Pedro Miguel Lima Monteiro
Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Machine Diagnosis Based on Artificial Immune Systems

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

Orientador: José António Barata de Oliveira, Professor Doutor, FCT-UNL
Co-orientador: André Dionísio Bettencourt da Silva Parreira Rocha, MSc, UNINOVA/CTS

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro de 2015

Pedro Miguel Lima Monteiro
Licensed in Electrical and Computer Engineering Sciences

Machine Diagnosis Based on Artificial Immune Systems

Dissertation for obtaining Master's Degree in Electrical and Computer Engineering

Supervisor: José António Barata de Oliveira, Professor Doutor, FCT-UNL
Co-supervisor: André Dionísio Bettencourt da Silva Parreira Rocha, MSc, UNINOVA/CTS

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September 2015

Machine Diagnosis Based on Artificial Immune Systems

To Juliana and my Family

Acknowledgments

Under this section, I would like to express my sincere gratitude to all those who contributed for the completion of this dissertation.

Firstly and foremost, I would like to thank my advisor, PhD Professor José Barata, for the opportunity of working in this subject, which is of major interest to me. This opportunity allowed me to develop myself as both person and academically speaking.

I would also like to express my deepest regards for my co-advisor, PhD Student André Rocha. For all the support, interest and dedication showed. A grand part of the achieved success is due to him.

This section would not be complete without a sincere thank you to Mafalda Parreira, whose company and assistance where of terrible importance for the conclusion of this document.

Furthermore, I would like to thank Ricardo Peres for his sympathy and friendship, in integrating myself in the workgroup.

I would also like to thank Hugo Rodrigues for all the friendship throughout the years, a friend in the true sense of the word.

A heartfelt and grateful thank you to Juliana, for enduring my temper in the hardest times, for being a consolation when things went sorrow, for all the help and assistance you gave me. To you, for all you are to me, thank you.

Finally, I would like to express my deepest gratitude to my family, namely, my parents, whose sacrifice and utterly dedication throughout these years made it possible for this document to come to light; also, to my aunt, Dina, for the relentless support and for accompanying my academic evolution; to my uncles, "Caco" and Armindo, for being a child's role-model.

To all of you, without whom, I would not be who I am and what I have achieved,

My deepest gratitude and a great Thank You!

Abstract

Nowadays, many of the manufactory and industrial system has a diagnosis system on top of it, responsible for ensuring the lifetime of the system itself. It achieves this by performing both diagnosis and error recovery procedures in real production time, on each of the individual parts of the system.

There are many paradigms currently being used for diagnosis. However, they still fail to answer all the requirements imposed by the enterprises making it necessary for a different approach to take place. This happens mostly on the error recovery paradigms since the great diversity that is nowadays present in the industrial environment makes it highly unlikely for every single error to be fixed under a real time, no production stop, perspective.

This work proposes a still relatively unknown paradigm to manufactory. The Artificial Immune Systems (AIS), which relies on bio-inspired algorithms, comes as a valid alternative to the ones currently being used.

The proposed work is a multi-agent architecture that establishes the Artificial Immune Systems, based on bio-inspired algorithms. The main goal of this architecture is to solve for a resolution to the error currently detected by the system.

The proposed architecture was tested using two different simulation environment, each meant to prove different points of views, using different tests.

These tests will determine if, as the research suggests, this paradigm is a promising alternative for the industrial environment. It will also define what should be done to improve the current architecture and if it should be applied in a decentralised system.

Keywords: Diagnosis, Artificial Immune Systems, Bio-Inspired Algorithms, Multi-Agent System, Manufacturing Systems.

Sumário

Hoje em dia, muitos dos sistema industriais e/ou de manufactura possuem um sistema de diagnóstico por ele responsável, no que ao tempo de vida diz respeito, através da realização de manutenção tanto por diagnóstico como por procedimentos de recuperação de erros em tempo real, em cada uma das peças que constituem o todo.

Existem diversos paradigmas que são utilizados para diagnóstico. No entanto, estes ainda falham em alguns dos requerimentos impostos pelas empresas tornando necessária uma diferente aproximação, nomeadamente, a nivel de recuperação de erros visto que a maior diversidade que se encontra presente no ambiente industrial torna altamente improvável que todos os erros sejam reparados numa perspectiva de tempo real.

Este documento propõe um paradigma, ainda desconhecido para a manufactura, baseado em algoritmos bio-inspirados como uma alternativa válida àqueles usados actualmente, os Sistemas de Imunidade Artificial.

O trabalho proposto baseia-se numa arquitectura multi-agente que segue este paradigma, baseado em algoritmos bio-inspirados. O principal objectivo desta arquitectura é procurar uma resolução para o erro detectado no sistema.

A arquitectura proposta foi testada usando dois ambientes de simulação diferentes, cada um com um objectivo diferente em mente.

Estes testes irão determinar se este paradigma é realmente tão promissor e adequado para um ambiente industrial como a pesquisa efectuada sugere. Irão também definir o que pode ser

feito para melhorar a arquitectura desenvolvida e também se esta deve ser aplicada num sistema descentralizado, como é pretendido, tendo em vista os melhores resultados, da perspectiva do desempenho.

Palavras-Chave: Diagnóstico, Sistemas Imunes Artificiais, Algoritmos Bio-Inspirados, Sistemas Multi-Agente, Sistemas de Manufactura.

Table of Contents

Table of Figures

Table of Tables

Acronyms

ADACOR	-	Adaptive Holonic Control Architecture
AIS	-	Artificial Immune Systems
AMS	-	Agile Manufacturing Systems
ANN	-	Artificial Neural Networks
AREIB	-	AchieveREInitiator Behaviour
ARERB	-	AchieveREResponder Behaviour
BCA	-	B Cell Agent
CBM	-	Condition Based Maintenance
CPA	-	Cure Provider Agent
CS	-	Clonal Selection
DA	-	Diagnosis Agent
EAS	-	Evolvable Assembly Systems
EPS	-	Evolvable Production Systems
FDI	-	Fault Detection and Identification

FIS	-	Fuzzy Inference System
FIPA	.	The Foundation for Intelligent, Physical Agents
FMS	-	Flexible Manufacturing Systems
GDA	-	Grouped Diagnosis Agent
HMS	-	Holonic Manufacturing Systems
I/O	-	Inputs/Outputs
JADE	-	Java Agent Development Framework
LC	-	Limit Checking
MAS	-	Multi Agent Systems
NM	-	Network Model
NS	.	Negative Selection
OSB	-	One Shot Behaviour
PB	-	Parallel Behaviour
PLC	-	Programmable Logic Controller
PM	-	Predictive Maintenance
PROSA	-	Product-Resource-Order-Staff Architecture
QM	-	Quantitative Methods
QLM	-	Qualitative Methods
RMS	-	Reconfigurable Manufacturing Systems
SB	-	Simple Behaviour
SI	-	Swarm Intelligence
SOA	-	Service Oriented Architecture
SPS	-	Standard Production Systems

TB - Ticker Behaviour

1

Introduction

1.1 The Problem

Usually, a simple rule based diagnosis system, on top of the production one, sufficed to perform diagnosis. However, as the costumers demand grew wider, they started to demand a more and more diversified and customized product. In order to satisfy the demand, the enterprises needed to change their production systems as well. This change made it impossible for a rule based diagnosis system to answer all the possible new errors in this highly customized environment.

The diagnosis paradigms that existed were rule-based and lacked the concept of evolution. This resolving methods were no match for the new, unforeseen, unrelated errors. Furthermore, these type of diagnosis paradigms were centralized, thus resulting in a lesser effective diagnostic. These paradigms became obsolete.

Several of this paradigms were further developed in order to face the needs of the industry. However, they still lack the evolutionary concepts required to face the new errors that may occur in the system. Despite there are already some decentralized implementations of this kind, they still lack to give the needed importance to this matter.

One of the mentioned paradigms may present itself as a possible solution to this matter. The Artificial Immune System (AIS), which rely on bio-inspired algorithms as the resolving method for the errors the system may face. This work proposes a multi-agent architecture that

offers decentralised diagnosis, along with evolvable procedures that aims at resolving any error occurred in the system.

1.2 Research Questions and Hypothesis

Given the above problem, and taking into account that diagnosis is a fundamental part of production systems, some questions arise:

1. Would it be possible to perform distributed diagnosis on a Multi-Agent System (MAS) using AIS in order to provide a better lifetime for the production systems?
2. Would it be possible for a distributed MAS using AIS as a diagnosing enabler to evolve and learn with its mistakes?

Having researched, studied and analysed both AIS and MAS, this work's author propose two hypothesis to answer the above questions.

Firstly and foremost, the author propose an MAS architecture that benefits of this paradigms' distributed capabilities, to perform diagnosis in a decentralised way. On the other hand, the author propose the use of AIS algorithms, such as Network Model (NM) as a diagnosing and evolution enabler, allowing for the system to diagnose itself and learn with its mistakes.

1.3 Motivation

The existing alternatives don't allow a system to completely diagnose and recover from every error that may occur. This happens because our world is too random, which makes it impossible to predict every hypothetical situation that may or may not occur.

Therefore, for a better diagnose, the system in charge of it needs to adapt, evolve and learn with the errors themselves. The best approach for such a system is still to be determined. However, there are several systems, in Nature, that resemble such an approach.

One of those is the Human Immune System, which uses its agents, B Cells, to track down and eliminate all the organisms that endanger our life. The process through which the B Cells operate is complex and vast. Theoretically, the process offers valid resolutions for the organisms that constantly enter and exit our body.

An engineering approach to the B Cells processes applied to manufactory may lead to a new Era on diagnosis. An Era in which the system does not predict, rather reacts to the changes made to itself, constantly evolving and learning.

Artificial Immune Systems offer just that: an evolvable, capable of learning, memorizing and reactionary system that aims at resolving every single error in a manufactory environment.

1.4 Accomplished Work

This work proposes a multi-agent based architecture which aims at diagnosing and recovering errors for any type of system. To do so, one must, beforehand, provide drivers' files with the associated part information concerning its known errors.

The designed architecture consists of two layers, the cloud, and the shop floor layer:

- The first contains the cures databases and the agents that abstract them.
- The latter contains the agents that communicate with the ones that abstract the hardware. These agents are constantly looking for an unrecognized procedure in order to trigger a recovery event. They can be grouped into more abstract entities that englobe several of the low level ones. Both entities are able to trigger the recovery event, abstracted by another type of agents.

Concerning the cloud layer, it works like a cloud, decentralizing the memorizing mechanism of the system. This layer only communicates with the diagnosis and recovery layer through the entities that abstract the recovery event. The cloud is responsible for storing and providing all the resolutions the system knows.

To what the shop floor layer may concern, it possesses the entities that are constantly analysing the system. This analysis occurs in a decentralised manner, since each part of the system is abstracted by its own entity. If a new error is found, and the error is unrecognised, the recovery event is triggered, and the algorithms start processing.

Whenever a new part is plugged in the system, its drivers are immediately processed by the agent that will diagnose it. All the errors known by the system will be available for this part since the agent that abstracts it will be able to contact the evolution layer if an error occurs.

After being designed, the architecture was implemented using the JADE (Java Agent Development Framework) framework. JADE is a Java-based middleware that eases multi-agent system development. One of the main system characteristics is the FIPA-Compliant agent-behaviours. All the agents present in the architecture fulfil the FIPA protocols when communicating.

1.5 Algorithms

Besides the proposed architecture, this work is based on the study, implementation and testing of the algorithms used. Currently, there are several bio-inspired algorithms that can be used under the scope of AIS.

After some research on the matter and using as criteria the usefulness and implementation time, three algorithms were selected for further research and development: Negative Selection (NS), Clonal Selection (CS), also known as CLONALG, and NM. Each one of the algorithms uses a different mechanism to solve the error given as input. Consequently, each one as advantages and disadvantages towards each other:

- The Negative Selection algorithm is trained with a given amount of strings, which will be the base of comparison in which it will operate. If somehow a different string is ever introduced, it will be spotted by bit-comparison. The algorithm then starts a pseudo random mutation mechanism of the strings used for training to try to find a resolution.
- The Clonal Selection algorithm is also trained with a given amount of strings. When a string that is not equal to any of them enters the system, the algorithm calculates the affinity of the initial population towards the error. The highest affinity ones suffer cloning and mutation. The algorithm then calculates the mutated strings affinity towards the error. The highest affinity ones suffer more cloning. This process is repeated until a resolution is found.
- The Network Model algorithm, like the others, is trained with a given amount of strings. This strings will provide the algorithm with its initial population. If an error occurs, these entities will test their affinity towards the error and their neighbours. The highest affinity ones are cloned and mutated. The process keeps repeating itself until a resolution is found.

The algorithms were tested using a dedicated machine in order to output better performance results. The tests had as outputting variables: the time lapse between the introduction of the error in the system and the founding of the resolution; and the number of entities launched. The results obtained were analysed using a Fuzzy Inference System (FIS) in order to, if you may, rank the algorithms according to its capability of getting a resolution for the error based on the above mentioned parameters.

There is no limitation as to where this algorithms might be used for, since its versatility allows them to be used in literally every type of system. Hence, the results hereby presented should only be considered under a JADE framework supported scope.

1.6 Major Contributions

The accomplished work offers an architecture that allows for the system to recover from any error that does not require replacement of any of the physical parts, which, for obvious reasons, cannot be dealt with using only software.

The ability of dynamically search for a resolution for any given error transforms the diagnosis mechanism, from a predictive, pre-emptive system, to a reactionary and evolutionary one. This system is able to learn with its mistakes, since the recovery event only occurs once for every unrecognised error. This learning capability is done by storing, in a database, all the errors that occurred in its lifetime.

Moreover, due to its distributed architecture, especially in the evolution layer, where the databases lie, it is now possible to perform diagnostic in a distributed way. This happens because, whenever the need arises, the diagnostic and recovery layer will communicate with all the elements in the cloud. Adding to this, each abstracting entity of each part of the system is capable of performing diagnosis by itself, with no need for a centralized super computer with all the data.

2

State of the Art

Originally, manufactory and industrial systems were thought and built bearing in mind a mass production goal. This means they were thought to output a small variety of products, in as little time as possible. They were not thought to output several, different products.

Since the products were not that much distinguishable between themselves, a simple rule based system was enough for the enterprises to produce them. Same goes for diagnosis, which was relatively easy to perform since a bunch of rules would suffice to recover pretty much every error the system could have.

As the market demands altered, it began to demand highly customizable and equal variety of products. Therefore, a new approach to the shop floor was in order for the enterprises for a better response to the customers' needs. This change in the production paradigm led to, at the time, yet unveiled research fields such as evolution, distributed processing, agile systems, etc.

As the production paradigms change, so must the diagnostics since these must operate on top of the first. Nowadays, the research is focused on both evolvable and distributed systems, along with plug-and-produce capabilities. Hence, to answer the demands of such a system, along with the enterprises requirements, the diagnostic paradigms must, as well, operate on a distributed and evolvable environment.

2.1 Manufacturing Paradigms

2.1.1 Traditional Production System

Since this has become a now relatively obsolete technology, it will not be approached in as much detail as the remaining paradigms. Suffice to say these systems (Groover, 2007) paradigm was the first of its kind to appear, thus leading to the first developments when it comes to industry robotics.

The first ones started out as simple, centralized systems, with very few diagnostic capabilities and almost none error recovery which did not comply with the, already, growing needs of the worldwide enterprises.

(Buzacott & Shanthikumar, 1993) covers a wide range of the so called traditional production and manufacturing systems such as flow lines, job shops, etc. and gives an overview on such systems, explaining, detailing and modelling.

(Koren, 1983) offers a review on computer controlled manufacturing systems, exploring this paradigm and offering solutions based on it. This was one of the early stages production paradigms, since it was when computers began to be introduced in the industrial environment.

2.1.2 Agile Manufacturing Systems

Agile Manufacturing Systems (AMS) got their name for their ability to manufacture a variety of components, with a low cost and in a short period of time. Therefore it should have as main characteristics: simple, flexible, reconfigurable and responsive to market changes (Da Silveira, Borenstein, & Fogliatto, 2001).

However, an AMS is not all about being flexible and responsive to the current customers' demands. It also requires a strong adaptive capability to be able to respond to future changes the industry may endure (Gunasekaran, 2001), due to market changes.

That being said, one may infer that the main concern that lead to this paradigm was change. In response to the market changes that kept occurring, manufacturing was forced to walk down the path of change and adjustment that, despite gradual, became a decisive factor for this industry (Gunasekaran, 2001).

Given the exponential growth of the amount of research this technology suffered in its early years, (Gunasekaran, 2001) proposed a framework for more consensus on both research and

development, where a classification of the previous research is based on the following criteria: strategies, technologies, systems and people.

In (Sanchez & Nagi, 2001), a classification scheme for the most common implementations is established. The authors of said paper describe the AMS as "new, post-mass-production system for the creation and distribution of goods and services". A number of different research topics is admitted, in order to provide a broader concept of the AMS whilst proving their usability in all the subjects regarding manufacturing.

In (Elkins, Huang, & Alden, 2004), an interesting perspective on AMS is given in the automotive industry context. The authors sustain that AMS applies "to the automotive industry's goals of operating profitably, and sensing and responding effectively to changing demand trends".

2.1.3 Reconfigurable Manufacturing Systems

For a better answer to the high-frequency market changes driven by global competition that manufacturing companies faced, a need for a more versatile paradigm than the AMS emerged. Without losing its agility, the new paradigm should be able to be easily reconfigurable, dealing with cost-effectiveness and quick reaction to market changes – Reconfigurable Manufacturing Systems (RMS). However, some authors (Zhang, Liu, Gong, & Huang, 2006) consider this paradigm to be a sub-paradigm of AMS, thus disregarding this as an actual paradigm.

In (Sirca, 2008), a definition for these type of systems is given, where they are described as systems "designed at the outset for rapid change in its structure, as well as its hardware and software, in order to quickly adjust its production capacity and functionality".

Consequently, a new manufacturing approach that could be able to combine both high throughput and high flexibility was in order for the enterprises to apply this new paradigm (Mostafa G. Mehrabi, Ulsoy, & Koren, 2000). According to the same authors, this was achieved by designing a system and respective machines in an adjustable structure. This structure would allow the system to scale. It was also achieved by designing a different manufacturing system for each product family part, with customized flexibility for producing all parts of that given product family.

For the establishment of this paradigm, some sort of design consensus was in order. (Koren & Shpitalni, 2010) offers just that, by establishing a method to evaluate and classify the configuration of the system, presenting some data on integrated RMS practical configurations.

In (Landers, Min, & Koren, 2001), a set of three different Reconfigurable Machine Tools (RMT) is reviewed: part change, feature change, cycle time change. The same authors establish both control and mechanical requirements for the RMT, such as kinematic viability and structural stiffness. The same authors propose an example of this type of systems design.

In (ElMaraghy, 2006), an overview on the different types of both flexible and reconfigurable manufacturing systems is given, whilst offering various descriptions, ranging from manufacturing system configuration to flexibility and configurability.

As in any other paradigm, some implementations result in more effective production than others. (M. G. Mehrabi, Ulsoy, Koren, & Heytler, 2002) enacted a survey specially designed for obtaining "a current assessment of flexible machining systems ... identify the potential benefits of, and key enabling technologies needed for reconfigurable machining systems". The same authors define five enabling technologies for RMS: high-speed machining, modular machine tools, open architecture, training of operators and education of engineers.

In (Bi, Lang, Shen, & Wang, 2008), a state of the art for RMS is given. The authors defend some requirements different from those seen above such as short lead-time, low and fluctuating volumes. The same authors propose strategies to deal with each of the requirements they enumerate along with both configuration and system design for RMS.

2.1.4 Holonic Manufacturing Systems

The next step towards a better, more productive manufacturing system involved the cooperation of autonomous, completely functional entities, each one of them having their own goals, which could even raise conflict with each other – Holonic Manufacturing Systems (HMS).

In (Leitão & Restivo, 2006), a Holonic architecture is developed, this one aiming for agile and adaptive manufacturing control – ADACOR (ADAptive Holonic Control aRchitecture). This architecture is based on both autonomous and cooperative holons. By holon, one may interpret "a manufacturing component that can be either a physical resource (…) or a logic entity (…). This architecture defines four manufacturing holon classes, product, task, operational and supervisor holons".

(Van Brussel, Wyns, Valckenaers, Bongaerts, & Peeters, 1998) proposes a reference architecture for Holonic manufacturing systems – PROSA (Product-Resource-Order-Staff Architecture). This architecture encompasses the creation of three distinct holons: Order Holon, Product Holon, Resource Holon, each with a different set of capabilities and responsibilities.

(Leitão & Restivo, 2008) proposes a Holonic approach to dynamic manufacturing scheduling, proposing an architecture with different holons, each with a set of duties and rights towards the global system. These holons possess decision-making capabilities, which makes them able to perform both control and scheduling functions. These holons are ADACOR based.

In (Valckenaers & Van Brussel, 2005), a "sub-paradigm", if you may, of HMS is proposed – Holonic Manufacturing Execution Systems. According to the authors, this paradigm is designed as an instance of the PROSA architecture, previously referenced in this document, and "augmented with coordination and control mechanisms inspired by natural systems".

2.1.5 Evolvable Production Systems

Despite the community commonly understands Evolvable Production Systems (EPS) as just another method of RMS, a "sub-paradigm", if you may, this is hardly the case.

(Mauro Onori, Barata, & Frei, 2006) offer a clarification on the difference between EPS and RMS. RMS works on further development of flexibility, from the starting point. However, its focus was still too limited in time, concerning both current products and company aspects. For a better understanding of the differences between each other, the authors focused these three aspects:

- "Main focus" – RMS focus on the re-configurability of the geometric setup, making this process not necessarily automatic, whereas EPS adapts the system components by capturing the emergent properties.
- "Development trigger issue" – RMS uses current product features for further development whilst EPS focuses on re-engineering the assembly system.
- "Modularity level" – RMS conventional subdivision results "in coarse granularity". EPS applies lower level modularity based on process-level characteristics.

The same authors also explain both the System and the Control concept behind EPS.

Despite the manufacturing technologies current path towards distributed systems makes it harder for a consensus on frameworks, an attempt was still made in this case (Lohse, Ratchev, & Barata, 2006). The authors analyse different modular assembly systems within the current paradigm, specifying requirements for modular assembly systems through the use of suitable ontological models. The paper also proposes a design framework, describing it as an "intensive process" as it was for the EUPASS project (Rütten et al., 2003) .Moreover, the paper defines an Ontology, onto MAS, to support the design decision making.

(M. Onori, Alsterman, & Barata, 2005) proposes an architecture development approach for EPS, where it specifies obligatory points to be covered when developing effective assembly systems. As it deals with a distributed system, both the individual and the community architecture must be defined. In the first, a number of functionalities are explained in order to be easily implemented in any individual module. In the latter, the main concern is communication and the way it is organized since it is through it every single individual interact. Moreover, a step by step approach is given on the suggested architecture.

In (J. Barata, Camarinha-Matos, & Onori, 2005), a control approach through Multi Agent to EPS is proposed. The authors attribute major importance to the emergence concept in EPS and highlight the importance and need of two different architectures, one individual and another for the community. It proposes the COBASA (Jose Barata & Camarinha-Matos, 2003), (Kordic, 2006) architecture referring the needed concepts "contracts, skills, credits, among others" are supported by ontologies. Moreover, it introduces the concept of "simple skill" and "complex skill", where a complex one represents a group of simple in a much simplified explanation.

Given the wide range of everyday new systems, implementations and frameworks for this type of systems, it became necessary to validate them. (Lohse, Hirani, Ratchev, & Turitto, 2005) proposes an ontology for both definition and validation of EPS, which is used to "describe and guide the assembly process specification for both new assembly system configuration as well as reconfigurations of existing assembly systems".

Since the Agent concept involves human integration, the manufacturing research began to concern on the human interaction with the manufactory environment. (Frei, Ribeiro, Barata, & Semere, 2007) elaborates on the possible EPS scenario of a human-robot interaction inspired by the concept of intelligent houses. The authors also refer the importance of both diagnosis and self-organization roles on EPS.

Still concerning the self-organization role on EPS, and considering the modular aspects of these paradigm, (Frei, Di Marzo Serugendo, & Barata, 2008) propose a designing for self-organization on EPS. They explanation comes as follows: "given a specified product order provided in input, the system's modules spontaneously select each other (…) and their position in the assembly system layout". The authors also identify two principal characteristics for an EPS to be considered as self-organizing: "modules self-organize to produce an appropriate layout for the assembly and (…) the assembly system as whole self-adapts to production conditions".

In (Frei, Ferreira, Di Marzo Serugendo, & Barata, 2009), an architecture for self-managing EPS is proposed, where a set of pre-determined classes of agents is established, along

with communication protocols, which use XML for serializing. A case study is also presented, in order to better test and conclude on this architecture. Some rules are also enumerated for failure events and grouped under the following main characteristics: self-re-configuration, self-repair and self-adaptation.

Given the already immense research done on this subject, this document's author believes a review on what already exists is important for a better understanding of this manufactory paradigm.

In this context, (Semere, Barata, & Onori, 2007) elaborates on two main principles in which EPS are based. Then, it describes both the EPS concept and the EPS Control concept, followed by a reference architecture to EPS, based on the EUPASS project, already referenced in this document.

A study on the implications of EPS is necessary for a better understanding of this paradigm. (Frei, Barata, & Onori, 2007) elaborate on the context in which the EPS emerged, giving particular attention to the following perspectives:

- Manufacturing engineering, where a procedural evolution of the manufacturing paradigms is correlated;
- Control systems and multi-agent systems, where an emphasis is given to both decentralized and centralized solutions concerning system controllers and software and an advantages/disadvantages analysis is elaborated.

The authors have also given emphasis towards the evolution concept in EPS, not forgetting to mention intelligent modules and the way they socialize with each other.

In (Mauro, 2009), an EPS concept is given: "although there are similarities in the exploitation and implementation phases, the paradigms (EPS and HMS) differ (…) in perspective and (…) only EPS achieves fine granularity". The authors also establish EPS formalized ontologies and definitions, using practical developments to elaborate on those.

Another good example of the EPS' virtuosities is the IDEAS project (L. Ribeiro et al., 2011) where a description of the mechatronic multi-agent architecture of IDEAS is elaborated. These project had to ensure the following aspects: functionality representation, offered by the agents in the system as skills; yellow pages service interaction, which allowed for the clients to request operations to the agents, messaging, through the use of FIPA compliant protocols. The test case took place in FESTO's MINIPROD. The results obtained "stand as proof-of-concept of

very important characteristics not easily quantifiable and of high added value from an application point of view".

In (Neves & Barata, 2009), a more environment concern perspective in EPS is elaborated. Nowadays, many enterprises sustain ecological and economical concerns that must be attended by the industry in order to fulfil the sustainability concept. As such, the authors developed a way of classifying the EPS, attending two different variables, control and adaptability of the studied system: "With this approach, EPS strengthens the link between product and system design phases (…) which leads to several advantages". The advantages include: lower investment costs, shorter deployment times, among others.

(André Dionísio Rocha, Barata, Orio, Santos, & Barata, 2015) offers yet another example of a modular, distributed, multi-agent based architecture, for the PRIME project, that aims to create new solutions for deployment by SMEs of highly adaptive, reconfigurable self-aware plug and produce assembly systems

Given the wide spread of this sort of paradigms in the industry, when concerning production and control, it was only natural that, eventually, an approach of such paradigms on diagnosis would take place. This could, as in the topics aforementioned, improve response times and, ultimately, greatly diminish the failure diagnosis and error recovery costs the enterprises sustained.

2.1.6 Multi-Agent Systems

Following the development of AMS, the market demands required yet more agility, in the sense that new technologies are continuously emerging, and competitors are multiplying globally.

To be able to follow this trend, a new technology, if you may, emerges. One that could introduce intelligence to the system, so it could learn new methodologies, learn whenever a new part is introduced into the system, one that would be able to adapt easily to market trends – Multi-Agent Systems (MAS).

These technology relies on an entity that is yet to reach a consensus among the community towards its definition (Balaji & Srinivasan, 2010). It so happens due to the universality of the word "Agent" since it cannot be owned by a determined entity and is capable of independent action on behalf of its user or owner, figuring out, on its own, what needs to be done to satisfy the current objectives of the system. Consequently, MAS is considered to be more of a technology,

rather than a paradigm, that allows for the aforementioned paradigms to be implemented, giving them more autonomy, along with decentralisation.

In this context, (Shen & Norrie, 1999) enumerate several agent-based approaches for intelligent manufacturing which shows the scope extent of the Agent concept. Some of these are: representation of manufacturing resources such as works, cells, machines, tools, etc., (Shen, 2002), (Van Dyke Parunak, Baker, & Clark, 2001).

In (Bellifemine, Poggi, & Rimassa, 2001), an introduction to FIPA compliant framework is made. According to the same authors, The Foundation for Intelligent Physical Agents "is an international non-profit association of companies and organizations sharing the effort to produce specifications for generic agent technologies". FIPA is not a technology, neither does it promotes a set of technologies, rather emerged as a process of standardization, where an agent is the fundamental actor.

In (Parker, Manson, Janssen, Hoffmann, & Deadman, 2003), several types of models are proposed in order to facilitate manufacturing when it comes to simulation of Land-Use and Land-Cover change, and where each of this models' characteristics are enumerated.

In (Shen, Lang, & Wang, 2005), an architecture and implementation of an intelligent shop floor is proposed, which ensures both collaborative and adaptive capabilities, using for the effect, an efficient type of communication in the form of message services that uses XML as serializing language.

In (Olfati-Saber, Fax, & Murray, 2007), a consensus on networked multi-agent systems is proposed. It provides a mathematical analysis in order to establish the relationship between all the agents in any given network.

2.2 Diagnosis Paradigms

Given the current mind set of researchers, it has become appropriate to differentiate the diagnosis paradigms according to its current use in the industry. As one may have already inferred from what was aforementioned, this split occurs on the following terms:

- Those of the paradigms used nowadays in industry, which contemplate centralized, rule based, predictive and mass production focused diagnosis paradigms will henceforth be acknowledged as Usual Diagnosis Paradigms.

- On the other hand, those which contemplate decentralized, evolutional and learning diagnosis systems and which are still being developed and researched, will be referred to as Emergent Diagnosis Paradigms.

2.2.1 Usual Diagnosis Paradigms

In (J. Barata, Ribeiro, & Onori, 2007) a brief review of diagnosis methods, systems and techniques is elaborated, being mentioned: case based reasoning, fault tracking, among others. As one may have inferred by now, the same concepts (two architectures: individual and community; etc.) applied to production and control should be used in diagnosis.

A chronologically ordered presentation (Luis Ribeiro & Barata, 2011) of the early stages of diagnosis' paradigms will be given, with a brief explanation concerning each of them.

The paradigm that can be traced back to the very beginnings of machine instrumentation and, therefore, being considered as the most primitive form of monitoring is that of Limit Checking (LC).

This primitive form of monitoring is mathematically explained and developed in (Isermann, 2006), exemplified with several uses of such in the overall industry.

LC, despite being one of the easiest of implementing, carries out serious drawbacks such as false alarms in the event of noise and the change of the operating point (Chen & Patton, 2012).

The next one on the list is Quantitative Methods (QM) which were the first to exploit the potential of the 70s microcontrollers (Luis Ribeiro & Barata, 2011). The majority of these methods are recognized as Fault Detection and Identification (FDI) and mark the beginning of Condition Based Maintenance (CBM) and Predictive Maintenance (PM).

As to what CBM relates, (Ellis, 2008) offers a review on the requirements for this paradigm such as management support, data analysis to "determine cost-effective monitoring points" and sustains that CBM provides diagnosis from a cost-effective point of view.

(Grall, Bérenguer, & Dieulle, 2002) provide an analytical modelling of a CBM for a stochastically and continuously deteriorating single-unit system considering both replacement threshold and the inspection schedule as decision variables.

Still concerning CBM (Jardine, Lin, & Banjevic, 2006) elaborated a review defining data acquisition and analysis practices in order for a proper diagnosis to take place, along with methodologies used in maintenance decision-making.

Concerning PM, (Mobley, 2002) provides an introduction to this paradigm sustaining "all preventive maintenance management programs are time-driven, elaborating on some concepts such as mean-time-to-failure which represents how long a product can reasonably be expected to perform.

(Grall, Dieulle, Berenguer, & Roussignol, 2002) propose a decision model that enables optimal inspection and replacement decision, in a PM point of view, considering two different maintenance decision variables.

(Zhou, Xi, & Lee, 2007) propose a junction of the two aforementioned paradigms, PM and CBM, resulting in a condition based predictive maintenance, for continuously monitoring a system which is subject to degradation.

This quantitative methods were not the solution for all the problems in industry making it necessary for a new type of methods to appear, Qualitative Methods (QLM). These can be classified according to the nature of the inference performed (Luis Ribeiro & Barata, 2011).

The qualitative reasoning began with simple fault trees, with component to function relationship but quickly evolved to stable diagnostic engines being the first one the General Diagnostic Engine that enabled symbolic abductive reasoning over composite devices.

The Livingstone (Williams & Nayak, 1996) engine is one of the kernels that attracted more attention due to its applications in space-travelling. Livingstone uses component-based declarative models and was designed to achieve a compromising relation between the conventional first-order logic approaches (used in other engines) and the reactive concurrent approaches.

In a more recent mind-set, the ACORDA (Lopes & Pereira, 2006) engine which supports prospective logic programming, that is based in abductive reasoning, was used in the diagnosis of intelligent shop floor components.

As to what History Based Methods concerns (Luis Ribeiro & Barata, 2011), these have been widely applied in industry, from its very beginnings, given that its popularity comes from the fact that industrial installations, as they were before, are not subject to major changes. Under this topic, Artificial Neural Networks (ANN) is the paradigm being approached as it is the one with more common characteristics to those of this document's topic.

That being said, ANN started to be recurrently used in industry as of the 90s. (Sorsa & Koivo, 1993) elaborate on the applications of this paradigm in pattern recognition and fault

diagnosis problems using a "realistic heat exchanger-continuous stirred tank reactor system" as a study case.

In (Aminian, Aminian, & Collins, 2002), ANN are used for successfully fault diagnosing actual circuits by pre-processing the collected data by wavelet decomposition, normalization and principal component analysis to generate optimal features for the training of the neural network.

Lastly, and approaching the late 90s, some Hybrid Methods came to light as the web spread worldwide and telecommunications became an integrant part of every day to day life (Luis Ribeiro & Barata, 2011). This paved the way for a completely different mind-set on what diagnosis concerns, allowing for a multidisciplinary research to take place.

2.2.2 Emergent Diagnosis Paradigms

Given the current evolution direction of the industrial systems, it made no sense that diagnosis kept being a centralized, non-communicative, unintelligent system on top of a, sort of, conscious, decentralized one. To enable this characteristics in diagnosis, several theories were elaborated but only some actually came to life.

(J. Barata, Ribeiro, & Colombo, 2007) proposed a Service Oriented Architecture (SOA) for diagnosis in which an extreme importance is attributed to intelligent devices, thus offering a distributed intelligence. This concept opens new doors as to what diagnosis concerns, since, having distributed devices, all capable of deploying diagnosis features, collaborative diagnosis is, henceforth, a possibility to be explored.

In (Feldmann & Göhringer, 2001), propose an Internet based diagnosis system. The authors argue that this approach could bring multiple benefits to all entities involved in the diagnosis mechanism, further comparing theirs to other existing ones.

In (Wu, Chen, Li, & Li, 2005), yet another remote, web-based monitoring and fault diagnosis system is proposed. The authors set to develop an architecture that answers the following five challenges: hugeness, distribution, high speed, automation and complexity. The proposed architecture uses VSN-NetMDS system that allows for both factory-based experts and office-based experts to communicate among each other. For a better understanding of the whole system, the authors chose the UML language to model the system.

(Fries, 2007) develop a soft computing approach to Multi-Agent fault diagnosis where it proposes a hybrid diagnostic approach, which merge symptom recognition and functional reasoning. For the multi-agent part of the diagnosis, an algorithm is elaborated and explained.

In (Mendes, Santos, & Costa, 2009), another multi-agent platform for fault tolerant control systems is proposed. The authors propose a multi-agent based architecture with organizational relationship. The work was tested in both simulated and real environments. (Desforges & Archimède, 2006) propose yet another multi-agent network for diagnosis systems in both sensors and actuators. The authors start by extending the sensor/actuator concept into an agent so it could be inserted into the network. The authors also propose a framework, which makes it easier for others to use their work. The agents, as in any other system, possess several different right and duties.

In (J. Barata, Ribeiro, & Onori, 2007), the authors propose an architecture for diagnosis in EPS, giving particular emphasis towards self-capabilities, stating there are still challenges and open issues that should be addressed. Some architectural principles are also elaborated taking into account the premises in which the current system is limited to.

In (Luis Ribeiro & Barata, 2012), a validation of a co-evolving diagnosis algorithm for EPS is given. To do so, an architecture which implements distributed diagnosis was also proposed in this paper. The architecture encompassed two main phases: the initial configuration phase, where "the system designer has to establish the initial interactions between the existing modules" and the runtime phase, in which "the system may undergo structural changes" by a variety of reasons.

(McArthur et al., 2007a) and (McArthur et al., 2007b) offer several concepts, approaches and technical challenges on multi-agent systems for power engineering applications. The concepts include: agency, intelligent agent, among others. The authors sustain there are several benefits towards the use the MAS technology in power engineering technologies and elaborates on several approaches that back up this claim.

Along with all these aforementioned approaches to diagnosis, some community members sustain that researchers should thrive for a better answer to this problem in bio-inspired paradigms, as it has already been done for self-organizing systems, with the ant's path-finder algorithms.

In this context, the main topic of this document emerges: Artificial Immune System (AIS), which is a bio-inspired algorithms based diagnostic paradigm. It was developed as an image of the actual human immune system. As such, it includes concepts such as B Cells and self and non-self-cells which will be approached later on. It is a learning and adaptive mechanism that grows with the system. This is a relatively new paradigm and there is still plenty of work to be done on this subject.

(Jon Timmis, Andrews, & Hart, 2010) sustain that AIS has direct parallels with Swarm Intelligence (SI), arguing that this two paradigms are complementary rather than competitors and should be used together to solve complex engineering problems.

Given the different diagnostic paradigms, an urge for a sort of consortium towards how to classify AIS against all the other paradigms emerged. In this context, (Garrett, 2005) suggest five different questions that, being answered, may be "of value as an introduction and critique of AIS, and its relationship to other paradigms". The questions are based on the usefulness of the AIS and how and why it was developed.

(J. Timmis, Andrews, Owens, & Clark, 2008) gives an interdisciplinary perspective on AIS arguing it has multiple applications on a variety of research fields. However, if one is to use AIS for any branch of development a framework must be first developed for a better integration to take place, which is what this paper elaborates on.

In the same context, (Smith, Timmis, Stepney, & Neal, 2005) elaborates on a conceptual framework for AIS and its algorithms and offers a perspective on how it should be adapted for the engineering AIS.

There are currently several implementations of this type of systems, in the most varied areas of expertise. Examples of these implementations are:

- Prognostic Methodology for Health Management of Electrical Equipment of Propulsion System in a Type of Vessel Based on Artificial Immune Algorithm, which utilize AIS algorithms to determine whether is there any problem with any of the electrical equipment of a certain installation, in this case, of a vessel (Hu & Qin, 2012);
- Mobile Agent Based Artificial Immune System for Machine Condition Monitoring (Hua, Gondal, & Yaqub, 2013), which takes advantage of the interoperability of this type of systems to monitor an agent based machine;
- A Fast Anomaly Detection System Using Probabilistic Artificial Immune Algorithm Capable Of Learning New Attacks (Mohammadi, Akbari, Raahemi, Nassersharif, & Asgharian, 2014), that can detect anomalies on any giving system using an algorithm capable of learning new attacks.

2.3 General Conclusions

As it is perceivable for those who take a closer look at what is said above, both manufacturing and diagnostic paradigms have evolved from unintelligent, centralized, rule-based systems to distributed, intelligent, autonomous ones.

This change was far from being fast, or even obvious. Only when the hardware evolved was it possible to make more intelligent systems. It so happens because, before the introduction of real software on the shop floor, it was not possible to introduce programmable logic on hardware more complex than that already offered by Programmable Logic Controllers (PLC).

Despite the current trends of the diagnosis paradigms point towards a more distributed, intelligent systems, it is still afar from what enterprises look for in a diagnostic system. The current systems are still operating from a predictive perspective, which leads to the system being incapable of detecting unforeseen threats.

AIS emerges as a non-predictive, rather adaptive paradigm, which uses bio-inspired algorithms to act like the human immune system therefore, literally, learning with its mistakes, making it, theoretically, able to detect every single error in any system.

3

Architecture

As sustained in the previous chapter of this document, the diagnosis paradigms currently being used do not offer real problem solving capabilities. To say the least, they are incapable of diagnosing a huge range of possible errors the system might sustain, as to what the current production systems demand. This happens because of the high level of customization that makes the diagnosis harder to perform. As such, this Artificial Immune System (AIS) based architecture presents itself as an alternative to the aforementioned paradigms in an attempt for a more diversified and capable error recovery and diagnostic system.

That being said, the proposed architecture is based in a Multi Agent System (MAS) that allows for each entity in the system to act accordingly to its needs without having a constant information about the overall state of the world that surrounds it. It uses bio-inspired algorithms in order to provide new possible fixes for every new error and has the capability of learning with its errors, which leads to a better overall performance of the system.

One of the system's most important entities is the B Cell Agent (BCA), which is launched by the bio-inspired algorithms. It is responsible for analysing and determining if it has the necessary components to solve for the fix to the current error in the system, thus acting like a "real" Human Immune System's B cell.

The proposed approach covers an environment composed of four distinct entities, where two of them are constantly diagnosing and analysing the production system in which they are inserted and the remaining two act as error recovery entities which are in charge of searching and resolving the errors the underlying production system may output.

3.1 Overview

The proposed architecture mainly consists of four different types of generic agents from which two are more concerned with diagnosing and the remaining two with the error recovery subject. Along with this concern, one of the four entities is responsible for the learning part, one might say, of the system, i.e., stores the fixes found by the system as a learning mechanism so the system no longer needs to search for a fix, if the same error is to occur once again.

Despite this has been already done in other works, the main difference that may lead for the use of this type of systems rather than the older ones is that this one is decentralized. This allows for a better performance of the overall system, since it may work on several machines instead of only one.

For a more insight perspective, the aforementioned architecture can be divided in two layers: Shop Floor Layer and Cloud Layer; as it is observable in Figure 3.1. The first encompasses both the diagnosis and recovery mechanisms of the system, and is where the low-level system is supervised by constantly searching for anomalies in the outputs – the Shop Floor Layer.

The second one refers to the cloud module of the system, responsible for acknowledging new error recoveries, thus providing the system with more responses in fault cases – the Cloud Layer.

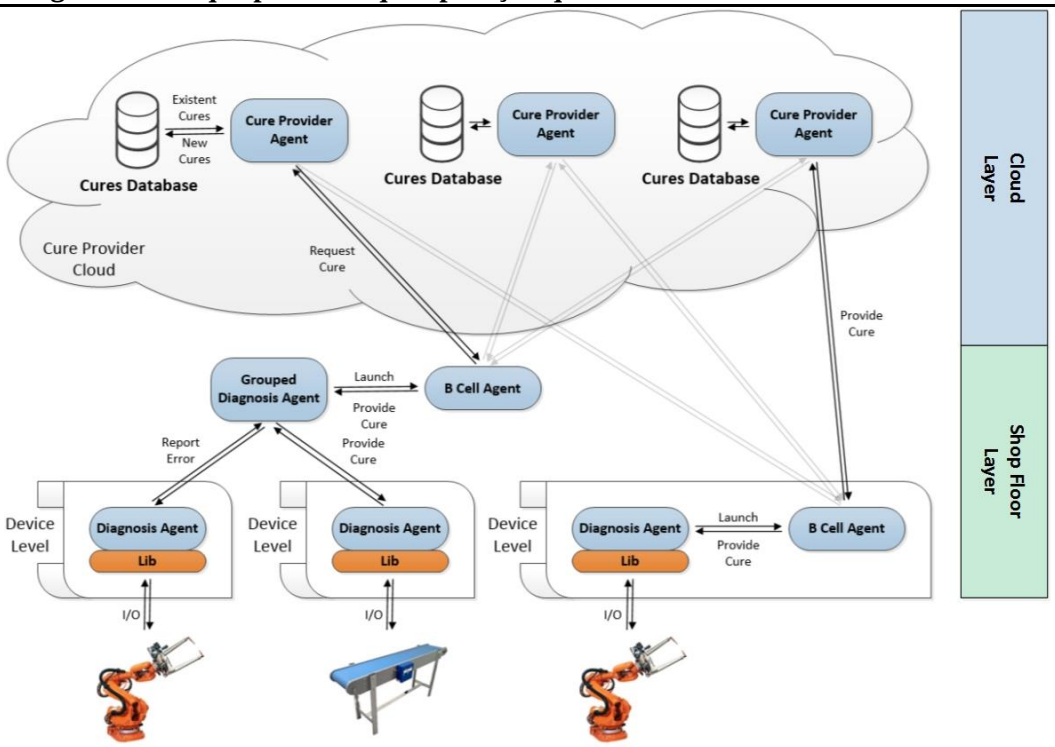


Figure 3.1 – Evolutionary modelling of the System's Architecture.

3.2 Generic Agents

As it was said before in this document, there is yet no consensus towards a definition of Agent. However, the community agrees on the following characteristics: autonomous, evolvable entity, capable of executing tasks. (Monostori, Váncza, & Kumara, 2006).

In the presented architecture, and as it was already said previously in this document, four different types of generic agents were used: Diagnosis Agent (DA), Grouped Diagnosis Agent (GDA), BCA and Cure Provider Agent (CPA). Each one of this agents has its own role in the system, which will be further explained.

Given the different tasks each of the agents will perform in the system, they were separated accordingly to different designations: the Shop Floor Agents and the Cloud Agents.

The Shop Floor Agents contain those directly involved in diagnosing and recovering a determined resource. Hence, this group contains the DA, the GDA and the BCA entities, since the first two abstract two different levels of the shop floor entities, thus being capable of diagnosing them, and the latter works as an error recovery operator.

3.2.1 Shop Floor Layer Agents

Composed of the DA, the GDA, in charge of diagnosing the system, and the BCA, launched by DAs and GDAs in order to search for a cure. The DA abstracts physical resources of the manufactory system and supervise them, by constantly verifying if its current output is under predetermined parameters.

The BCA is the one responsible for finding new cures for the system and is only launched if an error that has not occurred already is to occur.

3.2.1.1 Diagnosis Agent

The DA is the lowest level entity of the entire system. It is responsible for diagnosing a physical resource, such as a robot, conveyor belt, or any other component in a manufactory environment.

During execution time, the DA constantly verifies the methods of a specific resource, the one it is, sort of speak, in charge of, trying to find errors and/or failures. To do so, it constantly reads the outputs of the resource it is supervising and, shall the output value step out of predetermined values, it is considered a malfunction.

Once a malfunction is detected, the DA verifies if there are any available cures for the detected error that are of its knowledge; if so, the DA performs the cure immediately; if not, the DA performs one of the two following possibilities:

- In the event the DA is already grouped under a GDA, then the DA propagates the error top wards to its group for a broader search of the cure to take place.
- If, otherwise, the DA is not under a Grouped Diagnosis Agent, it runs the AIS algorithm to define what kind of B Cell Agent should be launched.

A simple, yet illustrative, description of this behaviour can be observed in Figure 3.2.

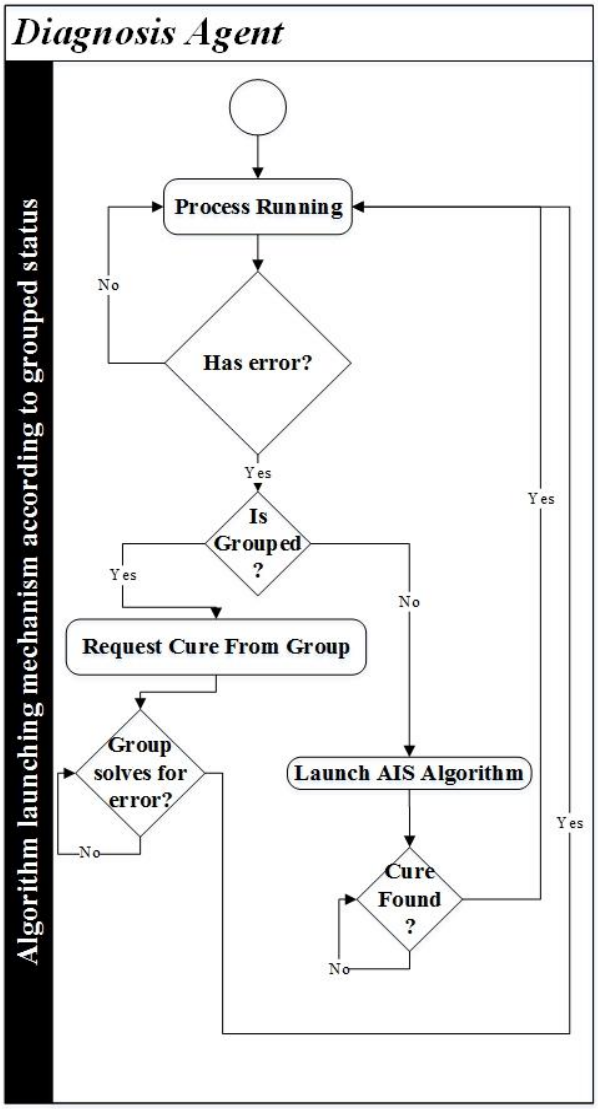


Figure 3.2 – Diagnosis Agent Behaviour.

The DA is constantly verifying the current process of the resource it is monitoring for errors. If one is to occur, the DA verifies whether or not it has a higher level entity. If so, it requests the cure from that high level entity. If not, it launches an instance of the AIS algorithm that will, hopefully, find the cure needed. If it so happens, the DA goes back into monitoring the resource it is in charge of.

3.2.1.2 Grouped Diagnosis Agent

The GDA is constituted by a group of physical devices that work together, consequently influencing the execution of each other by processing all the data from the different devices associated to it.

These entities are capable of collecting the malfunctions of the resources and understanding if the errors are correlated or not. By correlated, one should suppose the GDA is capable of looking for the cure for a given error in a specified DA in the other DAs' cures.

It is this entity's responsibility to search for a resolution for its subjects' errors, since it is a higher level entity. Therefore, when the GDA agent does not have a cure for a specific problem, itself proceeds to launch B Cell Agents according to the AIS algorithm defined.

A simple, deductive, illustration of this agent's behaviour is presented in Figure 3.3

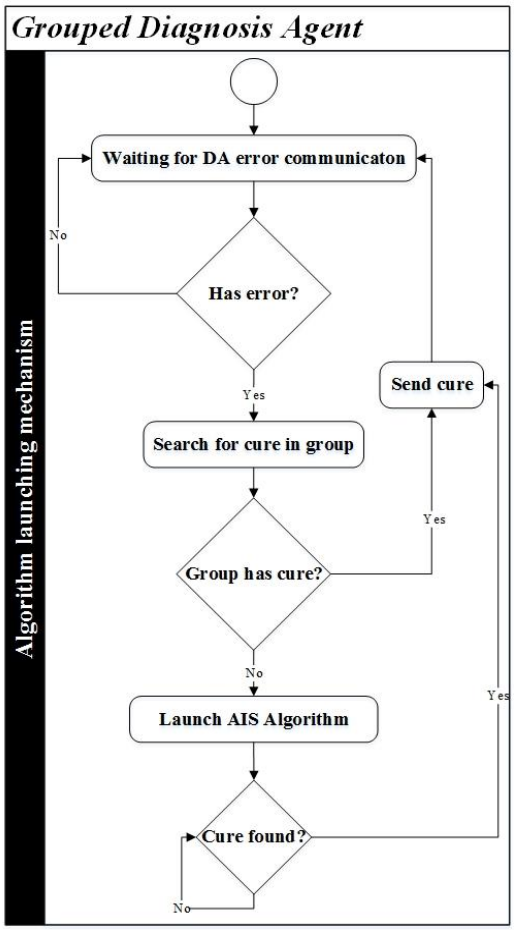


Figure 3.3 – Grouped Diagnosis Agent Behaviour.

The GDA is constantly waiting for any incoming message of the DAs registered in it. If a message arrives, the GDA verifies whether or not it has a cure for the requested error in the group. If so, it sends the cure to the requesting DA. If not, it launches an instance of the AIS algorithm that will, hopefully, find the cure needed. If it so happens, the GDA informs the DA of the new cure and goes back into monitoring all the DAs it is in charge of.

3.2.1.3 B Cell Agent

The BCA emulates a real B cell of the Human Immune System, thus being called upon service when an antigen (error that cannot be fixed by the system's known cures) is outputted by the low-level system.

Hence, when a malfunction is detected and neither the GDA nor the DA have cures for the error, in order to find new cures, these entities launch an AIS algorithm. This algorithm will create several BCAs that will select, clone and mutate themselves in order to find a cure for the current error.

A description of this agent's behaviour is depicted in Figure 3.4

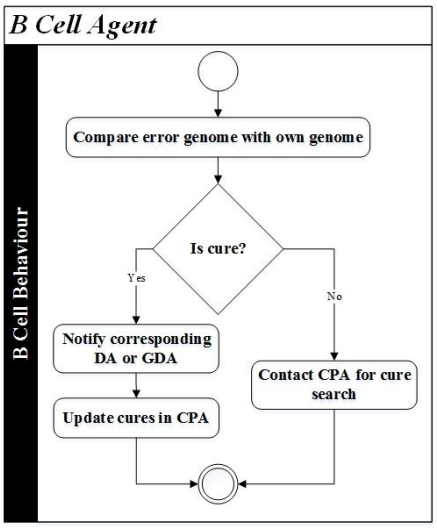


Figure 3.4 – B Cell Agent Behaviour.

The BCA is launched by the AIS algorithm with the error's genome and its own genome as parameters. It then compares both of them and determines whether or not its own genome is the cure for the error. If so, the corresponding DA and GDA, that launched the algorithm, will be notified and the cures in the cloud will be updated. If not, the BCA contacts the CPA in order to try and obtain a cure from it.

3.2.2 Cloud Layer Agents

In this topic, only one type of agent stands. The agent that will be responsible for holding all the cures in the system, and to inform and collect data from several systems that may connect to it, in order to provide more and better information on cures already present in their systems, so it can supply others with those same cures, should it be needed.

3.2.2.1 Cure Provider Agent

The CPAs constitute a cloud that covers the totality of the system, meaning this cloud is available to provide cures to all the lower level entities, if one ever contacts it requesting a cure.

Each of these entities has an associated database where all the known possible cures that are available for consult if any request for cure ever arrives to this entity are stored. This database is updated whenever a new cure is found in the system.

Hence, when a lower level agent asks for a possible cure to solve a given problem, the CPA queries the database for possible cures to the problem in hands. Shall a cure be found and the CPA returns the cure to the BCA that requested it. If no cure was found, it simply answers with a denial.

A short, illustrative description follows in Figure 3.5.

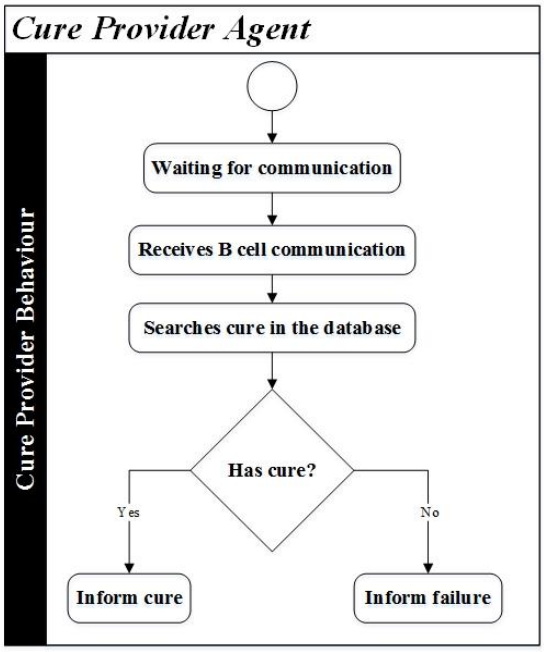


Figure 3.5 – Cure Provider Agent Behaviour.

This agent is also reached by the BCA when it discovers a new cure for further introduction into the database and availability for other BCAs, in case of need.

This behaviour is illustrated in Figure 3.6, in which one can infer about how the BCA and CPA interaction takes place.

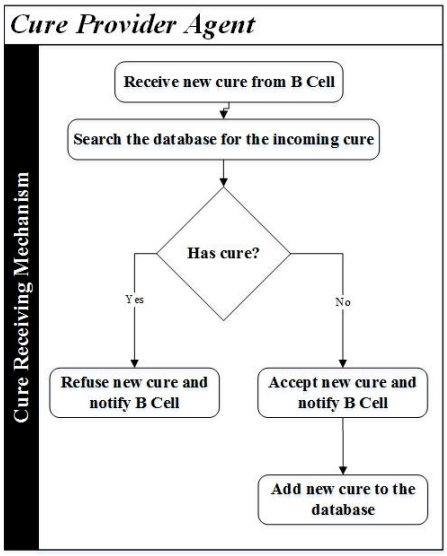


Figure 3.6 – Cure Provider Agent Cure Receiving Mechanism.

If a CPA is contacted by a BCA informing a new cure, the CPA will search its database for the cure that is coming in. If the CPA already knows that cure, than it is refused and the BCA is notified. If not, the cure is accepted, the BCA notified and the new cure is added to the database.

3.3 AIS Algorithms

Despite its obvious importance to the overall performance of the system, the main components of the architecture are not its entities, but the processes they launch to perform its tasks.

That being said and since this refers to a learning, decentralised diagnosis system's architecture, not only are to be referenced the agents by which it is supported but also the algorithms that allow it to develop new cures from the existing ones, being this the central component of this system, without which there were no learning mechanism.

The provided architecture is suited for all the algorithms framed in the AIS. However, and as it will be seen further down the road, three of the algorithms are more fit for the accounted architecture and were the ones studied and researched for the purpose of this document.

3.3.1 Fundamentals

There are several algorithms used for learning capabilities. Many of these are based on living being behaviours or, for that matter, on the cells behaviour in each of the living systems they populate.

The AIS Algorithms are bio-inspired algorithms that are based on the behaviour of the Human Immune System B Cells, hence the name, Artificial Immune Systems. These algorithms are supposed to reproduce the behaviour of the aforementioned cells in a decentralised system, as it is the Human's, in order to perform maintenance and error recovery on the named system, much alike the actual B cells do in our Human System (J. Timmis et al., 2008).

The real B Cells are launched by our System in a given area of our body, when that same location emits a distress signal that notifies the main system something went wrong, and that the task being currently held there is not under the allowed parameters. That "something that went wrong" is called an Antigen.

The B cells are then launched to the designated location and start to analyse the error. This analysis is done by inferring the B cell genome's affinity towards the antigen's genome. Those with higher affinity values are selected, cloned, mutated, etc. with the ultimate goal of finding a cure for the antigen. The first operation, selection, selects the higher affinity genome to fight the antigen; the second operation clones the higher affinity B cells; the latter mutates the B cells' genome.

Even though this is a very simple, didactic even, explanation for what really occurs at a cellular level in our system when a disease is detected, it serves as an introduction to how the B cells work and upon which the algorithms will be built.

That being said, one may conclude that the AIS algorithms are based in this three basic principles: selecting, cloning and mutating. Therefore, there will be three different algorithms: Negative Selection, Clonal Selection and Network Model.

For a better insight of the algorithms to take place, a description of what the algorithm will be monitoring, therefore scanning for differences, is in order. Instead of using the standard alphabets, such as the Latin, Cyrillic, etc., which would imply more complex searches, the author opted by using the binary alphabet, composed only by "0" and "1", thus increasing the performance of said algorithm exponentially. Hence, this algorithm will consider as an error every bit change in any given byte sized word. Taking this explanation into consideration, the author believe the reader is now ready for the explanation of the algorithms themselves.

3.3.2 Negative Selection

Its purpose is that of allowing some degree of tolerance for the "self"-cells (those normally present in the organism) by dealing with the Immune System's ability of detecting unknown antigens (harmful cells) without prejudicing its own cells. The Immune System's

generated B Cells, that fight these antigens, are formed by a pseudorandom genetic rearrange. Those of which that react against the antigens are used to destroy it and replicated in the organism as matured cells (Kim & Bentley, 2001). This algorithm works in two different stages: detector set generation and monitoring the protected data.

3.3.2.1 Detector Set Generation

Each detector is a string that does not match any of the protected data. This phase is illustrated in Figure 3.7.

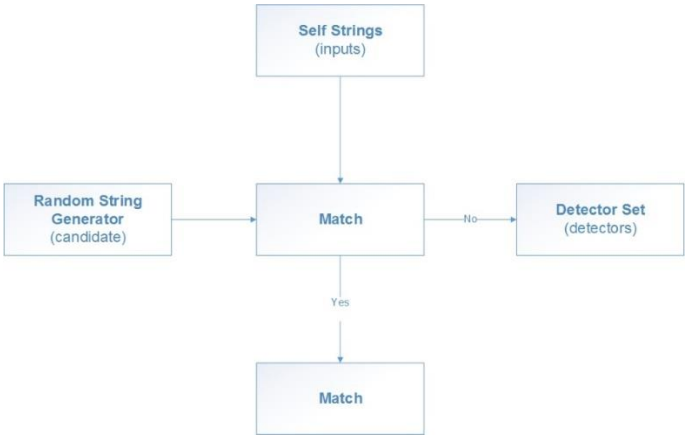


Figure 3.7 – Negative Selection Algorithm Overview.

The first step the algorithm takes is to calculate the probability that two random strings have of matching (Ayara, Timmis, de Lemos, de Castro, & Duncan, 2002). This value will be used to calculate the number of maximum strings that the detector set may contain (one must be attentive so that this number does not exceed the maximum number of possibilities enabled by the size of the word considered).

After getting the maximum size of the detectors array, the algorithm initiates the pseudorandom generation of strings, with size equal to the byte word given as a parameter. This step may be named as the training of the detector set. This concludes the training, and most complicated, phase of the algorithm.

3.3.2.2 Monitoring the protected data

Once a detector is triggered, an unscheduled change occurred. This stage is represented in Figure 3.8.

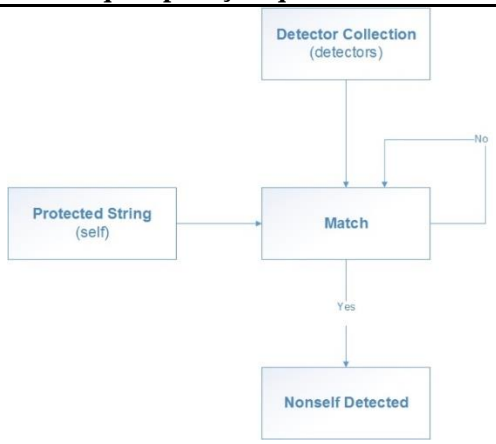


Figure 3.8 – Matching Process.

Lastly, one must launch some type of thread that continuously checks the inputs for changes, by comparing them with the strings in the detector set (Garrett, 2005).

Despite not looking like a promising approach, since the algorithm goes through all the possible strings, except for the ones in the original set, in a finite alphabet, it is a surprisingly feasible one, mathematically speaking, since that, by using a rather small set of detector strings there is a very high probability of noticing a random change to the original data. Plus, the number of detectors can remain constant whilst the size of the protected data grows. If the alphabet to be monitored was immense, one could, for efficiency sake, use several instances of the algorithm instead of a single one, in order for a more efficient detection of the algorithm, since each instance of the algorithm would have a rather different set of detectors (Forrest, Perelson, Allen, & Cherukuri, 1994).

The algorithm receives as inputs:

a) ***The probability of not detecting a change:*** since the algorithm is logarithmic, this value cannot be 0 – if one desires (almost) none probability of failure, simply instantiate it to a nanoscale value.
b) ***The Self-String:*** String that contains the inputs that will train the detectors. This string is to be divided into segments with the size of the byte word one may want to use. This segments will make for the inputs of the algorithm.
c) ***The word size:*** Number of bytes each input will have.

3.3.3 Clonal Selection

The main idea behind this algorithm is that only the B Cells that recognize the antigen will thrive and replicate. This principle describes the basic characteristic of an immunologic

response to an antigen caused stimulus. This algorithm's main characteristics may be enumerated as follows:

- New cells are copies of those they derive from, and are then subjected to a high rate mutation mechanism (somatic hyper mutation).
- Procedural elimination of the new cells that, after mutation, endanger the survival of the non-prejudicial cells for the organism (self).
- Further cloning and mutation of the cloned cells that respond positively to the antigen. This mechanism allows for a faster response to the antigen (White & Garrett, 2003).

According to this theory (and, as sustained by most experts, the one which represents the biological system more accurately), every molecule that can be recognized by the Immune System is known as an Ag. Whenever an animal is exposed to an Ag, some of its B cells respond by producing molecules whose priority is to recognize and bind to the Ags. Since each B cell secretes only one type of molecules, the more different B cells in the system, the easier it will be for it to detect the anomaly, in the form of an Ag. In order for this different B cells to emerge, a mutation of some kind must take place. Once a B cell identifies its molecule as the one which is capable of neutralizing the menace, it will proliferate exponentially and differentiate, for a better expansion throughout the system. This process can be observed in Figure 3.9.

Hence, the main features of the clonal selection theory (de Castro & Von Zuben, 2002) are:

1. Proliferation and differentiation on stimulation of B cells.
2. Generation of new random genetic changes, by a form of accelerated somatic mutation.
3. Estimation of newly differentiated lymphocytes carrying low affinity antigenic receptors.

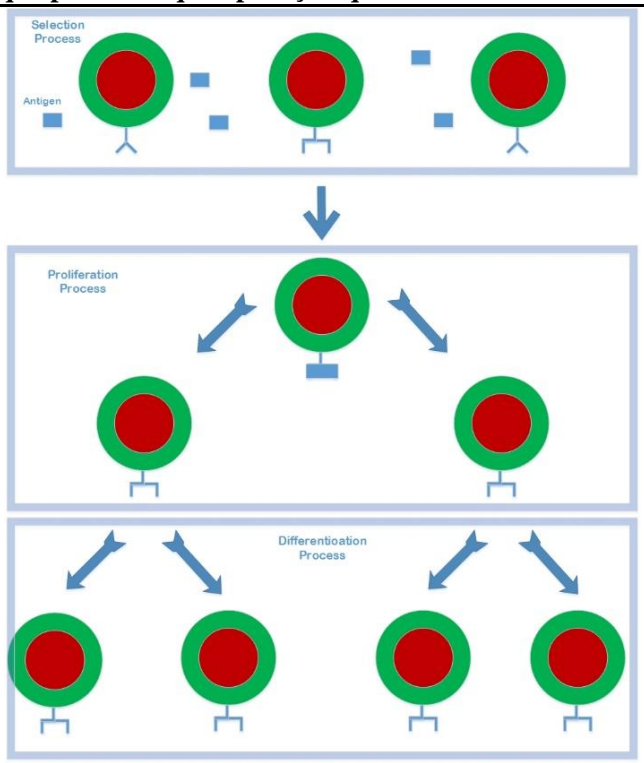


Figure 3.9 – Clonal Selection Principle.

This algorithm starts by determining the affinities between the antibodies given as inputs and the antigen found in the system. This is achieved by comparing each bit of the byte word, and incrementing the affinity whenever the bit value differs.

This affinity array will then be ordered and the high affinity array will be defined. For each high affinity antibody, affinity number of clones will be generated.

This clones will then be mutated to form new genomes for each one of them. A new evaluation on its affinities towards the antigen will take place.

If one of the cloned antibodies has a 100% affinity with the antigen, i.e., for every ‘1’ in the antibody genome there is a ‘0’ in the same position of the antigen genome and vice versa, then it is safe to assume that a cure was found and the system is notified. If there was not a 100% affinity in any of the clones, the process will go back to define the high affinity of the new clones and so forth until a cure is found.

3.3.4 Immune Network Theory

This theory sustains that the Immune System maintains a regulated network of interconnected B Cells which purpose is to ease the antigen detection. This cells stimulate and

supress each other with the ultimate purpose of providing stability. The connection between two B Cells is proportional to their affinity to each other (Farmer, Packard, & Perelson, 1986).

This network is formed by B cells that have the ability to recognise others in the system. The network self-organizes and stabilizes for a greater survival chance, since this is achieved by mutual reinforcement between every B cell in the network via a feedback mechanism. The more neighbours, the more stimulation a B cell will receive from the network. The maintenance and survivability of a given B cell depends of its affinities to both the antigen and to its neighbours. The new B cells only have new genomes that, in the case they provide a better match for the antigen, will proliferate and survive longer. The more mutations and selections in the network, it will learn to produce better matches for the antigen currently present in the system.

To reproduce this behaviour, the algorithm starts by creating the initial population (a group of B cells) whose genome will be abstracted from the Self string that is provided to it as an input. Immediately after its launch, the B cells will evaluate whether they do or do not possess the cure for the current antigen. In the event they do, the system is notified and the antigen nullified.

However, if the cure is not present, the B cell begins to calculate its affinities towards both their neighbours and the antigen currently present in the system.

After all the affinities are assembled, the B cell starts to mutate its genome affinity times for each neighbour and antigen. The mutated genomes will then be used to create new B cells that will provide further expansion to the network. This process will keep generating B cells until one of them provides a cure for the antigen in the system (Jon Timmis, Neal, & Hunt, 2000).

3.3.5 How to make the algorithms efficient

As one may have assumed by now, most, if not all, the processes in this algorithms involve some type of comparison between objects. For a better and more efficient comparison to be made by the algorithms, in order to provide the fastest output possible, given that time is one of the greatest concerns in manufactory, this document's author sustain that a string, for each of the genomes, the cure's and the error's, is the most simple method for an efficient comparison.

Taking into account that all of the algorithms previously mentioned use some form of pseudorandom mutation or generation, if it were to be considered that operations could be defined by some kind of codename constituted by any of the existing alphabets' (Latin, Cyrillic, etc.) letters the complexity of mutating or even generating random strings of such codenames would rise exponentially.

Therefore, to diminish the complexity of said pseudorandom operations, it is the author's belief that binary strings of data, representing the cures and diseases of the system, would satisfy the needs of any system, even those of the more complex ones, since the algorithms are prepared to accept any string length, albeit the bigger, the less efficient the algorithm would become.

With this binary method, the author hope to make the algorithms more efficient, thus reducing the time the production line is affected by the "disease", since it will be easier for the cure to be detected. If this explanation revealed itself somehow difficult to understand, a graphical explanation follows in Figure 3.10.

Figure 3.10 – Conversion of a simple move command to a binary representation.

In this figure, it is easy to understand that basic commands of a robot part or the whole robot itself are translated into a binary representation. This commands, operations if you may, are stored within its DA. If, somehow, the binary representation, the "genome", of the current operation is not equal to any of those previously defined than it is considered an error, a "disease" that must be dealt with.

Now that we have seen the fundamentals behind the AIS algorithms we can proceed to their actual implementation.

4

The Algorithms

As it was already said in this document, three Artificial Immune System (AIS) algorithms were studied, implemented and tested. In this section, the implementation of each of the three algorithms will be presented and discussed, along with the explanation and reasoning used for the choice of one algorithm over the others. The algorithms were implemented using the Java programming language under a behaviour oriented platform, the JADE (Java Agent Development Environment) framework (Bellifemine, Poggi, & Rimassa, 1999).

4.1 Negative Selection

This algorithm receives four different inputs:

- The training set, which is composed of the genomes of the known cures, and will be used to train the new genomes;
- The error genome; the probability of failure, which, according to the type of system, should be as low as possible, even though it cannot be absolute zero since the algorithm uses logarithmic functions;
- The word size, in bits.

This algorithm's agent mechanism, with its associated behaviours, can be observed in Figure 4.1. The agent starts by launching an **NSInstance**, implemented using the One Shot Behaviour (OSB) from the JADE framework that will set up the initial genome inputs, calculate the probability of failure, with the method **calculatePm**, and calculate the amount of detector strings, using the method **setInputs**.

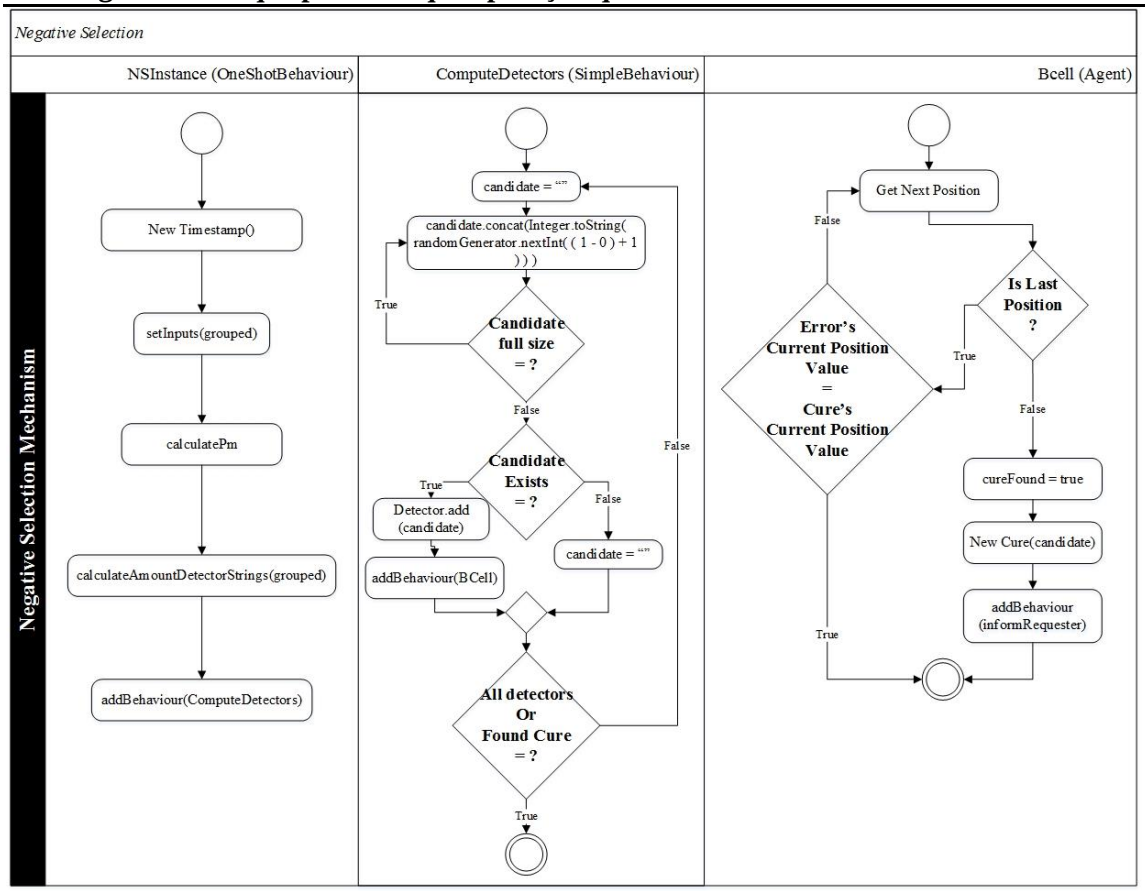


Figure 4.1 – Negative Selection Mechanism.

The algorithm then proceeds to compute the new genomes, by the means of the **ComputeDetectors**, implemented using the Simple Behaviour (SB) from the JADE framework that receives as input the genomes that will train the new genomes, the error's genome and the maximum size of the new genomes array. It will then start processing new candidates and, after evaluation, these candidates will become the new B Cell Agents' (BCA) genome. This procedures conclude the previously mentioned Detection Set Generation phase of the algorithm.

Once the above behaviour calculates a new genome, it immediately launches a BCA, which main responsibility is to infer if its genome is a valid cure for the current error. This BCA will then inform the launching Diagnosis Agent (DA) or Grouped Diagnosis Agent (GDA) of either success or failure by the means of an AchieveREInitiator Behaviour (AREIB) from the JADE framework.

4.2 Clonal Selection

This algorithm receives three different inputs: the training set, which is composed of the genomes of the known cures, and will be used to train the new genomes; the error genome; and, finally, the word size, in bits. The behaviours used to reproduce this algorithm are represented in

Figure 4.2. This algorithm's agent starts by launching the **CSInstance**, implemented using the OSB from the JADE framework that will determine the antibodies repertoire, an array of known cures that will be used for training the new genomes. It then proceeds to launch the **DetermineAffinities** Simple behaviour.

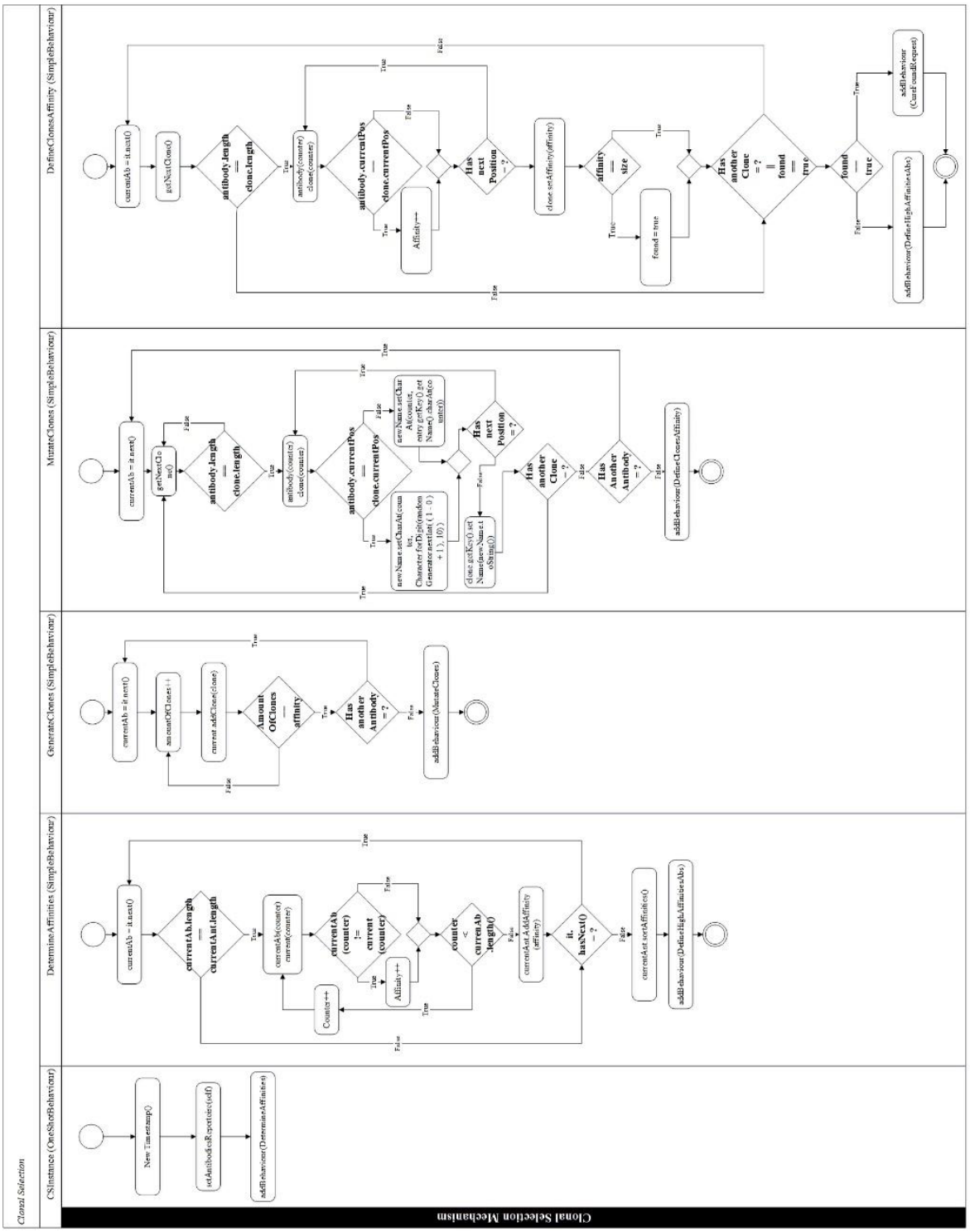


Figure 4.2 – Clonal Selection Mechanism.

The **DetermineAffinities**, implemented using the SB from the JADE framework, will calculate the affinity of each antibody's genome towards the error's genome. The affinity is increased whenever, for the same bit position, the value is not equal. After calculated, the affinity is then added to a list inside the error's class. After all the antibodies are processed, the **DetermineHighAffinitiesAbs**, implemented using the SB from the JADE framework, simply sets the HighAffinity array of the error as the upper half of its Affinity array and, therefore, it will not be graphically represented in this document.

After setting all the high affinity antibodies, this behaviour launches the **GenerateClones**, also implemented using the SB from the JADE framework. This behaviour creates a clone for each affinity times as there is in the correspondent genome's antibody. It is now time to mutate all the clones in order to find a cure for the current antigen.

To do so, the above behaviour launches the **MutateClones**, implemented using the SB from the JADE framework, which will mutate the previously obtained cloned genomes. This is done by altering the binary value of the bits position that are not valid as a cure for the current error. This operation is done by going through all the clones created earlier and mutating them in an attempt for better affinity values, which will, ultimately, lead to a future cure to the current error.

After mutation takes place, an evaluation of its results must happen in order to evaluate if a cure has been found, being this process very similar to those of calculating affinities. This behaviour, **DefineClonesAffinity**, implemented using the SB from the JADE framework, does precisely that. If, however, a cure is not found, the whole process will repeat itself, the old antibodies being replaced by the new, cloned, mutated ones that have a larger affinity towards the error's genome. The process will then resume from the DefineHighAffinityAbs behaviour and the cycle will only stop once a cure has been met.

4.3 Network Model

This algorithm, just like the previous one, receives three different inputs: the training set, which is composed by the genomes of the known cures, and will be used to train the new genomes; the error genome; and, finally, the word size, in bits. This algorithm's mechanism is represented in Figure 4.3.

This algorithm's agent start by launching the **NetworkInstance**, implemented using the OSB from the JADE framework that will determine the initial neighbours, an array of known

cures that will be used for training the new genomes. It then proceeds to launch the initial population of cells, based on this initial neighbours array.

To do so, the **LaunchInitialPop**, implemented using the SB from the JADE framework, will add neighbours to each of the B cells that initially constitute the network. It then proceeds to launch one **LaunchNewBCell**, implemented using the OSB from the JADE framework, for each of the initial B cells in the network.

The BCA receives three different inputs: the error genome, the candidate genome, and its neighbours. It then proceeds to launch two separate behaviours:

1. **EvaluateCure** – implemented using the OSB from the JADE framework that simply verifies if the candidate genome, which is the cells', is the cure for the given error.
2. **GetAffinities –** Parallel Behaviour that is responsible for the action of two sub-behaviours, **GetPairAffinity** and **GetNeighborsAffinity**.

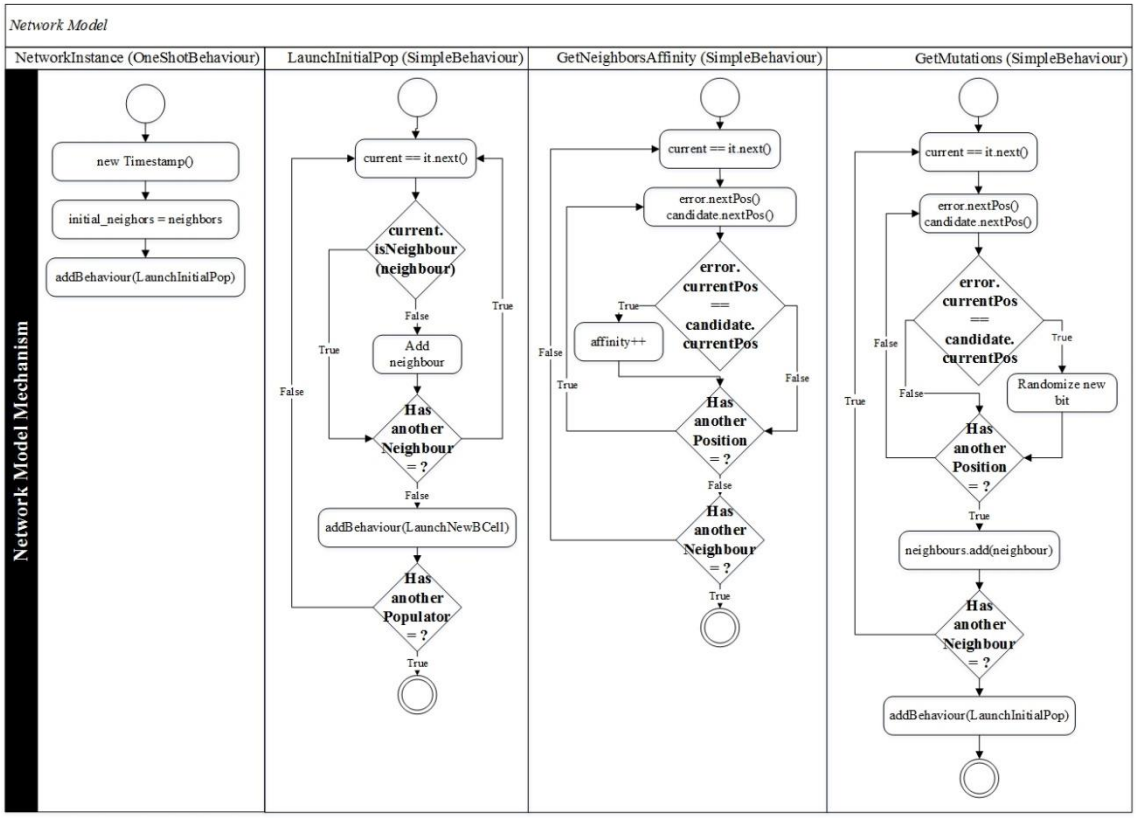


Figure 4.3 – Network Model Mechanism.

The first one simply gets the affinity between the error's genome and the B cell's genome by making a bit by bit comparison of each of the genomes and adds that affinity to the cell's

overall affinity towards the error. There are already several examples of this behaviour in this document, rendering it unnecessary to create another graphical representation for this behaviour.

The latter, implemented using the SB from the JADE framework calculates, for each of the cell's neighbour, its affinity towards the error. Once the **GetAffinities** Behaviour terminates, the **GetMutations**, implemented using the SB from the JADE framework, is launched. This behaviour will mutate the B cell's genome affinity times. This new mutations will constitute the population of the next step in the network and the whole process will repeat itself until a cure is found, much alike the **Clonal Selection Algorithm**.

4.4 Choosing the algorithm

For the testing of all three algorithms to take place in a secure, external intervention free environment, all the tests and data collection took place in a connectionless computer which main purpose is exactly that, of testing.

As it was already mentioned before, all three algorithms were launched on top of a Multi Agent system, supported by the JADE framework, in order to validate the data in a modular and distributed scenario. The algorithm started to process its inputs, which were passed to it through a user-friendly interface. After the training phase was completed, the algorithms started to look for an error endlessly.

This error was introduced by the previously named user interface. On success, meaning the cure for that error was found, there were two parameters measured:

- The time lapse between the error introduction in the system and the cure being found.
- The number of B Cells launched by the algorithm.

Each of the algorithms was tested with four initial inputs that constituted the initial population. To test them, 30 tests for six different byte size, totalising 180 tests for each of the three algorithms were made. The collected data was procedurally analysed as follows.

4.4.1 Collected Data Analysis

Since there is no developed work on AIS for an analysis towards its implementation to be made, it was necessary to develop a methodology capable of doing it. Therefore, in order to analyse the collected data two models were developed. For it, two simplistic models were designed.

4.4.1.1 Model's Development

The models' development was based on fuzzy logic. The first model's framework is described by Figure 4.4.

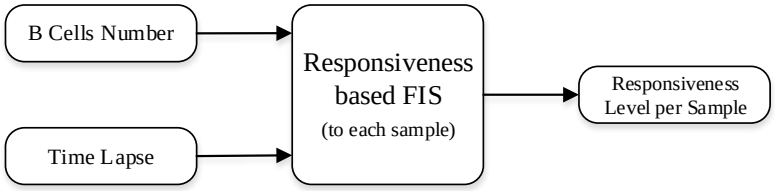


Figure 4.4 – Framework of the responsiveness based FIS (per sample).

The first model (Figure 4.4) uses the two performance indicators, the *B-Cell Number* and the *Time Lapse*, to define the system's *Responsiveness Level*. The *Responsiveness Level* $F(x(i,j))$, is defined by the FIS of each combined sample, i.e., for sample x_{ij}, where i represents the *B Cells Number* input and j the represents the *Time Lapse* input. This FIS is applied to all samples of all word sizes of the three algorithms. With the first model's outputs, the *Responsiveness Level* (per sample), the data are aggregated through the use of (1).

$$\bar{X}_k = \frac{\sum_{k=1}^{n} F(x(i,j))_k}{n} \quad \text{and} \quad \sqrt{S_k^2} = \frac{\sum_{k=1}^{n} (F(x(i,j))_k - \bar{X}_k)^2}{n-1} \quad (1)$$

Where k is the sample number for the *Responsiveness Level* result for each sample x_{ij}. The results from (1) were used on the second model (Figure 4.5).

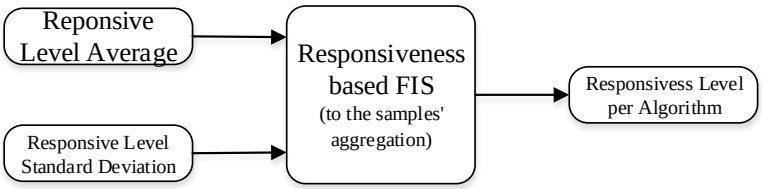


Figure 4.5 - Framework of the responsiveness based FIS (to the samples' aggregation per each algorithm).

The model's (Figure 4.5) result is the *Responsiveness Level* per algorithm. As higher the value is, the better is the algorithm result, i.e., the algorithm with the highest *Responsiveness Level* is the best algorithm.

4.4.1.2 Models' Implementation

The performance indicators data was analysed based on two different scales. The time lapse indicator was analysed based on Table 4.1.

These scales were defined according to information provided by automotive industry experts. The values were chosen taking into account that the average industrial PLC cycle, according to those same experts' opinion, takes approximately 100ms to perform.

Table 4.1 – Time lapse scale used on the FIS.

Scale	Description
Very Fast	≤ 5 ms
Fast	Between 5 and 10 ms
Moderate	Between 10 and 50 ms
Slow	Between 50 and 100 ms
Very Slow	≥ 100 ms

That being said, for values under 10ms, the attributed classification was "Fast" and "Very Fast" (for those under 5ms). From a range of 10 to 50ms, a "Moderate" classification was assigned. For values between 50ms and 100ms, "Slow" classification was attributed. For over 100ms, meaning it exceeds the average industrial PLC cycle, a "Very Slow" classification was used.

Table 4.2 – B Cells number used on the FIS.

Scale	Description
Very High	> 60 B Cells
High	Between 45 and 60 B Cells
Moderate	Between 30 and 45 B Cells
Low	Between 15 and 30 B Cells
Very Low	≤ 15 B Cells

The B cells number indicator was analysed based on Table 4.2. The number of BCAs per sample, the BCAs average and the BCAs per algorithm are also based on the following scale: very high, high, moderate, low and very low – this means that the higher the number of BCAs, the higher will the scale level be.

The standard deviation is defined by the following scale: low, moderate and high. All the defined scales were defined by experts.

Lastly, the FIS rules were defined. For the first model, eight rules were defined, as for the second, four sufficed. Yet again, all the rules were defined according to the experts' knowledge.

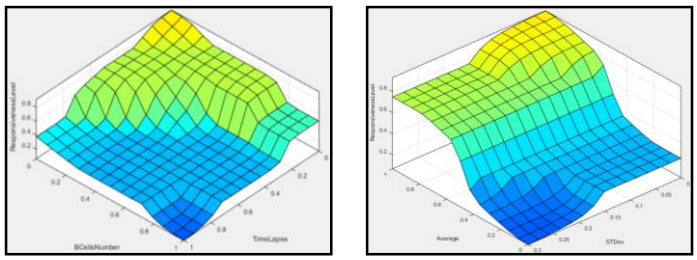
4.4.1.3 Models' Validation

As validation, the models were submitted to two different tests. Firstly, the extreme conditions' test (Table 4.3); secondly, the face validity test (Figure 4.6).

Table 4.3 – Extreme conditions' test for both models.

Time	B Cells	Responsiveness Level
0	0	0.917
1	1	0.069
Average	**Standard Deviation**	**Responsiveness Level**
1	0	0.931
0	0.3	0.069

The extreme conditions' test presented the appropriate response. Consequently, there were no changes to be made for both models. The face validity test, also presented the expected response, which results in a validation of both models.



(a) (b)

Figure 4.6 - Face validity test (a) Responsiveness level FIS (per sample) (b) Responsiveness level FIS (per algorithm).

4.4.1.4 Results' Analysis

For a better analysis of the results given as output of the aforementioned model, a graphical representation, as the one presented in Figure 4.7, was in order.

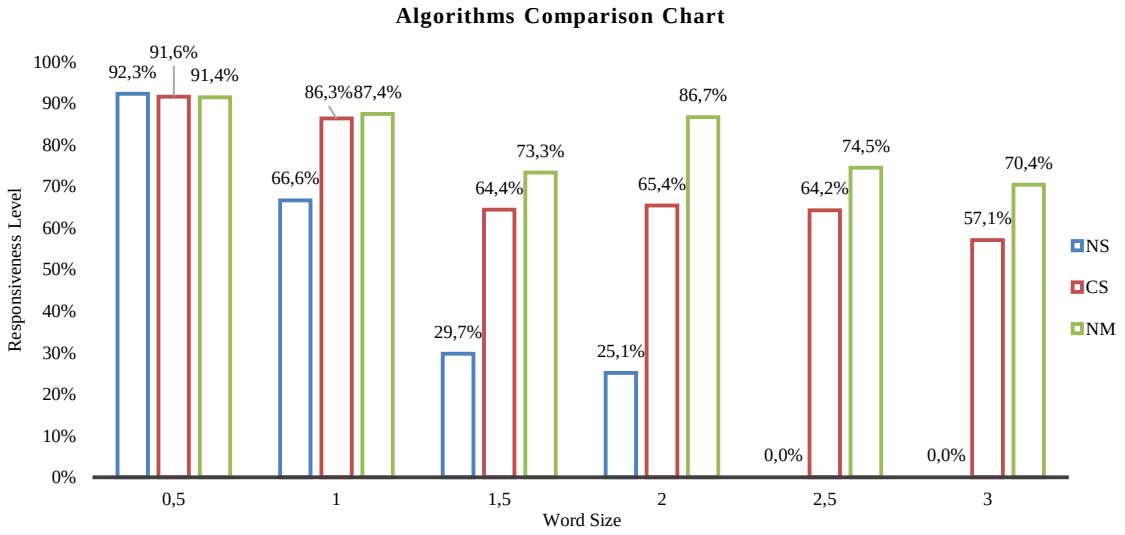


Figure 4.7 – Algorithms comparison chart.

This graphical representation allows for a better retrospective of the outputted results. These results are hereby explained and clarified as follows:

- **Cure finding speed:** All three algorithms revealed themselves capable of finding a cure for the error being treated in an irrelevant time lapse when compared to those of the PLC's cycle time, which is between seven and one hundred milliseconds (7~100ms) – this scale was based on the automotive industry case).
- **Algorithms Capability:** Considering an operating station as a station that needs, at least, one byte to read and write all inputs and outputs, respectively, it is safe to say that, from the three presented algorithms, two of them, Clonal Selection and Network Model, present fair, even good, results when tested on words larger than one byte, offering above 50% performance for words with a three byte size.
- **Size coverage:** This study also offers a view on which algorithm should be used, according to the word size being used, since, for words smaller or equal than four bits, the Negative Selection Algorithm should be used. On the other hand, for words larger than four bits, both Clonal Selection and Network Model could be used, despite the latter presenting far better results than the first.
- **Expected decaying:** Even though the test results were rather satisfactory, an expected decaying on the responsiveness level occurred, as the word size grew bigger.
- **One to rule them all:** Despite not being, as the results suggest, the best algorithm for words smaller than four bits, the Network Model algorithm is the one that presents better results for all the remaining case scenarios. Moreover, the first case scenario, for words four bits sized, should be deemed irrelevant for the present analysis since all the algorithms present extremely good results for that specific case scenario.

5

Implementation

In this chapter, an approach to the implementation done in this work will be made. Since this is a decentralized, learning system, the Artificial Immune System (AIS) architecture previously defined was implemented using the Java programming language under a behaviour oriented platform, the JADE (Java Agent Development Environment) framework (Bellifemine et al., 1999).

5.1 Agents

For a better explanation to take place, a detailed overview on all the agents used will be given, along with its role in the whole system.

5.1.1 Diagnosis Agent

Agent responsible for directly communicating with the robot/part of hardware that is going to be supervised by it. It constantly reads the current process data of the hardware and, shall it be outside the ruled parameters, a distress signal is emitted and the algorithms are launched.

5.1.1.1 Library Setup

On its launch, this agent starts by setting up the operations the device it is going to supervise is capable of executing and by storing its binary representation in memory. To do so, a library, driver if you may, file is uploaded to the agent so it can load the initial parameters under which the life cycle of the device should run, if no error were to be found.

Since this is supposed to be a generic system, to diagnose every type of robot/hardware, an interface, that is to be implemented by those who want to use this system, is provided to make it possible for the system to accept a variety of different driver files.

This library loading mechanism is illustrated in Figure 5.1.

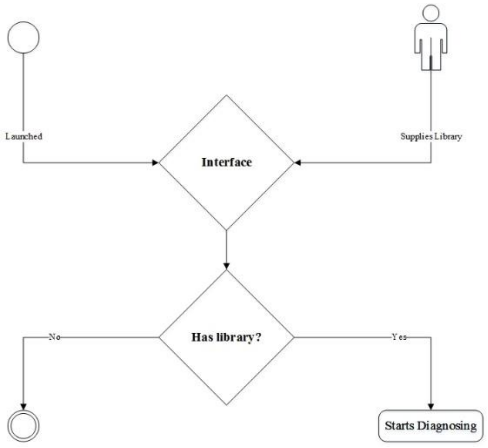


Figure 5.1 – Library loading by the Diagnosis Agent (DA).

This figure shows how an operator should proceed when using this architecture, indicating that, upon launch, the operator should supply a library, using a user-friendly interface. If such library is provided than the DA can start the diagnosing procedures.

An illustrative representation of how the DA interfaces with the library is given in Figure 5.2.

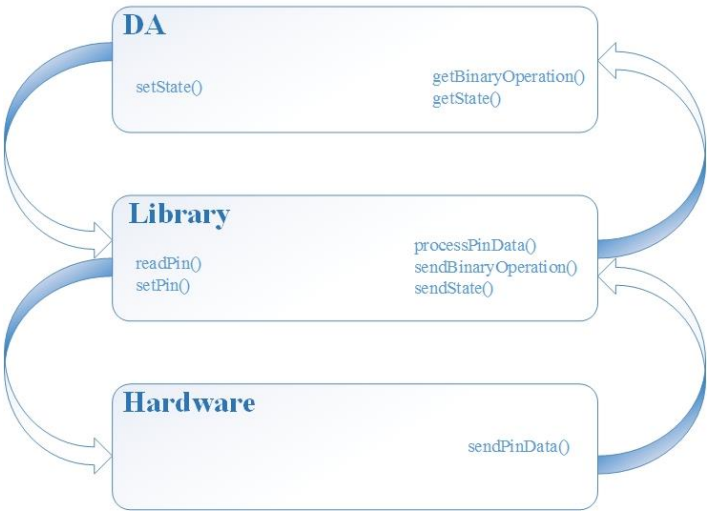


Figure 5.2 – DA Interaction with the Library.

This figure indicates the methods one should implement in order for the architecture to interface properly with the resource it is diagnosing. That being said, the DA can send information

to the resource, telling it what to do in order to solve a cure. On the other hand, it can also read information from the resource it is diagnosing such as reading the current operation.

5.1.1.2 Behaviours

Once the library is processed by the agent and all the parameters are correctly set up, the agent is ready to be set up.

Before anything, the DA must have a mechanism that allows it to request a group, whenever one is available. The **AskForGroup** behaviour**,** implemented using the AchieveREInitiator Behaviour (AREIB) from the JADE framework, enables just that by requesting a group to the Grouped Diagnosis Agents (GDAs) in the system, if the DA is not yet grouped. An overview on this behaviours mechanism is given in Figure 5.3.

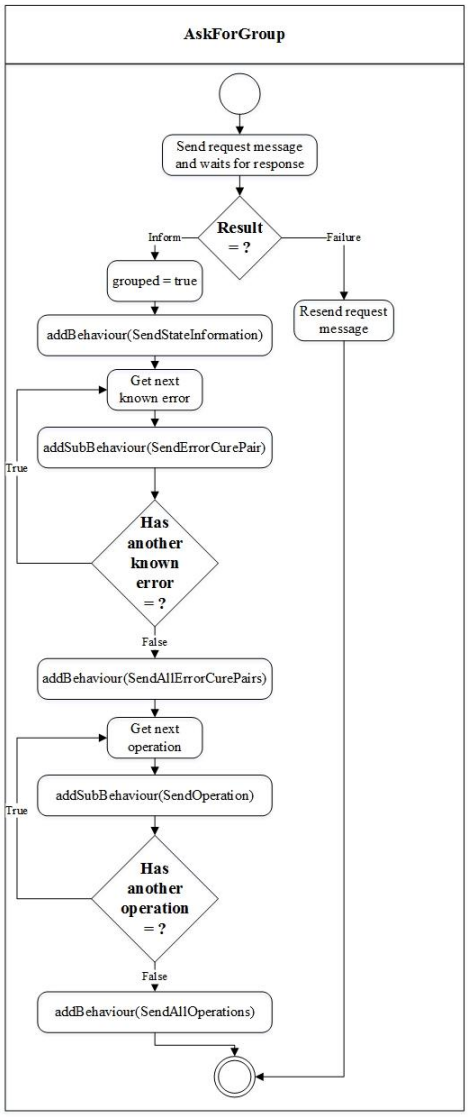


Figure 5.3 – AskForGroup AchieveREInitiator Behaviour.

If any GDA accepts the DA, then it starts sending all the errors and operations it knows from the library that was passed to it. To do so, it uses the **SendAllErrorCurePairs / SendAllOperations** behaviours, implemented using the Parallel Behaviour (PB) from the JADE framework. They send both the error/cure pairs and the operations of any given DA. To do so, it uses two different sub-OSB that will be approached next. This behaviours diagram is exactly the same and is represented in Figure 5.4.

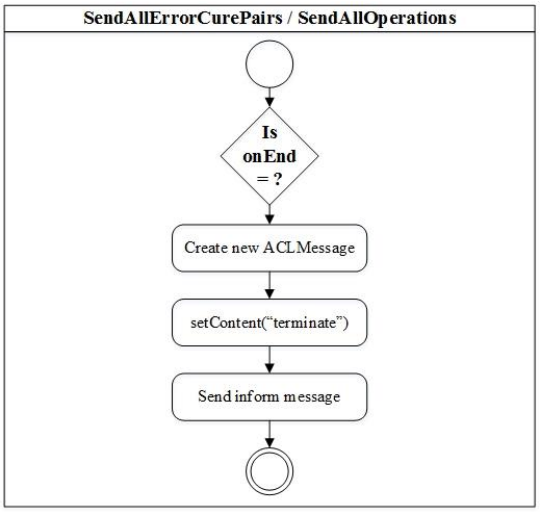


Figure 5.4 – SendAllErrorCurePairs / SendAllOperations Parallel Behaviours.

The aforementioned behaviours in charge of sending a single operation or error/cure pair are called **SendOperation / SendErrorCurePair** and were implemented using the OSB from the JADE framework. They are very similar among them since the only difference that tells them apart is the content of the message sent, which, according to the objective, it sends either the operation or the error/cure pair. An overview on this behaviours mechanism is given in Figure 5.5.

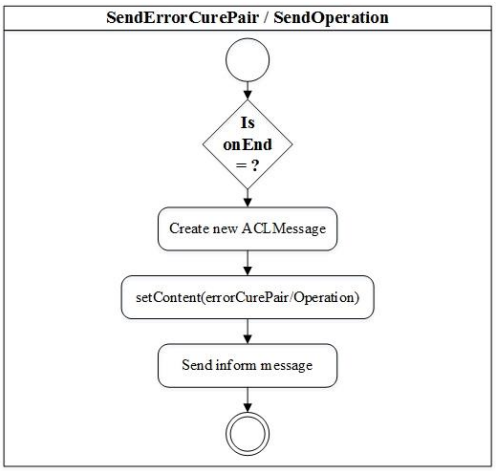


Figure 5.5 – SendErrorCurePair / SendOperation One Shot Behaviours

These behaviours conclude the setup, if you may, procedure of any DA in this architecture. Once this is complete, the DA is ready to initiate its procedures of actually diagnosing its resource.

To do so, it uses two separate behaviours, one that constantly observes the current operation being done in the resource and the other that reads the state information of the DA. That being said, the **OperationsObserver**, implemented using the Ticker Behaviour (TB) from the JADE framework. It is constantly verifying if the current skill being executed by the device being supervised is within the allowed parameters. If not, the state of the agent is modified. An overview on this behaviours mechanism is given in Figure 5.6.

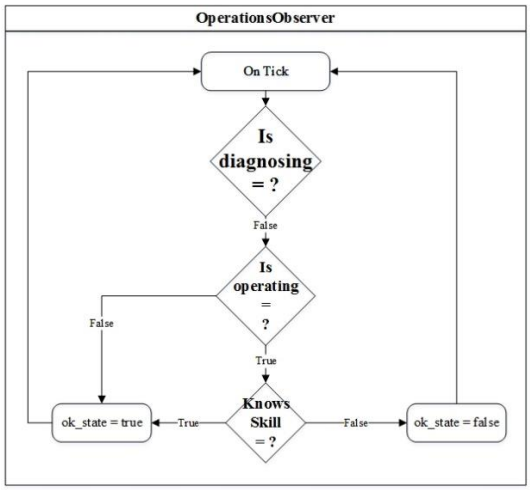


Figure 5.6 – SkillsObserver Ticker Behaviour.

The other one that was mentioned, the **StateObserver**, implemented using the TB from the JADE framework. This behaviour is constantly verifying if the state of the agent is unaltered, i.e., if there was no error found. If there was, the error detecting mechanisms are launched. This behaviours' mechanism can be observed in more detail in Figure 5.7.

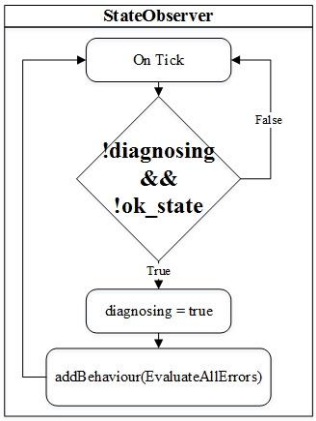


Figure 5.7 – StateObserver Ticker Behaviour.

If the DA is grouped, the information obtained from the two above behaviours is reported top wards, to the GDA. This is done by the **SendStateInformation** behaviour, implemented using the TB from the JADE framework**.** It is constantly informing the GDA in which the DA is registered of its current state. This behaviours mechanism can be observed in Figure 5.8**.**

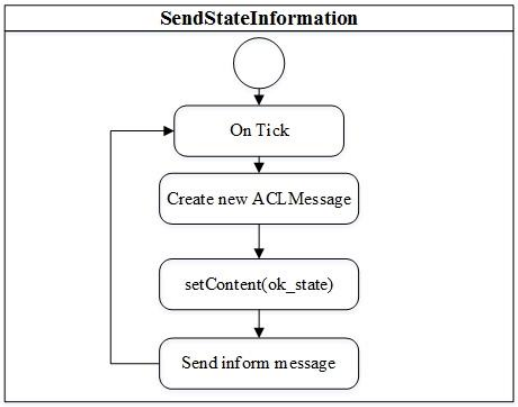


Figure 5.8 – SendStateInformation Ticker Behaviour.

If an error is found, the DA launches the **EvaluateAllErrors** behaviour, which was implemented using the PB from the JADE framework. This behaviour's sub-behaviours are EvaluateIndividualError which were implemented using the OSB from the JADE framework.

The **EvaluateIndividualError,** will evaluate whether the agent contains, in its library, the resolution for the current error and can be observed in Figure 5.9.

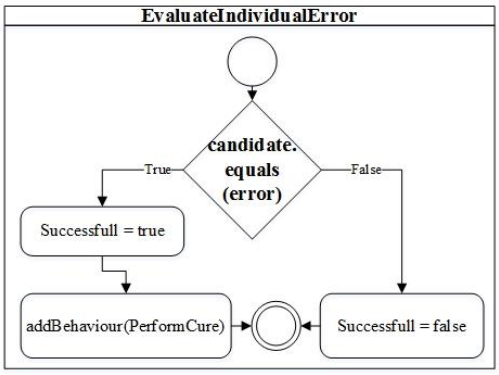


Figure 5.9 – EvaluateIndividualError One Shot Behaviour.

If a cure is found, the **PerformCure**, which was implemented using the TB from the JADE framework is launched. This behaviour will "tick" as many times as there are seconds for the found cure to take effect. Once this number is reached, the agent resets its "ok_state" and "diagnosing" states to its defaults, true and false, respectively.

If the **PerformCure** behaviour was set by a GDA command, then an "inform_to_send" FIPA-Inform message is sent to the GDA. A detailed overview of this behaviour can be viewed in Figure 5.10.

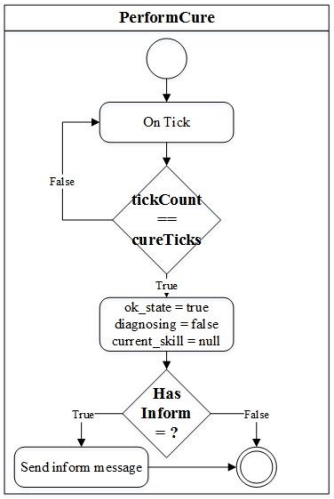


Figure 5.10 – PerformCure Ticker Behaviour.

When all the sub-behaviours terminate their execution, the aforementioned parallel behaviour (the **EvaluateAllErrors**) will assess if any of the sub-behaviours found a cure. If so, the cure will be immediately applied; if not, the error will be spread upwards to the agent's GDA, if it exists, so it can try to find a cure. Otherwise, it will launch the algorithm for finding a cure. This behaviours mechanism can be duly noted in Figure 5.11.

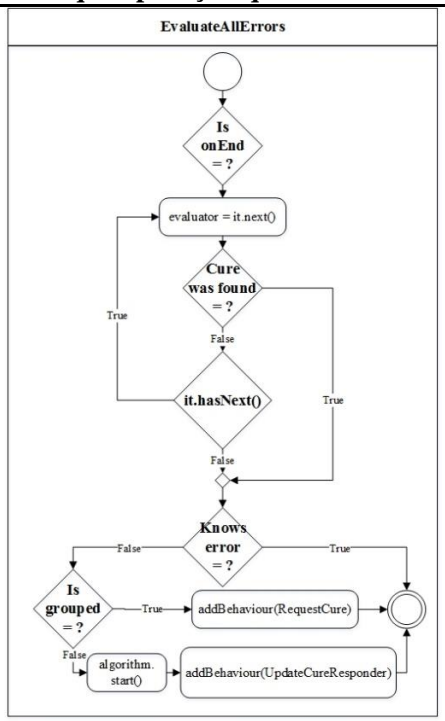


Figure 5.11 – EvaluateAllErrors Parallel Behaviour.

If the DA is grouped and it cannot find a cure in its library, it launches the **RequestCure** behaviour, implemented using the AREIB from the JADE framework. This behaviour requests a cure to the GDA into which it is registered into. If the GDA has a cure it will immediately return it; otherwise, it informs the DA that a cure finding algorithm has been launched. This behaviour is illustrated in Figure 5.12.

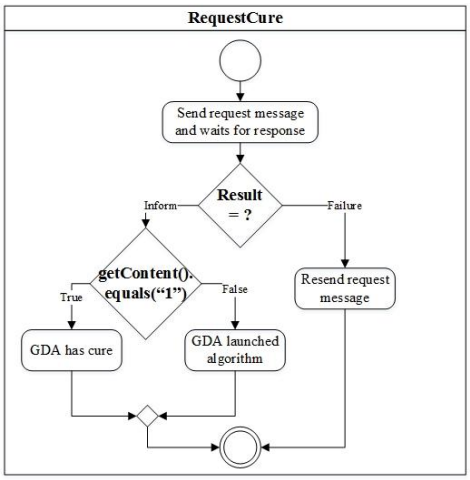


Figure 5.12 – RequestCure AchieveREInitiator Behaviour.

Since a request can be made to the group in order to find the cure, the DA must also be able to receive a cure from the GDA, if one is ever to be found. This is done by the **FixErrorByGroupResponder** behaviour, implemented using the AchieveREResponder

(ARERB) Behaviour from the JADE framework. It waits for a communication from the GDA if a cure ever arrives from it. If so, this behaviour launches the appropriate mechanisms to apply the received cure. This behaviour is illustrated in Figure 5.13.

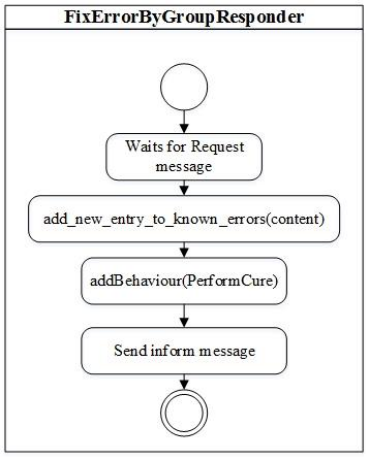


Figure 5.13 – FixErrorByGroupResponder AchieveREResponder Behaviour.

5.1.1.3 Class Diagram

The class diagram for the DAs used in this architecture is detailed, in Figure 5.14, where it is possible to observe its own structure. Since they were already mentioned and explained, there will not be any reference to the behaviours of this agent in the figure.

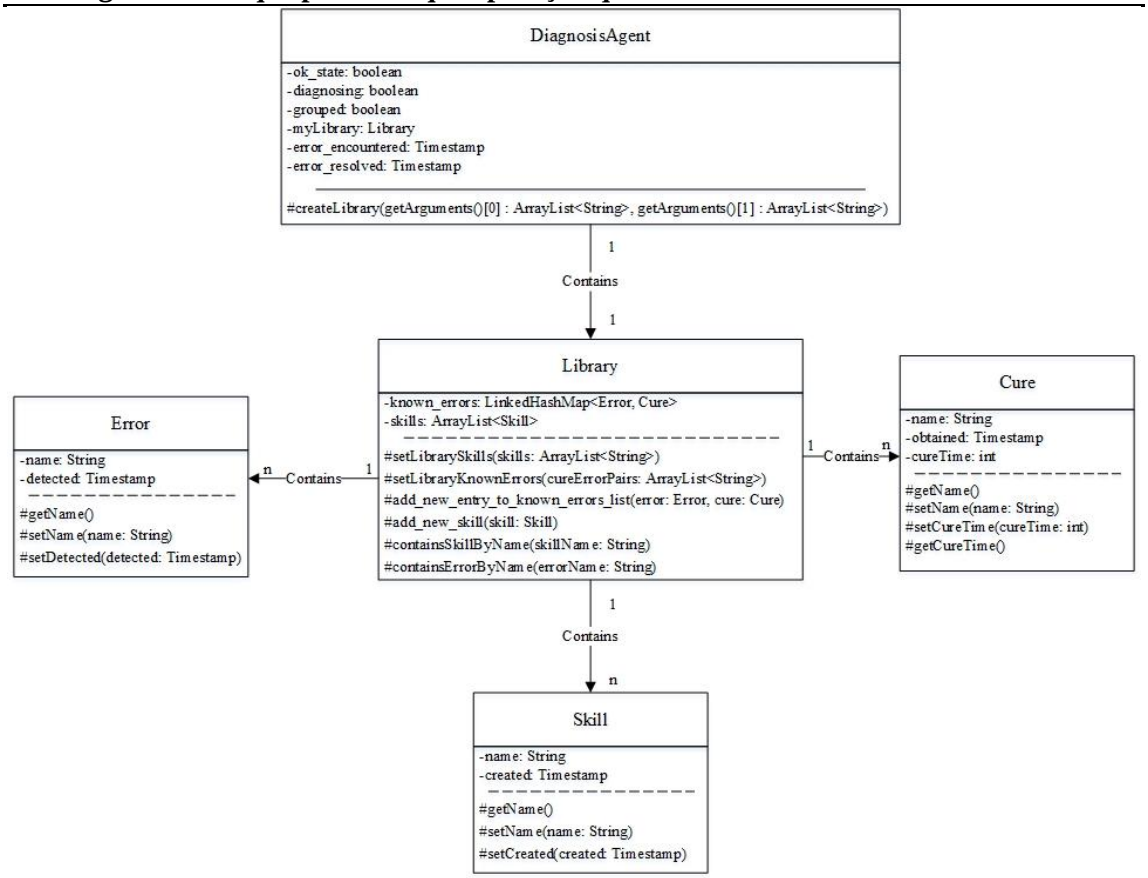


Figure 5.14 – Diagnosis Agent's Class Diagram.

By taking a closer look to the above figure, it is possible to understand that all the errors, cures and skills known to any given DA were stored in a Library, unique to each DA and that accounts for all the known errors and skills of its agent.

5.1.2 Grouped Diagnosis Agent

Agent responsible for handling all the information in and out of the DAs by which it is composed of. If the DA encounters an error for which it has no cure, it is the GDA's responsibility to deal with this problem by launching an AIS algorithm or by looking for a cure in its library, provided by another DA.

This agent's library of known errors is composed of the known errors of each of the DAs that are associated with it.

5.1.2.1 Behaviours

This agent launches only two behaviours on its launch, which are no more than two FIPA Protocol compliant behaviours that allow the GDA to communicate with the DAs that are grouped under its responsibility. These behaviours are the ones in charge of the setup procedure, if you

may, of the GDAs. They are the ones responsible for acknowledging the new DAs that may want to register itself and any of the error/cure pairs and operations known to them.

That being said, the **ReceiveDiagnosisAgentGroupRequest** behaviour (that was implemented using the ARERB from the JADE framework) waits for a group request from a DA, and grants it, or not, access to the group, thus allowing it to have access to more cures from the others DA in the group, without actually knowing them. This behaviour is demonstrated in Figure 5.15.

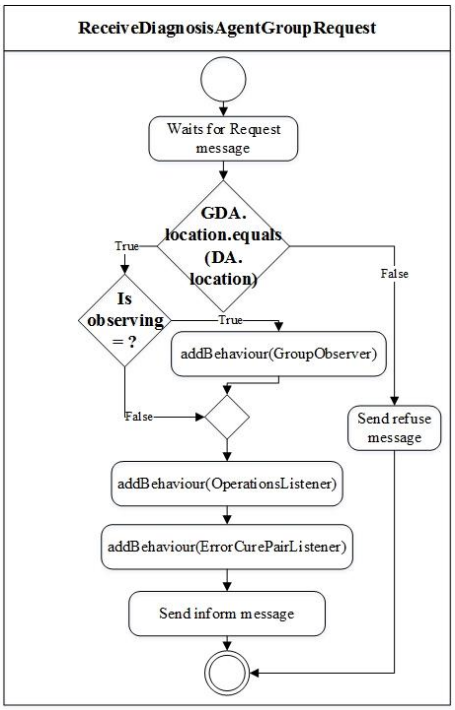


Figure 5.15 – ReceiveDiagnosisAgentGroupRequest AchieveREResponder Behaviour.

As it is observable in the above figure, the **ErrorCurePairListener** behaviour was implemented using the TB from the JADE framework. It is launched in order to receive messages sent by the DAs, upon their registration, which content is an error/cure pair that is to be stored in the GDA in which the DA is registered. This behaviour is depicted in Figure 5.16.

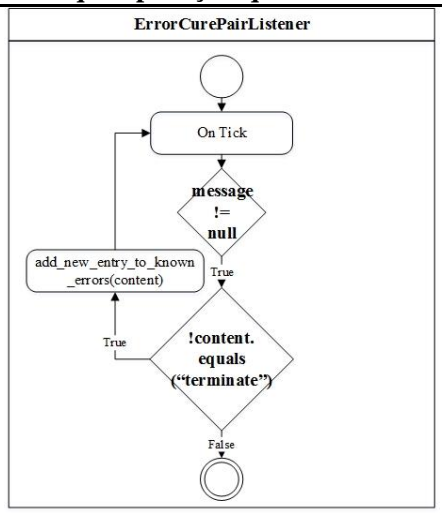


Figure 5.16 – ErrorCurePairListener Ticker Behaviour.

The same goes for the **OperationListener b**ehaviour, implemented using the TB from the JADE framework. This behaviour receives messages sent by the DAs, upon their registration, which content is an operation that is to be stored in the GDA in which the DA is registered. This behaviour is depicted in Figure 5.17.

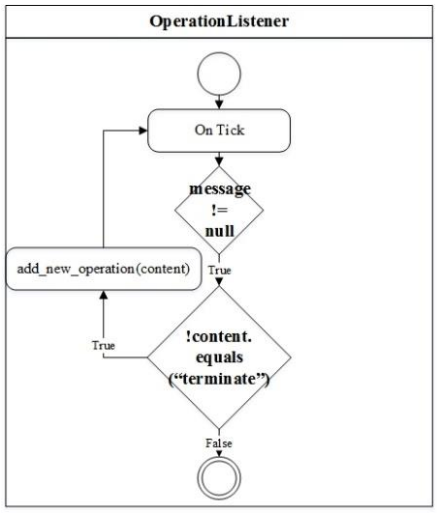


Figure 5.17 – OperationListener Ticker Behaviour.

Once the loading of every operation and known error/cure pair is done, the GDA can start its operations. For the GDA to keep track of the state of the DAs, the **GroupObserver** behaviour, implemented using the TB from the JADE framework is launched. It constantly launches the Parallel Behaviour **EvaluateStates**, responsible for an assessment of the state of each of the DAs registered in a given GDA. This behaviour can be observed in Figure 5.18.

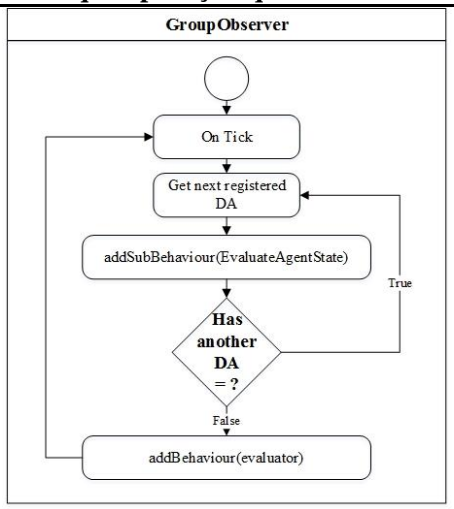


Figure 5.18 – GroupObserver Ticker Behaviour.

The aforementioned **EvaluateStates** behaviour was implemented using the PB from the JADE framework. It is responsible for launching the sub-behaviour **EvaluateAgentState** that will report the given DA's agent state to the GDA. If any of the DAs is having faulty behaviour, then the state of the GDA changes and error recovery mechanisms are launched. This behaviours can be observed in Figure 5.19.

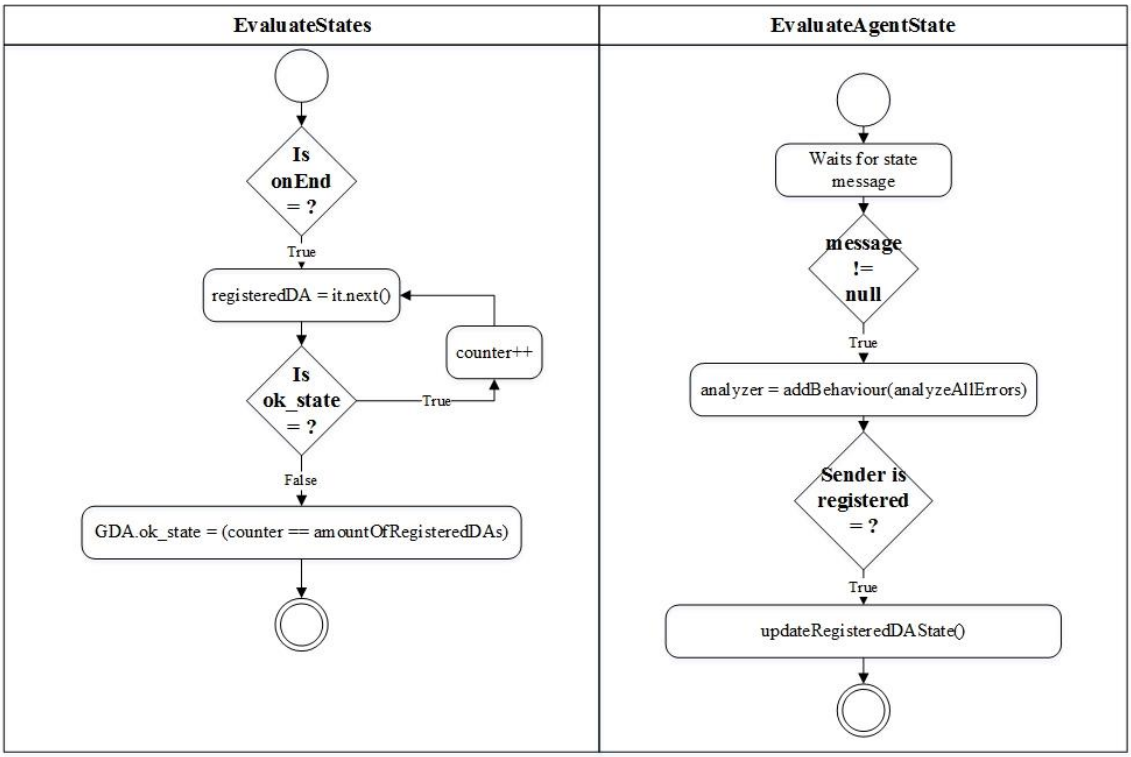


Figure 5.19 – EvaluateStates Parallel Behaviour and EvaluateAgentState Sub-Behaviour.

For the GDA to be able to answer the cure requests from the DAs to it assigned, the **RequestCureResponder** behaviour, implemented using the ARERB from the JADE framework

was used. It waits for a cure request from a DA so it knows that same DA has encountered an error and needs the GDA's help to solve it. The error will then be solved with either the other known errors in the GDA or with an AIS algorithm. This behaviour is hereby represented in Figure 5.20.

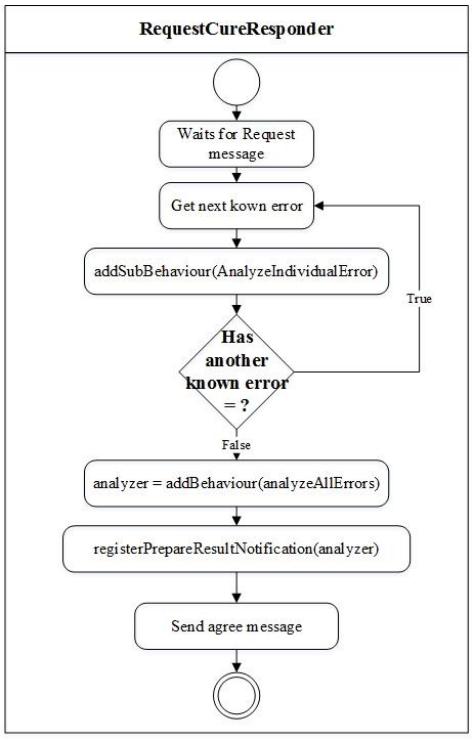


Figure 5.20 – RequestCureResponder AchieveREResponder Behaviour.

Therefore, and once a request arrives to the GDA, it launches the **AnalyseAllErrors** behaviour, implemented using the PB from the JADE framework. It will verify if the current error sent by any given DA and compares its genome to the known errors'.

To do so, it uses the **AnalyseIndividualError** behaviour, implemented using the OSB from the JADE framework. This behaviour simply compares a given genome to the errors and evaluate if it is the cure or not.

If a cure is found, it notifies the DA and requests to perform cure. If not, it launches the AIS algorithm. A graphical illustration of these behaviours follows in Figure 5.21.

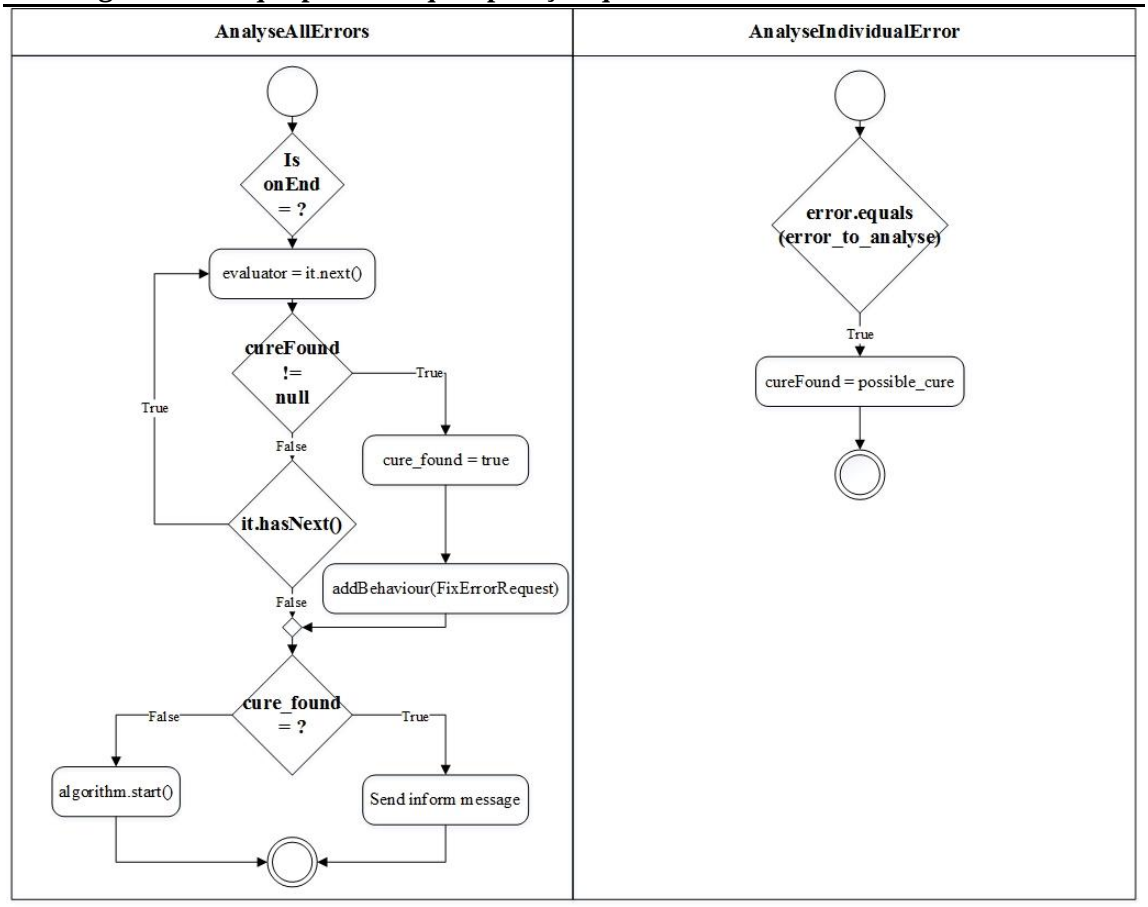


Figure 5.21 – AnalyseAllErrors Parallel Behaviour and AnalyseIndividualError One Shot Behaviour.

If a cure is found inside the GDA, the **FixErrorRequest** behaviour, which was implemented using the AREIB from the JADE framework is launched. It sends, to the DA, the cure that was found by the algorithm launched by the GDA. An illustration for this behaviour comes in Figure 5.22.

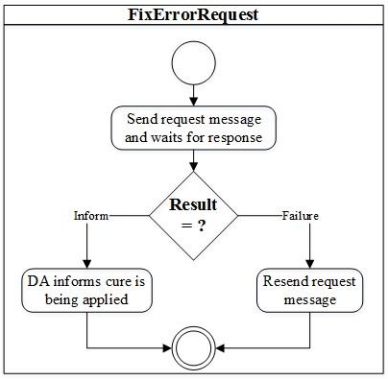


Figure 5.22 – FixErrorRequest AchieveREInitiator Behaviour.

Since the GDA needs to be notified once the cure is applied in the DA, it launches the **UpdateCuresResponder** behaviour, implemented using the ARERB from the JADE framework.

This behaviour waits for a B Cell Agent (BCA) to contact it for storing the cure. If the cure presented by the BCA is a valid one, than the cure is accepted, stored and sent to the DA. This behaviour is demonstrated in Figure 5.23.

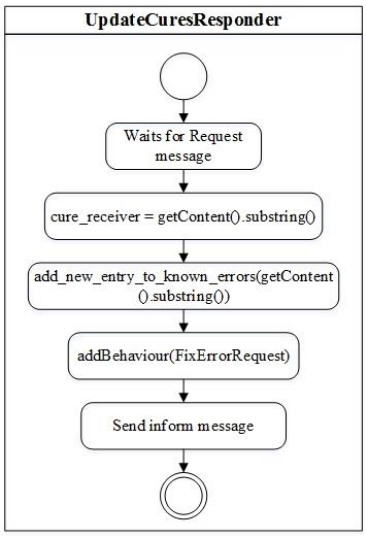


Figure 5.23 – UpdateCuresResponder AchieveREResponder Behaviour.

5.1.2.2 Class Diagram

The class diagram for the GDAs used in this architecture is similar to that of the DAs and is detailed, in Figure 5.24, where it is possible to observe its own structure. Once again, there will not be any reference to the behaviours of this agent in the figure.

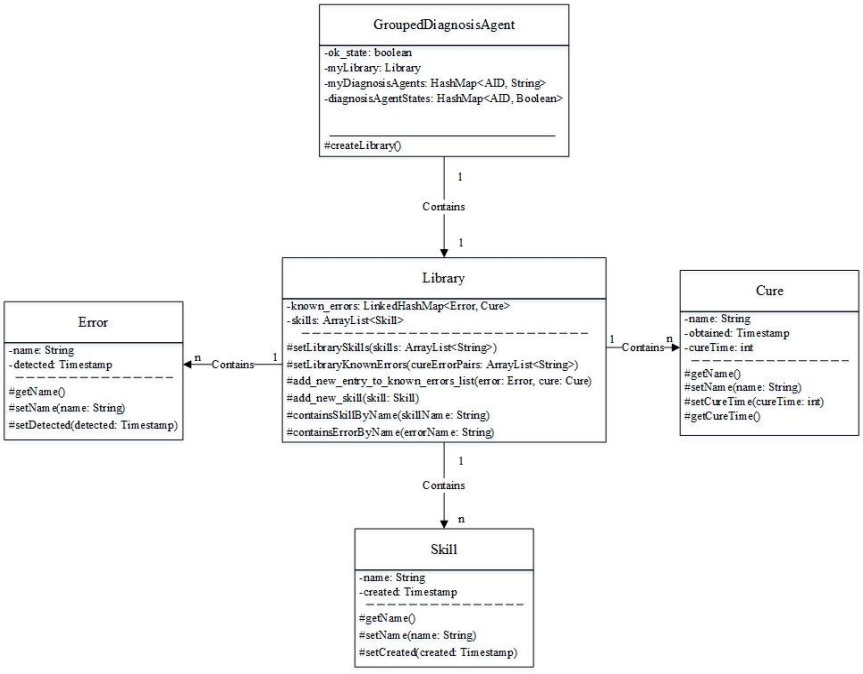


Figure 5.24 – Grouped Diagnosis Agent's Class Diagram.

As in the DA's class diagram, it is possible to observe that the GDA also possesses a Library that is used to account for all the errors, cures and skills that its GDA knows.

5.1.3 B Cell Agent

Agent responsible for evaluating its own genome towards the errors. This is done by calculating the BCA genome's affinity towards the errors. This is done by incrementing a counter whenever one of the binary genomes has '1' and the other has '0'.

This agent was adapted to fit better with the chosen algorithm, i.e., as a member of a B Cell network it much behave as such. To do so, if no cure is found with its genome, it must be able to expand the network. Moreover, it must also be able to, once commanded to do so, commit suicide, since it will no longer be needed once the cure was found by another element in the network.

5.1.3.1 Behaviours

For this agent to behave as it is supposed to, several behaviours are required.

The **AnalyseError** behaviour that was implemented using the OSB from the JADE framework. It compares the error's genome to its agent's genome, by comparing its differences in the binary code that represents their genome.

This is achieved by comparing each position of both genomes and checking if they have different binary values. If this statement occurs for every single position in the genome, then a cure has been found and the BCA notifies the DA that it found a new cure, through the **CureFoundRequest** behaviour.

Otherwise, the BCA will contact the Cure Provider Agents (CPA) in the system in order to check if a cure is available in the database. This is done by launching the **RequestCureCPA** behaviour. It will also keep expanding the network by launching the **GetAffinities** behaviour from the Network Model Algorithm. An illustrative example of this behaviour can be viewed in Figure 5.25.

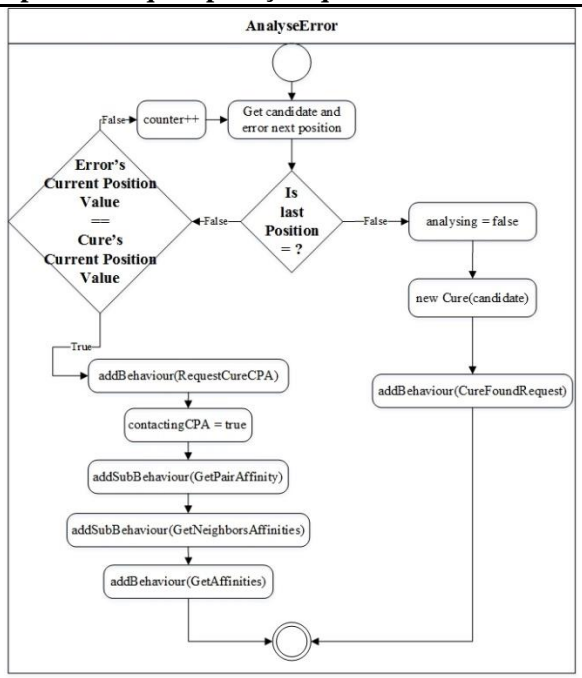


Figure 5.25 – AnalyseError One Shot Behaviour.

The **CureFoundRequest** behaviour was implemented using the AREIB from the JADE framework. It sends a message to the DA/GDA that ordered its launch, informing it has found a cure and making it available to this agents. This behaviour can be observed in Figure 5.26.

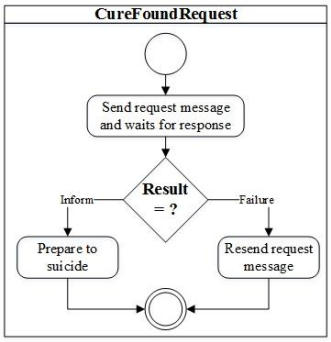


Figure 5.26 – CureFoundRequest AchieveREInitiator Behaviour.

The **RequestCureCPA** behaviour was implemented using the ContractNetInitiator Behaviour (CNIB) from the JADE framework. It sends a message to all the CPAs in the cloud asking for a cure to the current error, given that it was not capable of solving it by itself. This behaviour can be observed in Figure 5.27.

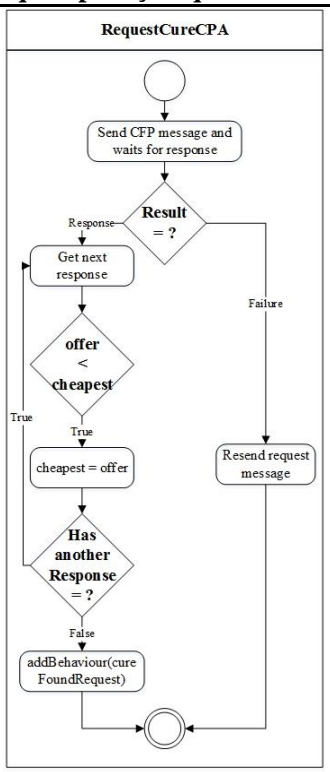


Figure 5.27 – RequestCureCPA ContractNetInitiator Behaviour.

The **SuicideResponder** behaviour was implemented using the ARERB from the JADE framework. It waits for a message from the DA/GDA for him responsible, according to which its services are no longer needed and may be terminated. This behaviour can be observed in Figure 5.28.

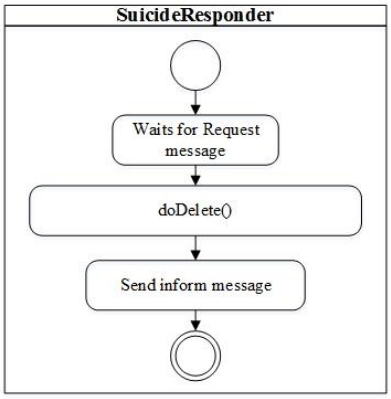


Figure 5.28 – SuicideResponder AchieveREResponder Behaviour.

5.1.3.2 Class Diagram

The class diagram for the BCAs used in this architecture is as simple as it gets and is represented in Figure 5.29, where it is possible to observe its own structure.

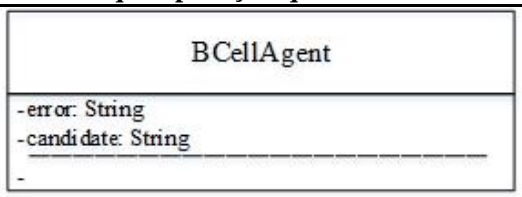


Figure 5.29 – B Cell Agent's Class Diagram.

5.1.4 Cure Provider Agent

Agent responsible for providing a cure to the BCAs when they cannot find a cure for the current error. This agent has an associated database with which it communicates in order to get and store new cures for the error requested by the BCA that is contacting it.

5.1.4.1 Behaviours

The associated database is generated on the agent's launch and the agent must be able to answer any incoming requests from the BCAs. Also, the agent needs to be able to update the database whenever new information is obtained.

Hence, and as in the DA and the GDA, this agent has what can be called as a setup stage where the database is created and filled with the initial data. To do so, the **CreateAssociatedDatabase** behaviour was implemented using the OSB from the JADE framework. This behaviour creates a database with three tables: cure, error and pairs, containing the cures, the errors and the pairs they make, respectively (if it does not exists already). This behaviour is presented in Figure 5.30.

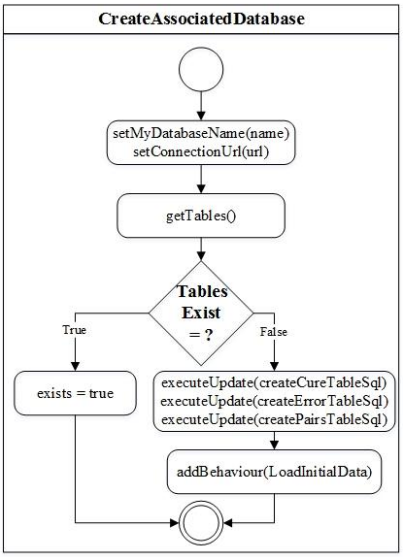


Figure 5.30 – CreateAssociatedDatabase One Shot Behaviour.

Once the database is created, and in the event there was not any to start off, the **Load Initial Data** behaviour (implemented using the SB from the JADE framework) is launched. It is responsible for loading all the data of a given CPA errors, cures and pairs.

This behaviour fills the database according to the data file that instantiates a CPA. If no file is provided, the database is initialized with no data. The aforementioned behaviour is displayed in Figure 5.31.

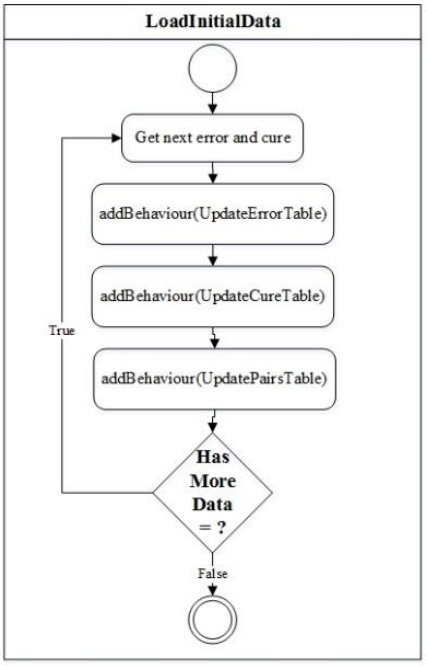


Figure 5.31 – LoadInitialData Simple Behaviour.

This completes the setup procedures for this agent. It must now be able to answer any incoming cure requests of any BCA that contacts it.

To do so, it uses the **CureResponder** behaviour, implemented using the ContractNetResponder Behaviour (CNRB) from the JADE framework, that waits for a BCA request for a cure.

It then proceeds to consult the database in order to infer if it has a cure for the requested error. If so, the cure is sent back to the BCA, if not, an error message informing it has no cure for that error is sent. This behaviour can be observed in Figure 5.32.

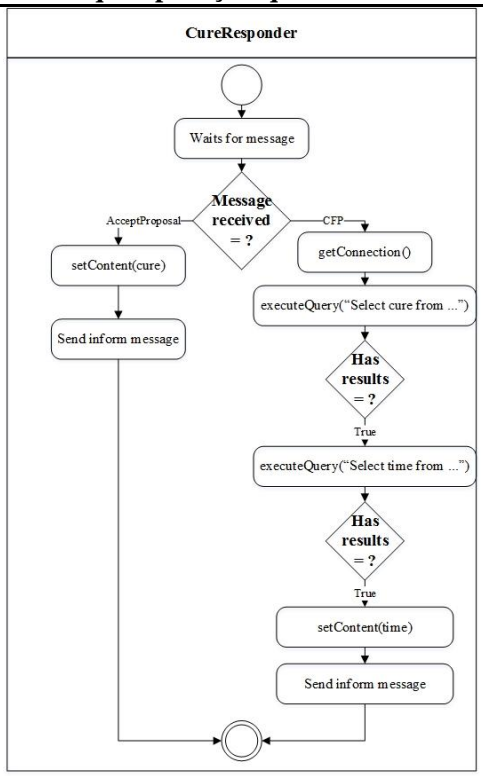


Figure 5.32 – CureResponder ContractNetResponder Behaviour.

Either for the setup procedures as for the cure providing procedures, the agent uses the **UpdateCureTable, UpdateErrorTable, UpdatePairsTable** behaviours, implemented using the OSB from the JADE framework. This behaviours are used to update its CPA's database tables, by either adding new information, or removing outdated information. Figure 5.33 graphically demonstrates this behaviour.

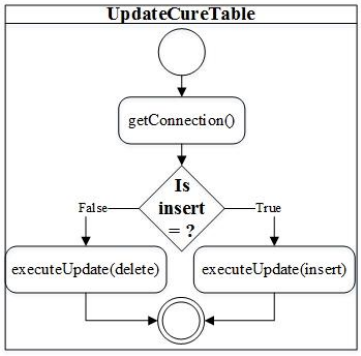


Figure 5.33 – UpdateCureTable One Shot Behaviour.

5.1.4.2 Class Diagram

Once again, and given its simplicity, the class diagram presented in Figure 5.34, refers to the CPAs database's hierarchical organization and not to the agent itself. That being said, suffices to say the database is organised in errors, cures and pairs.

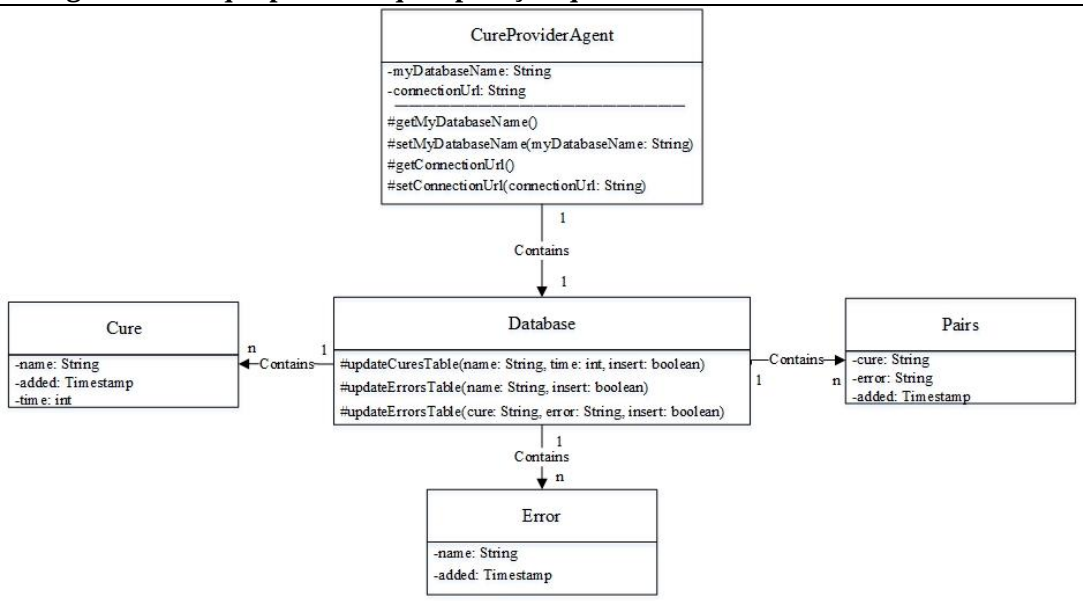


Figure 5.34 – Cure Provider Agent's Class Diagram.

As it is possible to observe from the above figure, the CPA communicates with a database that is constituted by three different tables, one for the cures, one for the errors and the last one for the pairs.

5.2 Communications

Considering this is a decentralised system, the agents need to be capable of performing tasks without being physically present in the same place. To do so, a communication of some sort needed to be implemented so the agents could communicate among themselves in order to perform tasks that require the intervention of more agents than only themselves.

As such, for this system to work properly, several FIPA Protocol compliant communication mechanisms were implemented: **FIPA Contract Net and FIPA Request** protocols. The first one is an auction based interaction protocol that accepts the better offer. The latter refers to a request-response mechanism protocol.

There are two main functioning methods for this system, when it comes to communications, and those depend on whether the DA where the error was found is grouped or not. If it is grouped, a message to it must be sent and the group will deal with the all cure finding mechanism. If not, the DA itself must handle the cure finding mechanism. First, a view on the grouped diagnosis situation will be given in Figure 5.35.

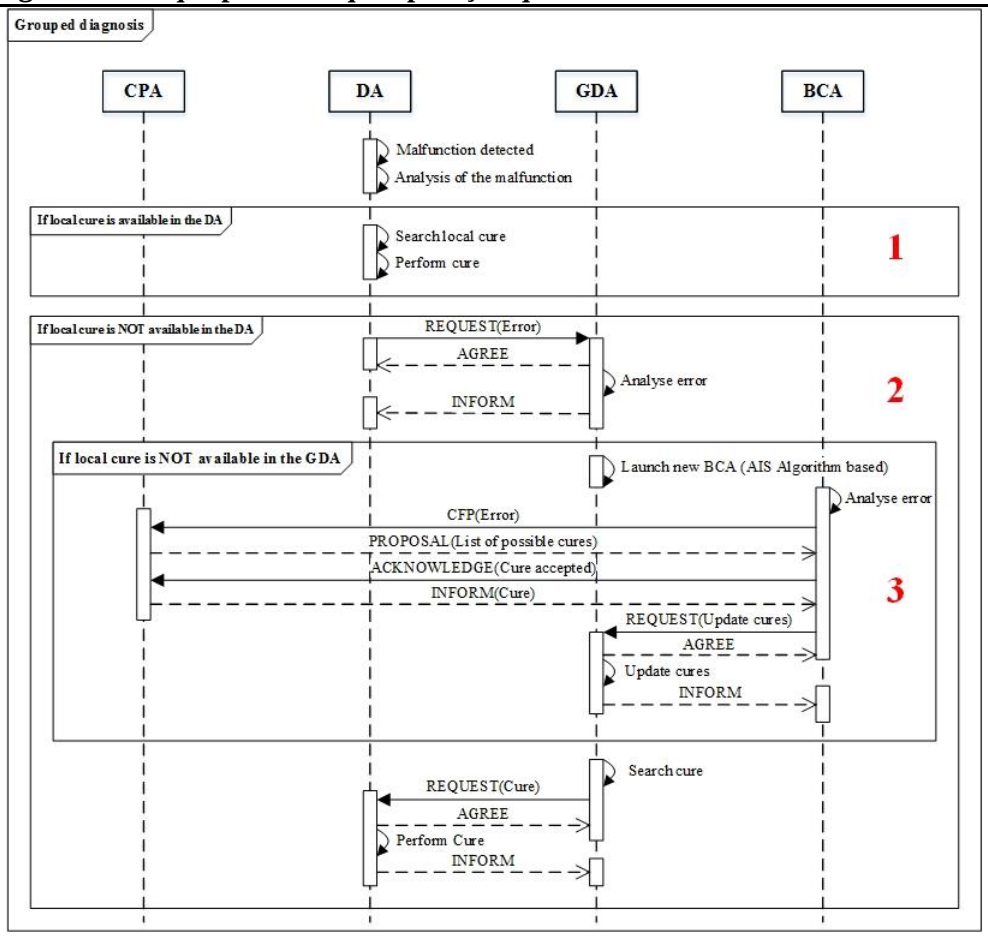


Figure 5.35 – Grouped Diagnosis Sequence Diagram.

If one takes a closer look at the above image, it becomes rather easy to understand how this system works, as the points below are marked in the image:

1. Once a malfunction is detected, the DA that detected it will try to solve it by itself. If this cannot be achieved, a request for a cure for the given error is made to the GDA.
2. The GDA analysis the requested error and, if it knows the cure, simply sends it back to the DA. If it does not know the cure, it will launch several BCAs, according to the AIS algorithm under which it is running.
3. The BCA will analyse the error once again and compare it to its own genome. Shall it be the cure, and the BCA immediately notifies the GDA. If not, the BCA will contact the CPA in order to obtain a database stored cure for the current error.

If, otherwise, the DA is ungrouped, it must, as said before, by itself, solve the error, in a very similar process to that of when a GDA is present. An overview of this mechanism is presented in Figure 5.36.

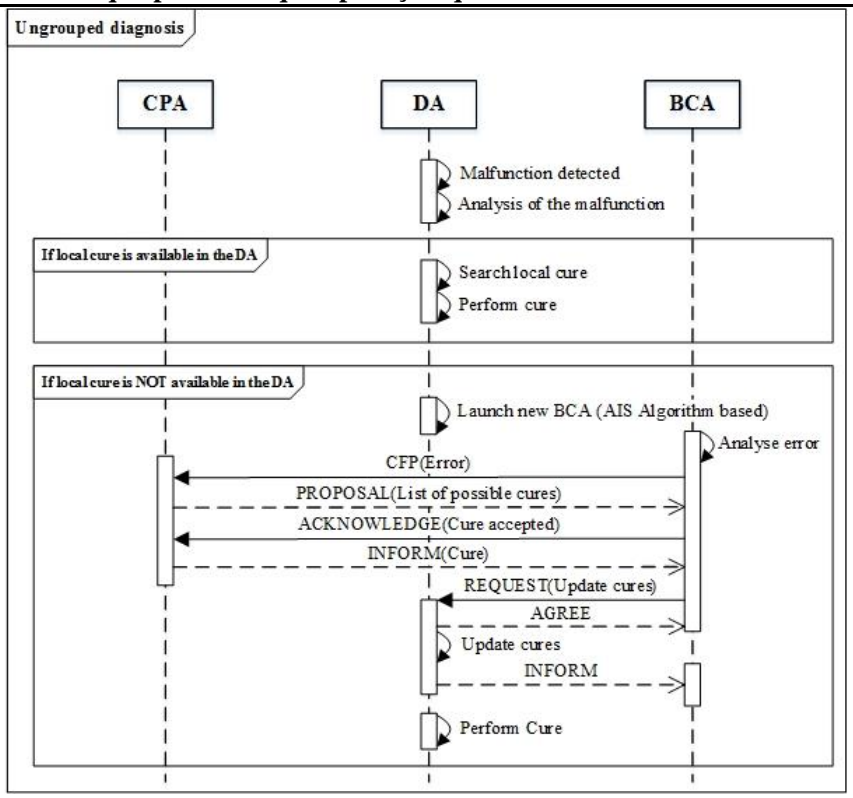


Figure 5.36 – Ungrouped Diagnosis Sequence Diagram.

6

Validation

In this chapter, the validation mechanism will be approached in order to prove the algorithms are valid from an industrial perspective. To do so, an optimal way of testing would be in a real system. However, given the impossibility of such, a simulation environment, built upon the aforementioned architecture, was elaborated and developed in order to approach this chapter's objectives.

Hence, this chapter starts with an introduction to the simulation environments used for the type of tests performed in order to infer the usefulness of the chosen algorithm, in an industrial environment, thus decentralised and capable of a not so great processing mechanism.

Furthermore, two type of tests were executed for this validation to occur, one concerning the total payload the system was able to withstand and the other concerning the overtime capability of the system to improve its error solving capabilities by learning with the previous ones:

1. Payload tests – Type of tests in which the entirety of the system would be tested for errors in order to evaluate its responsiveness to such a scenario.
2. Overtime tests – Type of tests in which the components of the system would receive an error from time to time in an effort to evaluate the system's capability to improve its error recovery mechanism with the previously learned cures.

6.1 Simulation Environments

In this section, a view on the approached simulation environments will be given, with concerns towards the entities in the system, as to why there were no more or less entities in it, and towards its organisation.

Thus, for a proof to be extrapolated as to the benefits of using a decentralised system, there were, in fact, two simulation environments conceived. Despite each of the scenarios has its own differences, there are some points in which they are equal:

1. Six low level entities - After some tests, it was concluded that, for both simulation environments, either de or centralised, a total of six low level entities was appropriate to populate the system with.
2. One high level entity responsible for three low level entities – In order to simulate a scenario as real as possible, it was thought that one Grouped Diagnosis Agent (GDA) for three Diagnosis Agents (DAs) would represent the ideal entity-level organization of this system in an industrial environment.

6.1.1 Centralised Simulation Environment

In this scenario, the six DAs gathered in two GDAs were tested in a single machine in order to understand if it was or not faster than the decentralised one. An architectural perspective on this environment follows in Figure 6.1.

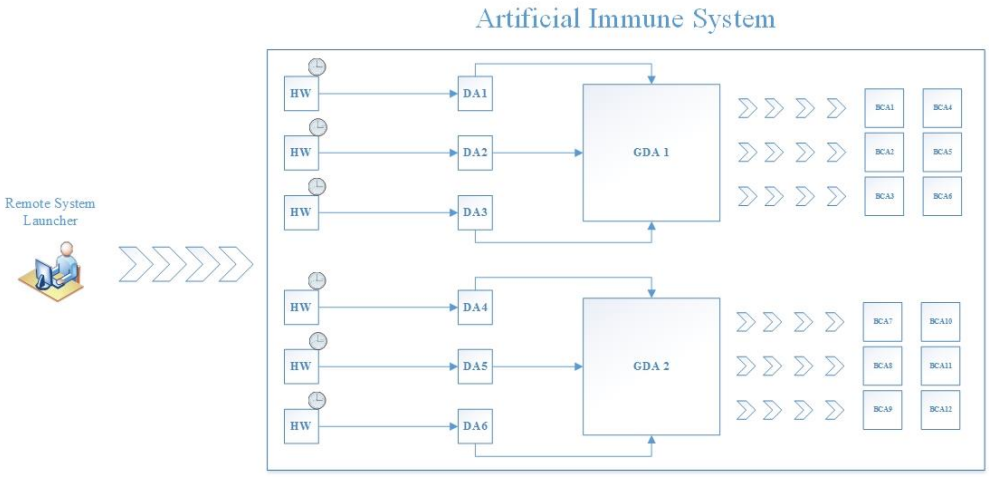


Figure 6.1 – Centralised Simulation Environment.

As it is of easy observation in the above figure, the centralised simulation environment is constituted by six low level entities, known as DAs, gathered three ways into two GDAs, the high level entities of the system.

With this type of environment, the author is set to test, firstly, if it is achievable to perform Artificial Immune System (AIS) diagnosis in a centralised system; secondly, how fast can the system perform in an event of overall shutdown due to unknown errors, and further cure discovery.

6.1.2 Decentralised Simulation Environment

In this scenario, differently from the above, and as the name may suggest, the six low level entities, as the two high level ones, for that matter, are distributed in two separate machines, hence the decentralised capability.

The author hopes that, in this way, the overall performance of the system improves greatly, since the absurd amount of B Cell Agents (BCAs) being launched is now divided between two separated, capable of high processing, machines. The aforementioned simulation environment is hereby depicted in Figure 6.2.

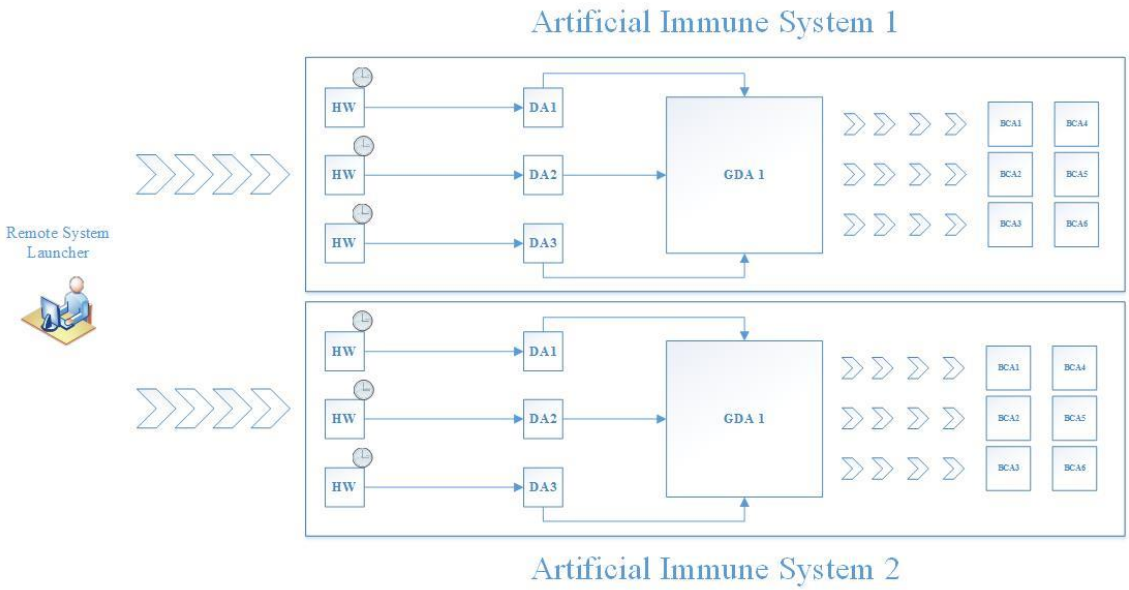


Figure 6.2 – Decentralised Simulation Environment.

In this type of environment, the six low level entities are split three ways between the two separate machines, which means, consequently, the high level will be too. Thus, in each of the machines there will be three DAs grouped under the responsibility of one GDA.

By observing the above figure, it is possible to distinguish two separate AIS, each operating in a separate machine, which hosts three DAs and one GDA, and where the BCAs will be launched in an event of error. Now that all the environments used are acknowledged, it is possible to present some results obtained from the tests performed in them.

6.2 Payload Tests

As it was said before, and as an introduction to this section, these tests were used to determine the overall responsiveness of the system when its entirety was under an error scenario.

To do so, the system was launched and was to wait until an order from a remote system was given in order to launch errors in the entirety of the system. This was to be done in as much of the same time as it was possible, with the current hardware. The purpose of these tests was to infer if the system was able to recover from a full inactive situation and restore its previous state, in the lesser time possible.

6.2.1 Tests' Schematic

These type of tests were performed in both the simulated scenarios since these allowed to determine whether the decentralised environment would fit better in an industrial like scenario, thus leading to a comparison between the two tested environments as a way to establish some terms of comparison between the two of them.

A graphical description of the way this tests were performed follows in Figure 6.3.

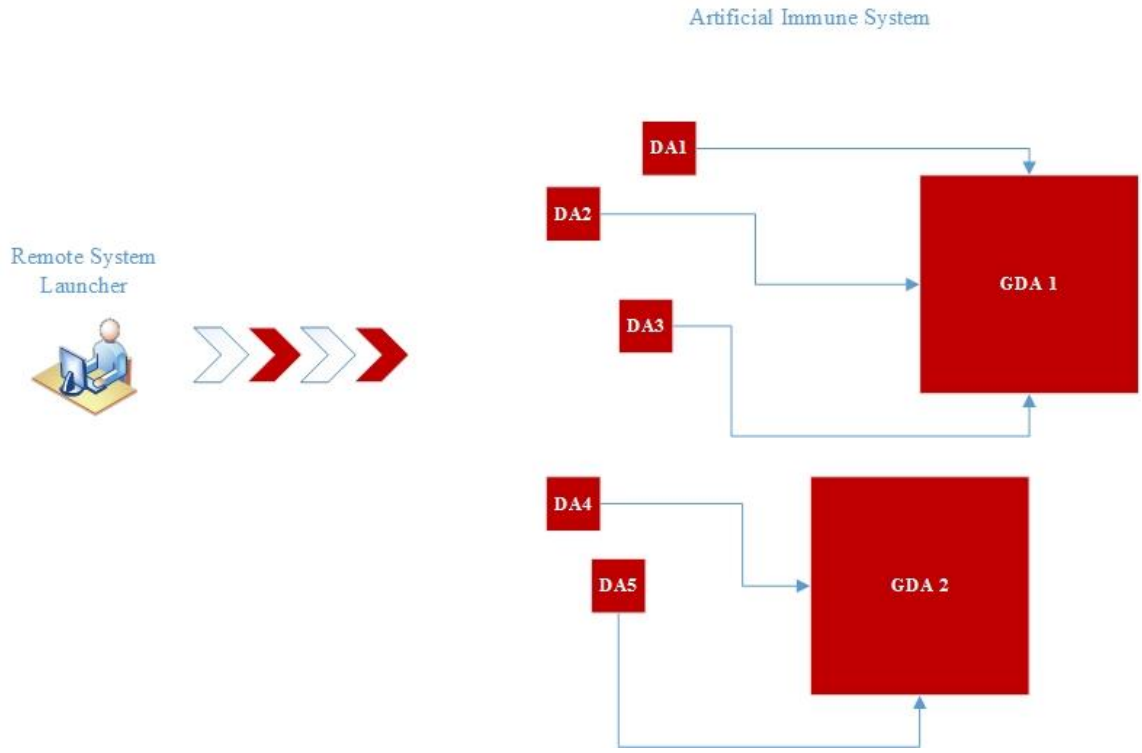


Figure 6.3 – Payload Tests Schematic.

Please, take note that the represented AIS does not represent, in any way, the simulation environment, serving only, in this instance, as a demonstrator.

As it is of easy perception, once the remote system emits the error launching message, the whole system is uploaded with errors, in a, as far as possible, instantaneous attempt to perform an overall shutdown, if you may, of the system.

6.2.2 Results

As to what results concern, three subjects will be approached: whether the increasing word size results, or not, in an increasing cure finding time; in which scenario does the presented architecture outputs better results, centralised or decentralised; lastly, if this architecture has, in fact, any learning capabilities, by analysing the cure finding time of consecutive errors.

For a logical and organised data presentation, the first graphics presented will refer to the obtained data for the centralised system tested, following the ones for the decentralised one.

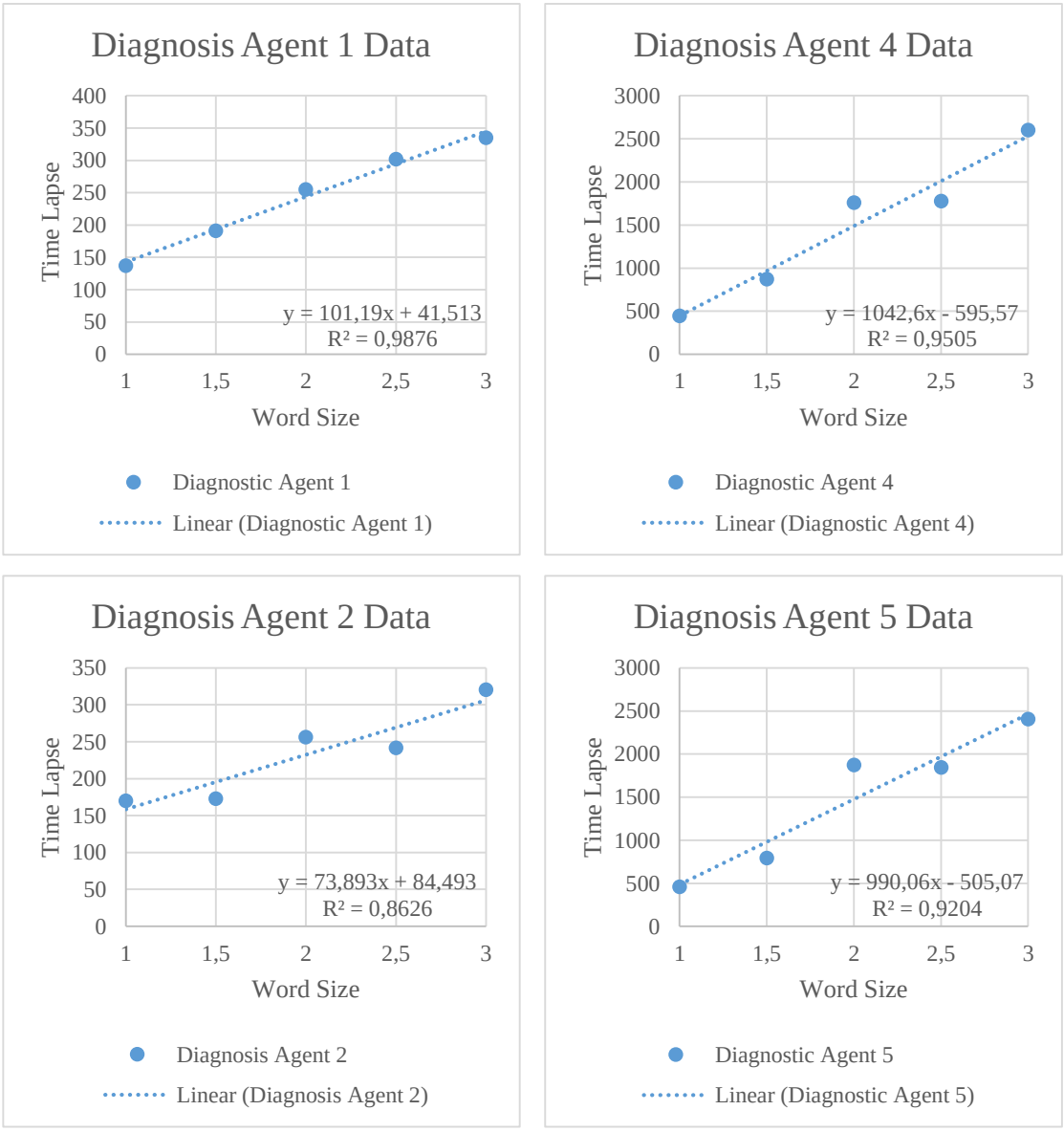
For what this document's objectives concerns, from the analysis of the following graphics one must be able to conclude that the cure finding time grows bigger along with the word size.

Given the huge amount of data collected from the three hundred performed tests, only the average values will be presented in this document. This average values were calculated by summing all the thirty tests for each of the DAs and word sizes and dividing by thirty.

That being said, there will be presented a total of twelve different graphics, divided into two section of six graphics, each of the sections corresponding to the results obtained for each of the tested word sizes in both the centralised environment and the decentralised.

6.2.2.1 Centralised Data

Has it was said before, hereby follows the six obtained graphics for the centralised data. These graphics are presented in Figure 6.4.



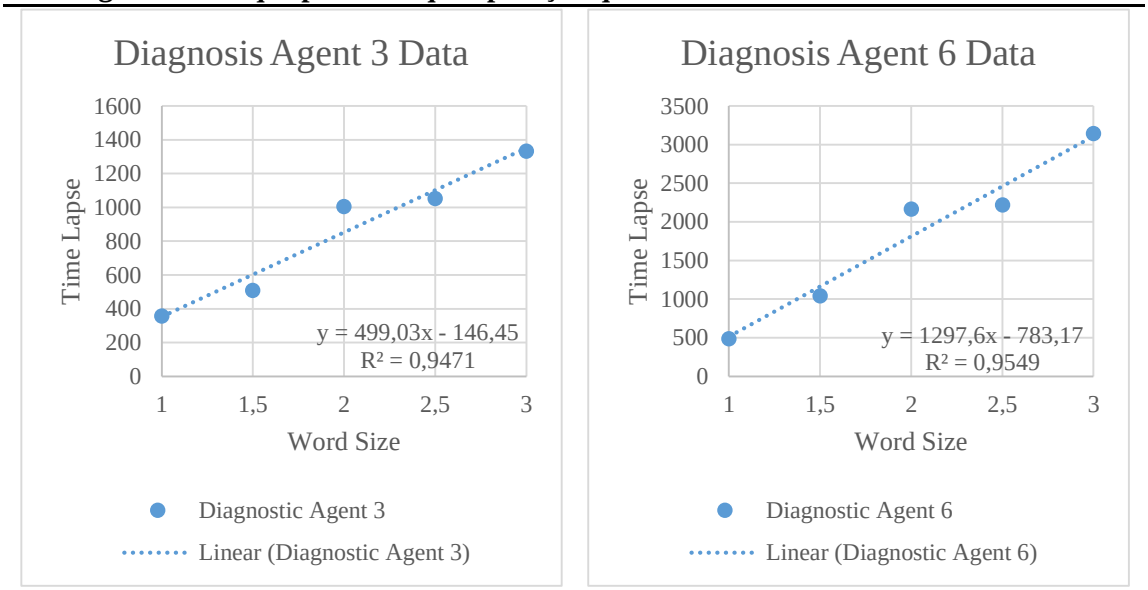


Figure 6.4 – Centralised Diagnosis Agents Data.

By carefully analysing the above graphics, it is possible to note a trend in each of them, common to all. The trend is signalled by a trend line in each of the graphics, in which their equation is depicted as well.

This trend line correlates the scattered points that represent the average values outputted by the DAs for each word size. It offers some knowledge on how the system would behave if one was to test larger word sizes, i.e., if it is possible to conclude that the time lapse for the cure finding mechanism would grow bigger following a linear tendency.

This tendency was somewhat expected as in it takes longer for the system to calculate the affinities between the errors and the possible cures, because the higher the word size, the bigger the genome of the error, hence the time solving increase.

Therefore, and as it may concern a centralised system, it is verifiable that, a larger word size forcibly results in a slower cure finding time, by the reasons stated above.

Now that all the centralised data has been presented and explained, the data for the decentralised environment shall be presented.

6.2.2.2 Decentralised Data

In this section, the six graphics corresponding to the obtained decentralised data will be presented and further discussed. The three graphics on the left refer to one of the independent systems and the ones on the right to the other independent system.

The graphics follow in Figure 6.5.

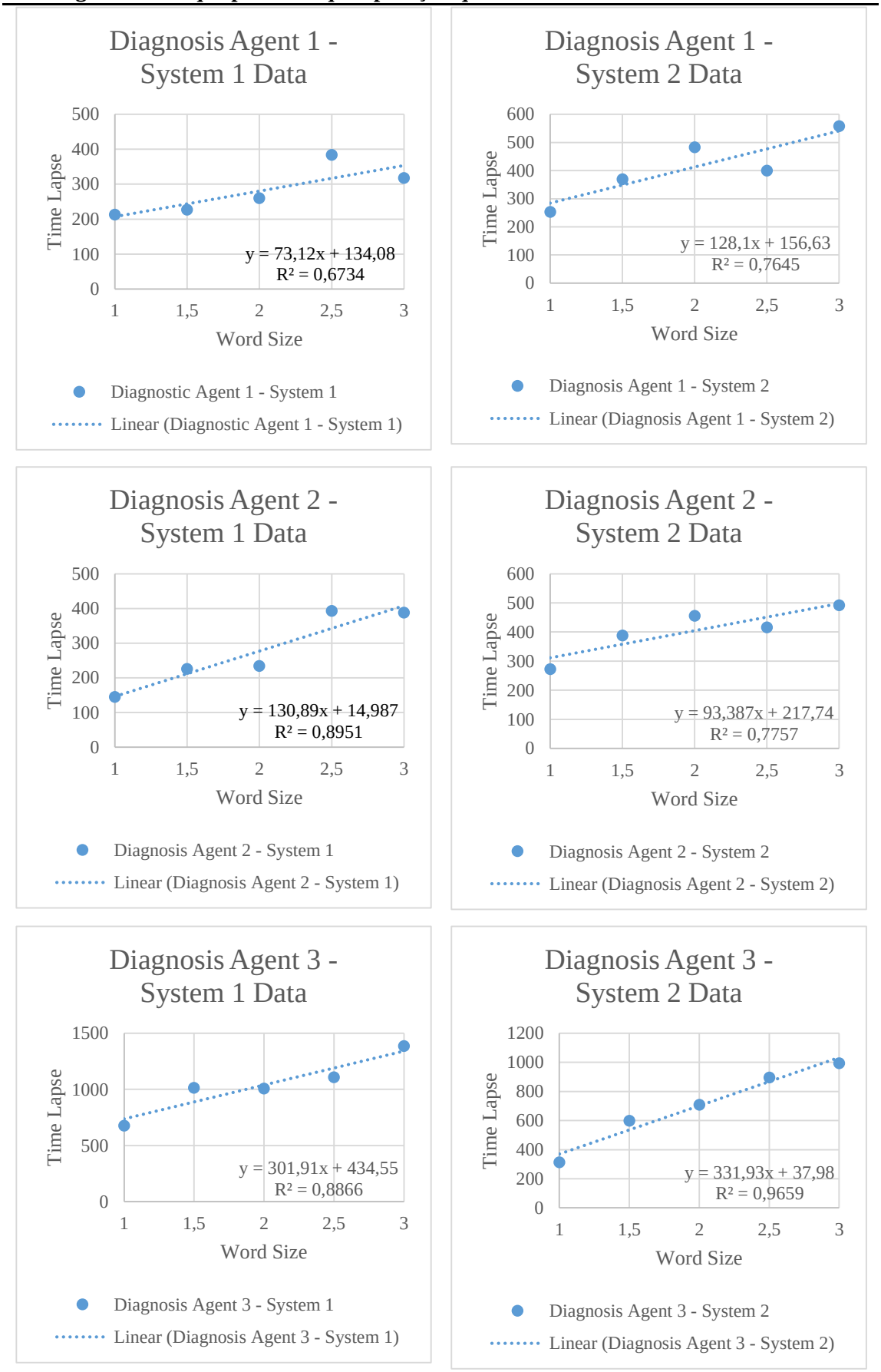


Figure 6.5 – Decentralised Diagnosis Agents Data.

As in the centralised environment, approached above, it is also possible, by observing the above graphics, relative to the decentralised environment, to notice a trend in each of them, common to all; once again, this trend is signalled by a trend line in each of the graphics, along with their equation.

The trend lines, which represent what was already aforementioned, yet again make it possible to conclude that, as the word size gets larger, the slower will be the cure finding time. Once again, this trend was also expected for the reasons stated above.

Moreover, some conclusions can also be drawn from comparing the both systems outputted data. Both follow approximately the same trend despite the system signalled as System 2 outputted values somewhat slower than System 1, which can be explained by the JADE framework and the consequent queued messages in the platform.

That being said, it can be concluded that, for a decentralised system, as in a centralised one, the bigger the word size the more time it will take the system to find a cure.

6.2.2.3 Centralised Data vs Decentralised Data

Now that both the centralised and the decentralised data have been presented, analysed and explained, it is now possible to compare both of them in order to draw a conclusion on which scenario presents better results for the developed architecture, centralised or decentralised.

This comparison can be observed in Figure 6.6, where the average values for each of the word sizes is presented, considering all the thirty tests performed for each of them. The decentralised system is represented in a blue and the centralised system is represented in a red colour.

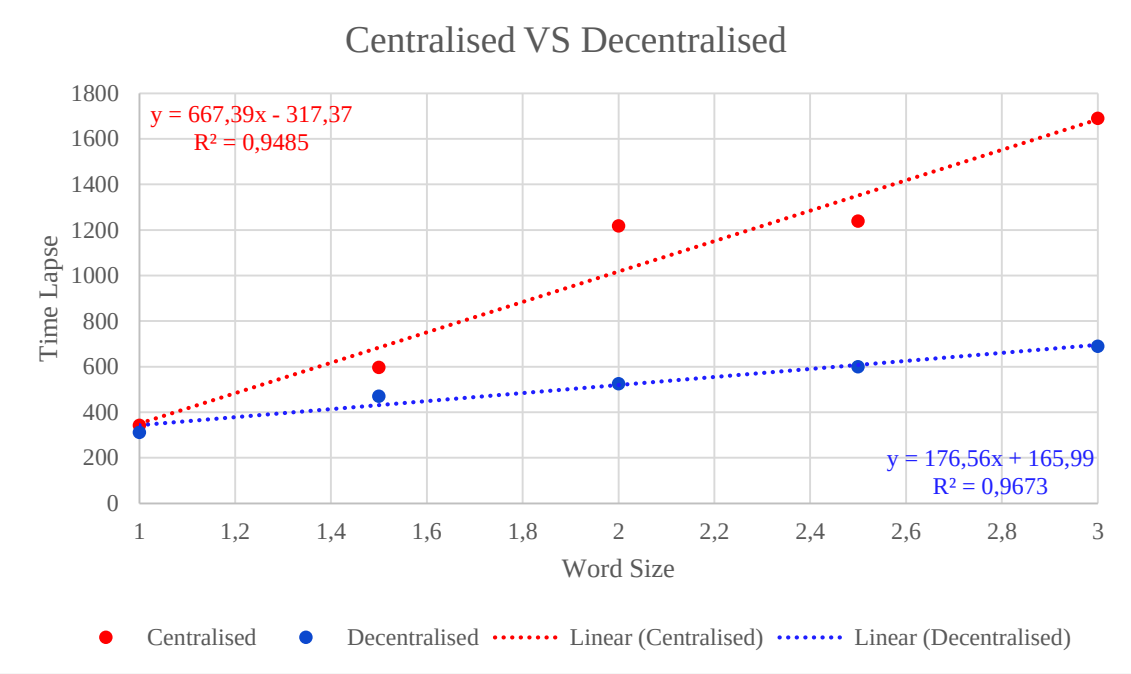


Figure 6.6 – Centralised VS Decentralised Data Comparison.

As it was expected, for the reasons already mentioned, the decentralised environment presents far better results than the centralised one, representing a decrease of, as the word size grows bigger, more than half of the cure finding time.

Another conclusion that can be withdrawn from Figure 6.6 is that, as the word size grows bigger, so does the gap between both environments, as to what time lapse concerns. This was expected and is logical, since the bigger the word size, the harder it will be, theoretically, to find a cure for it (this is sustained by the increasing time lapses in both environments).

That being said, and considering that, in a centralised system, our processing capability is rather diminished, it is only normal that the rate at which the time lapses grow in a centralised system are immensely superior to those of a decentralised one.

Last, but not least, it is possible to conclude that this architecture is far more scalable in a decentralised environment rather than in a centralised one. This happens because the time lapses are smaller in the first scenario, which leads to the ultimate conclusion that, theoretically, it is possible to sustain bigger systems in a decentralised fashion rather than in a centralised one.

Hence, it is hereby proven that a decentralised environment is, indeed, and for all it matters, better for these architecture. Therefore, for the following tests to be presented it will be assumed that the environment being tested is no other but the decentralised one.

6.3 Overtime Tests

Once again, this tests were used with the sole purpose of evaluating the learning capability of the system by evaluating the cure finding times it was outputting in an attempt to evaluate if they were getting smaller, as they should.

In order to achieve this purpose, the system behaves very similarly to the aforementioned one in the beginning, as in it awaits for a communication from a remote machine so it can start its procedures, and this is where it all changes.

6.3.1 Tests' Schematic

Instead of launching errors throughout the entirety of the system, this test launched a hardware class in each of the low level entities in the system. This class is meant to simulate an actual hardware output situation, such as those from the Programmable Logic Controllers (PLCs). It would, occasionally, send an error to the entity responsible for it.

This way, it was possible to simulate an actual, industrial like, situation, where new, unexpected errors may occur every now and then. The above described behaviour is depicted in Figure 6.7

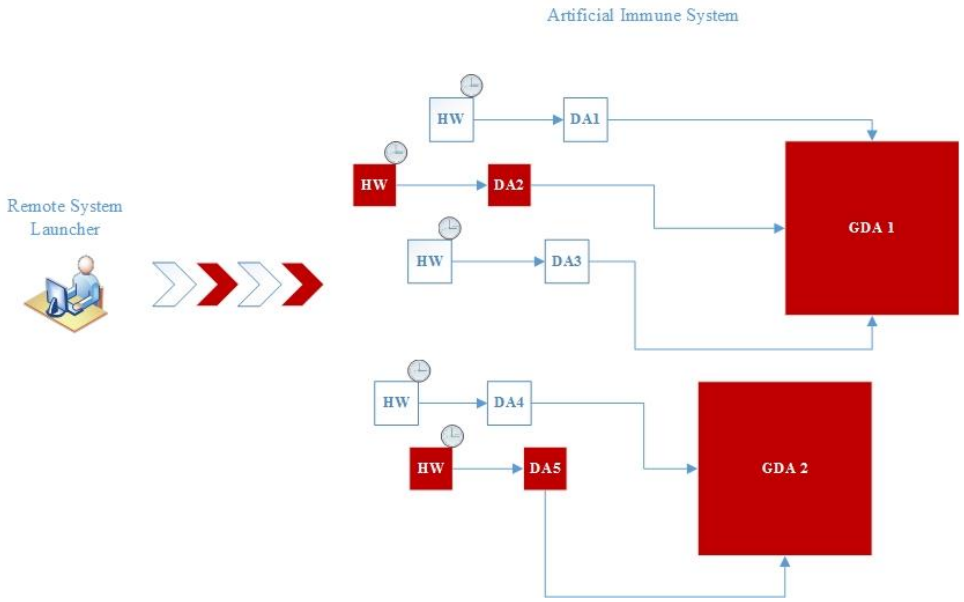


Figure 6.7 – Overtime Tests Schematic.

These type of tests were only performed in the decentralised simulation environment, since this is the type of environment this study aims at and it only makes sense to test for a

durability test in these environment. Moreover, it has already been proven that the decentralised environment outputs better results.

6.3.2 Overtime Efficiency

Now that it has been proven that the decentralised environment is, in fact, better than the centralised one, it is time to test if the cure finding time actually diminishes with the amount of errors that have previously been found and cured.

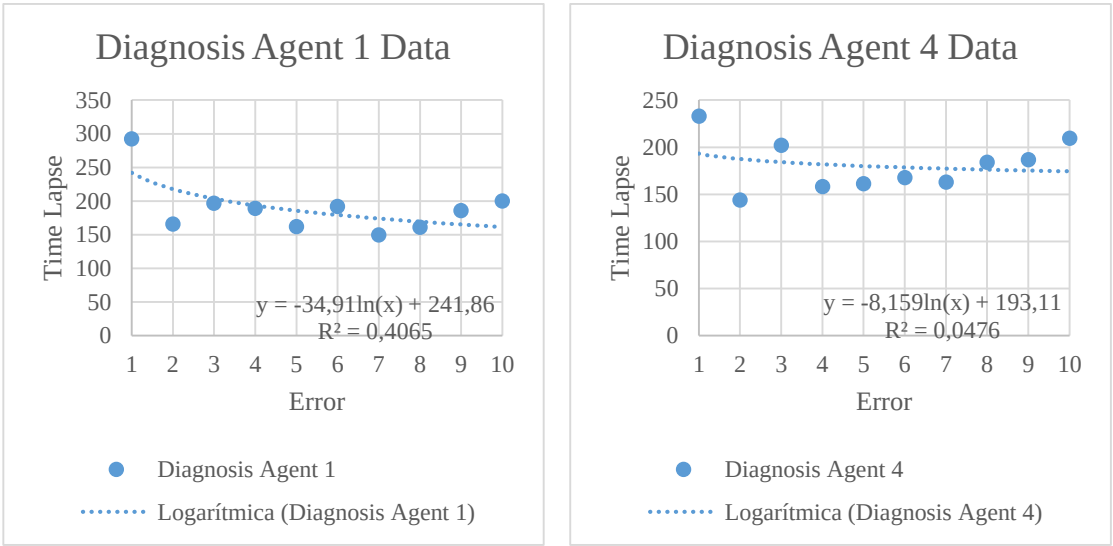
This is expected to occur due to the fact that as a new cure is learnt, it will constitute, along with the previous ones, the initial population for the cure finding algorithm in a new error event. Given that this algorithm is based in the affinity between the known cure's genomes and the error's, it is only natural that, the more cures that constitute the initial population, the bigger the chances will be of finding a quicker cure.

That being said, two different analysis are to be made when it comes to this subject. Firstly, an analysis of each of the DAs behaviour over time for a better understanding of what is actually happening. Lastly, an analysis of the average time the system takes to find the cure for each error in order to understand if it does, in fact, gets faster.

6.3.2.1 Diagnosis Agent Analysis

With this analysis, the author hopes to better understand the behaviour of each of the individual DAs and to shed some light on the whys it is behaving as such.

To do so, six different graphics will follow, each with one DA data presented. The aforementioned graphics are presented in Figure 6.8.



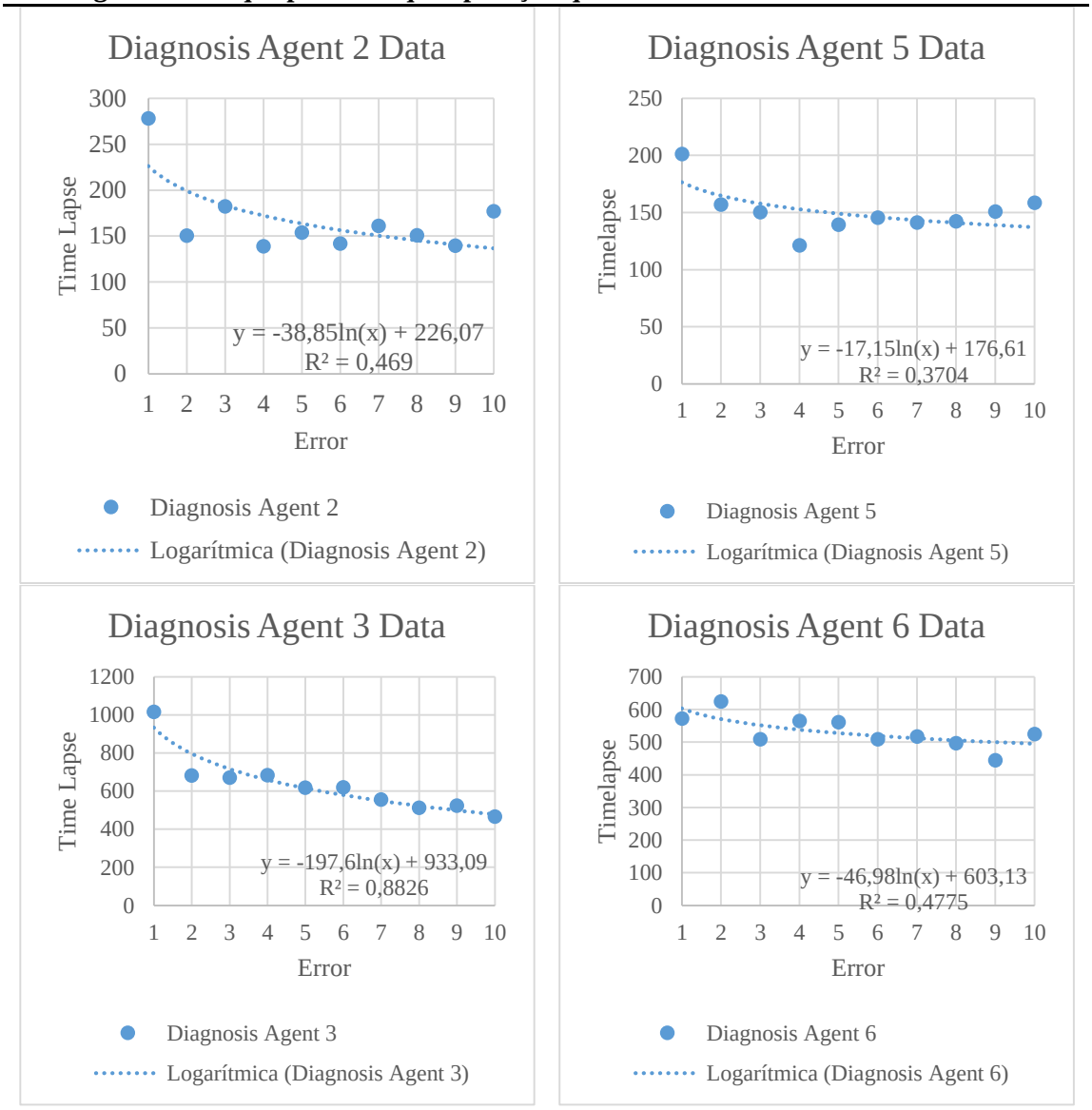


Figure 6.8 – Diagnosis Agents Overtime.

By analysing the above graphics, it is possible to observe the behaviour of six different DAs, divided three ways between the two separate machines. That being said, the first to third DA are hosted in machine one and the fourth to sixth DA in machine two.

It is easy to perceive that the first two DAs, i.e., first and fourth DA, for both systems present time lapses rather inferior when compared to the last one. Despite this may sound odd, it is actually quite natural if we take into account the framework used. It so happens because, no matter the amount of DAs or how big the network is, the last DA in it will always be slower than any of the others. This happens because it is at that time that the system will have more queued messages thus leading to a delay in sending the latest cure to the last DA.

One can also extrapolate, from the above graphics, that the time lapse for each of the error tend to diminish as more and more errors are found, despite this behaviour will be more perceivable in the upcoming topic.

6.3.2.2 Error Average Overtime

Now that the DAs behaviour is explained, it is time to finally evaluate whether the system is capable of diminishing the cure finding time as new errors are cured. This may lead to the ultimate conclusion that the system is, in fact, capable of learning and of improving with those same learning mechanics.

To do so, an analysis on the average time each error took to be resolved is in order. Such an analysis will be based on the underlying Figure 6.9.

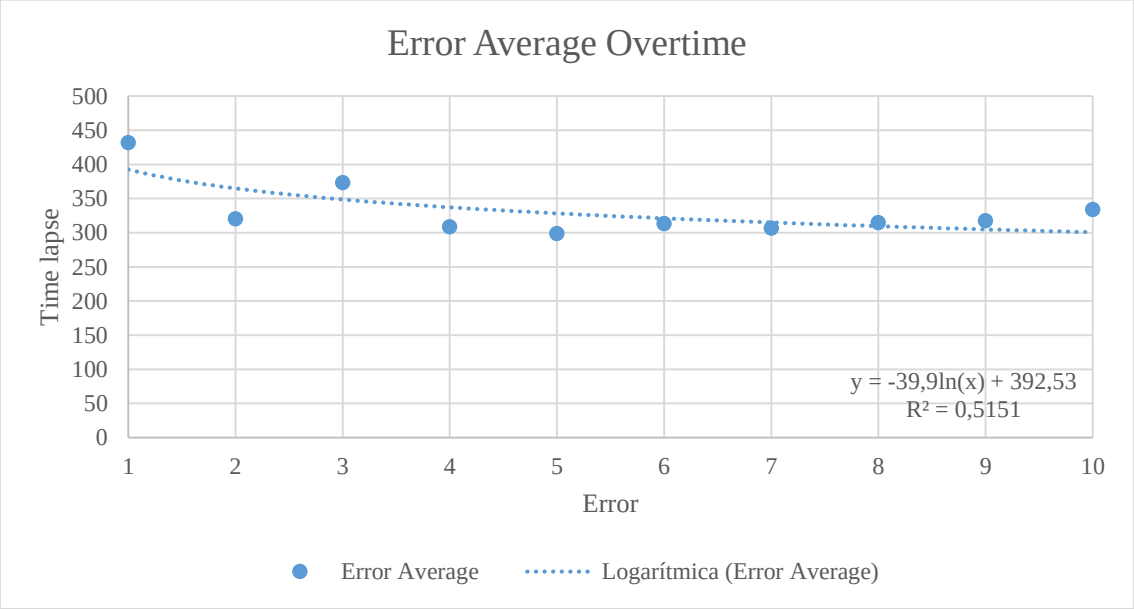


Figure 6.9 – Error Average Overtime.

The above figure depicts a graphic that represents the average time lapse obtained from all the DAs for each consecutive error. If one is to take a closer look, it becomes easily observable that the time lapses do slightly diminish as new cures enter the system once the error is resolved.

The sole exception to this trend is the last error, where the average time goes up. It so happens because, once again, the last DA means the return of the system to its prior state, where no BCAs are present in the system, thus meaning a lot of queued messages to be processed which ends up delaying the cure finding time of the last DA.

Despite the last value does not follow the overall tendency, this faulty behaviour, if you may, may be caused due to the accumulated messages in the JADE scheduler or due to the JAVA garbage collector.

That being said, and with all the presented data, it is fair to conclude that this system possesses a learning mechanism, corroborated by the above data, and which results in a faster cure finding from the system, as long as there has been previous errors detected.

7

Conclusions and Further Work

In this last chapter, some conclusions towards the developed work will be drawn, along with some coverage of the topics that should be approached in a near future in order to continue and further develop the work hereby presented in this document.

7.1 Conclusions

The main purpose of this document's presented research and development was to offer an alternative to the current diagnosis paradigms through the means of the Artificial Immune Systems (AIS). This was quite a challenge since this is still a much undeveloped topic and there was not that much of research material to work with.

That being said, there are three principal conclusions that can be drawn from the work presented in this document.

Firstly, the work developed around the AIS algorithms allowed for some sort of ranking between them to be established which, as much as this document's author concerns, was yet to be drafted. With this ranking, it was possible to conclude on which of the algorithms, Negative Selection (NS), Clonal Selection (CS) or Network Model (NM), would fit better in an industrial environment, with all the processing (dis)abilities to it associated. The NM proved to be the most efficient of the algorithms for the tested word sizes, accounting for an improvement of around

forty pp in the biggest word sizes. However, the AIS is based on the launching of several B cell like entities which are, if you may, a "performance killer".

Secondly, this document also shows that AIS is a viable, promising alternative to the current diagnosis paradigms. The developed architecture does not aim to be the best, but it presents itself as a valid alternative to the, if you may, classic diagnosis architectures.

It proves that, with no more than four distinct entities, it is possible to somewhat mimic the Human Immune System. Once an error, which is the representative of an infection, is detected, the system triggers a cure finding mechanism that launches countless B cells alike entities (much like our own Immune System does). These cells' genome will be compared to the error's, (being both represented by a binary string) in as much of a resemblance as possible with the Human Immune System.

Not only that, it has proven to be a somewhat rather efficient methodology when it comes to error recovery since it does not take that much to actually find a cure, as the tests have shown. The average is around the five hundred milliseconds, which represents five times the average PLC cycle.

Moreover, it may serve as an argument in favour of those who sustain that the Human System is possible to be mimicked and represented by mathematical equations, making it possible to adapt natural behaviours to unnatural proceedings, as it is the industrial mechanism.

Lastly, but not least, the work hereby developed also constitutes proof that, in yet another paradigm, a distributed, decentralised system presents better results than a centralised one. This comes to show that, in a cloud like system, it is possible to withstand a ridiculous amount of processing in order to achieve better and/or faster results for the case of study.

With only two separate machines, which means the tested network was divided into two separate, independent machines, it was possible to halve the time it took for the system to find a cure. These comes to show that some further work should be made in this area, in order to further sustain these claims and, perhaps, even enhance them.

To wrap things up, the AIS paradigm shows great potential, given that the basis in which it was built upon have proven its usefulness – The Human Immune System; and it should definitely go under a more thoroughly investigation by the manufactory paradigms investigators.

For the time this work has been under development, two conference articles have been written. The first one was presented in the Flexible Automation and Intelligent Manufacturing

(FAIM) conference (Andre Dionisio Rocha, Monteiro, & Barata, 2015) and the second one in the IECON conference (Andre Dionisio Rocha, Monteiro, Parreira, & Barata, 2015).

Also under this work procedures, an article is being developed for posterior submission in an international scientific journal.

7.2 Further Work

From a further work perspective, there are some points that particularly need to be pointed out, since they are the ones that should receive the larger percentage of focus under this perspective. These points shall be mentioned by their order of importance, according to this document's author point of view.

Firstly, it is this document author's belief that the B Cell Agent (BCA) launching mechanism should be revised and improved in order to allow for better performance results. As it is, and considering the existing hardware, especially in an industrial like environment, it becomes impossible to have a reasonable sized network in only one machine given that the BCAs outburst on an error event is so great it terminates the Java Virtual Machine. This leads to the conclusion that better results are dependent on hardware improvements.

Secondly, a revision on whether the used framework, JADE, is the most assertive, hence the most adequate for this type of the system should be made. This is justifiable due to the huge amount of messages this framework uses according to its FIPA compliant Protocols. This protocols largely increase the processing capability killing power of this system. Therefore, this document's author sustains a new framework for these type of systems should be searched/developed since the author believes this would be a turning point for this and other alike systems that largely depend on communication.

Thirdly, the proposed architecture should be revised. There are some points in which it could be improved, such as adding another entity to regulate each of the algorithm's instances launched upon error events. Another point under which it could be revised is that of the Cure Provider Agent (CPA) cloud being accessible from both the Diagnostic Agents (DAs) and from the Grouped Diagnosis Agents (GDAs). In the author's point of view, it would improve the system's performance and sustainability in a distributed and decentralised environment.

That being said, this document's author still sustains that this paradigm should be further developed, maybe under another framework in a continuous, ever-lasting search for better results on the overall performance of the system, given that it presents itself, with the work hereby

developed, as a strong and promising alternative to the current diagnosis paradigms in the industrial environment.

Bibliography

Aminian, F., Aminian, M., & Collins, J. . H. W. (2002). Analog fault diagnosis of actual circuits using neural networks. *IEEE Transactions on Instrumentation and Measurement*, *51*(3), 544–550. http://doi.org/10.1109/TIM.2002.1017726

Ayara, M., Timmis, J., de Lemos, R., de Castro, L. N., & Duncan, R. (2002). Negative selection: How to generate detectors. In *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)* (Vol. 1, pp. 89–98). Canterbury, UK:[sn]. Retrieved from http://neuro.bstu.by/our/Anomaly-Detection/ayara-etal.pdf

Balaji, P. G., & Srinivasan, D. (2010). An Introduction to Multi-Agent Systems. In D. Srinivasan & L. C. Jain (Eds.), *Innovations in Multi-Agent Systems and Applications - 1* (pp. 1–27). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-14435-6_1

Barata, J., & Camarinha-Matos, L. M. (2003). Coalitions of manufacturing components for shop floor agility - the CoBASA architecture. *International Journal of Networking and Virtual Organisations*, *2*(1), 50–77. http://doi.org/10.1504/IJNVO.2003.003518

Barata, J., Camarinha-Matos, L., & Onori, M. (2005). A multiagent based control approach for evolvable assembly systems. In *2005 3rd IEEE International Conference on Industrial Informatics, 2005. INDIN '05* (pp. 478–483). http://doi.org/10.1109/INDIN.2005.1560423

Barata, J., Ribeiro, L., & Colombo, A. (2007). Diagnosis using Service Oriented Architectures (SOA). In *2007 5th IEEE International Conference on Industrial Informatics* (Vol. 2, pp. 1203–1208). http://doi.org/10.1109/INDIN.2007.4384902

Barata, J., Ribeiro, L., & Onori, M. (2007). Diagnosis on Evolvable Production Systems. In *IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007* (pp. 3221–3226). http://doi.org/10.1109/ISIE.2007.4375131

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM* (Vol. 99, p. 33). London. Retrieved from http://www.dia.fi.upm.es/~phernan/AgentesInteligentes/referencias/bellifemine99.pdf

Bellifemine, F., Poggi, A., & Rimassa, G. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *Software-Practice and Experience*, *31*(2), 103–128.

Bi, Z. M., Lang, S. Y. T., Shen, W., & Wang, L. (2008). Reconfigurable manufacturing systems: the state of the art. *International Journal of Production Research*, *46*(4), 967–992. http://doi.org/10.1080/00207540600905646

Buzacott, J. A., & Shanthikumar, J. G. (1993). *Stochastic Models of Manufacturing Systems*. Prentice Hall.

Chen, J., & Patton, R. J. (2012). *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Springer Science & Business Media.

Da Silveira, G., Borenstein, D., & Fogliatto, F. S. (2001). Mass customization: Literature review and research directions. *International Journal of Production Economics*, *72*(1), 1–13. http://doi.org/10.1016/S0925-5273(00)00079-7

de Castro, L. N., & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, *6*(3), 239–251. http://doi.org/10.1109/TEVC.2002.1011539

Desforges, X., & Archimède, B. (2006). Multi-agent framework based on smart sensors/actuators for machine tools control and monitoring. *Engineering Applications of Artificial Intelligence*, *19*(6), 641–655. http://doi.org/10.1016/j.engappai.2006.03.006

Elkins, D. A., Huang, N., & Alden, J. M. (2004). Agile manufacturing systems in the automotive industry. *International Journal of Production Economics*, *91*(3), 201–214. http://doi.org/10.1016/j.ijpe.2003.07.006

Ellis, B. A. (2008). Condition based maintenance. *The Jethro Project*, 1–5.

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

Farmer, J. D., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, *22*(1–3), 187–204. http://doi.org/10.1016/0167-2789(86)90240-X

Feldmann, K., & Göhringer, J. (2001). Internet based Diagnosis of Assembly Systems. *CIRP Annals - Manufacturing Technology*, *50*(1), 5–8. http://doi.org/10.1016/S0007-8506(07)62058-7

Forrest, S., Perelson, A. S., Allen, L., & Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *2012 IEEE Symposium on Security and Privacy* (pp. 202–202). IEEE

Computer Society. Retrieved from http://www.computer.org/csdl/proceedings/sp/1994/5675/00/56750202.pdf

Frei, R., Barata, J., & Onori, M. (2007). Evolvable Production Systems Context and Implications. In *IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007* (pp. 3233–3238). http://doi.org/10.1109/ISIE.2007.4375132

Frei, R., Di Marzo Serugendo, G., & Barata, J. (2008). Designing Self-Organization for Evolvable Assembly Systems. In *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 2008. SASO '08* (pp. 97–106). http://doi.org/10.1109/SASO.2008.20

Frei, R., Ferreira, B., Di Marzo Serugendo, G., & Barata, J. (2009). An architecture for self-managing evolvable assembly systems. In *IEEE International Conference on Systems, Man and Cybernetics, 2009. SMC 2009* (pp. 2707–2712). http://doi.org/10.1109/ICSMC.2009.5346137

Frei, R., Ribeiro, L., Barata, J., & Semere, D. (2007). Evolvable Assembly Systems: Towards User Friendly Manufacturing. In *IEEE International Symposium on Assembly and Manufacturing, 2007. ISAM '07* (pp. 288–293). http://doi.org/10.1109/ISAM.2007.4288487

Fries, T. P. (2007). Multi-Agent Fault Diagnosis in Manufacturing Systems Using Soft Computing. In *International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2007. KIMAS 2007* (pp. 168–173). http://doi.org/10.1109/KIMAS.2007.369804

Garrett, S. M. (2005). How do we evaluate artificial immune systems? *Evolutionary Computation*, *13*(2), 145–177.

Grall, A., Bérenguer, C., & Dieulle, L. (2002). A condition-based maintenance policy for stochastically deteriorating systems. *Reliability Engineering & System Safety*, *76*(2), 167–180. http://doi.org/10.1016/S0951-8320(01)00148-X

Grall, A., Dieulle, L., Berenguer, C., & Roussignol, M. (2002). Continuous-time predictive-maintenance scheduling for a deteriorating system. *IEEE Transactions on Reliability*, *51*(2), 141–150. http://doi.org/10.1109/TR.2002.1011518

Groover, M. P. (2007). *Automation, Production Systems, and Computer-Integrated Manufacturing* (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Gunasekaran, A. (2001). *Agile Manufacturing: The 21st Century Competitive Strategy: The 21st Century Competitive Strategy*. Elsevier.

Hua, X.-L., Gondal, I., & Yaqub, F. (2013). Mobile agent based artificial immune system for machine condition monitoring. In *2013 8th IEEE Conference on Industrial Electronics and Applications (ICIEA)* (pp. 108–113). http://doi.org/10.1109/ICIEA.2013.6566349

Hu, B., & Qin, S. (2012). Prognostic methodology for health management of electrical equipments of propulsion system in a type of vessel based on artificial immune algorithm. In *2012 IEEE Conference on Prognostics and System Health Management (PHM)* (pp. 1–8). http://doi.org/10.1109/PHM.2012.6228812

Isermann, R. (2006). *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer Science & Business Media.

Jardine, A. K. S., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, *20*(7), 1483–1510. http://doi.org/10.1016/j.ymssp.2005.09.012

Kim, J., & Bentley, P. J. (2001). Towards an artificial immune system for network intrusion detection: an investigation of clonal selection with a negative selection operator. In

Proceedings of the 2001 Congress on Evolutionary Computation, 2001 (Vol. 2, pp. 1244–1252 vol. 2). http://doi.org/10.1109/CEC.2001.934333

Kordic, V. (2006). *Manufacturing the future: concepts, technologies & visions*. Mammendorf: Pro-Literatur-Verl.

Koren, Y. (1983). Computer Control of Manufacturing Systems / Y. Koren.

Koren, Y., & Shpitalni, M. (2010). Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems*, *29*(4), 130–141. http://doi.org/10.1016/j.jmsy.2011.01.001

Landers, R. G., Min, B.-K., & Koren, Y. (2001). Reconfigurable Machine Tools. *CIRP Annals - Manufacturing Technology*, *50*(1), 269–274. http://doi.org/10.1016/S0007-8506(07)62120-9

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

Leitão, P., & Restivo, F. (2008). A holonic approach to dynamic manufacturing scheduling. *Robotics and Computer-Integrated Manufacturing*, *24*(5), 625–634. http://doi.org/10.1016/j.rcim.2007.09.005

Lohse, N., Hirani, H., Ratchev, S., & Turitto, M. (2005). An ontology for the definition and validation of assembly processes for evolvable assembly systems. In *The 6th IEEE International Symposium on Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005)* (pp. 242–247). http://doi.org/10.1109/ISATP.2005.1511480

Lohse, N., Ratchev, S., & Barata, J. (2006). Evolvable Assembly Systems - On the role of design frameworks and supporting ontologies. In *2006 IEEE International Symposium on Industrial Electronics* (Vol. 4, pp. 3375–3380). http://doi.org/10.1109/ISIE.2006.296008

Lopes, G., & Pereira, L. M. (2006). Prospective logic programming with ACORDA. *Procs. of the FLoC, 6*. Retrieved from http://ceur-ws.org/Vol-192/paper10.pdf

Mauro, O. (2009). Evolvable production systems : Mechatronic production equipment with process-based distributed control. In H. Hideki (Ed.), (pp. 80–85). http://doi.org/10.3182/20090909-4-JP-2010.00016

McArthur, S. D. J., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., & Funabashi, T. (2007a). Multi-Agent Systems for Power Engineering Applications #x2014;Part I: Concepts, Approaches, and Technical Challenges. *IEEE Transactions on Power Systems*, *22*(4), 1743–1752. http://doi.org/10.1109/TPWRS.2007.908471

McArthur, S. D. J., Davidson, E. M., Catterson, V. M., Dimeas, A. L., Hatziargyriou, N. D., Ponci, F., & Funabashi, T. (2007b). Multi-Agent Systems for Power Engineering Applications #x2014;Part II: Technologies, Standards, and Tools for Building Multi-agent Systems. *IEEE Transactions on Power Systems*, *22*(4), 1753–1759. http://doi.org/10.1109/TPWRS.2007.908472

Mehrabi, M. G., Ulsoy, A. G., & Koren, Y. (2000). Reconfigurable manufacturing systems and their enabling technologies. *International Journal of Manufacturing Technology and Management*, *1*(1), 114–131. http://doi.org/10.1504/IJMTM.2000.001330

Mehrabi, M. G., Ulsoy, A. G., Koren, Y., & Heytler, P. (2002). Trends and perspectives in flexible and reconfigurable manufacturing systems. *Journal of Intelligent Manufacturing*, *13*(2), 135–146. http://doi.org/10.1023/A:1014536330551

Mendes, M. J. G. C., Santos, B. M. S., & Costa, J. S. da. (2009). Multi-agent Platform and Toolbox for Fault Tolerant Networked Control Systems. *Journal of Computers*, *4*(4), 303–310. http://doi.org/10.4304/jcp.4.4.303-310

Mobley, R. K. (2002). *An Introduction to Predictive Maintenance*. Butterworth-Heinemann.

Mohammadi, M., Akbari, A., Raahemi, B., Nassersharif, B., & Asgharian, H. (2014). A fast anomaly detection system using probabilistic artificial immune algorithm capable of learning new attacks. *Evolutionary Intelligence*, *6*(3), 135–156. http://doi.org/10.1007/s12065-013-0101-3

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

Neves, P., & Barata, J. (2009). Evolvable production systems. In *IEEE International Symposium on Assembly and Manufacturing, 2009. ISAM 2009* (pp. 189–195). http://doi.org/10.1109/ISAM.2009.5376907

Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*, *95*(1), 215–233. http://doi.org/10.1109/JPROC.2006.887293

Onori, M., Alsterman, H., & Barata, J. (2005). An architecture development approach for evolvable assembly systems. In *The 6th IEEE International Symposium on Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005)* (pp. 19–24). http://doi.org/10.1109/ISATP.2005.1511444

Onori, M., Barata, J., & Frei, R. (2006). Evolvable Assembly Systems Basic Principles. In *Information Technology For Balanced Manufacturing Systems* (pp. 317–328). Springer US. Retrieved from http://link.springer.com/chapter/10.1007/978-0-387-36594-7_34

Parker, D. C., Manson, S. M., Janssen, M. A., Hoffmann, M. J., & Deadman, P. (2003). Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change: A Review. *Annals of the Association of American Geographers*, *93*(2), 314–337. http://doi.org/10.1111/1467-8306.9302004

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. http://doi.org/10.1016/j.compind.2011.03.001

Ribeiro, L., & Barata, J. (2012). IMS 10—Validation of a co-evolving diagnostic algorithm for evolvable production systems. *Engineering Applications of Artificial Intelligence*, *25*(6), 1142–1160. http://doi.org/10.1016/j.engappai.2012.02.008

Ribeiro, L., Barata, J., Onori, M., Hanisch, C., Hoos, J., & Rosa, R. (2011). Self-organization in automation - the IDEAS pre-demonstrator. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society* (pp. 2752–2757). http://doi.org/10.1109/IECON.2011.6119747

Rocha, A. D., Barata, D., Orio, G. D., Santos, T., & Barata, J. (2015). PRIME as a Generic Agent Based Framework to Support Pluggability and Reconfigurability Using Different Technologies. In L. M. Camarinha-Matos, T. A. Baldissera, G. D. Orio, & F. Marques (Eds.), *Technological Innovation for Cloud-Based Engineering Systems* (pp. 101–110). Springer International Publishing. Retrieved from http://link.springer.com/chapter/10.1007/978-3-319-16766-4_11

Rocha, A. D., Monteiro, P. L., & Barata, J. (2015). An Artificial Immune Systems Based Architecture to Support Diagnoses in Evolvable Production Systems Using Genetic Algorithms as an Evolution Enabler. Presented at the Flexible Automation and Intelligent Manufacturing.

Rocha, A. D., Monteiro, P. L., Parreira, M., & Barata, J. (2015). Artificial Immune Systems Algorithms as a Solution to Perform Diagnosis on Distributed Manufacturing Systems: A Performance Test. Presented at the IECON.

Rütten, A., Vuillemin, A., Ooijendijk, W., Schena, F., Sjöström, M., Stahl, T., … Ziemainz, H. (2003). Physical activity monitoring in Europe. The European Physical Activity Surveillance System (EUPASS) approach and indicator testing. *Public Health Nutrition*, *6*(04), 377–384. http://doi.org/10.1079/PHN2002449

Sanchez, L. M., & Nagi, R. (2001). A review of agile manufacturing systems. *International Journal of Production Research*, *39*(16), 3561–3600. http://doi.org/10.1080/00207540110068790

Semere, D., Barata, J., & Onori, M. (2007). Evolvable Assembly Systems: Developments and Advances. In *IEEE International Symposium on Assembly and Manufacturing, 2007. ISAM '07* (pp. 282–287). http://doi.org/10.1109/ISAM.2007.4288486

Shen, W. (2002). Distributed manufacturing scheduling using intelligent agents. *IEEE Intelligent Systems*, *17*(1), 88–94. http://doi.org/10.1109/5254.988492

Shen, W., Lang, S. Y. T., & Wang, L. (2005). iShopFloor: an Internet-enabled agent-based intelligent shop floor. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *35*(3), 371–381. http://doi.org/10.1109/TSMCC.2004.843224

Shen, W., & Norrie, D. H. (1999). Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems*, *1*(2), 129–156. http://doi.org/10.1007/BF03325096

Sirca, A. D. (2008). *Reconfigurable Manufacturing Systems*. Retrieved from http://www.imtuoradea.ro/conf_stud/ss_2008_files/Oradea_Sirca_Anca_Reconfigurable_Manufacturing_Systems.pdf

Smith, R. E., Timmis, J., Stepney, S., & Neal, M. (2005). Conceptual frameworks for artificial immune systems. Retrieved from http://cadair.aber.ac.uk/dspace/handle/2160/1739

Sorsa, T., & Koivo, H. N. (1993). Application of artificial neural networks in process fault diagnosis. *Automatica*, *29*(4), 843–849. http://doi.org/10.1016/0005-1098(93)90090-G

Timmis, J., Andrews, P., & Hart, E. (2010). On artificial immune systems and. *Swarm Intelligence*, *4*(4), 247–273. http://doi.org/10.1007/s11721-010-0045-5

Timmis, J., Andrews, P., Owens, N., & Clark, E. (2008). An interdisciplinary perspective on artificial immune systems. *Evolutionary Intelligence*, *1*(1), 5–26. http://doi.org/10.1007/s12065-007-0004-2

Timmis, J., Neal, M., & Hunt, J. (2000). An artificial immune system for data analysis. *Biosystems*, *55*(1–3), 143–150. http://doi.org/10.1016/S0303-2647(99)00092-1

Valckenaers, P., & Van Brussel, H. (2005). Holonic Manufacturing Execution Systems. *CIRP Annals - Manufacturing Technology*, *54*(1), 427–432. http://doi.org/10.1016/S0007-8506(07)60137-1

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., & Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, *37*(3), 255–274. http://doi.org/10.1016/S0166-3615(98)00102-X

Van Dyke Parunak, H., Baker, A. D., & Clark, S. J. (2001). The AARIA agent architecture: From manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering*, *8*(1), 45–58.

White, J. A., & Garrett, S. M. (2003). Improved Pattern Recognition with Artificial Clonal Selection? In J. Timmis, P. J. Bentley, & E. Hart (Eds.), *Artificial Immune Systems* (pp. 181–193). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-45192-1_18

Williams, B. C., & Nayak, P. P. (1996). A model-based approach to reactive self-configuring systems. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 971–

978). Retrieved from http://www.aaai.org/Papers/Workshops/1996/WS-96-01/WS96-01-033.pdf

Wu, X., Chen, J., Li, R., & Li, F. (2005). Web-based remote monitoring and fault diagnosis system. *The International Journal of Advanced Manufacturing Technology*, *28*(1-2), 162–175. http://doi.org/10.1007/s00170-004-2324-z

Zhang, G., Liu, R., Gong, L., & Huang, Q. (2006). An Analytical Comparison on Cost and Performance among DMS, AMS, FMS and RMS. In P. A. I. Dashchenko (Ed.), *Reconfigurable Manufacturing Systems and Transformable Factories* (pp. 659–673). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/3-540-29397-3_33

Zhou, X., Xi, L., & Lee, J. (2007). Reliability-centered predictive maintenance scheduling for a continuously monitored system subject to degradation. *Reliability Engineering & System Safety*, *92*(4), 530–534. http://doi.org/10.1016/j.ress.2006.01.006