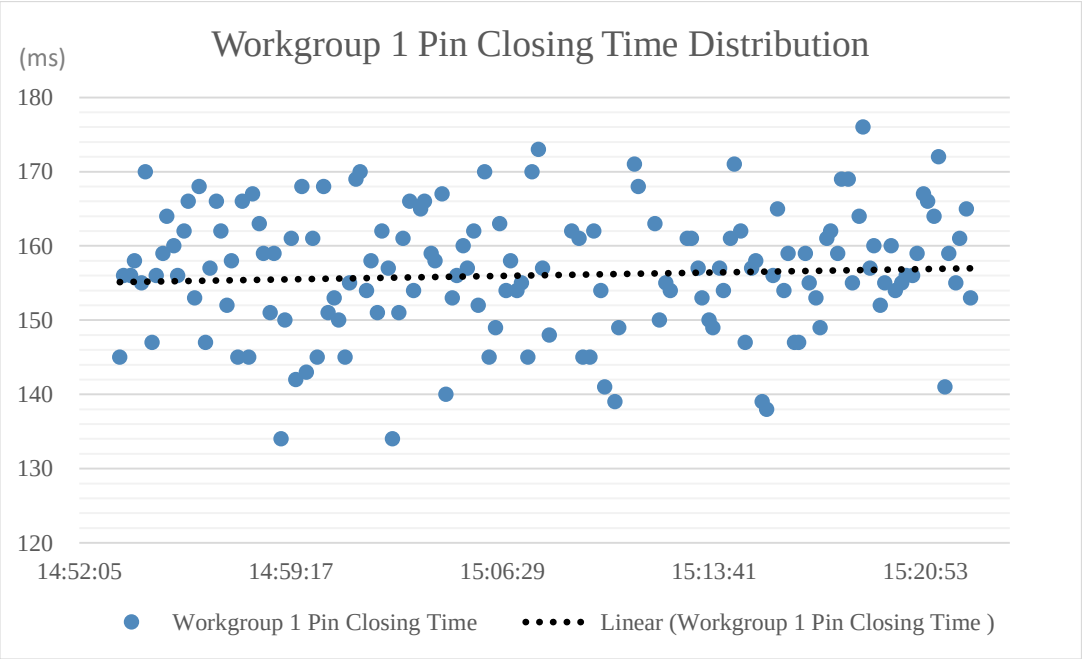


**Figure 5.11 - Workgroup 1 Pin Closing Time Distribution**

Similar results were obtained from the KUKA/Ford Cell, as suggested by the data referent to Clamp 1 displayed in Table 5.7. However, the clamps displayed a tendency to act slightly slower than the ones previously shown due to differences in the adjustments done to their respective air valves. The *closing time* data distribution in special can be seen in Figure 5.12.

**Table 5.7 - Workgroup 2 Clamp 1 Monitored Data**

| Raw Data | | | Computed Data | | |
|---|---|---|---|---|---|
| **Monitored Data** | **Value** | **Timestamp** | **Monitored Data** | **Value** | **Timestamp** |
| open | true | 14:53:35 | current_state | open | 14:53:37 |
| closed | false | 14:53:35 | current_state | closing | 14:53:42 |
| signal_change_open | false | 14:53:35 | action_time | 281ms | 14:53:42 |
| signal_change_close | false | 14:53:35 | current_state | closed | 14:53:43 |
| open | false | 14:53:42 | closing_time | 1311ms | 14:53:43 |
| signal_change_close | true | 14:53:42 | current_state | opening | 14:53:47 |
| closed | true | 14:53:43 | action_time | 796ms | 14:53:47 |
| signal_change_close | false | 14:53:43 | current_state | open | 14:53:48 |
| signal_change_open | true | 14:53:46 | opening_time | 1732ms | 14:53:48 |
| closed | false | 14:53:47 | current_state | closing | 14:53:54 |

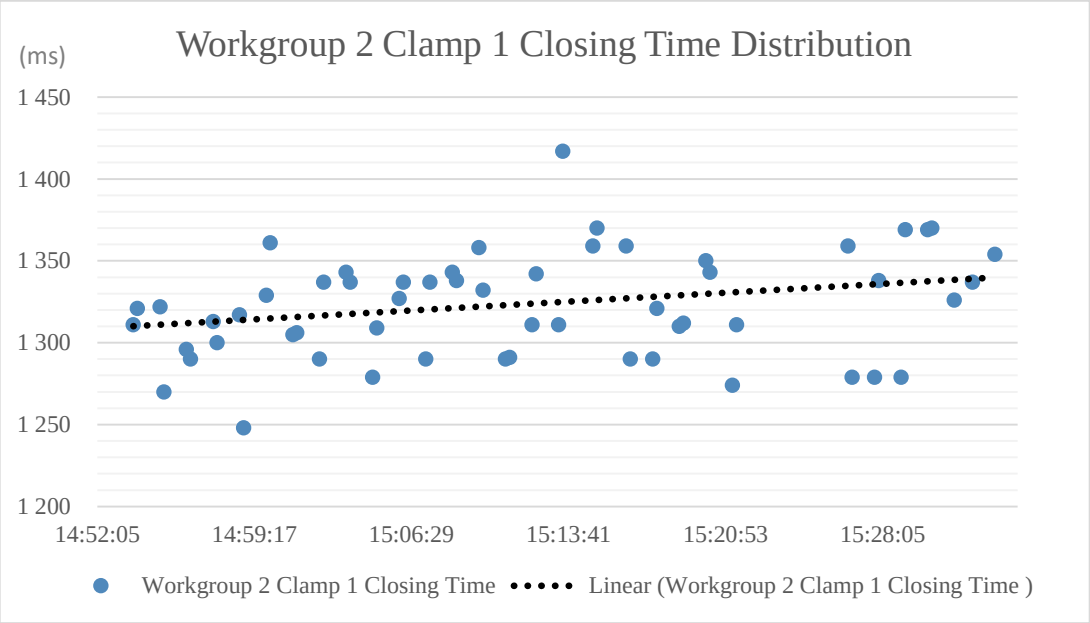


**Figure 5.12 - Workgroup 2 Clamp 1 Closing Time Distribution**

The slightly increasing tendency can be explained by the much smaller sample size used for this chart (fifty seven samples). Despite the fact that both cells were monitored during the same timespan, since the KUKA/Ford cell's process differed from the one executed its sister cell, resulting in a larger cycle time, it translated into a smaller sample size for this data.

Other types of data were also collected, such as information regarding security components, modes of operation, cycle times and part counts. Some of it can be seen in Figure 5.13.

| **Workgroup 2** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Safety Group** | | | **Scanner** | | | **Light Barrier** | | |
| **Data** | **Value** | **Timestamp** | **Data** | **Value** | **Timestamp** | **Data** | **Value** | **Timestamp** |
| safety_on | true | 14:53:17 | | | | | | |
| autoStart | true | 14:53:17 | interrupted | false | 14:53:39 | interrupted | true | 14:53:38 |
| manual | false | 14:53:17 | | | | | | |
| auto | true | 14:53:17 | | | | | | |
| starved | false | 14:53:17 | interrupted | true | 15:14:18 | interrupted | false | 15:14:18 |
| blocked | false | 14:53:17 | | | | | | |
| cycle_time | 73s | 14:53:17 | interrupted | false | 15:14:28 | interrupted | true | 15:14:28 |
| part_count | 1 | 14:53:17 | | | | | | |

**Figure 5.13 - Workgroup 2 Monitored Data Example**

As mentioned in Section 5.1.5, all the monitored data collected and generated the MAS module is then fed to the external processing network by the Monitoring Daemon. The network is then responsible for its processing, ultimately generating forecasts (see Figure 5.14) via processes such as trend analysis that can be used to display warnings that can be consulted by a human agent in the HMI, as illustrated in Figure 5.15.

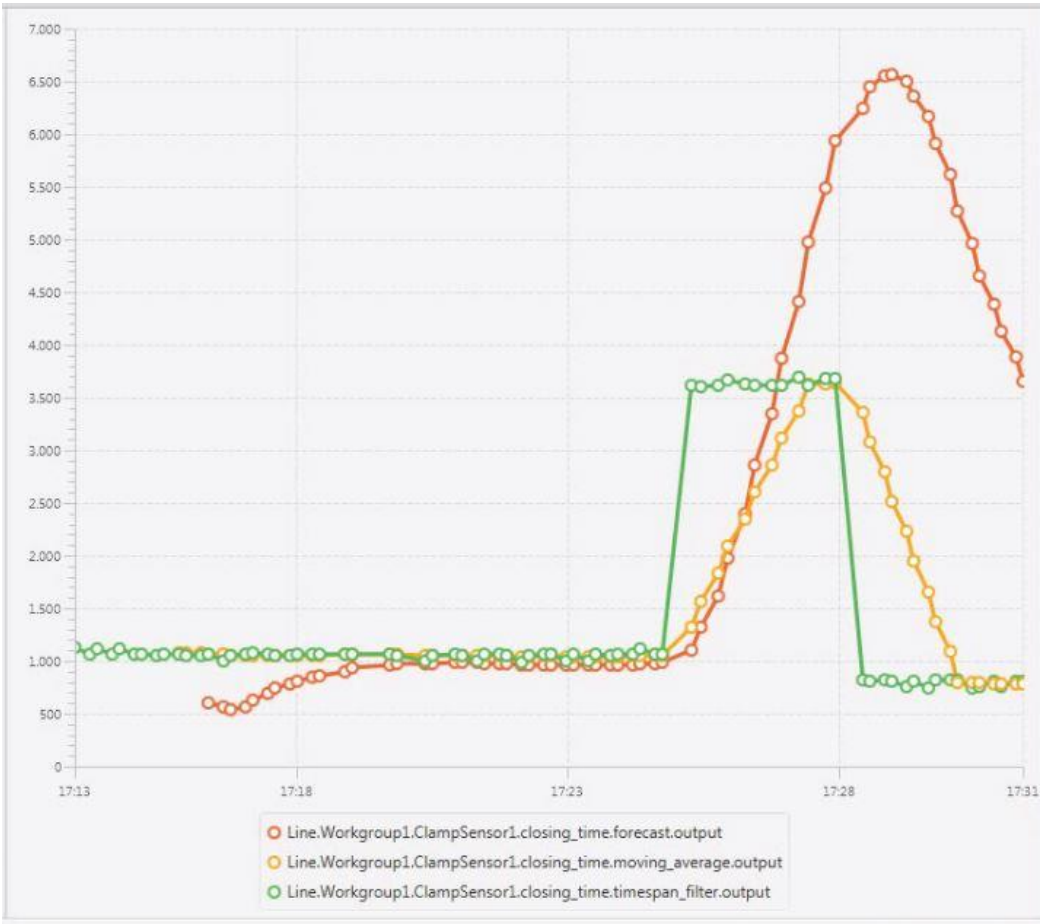


**Figure 5.14 - Processing Network - Data Analysis**

Figure 5.14 displays the data analysis performed by the processing network on the values of Workgroup 1 Clamp 1's *closing time* over an eighteen minute period. Three main values are shown, namely the forecast (orange), the moving average (yellow), based on a window encompassing the latest ten samples, and the actual data value as processed by the MAS module (green).

As it can be observed, at around 17:25 a disturbance was introduced in the system by tightening the clamp's air valve, therefore slowing its movement and increasing the time it takes to close, hence the sudden spike in the chart's value displayed in green. As the moving average increases, the forecast also starts increasing, eventually surpassing a pre-defined threshold (in this case 2000ms) and triggering an alert in the HMI, as shown in Figure 5.15.

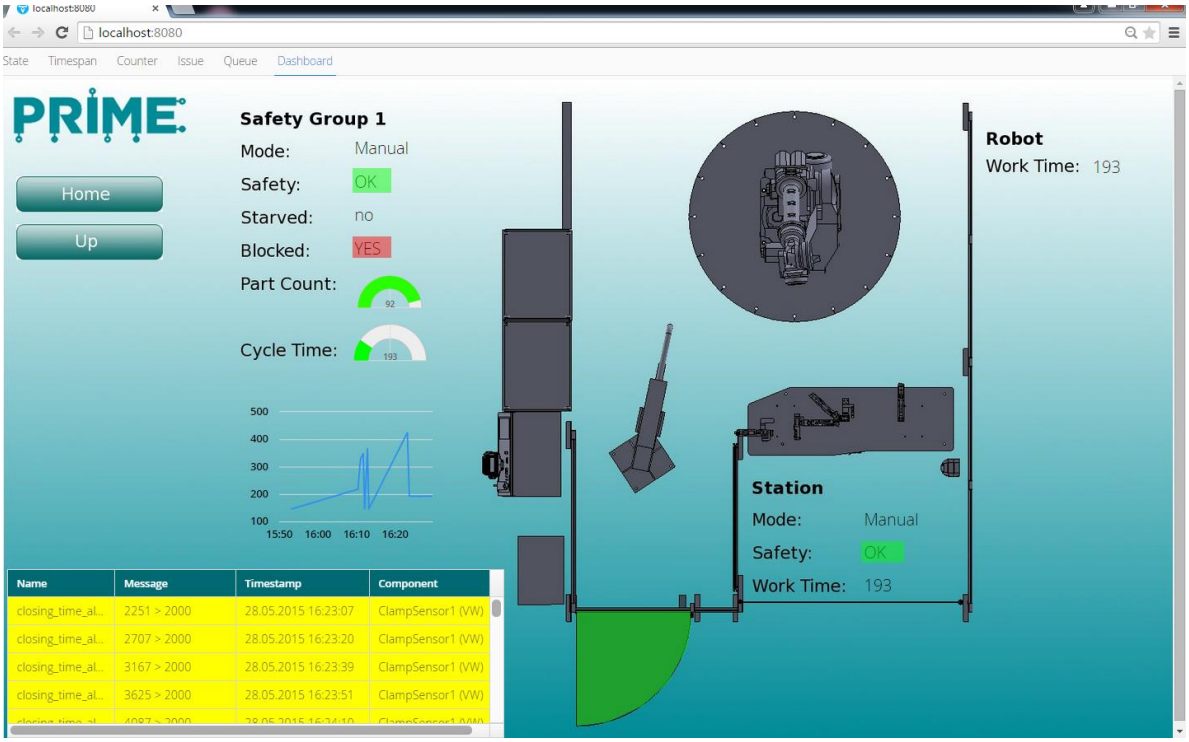


**Figure 5.15 - HMI Closing Time Alerts**

Lastly, some variation in the results was observed, in special when comparing results and trends from one cell to the ones obtained from the other, such as the case evidenced in Figure 5.12, which showcases an increasing trend for the clamp's closing time values. These slight variations can be due to the differences in the processes being executed in both cells, which in turn resulted in different cycle times and therefore uneven sample sizes, as previously explained. However, it should also be taken into account that these are real-world, physical systems, and as such other factors and disturbances should be weighed. Some examples would be for instance pressure variations in the air supply, physical friction and the deterioration of the components themselves, which can all affect the results to a certain extent.

Regardless, considering that the focus of the proposed architecture is purely data extraction, it is important to mention that the data obtained during the tests documented in this chapter was within the value range expected by the manufacturer. Overall, it can be said that the system behaved as expected, being able to successfully extract data from the physical components despite any possible real-time constraints, and use it to derive new, more complex information based upon the combination of said lower-level data values.

# 6

# Conclusion and Future Work

## 6.1. Conclusion

From the work developed and described in this document it is possible to conclude that performing monitoring in modular manufacturing systems whilst preserving their P&P capabilities and modularity is in fact achievable through the use of a distributed MAS-based approach.

Furthermore, the tests performed in the industrial robotic cells to validate the proposed architecture and its implementation have also shown that via the processing tasks executed by the monitoring agents, more complex knowledge can be extracted from the system through the combination of previously collected raw data and even other processed data values, in accordance with a pre-determined set of data description rules.

These tests validate the proposed solution's applicability in a real world industrial scenario, having required minimal effort from the manufacturer company to setup and apply to the existent cells, since in spite of both having different components and standards, no re-programming or structural changes or were required whatsoever, demonstrating the solution's technology independence.

The test results laid out in Chapter 5 suggest that the proposed architecture presents a robust knowledge extraction solution for real-world distributed manufacturing systems, being able to successfully perform data collection and processing despite the expected real-time constraints and disturbances commonly associated with real world applications involving physical hardware components.

## 6.2. Future Work

Future efforts should focus on extending the architecture, enabling more complex data analysis, such as data trends and tendencies, to be performed within the architecture without relying on external processing entities to do so.

Moreover, the proposed architecture and its implementation should also be applied and tested in other industrial cells with different processes and standards, further showcasing the solution as being independent from the underlying technology and requiring minimal effort from manufacturers to be applied in pre-existing shop floors.

# 7

# Bibliography


Abbas, H. a. (2014). Future SCADA challenges and the promising solution: the agent-based SCADA. *International Journal of Critical Infrastructures*, *10*(JANUARY 2014), 307. doi:10.1504/IJCIS.2014.066354

Alexakos, C., Georgoudakis, M., Kalogeras, a., Charatsis, K., Gialelis, J., & Koubias, S. (2006). A model for the extension of IEC 62264 down to the shop floor layer. *2006 IEEE International Workshop onFactory Communication Systems*, *62264*, 243–246. doi:10.1109/WFCS.2006.1704162

Antzoulatos, N., Rocha, A., Castro, E., de Silva, L., Santos, T., Ratchev, S., & Barata, J. (2015). Towards a Capability-based Framework for Reconfiguring Industrial Production Systems. *IFAC-PapersOnLine*, *48*(3), 2077–2082. doi:10.1016/j.ifacol.2015.06.395

Arai, T., Aiyama, Y., Maeda, Y., Sugi, M., & Ota, J. (2000). Agile Assembly System by "Plug and Produce." *CIRP Annals - Manufacturing Technology*, *49*(1), 1–4. doi:10.1016/S0007-8506(07)62883-2

Barata, J., & Onori, M. (2006). Evolvable Assembly and Exploiting Emergent Behaviour, 3353–3360.

Barata, J., Santana, P., & Onori, M. (2006). Evolvable Assembly Systems: A Development Roadmap. In *12th IFAC Symposium on Information Control Problems in Manufacturing*.

Barbosa, J. (2015). *Self-organized and evolvable holonic architecture for manufacturing control*. Université de Valenciennes et du Hainaut Cambrésis. Retrieved from https://tel.archives-ouvertes.fr/tel-01137643

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. *Developing Multi-Agent Systems with JADE*. doi:10.1002/9780470058411

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE–A FIPA-compliant agent framework. *Proceedings of PAAM*, 97–108. doi:10.1145/375735.376120

Bolwijn, P. T., & Kumpe, T. (1990). Manufacturing in the 1990s—Productivity, flexibility and innovation. *Long Range Planning*, *23*(4), 44–57. doi:10.1016/0024-6301(90)90151-S

Borgo, S., & Leitão, P. (2004). The Role of Foundational Ontologies in Manufacturing Domain Applications . In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, Agia Napa, Cyprus, October 25-29, 2004, Proceedings, Part I*. doi:10.1007/978-3-540-30468-5

Bunch, L., Breedy, M., Bradshaw, J. M., Carvalho, M., Suri, N., Uszok, A., … Marik, V. (2004). Software agents for process monitoring and notification. *Proceedings of the 2004 ACM Symposium on Applied Computing - SAC '04*, (September 2015), 94. doi:10.1145/967900.967921

ElMaraghy, H. a. (2006). Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, *17*(4), 261–276. doi:10.1007/s10696-006-9028-7

FIPA. (2000). FIPA Contract Net Interaction Protocol Specification. Retrieved from http://www.fipa.org/specs/fipa00029/

FIPA. (2002). FIPA Request Interaction Protocol Specification. Retrieved from http://www.fipa.org/specs/fipa00026/SC00026H.pdf

Frei, R., Barata, J., & Onori, M. (2007). Evolvable production systems context and implications. *IEEE International Symposium on Industrial Electronics*, (September 2015), 3233–3238. doi:10.1109/ISIE.2007.4375132

Hou, T.-H., Liu, W.-L., & Lin, L. (2003). Intelligent remote monitoring and diagnosis of manufacturing processes using an integrated approach of neural networks and rough sets. *Journal of Intelligent Manufacturing*, *14*, 239–253.

Jahromi, M. H. M. a, & Tavakkoli-Moghaddam, R. (2012). A novel 0-1 linear integer programming model for dynamic machine-tool selection and operation allocation in a flexible manufacturing system. *Journal of Manufacturing Systems*, *31*(2), 224–231. doi:10.1016/j.jmsy.2011.07.008

Koestler, A. (1967). *The Ghost in the Machine*. Hutchinson.

Koren, Y. (2006). Reconfigurable manufacturing systems and transformable factories. In A. I. Dashchenko (Ed.), *Reconfigurable Manufacturing Systems and Transformable Factories* (pp. 27–45). doi:10.1007/3-540-29397-3

Leitão, P., & Restivo, F. (2006). ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, *57*(2), 121–130. doi:10.1016/j.compind.2005.05.005

Marik, V., & McFarlane, D. (2005). Industrial Adoption of Agent-Based Technologies, (February), 27–35.

Merdan, M., Vallee, M., Lepuschitz, W., & Zoitl, A. (2011). Monitoring and diagnostics of industrial systems using automation agents. *International Journal of Production Research*, *49*(March 2015), 1497–1509. doi:10.1080/00207543.2010.526368

Monostori, L., Váncza, J., & Kumara, S. R. T. (2006). Agent-Based Systems for Manufacturing. *CIRP Annals - Manufacturing Technology*, *55*(2), 697–720. doi:10.1016/j.cirp.2006.10.004

Nagel, R. N., & Dove, R. (1991). *21st century manufacturing enterprise strategy: An Industry-Led View* (Volume 1.). Lehigh University Press.

Nagorny, K., Colombo, A. W., & Schmidtmann, U. (2012). A service- and multi-agent-oriented manufacturing automation architecture: An IEC 62264 level 2 compliant implementation. *Computers in Industry*, *63*(OCTOBER), 813–823. doi:10.1016/j.compind.2012.08.003

Onori, M., Lohse, N., Barata, J., & Hanisch, C. (2012). The IDEAS project: plug & produce at shop-floor level. *Assembly Automation*, *32*, 124–134. doi:10.1108/01445151211212280

Orio, G. Di, Rocha, A., Ribeiro, L., & Barata, J. (2015). The PRIME Semantic Language : Plug and Produce in Standard-based Manufacturing Production Systems. In *The International Conference on Flexible Automation and Intelligent Manufacturing 2015 (FAIM'15), At University of Wolverhampton, UK*.

Ouelhadj, D., Hanachi, C., & Bouzouia, B. (2000). Multi-Agent Architecture for Distributed Monitoring in Flexible Manufacturing Systems (FMS). *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*, (April).

Ribeiro, L., & Barata, J. (2011). Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms. *Computers in Industry*, *62*(7), 639–659. doi:10.1016/j.compind.2011.03.001

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

Rocha, A. D., Peres, R., & Barata, J. (2015). An agent based monitoring architecture for plug and produce based manufacturing systems. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on* (pp. 1318–1323). doi:10.1109/INDIN.2015.7281926

Rocha, A., Orio, G. Di, Barata, J., Electrotécnica, D. D. E., De, F., Antzoulatos, N., … Ribeiro, L. (2015). An Agent Based Framework to Support Plug And Produce. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 1318–1323.

Russel, S. J., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. *Artificial Intelligence* (Vol. 72). Pearson Education, Inc. doi:10.1017/S0269888900007724

Santos, T. (2015). *Infraestrutura para gestão de sistemas de automação híbridos baseados em controladores lógicos programáveis e agentes*. Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia.

Ueda, K. (1992). A Concept for Bionic Manufacturing Systems Based on DNA-type Information, 853–863. Retrieved from http://dl.acm.org/citation.cfm?id=647327.721953

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., & Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, *37*, 255–274. doi:10.1016/S0166-3615(98)00102-X

Wooldridge, M. J. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd.

Yu, J., & Xi, L. (2009). A neural network ensemble-based model for on-line monitoring and diagnosis of out-of-control signals in multivariate manufacturing processes. *Expert Systems with Applications*, *36*(1), 909–921. doi:10.1016/j.eswa.2007.10.003

Yu, J., Xi, L., & Zhou, X. (2008). Intelligent monitoring and diagnosis of manufacturing processes using an integrated approach of KBANN and GA. *Computers in Industry*, *59*(5), 489–501. doi:10.1016/j.compind.2007.12.005