Flávio Gil Albano Páscoa

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Lightweight Robust Behaviour Industrial Agent Methodology

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

Orientador: Prof. Doutor José Barata de Oliveira

Júri:

Presidente: Doutor Pedro Alexandre da Costa Sousa
Arguente: Doutor Luís Domingos Ferreira Ribeiro
Vogal: Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro 2013

Lightweight Robust Behavior Industrial Agent Methodology

Agradecimentos

Gostaria de começar por agradecer ao meu orientador, Professor José António Barata. Gostaria de agradecer também ao Dr. Niels Lohse pela oportunidade e um obrigado especial ao Dr. Pedro Ferreira por toda a ajuda, disponibilidade, motivação, orientação, companheirismo e enorme paciência durante todas as etapas da elaboração desta dissertação. Aos meus colegas de curso por toda a ajuda, pelos agradáveis momentos e amizade, em especial, Carlota Maçaneiro, Gonçalo Amaro, Bruno Rodrigues, Filipa Matias, Júlio Oliveira, Hugo Viana e João Santos.

Aos meus colegas de casa, aos atuais e aos que já foram, mas que por lá deixaram um vazio e saudade. Aos meus amigos fora da faculdade que, apesar da ausência, sempre lá estiveram por mim, Fábio Sebastião, Ana Paula, Adriano Pedro e em especial Inês Catarino que comigo sofreu e sorriu ao longo desta etapa.

Um muito obrigado a toda a minha família pelo suporte, apoio e confiança. Aos meus pais, Maria Júlia e José Gil Páscoa, e a minha irmã Flávia pelo 'levantamento da moral' nos momentos necessários.

Resumo

A indústria atualmente exerce uma enorme pressão nos sistemas de manufatura para que estes providenciem soluções adaptáveis e de rápida implementação que consigam facilmente responder às imprevisíveis necessidades do mercado.

Estes sistemas são predominantemente controlados usando 'Programmable Logical Controllers' (PLC) que não fornecem os mecanismos necessários para facilmente lidar com estes novos desafios.

O conceito de controlo baseado em agentes industriais foi apresentado como uma solução para dar resposta a estes desafios e suportar os novos paradigmas de produção baseados no conceito de 'Plug and Produce'.

A performance destas soluções de controlo baseada em agentes não foi ainda no entanto aceite como uma alternativa viável em relação às tradicionais PLC.

Este trabalho propõe uma arquitetura de controlo hibrida entre agentes e PLC tirando partido das funcionalidades de ambas as tecnologias.

O objetivo será avaliar a performance e viabilidade da utilização de agentes na implementação destas novas arquiteturas de controlo, e com isso, fortalecer a confiança na tecnologia perante uma indústria avessa a mudanças.

Palavras chave - Sistemas de Agentes; Performance; Sistemas de Manufatura; Controlo Industrial; Controlo Inteligente;

Abstract

Assembly systems today face significant pressure to provide highly adaptable and quickly deployable solutions in order to deal with unpredictable changes according to market trends. However, control of assembly processes are dominated by the use of Programmable Logical Controllers (PLC) which do not provide the necessary mechanisms to easily deal with these challenges. The concept of agent-based control has been introduced as a solution to deal these challenges and support new production paradigms based on the plug and produce concept. However, this solution has not yet been proven to be a real alternative to the traditional PLC approach in terms of performance. This work is investigating the use an approach that is able to benefit from the relative advantages of both PLC and agents solutions. A new hybrid architecture is presented which combines the functionalities of a PLC with those of industrial agents. The focus is on assessing the performance of this approach and help change the minds of an industry averse to changes.

Keywords - Multi-agent systems; System performance; Assembly systems; Industrial control; Intelligent Control.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Current Context

Nowadays, markets are globalized and consumers can be found everywhere through the globe. They are all different, but at the same time, all very similar with slight variances in their tastes. Attending to the consumers desires is more difficult than ever as fashion dictates short trends. Every new product has a reduced lifecycle and stays less and less time on the market which is increasingly getting more dynamic and unpredictable. At the same time, manufacturers need to be able to provide answers to all these market changes and requests. It is crucial that the manufactures can address these market uncertainties in the fastest and cheapest way possible.

Today assembly systems are customized and designed specifically to assemble a given product. If there is the need to change something in the product, or if a brand new product is required, all the manufacturing setup has to be changed. This requires a system design configuration and preparation which consumes a lot of time and is cost ineffective. Manufacturers require a way to adapt their manufacturing processes to this new dynamic market scenario. Therefore, the current way for assembly products, although fulfilling the objectives, could be improved to overcome all those market imposed challenges. There is the need for a new, more flexible, highly adaptable and reusable kind of manufacturing systems.

This have provided two different approaches to try to deal with the situations. introducing extra system functionalities for the eventuality that they'll be needed in the future [1]. This approach will solve part of the problem although increasing overall systems costs. Besides bringing extra flexibility into the system, it's impossible to predict what will be needed in the future, so those extra capabilities could prove themselves useless.

Other approach was a "Plug and Produce" system concept. Having the system divided by equipment modules that could be added or removed when needed. A modular system approach could then provide the basic structure so that these market needs could be met. Equipment modularity is then needed to archive full 'Plug and Produce' systems. However, equipment standardization is required for it which is quite a hard subject to obtain mainly because of all different equipment suppliers.

A modular based assembly line that can provide flexibility and reusability, easy configuration and reconfiguration, with quick and easy deployment and capable of delivering the same performance in the assembly process as one custom made, is then the ideal objective of these new kinds of systems. These modular approach systems were named Modular Assembly Systems (MAS).

Control architectures that allow for each, somehow independent, module to communicate and interact with the rest of the system are needed. Control architectures that allow any system topology, and that could easily deal with system changes is also required. Agent technology is seen as a viable solution to be used [2]. However, there are not many real industrial solutions using them. This is due to several factors, one of them is the lack of confidence by the industry in these new systems. This work will investigate about how reliable this systems are and that they can provide a viable solution.

1.2 Research Scope

Current advances in the MAS area and with the stabilization of knowledge around it allowed that new architectures that could implement MAS start to arise. These however doesn't cover certain aspects which will be the focus of this work.

Agent technologies by its own offer some of the dynamic and modularization that is needed for these types of systems [1]. Agents' sole objective is to work together to obtain through collaboration the emergence of optimal solutions [3]. It is possible to establish different base rules for each individual agent type. Each agent can then be a representation of an entity within the system. An agent can import into their model the specificities of each module and adapt their behaviour based on instances.

A few approaches to implement MAS have been proposed and studied [4];. However, manufacturers are still reluctant about the functionality and performance of MAS. It's important that they become more confident and receptive to such systems. For that reason, demonstrators need to be developed and tested so that performance and functionality could be evaluated.

1.3 Aim and Objectives

The objective of this work is to help to improve the studies and research being made around MAS. This work will aim to provide a comparison between these new systems against the used solutions. This will provide the means to access if they can provide a viable future solution. For that, a lightweight architecture must be developed. It will be simple and basic targeting optimal performance results. The viability of more complex architectures could then be assessed based on the obtained results. Advanced controller technology will be used as in the future this technology should be the normally used.

The lightweight architecture must be developed maintaining the trustworthiness that these systems requires. Several tactics must then be develop along with the architecture in order to address this reliability issue.

This work objective is to demonstrate that these new systems are functional while at the same time maintaining the characteristics of these systems paradigm; and also to provide some confidence about MAS by demonstrating the feasibility and reliability of such systems.

2 Literature Review

Assembly is a fundamental part in the manufacturing process. The assembly of a product involves a series of assembly processes that need to take place. These in turn require a series of equipment to deliver the final product. The control of this equipment is seen as a challenge in the Modular Assembly Systems (MAS) concept. Recently there have been considerable developments in this area, namely the Instantly Deployable Evolvable Assembly Systems (IDEAS) project [4]; [5]; [6]; [7]; [8]. This project provided the means for MAS concept to be demonstrated. However, the aim was not to provide system performance comparable to the currently used control methods.

In this chapter a review of the current state-of-the-art about manufacturing systems and agent technology will be made. This will cover which mechanisms are used and how it is evolving. This will cover which and how agent technologies are used, their limitations and advantages.

2.1 Assembly Systems Overview

Taking a glance at current assembly systems, these are mainly design and configured to produce specific products. Introducing changes to already deployed systems is hard and expensive.

One idea for dealing with this new market scenario is called Flexible Assembly Systems (FAS). The FAS point was to initially provide system capabilities that were not yet needed. These not needed but available capabilities have costs that are hard to justify, considering that the spare capabilities might never be used. This reasons made this kind of systems hard to deploy in industry [1].

Other manufacturing approaches were then considered and studied in order to attend to the manufacturer needs, some of them originated by the FAS idea. Bionic Manufacturing Systems (BMS) [9], Holonic Manufacturing Systems (HMS) [10];[11], Reconfigurable Manufacturing Systems (RMS) [12];[13], Reconfigurable Assembly System (RAS) [14], Evolvable Assembly Systems(EAS) and Evolvable Production Systems (EPS) [4]; [15]; [16]; [17] are some if these new ideas. All these systems have mechanisms to encapsulate functionalities in independent and self-contained modules. Those modules when joined together will provide the concept of a “Plug and Produce” assembly line [18]. This concept states that modules could be the base of any assembly system, from the simplest to a more complex, and new capabilities could be introduced by just adding or removing modules (equipment) [18]. This modular assembly system approach will be able to provide the needed capability of adaptation and scalability in a simple way [19]. This will allow manufacturers too quickly and easily adapt to any kind of market variations and necessities [18].

A modular based system can create several types of assembly system layout by just adding or removing modules [20]. Due to that, systems could be able to be adapted to any type of requirements. Such systems could provide almost infinite number of improvements [21].

MAS is one of the main concepts to take care of system configuration and reconfiguration [18]. The modularity can be considered on different levels on the system, from control approach or physical equipment [18]. The main issue behind the modularity is to achieve some standardization so that modules can easily interact with one another [19]. Using the modular approach could provide quite a number of advantages to the system, such as easy scalability, quick adaptability to new necessities, easy maintenance due to quick and easy module replacement, etc. [19];[22].

MAS provides all the required structures so that the idea of “Plug & Produce” systems could be used [18]. These systems are based on the standardization of system components and assembly processes.

This MAS approaches require control architectures that could adapt themselves automatically to any kind of changes made to the system by introducing different types of modules [23]. Some developments in towards that path have been made, namely by the creation of platforms with various levels of granularity, although none of them being yet deployed in any real industrial environment [24]; [25] [26]; [27]. However, recent research on modular systems was made targeting their deployment in real industrial environments [8].

To take full advantage of a modular system, it is not as simple as join all the modules and their capabilities together. To evaluate the full capability of a system, ways to identify the joined capabilities of the modules and its behaviours are needed. Having in mind the assembly system architecture, this can provide the foundation to define the system organization as well as all the possible modifications that could be made to it [12]. Any modular system can be used to perform any given task just by changing, adding or removing some modules (functionalities).

All the developments around MAS research allowed that new models and concepts to be developed [28],[8]. At the same time, new control solutions could then be established to validate the MAS concept [4].

2.2 Assembly Systems Control Overview

The current control process is pre-programmed to the considered system setup. When a new product needs to be assembled, all the infra-structure has to change, including the control. A change from these hardcoded systems to a more flexible and dynamic ones needs to be made [29].

Programmable Logic Controller (PLC) has been established as the dominant standard for system control [30]. One of the main reasons for this is its ability to provide real time solutions [30]. PLC’s have evolved in the assembly system domain so they could be able to deliver more flexible solutions [31]. One of these changes consists in the standardization of an open

architecture for distributed control through the IEC 61499 standard [32]. This is based in a function block concept and target distributed control of systems. This will provide some reusability to PLC code and much quicker control integration [32]. All improvements made in the PLC technology are proving solutions for current problems, nevertheless, some more effort are still required so it could fully comply with what is needed [4]. PLC challenges have been identified and are being addressed by its suppliers, which are currently providing solutions to enable interfaces with the higher lever programming languages. However, this new solutions have not been fully tested and there are still some challenges in their usage, particularly about their performance.

Some fundamental issues that could provide a stimulus in the implementation of more adaptable systems have been recognized in the literature [29]. Reducing manufacturing costs, increment in productivity using automatic processes, and also, the market increasingly unforeseen needs were seen as one of them [33]. This system change could significantly reduce the costs related with the manufacturing of new products, as well as in the adaptation of existing assembly lines to markets fluctuations. Multi-agent systems are seen as a viable solution to control MAS [2].

2.3 Multi-Agent Systems

A few definitions for ‘agent’ are proposed by the literature. However, there is no agreement in a single definition. An agent could be described as a complex software ‘entity’, that has the capability of a certain degree of independency, as well as being capable of interacting with others, and with the surrounding environment, in order to achieve a certain determined goals [3].

An agent could be basically defined by all the behaviours it has [34]. It’s easy to establish different types of behaviours for each agent type. These behaviours could be customized to the entity the agent represents, or just customized to fill some of the system needs. An agent type would then have equal behaviours, changing only each agent internal information. Having different agent types, each with their own customized behaviour could then allow complex objectives to be achieved through combined agent collaboration. This approach type (bottom-up approach) has been identified in the literature as the best approach to solve any kind of modular problems [2].

Agents are capable of making reasoning beyond their initial defined behaviour which makes them an improvement comparing to conventional artificial intelligence. Agent capability to communicate can then be used by agents so that collaboration can be obtained. This agent collaboration will then provide the means for complex solutions to be achieved. These agents key features allied with their communication abilities allows the existence of multi-agent systems [2]; [3].

2.3.1 Multi-Agent System Architectures

An agent type could be seen as an individual existence, which represents some software or hardware capability present in the system, and to accomplish that each of them should have different personalized behaviours and objectives [3]. This organized agent environment will be populated by different agents. To accomplish each independent existing agent objective more than one agent could be necessary [3]. Defined and pre-established mechanisms of interaction between the agents are needed. Agents should use their inherited capabilities, like actuating, communicating, etc. with the other agents' presents in the system to accomplish it. This multi-agent environment complexity could be very different from one system to another. The complexity of a system should be defined by the agents' behaviours and its organizational structure [3].

Agent architectures need to clearly define all agents that are present on it, their organization, objectives and behaviours. To obtain that, a methodology is needed. Some methodologies to define multi-agent environments have been considered [35]; [36]; [37]; [38]; They all agree that a clear definition of the needed requirements as well as the objectives is needed before the multi-agent environment definition [39].

2.3.2 Multi-Agent Systems in Assembly Environments

Agent technology is considered a viable solution to implement the next generation of assembly systems [1]. Agent technologies have already been used in multiple manufacturing areas such as concurrent engineering, collaborative engineering design, manufacturing enterprise integration, etc [1]; [40]; [41]. All of these demonstrate how much this technology could provide solutions to the assembly domain.

Agent technology could be an advantage in modularized assembly environment in several different ways. Using agents could allow different approaches in the areas of product design, engineering analysis, simulation and execution, etc. [42]; [43]; [44]; [45]; [46]. Improvements in any of these mentioned fields will mean a huge step forward in system integration allowing significant costs reduction [47]. These are however, in its majority, specific to some sort of application [48].

Agent architectures to implement a MAS have been proposed and studied [49]; [50]; [51]; [52]; [48]. One of the biggest advantages about using agent technology is its adaptability and embedded capabilities. This allows it to easily be used in a diverse type of systems and areas.

2.3.3 Multi-Agent System Organization

When several agents are present in a system, the way these agents are arranged has a crucial role in the overall system performance [3]. One of the main factors that influences the complexity of agent based systems is the type of agent present and their roles. Some methods on how to organize the agents can be found in the literature [47]. Typically agents can be organized in three ways: Hierarchical, Federation and Autonomous.

The hierarchical method takes advantage of the existing structure of manufacturing environments, where there is a workstation that contains equipment units that execute certain operations. A few demonstrations of hierarchical system could be found in the literature [53]; [54]; [55].

The federation method establishes the formation of agents groups while a dedicated agent is responsible for all the agents in a group. This dedicate agent is in charge of all the communication among the agents, local (between members of the same group) and remotes (with other groups).

The autonomous approach leaves each agent responsible for itself. It has no other entity (agent or operator) responsible for him. All the agent interactions are handled by agents themselves, without the need to use mediators or any other entities. Because each agent is autonomous and independent, each agent should also be able to be aware of the environment they are placed in, and also any other agents that are part of it [45]; [56].

2.3.4 Multi-Agent System Communication Protocols

In any agent environment, more than one agent type is expected to be present. Like in a community, each agent is also expected to know how to communicate with any other agent. These is obtained using protocols. There is the need to create protocols so that messages could be understood between all the intervenients [57]. These should define all interactions that could be made between the agents [57]. Protocols should be focused only on the content of the messages and not how the messages are sent and received. Also protocols should be specific and created to a focused domain and generalized as much as possible [57].

The most used protocols are based on the Contract Net Protocols [58]. However, the majority of them are made to some specific problem [53]; [59]; [60]. There are some market-based protocols based in auctions, which are easy and simple to use, but these are mainly being applied in scheduling systems [61].

For the protocols to be correctly used, communication methods are needed. Studies around this subject converge to two main communication languages KQMP [62] and FIPA [63]; [64]. FIPA provides an open and flexible method so that agent language and protocols could be easily defined [63]. FIPA is the largely used method, so it is considered the best option to be used.

Considering manufacturing systems with an agent based environment controlling it, communication could mainly be used in the process of picking which intervenient could provide the best service to accomplish some goal [47].

2.3.5 Multi-Agent System Negotiation

Negotiation can be seen as a group of coordinated interactions that aim to achieve the best final scenario for all participants [65].

In any multi agent system, agents need to collaborate to achieve their goals. All these agents could be similar, with the same interior behaviours and slightly different objectives. To achieve system equilibrium and viability, a way to allow proper means of interaction between all the agents needs to be present and well known. Only with this it is possible to create negotiation mechanisms. These will provide the means for cooperation so that diverse goals can be achieved, or even to settle any kind of disputes that may occur [57].

The negotiation process ends when an equilibrium state in the system is achieved [66]. This can only occur when all the negotiations have been made and have been settled [66]. This state is obtained when all the agents have reached an agreement that is suitable for each involved part and there is no need for changes [66].

2.3.6 Multi-Agent System Negotiation Strategies

Strategies should define what actions should be made so that the agent objectives can be fulfilled [57]. What should be compromised against what should be gained will be defined by the agent given strategies [65]. The objective of having them will be to try and optimise the negotiation procedures [65]. Protocols are pre-established and should not be changed, however, how to react to each received message is defined by the agent negotiation strategies respecting the existing protocols [57]. Well design negotiation strategies could help to considerably improve the negotiation process [57].

Taking assembly systems into consideration, a few negotiations strategies have been proposed by the literature [1], such as game theory based negotiation, contract based negotiation or AI based negotiation. However, it does not establishes a clear method to be used. Nevertheless it is suggested that a process analysis should be made and a mix between strategies could provide a good solution [67].

In multi agent based systems, negotiations strategies should be able to deal with all the unexpected scenarios that may occur in the agent environment [57]. However, during negotiation some conflicts may arise. An agent enters a conflict state when he encounter something that may differ or even goes against his objectives [57]. Even when several intervenients can satisfy some need, which one should be chosen is also a conflict that needs to be solved. When a conflict is acknowledged, a viable way for the agents to handle it should always be provided. These can be partially solved using negotiations procedures [57] or designed in the agent environment in such a way that these conflicts are minimized [47], or even using both strategies.

2.3.7 Agent Technology Applied to Assembly Systems Overview

When applied to system control, agent technology will have significant challenges on performance [4] although bringing a high adaptable solution. Agent systems are, in its majority, currently implemented in a Java-based JADE platform [68]. Java is known to have some issues with performance concerning real time processes [69]. Some considerable progresses are

being made to attend to this issue [69]. Also, it is acknowledged the fact that agent-based control system will also need to take in consideration the time taken for agent communication and negotiation [68].

However, agent technology turns out to be one of the most interesting solutions to deal with the trend of modular assembly systems. Agents themselves can be considered as modules which can be arranged and rearranged to adjust to the needs of a system [48]. Agent technology could use a representation approach of the real system (e.g. machines, robots, tools, etc.), and parts, operations and processes [48];[8]. The key feature of agent technology is the agents' communication capabilities which aim to establish collaboration which can lead to solving problems [3]. Agents having the communication embed in the technology which will lead to lower system integration times [57]. Also, the agent communication capabilities allow them to interact with the environment which is crucial to archive the full concept of "Plug and Produce" [1].

2.4 IDEAS (Instantly Deployable Evolvable Assembly Systems) Project

The IDEAS project [8] aims to reduce the time taken in the configuration and reconfiguration of assembly systems [4]. This is accomplished by having modular assembly equipment with standardized interfaces and integrated control mechanisms. The IDEAS project establishes an assembly process model that specify Mechatronic Agents for the new MAS scenario [4]. The modular structure and the Mechatronic Agents put together will provide the means to swiftly connect modules and also allow autonomous configuration [4]. Each mechatronic agent will represent a separate module. With this a vast variety of assembly processes could be added or removed in a simple manor [4].

The Mechatronic Agent concept goes even further than just to allow modular equipment plug-in capabilities. It also aims to allow modularization of the capabilities that are needed to produce the product [4]. These capabilities have a direct relation with the existing assembly module blocks available in the system. Also, when a mechatronic agent is plugged into the system, it should provide its own process capabilities, named Skills [70].

The IDEAS assembly process model establishes the concept of Skill and also defines how Skills should be used in the configuration of mechatronic assembly systems [28]. These Skills need to provide the means for different agents to be introduced by the equipment providers to the system and also the ways for them to work with one another.

The Skill Model was defined for the IDEAS project as a representation of the assembly capabilities to be used in an agent control environment [28]. For that reason, its specifications will be analysed.

2.4.1 Skill Concept

The IDEAS assembly process (skill) [49] should describe and represent an assembly process capability, which can be simply processed and understood in an agent technology

environment. This concept also provides the functional description for assembly processes and a definition for its execution.

Basically, a skill is a capability offered by an agent that could accomplish an assembly process step. Each skill will be a representation in the agent architecture of an available assembly process. For a Skill to be defined four main aspects needs to be clarified. Namely the assembly process type it relates with, the level of granularity, the skill control ports and its parameters ports [28]. An overview of this can be seen in Figure 2.1.

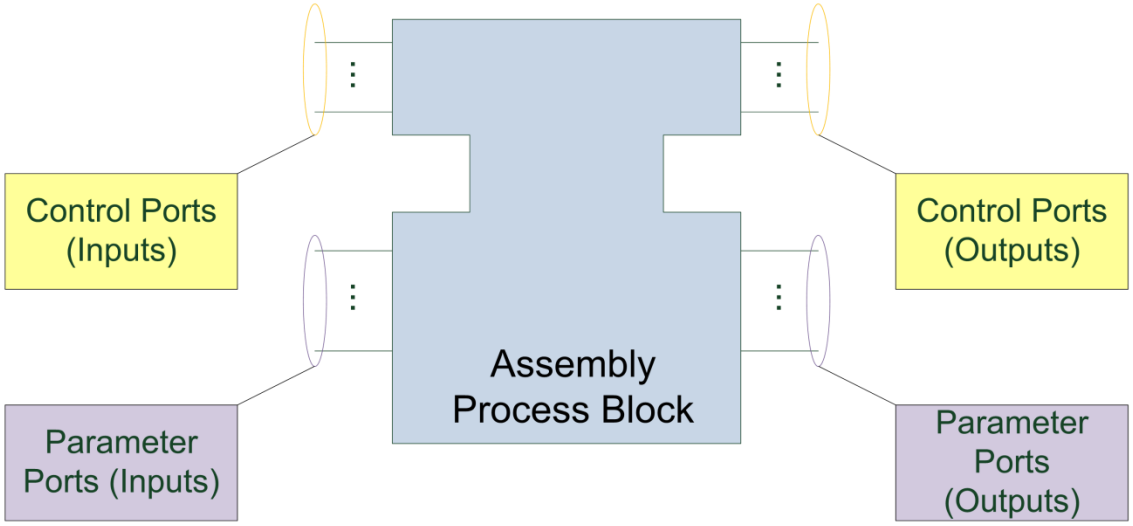

Figure 2.1 - Conceptual skill overview [28]

The control ports objective is to control the execution of a skill (start, stop, finish, etc.). With this the assembly process associated with the skill can be controlled [28]. This is the base so that a process sequence could be created since a connection between the different skills (assembly processes) could then be established [28]. The parameter ports are the means for different nature of information to transit from one skill to another [28]. In some cases the parameter ports will not be used at all, but some assembly activities could need some sort of information to be handed from one process to another. These ports provide the means so that can occur [28].

The assembly process type will allow relating the skill with a specific class of assembly processes [28]. This can then be used to obtain a group of Skills that can accomplish a same task. A skill should be defined by the assembly process characteristics that it's associated with [28]. Using the function block concept it is possible to create more complex skills from lower level ones [28]. The granularity of a skill will define if it's an Atomic Skill or a Composite Skill [5]; [6]; [49]. Each Atomic Skill will have a single assembly process associated with it. A Composite Skill can be constituted by a group of Atomic Skills or by a group of other Composite Skills, or even a mixture between Composite and Atomic Skills [5]; [6]. The external difference between

them should be inexistent. However, the Composite Skill should organize the process sequence and the information flow between each of the Skills that it is constitute by. A schematic the inside structure of a Composite Skill is shown in Figure 2.2.

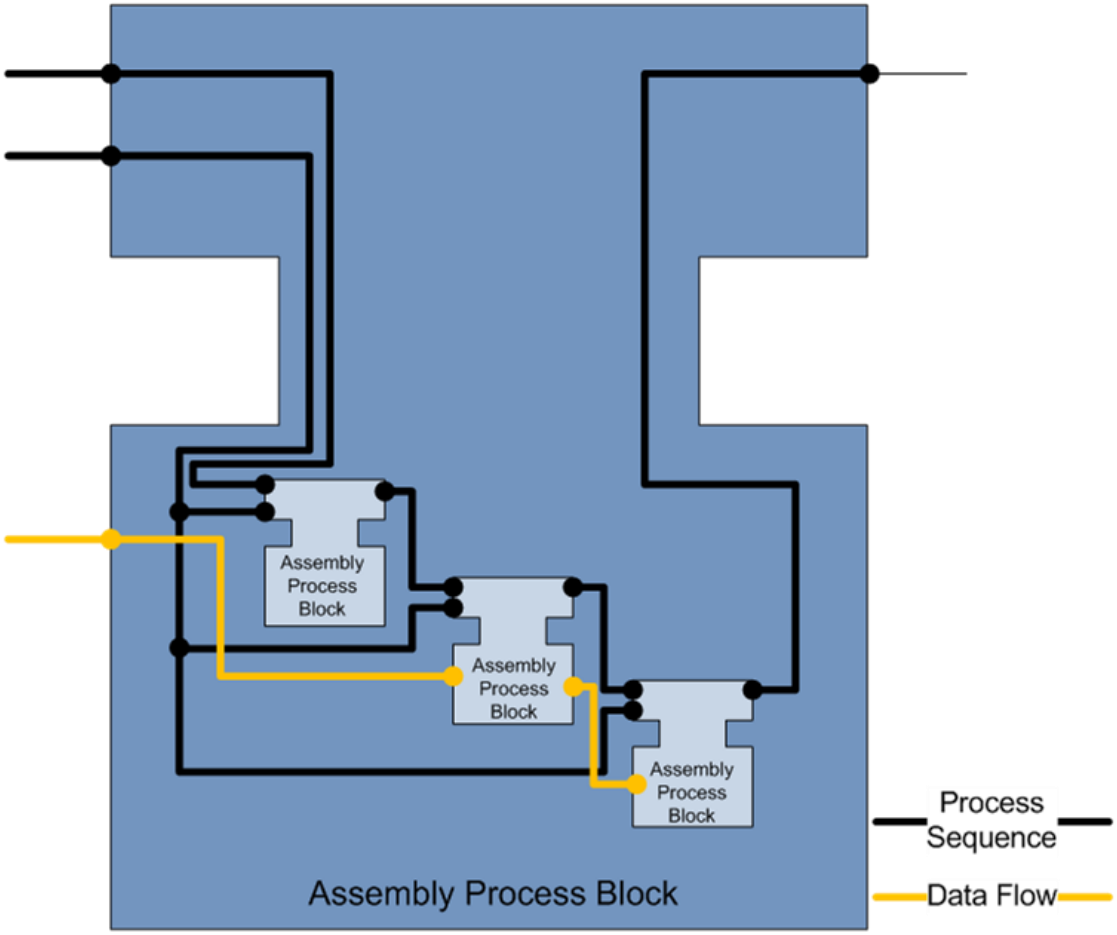

Figure 2.2 - Composite skill concept overview [28]

2.4.2 Skill Requirement

A Skill Requirement has the same conceptual attributes has a skill. However, instead of representing an assembly process, it will represent an assembly process that needs to take place so that a product could be assembled [28]. Each Skill Requirement should match to one of the Skills (Atomic or Composite) available in the system. A complete match between the entire Skill Requirements and Skills provided by the agents should be made so that a product can be assembled [28].

Figure 2.3 provides a conceptual view of this process.

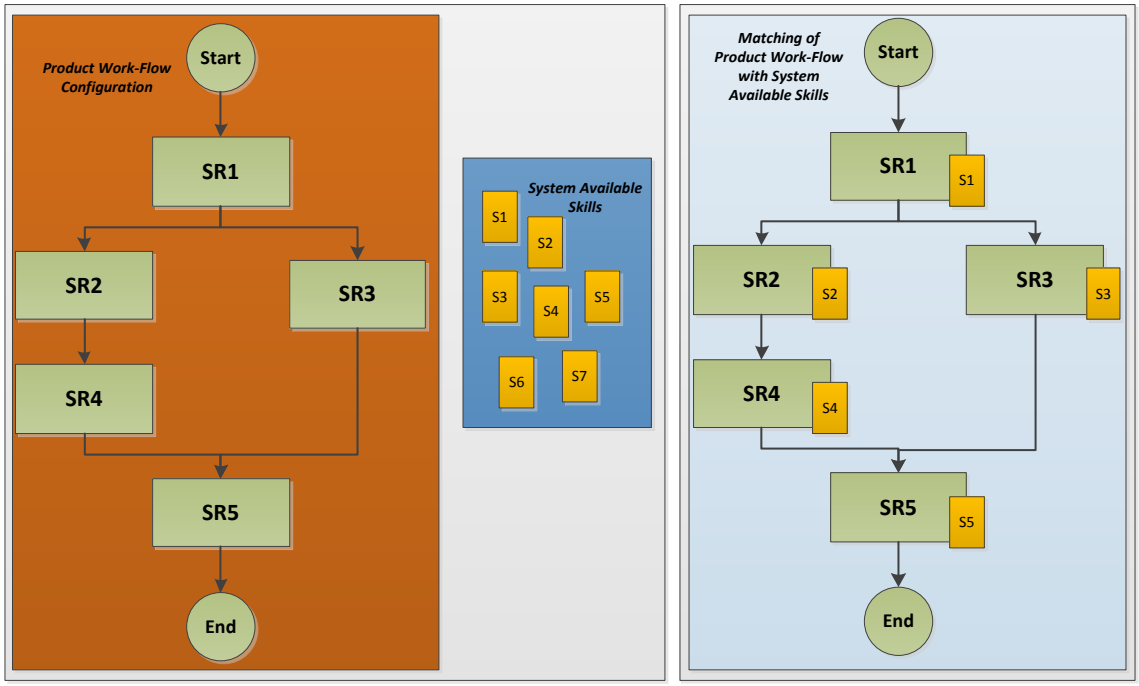

Figure 2.3 - Product work-flow configuration

The initial configuration step consists in defining all the required assembly processes (Skill Requirements) needed for the product [28]. The assembly sequence and necessary parameters can then be considered. The collection of Skill Requirements and how they are linked will represent the product workflow [28], this process is shown in the red rectangle in

Figure 2.3. Having the product work-flow, a full match between the Skill Requirements and the Skills available in the system (represented in the dark blue rectangle in

Figure 2.3) need to exist so that the product can be assembled [28]. This can occur in real time, or pre-established during the workflow creation time by giving to each Skill Requirement the system associated Skill [28]. This last attribution mechanisms overview is seen in

Figure 2.3 in the light blue rectangle.

2.4.3 Precedence Execution Methodology

Once the product work-flow is defined, there is the need to understand its internal execution structure. A functional sequence between all the Skill Requirements should be made [28]. Normal execution use a linear approach, where a Skill is executed one after another in a pre-established order. The Precedence Methodology provides the means for the Skill execution decision process to be made during running time [8]. Some skills might be executed in parallel, but not at the same time. Others could happen in parallel and at the same time. Other Skill might even require that some other skills end before the actual start of their operation. Defining the execution mechanisms in the assembly process could be achieved by simply defining

before and after relations between each assembly process (precedence constrains). This could allow different execution mechanisms to be included in the same model. Figure 2.4 shows a general view for the execution of the Precedence Methodology.

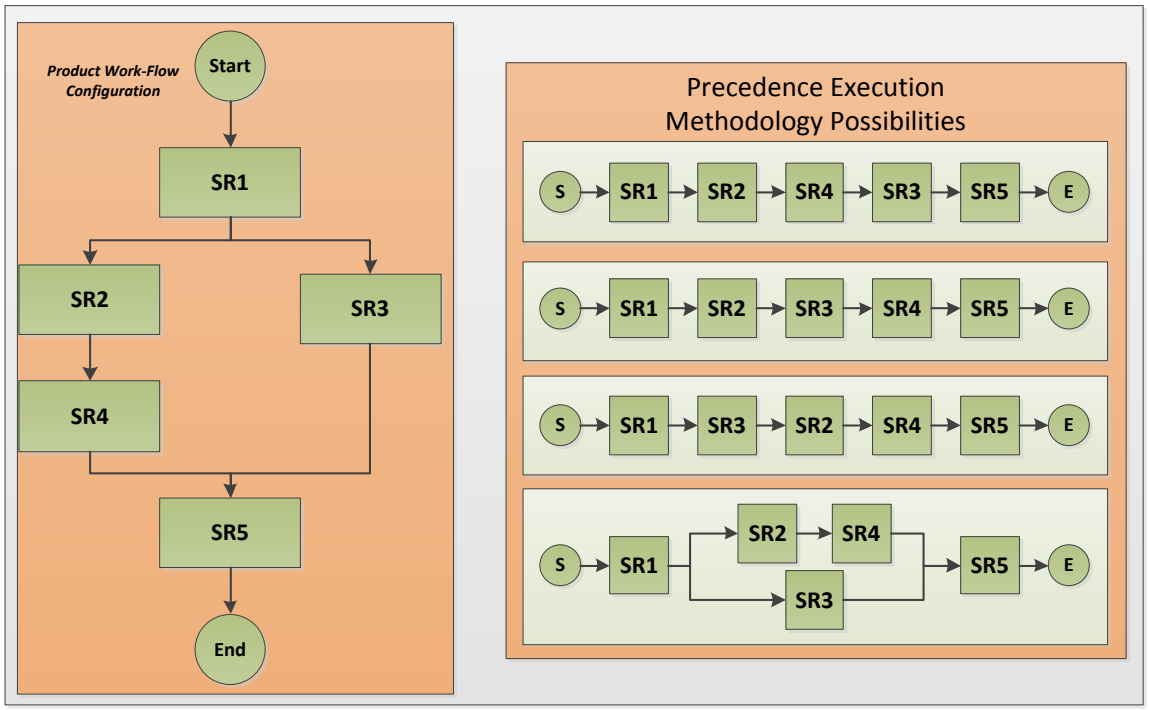

Figure 2.4 - Precedence execution methodology overview

There is a clear relation between each assembly process step. Skill SR2 and SR3 can only occur after SR1 has ended its execution. SR2 or SR4 could be executed in parallel with SR3, however SR2 and SR4 could not execute at the same time. SR4 could only start its execution once SR2 has finished. SR5 can only be executed when SR4 and SR3 have been executed. This skill execution behaviour defines the Precedence Execution Methodology [8].

3 Research Approach

3.1 Introduction

Current market scenario, where changes occur quickly and even small future projections are hardly made, opened the path for new assembly systems concepts to emerge. Modular Assembly Systems (MAS) is one of them. This work was encouraged by the current developments being made around MAS. To comprehend the goals of this work it is important to analyse the current state-of-the-art in the control of MAS.

One method for MAS control is a hybrid approach joining agent and PLCs technology. However, this method is not yet been proved as an ideal alternative for MAS control. A hybrid control architecture model that takes advantage of both technologies is then proposed. It will then be used to access the performance of this hybrid approaches and provide an initial feedback regarding their usage. Having the models is not enough. Validation scenarios will also be designed and executed to confirm the proposed models objective.

3.2 Problem Definition

Markets nowadays are more active than ever. They are dynamic and unpredictable. Manufacturers need to have the means to answer to unforeseen market demands and requests. Industrial assembly lines are mainly using PLCs to control the assembly processes. This approach provide real time control response in the system, however has a downside concerning system configuration, reconfiguration and reusability. Literature shows new approaches to deal with these challenges.

Pure PLCs controlled assembly lines are hard to readjust and reuse. It is complex to implement ‘Plug and Produce’ using only PLCs. Also, PLCs are not easily changed. Alterations to PLC code is a hard task to be performed. Reconfiguration of an existing system is also very difficult. Therefore, only PLC code is not the most suitable option for this new concept of assembly. Agent technology could be used to overcome the PLC limitations concerning flexibility and adaptability. However, agents also have limitations concerning real time responses.

A hybrid approach using agent and PLC technology can then allow a more suitable response. However, few performance tests to assess their viability have been made. For that reason, a light control architecture model using the Precedence Methodology is proposed. The Precedence Methodology for a MAS implementation is well documented and studied but a deployable architecture to take full advantage of it is still missing. This model will allow a benchmark for this hybrid approach to be made, as well as to validate the Precedence Methodology.

3.2.1 Requirements for the Hybrid Control Architecture Benchmark

The proposed control architecture must respect the concept of MAS;

A proper method for the representation of the assembly processes within the agent environment is needed; this will then require a mechanism to execute these assembly processes;

A mechanism to store and broadcast the system available assembly processes is required; it needs to be reliable and possess the adaptation methods to deal with system changes and errors; it also needs to provide the assembly processes information redundancy and delocalization.

For the performance benchmark to be obtained, a method to measure the actual assembly methods performance is needed; also, a method to measure the performance of the proposed architecture is also required. A process to compare both performances is then also required.

3.2.2 Definition of Research Objectives

The goal of this work will be to obtain a fully functional lightweight control architecture for a MAS model that takes advantage of the Precedence Constraint Implementation approach. This architecture will take advantage of the agent technology adaptability and scalability while maintaining the PLC reliability. This control architecture will provide the minimum functionalities needed. This will allow for the top results to be obtained. Several negotiation strategies will be compared in order to evaluate which provides the best performance.

A mechanism for the agents to interact with the PLC will be established, as well as a method for the agents to trigger the assembly processes execution.

A Capability Dissemination Agent (CDA) will be introduced to store and broadcast the existing system assembly capabilities. Also, this agent will provide redundancy and delocalization of the assembly capabilities information.

The aimed architecture should be liable and viable so it can be used in industrial environmental setups. The architecture should provide redundancy having in mind the ‘Plug and Produce’ idea. Abstract means of deployment of the architecture will be made and also independent ways of assembly processes execution are presented.

3.3 Research Methodology

The developed efforts referred in this thesis were originated by first understanding the basics concerning the specific topic this work is about. It started by a deep literature review about the domain and an analysis of the State-of-the-Art around it. When the basis had been understood, gaps and needs could then be identified, which leads to a problem.

Having a problem defined a set of requirements and objectives could then be formulated. This allows for the research hypothesis to be created. The hypothesis for this thesis is the lack

an existing agent architecture that could be used for the execution of a deployed MAS that runs the Precedence Methodology as its core.

The advance in informatics systems will allow agent technology to provide a quick response performance. With that, it will make them viable for the control of MAS. To validate this, a simple control architecture using agents is created. This will allow the idea to be validated and confirmed. Analysing the performance of it will then provide the knowledge about the viability of these systems.

For this, the architecture must guarantee that the flexibility that makes agent technology a good approach for these system is maintained. After the research hypothesis had been establish, an enumeration of a range of scenarios to validate it could then be obtained. To finalize this work, a review about it should be made, where conclusions and future work should be established.

3.4 Definition of Validation Scenarios

To prove the architecture performance, a simple micro scale assembly line with several stations will be used. The station is part of the SMC HAS 200 assembly system showed in Figure 3.1.

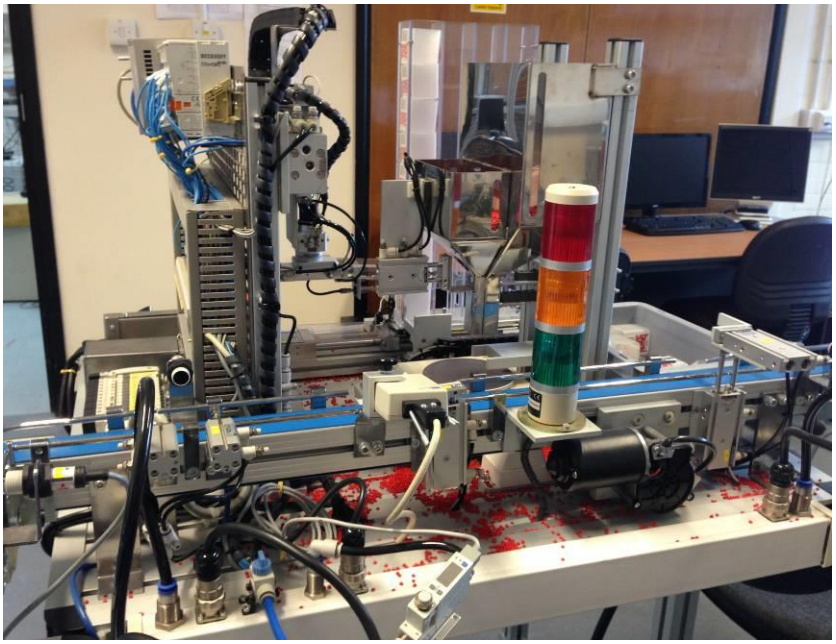

Figure 3.1 - SMC HAS-200 station

A station will consist in a number of electro-pneumatic actuators and a number of sensors. The control of these will be made through an industrial controller (Beckhoff CX-1030) which will be running embedded Windows XP operating system. The controller will also be emulating soft PLC's. The mentioned station has as its main objective to produce a product. It achieves this

task by executing a sequence of assembly tasks (Skills). The final product that the station aims to produce is a box full of components. This box can be seen in Figure 3.2.

Figure 3.2 – Final product

The first Skill is the feed of an empty box to the station working space. Next, the box will be picked up by a manipulator, using a simple pick and place process, and taken to the bar code reader. After it is scanned, it is taken from the scanner to the filling area. The filling Skill is then executed and the box is filled with components. After the filling ends, the box is then carried from the feeling area to the conveyor belt to be dispatched from the station. This execution sequence gives the product workflow. The execution of the mentioned skills will create the final product. This Skill execution sequence is presented in Figure 3.3.

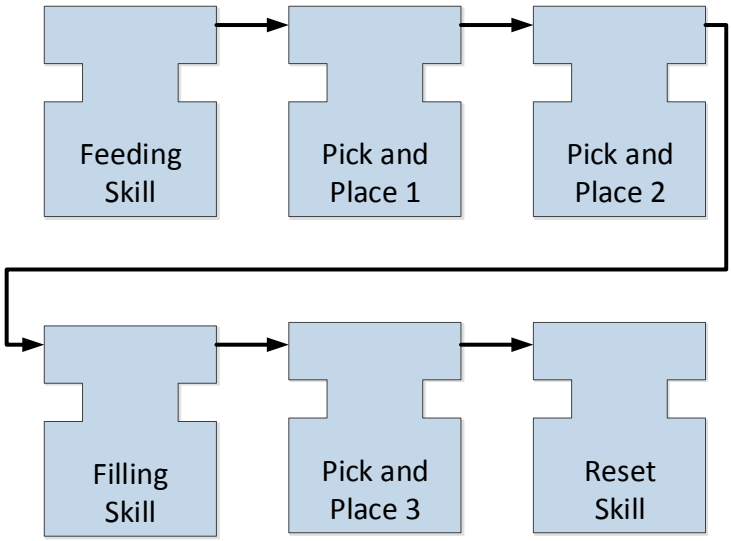

Figure 3.3 - Process workflow overview

This product creation will be used for the final performance comparison. The first experimental scenario will be used to benchmark the PLC performance. For that, PLC code is optimized and timers will be introduced into it taking the time the process takes to each individual skill, and also the total time taken to assemble the product.

The next experimental scenario the assembly processes to be executed by the PLC will be triggered by java. This will allow that the proposed interaction model between the object oriented programming language and the PLC to be validated. Also, in this scenario the delays of interfacing with java can be measured.

In the third scenario, the agent architecture will be deployed and connected with the system. This system will also be executing the PLC code necessary to execute each of the previous mentioned Skills. The agents will then trigger each Skill in order to accomplish the final desired product. This scenario will be used to validate and analyse the performance of the architecture. This will validate the proposed lightweight control architecture.

For the CDA validation, the previous experimental scenario will be using it to store and broadcast the assembly capabilities. To validate its delocalized and redundant characteristics, three CDAs will be running in the same network. One will then be shut-down and reintroduced into the system to analyse the CDA adaptation capabilities.

This experimental scenarios will then cover all the aspects the models are designed for.

4 Hybrid Control Architecture

4.1 Introduction

The need for a reconfigurable and reusable new type of manufacturing systems has been identified in the literature. Due to the concept of hardware modularity, distributed control systems are also required to control each individual module [29].

Current manufacturing systems are mainly PLC controlled. These are well established in the industry mainly due to their control time response. PLC are used in a dedicated hardware to read inputs, apply the control logic and set outputs within some defined real time constraints [30]. Also, PLCs are strong against noise, humidity, temperature, etc. which makes then ideal for industrial environments [30]. Although PLC is an established technology that has the capability to be able to be programed by technicians, it has some significant limitations regarding reusability and adaptation.

As seen in the literature review, having complex systems, the control solution can also be complex and difficult to readapt. Readapt an existing PLC control to a new system configuration can prove to be a difficult task to accomplish. Despites the PLC improvements to obtain modular control solutions, it stills falls short for the requirements. Thus the Modular Assembly System (MAS) control is mainly being designed in high level languages. These can provide an abstraction to the component level in order to control multiple instances of the same entity. Agent technology is seen as a suitable solution to execute the high level control logic on MAS. Agents use cooperation in order to obtain their individual objectives, just as equipment modules in a MAS. Agent approaches provides an object oriented architecture, inheriting advantages such as modularity, method transferability and extendibility. Although the advantages of agents have been demonstrated in a number of production systems, industry has not yet adopted the concept on any larger scale. Real time control using this technology is not yet completely accept. Most agent systems are currently implemented in JAVA based platforms which have known real time response issues. Finally, the current agent solutions report significant challenges to achieve real time performance in production systems. These issues have led to limited industrial application of agent-based control systems. Industry is reluctant to change existing working solutions completely. Instead it prefers to apply step-by- step changes which the PLC technology has been supporting. Combining the relative strength of PLCs with those of agents, instead of using mutually exclusive approaches, could provide a satisfactory solution maintaining the required real time control response, while adding the also needed re-configurability. This chapter proposes a MAS hybrid control model solution to obtain it.

4.2 Hybrid Control Architecture Requirements

The MAS delocalized hardware modules also requires a delocalized control architecture. The ability to place pieces of PLC code into modules exists, however this needs to be formalized and aligned for independent and parallel triggering.

Introducing control adaptability and configurability must maintain the almost real time control responses pure PLC solutions can provide.

The system assembly capabilities description must be decoupled from their execution in order to be triggered independently.

A triggering mechanism for a high level control to start the assembly capabilities execution and acknowledge its termination is also crucial. The assembly process termination information must be sent to the high level immediately. For this, a two way communication between both levels is also required.

4.3 Hybrid Control Architecture Model

PLC logic is mainly used in the assembly process control due to their control response time. However it lacks in adaptation as it is not designed for a MAS. Control strategies based on java are seen as a good solution to provide a MAS the reusability and re-configurability it needs [48]. However these control architectures have known slow response issues on direct hardware control [69]. Combining PLC and java can then provide a feasible solution taking advantage of the qualities of both technologies for the MAS control. An overview of mentioned scenario can be seen in Figure 4.1.

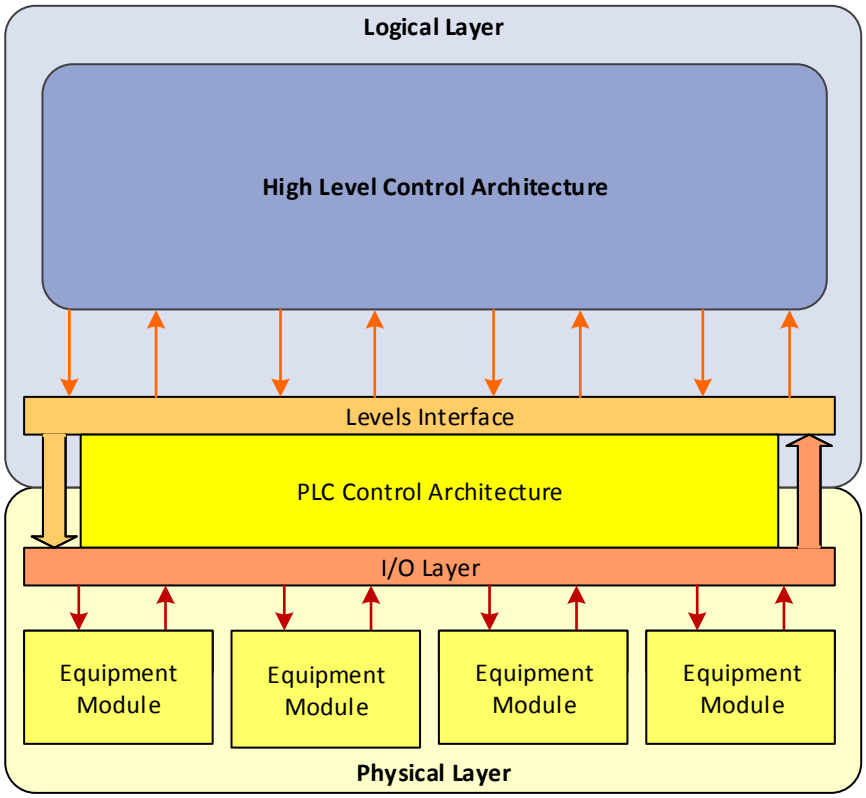

Figure 4.1 - Hybrid Agent / PLC architecture overview

The high level control architecture will be where the complex control of the MAS will be made. This will provide the features to allow the Plug and Produce concept. The Levels Interface will allow the interaction between the high level control and PLC control. The PLC control architecture will be controlling the I/Os of the hardware modules, performing the needed real time control.

Based on the advantages and disadvantages of both technologies this chapter objective is to define a hybrid control architecture to take benefit of the PLC near real time control capabilities while having the high adaptability and re-configurability provided by a high level control architecture.

4.3.1 PLC Control Architecture Description

PLC control architectures directly actuate the hardware I/O interfaces which results in the execution of the assembly processes. When the PLC executes an assembly process, it follows the defined PLC control logic. This control logic is however designed for specific systems and can be very limited in its reusability.

By breaking the hardware into modules, the PLC control logic associated with that module needs to also be broken. This will formalize the concept of a plug and produce system having the hardware along with its control logic modularized. An overview of this idea can be seen in Figure 4.2.

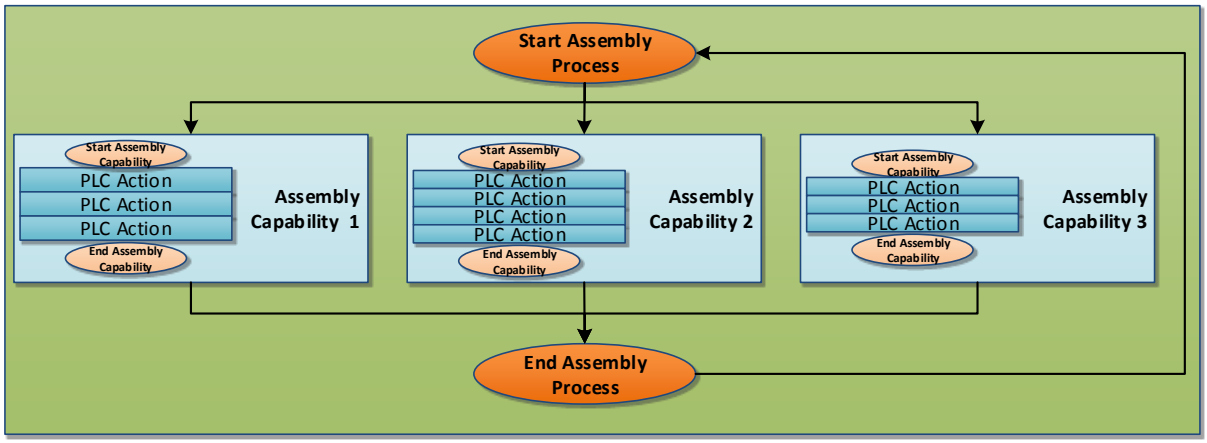

Figure 4.2 - PLC logic modularization

The PLC Action are each individual action the PLC needs to execute in the hardware to get the desired Assembly Capability executed. These can be reading sensors, actuating valves, etc. Considering the modular concept, each different assembly capabilities could be divided in blocks of actions. These blocks will then be the PLC control logic for a specific hardware module assembly capability. A mechanism to trigger each block independently is then required so that PLC Execution parallelism can be obtained. With this, different PLC execution sequences can then be easily generated using the same code.

Having the PLC code modularized, the capability execution triggering can be obtained expanding the PLC code by adding control variables. These will delimitate the PLC control logic for each of the individual assembly capability. One is used to start the capability execution and other to acknowledge its termination. These need to be unique for each different block.

These variables provides the means for each block to be individually executed. It can be used by the PLC control architecture itself, or for high level control architectures to independently control each modularized assembly process execution. The proposed variables extension model overview can be seen in Figure 4.3.

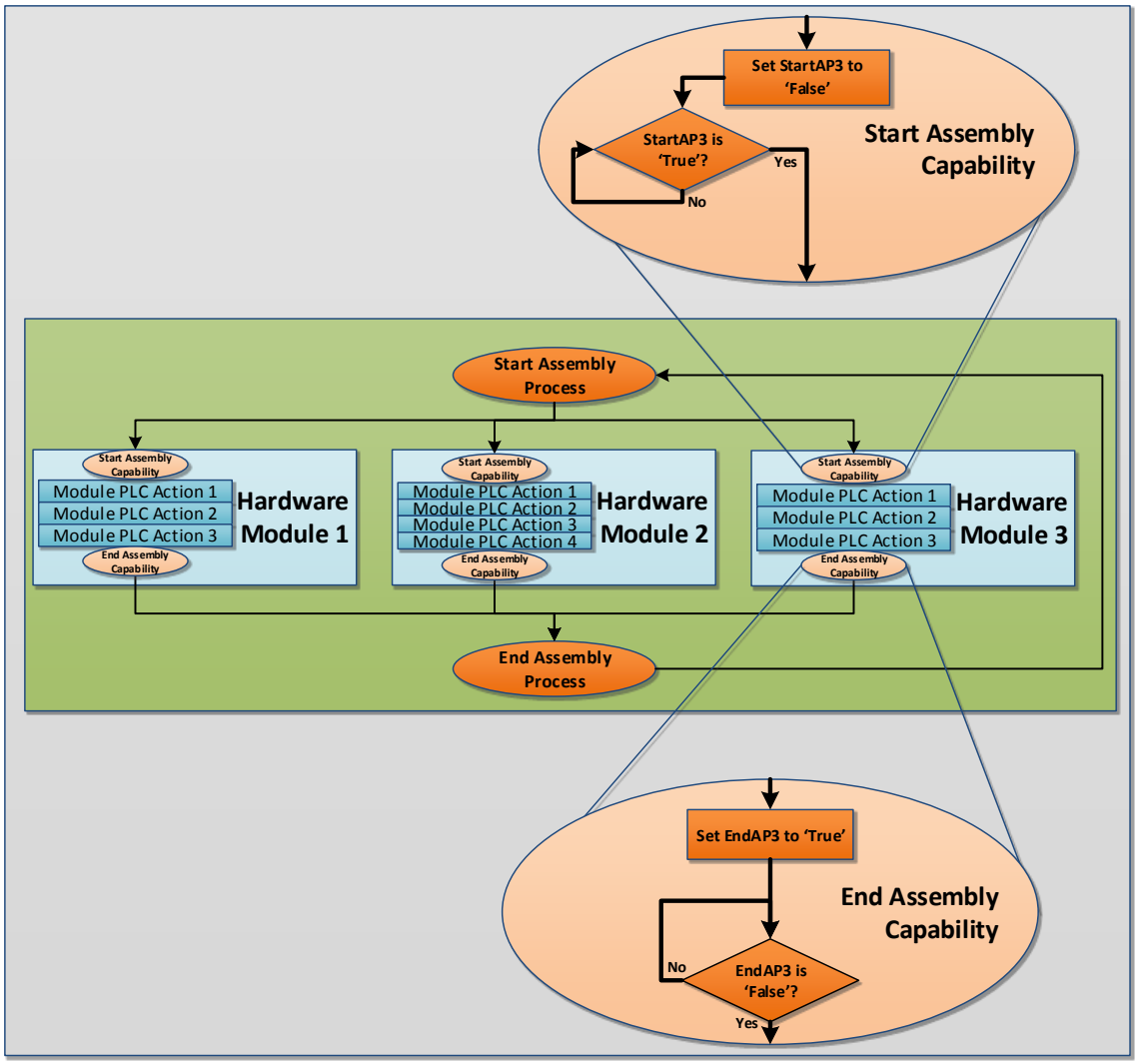

Figure 4.3 - PLC control architecture overview

The **StartAP3** is the logical variable considered to start the block code execution, and **EndAP3** is the logical variable to signal its end. When **StartAP3** is set to '**true'**, the execution of the PLC code associated with the block starts, which will then do the desired assembly capability the block represents. When this assembly capability ends the end block logic variable will be set to '**true**', in this example **EndAP3**. This will allow the block triggering entity to know that the block execution has ended. It will then be this entity to again set the logical control end variable back to '**false'**. This will provide a consistent and reliable way to control each of the PLC execution blocks. If the block end variable was changed by the PLC, some errors or misunderstands could be transferred to triggering entity. This could occur due to the difference in the response time capabilities of both PLC and high level architectures. Having the triggering entity controlling the block control variables will help to provide a robust and functional solution to control the block execution.

This process describes the PLC control architecture shown in the physical layer of Figure 4.1.

With the proper enhancements the usage of high level control algorithms and technologies can be used. Nevertheless, a method to link the PLC control architecture with the high level control architectures is then required.

4.3.2 Interaction between Layers

An accurate method to make the interaction between the PLC and the high level control architecture is required. This method should be able to provide the means so that values in the PLC can be changed and accessed by the high level control architecture in a clear and transparent way. The high level control architecture will then be able to actuate the control variables, triggering the execution of the modular assembly capabilities.

PLCs start to provide interface solutions that enable the integration of PLCs with higher level programming languages. One of these solutions is Automation Device Specification (ADS) interface provided by Beckhoff [31]. The ADS interface is the basis for the creation of this hybrid control architecture, since it provides the link between the physical and logical layers with minor impact on the PLC performance. The ADS will then provide an open communication flow between the PLC and the high level control architecture. The ADS interface provides the means to interface with PLC code or directly with the I/O's. This interaction overview can be seen in Figure 4.4.

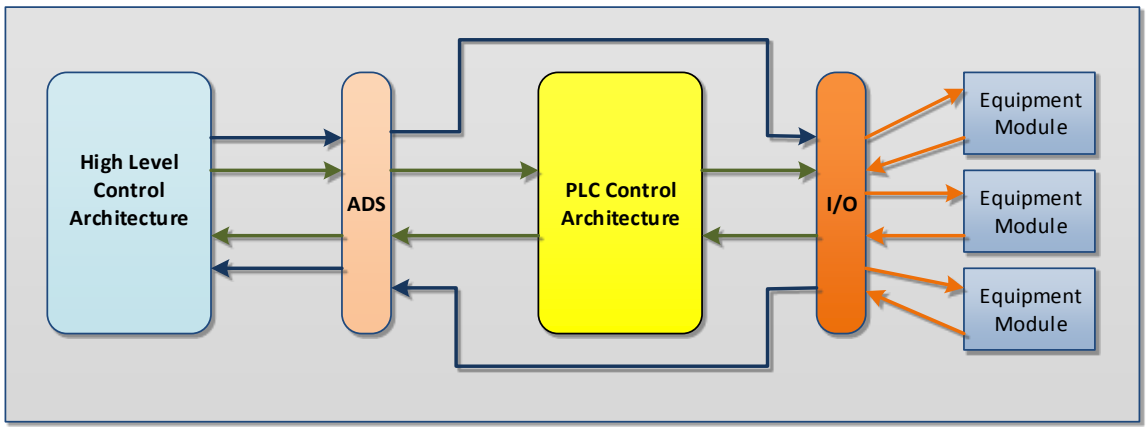

Figure 4.4 - ADS functionality overview

Using the ADS, the high level control could actuate the control variables defined previously, in order to transfer the assembly capability execution to the PLC. The ADS provides call back methods that trigger high level programs based on changes on any parameters tracked in the PLC, e.g. inputs and outputs but also internal variables. This functionality provides the means for an architecture to work without the need to constantly monitor I/Os, which would lead to a significant increase in the overall performance.

The ADS will provide the required layers independent interaction method allowing the needed modularity. It will allow the high-level control architecture to properly communicate with the PLC and vice versa.

4.3.3 Hybrid Control Architecture Description

High level control architectures are mainly implemented using object oriented programming languages. These will allow other technologies to be used to perform the control logic. As seen in the literature, agent technology can provide autonomous behaviours modelled to actuate accordingly to the entity's they represent. Also, agents have a modular structure ideal for this type of required control solutions. For the high level control architectures to start an assembly capability execution, using the proposed PLC control model defined in chapter 4.3.1, it just needs to actuate the variable which is part of the PLC extension. Using the call back methods provided by the ADS will then provide an optimal solution to know the assembly capability execution termination. An overview of this mechanism can be seen in Figure 4.5.

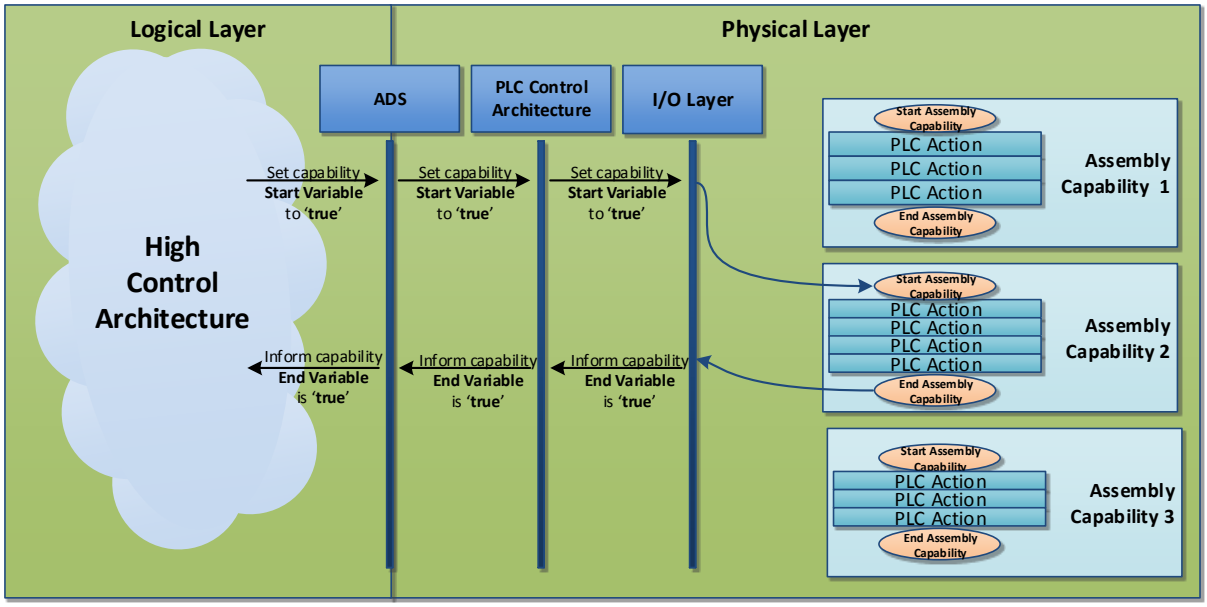

Figure 4.5 - Hybrid control architecture overview

For the high control logic to execute an assembly capability, it will use the ADS to actuate the right variable in the PLC. The PLC will then perform all the actions present in that block, obtaining the wanted assembly capability executed.

When the PLC terminates all the block execution actions, the block end variable is changed. This will trigger the method in the ADS to inform the high control architecture of this event. Only one agent must be responsible for an assembly capability execution, it is then its responsibility to correctly trigger its execution.

Having the control variables for each block known in advance by the high control architecture, it can then execute each PLC block individually, just by actuating the right set of variables. With this process, several different blocks can then be actuated simultaneously providing the means to obtain the parallel and independent module execution wanted for the MAS control.

4.4 Chapter Summary

In this chapter, first the requirements for a hybrid modular assembly system control were defined. A hybrid control architecture was then proposed using the advantages of both PLC and agent control architectures.

The proposed hybrid architecture allows both PLC and agent parallel control to be done. The assembly capabilities execution control is made by the PLC while the complex control decisions are being made by the agent architecture. Complex control logics can then be created and deployed regardless of the assembly capabilities execution procedures.

For a MAS control implementation, an agent can be responsible for a hardware module. This agent will have the previous model embed in its structure, having the required set of variables for the assembly capabilities it will execute in its description.

5 Redundant and Decentralized Capability Dissemination Agent

5.1 Introduction

Modular Assembly Systems (MAS) concept will allow a countless number of assembly modules to co-exist in the same system. Each of them will have their own assembly capabilities with their own different characteristics to execute them. Control architectures to take advantage of MAS are being studied and developed [48]. Agent technology is seen as a capable option to obtain them. This technology provides the needed dynamic control capabilities needed for the MAS. Along with that, ways to clearly represent each assembly capability inside the control agent architectures is also been subject to research and analysis [28].

A method to keep track, organize and disseminate the existing assembly capabilities through the MAS control agent environment is then essential to achieve optimal system functionality and performance. This method will provide the information about all the existing system assembly capabilities to all the agents in the environment.

To achieve that, proper information maintenance mechanisms need to exist in the control architecture. One solution to accomplish this is a dedicated repository, where the information about all the available assembly capabilities can be stored, grouped and organized, such as the JADE platform Directory Facilitator [68]. The DF is part of the JADE platform, being a centralized service and having some limitations regarding the exchanged message content may not provide the optimal solution for a MAS. Having this centralized service may bring issues in performance when system scalability is needed. Also, when a considerable amount of requests are being made bottlenecks may appear. This will constitute a severe and harmful problem to the overall MAS performance, as delays on the repository response will mean delays on the executions of the assembly processes. Another issue is if this central service repository has problems or malfunctions the MAS will not work. If it needs to be shut down or restarted, the entire MAS will be stopped. These issues can be a significant problem for the success of MAS control architectures.

A Capability Dissemination Agent (CDA) can then be modelled to be an extension of the available DF, giving to other agents the same capability as the provided DF but however allowing a decentralized and redundant approach.

The assembly systems are hierarchy. These may be composed by assembly lines, which themselves are composed by cellules containing workstations. This hierarchy can then be explored in the creation of work clusters. A CDA can then be associated with each cluster, keeping track, organizing and informing the entire system of the cluster assembly capabilities. The CDA however independent will be connected between each other so that any CDA will have the knowledge of all the capabilities the system has. The CDA can provide a similar

functionality as a centralized information repository, however providing the redundancy and system reaction to changes needed by the Plug and Produce systems concept. Figure 5.1 gives an overview of the proposed CDA functionality.

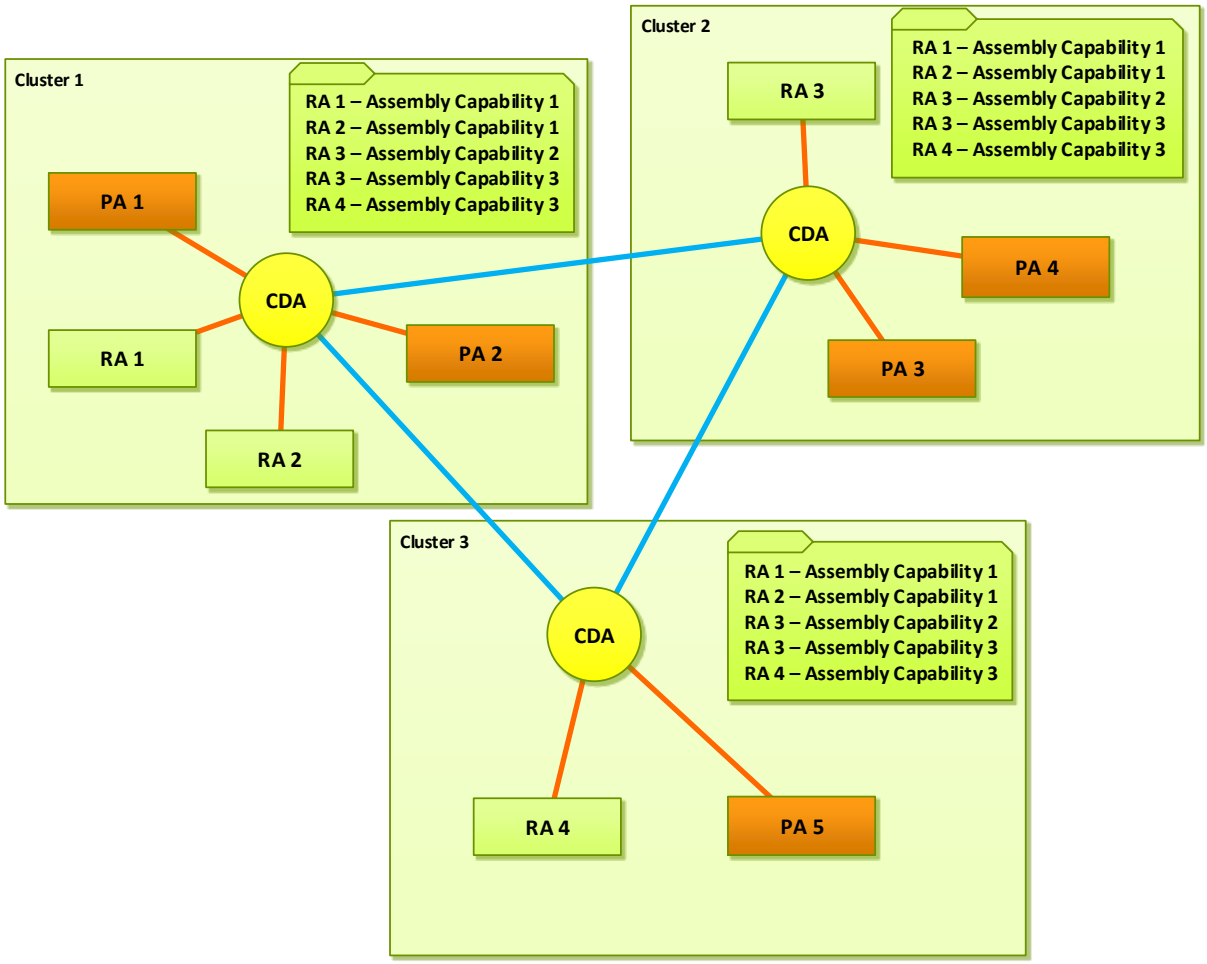

Figure 5.1 - CDA functionality overview

In the figure, the RA represents the agents that want to register assembly capabilities in the CDA while the PA represents the agents that needs to consult these capabilities. Three CDAs are connected between themselves. The CDA provides the service for RAs to register themselves and then put the registered information available for whatever agent that needs it. Each CDA is then required to interact with the agents within the cluster and also transmit the information about the modules capabilities to the other CDA. Each CDA will then have the information about the capabilities existing in other clusters, which allows it to have a good overview of the entire network. Also, Clusters of clusters can be created to allow superior system scalability. This chapter proposes a CDA model to be used by an agent control architectures that may bring benefits on the overall MAS performance.

5.2 Redundant and Delocalized Capability Dissemination Agent Requirements

A well-known and effective way of providing and organizing all the assembly capabilities available in a system is critical for enabling MAS. These assembly capabilities will need to be represented within the control environment. Using the Skill concept [28] will provide an established representation for them. A method for the CDA to store and broadcast these Skills to other CDAs is then required.

Due to the large amount of information that is expected to be stored, quick mechanisms to store the information and access it are needed. Only with those, performance targets can be achieved as a slow CDA response to requests will cause delays on the system functionality.

The CDA must provide a method for other CDA's and other agents in the system to know when a CDA is introduced or removed. This method requires to be independent of the platform where the CDA is launched as MAS allows multiple platforms to exist. Also, the means to acknowledge that modules are added or removed is also required. Only with those discovery mechanisms a coherent and updated Skill information can be maintained.

Due to the usage of agent technology, interactions between all the CDA and other agents must be defined. This would allow them to communicate with one another, allowing them to cooperate so that objectives can be obtained.

These requirements will provide the CDA the means for it to be properly modelled as well as taking advantage of all its characteristics.

5.3 Redundant and Delocalized Capability Dissemination Agent Model

In a MAS, the assembly capabilities are executed by the hardware modules present in the system. These modules are independent having no knowledge about other modules and their capabilities. If each individual module capabilities are propagated through the system to all its existing members, an enormous number of messages would be exchanged, which could cause undesirable system underperformance.

A place to gather and organize the distributed assembly capabilities available in the system is then required. The proposed CDA is then used as an assembly capability gathering point, organizing them and informing them through the system. A CDA will then have the information about the registered local capabilities. However, several CDAs can then be present. Each CDA must also know the local capabilities registered in all the other CDAs. This is obtained by regular information exchange between CDAs. Each CDA will then have the knowledge about all the assembly capabilities available throughout the system.

During the CDA life cycle, it needs to always maintain a correct and updated information about the entire system assembly capabilities. Only then can each CDA maintain an accurate system assembly capabilities information.

To establish the CDA model, one needs to know what interactions are needed. The need to clearly enumerate each of the available relations and interactions arises so that its behaviours can be defined. Also, for this model, the Skill concept will be used to represent the assembly capabilities. The interactions that may occur with the CDA can be seen in Figure 5.2.

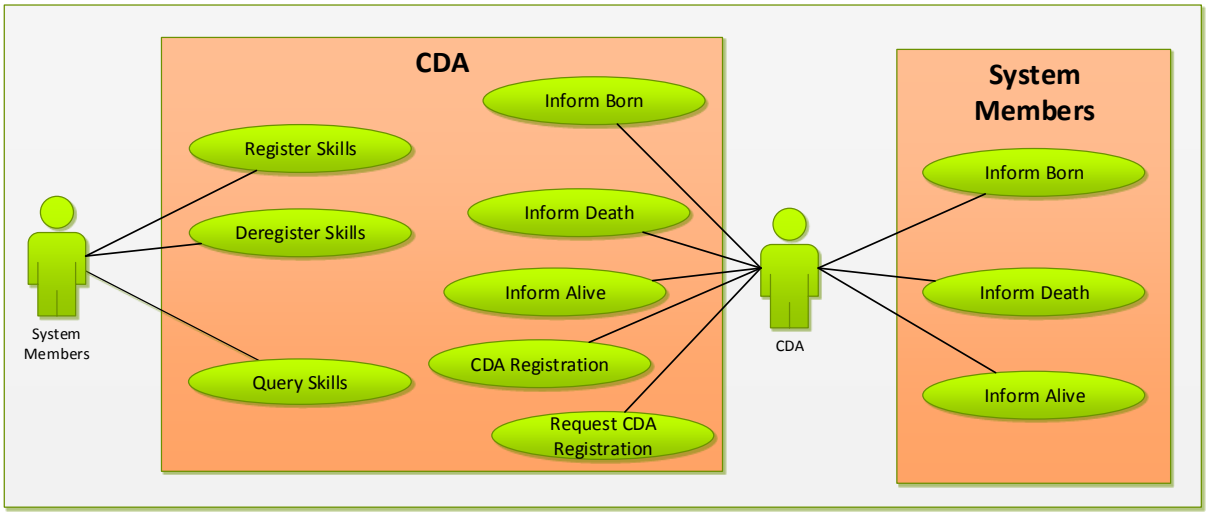

Figure 5.2 - CDA use cases overview

Any agent in the environment can register or deregister Skills within the CDA. Queries about existing Skills providers can also be made. The CDA will broadcast a message to the system informing it is born when he is first launched and a death message when he leaves the system. Periodic messages regarding the internal information status of the CDA will also be broadcasted.

These define the possible interactions with the CDA. Its analysis will provide all the necessary protocols that need to exist. These protocols are the basis for creating an agent behaviour model which uses the protocols to achieve the agent objectives. Only with a clear and structured agent behaviour model will it be possible to create, implement and run the intended CDA. The first step to obtain the wanted model is to define the internal information each CDA needs to hold.

5.3.1 Capability Dissemination Agent (CDA) Internal Data Model

The CDA purpose will be to provide a delocalized and redundant Skill repository in an agent control environment. For that, several CDAs may exist. To identify unambiguously each one is therefore necessary. Also, to fulfil its objectives, information needs to be stored in the CDA. First it is necessary to specify how the information will be organized in the CDA. Each CDA will have its local information, as well as the information about the Skills in the system, their Providers and also all other existing CDA neighbours. A representation of the CDA internal data model can be seen in Figure 5.3.

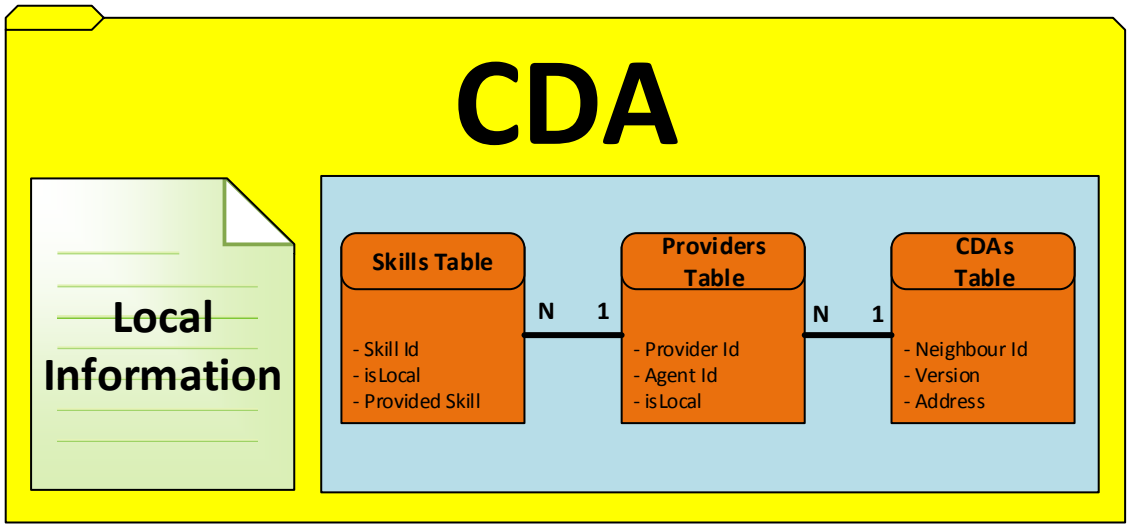

Figure 5.3 - CDA internal information organizational model

The **Local Information** will contain the information about the CDA. The **CDAs Table** will contain the information about all the CDA present in the system, even local CDA itself. Each will then possess registered Skill providers. These Skill providers, the locals and the ones registered in other CDAs, will be stored in **Providers Table**. Also, each Skill provider may have several Skills. All the Skills available in the system will then be stored in **Skills Table**. The **Local Information** fields can be seen in Table 5.1.

Table 5.1 - CDA Local Information content definition

Local Information		
Attribute	**Description**	**Type**
CDA Id	Unique name of the CDA	String
Address	Used to store the CDA IP location within the system network.	String
Version	Identification of the information stored in the CDA.	Long
Broadcast Port	Port to be used in the broadcasted messages.	Integer

A unique identification for each CDA is required. This will be stored in the **CDA Id.** The CDA will need to know what its system location is. This will be used by other agents in the system to contact him when necessary. This will be stored in the **Address** variable. The nature of MAS is highly changeable so frequent Skills changes are expected. The existing Skills may not remain fix throw-out the system life cycle and for that reason a method to keep track of the information status is then needed. The **Version** variable will be used to address this by using an incremental approach to the variable. The **Version** starts at 0 and will be incremented every time a change in the internal CDA information is made. The **Broadcast Port** will store the information about the port to listen and to send the broadcasted messages.

The available system assembly capabilities need to be stored within the CDA so they can then be consulted by any other agent in the system. The CDA is then expected to store a great amount of information while ensuring the best performance on its store and retrieve. This is vital as a slow response from the repository will harm the overall system performance. The information about the CDAs present in the system will be stored in **CDAs Table.** The **CDAs Table** fields can be seen in Table 5.2.

Table 5.2 - CDA Neighbours Table fields definition

CDAs Table		
Attribute	**Description**	**Type**
(PK) Neighbour Id	Unique Name of the CDA neighbour.	String
Version	To control the information status of the CDA.	Long
Address	Network system location where the CDA is.	String

The **Neighbour Id** is the table primary key. It is the CDA neighbour name which identifies him. The **Version** is the variable used to know the internal information version of the CDA neighbour. The **Address** will store the information about the system location of the neighbour CDA.

Having how each CDA neighbour will be stored; the need to store each of the Skill providers in the system is also needed. Each provider available in the system will have an entry in the **Providers Table**. Each entry will then be associated with a **Neighbours Table** entry. The **Providers Table** fields can be seen in Table 5.3.

Table 5.3 - CDA Providers Table fields definition

Providers Table		
Attribute	**Description**	**Type**
(PK) Provider Id	Unique name value for the internal agent provider identification.	String
AgentId	Agent identification associated with the entry.	AID
isLocal	Information about the location where the provider is registered in. - 'true' - if the provider is registered locally; - 'false' - if the provider is registered in another YPR agent.	Boolean

The **Provider Id** is the table primary key. It will be the Provider agent name. The **Agent Id** will contain the agent identification of the provider. This will be used to contact the provider, or given to others so they can contact it. The **isLocal** field will be used to control if the provider is local (directly registered in the CDA) or remote (registered in a CDA neighbour).

Each of the Skill Providers registered in the Providers Table can have several Skills registered in the CDA. These Skills will be stored in the **Skills Table**. Each entry in this table

needs to be associated with an entry in the **Providers Table**. The **Skill Table** fields can be seen in Table 5.4.

Table 5.4 - CDA Skills Table fields definition

Skills Table		
Attribute	**Description**	**Type**
(PK) Skill Id	Unique Name of the Skill Type the entry is associated with.	String
Skill Type	Name of the Skill type the entry is associated with.	String
Provided Skill	Skill instance information associated with the entry.	Skill Class

The **Skill Id** will be the table primary key. It is used to identify each individual skill. The **Skill Type** will be used to identify the type of the Skill the entry is related with. The **Provided Skill** will be the Skill instance type registered in in the CDA.

The CDA model will be made using the next defined method. Yet, other approaches could be used for the instantiation of this model. Each of the CDA tables' content will be an instance that will contain the needed information. The three tables will then be created using three hashtables <key : value>. Each hashtable key will be the (PK) Id associated with each entry. The value will be the instance object itself representing the entry.

To interlink each table, tree extra hashtables will be created. The Available Skills hashtable <Skill Id : Providers Array Id> will be formed by the (PK)Skill Id as the hashtable key and an array containing all the (PK)Providers Id that can perform that Skill as its value. The Available Providers <Provider Id : Skills Id Array> formed by the (PK)Providers Id as the hashtable key and the array containing the (PK) Skills Id as value. And the Available Neighbours hashtable <Neighbour Id : Providers Id Array> formed by the (PK) Neighbours Id and the array with all the Providers Id registered with it as value.

Each time there is to update the tables; the respective hashtables will be checked, the needed data obtained and the information updated accordingly. This storage method can provide easy and considerably fast access to the stored information as hashtables have been identified in computer science has a considerable fast approach to retrieve information.

Having the information structure for the CDA model and how to used established it is now possible to establish the desired interactions for the correct agent functionality.

5.3.2 Capability Dissemination Agent (CDA) Interactions

All the interactions that may occur with the CDA need to be defined so that the proposed solution is stable and scalable. These interactions are mainly made using agent communication mechanisms which will be defined based on FIPA [63]. As seen in the literature review, FIPA covers generic agent interactions which will significantly facilitate the development of the communication methods. For each interaction to be properly made, protocols need to be established. These protocols will provide the means for each member in the environment to interact with the CDA.

FIPA establishes in its message contents the necessary information for a reply to be sent to the specific agent. The remaining message content is specific to each message and is defined accordingly.

5.3.2.1 Capability Dissemination Agent (CDA) Born Protocol

One requirement is to provide the means for the CDA to communicate with any other agents in the system. To obtain that, a message will be broadcasted to the network. This message will also be based on FIPA as it's a packet, however it will be sent using a method external to the normal agent communication mechanisms. To reach a remote agent platform that may exist in the system, a UDP packet [71] is sent. This packed is formed by the destination ip address (which can then provide the address for a reply to be sent), the destination port (which is the **Broadcast Port** defined before), the content size and the content itself. This message is to be received by all the others CDAs as well as existing agents that may need to know about a CDA presence. For that they need to be listen to the **Broadcast Port** so that these broadcasted messages can be received. When a CDA is introduced in the system, it will use the previous defined method to broadcast a message. This messages will be sent using the **CDA Born protocol** defined in Figure 5.4.

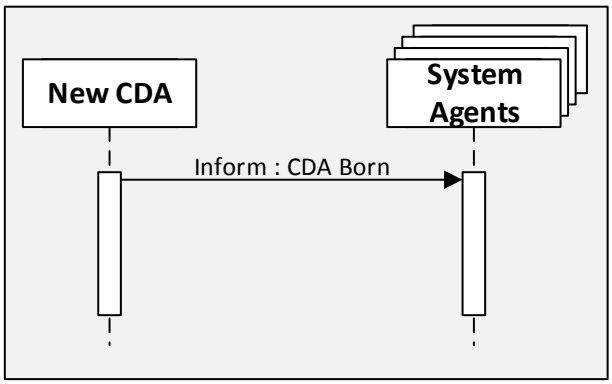

Figure 5.4 - CDA Born Protocol message exchange definition

This protocol will consist in a broadcasted **Inform : CDA Born** message to the network. The message content is the CDA **Local Information** so that any other agent can use it to communicate with the message CDA sender.

5.3.2.2 Capability Dissemination Agent (CDA) Registration Protocol

When a new CDA is introduced into the system, a message is broadcasted using the protocol defined before. When this message is received by another CDA it will need to inform the new born agent that it is his neighbour. This is obtained using the **CDA Registration Protocol**. This protocol can be seen in Figure 5.5.

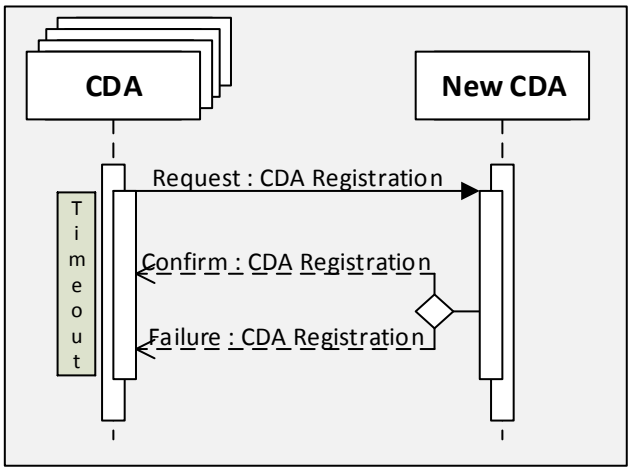

Figure 5.5 - CDA Registration Protocol message exchange definition

A **Request : CDA Registration** containing the CDA **Local Information** will then be sent triggering the neighbour registration to be made. Also, an array containing all the local registered skill with their providers will be sent. These array will contain the information contained in the **Skills Table**, the **Providers Table** and the **CDAs Table**. These tables are the same defined in Capability Dissemination Agent (CDA) Internal Data Model chapter. If the receiver already have the information contained in the tables, it will discard it. Otherwise, the receiver tables will be updated accordingly.

This message will have a "**Confirm : CDA Registration**" reply message confirming the registration or a "**Failure : CDA Registration**" reply message if the registration fails.

5.3.2.3 Capability Dissemination Agent (CDA) Request Registration Protocol

The MAS concept is designed for constantly changing environments. The need for a CDA to register itself in another CDA will arise in such a dynamic environment. This can occur due to the possibility of a package being lost, or a message from a CDA that is not present in the **CDAs Table** received. The need to ask for its registration is then needed. This is obtained using the **CDA Request Registration Protocol**. This protocol can be seen in Figure 5.6.

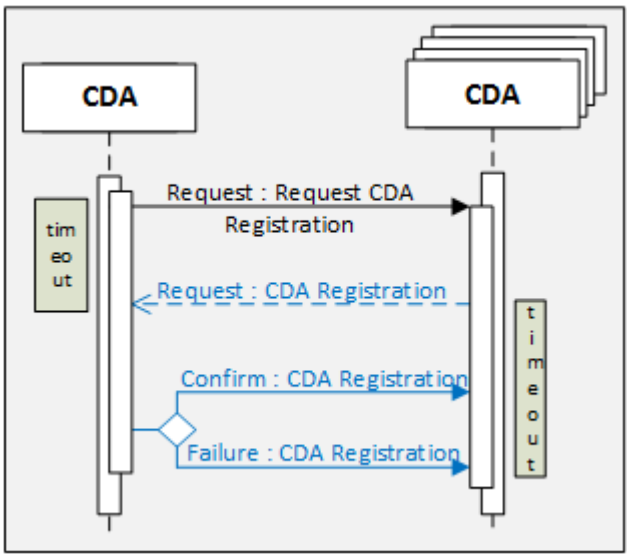

Figure 5.6 - CDA Request Registration Protocol message exchange definition

To request another CDA registration, a **Request : Request CDA Registration** message will be sent to the CDA. After this message, the receiver CDA will use the **CDA Registration Protocol** defined in chapter 5.3.2.2 to finish this protocol and register itself in the requester CDA.

5.3.2.4 Capability Dissemination Agent (CDA) Status Protocol

During a CDA life cycle periodic messages will be exchanged between all the CDA in the system. This will allow them to know about each other internal information status as well as their existence. This is made using the **CDA Status Protocol**. The protocol can be seen in Figure 5.7.

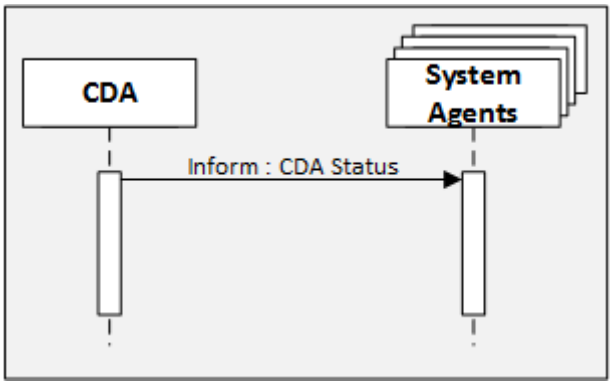

Figure 5.7- CDA Status Protocol message exchange definition

A **Inform : CDA Status** message is broadcasted through the system. As in the **CDA Born Protocol**, the broadcasted message will contain the CDA **Local Information** and have the same characteristics of the defined UDP packet. This protocol will provide the capability for all the CDAs in the system to keep synchronized with each other's having their local registered

assembly capabilities updated. This is accomplished analysing the CDA internal information **Version** present in the message content.

The need to request for the internal information status may arise, this is made using the **CDA Request Status Protocol**.

5.3.2.5 Capability Dissemination Agent (CDA) Request Status Protocol

When for some reason a CDA needs to know other CDA internal information, it will use the **CDA Request Status Protocol** to obtain it. This protocol can be seen in Figure 5.8.

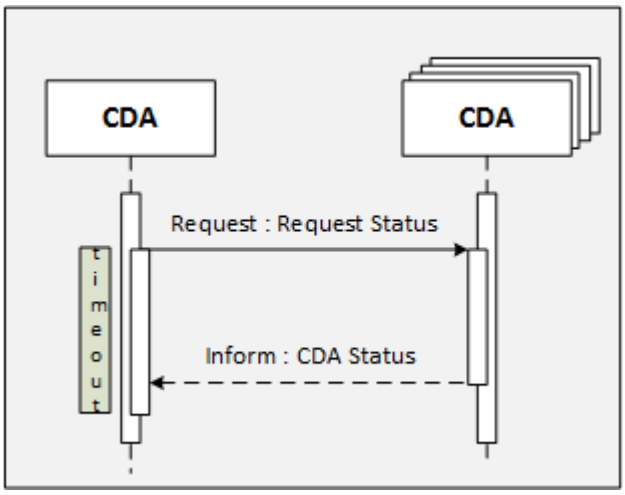

Figure 5.8 - CDA Request Status Protocol message exchange definition

A **Request : Request Status** message is sent requesting the CDA internal information which will then be replied with a message containing the CDA **Local Information**. This protocol can be used to confirm that another CDA is still present in the system, or to confirm if the current information stored regarding a specific CDA is still up to date or not.

5.3.2.6 Capability Dissemination Agent (CDA) Termination Protocol

When a CDA has a controlled exit of the system, it will broadcast a message informing it. This obtained using the **CDA Termination Protocol**. This protocol can be seen in Figure 5.9.

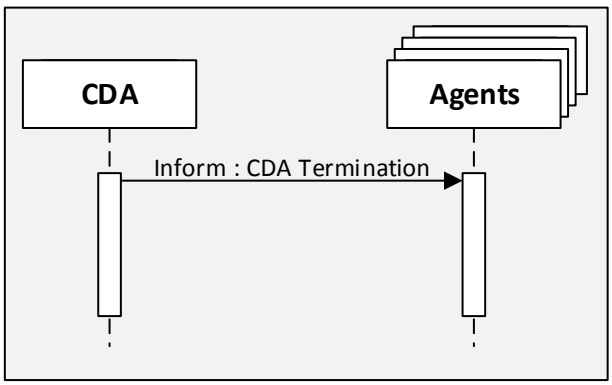

Figure 5.9 - CDA Termination Protocol message exchange definition

A message containing the CDA **Local Information** will be broadcast to the system. This message will have the same characteristics as the **CDA Born** message defined in the **CDA Born Protocol**. This message will be used by CDAs and other agents in the system to know that a specific CDA is no longer available in the system.

5.3.2.7 Skill Registration Protocol

When an agent knows how to execute some Skills and wants other agents know about it, it will register these Skills in the local CDA. This is made using the **Skill Registration Protocol**. This protocol can be seen in Figure 5.10.

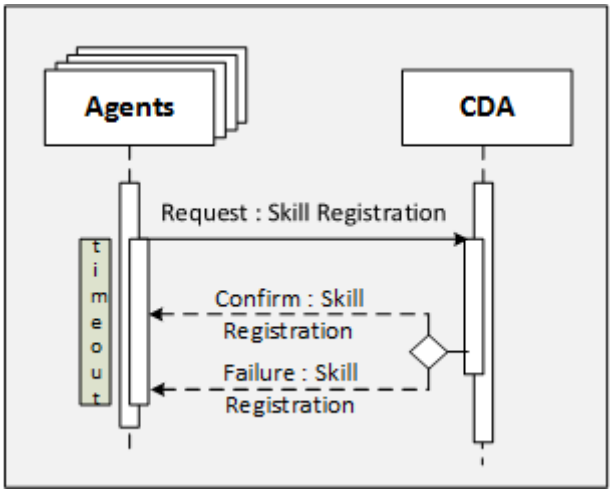

Figure 5.10 - Skill Registration Protocol message exchange definition

The agent who wants to register the Skills will sent a **Request : Skill Registration** message containing all the **Provided Skills** to be registered to the CDA. When this message is received by the CDA, it will send **Confirm : Skill Registration** message informing the member the Skills were successfully registered. On the other hand if an error occurs, a **Failure : Skill Registration** containing all the Skills whose registration failed will be sent.

Having the **Skill Registration Protocol** defined to register Skills in the CDA, the need for deregistration arises.

5.3.2.8 Skill Deregistration Protocol

The Skill deregistration is obtained using the **Skill Deregistration Protocol**. The **Skill Deregistration Protocol** is similar to the previous one, changing the messages headers exchanged. This protocol can be seen in Figure 5.11.

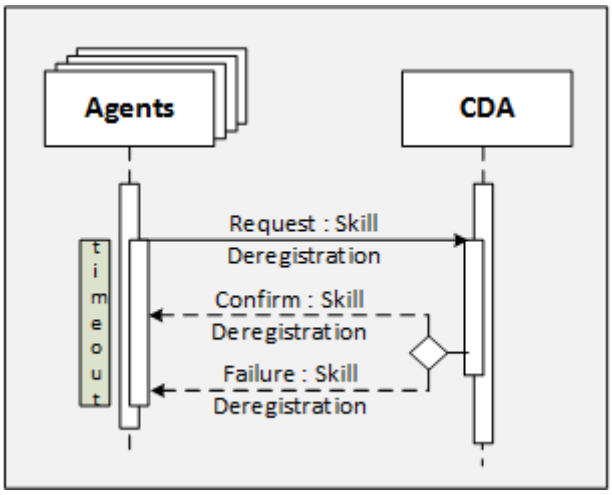

Figure 5.11 - Skill Deregistration Protocol message exchange definition

An agent will send a **Request : Skill Deregistration** message to the CDA with all the Skills to be deregister. The CDA will then respond with a **Confirm : Skill Deregistration** if the Skills were deregistered without errors, or, with a **Failure : Skill Deregistration** message with the failed Skills deregistration if errors occurred.

With the method to add and remove Skills from the CDA, a protocol to query for them is then needed.

5.3.2.9 Skill Query Protocol

The **Skill Query Protocol** will provide the means for the system available Skills to be queried. This protocol can be seen in Figure 5.2.

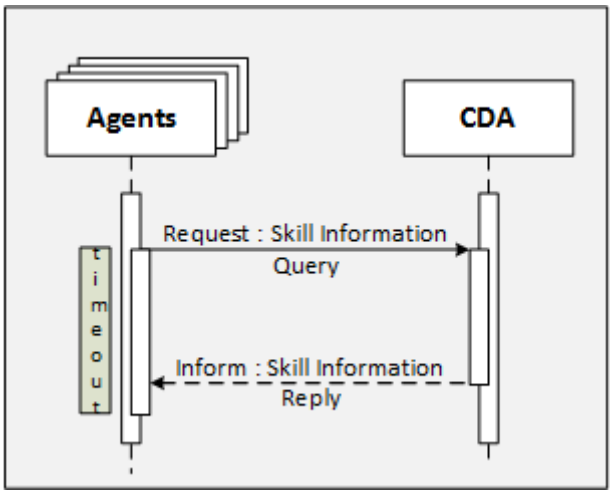

Figure 5.12 - Skill Query Protocol message exchange definition

When a member needs to know which agent can perform a Skill, it will send a **Request : Skill Query** message to the CDA containing the Skill Type name of the wanted Skill, and, if it wants local providers or ones from the entire system. The CDA will then reply with a **Inform : Skill Information Reply** message containing an array with the **AID** of all the local (and remote depending of the request) agents that know how to perform the queried skill, or an empty array if no agent to perform that Skill is registered. Also, if a requested Skill is null, an entire snapshot of the system will be replied. This will be formed by the **Skills Table** and the **Providers Table** the CDA has.

These protocols need to be respected so that proper interactions can be achieved. With those established, the CDA behaviours can be defined to implement them.

5.3.3 Capability Dissemination Agent (CDA) Behaviours Definition

Having the internal information structure defined, and also how each CDA will interact with other CDAs and other members in the system, it is now possible to create the behaviours models to obtain the wanted CDA objectives.

The CDA requires a mechanism to broadcast messages to the system. Its behaviours can then take advantage of this functionality to obtain its purposes. Each CDA is wanted to be functioning independently from each other. Each CDA will be associated with a different main container so that if one fails, the others can still keep functioning. A CDA registered in a JADE main container [68] cannot send messages to agents registered in others without previously know their individual address and names.

A mechanism so that a CDA can broadcast messages to agents in other containers is defined. This will be obtained using the Ethernet UDP Protocol [71]. It will provide a good performance approach as it does not requires connections to be established and maintained through the system, and besides the UDP protocol not being 100% reliable, its usage within a controlled environment will prove itself to be robust enough. For this UDP broadcast to be functional, all the system members will be part of the same network as well as known the **Group Port** that the messages are sent and received through. All the agents that want to receive these messages must be listening to this defined port. Only with those prerequisites will allow that each message sent this way can be correctly received. This will allow the wanted message to conceptually reach every agent in the system, present in remote containers or in the local container.

5.3.3.1 Receive Messages Behaviours

Each CDA will need to receive messages to accomplish its objectives. These messages can be sent from the same agent platform where the CDA is, or from remote platforms. For it to receive both messages, two behaviours will be defined. One will be receiving messages sent using the Broadcast method, and for it, a UDP client will be launched always listening to the **Broadcast Port**. This will allow it to know when a new message is available, to receive it and

process it. The other behaviour will receive the messages sent within the same agent main container platform. These behaviours will be fundamental as each received message will trigger the execution of other behaviours allowing then the CDA to fulfil its objectives.

5.3.3.2 Capability Dissemination Agent (CDA) Born Behaviour

When a CDA is launched in an agent environment, it will use the broadcast behaviour to send a **CDA Born** message to all the agents in the network informing it is operational. This is done using the **CDA Born Protocol**. When the **CDA Born** message is received by another CDA, it will react to it accordingly to the behaviour algorithm shown in Figure 5.13.

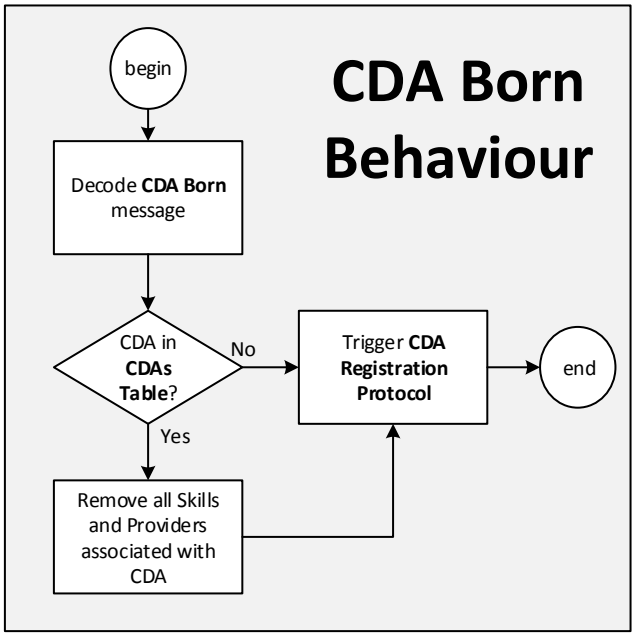

Figure 5.13 - CDA Born Behaviour algorithm

When this behaviour is launched, a **CDA Born** message will be passed to it. It will first decode the message obtaining the CDA **Local Information** the message contains. It will then check the **Neighbours Table** for the presence of the CDA neighbour. If it is present, it means that some error occurred on the sender side. The information stored about it will no longer be up to date. The **Skills** along with the **Providers** entries associated with it will be removed, and the **CDA Registration Protocol** initiated to register it as neighbour of the born CDA. On the other end if no information about the new CDA is present, the **CDA Registration Protocol** is immediately initiated.

5.3.3.3 Capability Dissemination Agent (CDA) Neighbour Registration Behaviour

When a new CDA joins the system, or if for some reason there is no knowledge about another CDA present in the system, the **CDA Registration Protocol** will be used to exchange information about the CDAs local information. When a **CDA Registration** message is received,

the **CDA Neighbour Registration Behaviour** will be launched. The algorithm defining this behaviour is shown in Figure 5.14.

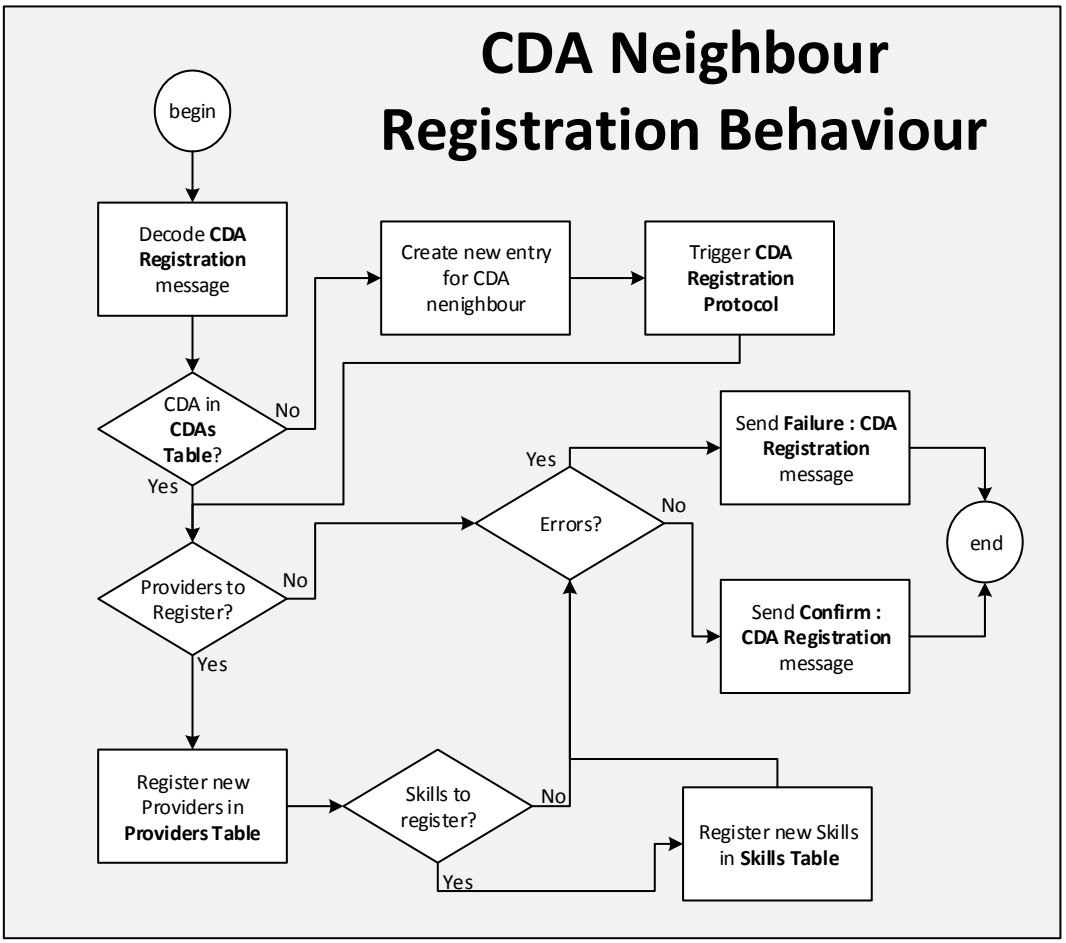

Figure 5.14 - CDA Neighbour Registration Behaviour algorithm

When this behaviour is launched, it will first decode the **CDA Registration** message obtaining the CDA **Local Information** along with its **Skills Table**, **Providers Table** and **Neighbours Table**. If the CDA is not yet on the local **Neighbours Table**, a new entry for it will be created and also the **CDA Registration Protocol** initiated. This will allow that both CDAs to have knowledge and information about each other.

After this process, if the received providers table is empty, there are no providers or skills to be registered. A **Confirm : Skill Registration** message or **Failure : Skill Registration** message is then sent depending if there were errors or not. Otherwise, there are entries to be processed. For each entry, if the **Agent Id** is not yet on the **Providers Table**, a new entry for it will be created. Also, if the **Skill** doesn't exists on the **Skills Table** a new entry for it will be created and the tables updated accordingly.

When there are no more entries to be processed, a reply message will be sent to the CDA. If no errors were detected a **Confirm : CDA Registration** message is sent, otherwise a **Failure :**

CDA Registration is sent. However, how to deal with errors goes beyond the scope of this work.

5.3.3.4 Capability Dissemination Agent (CDA) Status Behaviour

During the CDA life cycle, a **CDA Status** message will be broadcasted periodically. This message is defined in the **CDA Status Protocol.** This will allow that every CDA in the system to know about the other CDA internal information **Version.** Also, this will provide the means for each CDA to know if other CDAs are still present in the system. The behaviour algorithm is shown in Figure 5.15.

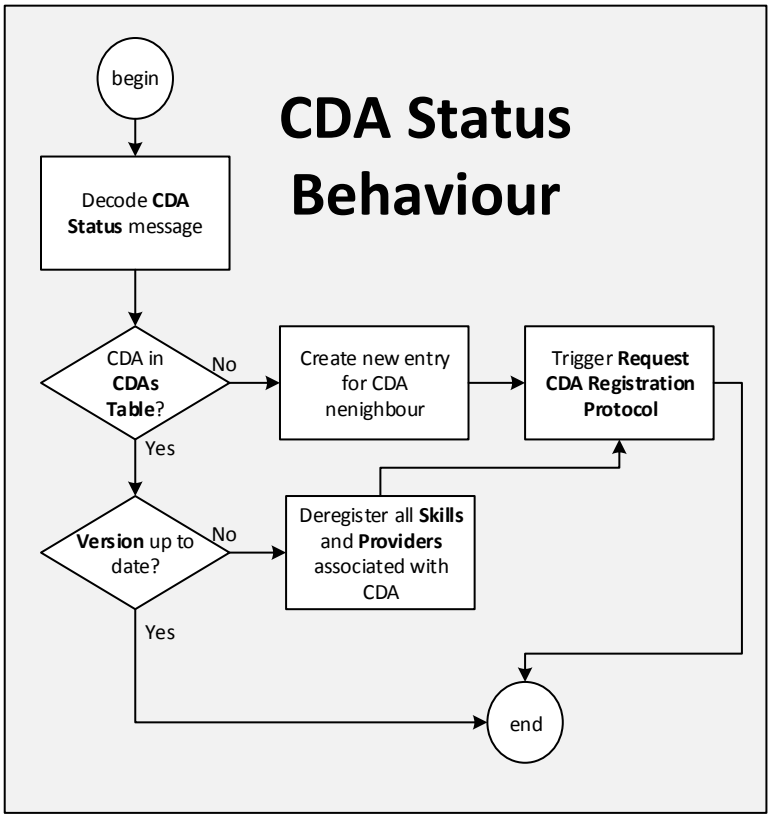

Figure 5.15 - CDA Status Behaviour algorithm

When this behaviour is launched, it will first decode the **CDA Status** message obtaining the CDA **Local Information**. If the CDA has not yet an entry in the **Neighbours Table**, a new entry will be created and the **Request CDA Registration Protocol** initiated. This will get the unknown CDA to register itself and send its internal information.

Otherwise, if the CDA is already known, the current CDA internal information **Version** which is in the message, will be compared with the stored on **Neighbours Table**. If it is higher, the neighbour CDA internal information has changed. Due to that, the content associated with the CDA will be updated in the **Providers** and **Skill Tables**, and the **Request CDA Registration Protocol** initiated for the correct information to be resent. On the other hand, if the **Version** is

the same or lower, the existing information about the CDA is up to date and no action will be made. The behaviour can then terminate.

This mechanism will provide the means for every CDA to have updated information about the registered Skills in all the other CDAs throughout the system. The need for a CDA to remove itself from another CDA arises.

5.3.3.5 Capability Dissemination Agent (CDA) Deregistration Behaviour

CDAs may need to be disconnected from the system, and broadcast a **Inform : CDA Termination** message. This message is defined in the **CDA Termination Protocol**. The need for its information to be removed from the internal tables must be provided. This is obtained using the **CDA Deregistration Behaviour**. Its algorithm can be seen in Figure 5.16.

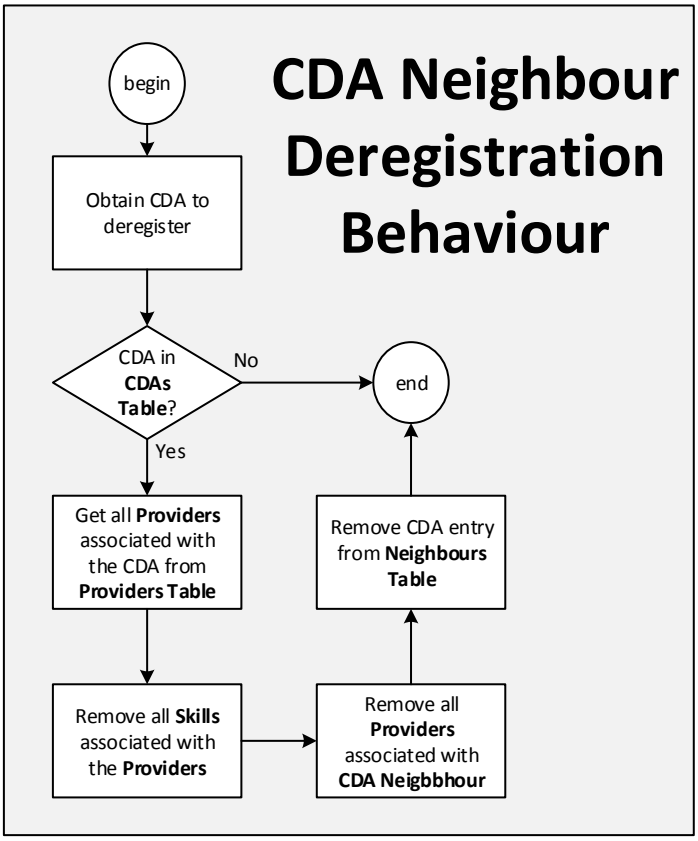

Figure 5.16 - CDA Deregistration Behaviour algorithm

The information about the CDA to deregister will be passed to the behaviour when it is launched. If the CDA is not in the **Neighbours Table**, no information associated with the CDA is present and no action will be made. However if it is present, its associated Provider agents will be obtained. All the Skills entries having those Providers will be removed from the **Skills Table**. All those providers will then be removed from the **Providers Table** and finally the CDA entry in the **Neighbours Table** deleted.

When a CDA ceases to exist in the system, the other CDA tables will be updated by removing all entries associated with it. However, a method to know when a CDA ceases to exist unexpectedly is needed. This is obtained using the **CDA Neighbours Still Active Acknowledgment Behaviour**.

5.3.3.6 Capability Dissemination Agent (CDA) Neighbour Still Active Acknowledgment Behaviour

A CDA may inform the entire system that it will no longer be in the system. It will do so using the **CDA Termination Protocol**. This will then immediately launch the **CDA Deregistration Behaviour**. However, for some reason this message might not be sent or even be lost. Each CDA will be broadcasting a **Inform : CDA Status** message. If this message for some reason is not received, it may indicate that the CDA is no longer in the system. A method to acknowledge this and launch the **CDA Deregistration Behaviours** is then required.

Having a periodic behaviour that will check if the CDA neighbours are sending their **CDA Status** messages could solve this issue. To obtain the behaviour desired objective, a **isAlive time period** needs to be established and related with the periodicity of the **CDA Status** message. This time period will be used to know when to analyse if the **CDA Status** messages have been received or not.

An **isAlive Table** will be created to store which CDA neighbour have send its **CDA Status** message. When a CDA Status message is received, its correspondent CDA will be added to the isAlive Table. A periodic check on this table will then be made accordingly to the established **isAlive time period** defined before. If any of the CDA neighbours present in the **CDAs Table** does not have an entry in the **isAlive Table**, it means the CDA neighbour has not sent its **CDA Status** message, and some problem may exist. This behaviour algorithm can be seen in Figure 5.17.

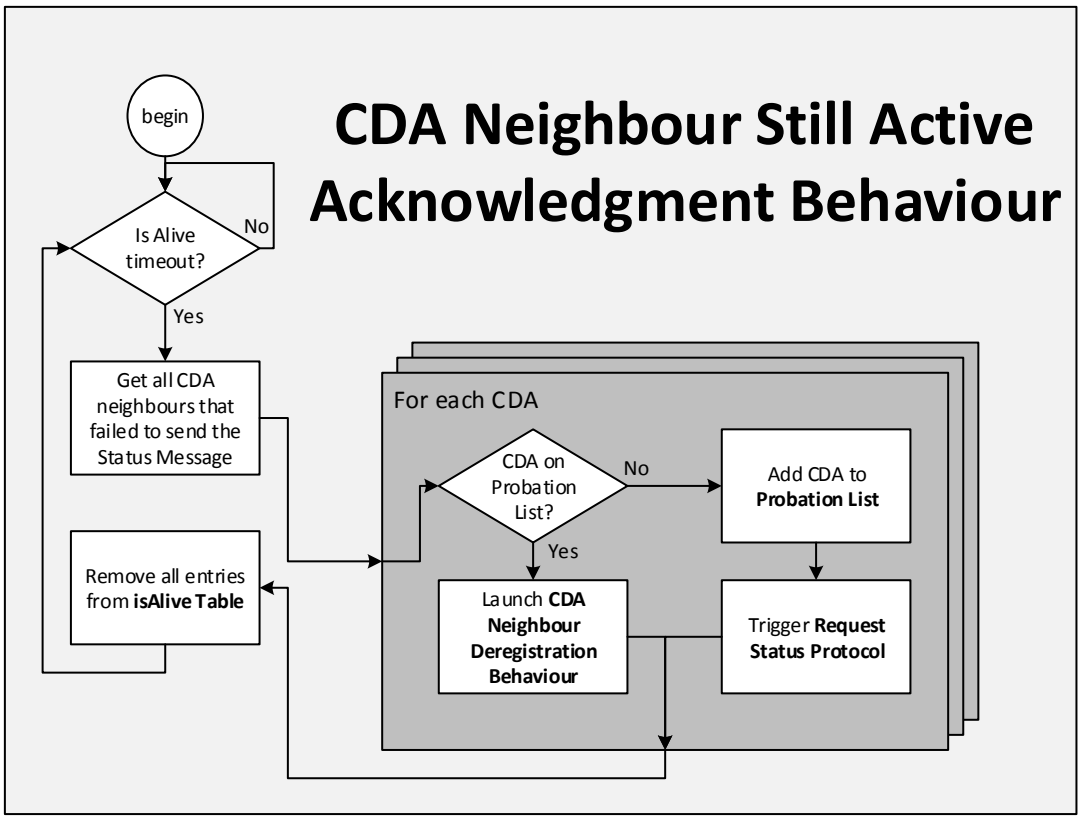

Figure 5.17 - CDA Neighbour Still Active Acknowledgment Behaviour algorithm

When the **isAlive time period** is reached, all the CDA neighbours whose status message were not received will be obtained. For each, a check will be made, if they are already on a **Probation List**, it means they failed to send the status message twice. The **CDA Neighbour Deregistration Behaviour** will then be launched for those CDAs. Else, if the CDA is not yet in the **Probation List**, it will be added and the **Request Status Protocol** initiated asking that specific CDA for its status message. When this message is received, if the CDA is on the probation list it will be removed. Else, if this message is not received twice in a row, following the behaviour logic the CDA will be deregistered.

Before the behaviour restarts, the **isAlive Table** is cleared so that the next interaction can properly function.

5.3.3.7 Skill Registration Behaviour

When an agent in the environment has the means to execute a Skill, it should inform all the other agents about it. Instead of informing them one by one, which will create a massive message flow through-out the system, it will use a CDA which will serve as a gathering point for all the Skills. For that, the agent will register all its available Skills in the CDA. This is made using the **Skill Registration Protocol**. The behaviour algorithm to deal with the Skill registration requests can be seen in Figure 5.18.

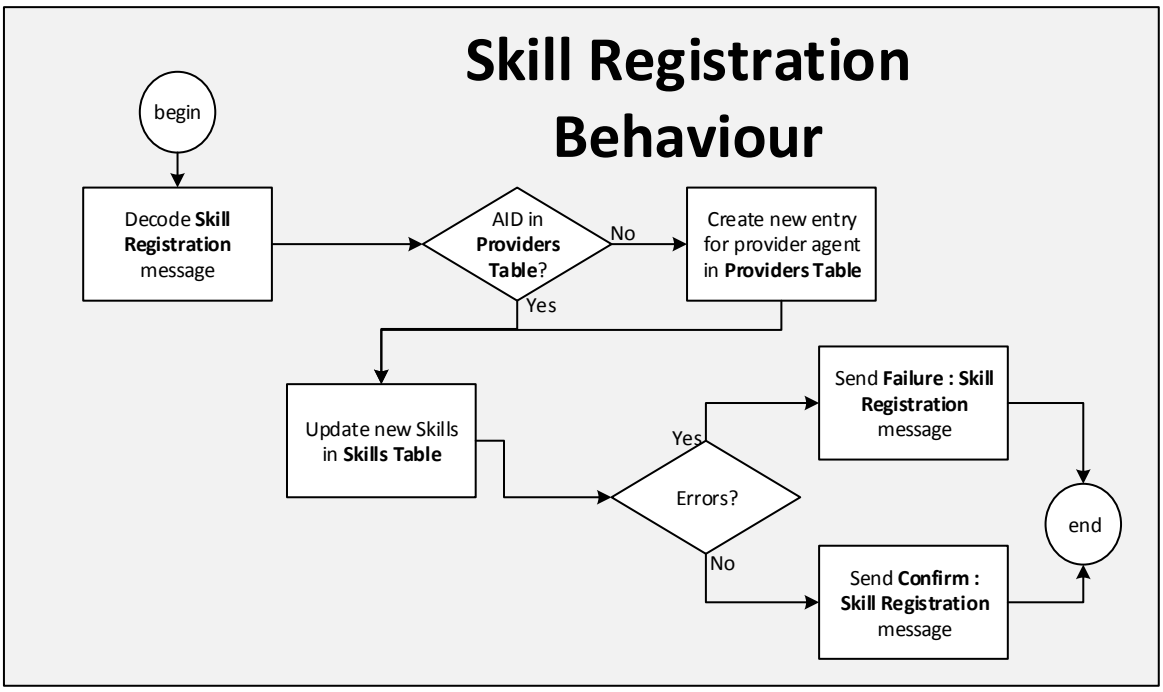

Figure 5.18 - Skill Registration Behaviour algorithm

The **Skill Registration** Message will first be decoded, obtaining the agent information along with the skills to register. If the Provider agent is not yet in the **Providers Table**, a new entry for it will be created. After that the **Skills Table** will be updated with the skills sent in the massage. Else, if the Skill provider is already in the **Providers Table**, the **Skill Table** will be updated by adding the new skills to it.

After this table update process, a **Confirm : Skill Registration** message is sent if no errors occurred, or a **Failure : Skill Registration** message with the Skills that failed to be registered.

This registration request will create a change on the CDA local information. This will create a new internal **Version**. A new **Status message** will immediately be broadcasted, triggering the actions defined in the **CDA Status Behaviour**. Eventually, the need to deregister capabilities may come, just one or even all of them if the Provider agent ceases to exist. This is obtained using the **Skill Deregistration Behaviour**.

5.3.3.8 Skill Deregistration Behaviour

When a Provider agent wants to deregister Skills from the CDA, it will follow the **Skill Deregistration Protocol**. The behaviour algorithm to deal with the skill deregistration can be seen in Figure 5.19.

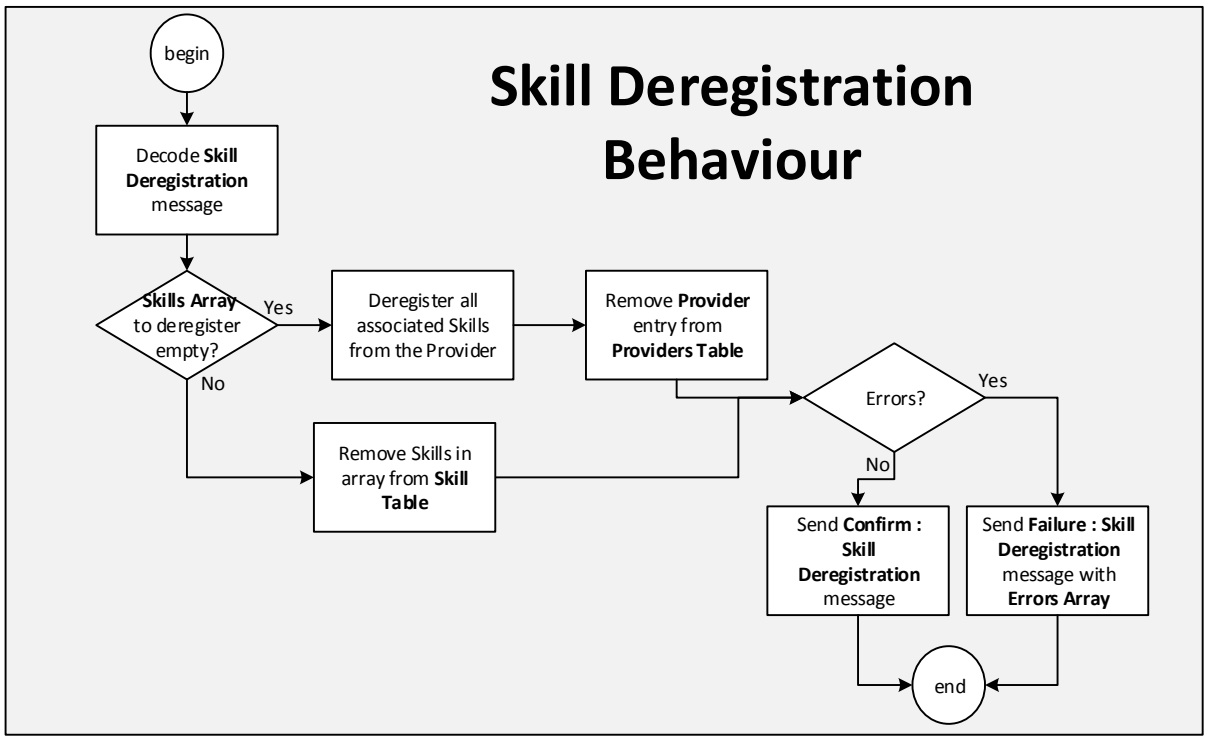

Figure 5.19 - Skill Deregistration Behaviour algorithm

The **Deregistration Message** will first be decoded, obtaining the agent information along with the **Skills Array** containing the **Provided Skills** to deregister. If the **Skills Array** is empty, it means that the **Provider Agent** wants to deregister all the skills and itself. Else, all the Skills to be deregister will be processed and removed from the **Skills Table**.

5.3.3.9 Skill Provider Unexpected Termination Behaviour

A Skill Provider may not be able to inform the CDA of its deregistration desire. This can occur due to errors or system malfunctions. Each CDA will then need to be able to know which providers with local registered Skills are still available in the system. Periodical messages can be sent to the registered providers querying them about their availability. If no response is received the agent will be considerate lost and the Skill Deregistration for it launched. This is crucial as it will prevent the CDA to have erroneous local information and disseminate it. Agent deaths in remote main containers can be acknowledged using this process. However, JADE platform has a mechanism to know when an agent in the same main container leaves the environment [68]. This can then be used to know when a local registered agent ceases to exist.

5.3.3.10 Skill Query Behaviour

One of the CDA service is to inform agents about which members can perform a specific Skill. This is obtained with the Skill Query Protocol. The required Skill Type will be transmitted to the CDA so it can properly respond to it. It is not the CDA objective to provide the optimal

member to fulfil a specific Skill as it has no information whatsoever about the requirements. This service will only provide all the members who can perform that Skill. Also, the means to obtain all the Skills present in the system will be made available using this service. This can be used by external processes to the environment to know information about the system capabilities along with its members' information. This will provide a snapshot of the system capabilities. This can be requested to any CDA as they will all the information about it. The proposed behaviour algorithm can be seen in Figure 5.20.

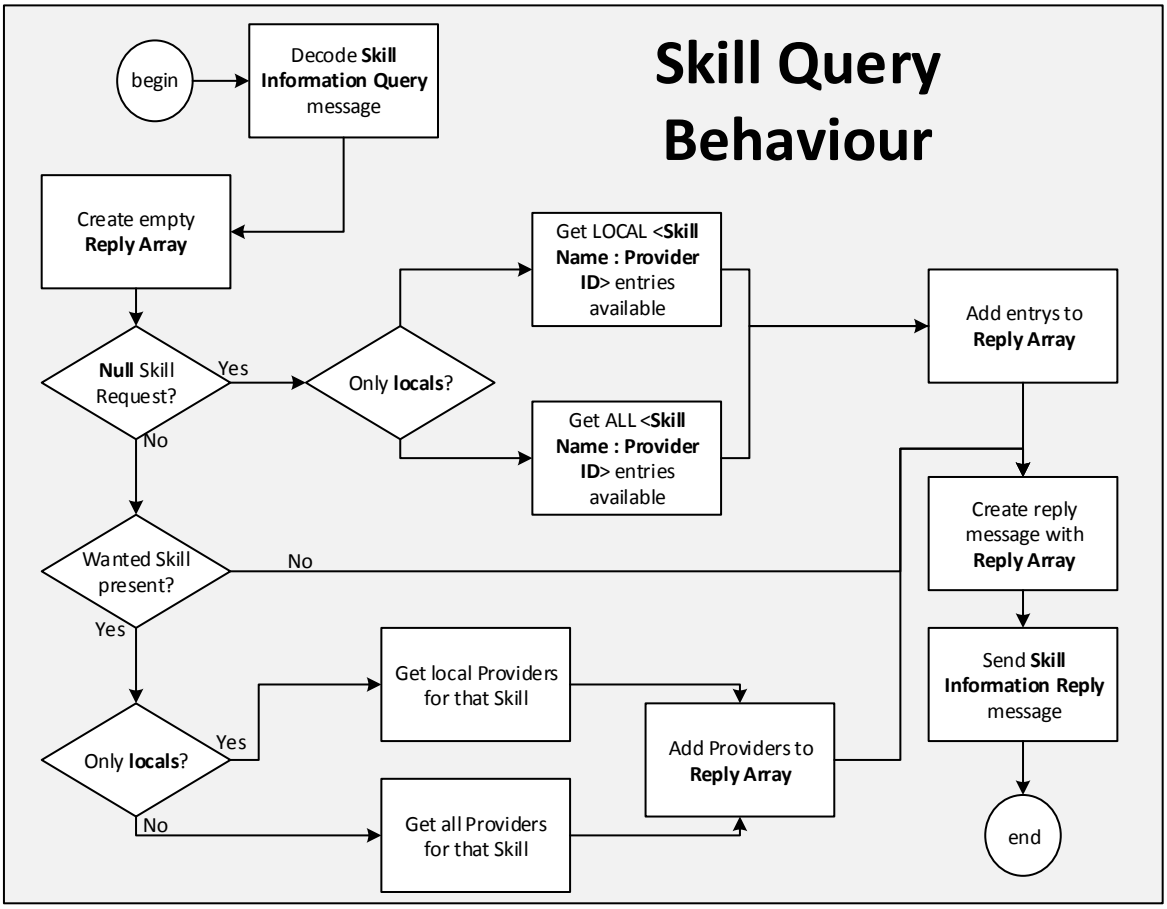

Figure 5.20 - Skill Query Behaviour algorithm

The **Skill Information Query** message will first be decoded, obtaining the wanted Skill Type name and the wanted Skill location. An empty **Reply Array** will be created to be sent in the reply message. If the wanted skill is 'null', a system snapshot is intended. The **Reply Array** will then be filled with < **Skill Name: Provider ID**> entries for all the existing Skills in the system, or just the locals according with the received request message. This entries will be filled with information taken from the **Skill Table** and **Providers Table.** If the wanted skill has no providers, an empty array will then be sent back to the requester. However, if the wanted Skill

exists in the **Skills Table**, all its **Provider Id** will be added to the **Reply Array** accordingly with the Skill location wanted.

These defined behaviours establish the methods for the correct CDA functionality to be obtained. The correct strategy for it to be deployed in the system is also required.

5.3.4 Capability Dissemination Agent Deployment Strategy

Having all the CDA functional behaviour defined, its correct deployment strategy remains. The number of existing agents along with their relations may help to define a system complexity. The more agents, the more difficult it is to control and track all of them as well as their capabilities. These capabilities can be at least as the number of agents in the system. If all these capabilities were organized and stored by the same agent, answering requests about them may bring system underperformance and undesirable bottlenecks.

A proper way to scale a system and facilitate the increasing number of messages avoiding the rising load problem is required for the optimal MAS functionality. This can be obtained by using a group of delocalized CDAs that exchanges information about existing system capabilities between themselves.

The CDA deployment is defined and adjusted considering the analysed system. On the limit, a CDA for each module can be created. However, this implies a massive communication load. For that, grouping agents' information and use the CDA to broadcast it can then provide an optimized solution. However if it's not well analysed and designed; its objective to reduce the system message exchange can be lost. An overview of the CDA deployment strategies can be seen in Figure 5.21.

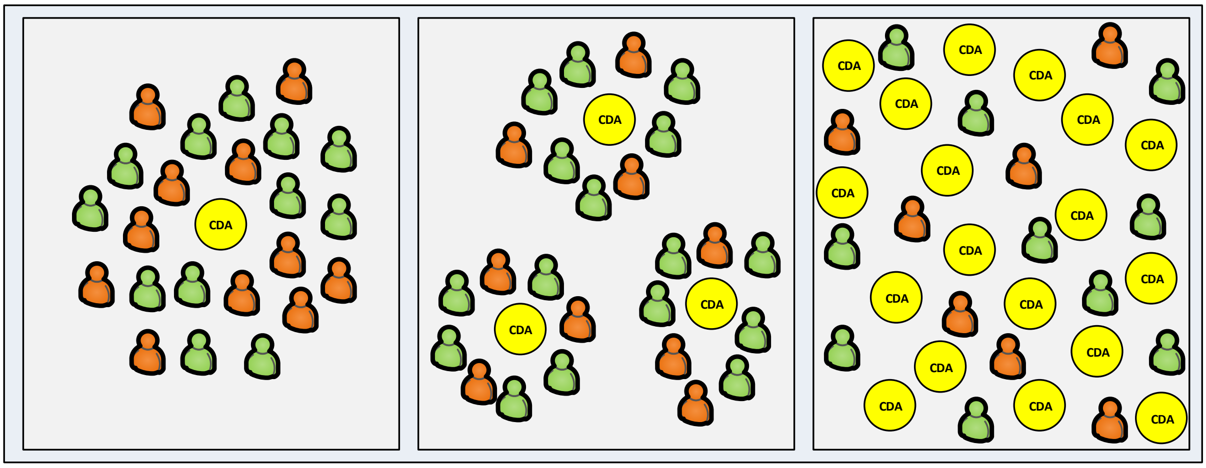

Figure 5.21 - CDA deployment strategies overview

In the figure three strategies are considered. The left strategy has one CDA for the entire system. This will represent the central repository approach that in the case of failure will bring the MAS to a complete stop. The middle strategy has several clusters formed having a CDA representing them. This will reduce the number of exchanged messages across the system but

at the same time providing some system redundancy. The right strategy has a considerable amount of CDAs along with other agents. This will guarantee the system redundancy but will again increase the number of messages exchanged between the existing CDA.

Adding system functionalities were they are not needed can bring no advantages, even increasing the complexity of an already complex system. The right deployment strategy can then be used to help optimize the overall system performance.

Considering MAS, the system granularity is given by the hardware modules themselves which are normally placed in stations. The MAS physical model can then be used for the CDA deployment. Each CDA can be used to represent a group of modules that are meant to work together to achieve some goal, or representing a group of modules that are physically on the same station, or even on a same room. The MAS are divided in workstations which normally have a controller for a group of modules. This controller will be used to run the main agent platform.

Having then a CDA in each main agent platform provides a natural way to address the deployment issue. However, tests on the load in the network should be conducted in future work to establish the optimum deployment strategy.

5.4 Chapter Summary

Having a central assembly capabilities repository in a modular conceptual world can be seen as a drawback in the MAS concept. This chapter proposes a delocalized CDA to maintain the information of all the system capabilities while being capable to readapt to changes. It can communicate across different platforms and keep track of other CDAs, along with their registered Skills. Redundancy and persistence emerges from the interaction of this collaborative behaviour. It increases the system robustness by not having the central capability repository allowing the system to keep functioning even if part of it goes down. Being capable of delocalization, it can be adjusted to diverse system topologies and sizes.

All these features go in the same direction of the MAS Plug and Produce concept, an individual but however always updated CDA can provide a viable solution to maintain, broadcast and secure all available assembly capabilities in a MAS.

6 Lightweight Agent Control Architecture

6.1 Introduction

The current market situation seen in the literature review has set the path for new assembly systems to be researched and developed. These new assembly systems aim to provide the industry a cost and time effective method for it to answer to the new market demands. The research is converging to a concept of “Plug and Produce” assembly systems [18] based on a modular approach. This is particularly important in the assembly domain. These modular system concept is denominated Modular Assembly Systems (MAS). MAS can then be easily adapted and reconfigured to quickly attend to the market demands. MAS allow assembly capabilities to be added or removed from a system just by introducing or eliminating hardware modules. This will provide the ability to react to product changes or new product introductions by changing the right set of hardware modules.

Multi agent control systems are seen as a promising solution to control MAS [48]. Agent collaboration draws a direct parallelism with the hardware modules of a MAS working together for creation of a desired product. An agent can then be modelled to represent any physical or conceptual entity required by the control architecture for its correct functionality. With this, a multi agent environment could be modelled to represent a MAS.

Industry requires the same performance that the currently used control methods can deliver. However, there are not yet many fully tested implementations that could be deployed to control these modular systems. Due to that, also few real systems analyses on the performance and response of these control implementations have been made. More analysis and tests are needed so to prove the concept to a change averse industrial domain. Having a light control model based on agent technology can then provide a benchmark assessment against the currently used control methods.

This chapter objective is to create a light multi-agent control architecture for MAS. It will be constructed with the minimum required functionalities so that it can provide a quick control response. With that, a contribution for the research being made around the usability and performance of these control architectures will be made.

6.2 Agent Control Architecture Requirements

To achieve a functional agent control architecture implementation, its model should be correctly defined so it may grow based on strong foundations. The required agent types along with their objectives and interactions will be crucial for the correct model to be obtained. For a light implementation, one agent to represent the hardware modules with their assembly capabilities and another representing the product are the bare minimum for such systems.

The assembly processes can be simple or complex, and of any type. Due to that, a transparent way to represent all of them within the agent environment in such a way that they can be easily understood by the agents needs to be provided. This will provide the means to represent all assembly processes that a product may require, and at the same time, all the equipment module assembly capabilities. A proper way to match between the module assembly capabilities and the ones needed by the product should also be made available.

The agent architecture ultimate objective will be to provide the means to establish an execution environment in which a product can be obtained. The agent environment will then need to be able to interact with the hardware modules so that assembly processes can be executed. Also, the agent representing the product requires a correct execution sequence for the assembly processes so the product can be correctly obtained.

MAS are divided in independent equipment modules that do not require information about one another. Each of these modules will have their own assembly capabilities, and will provide their representation in the agent environment. An overarching assembly processes repository should be made available and known across all agents in the environment. This will provide the condition for agents to have information about the existing assembly processes in the system.

For the correct agent architecture functionality, agents need to cooperate with other agents in the environment. This will allow individual and group goals to be achieved. Protocols are then needed so that correct agent collaborations can be obtained. Also aiming at the agent control architecture performance, agent communications must be minimized to decrease agent decision times.

With all these requirements met, the right model for the agent control architecture can be constructed which will then allow the needed performance tests to be made.

6.3 Agent Environment Assembly Process Representation

The assembly processes can be simple or complex (formed by combination of simple processes). Also, the same assembly process can be obtained using different techniques with different execution characteristics. A clear way to represent these processes inside the agent environment is then needed so that each agent can understand each assembly process. An appropriate representation was proposed in [70] and it establishes the skill concept as the basis for the definition of the assembly capabilities of a system.

The skill concept will provide the means to describe and represent assembly processes in a clear object oriented control structure which can easily be interpreted within the context of the proposed agent environment [28]. The key innovation is the composite definition which incorporates both the functional description and the actual execution definition of the assembly process [28].For the proposed light agent control architecture, only some skill model aspects will be considered to allow the execution complexity to be minimized. The assembly processes along with their execution characteristics will be encapsulated in an Atomic Skill. The Atomic

Skill inherits all the Skill features, adding the mechanisms to trigger the execution of the assembly process and acknowledging its termination. This is accomplished using the approach defined in chapter 4. Also, it includes the module characteristics to accomplish that specific assembly process the Skill is representing. The internal Atomic Skill data model is shown in the Table 6.1.

Table 6.1 - Atomic skill class content definition

Atomic Skill		
Name	**Description**	**Type**
Skill	Skill Class type variable containing the information about the assembly caability.	Skill Class
Zone	Location where the module is.	String
StartVar	Variable to trigger the assembly process execution start in the PLC.	String
EndVar	Variable to acknowledge the assembly process end in the PLC.	String

To keep the architecture as light as possible, the only considered assembly process attributes are the location where the physical module is in the system, which is stored in **Zone.** The **StartVar** and **EndVar** will respectively represent the variables needed by the process described in chapter 4 to start the assembly process execution and to acknowledge its termination. The hardware interaction methods will also be available in the Atomic Skill class. The **Atomic Skill** extends the Skill class described in the literature [28]. Its UML Class overview can be seen in Figure 6.1.

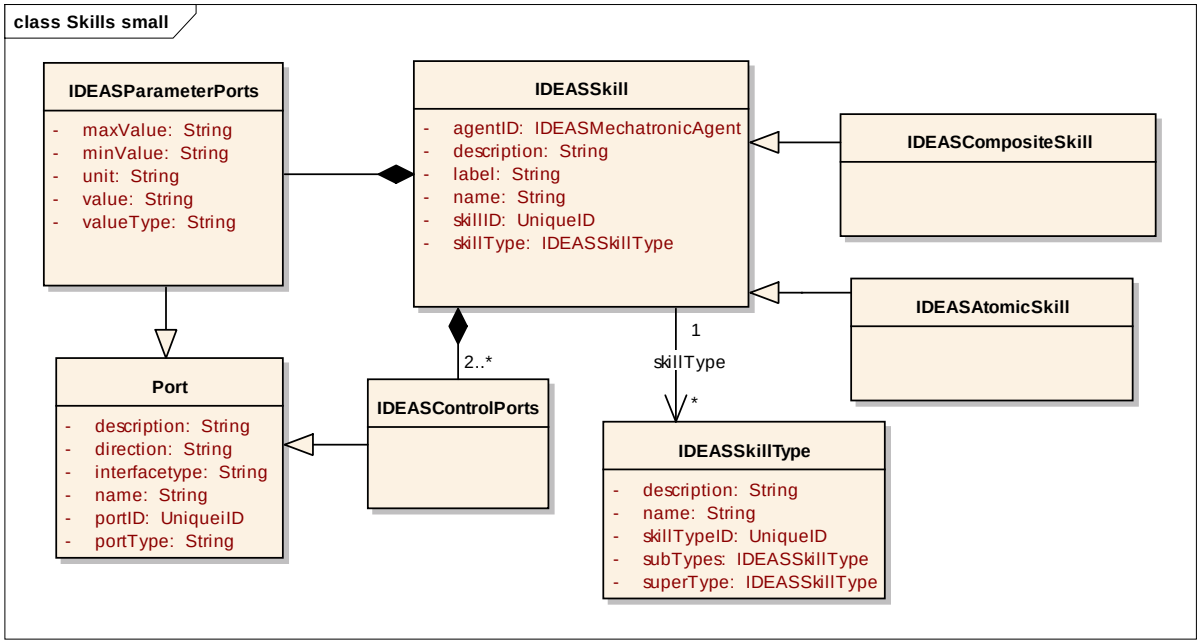

Figure 6.1 - Skill Definition main concepts [8]

The Atomic Skill will represent the system assembly capabilities in the agent environment. A representation for the required Product assembly capabilities is also needed. This will be achieved using the Skill Requirements concept [28]. It provides a similar structure to the Skill concept, containing ports and skill types that enables the automatic mapping between the two concepts [28]. Both Atomic Skill and the Skill Requirement will be used to match the product needed assembly capabilities with the capabilities available in the system.

This allows the existing assembly processes to have a representation in the agent environment, and also that matches between the product required assembly processes to be made with the ones available in the system.

6.4 Agent Control Architecture Definition

As seen in the literature, JADE platform [68] can provide the basic foundations for an agent architecture to be created. The proposed model will be defined based on the platform capabilities. For a correct agent architecture functionality to be obtained its agent members should be clearly identified and defined. The minimum required agents that need to exist are the Product Agent (PA) representing each product and the Resource Agent (RA) representing each module and its capabilities. These proposed agents will be based on the model seen in Figure 6.2.

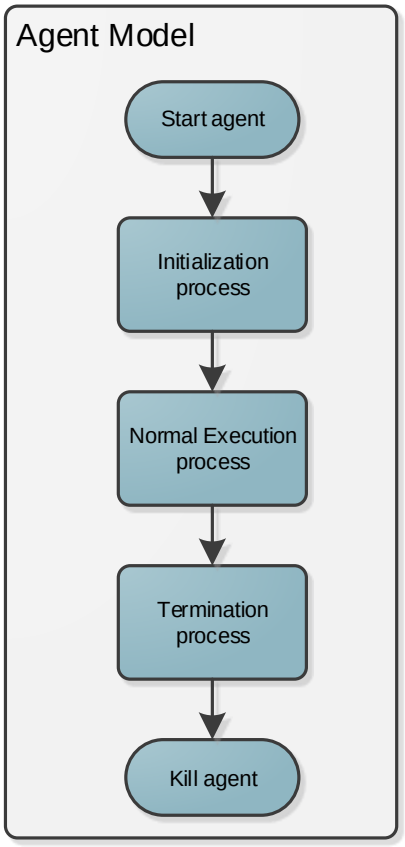

Figure 6.2 - Agent life cycle model

Looking at the figure, each agent in the architecture will have an Initialization Process after its creation. This is used to process and prepare the information it needs for its correct functionality. After that it will enter its Execution Process, in which the agent will be trying to achieve its individual objectives, which for some includes helping others achieving theirs. When an agent has fulfilled its objectives and is removed from the environment, first it will go through its Termination Process. In this process it should do all the necessary tasks so it can leave the environment without harm. Only then the agent can remove itself or be removed from the environment. Nevertheless, the environment should have in place mechanisms to deal with an abrupt agent termination, to minimize the impact in the system.

The interaction and cooperation between the agents is what will enable the operation of the environment under the specification of the architecture. Therefore, agent interactions are the initial step for the agent control architecture implementation. This model will use the Capability Dissemination Agent (CDA) defined in chapter 5 as the environment skill repository. It will be used to maintain the information about all the skills in the system, along with the agents that can provide them. The use cases for the agents can be seen in Figure 6.3.

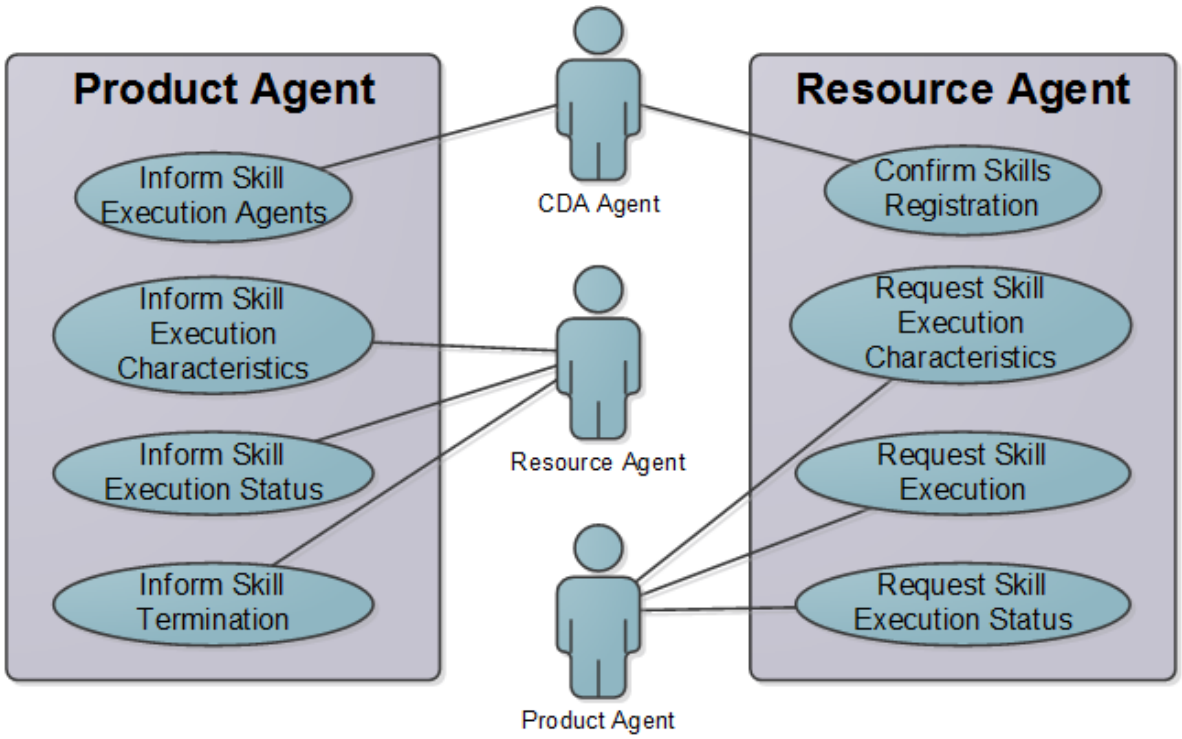

Figure 6.3 - Architecture agents use cases

As observed the previous figure, the CDA can inform the PA about which agents can execute the Skills, also, it can confirm the RA about its Skills registration request. A RA will have no interaction with others RA in the environment neither a PA with other PAs. A PA can request a RA about its characteristics to perform a Skill, it can request a Skill execution and also request status about a Skill execution progress. The RA can then inform the PA about its Skill execution characteristics, the Skill execution progress and the Skill execution termination. These use cases will define the main interactions that will occur between the agents present in the control environment. For this to be obtained, protocols should be created so that each agent can understand one another.

6.4.1 Agent Communication Protocols

In an agent environment communication is crucial as it is the instrument agents have to collaborate and achieve their individual objectives. Communication is obtained through the exchange of messages between agents. For these messages to be understood they need to be clearly defined and their sequence established. This is achieved by defining the communication protocols.

When a message is send and a reply to it is expected, a timeout should be launched to deal with the eventuality of a response failure. There should be some common sense about the amount of time to wait for a response. If it is to short, the other side may not have enough time to process message and send a response in time. Also, if it is too long, system

underperformance or even complete incorrect functionality may occur. With an appropriate timeout definition, agents can have behaviours to detect errors and try to overpass them.

The simplest message exchange protocol that ensures acknowledgement from both sides of the communication is FIPA Contract Net protocol [63]. Due to the desired light control model, the created protocols will be based on it. These protocols will represent the needed interactions required for the correct functionality of the agent control architecture.

For this implementation, two protocols are fundamental. The first protocol is used to obtain the specific information on the agent capabilities which can lead to their triggering. The second is to acknowledge the assembly process termination. This will be obtained using, respectively, the **Negotiation** and **Execution Protocols**.

6.4.1.1 Negotiation Protocol

The **Negotiation Protocol** is used to query a RA for its execution characteristics for a specific skill. The process can culminate in a request to start the execution of that skill. The message sequence of this protocol can be seen in Figure 6.4.

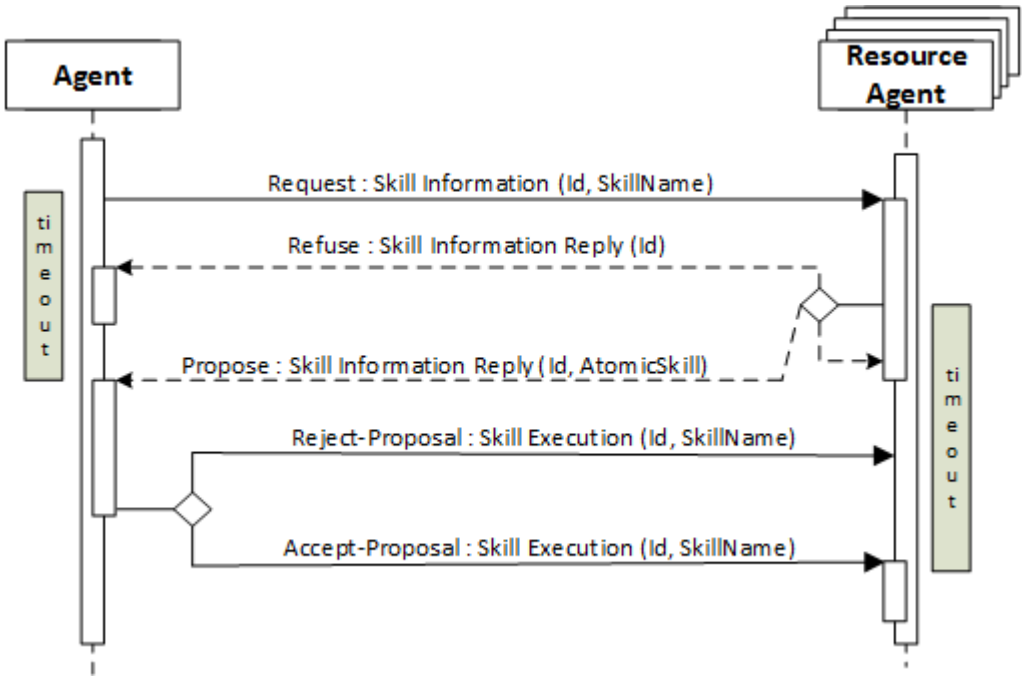

Figure 6.4 - Negotiation protocol message exchange definition

Each message will have a **Communication Id**, which will identify the specific Skill the message is associated with. The same **Communication Id** is used in the reply messages. Also, a **SkillName** will be part of some messages and is used to inform which the skill type the message is associated with is.

A **Request : Skill Information** message is first sent requesting the attributes for the execution of a particular Skill. When this message is received, it will be replied with the agent capabilities concerning the execution of the requested Skill (**Propose : Skill Information Reply)**. This message will contain the Atomic Skill for that assembly process. As seen before, it provides all the information for the execution of that Skill by the given equipment. However, if the agent is not interested on performing the Skill request, a **Refuse : Skill Information Reply** message is then sent, or even no message if the RA declines to provide any information.

A last message requesting the Skill execution will be sent, this will be the **Accept-Proposal : Skill Execution**, or if the proposal is not chosen, a **Reject-Proposal : Skill Execution** is sent instead.

This message exchange defines the **Negotiation Protocol**. This protocol should be followed in case of a successful negation by the **Execution Protocol** which will be used to control the Skill execution process.

6.4.1.2 Execution Protocol

This protocol will be used to know about the skill execution status and also its termination. For it, an answer from the RA informing about the start of a Skill execution status is expected. After that, periodic messages between both agents will be exchanged to see how the execution progress is progressing. On the execution end, a final message will be sent from the RA informing the Skill has terminated. The protocol message exchange sequence can be seen in Figure 6.5.

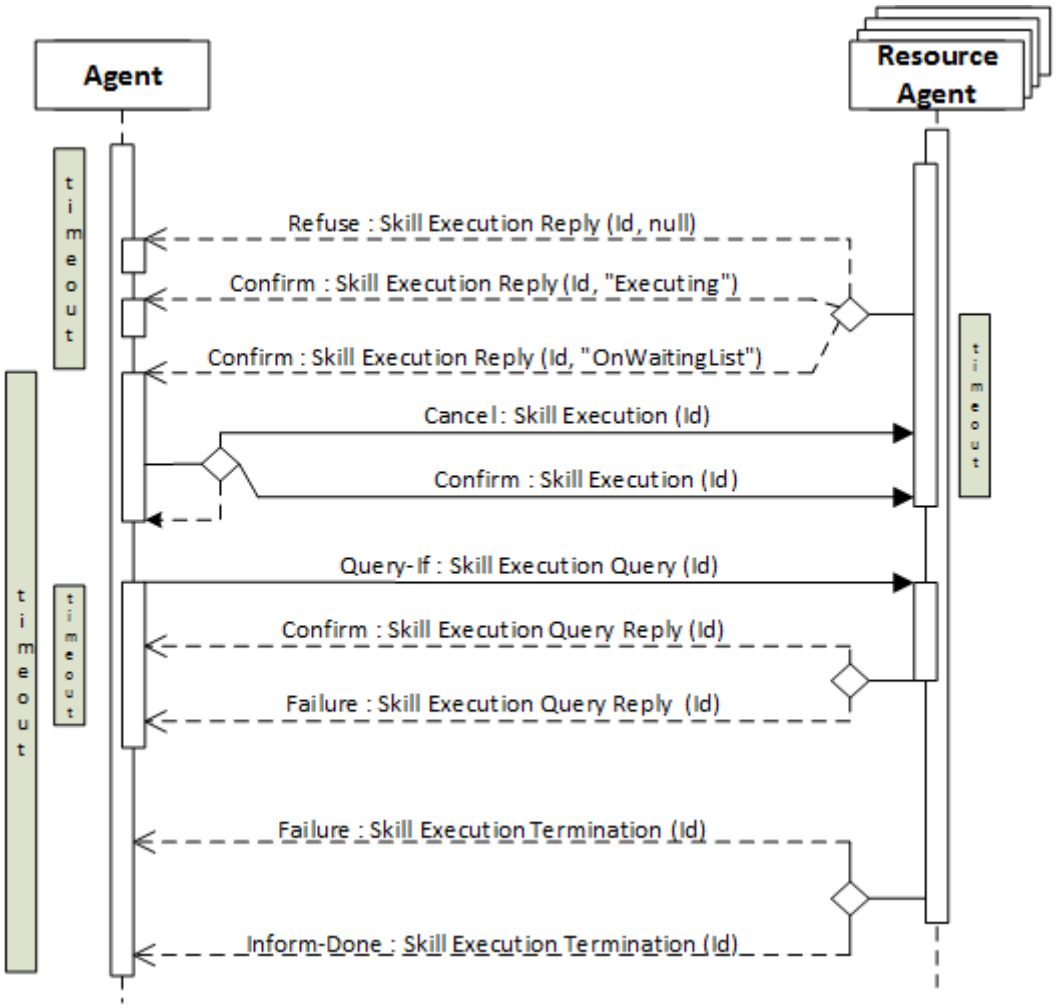

Figure 6.5 - Execution protocol message exchange definition

This protocol should take place after the end of the **Negotiation Protocol**. As in the previous established protocol, a **Communication Id** field will be used for the same aforementioned reasons.

When a Skill negotiation is ended, a reply message informing the requested Skill execution is then expected. A **Refuse : Skill Execution Reply** message will be received if the agent is not able to perform the Skill. A **Confirm : Skill Execution Reply** message is used if the agent will be able to execute the Skill. This message has two statuses. An "**Executing**" status will confirm the immediate start of the skill execution. An "**OnWaitingList**" status will inform that the agent is currently occupied, but it will execute the skill when possible. This can then be replied with a **Cancel : Skill Execution** to remove the desire for the Skill execution, a **Confirm : Skill Execution** to confirm the interest on the Skill execution, or not replyed at all waiting for the RA to become available for the execution. When the RA is free to execute the Skill, the **Request Execution Confirmation Protocol** defined below will be used.

During the Skill execution process, there is the need to keep track of the execution progress. This is obtained using the **Query-If : Skill Execution Query** messages. Which should be respond affirmatively if the skill is being executed (**Confirm : Skill Execution Query Reply**) or with a negative message if not (**Failure : Skill Execution Query Reply**).

To finalize the protocol, the final message will serve to inform that a skill execution has ended. This is obtained with the **Failure : Skill Execution Termination** message if the Skill had an error during its execution, or a **Inform-Done : Skill Execution Termination** if the Skill was executed without problems.

6.4.1.3 Request Execution Confirmation Protocol

For some reason the RA may not be able to immediately execute a Skill. When it can execute it, a protocol to inform the execution requester agent needs to exist. This will be made using the **Request Execution Confirmation Protocol**. This protocol can be seen in Figure 6.6.

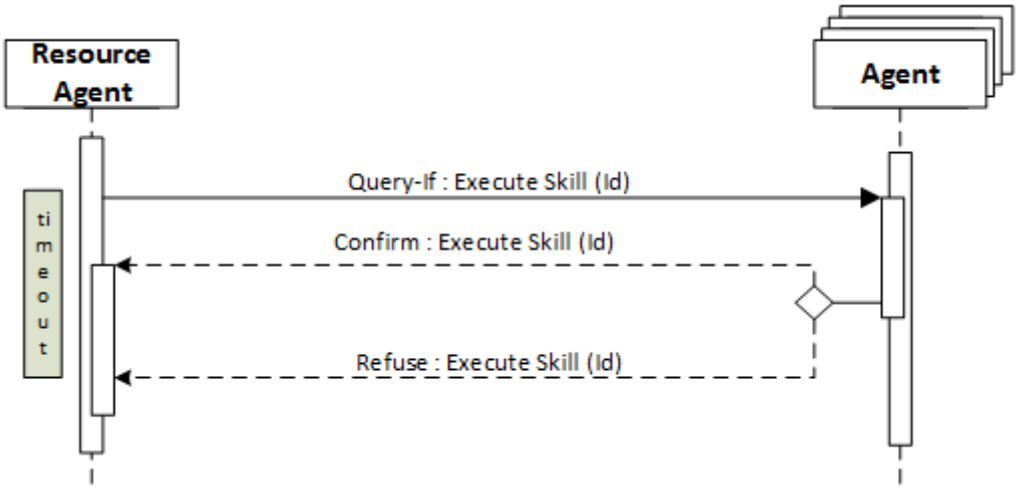

Figure 6.6 - Request execution confirmation protocol message exchange definition

When a RA becomes able to execute a requested skill it will sent a **Query-If : Execute Skill** message asking the requester agent if it is still interested in the skill execution. This agent will then reply with a **Confirm : Execute Skill** if he is still interested, or a **Cancel : Skill Execution** otherwise.

6.4.2 Resource Agent Definition

To fulfil its purposes, the architecture will represent each assembly processes through the Skill concept [28]. This will provide the means for the assembly processes to be easily understood inside the agent environment. The agent environment can then provide the proper means to execute each represented assembly process. The RA exists to interface when needed with the equipment and ensure the execution of the capabilities. It is the RA objective to serve as the bond between the physical and virtual domains. This is achieved by creating an

Atomic Skill for each assembly process an equipment module can provide. The information of what equipment module the RA represents should be passed during the Initialization Process.

As the RA will be able to interact with the physical layer, it can then be considered the base agent in the architecture. This interaction enables the start of the execution of each assembly process and also to recognize its termination. This process will be symbolized in the agent environment by the start of a skill execution and the acknowledgement of its termination.

The RA objective will be to execute the maximum number of assembly processes possible. To accomplish it, its implementation will follow the model defined in Figure 6.7.

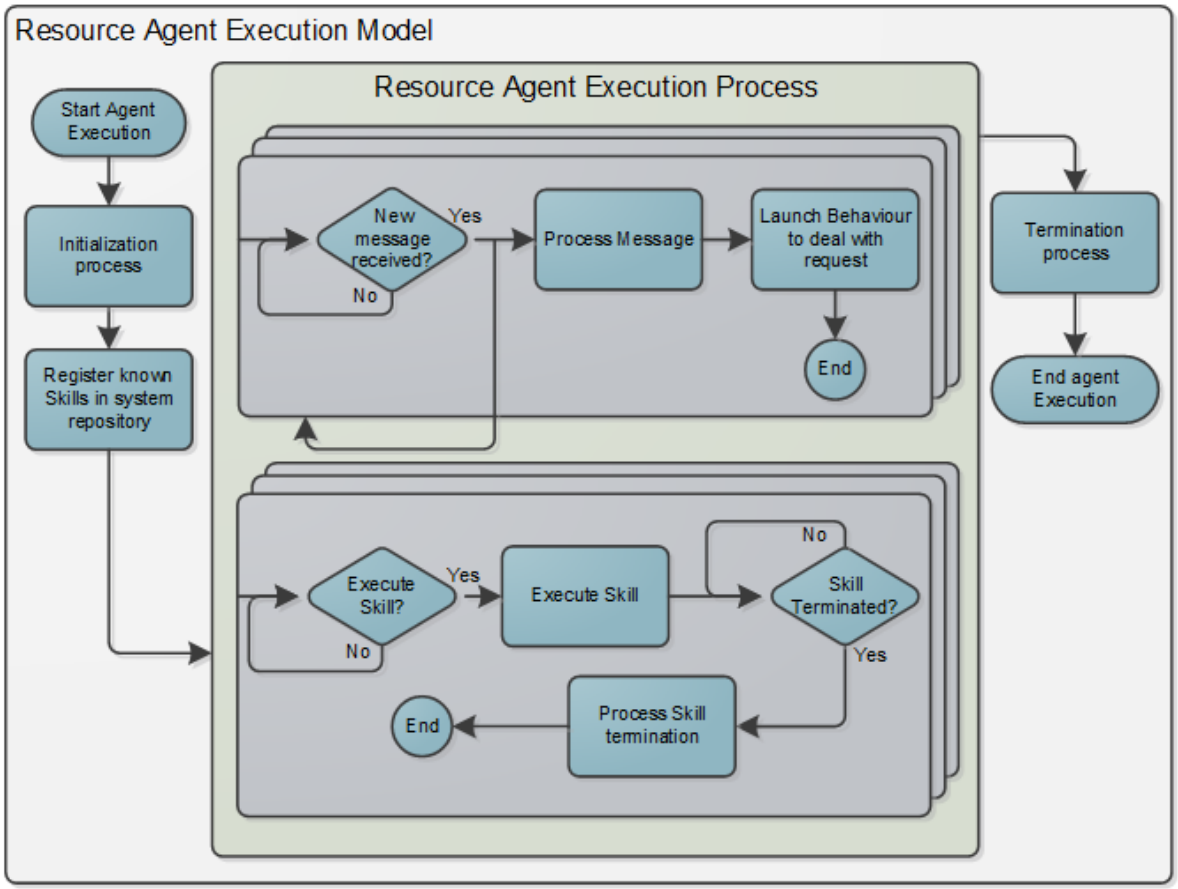

Figure 6.7 - Resource agent execution model

After the RA is introduced into a system, during the Initialization Process it will process the necessary information to allow its correct functionality.

Each of the RA Atomic Skills will then be registered in the Skill Repository Agent so they can be properly known within the agent environment. After this registration the RA will enter its Execution Process where he will be waiting for messages. These messages may be received in parallel. The RA needs to be able to attend to all of them simultaneously. The RA needs to know how to deal with the individual Skill execution requests in any possible occasion. Also, it

should inform the agent that requested the execution of its decision so that the other agent can decide how to proceed. A message may be received triggering a Skill execution. This process to execute a Skill will be made parallel to the message receiving.

As each RA will represent an equipment module and its capabilities, it should be present in the system while the module is connected to the system. Accordingly, once the module is disconnected the RA should inform the environment that it will no longer be available and deregister its Skills from the Skill Repository Agent. After that, it will remove itself from the environment. The internal information to fit the RA needs is summarized in Table 6.2.

Table 6.2 - Resource Agent internal data content definition

Resource Agent		
Name	**Description**	**Type**
ResourceId	Unique identification of the RA.	String
Skills	List with all the Atomic Skills the agent know how to perform.	ArrayList[](ModuleSkill)
RepositoryAgent	Agent identification of the agent that will be used has a service repository.	AID
Zone	Ip address of the module the RA is associated with.	String
QueueList	List to store all the skill execution requests that cannot be immediately executed.	ArrayList[](ACLMessage)
Executing	ArrayList with all the Skill names the agent is currently executing.	ArrayList[](String)

The **ResourceId** will store the agent unique identification within the environment. It needs to be unique to unequivocally identify each RA. The **Skills** array will have all the Atomic Skills that the module can execute. The agent to be used as a Skill Repository will also be needed. The used agent is a CDA. Its information will be stored in **RepositoryAgent**. The **Zone** will contain the information about where the module is located. The RA will have a **QueueList** array, which will be used to store the received Skill execution requests messages while the agent cannot execute them immediately but will eventually do it. For this light model this feature will not be used, however, in a complex environment where several PA could be interested in a RA at the same time and parallel requests could be made, this feature will allow the correct functionality of the RA. The **Executing** will be used to control the agent execution status, it will contain all the Skills currently being executed by the RA.

This model description provides the grounds for the RA implementation. The first request the RA may receive is to provide information about how he can accomplish an assembly process.

This is made using the **Skill Information Response Behaviour**. Also, when a **Skill Execution Request** is received, it will be processed using the **Skill Execution Behaviour**.

6.4.2.1 Skill Information Response Behaviour

When any other agent in the environment requires the information about how a RA can accomplish a specific Skill, it will send him a message requesting its Skill execution information. This should be made respecting the **Negotiation Protocol**. A **Propose : Skill Information Reply** message will then be created and sent to the agent that triggered the process. This message will contain the Atomic Skill associated with the request as it contains all the information about the skill execution. Or, in the event of the RA is unable to perform the Skill, a **Refuse : Skill Information Reply** is then sent.

After this, the agent may want the RA to execute the assembly process. This is made sending the **Accept-Proposal : Skill Execution** message. When this message is received, the **Skill Execution Behaviour** will be launched to process it. It is important to note that the skill information response behaviour is always operating to process any requests in parallel with the execution behaviour.

6.4.2.2 Skill Execution Behaviour

When an agent in the environment decides which RA he wants to execute a skill, it should send a **Accept Proposal : Skill Execution** message to that RA requiring its execution. The RA will always be able to receive requests and process them accordingly besides being execution or not a Skill. When an **Accept-Proposal : Skill Execution** message is received, several behaviours will be used to complete the execution request. The first behaviour to be launched is the **Process Skill Execution Request Behaviour**. The algorithm to deal with this message can be seen in Figure 6.8.

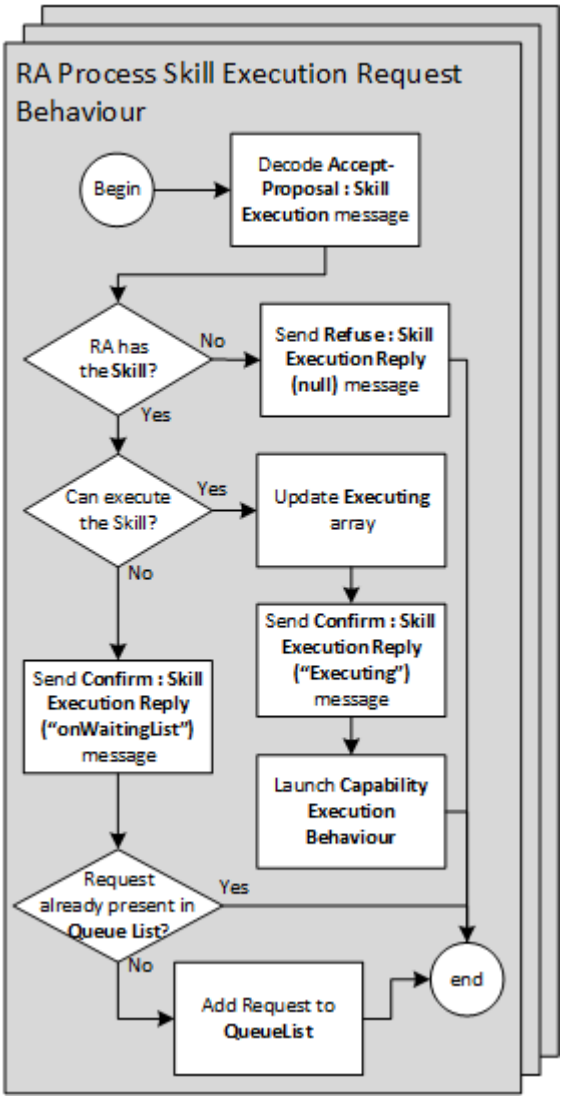

Figure 6.8 - Resource agent process skill execution request behaviour algorithm

This behaviour will process the request message and act accordingly. Several reactions can be triggered depending on the execution status of the RA. To begin with, the wanted Skill will be extracted from the request message and check if it is available in the RA **Skills** array. If it is not, a **Refuse : Skill Execution Reply** message will be sent and the behaviour ended.

On the contrary if the RA has the Skill it will see if it can execute the requested Skill. This is achieved by checking if the requested Skill is not in the **Executing** array and, the **Process Queue List Behaviour** (defined below) is not currently being executed. As if it his, a request for that same Skill may already be made and it will eventually be processed. If the Skill can't be executed, a **Confirm : Skill Execution Reply ("onWaitingList")** message is sent. If the execution request from that agent and for that **Skill** is not yet in the **Queue List** it will be added, otherwise no extra action will be made and the behaviour will be terminated.

On the other hand if the RA can initially execute the Skill, it will update the **Executing** array by adding all the Skills that cannot be executed while the requested Skill is being so. A **Confirm : Skill Execution Reply** message will be sent to the requester agent informing that the Skill execution have started and the **Capability Execution Behaviour** launched to execute the Skill. This behaviour algorithm can be seen in Figure 6.9.

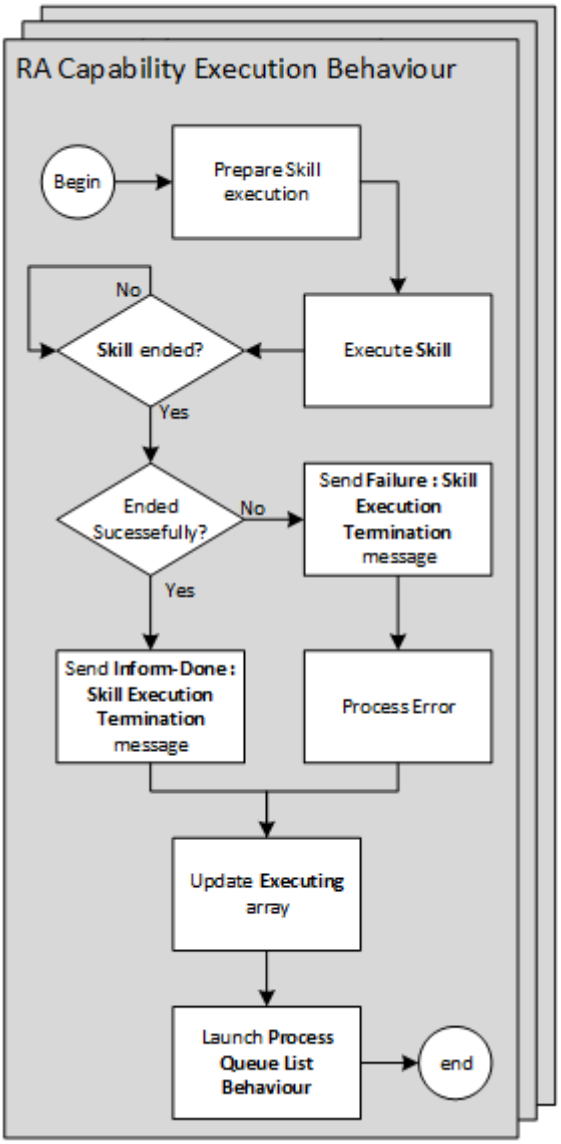

Figure 6.9 - Resource agent capability execution behaviour algorithm

The **Capability Execution Behaviour** will execute an assembly process associated with a Skill. In this model it will be achieved using the process established in chapter 4. Initially the Skill execution will be prepared by obtaining all the needed information for the aforementioned process. The actual capability execution will then be activated. When the skill termination is acknowledged, the RA will know if it has been completed successfully or with errors. A message informing the skill has ended will then be sent to the agent that requested the execution,

informing it of the skill termination status. An update to the **Executing** array will also be made, removing all the Skills that can now again be executed. During the Skill execution new Skill execution requests might be received. The **Process Queue List Behaviour** will be launched to process these requests. This behaviour algorithm can be seen in Figure 6.10.

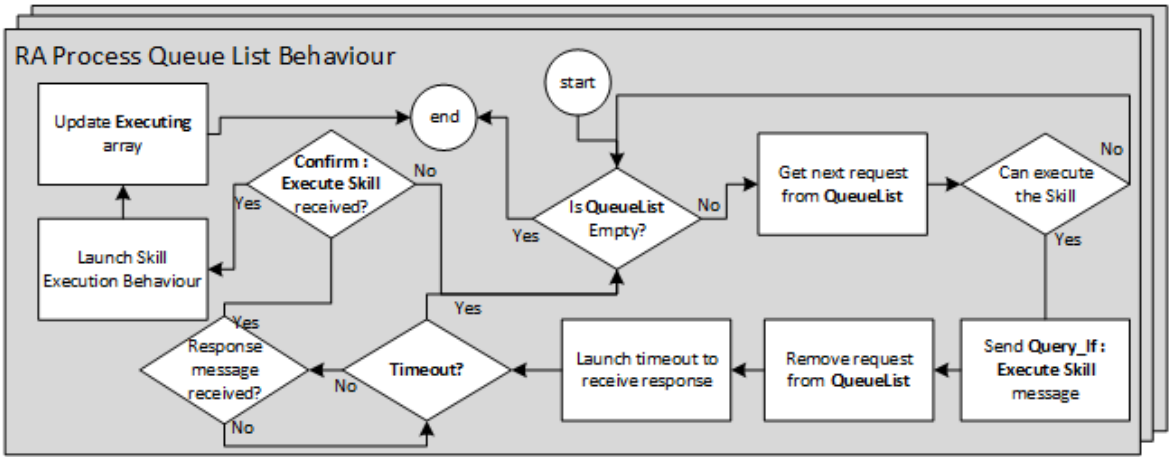

Figure 6.10 - Resource agent process queue list behaviour algorithm

The RA will see if the **Queue List** is empty or not. If it is, no execution requests were made while the capability was executing and the behaviour is terminated. Otherwise, there are execution requests pending and the RA will process them.

The RA will take the first request from the **Queue List** and analyse if the skill can be executed by checking if it is not in the **Executing** array. If it cannot, the next request in the **Queue List** will be process. Else the **Request Execution Confirmation Protocol** will be followed to see if the execution requester agent still wants the Skill to be executed. A **Query-If : Execute Skill** will be sent. The request removed from the **Queue List** and a timeout launched for a reply message to be received. If an **Confirm : Execute Skill** message is received, the timeout will be disabled, the **Executing** array updated with all the Skills that can no longer be executed, the **Capability Execution Behaviour** launched for that execution request and the **Process Queue List Behaviour** terminated. Otherwise, if no reply is received and the timeout is triggered or if a **Cancel : Skill Execution** message is replied, the next request in the **Queue List** will be processed. While the RA is processing its **Queue List**, if a new execution request is received, it will be added to the end of the **Queue List** and processed when the RA reaches it. This will be the model for the RAs to handle all the execution requests.

6.4.3 Product Agent Definition

The PA can be seen as the triggering member in the architecture as it will be representing the products in the agent environment. The PA should be unaware of the assembly system topology and also have no means of performing any assembly process by itself. It does

however have the necessary requirements to create a product. In order to do so it has to interact with other agents in the system using the previously established protocols, obtaining then other agents to perform the assembly processes in his behalf.

The PA should be able to represent any needed product. Therefore it is important to establish the means to represent the product requirements. This representation is obtained by the product workflow as defined in literature [28]. The PA main objective will be to get all the assembly processes requirements in the workflow executed by collaborating with RA agents. When all the assembly processes requirements are completed, the wanted product has been obtained and the PA has fulfilled its objective. The PA will then remove itself from the environment as it is no longer needed. It is important to note that before termination it will store all the information related to assembly process execution which led to the creation of a given product. The PA execution model can be seen in Figure 6.11.

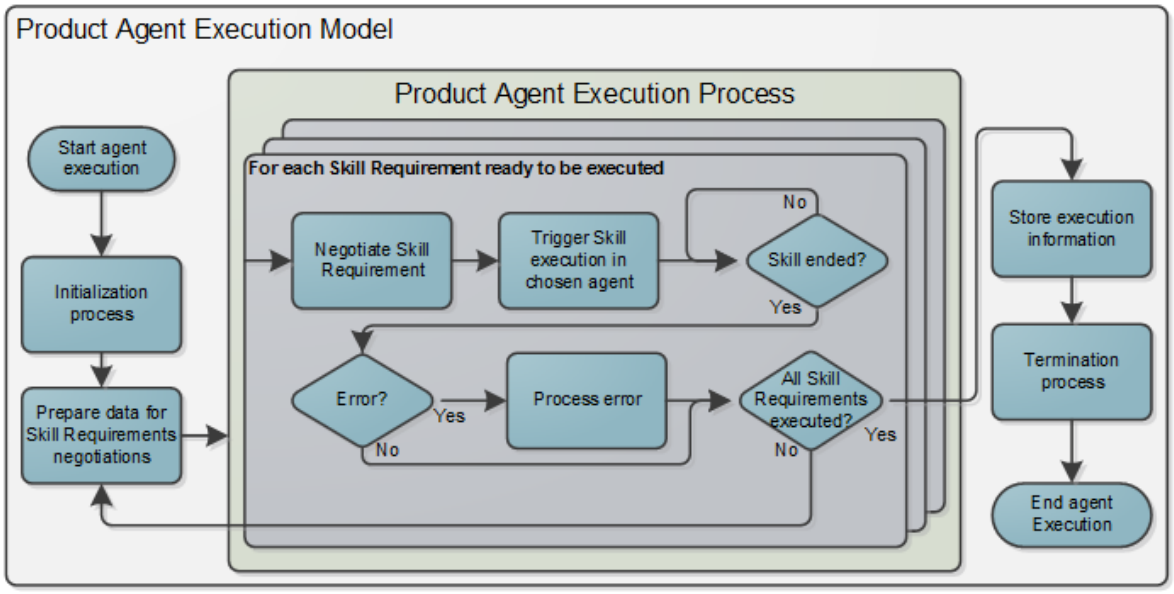

Figure 6.11 - Product Agent execution model

When the PA agent is created it will receive all the required information for its correct functionality. This information can be seen in Table 6.3.

Table 6.3 - Product Agent internal data content definition

Product Agent		
Attribute	**Description**	**Type**
ProductId	Unique identification of the Product within the environment	String
RepositoryAgent	Agent identification of the agent that will be used has a assembly capability repository.	AID
Zone	Ip address where the product currently is.	String
ProductWorkflow	All represent the product workflow, with all the Skill Requirements and their connections needed to produce the product.	CompositeSkill

Each PA should be uniquely identified within a system, the PA identification is stored in the **Product Id**. Each PA also needs to know the required assembly processes along with their correct execution sequence. This will be represented in the **ProductWorkflow** as a Composite Skill [28]. Each PA will also need to be aware of its current location within the system, which will be stored in **Zone**. However, it does not need to possess the knowledge of the system topology or existing zones to properly function. The responsible agent that will have the information about all the available Skills in the system will be stored in the **RepositoryAgent**. This will be used to search for the agents that can perform each of the Skill Requirements in the workflow. For this model the proposed CDA in chapter 5 is considered.

After the Initialization Process, the PA will prepare all the Skill Requirements that can be executed. For each of them, a RA to execute the required assembly process is used. The Skill negotiation will be made to obtain the RA, next the skill execution request will be sent to it triggering the execution. When the skill termination is acknowledged without errors, if there are skills to be executed in the workflow, the Skill Requirement execution preparation will again be made, and the process repeated. Once all the Skill Requirements are executed, the PA will enter its Termination Process, in which the PA will store all the information related to the complete assembly process execution which led to the creation of a given product and then remove itself from the system.

For the PA to execute the Skill Requirements in the right sequence, a precedence approach will be used [8]. This will provide the required means to clearly define the product workflow and enable the execution of the necessary Skill Requirements.

6.4.3.1 Precedence Execution Methodology Definition

The PA objective will be to go through the product workflow, and define which Skill Requirements can be executed. This execution needs to be made in the correct order so that the desired product is correctly obtained. The Precedence Execution Methodology [8] was first

created to be used in the IDEAS project. However, as it will constitute a fundamental part for the PA its detailed analysis will be made. A view of the Precedence Methodology scheme can be seen in Figure 6.12.

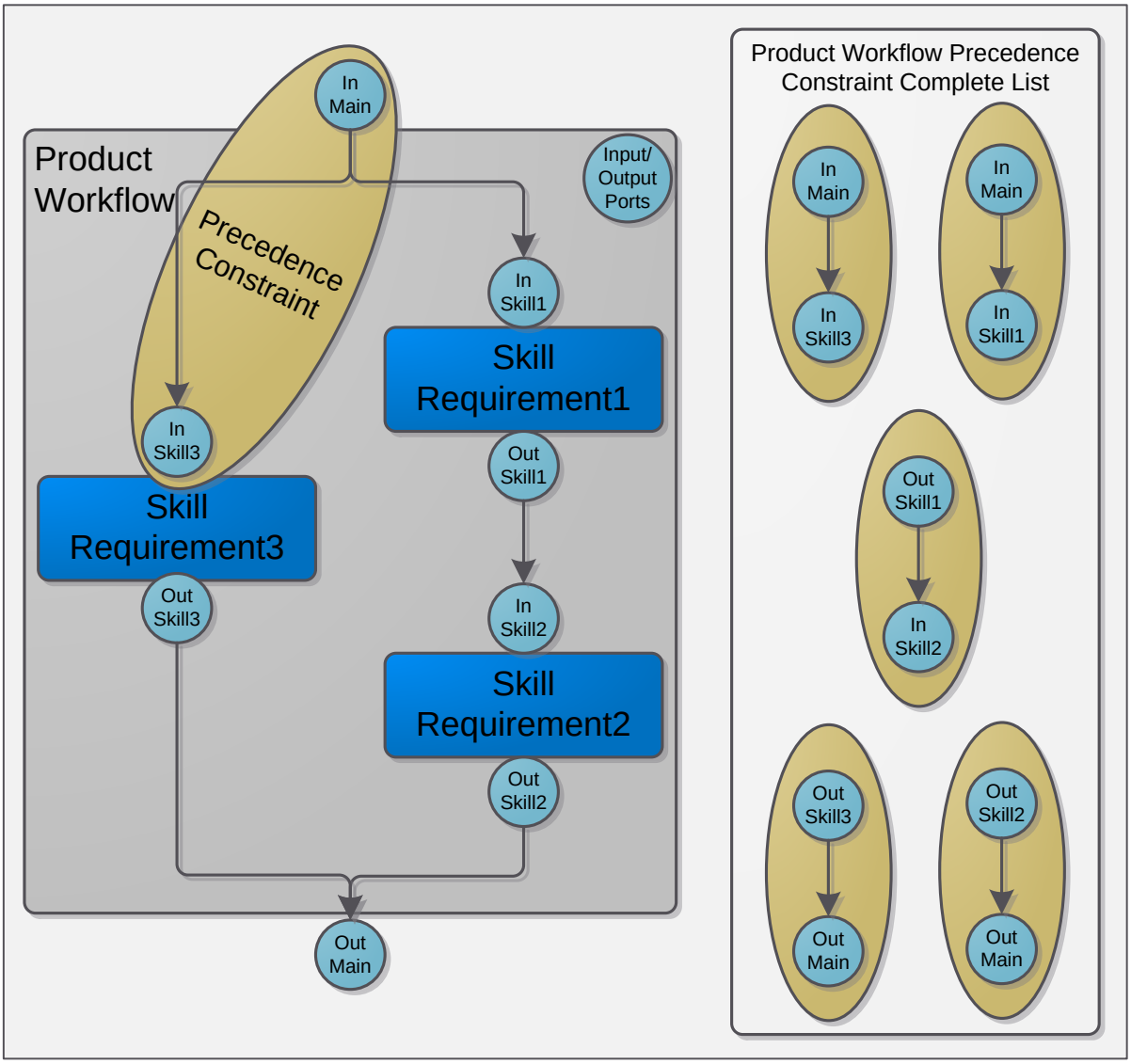

Figure 6.12 - Precedence constraint model overview

The precedence methodology is based on the skill concept. Correspondingly, the product workflow is then represented as a Composite Skill, in grey, accordingly, the Skill Requirements, in dark blue, represent each needed assembly processes [28].

Each Skill Requirement has associated two unique input and output ports, represented in light blue. The input ports are used to enable the Skill Requirement execution while the output ports are used to acknowledge that the Skill Requirement is executed. The Precedence

Constraint, in brown, represents the precedence connection that needs to exist between each individual Skill Requirement execution. This methodology attributes are the fundamental blocks for the Precedence Constraint Methodology execution to be comprehended.

In Figure 6.12, each Precedence Constraint is also characterized by two ports. Simultaneously, each of these ports are also associated with a Skill Requirement. The Precedence Constraint can be analysed from top to bottom, having the top port as its input and the down port as its output. Having that established, the Precedence Constraint functionality can be seen as a requisite, which will only enable its output port when the input port has also been enabled. Each Precedence Constraints ports are also Skill Requirement input and output ports. Moreover, each port can be present in two (or more) Precedence Constraints. Following the same idea as before, for a Precedence Constraint output port to be enabled, it needs that all the input ports present in to also be enabled. This can be observed in Figure 6.12 where the **OutMain** port will only be enabled when the **OutSkill2** and **OutSkill3** ports are.

Using the Precedence Constraint to connect each Skill Requirement will result in the notion of sequence, as the next Skill Requirement will only be triggered when all the needed previous ones have ended. Also, the Precedence Execution Methodology (PEM) provides the capability of parallel assembly processes execution as it allows Skills Requirements triggering process to occur in parallel.

The PEM will require an algorithm that provides the means to follow the product workflow correctly knowing which Skill Requirement can be executed, while at the same time keeping track of which ones have already ended. To achieve this, some necessary information must be used. The information about each Skill Requirement present in the workflow execution status will be stored in an array of **Executing Skills** objects. The object information is defined in Table 6.4.

Table 6.4 - Executing skills class content definition

ExecutingSkills		
Attribute	**Description**	**Type**
bPorts	Will have all the ports that need to be validated so the associated skill can be triggered.	ArrayList[](String)
aPort	Will have the input port of the associated skill	String
isNegotiated	Used to represent if the Skill is has been negotiated or not.	Boolean
isBeingNegotiated	Used to represent if the Skill is being negotiated or not.	Boolean
Terminated	Used to represent when a Skill Requirement is been executed to the product.	Boolean
AssociatedSkill	Skill Requirement associated with the variable.	AtomicSkill
Executer Agent	Agent that is obtained in the negotiaton process and is executing the Skill.	AID

The **AssociatedSkill** variable will be the same as the Skill Requirement in the workflow that this variable is representing. The **aPort** will be the represented Skill Requirement input port. The **bPorts** will have the information about all the ports that need to be validated so that the Skill Requirement can be executed. These ports are obtained by checking all the Skill Requirement precedencies and getting all the input ports that have the **aPort** as their output port. The **isNegotiatited, isBeingNegotiated** and **Terminated** will have True or False values that will change accordingly at the same time as the skill execution progresses. Initially these three will all be False. This allows to keep track of each execution. The **Executer Agent** is the RA that executes the Skill Requirement the entry is associated with.

Having the basic data for the implementation of the PEM, a way to keep it organized is also needed. This is made using the Arrays seen in Table 6.5.

Table 6.5 - Precedence Execution Methodology data content definition

Precedence Constraint Methodology Data		
Attribute	**Description**	**Type**
SkillExecutionTable	Skills from the product workflow that are executing or in the eminence of being executed along with their execution status information.	ArrayList[](ExecutingSkillsClass)
Validated Ports	Will store all the Ports used to check if a Skill from the product workflow can or not be executed.	ArrayList[](String)

The **SkillExecutionTable** will be used to store a group of **ExecutingSkills** variables for those Skill Requirements that are being executed, are executed or are about to be. The **ValidatedPorts** will have all the output ports from the Skill Requirements that have ended their execution. This array is shared across all executions and should then be consulted to see if a Skill Requirement can or cannot be triggered.

With the needed data information established, the PEM algorithm can be constructed. It will be divided into three behaviours. A **Preparation Behaviour** to check which Skill Requirements are to be executed. An **Execution Behaviour** to execute and keep track of the execution status of the Skill Requirements and a **Termination Behaviour** to process each Skill termination. Also, for the correct algorithm implementation, the Arrays presented in Table 6.5 must always be updated. For this, regular accesses and updates need to be made during the entire process. These conceptual actions can be seen in Figure 6.13.

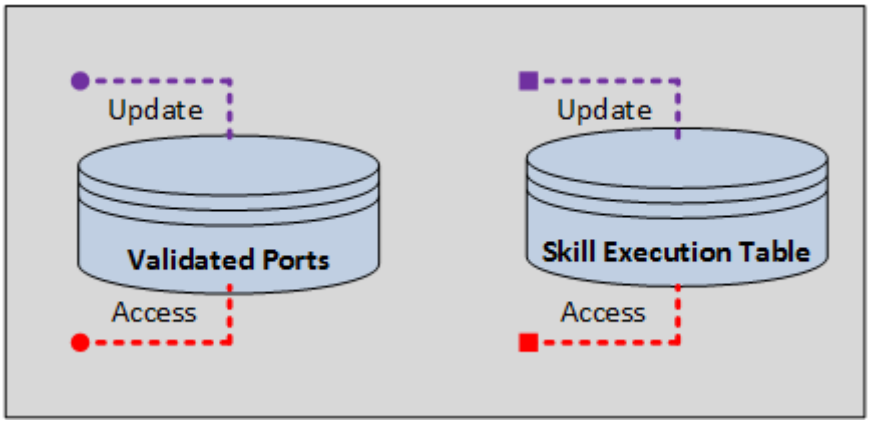

Figure 6.13 - Precedence Execution Methodology database access overview

The updates are represented by a purple line while the accesses are by a red. Also, actions to the **Skill Execution Table** will be symbolized with a square, while actions to the **Validated Ports** are with a circle. Having these considered, the **PEM Preparation Behaviour** algorithm can be defined. It can be seen in Figure 6.14.

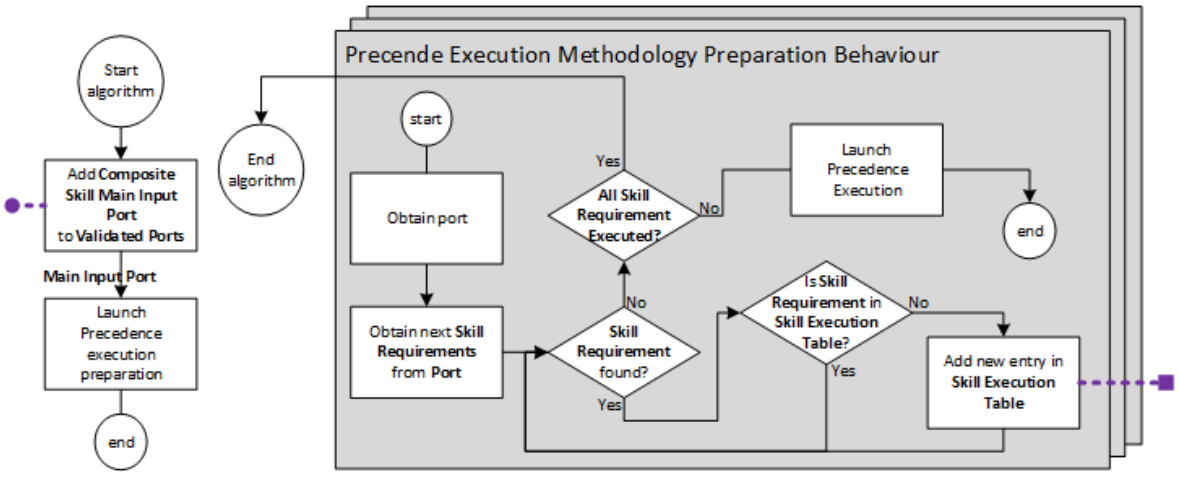

Figure 6.14 - Precedence Execution Methodology preparation behaviour algorithm

The product workflow will be represented by a Composite Skill. As a Skill with any skill instance, it will have an input and output port. To start the algorithm, the Composite Skill input port will be added to **ValidatedPorts** and the **PEM Preparation Behaviour** launched. Initally, using the constraint precedencies available in the composite skill definition the next Skill Requirements to be executed will be obtained. For all the obtained Skill Requirements, an entry will be created and added to the **Skill Execution Table** if one is not yet there for that particular Skill Requirement. If no new **Skill Requirements** are found, and all Skill Requirements executed, it will mean that the product workflow have been totally executed and the desired

product is obtained. This will also mean the end of the algorithm. Else the PEM Execution n Behaviour is launched to execute all the next Skills Requirements that can be executed. This behaviour algorithm can be seen in Figure 6.15.

.

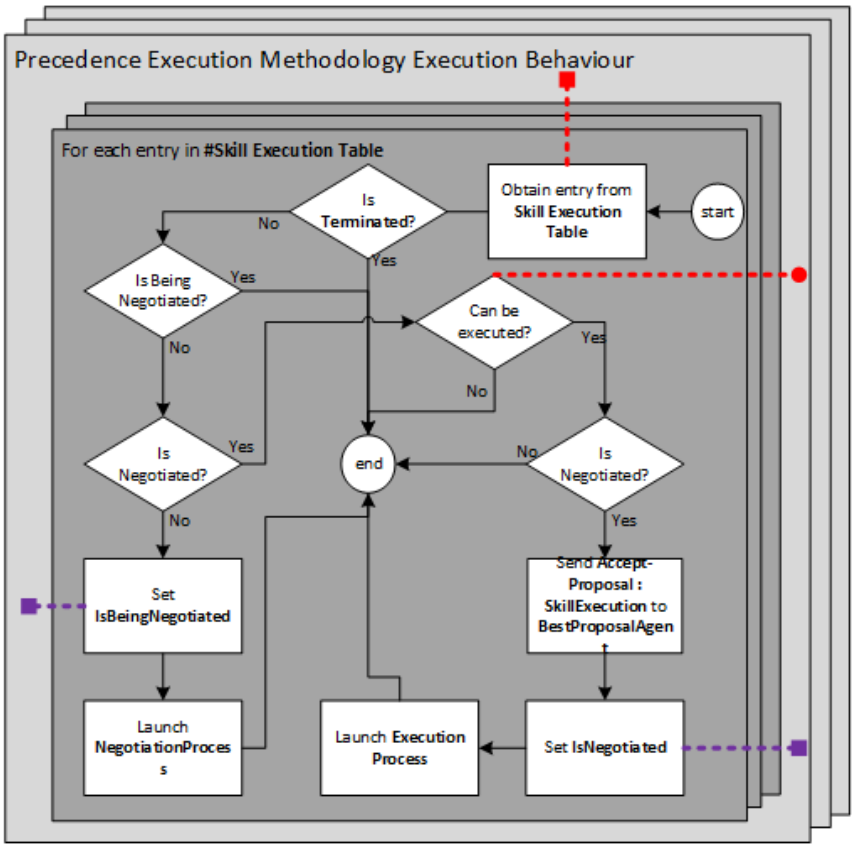

Figure 6.15 - Precedence Execution Methodology execution behaviour algorithm

For each entry in **Skill Execution Table** the associated Skill Requirement status will be analysed and the next action to be applied decided. If the Skill Requirement has been executed, no action will be made. Else, if it is not being negotiated and is not yet negotiated, **isBeingNegotiated** is set to '**true'** and the **Negotiation Process** launched, this will be defined below, but will be the implementation for the **Negotiation Protocol**. Else, if it is being negotiated nothing else will be done.

On the other hand, if the **Skill Requirement** has already been negotiated, the verification to see if all its precedence constraints are met is made. This is accomplished by checking if all the **bPorts** entries are present in **ValidationPorts**. If they are, all the needed precedencies are met, and the Skill Requirement can be executed. Else, the Skill Requirement cannot yet be executed and it will wait until all the needed precedencies are met. If the Skill Requirement can be executed, and its negotiation has not yet ended, this means the final message triggering the execution of the skill is still missing. The entry status of **isNegotiated** is set to '**true**' and the **Execution Process** launched. This will also be defined below, but will consist on sending the

last message of the **Negotiation Protocol** accepting the proposal and also the implementation for the **Execution Protocol**. When a Skill terminates its execution, the **PTM Skill Termination Behaviour** will be launched to process it. This behaviour algorithm can be seen in Figure 6.16.

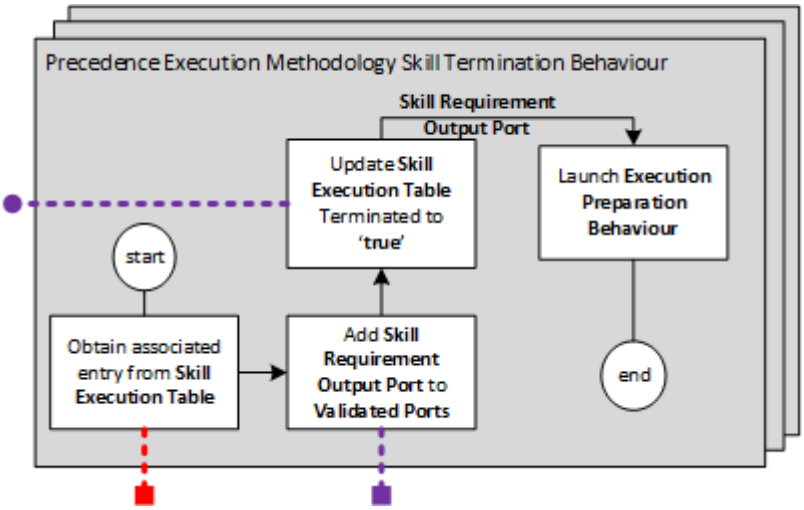

Figure 6.16 - Precedence Execution Methodology skill termination behaviour algorithm

When a Skill terminates its execution, the **PEM Skill Execution Termination Behaviour** is launched. It will receive as parameter the terminated Skill Requirement. The respective entry for that Skill Requirement is obtained from **Skill Execution Entry**. The **Skill Requirement Output Port** will be added to the **Validated Ports** so that the **Skill Requirements** that may have it as precedence can now be executed. Next, the entry **Terminated** status will be set to '**true**'. The next step will be to get all the next **Skill Requirements** that follow the one that have just ended and repeat the process once again. Which is made launching the **PEM Execution Preparation Behaviour** with the **Skill Requirement Output Port**.

6.4.3.2 Negotiation Process Behaviour

A Skill is a representation of an assembly process within the agent environment. For that reason, only some agents may know how to execute it. The Negotiation Process for the PA establishes the steps that needed to match between a Skill Requirement and a Skill. The agents who can perform a Skill are obtained using the CDA. The choice of the agent to execute the Skill can be obtained using some predefined complex rules [72]. When this agent is finally identified, a request asking for it to execute the skill can then be sent when possible.

When a Skill Requirement is to be executed, the Negotiation Behaviour will be launched. The Negotiation Behaviour will follow the **Negotiation Protocol** established before. The Negotiation Behaviour algorithm for the PA can be seen in Figure 6.17.

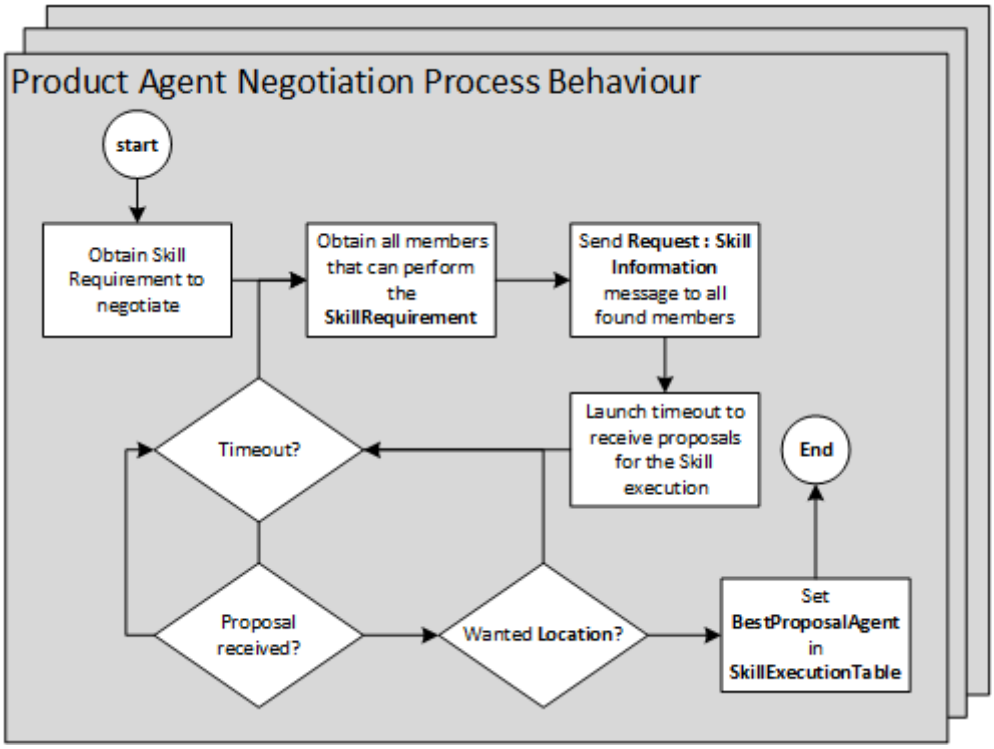

Figure 6.17 - Product Agent negotiation process behaviour algorithm

To negotiate a Skill Requirement, first the PA will query the **Repository Agent** receiving all the agents in the environment that are able to perform that Skill. This process will use the protocols defined for it in chapter 5. With that information, the next step will be to query those agents about their conditions to perform the Skill. At the same time, a timeout will be launched and proposals for that Skill will only be considered during that period. Each Skill query message will have the Skill Requirement identification and, the received proposals should bring it as well. This will allow each proposal to be correctly identified for each Skill Requirement negotiation that may be occurring in parallel.

In this algorithm and for simplicity, the PA will only consider the location origin of the proposals. And also, for process swiftness, the first proposal that met the conditions will be considered the best proposal. This will provide a quick response method however may not guarantee that the best available option is chosen. The bigger the system complexity the most elaborate this proposal decision mechanism may be. Identifying the most suitable agent to perform the assembly process may have a great impact on the overall system performance, particularly in complex systems. When the suitable agent is identified, the entry on **Skill Execution Table** for that Skill Requirement is updated, setting the **Executing Agent** as the chosen proposal agent and then the algorithm terminated. To finalize the **Negotiation Protocol**, a **Accept-Proposal : Skill Execution** message need to be sent. This will be made in the **Execution Process Behaviour**.

6.4.3.3 Execution Process Behaviour

When a Skill Requirement Negotiation Process ends, the Execution Process will take place. Initially the **Accept-Proposal : Skill Execution** is sent to the **Executing Agent** and a timeout for its reply enabled. This reply will confirm that the agent will execute the Skill, or not. If no reply, or a negative response is received, the Execution Process should stop and the Negotiation Process should be launched. Else, if a positive response is obtained; periodic status messages should be exchanged with the agent performing the execution with the purpose of knowing how the process is progressing. This will be made until the Skill termination message is then received. This will follow the established **Execution Protocol**. The **Product Agent Execution Process Behaviour** overview can be seen in Figure 6.18.

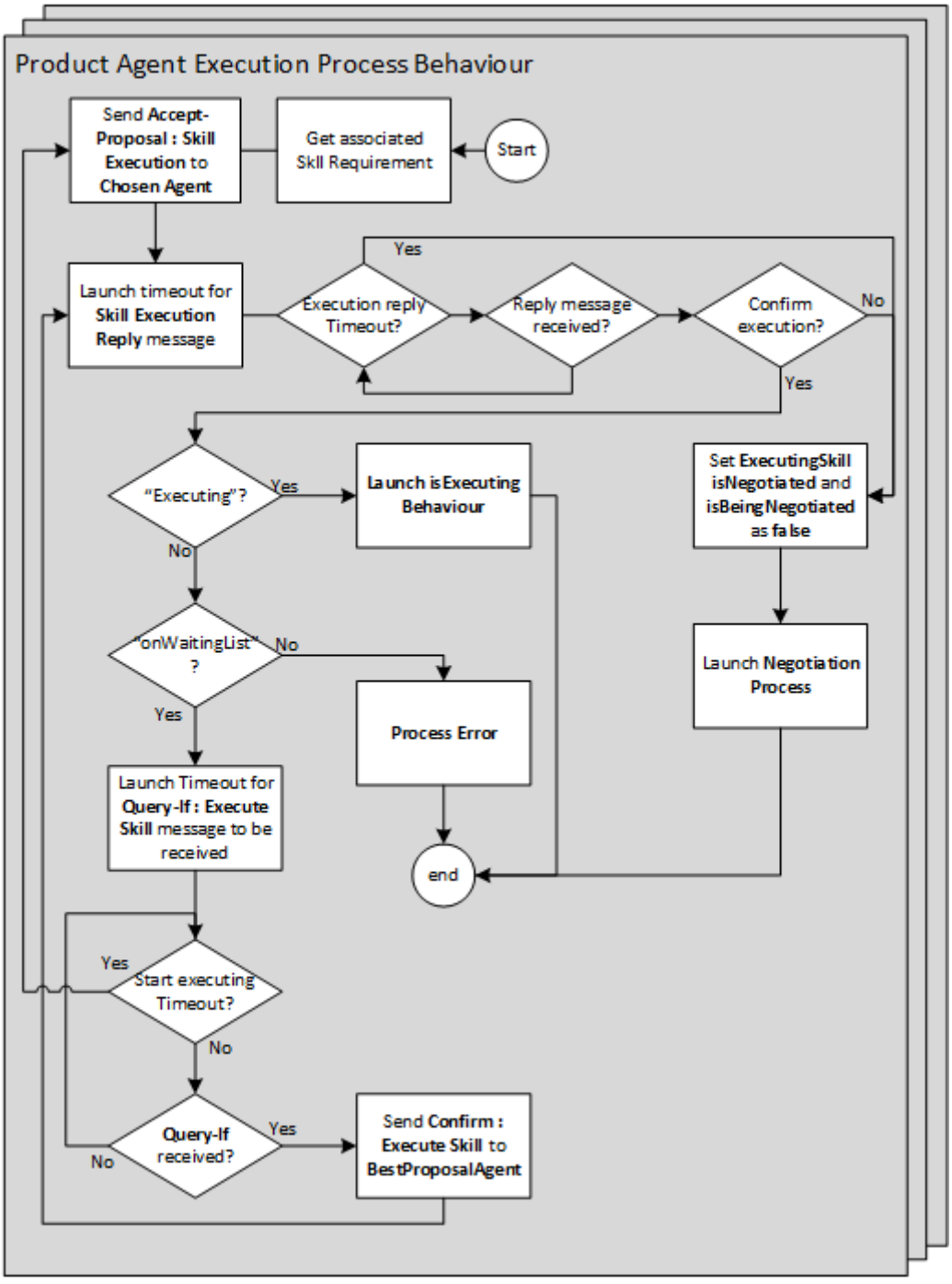

Figure 6.18 - Product Agent Execution Process Behaviour algorithm

Having the agent to execute the skill obtained during the Negotiation Process a **Accept-Proposal : Skill Execution** message will be sent to it requesting the Skill execution. A timeout to receive a response for this request is then enabled. If no response is received or the response is a **Refuse : Skill Execution Reply** message, the **Negotiation Process Behaviour** for this Skill Requirement will be restarted and the **Execution Process Behaviour** will end.

Else, if a positive response is obtained, it status can be "**Executing**", meaning the agent will immediately execute the skill or "**onWaitingList**" which means the agent cannot execute the Skill at the moment.

If a "**onWaitingList**" status is received, the execution request was put on hold by the executioner agent. Following the **Request Execution Confirmation Protocol** a **Query-If : Execute Skill** message informing it is able to execute the Skill is then expected. A timeout for this message to be received is then enabled. If it is not received, the initial **Accept-Proposal : Skill Execution** message will again be sent and the entire process repeated. Else, if the reply message is received, a **Confirm : Execute Skill** is sent and the timeout for a reply confirming the Skill execution enabled.

If an execution confirmation status other that "**Executing**" or "**onWaitingList**" is received, an error has occurred. However, maintaining the model simple, how to deal with errors is not considered as it would increase the control time response.

When the Confirm execution message is then received informing the Skill is being executed, periodic status messages will be exchanged with the agent performing the execution with the purpose of knowing how the execution is progressing. This will be made until the Skill termination message is then received. This will follow the defined **Execution Protocol**. The **Product Agent Skill Execution Status Behaviour** overview can be seen in Figure 6.19.

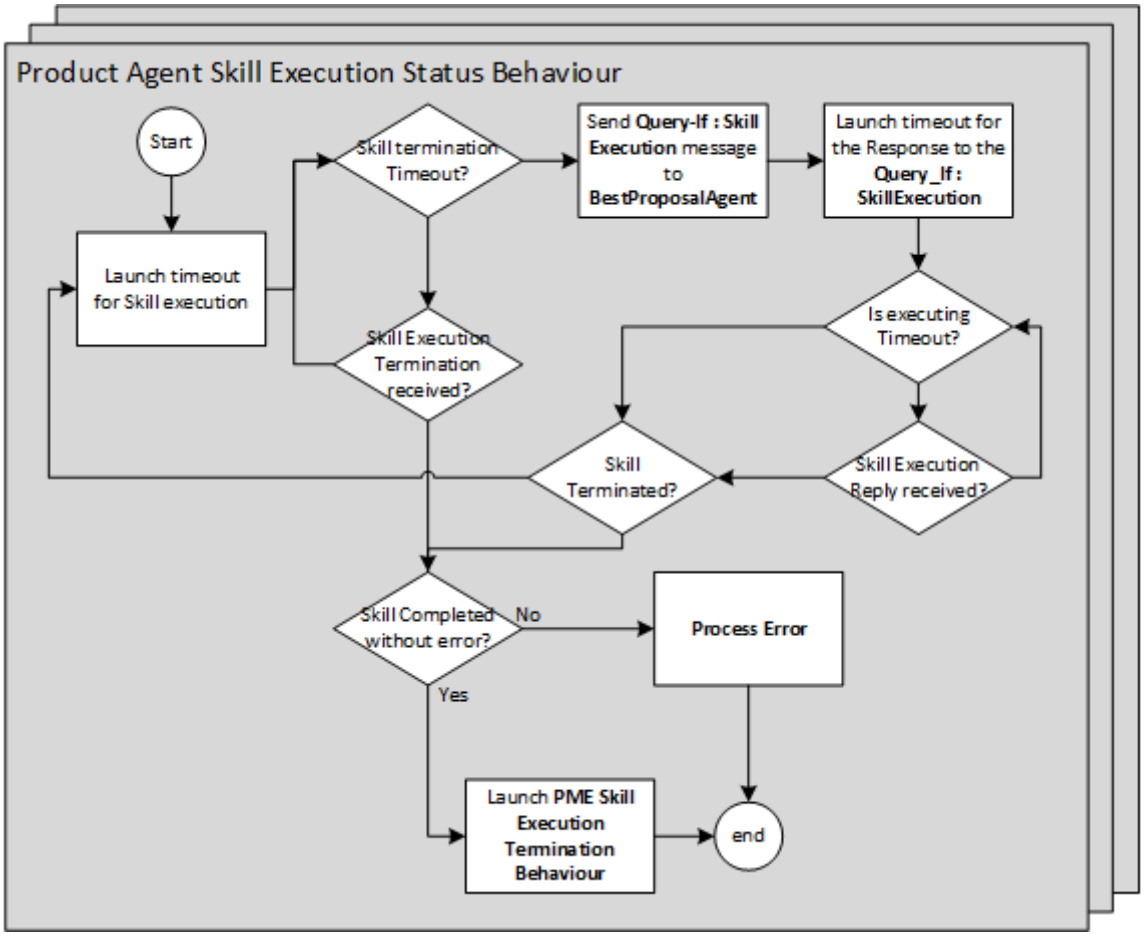

Figure 6.19 - Product Agent Skill Execution Status Behaviour algorithm

A timeout for the **Skill Execution Termination** message to be received will be launched. If this message is received, the timeout will be disabled and the termination of the Skill processed. This will consist in checking if the Skill terminated with errors or not, and actuate accordingly. If this message is not meanwhile received, a **Query-If : Skill Execution** message is sent to the RA. Again, a timeout will be launched for the response to be received. If a response is received, or the timeout expires, a check if the Skill has terminated will be made. If the Skill is still executing, the initial timeout will be set and the process repeated. Else, if the Skill has finished with no errors detected, the **PEM Skill Termination Behaviour** will be launched and the **Skill Execution Status Behaviour** terminated. For the event of errors occurring, it is important to establish a method to recover. However, this goes beyond the scope of this work.

This PA internal definition will not consider other options after the RA to perform the assembly process has been chosen. It will wait for it to execute the desired process. For simplicity reasons, search for other agents will not take place unless the chosen RA will send a **Refuse : Skill Execution Reply** message for the Skill execution, or if for some reason the RA no longer exists in the system.

6.4.3.4 *Skill Negotiation Execution Strategies Definition*

Several skill negotiation execution strategies were considered aiming to reduce the negotiation time resulting in the overall system execution time reduction. The Negotiation Process was defined before having as a goal the identification of the agent most suitable to perform a Skill, and establish a collaboration to execute the skill with this agent. The Skill Negotiation process might play a critical part in the performance of the system, since it adds to the time for the execution of a skill. This means that the complete execution sequence will sum the Negotiation and Execution processes to the total time it takes. If one process could be reduced, an improvement on the total time could eventually be obtained. To attend to that, three negotiation strategies were analysed. Their model can be seen in Figure 6.20.

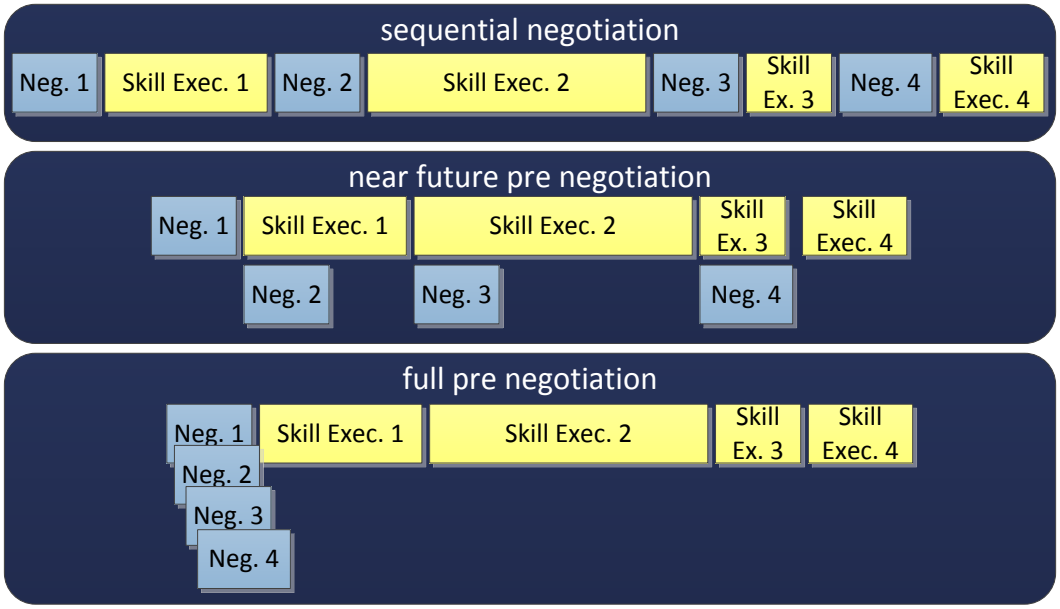

Figure 6.20 - Skill negotiation strategies overview

The first strategy will be the most common to be used, the "**sequential negotiation**". A normal negotiation strategy will be to always negotiate a Skill Requirement when it has all the necessary conditions to be executed. This strategy will allow an almost real time decision making process, as it gets the available agents along with their capabilities for the execution when those are indeed needed. It is most suitable for systems constantly changing, where for instance agent conditions to perform skills are fluctuating or even modules removed or added on regular basis. On the other hand, this strategy is also the heaviest in terms of system performance since it adds the Negotiation and Execution, and, as mentioned before, the negotiation could be a complex and time consuming process.

Other considered strategy will be the "**near future pre negotiation**". In this case, the next Skill Requirements in the product workflow will begin its Negotiation Process while their precedent ones are in their Executing Process. This will provide a composed solution for

systems in constant change while improving the overall performance as the negotiation times will be highly attenuated due to the parallel Negotiation and Execution Processes. Yet, the last part of the negotiation process, by sending the execution message to the chosen agent to perform the Skill, will only start when all its precedence constraints are met.

The last considered strategy will be the "**full pre negotiation**". In this strategy all the Skill Requirements will start their Negotiation Process immediately, however, like in the previous strategy; triggering of the execution of the Skill will only start when all the precedencies have been met. This strategy will be the less sensitive to system variations as all the Negotiation Processes are made in the beginning, however it will minimize the negotiation time during the triggering process, launching the Execution Process as soon as it is possible.

The negotiation strategies can be associated to Precedence Execution Methodology algorithm as seen in Figure 6.21.

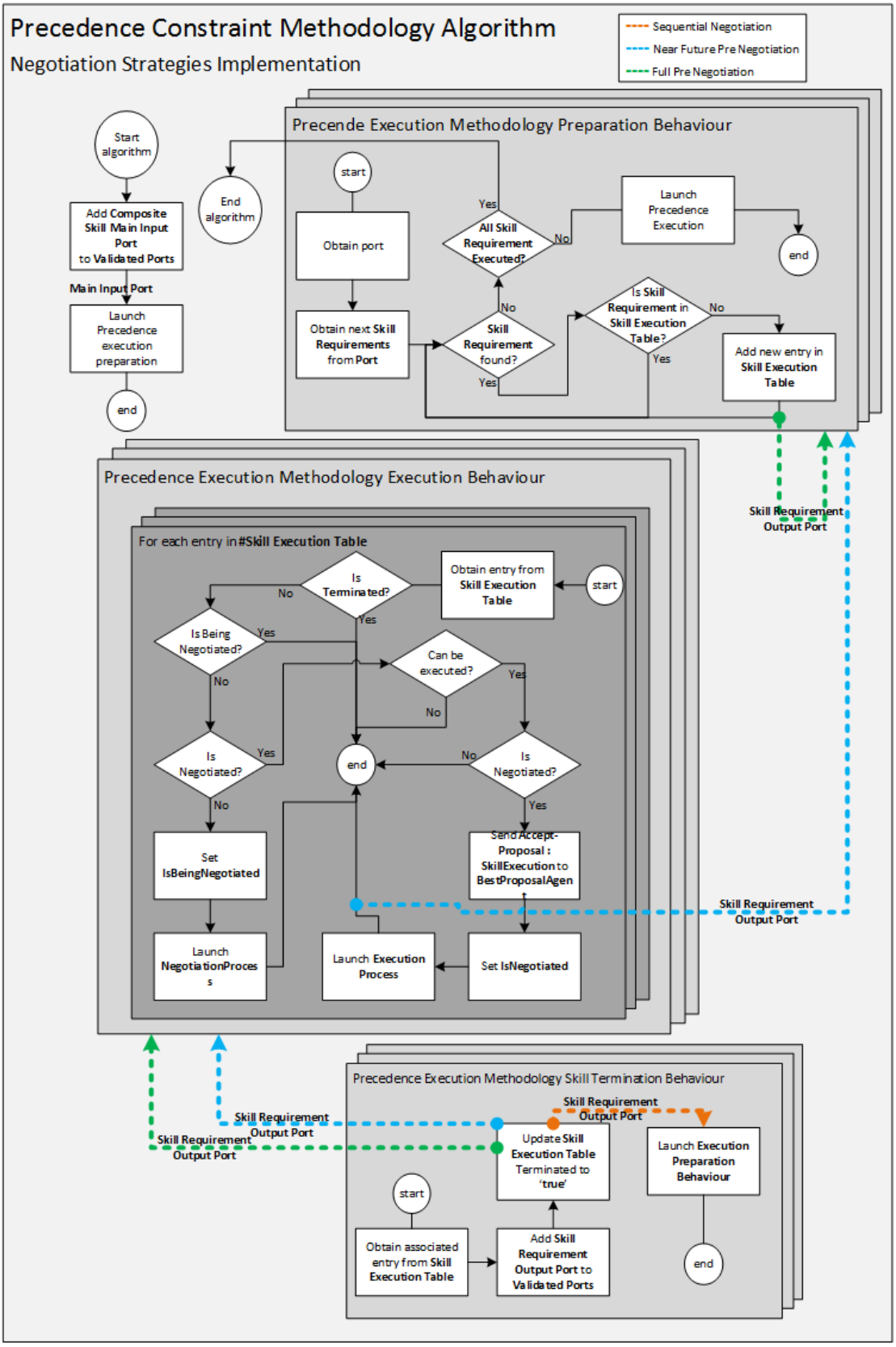

Figure 6.21 - Negotiation Strategies Implementation algorithm

In the figure, the Precedence Execution Methodology algorithm is again displayed. However, extra coloured links were added to show how each individual strategy was implemented. Each dotted line will represent a negotiation strategy implementation, and are only considered one at a time.

To implement the Sequential Negotiation strategy, shown in orange, the normal PEM behaviour will be followed. The blue dotted line will represent the Near Future Pre Negotiation implementation, launching the Preparation Behaviour to negotiate the next Skills while the previous ones are still being executed. The last is the Full Pre Negotiation represented by the dotted green line. In here, the Preparation Behaviour is launched immediately for each of the Skill Requirements having the execution process only being made when all precedencies for that Skill have been met. Having these negotiation strategies enables the evaluation of the Negotiation Process on the overall system performance.

6.5 Chapter Summary

This chapter provides the description of the proposed Light Agent Control Architecture for MAS. It only provides the minimum agents required so it can be deployed and control a basic MAS. However, some of the agent functionalities could be improved as they behaviour models can be easily extended. This will allow the existing agents to evolve and be adjusted to more complex control architectures.

With this simple architecture, the required performance studies for these agent environments control architectures can then be made. How they behave and operate can then be analysed, which can lead to the necessary improvements to be made. As the control architecture is kept simple, the obtained results could then be seen as the basis for the performance for such systems, since as the control architecture gets more complex and elaborate, the control mechanisms will eventually become more time consuming.

These agent control architecture can provide the means to make a small, however helpful contribution to the research being made around the control architectures for MAS.

7 Validation

7.1 Introduction

The proposed models validation is obtained using a representative MAS setup identified later in this chapter. This setup will be used to run a set of experimental scenarios in order to access the models functionality. This chapter will contain the achieved results along with its analysis.

7.2 Validation Scenario Setup

In order to validate the work presented in this thesis, a micro scale assembly station will be used. The station is part of the SMC HAS 200 assembly system which is composed by set of hardware modules. These will be using a number of electro-pneumatic actuators and a number of sensors to execute the assembly capabilities. This station is presented in Figure 7.1.

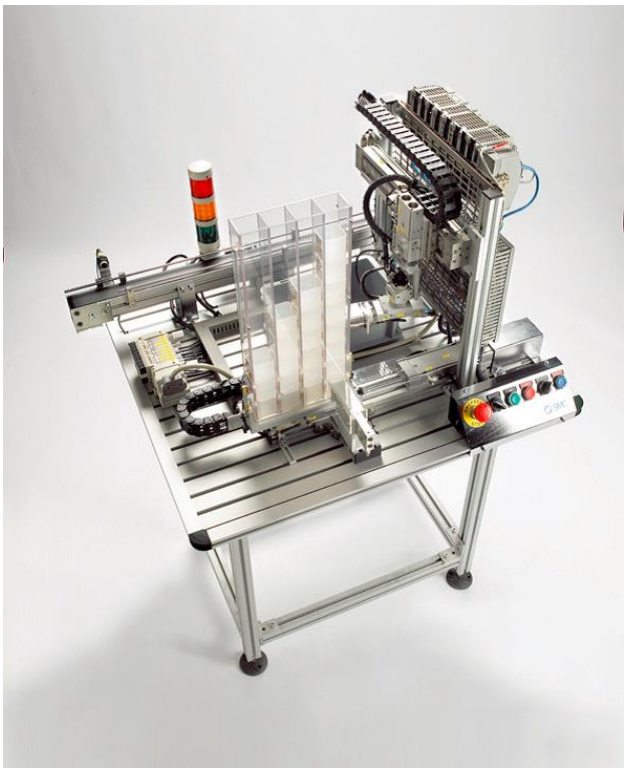

Figure 7.1 - SMC HAS-200 Station

The station control is delivered through an industrial controller (Beckhoff CX-1030) which is running embedded Windows XP operating system. The controller emulates soft PLC's and can guarantee a scanning cycle of up to 50 µsec by dedicating processing power on the PLC software (TwinCAT). The station is delivering a number of parallel cyclic processes.

The station objective is to deliver a box full of components (product). This is achieved by performing a number of assembly capabilities (skills). The skills are executed in a sequence, starting with feeding a box in the working space. Then the box is picked up from a manipulator that through a pick and place process takes it for a bar code scan. Then the box is taken to the filling area and the filling process is carried out. After that the box is placed on a conveyor belt to be dispatched and the station is initialised through the reset skill. This constitutes the station's workflow that is presented in Figure 7.2.

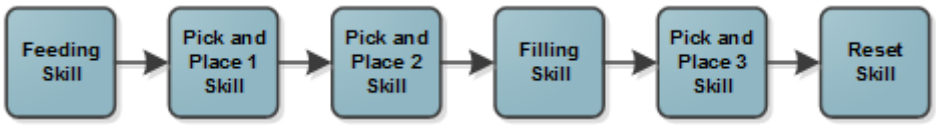

Figure 7.2 - Stations workflow overview

Every Skill is executed through a subroutine in a combination of Sequential Functional Chart (SFC) and structure text (ST) programming language. In order to deliver the process, both PLC code, Java and agent based control code are implemented. The interface with the PLC environment is achieved through the TwinCAT Java communication library which uses ADS [31].

7.3 Experimental Scenarios Definition

To validate the models, different experimental scenarios are proposed using the previously defined MAS setup. These scenarios will be executing the stations workflow. The time for each individual skill execution is taken, as well as the total time taken to complete the process. The first scenario is based on a PLC execution and will be used to benchmark the performance of the system. The second scenario is a hybrid approach of PLC and java control without using agents. The Skills execution by the PLC is triggered by java using the chapter 4 proposed model. The third scenario is also a hybrid approach joining PLC and the proposed light control architecture model defined in chapter 6. This scenario will use the Capability Dissemination Agent (CDA) defined in chapter 5 as the system Skill repository. In order to assure the reliability of the experiments, each will be conducted 10 times and the average time taken for the performance analysis purposes. The last experimental scenario will be used to assess the CDA re-adaptation ability against system changes.

7.3.1 PLC Benchmark

The PLC code was optimized and timers introduced in order to measure the execution time of every skill and also the overall process. The stations workflow was completed in 23745.2ms with a standard deviation of 26.21ms. Has seen in the literature, the PLC is known to provide the best possible result in the control of assembly systems. This is important since it establishes the base line for the comparison for the hybrid approaches.

7.3.2 High Level Hybrid Control Benchmark

The second experimental scenario is a hybrid approach of PLC and Java based control without the use of agents. The subroutines are triggered in sequence in order to subtract the agent control logic and assess the java control implementation. The communication between the JAVA and the PLC is done through the TwinCAT IO control. The runtime values are stored in a text file in the end of each run, which are then used to measure the average. In the chart of Figure 7.3, the values for this experimental scenario are displayed next to the obtained in the previous experimental scenario.

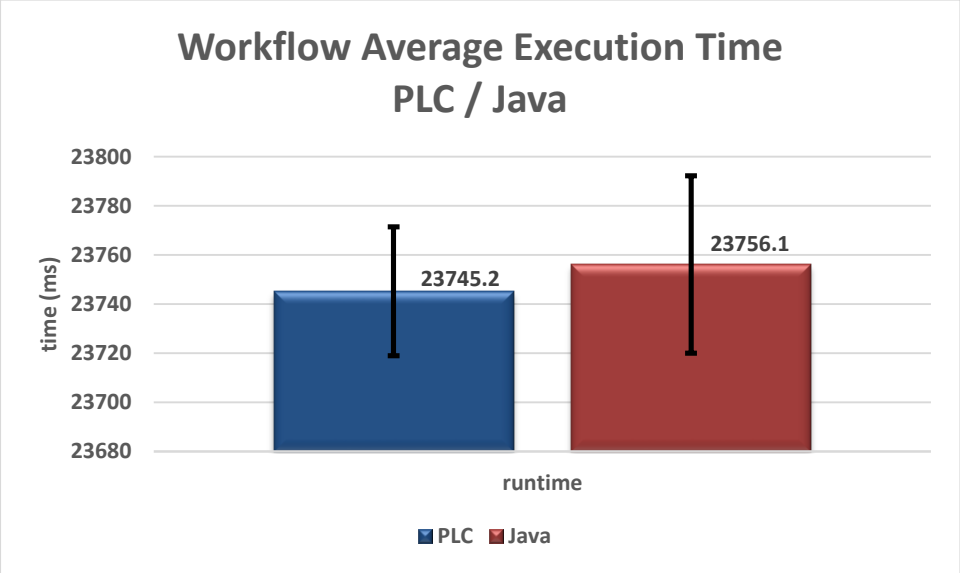

Figure 7.3 - PLC / Java assembly execution runtime

This experimental scenario validates the interaction model proposed in chapter 4. Comparing the java control with the PLC, it took an average of 10.9ms more to complete the process with a standard deviation of 36.13ms. Comparing it with the standard deviation of 26.21ms obtained in the previous scenario, a difference of 9.92ms. Despite the fact that this is a system result, this is an indication that the proposed hybrid solution delivers a reasonable performance.

7.3.3 Light Agent Control Architecture Validation and Performance Analysis

The agent control implementation is done based on the agent control architecture model proposed in chapter 6. It will be created using the JADE platform [68] and programmed in Java. The communication between the agents and the PLC is also achieved through the TwinCAT IO control. The light agent control architecture model implementation was embedded in a product workflow definition tool. This tool was developed to be used in the IDEAS project [8] which highlights the relevance of the control performance analysis this model is designed for. The tool user interface can be seen in Figure 7.4.

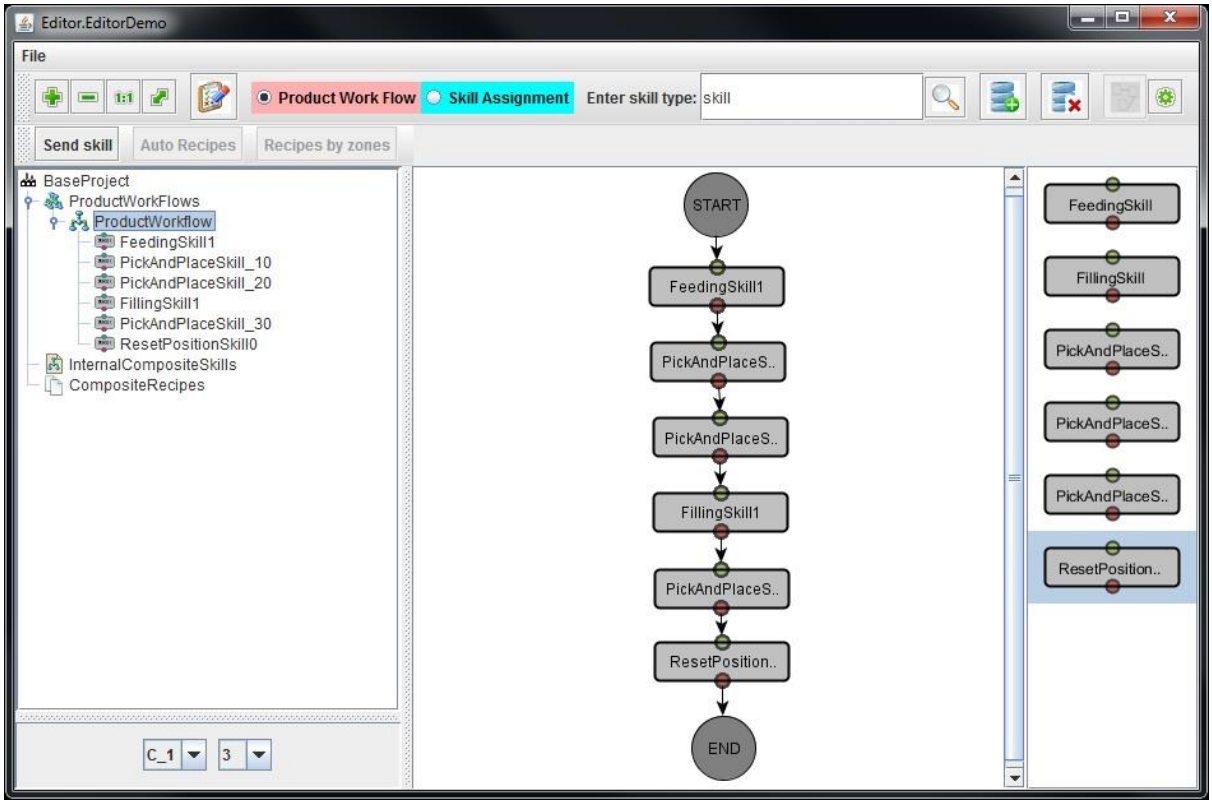

Figure 7.4 - Product agent workflow definition tool

This tool, among others, allows to define the Product Agent (PA) workflow along with its need initialization variables and also to launch it into the agent control environment. The Resource Agents (RA) are deployed on the system start up, according to the equipment modules of the system, namely there are 3 RA agents. It is important to note that some agents will be responsible for the execution of multiple Skills in the system. When the PA terminates all the skills in its workflow, it will save a file containing all the skill execution information needed to for the time results to be analysed. The agents are deployed through JADE and will run on a separate computer from the PLC.

The Capability Dissemination Agent (CDA) proposed in chapter 5 will also be launched on the system start up. Each RA will register its Skills on the CDA while the PA will use it to know which RAs can execute any required Skill. Figure 7.5 provides an overview of the launched agents.

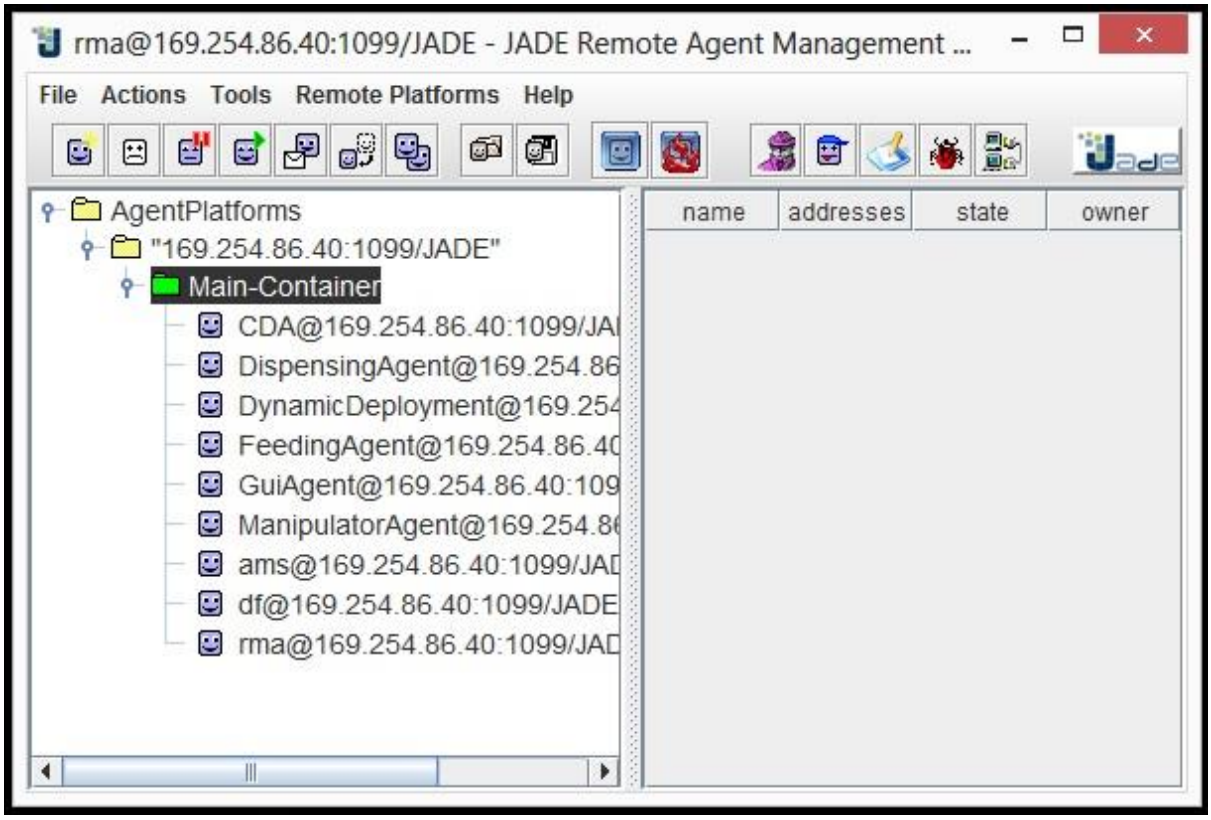

Figure 7.5 - JADE main container overview

In the picture, besides all the necessary Jade agents, there is the DispensingAgent which is responsible for the Filling Skill, the FeedingAgent responsible for the Feeding Skill, the ManipulatorAgent responsible for the Pick and Places and the Reset Skill; and also the CDA.

Due to the small number of agents in the experimental layout, the communication between the RA and the PA is expected to be very short. Realistic solutions tend to be more complex, thus in order to achieve a realistic negotiation process, a delay was introduced in the negotiation process to emulate the decision making process of the agents. More specifically, the delays are placed on the RA, for emulating physical processes, and the PA agent after the skill request is received and the before the acceptance message is sent, for emulating multiple negotiations. This can be seen in Figure 7.6.

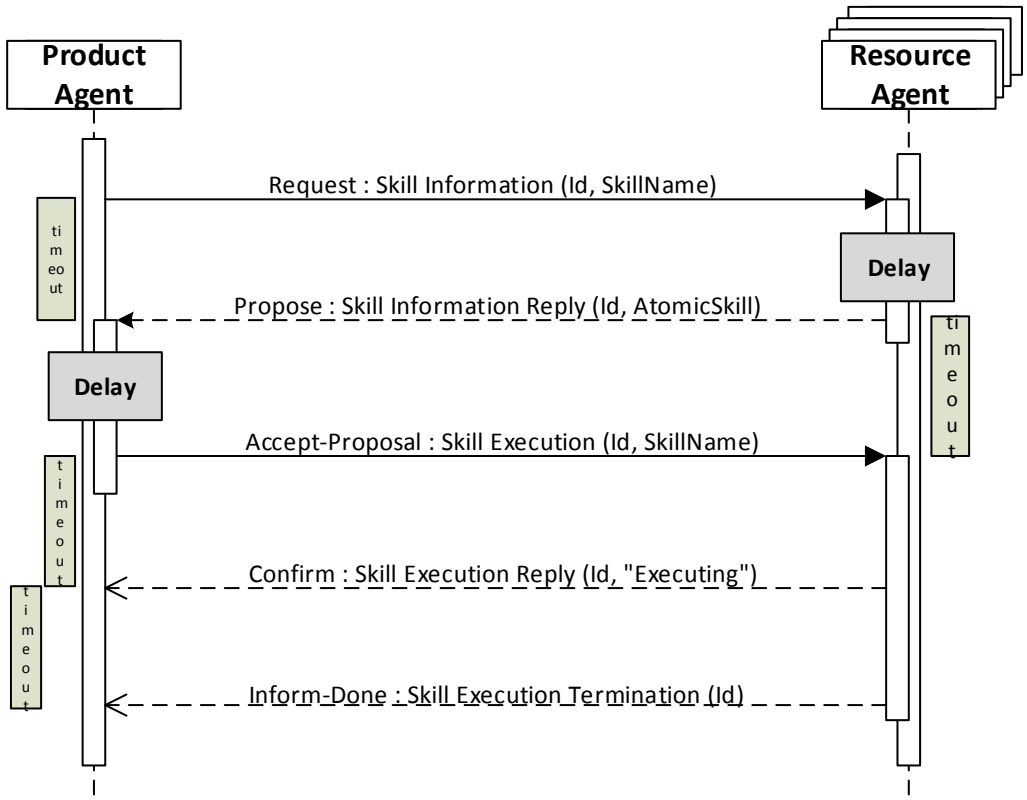

Figure 7.6 - Agent communication overview with delays

To analyse the effect of the delay, the experiments are run with a delay of 20ms, 50ms and 100ms as well as without it.

The Skill negotiation will play a critical part in the performance of the system, particularly when the decision making process of the agents increases in complexity. Therefore it is important to have alternative negotiation execution strategies to be able to ensure that the architecture is able to deliver a good performance. These negotiation strategies were defined in chapter 6. This experimental scenario will be sub divided in three experimental setups, accessing each negotiation strategy performance. The sniffer tool provided by JADE will be used to validate each communication implementation. This tool shows the message sequence between the agents.

The skill negotiation strategies will now be validated. For simplicity reasons, only the first two initial skill execution will be analysed. The first will be highlighted with a red vertical line, and the second in blue. Also, the initial RA skill registration in the CDA will be highlighted by a green vertical line. The skill execution is divided in several exchanged messages that respect the protocols defined in the chapter 6 of this work. Also, the CDA skill registration and query messages will respect the protocol defined in chapter 5. Figure 7.7 shows the internal CDA tables after the RAs skill registration occurs.

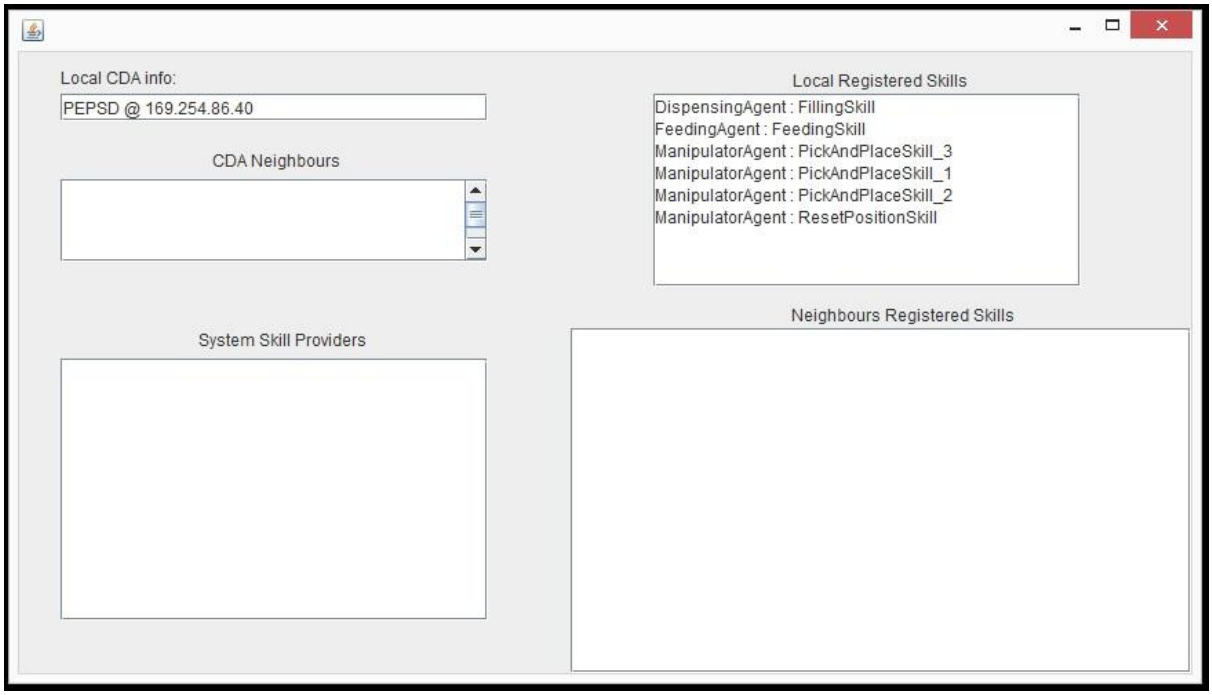

Figure 7.7 - CDA local registered skills overview

Figure 7.8 shows the exchanged messages during the skill execution using the Sequential Negotiation strategy.

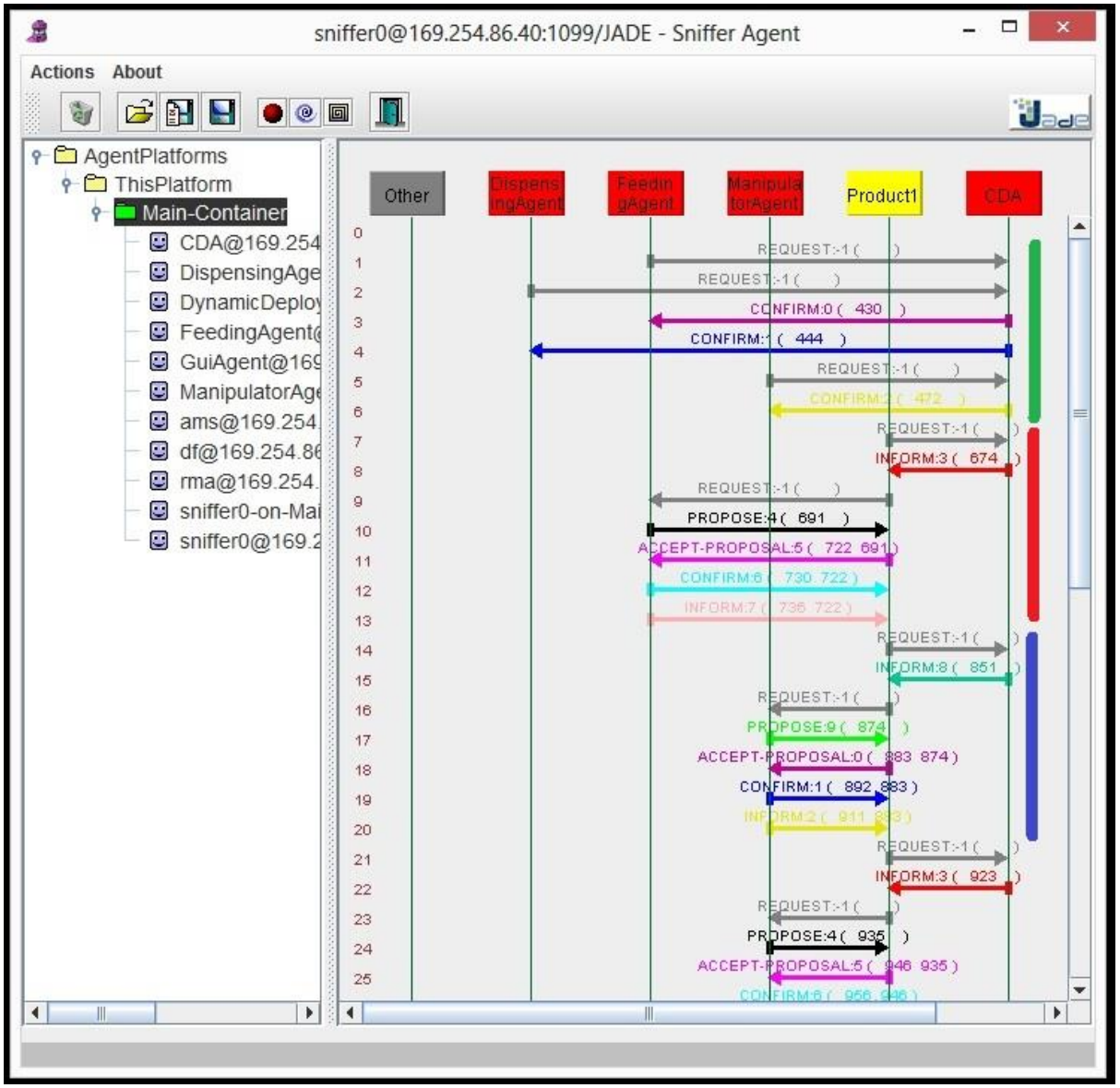

Figure 7.8 - Sequential negotiation exchanged messages overview

The initial RA skill registration is made when the RA are launched. Next the product is activated and it will query the CDA for the agents to do the skill. It will then use the protocols to trigger the skill executed by the respective RA. When the skill execution ends, as expected, the agents to execute the next skill are queried to the CDA. After this, the next skill will be negotiated and executed.

Figure 7.9 shows the exchanged messages during the skill execution using the Near Future Pre Negotiation strategy.

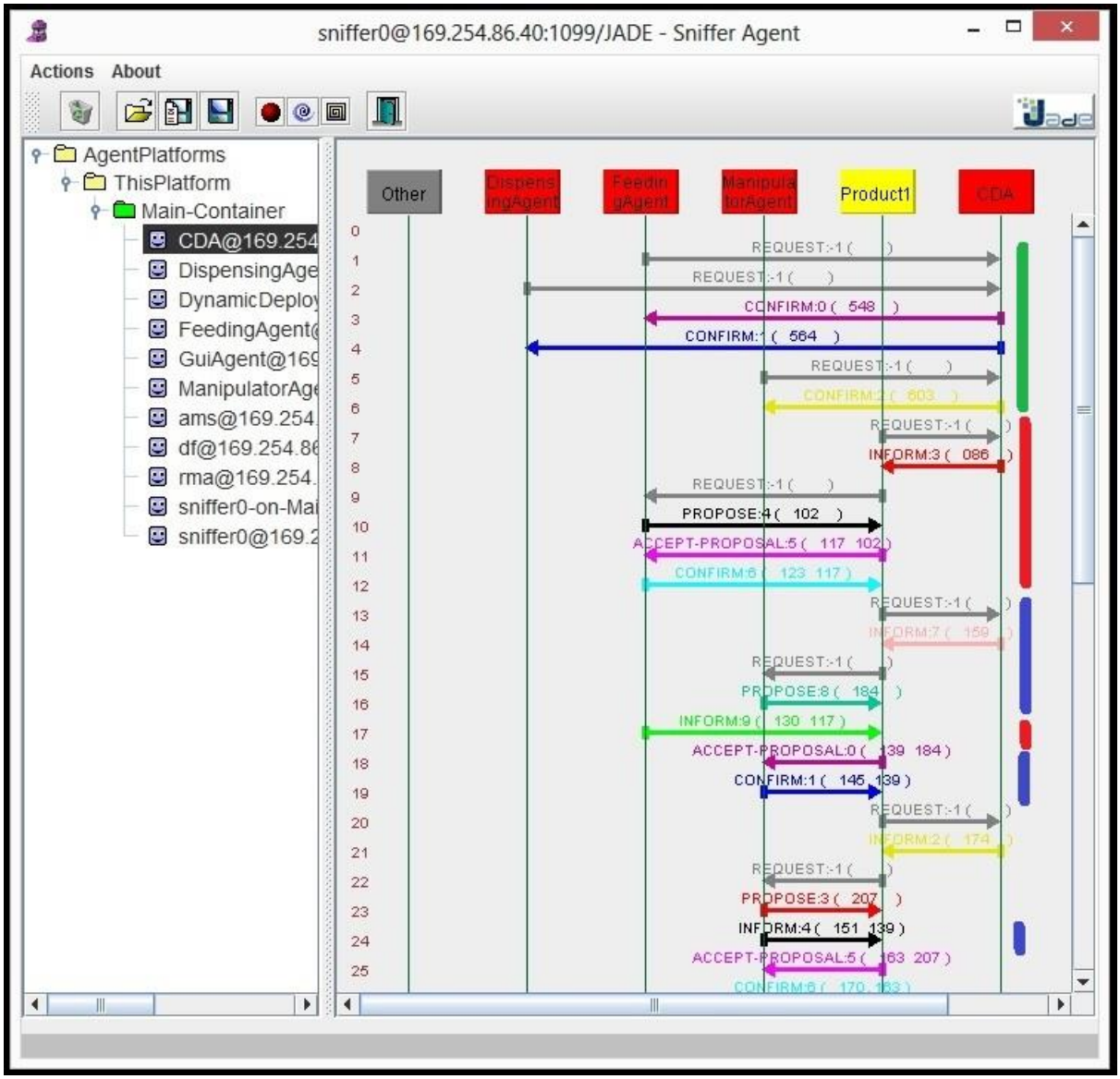

Figure 7.9 - Near future pre negotiation exchanged messages overview

Again, initially the RA will register their skills in the CDA. Next, the PA is activated and it queries the CDA for the agents to execute the first skill. Its negotiation is then started. When a message is sent by the RA informing the skill will be executed (CONFIRM message), the next skill will start to be processed. The PA will query the CDA for the agents to perform it, and will start the negotiation process. However, the next skill execution will only start once the first one terminates.

Figure 7.10 and Figure 7.11 show the exchanged messages during the skill execution using the Full Pre Negotiation.

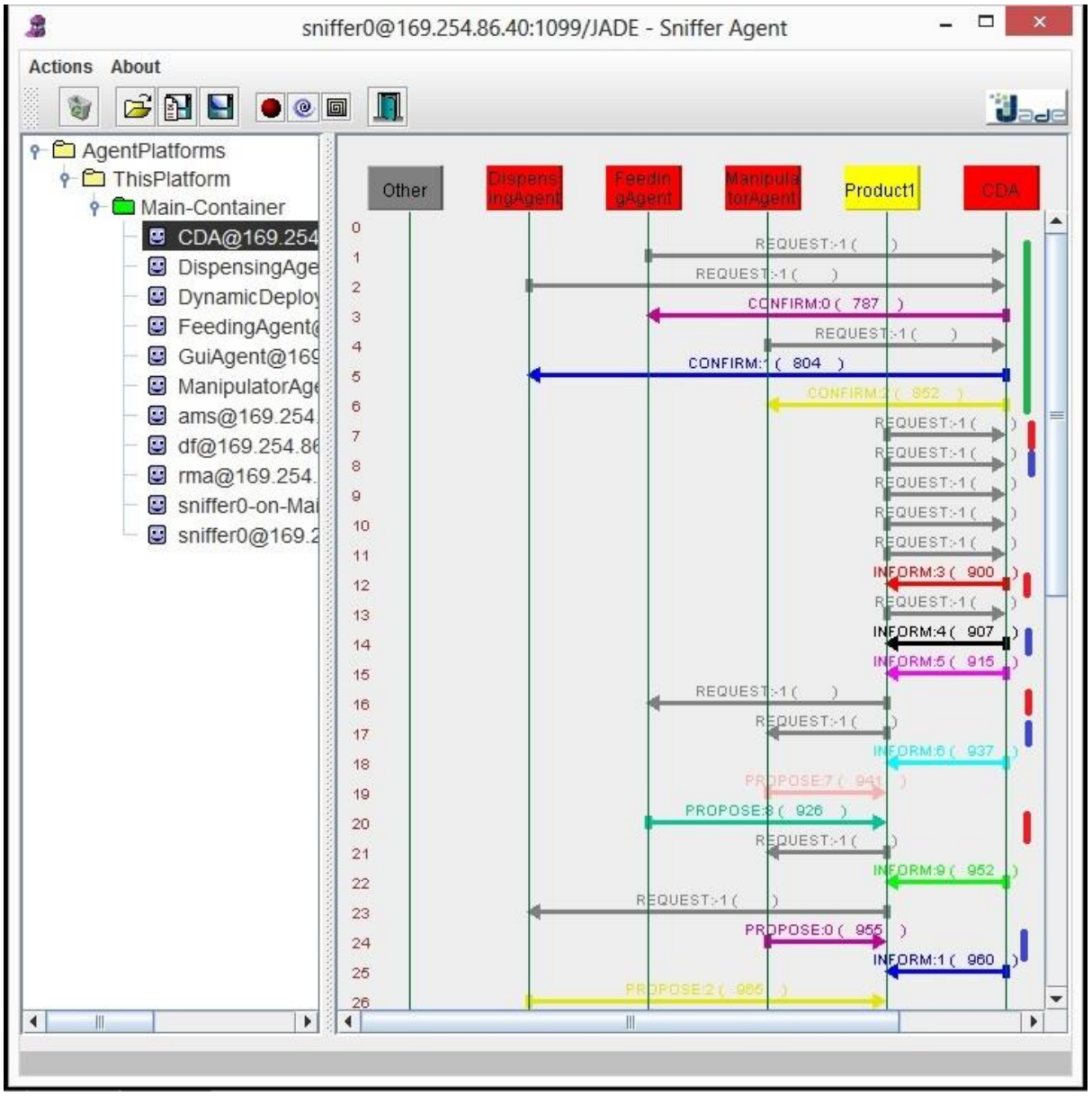

Figure 7.10 - Full pre negotiation exchanged messages overview (part1)

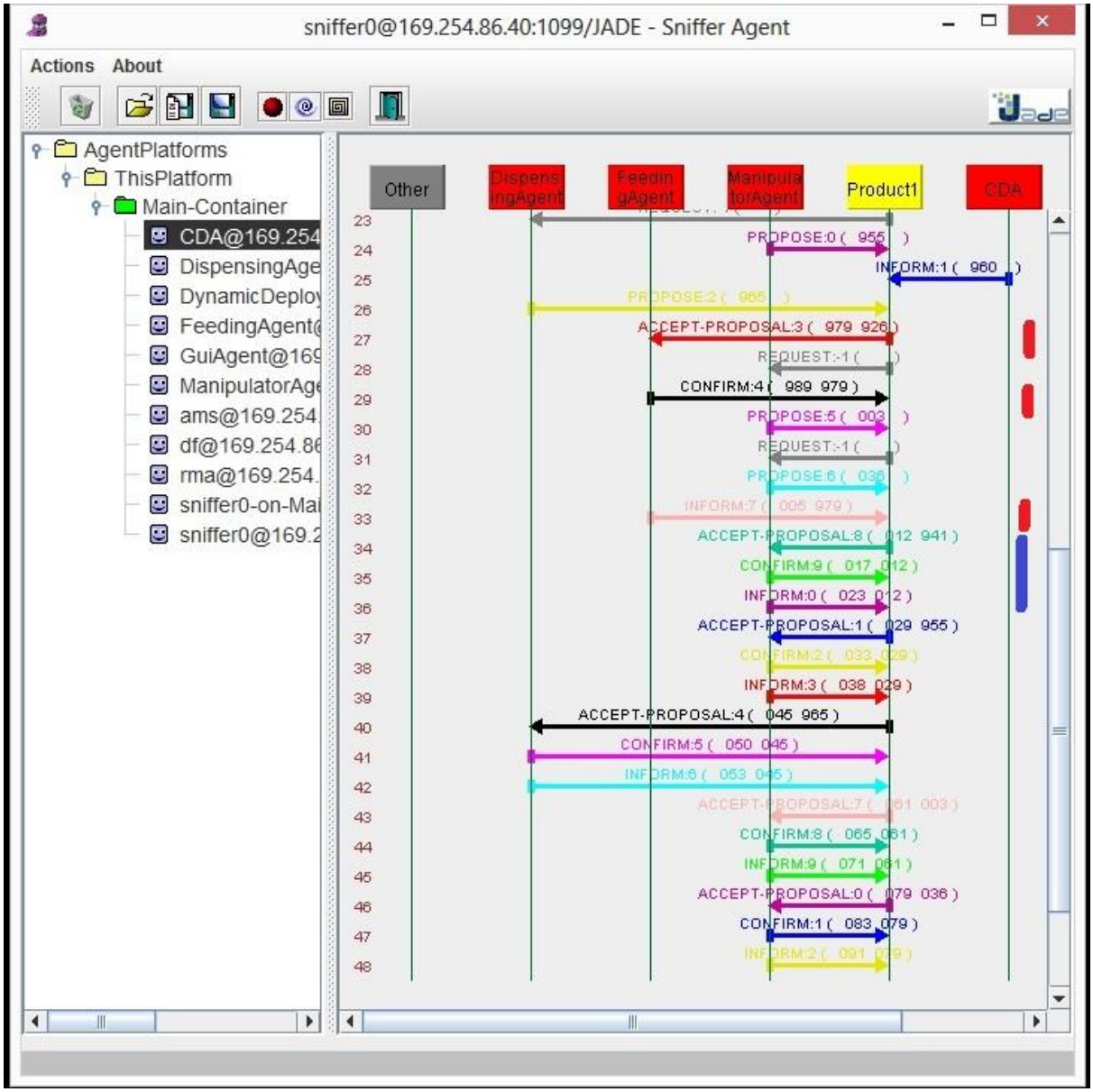

Figure 7.11 - Full pre negotiation exchanged messages overview (part2)

A seen in the previous figures, the entire message sequence between the agents can be seen. Besides the initial parallel negotiation for all the skills, only the first skill execution is triggered. The next skill is then only executed once the first one finishes. This happens until all the skills have been executed and the product is assembled.

This validates the proposed negotiation strategies models and also the Precedence Methodology. Having the models validated, the results for each defined experimental scenario will be analysed. These can be seen in Figure 7.12.

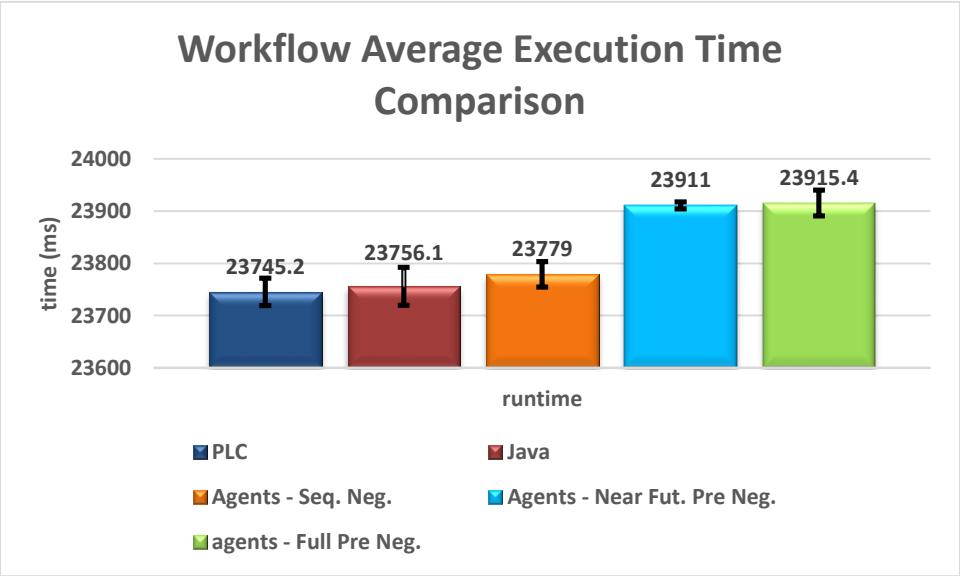

Figure 7.12 - PLC / Java / Agents assembly execution time

Relating the previously obtained values for all experimental scenarios so far, using the Sequential Negotiation strategy took in average more 33.8ms against the PLC obtained values. Using the Near Future Pre Negotiation and Full Pre Negotiation strategies, this value increases to 165.8ms and 170.2ms accordingly.

As expected, using the PLC control provides the best performance times. Considering the hybrid PLC and agent control, an overhead regarding the previous experiments is seen. This can be explained due to the agent control logic and negotiations. The result is quite interesting when compared with the pure Java execution, since that value provides a quite stable value for the interaction of a high level programing language and the PLC.

The overhead for each negotiation strategy using the proposed delays will now be analysed. For each scenario, the total execution time measured by the PLC will be considered in each of the ten runs. Next, each individual assembly process time will be subtracted from it. The found times will give the negotiation overhead. Averaging these time values will give the comparison values to be analysed. The negotiation overhead for each negotiation strategy is shown in Figure 7.13.

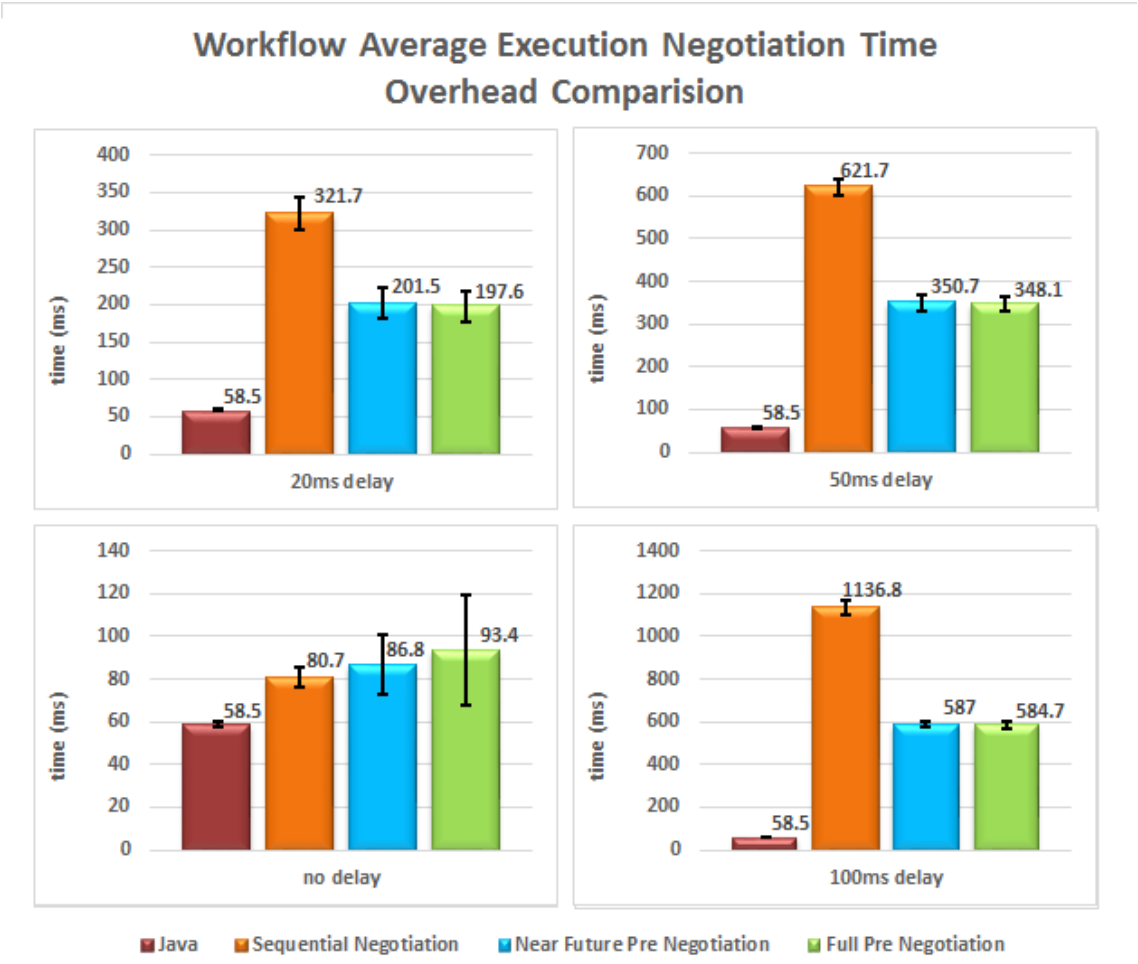

Figure 7.13 - Assembly execution overhead comparison

By comparing the negotiation strategies without introducing the delays, the biggest overhead is on average for the full process 93.4ms. This means that the average value per assembly process is 15.57ms. In fact if one discounts the java interface overhead of 58.5ms which will always be required for any interfacing solution, each assembly process overhead is reduced to 5.82ms. This means that the negotiation impact per assembly process is quite low having a simplistic negotiation implementation.

Analysing the results with delays in the negotiation, which emulates a more complex decision making processes, one can see the inversion on the best negotiation execution strategy. This suggests that there is a turning point which determines when one can alternate from one approach to the other depending on the decision making processes overhead.

The comparison between the “near future pre negotiation” and the “full pre negotiation” strategies shows that there is very little difference between the two approaches. This is not at all unexpected since all assembly processes take significantly more time for completion than any

negotiation. In fact one would not expect to use a full pre negotiation strategy, unless processes are extremely fast and the decision making process is quite complex.

Despite the fact that this is a system result, this is an indication that the proposed hybrid solution delivers a reasonable performance. One unexpected result is the fact that the best performance is achieved with the sequential negotiation approach. However, if analysed closely this results can be justified by the overhead of creating multiple agent behaviours in the system. This implies there is a significant performance cost to create multiple agent behaviours, which should be considered when choosing the best negotiation execution strategy. This also means that the best negotiation execution strategy is dependent on the complexity the decision making processes of the agents involved in the negotiation.

The interfacing and communication overheads do not pose a large impact on the performance of the system. Having the hybrid control using Java and the PLC also provides a fine performance having almost no delay compared to the PLC. Finally, it is important to state that the complexity of the decision making process will make or break any agent approach.

7.3.4 Capability Dissemination Agent Adaptability Validation

The initial validation for the CDA functionality was obtained in the previous experimental scenarios, as the CDA was used to store the system existing Skills, and diffusing then through the system. To validate the re-adaptation capabilities of the CDA, two stations equal to the one used in the previous chapters will be used along with an external computer terminal.

Two CDAs will be initially launched, one in each of the two workstations. The RA's used in the previous validations scenarios will also be launched for each one of the stations. Afterwards, another CDA will be launched in the external computer without any RA associated with it. Its sole objective will be to display the information stored in its internal defined tables, and with that, observe the CDA functionality. To obtain that, an interface to display these tables was created. When the external CDA is launched, as expected, no assembly capabilities are present in the tables. This can be seen in Figure 7.14.

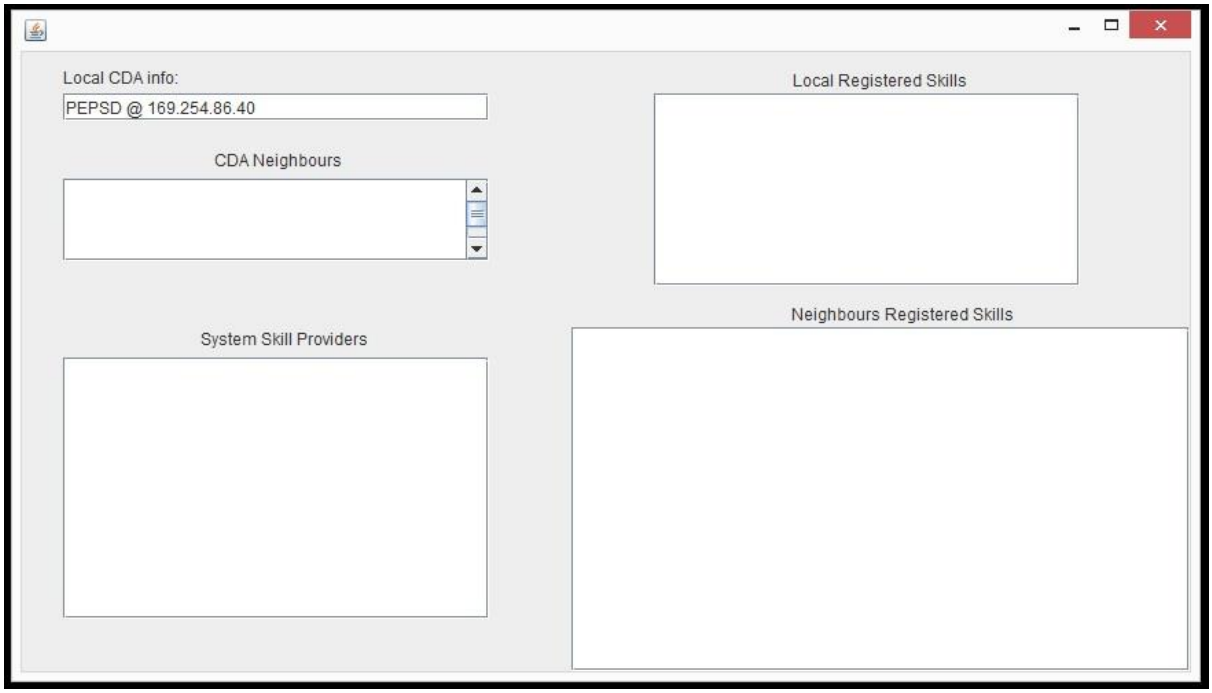

Figure 7.14 - CDA initial table information display

After the initial synchronization and information exchange, all the assembly capabilities registered in each individual station CDA are known by the external CDA. Also, their registered local skill providers are known. This can be seen in Figure 7.15.

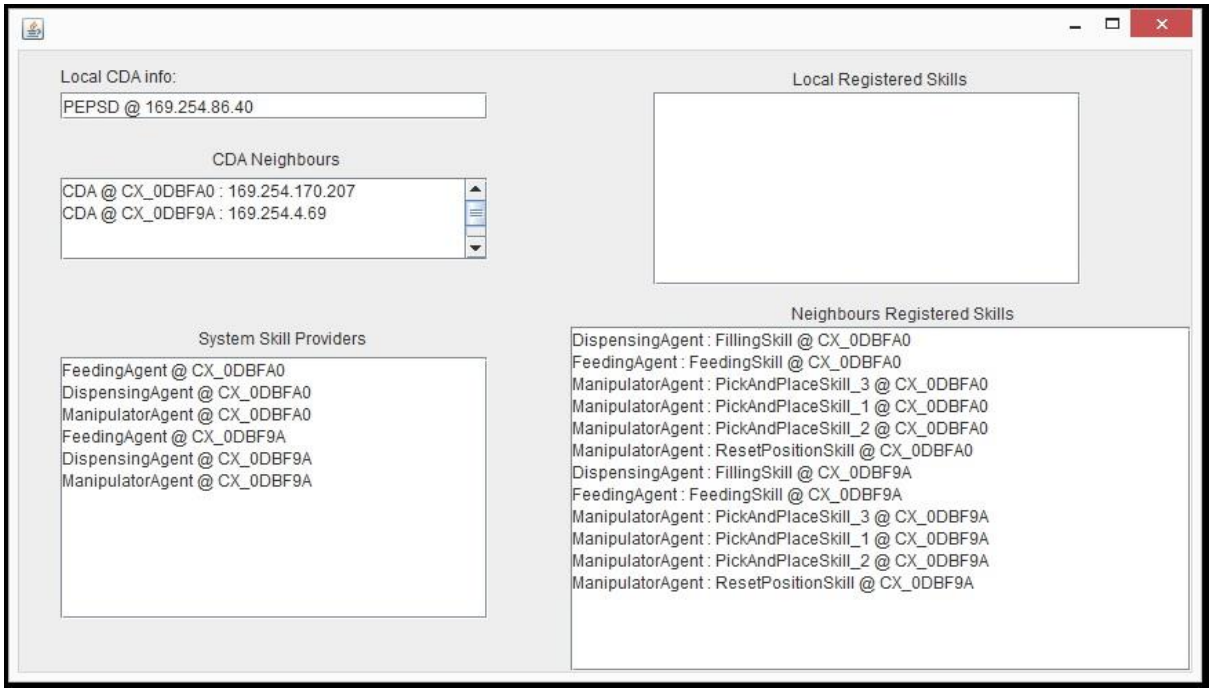

Figure 7.15 - CDA tables information display

As seen in the figure, all the system existing capabilities are known by the CDA. Next, the station located on 169.254.170.207 will be abruptly shut down. This will allow to confirm the CDA capability to readapt to partial system breakdowns. All the information regarding that CDA will no longer be available after the system synchronization. The information that will disappear is shown highlighted by a red rectangle in Figure 7.16.

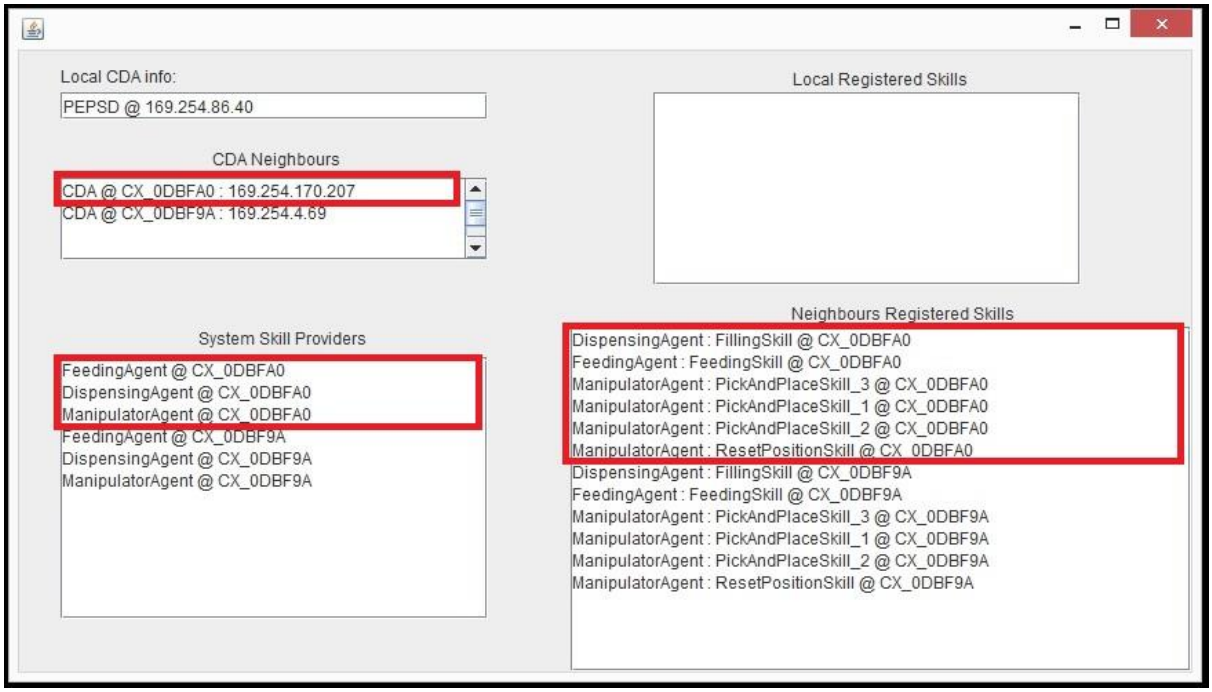

Figure 7.16 - CDA tables information display before neighbour CDA removal

The resultant system capabilities information after the CDA information exchange can be seen in Figure 7.17.

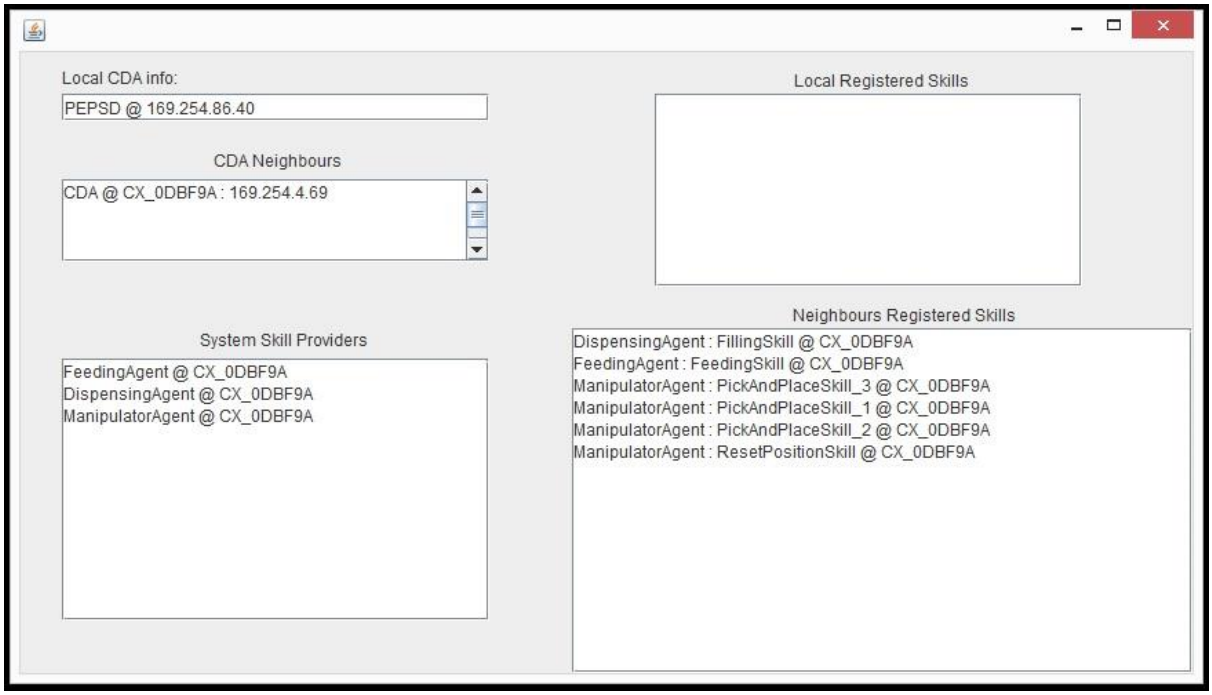

Figure 7.17 - CDA tables information display after neighbour CDA removal

The capabilities and skill providers associated with the removed station were removed from the CDA tables as expected. Next, the same station is reintroduced to the system. After the CDAs synchronization the tables' information can be seen in Figure 7.18.

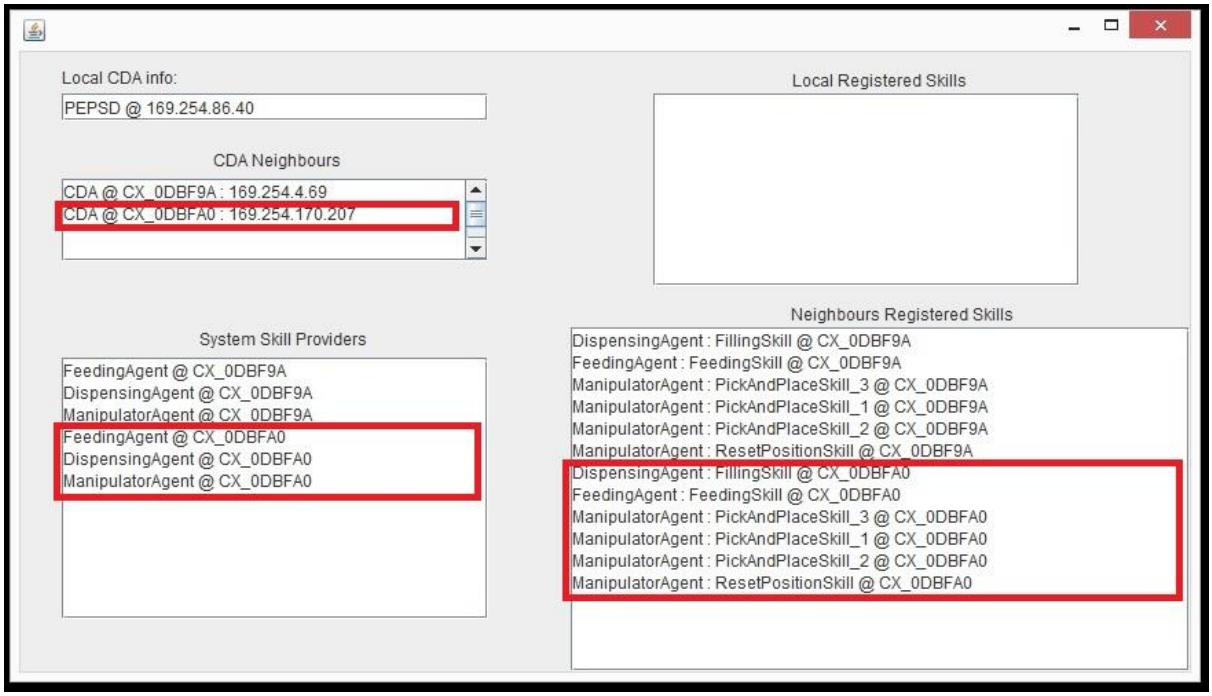

Figure 7.18 - CDA tables information display after neighbour CDA reconnection

As observed in the figure, highlighted by the red rectangle, the capabilities from the new introduced system were correctly propagated and its existence acknowledged by the external CDA.

This experimental scenario validates the proposed CDA model providing the assembly capabilities correct propagation and re-adaptation to changes. Also, de discovery mechanism fulfilled its task, as the neighbours know when a new CDA was introduced.

7.4 Chapter summary

In this chapter several experimental scenarios were run in order to validate this work proposed models. The experimental results were then displayed and analysed in order to conclude about the models performance. The proposed light control architecture was used to obtain real results so that a simple analysis about the viability of these control technologies could be produced.

8 Discussion & Future Work

8.1 Introduction

In this chapter the conclusions for the work contained in this thesis are presented. The chapter will also contain a perspective about the future work to be done.

This work targets to provide a contribution to the research being made around the control of MAS using agent architectures. This was made producing a model for a light agent control architecture to be deployed in a Modular Assembly System (MAS) and then access its viability and performance.

The interaction between the agent architecture and the PLC was first proposed in chapter 4. This allows that a right mechanism to execute the assembly capabilities is provided. Next, in chapter 5, a Capability Dissemination Agent (CDA) was defined to provide the agent architecture with a redundant and delocalized repository for all the Skills present in the system. This agent would then be used by all the other agents to inform and know about those existing Skills. Finally, the light agent control architecture model was defined in chapter 6. This model along with is agent behaviours and definition provides a light approach to a control architecture. This architecture aimed to provide a simple however robust control approach to be used in a MAS.

8.2 Knowledge Contributions

The proposed agent architecture follows the research being made about control architectures to be used in MAS. The architecture provides a light agent control logic allowing that performance tests can be made around the use of this technology in the MAS control. However light, the architecture was modelled to allow expandability and to new control logic to be added or modified.

The developed work helped in the publication of the next scientific article:

- Ferreira P., Doltsinis S., Anagnostopoulos A., Pascoa F., Lohse N.,"A Performance evaluation of Industrial Agents - A Benchmark against Programmable Logic Controllers", IECON 2013 - The 39th Annual Conference of the IEEE Industrial Electronics Society, Accepted

8.3 Future Work

During the CDA validation, the aim was only to assess its functionality. Its performance was not considered. The main limitation regarding the CDA may arise from its response times to service requests. The response times need to be studied and tested in order to completely analyse the practicality of the proposed CDA.

The CDA deployment strategy for the optimal performance needs to be well considered and analysed as it may help increase the overall system performance. Studies about the best deployment strategies have not been made. Future studies about the ideal deployment strategy need to be planned so that the contribution about the CDA improvement on the overall system performance can be obtained.

The light agent control architecture proposed in chapter 6 allows that performance analyses on this technology to be made. The results provide an insight on the behaviour of the agent environment and the importance of the decision making processes in the success of these approaches. Further work will focus on extending the complexity of the models and study in detail all negotiation execution strategies by creating more complex operational scenarios. The new studies will try to identify the point at which one should opt for each negotiation execution strategies. Also, introducing new agents to the system and expanding the system functionality and control logic can allow the architecture to be robust and fully deployable in MAS. These new agents

How to react to errors was considered during the definition of the model. How to act when these are detected needs to be also modelled and defined.

The usage of optimized timeout values in the proposed models can also help to increase the overall system performance. Measurements and tests are needed for these optimum values to be obtained.

8.4 Concluding Remarks

I've had the opportunity to come to The University of Nottingham, in the United Kingdom, to do a six month placement with the main objective of doing my masters dissertation. I've taken this journey because I believe it will bring me benefits not only professionally but also in a personal level.

In here, I came across a supporting environment for a student in the finals steps of his masters'. To attend to my main objective, I found a micro scale assembly line at my disposal. Using it, I could test and see real results on the studies being developed concerning MAS model.

My first objective in this new atmosphere was to find a challenging project that I would cherish to do and, at the same time, could provide the best possible contribute to the group projects. The research being made on MAS inspired me to contribute to it. This was made by designing a control architecture model to provide the means for simple performance analyses to be made. With it, a real validation of these system could be presented and reinforcement for their feasibility presented.

9 Bibliography

1. Shen, W., D.H. Norrie, and J.-P. BarthÃ¨s, *Multi-agent systems for concurrent intelligent design and manufacturing*. 2004: CRC press.
2. Jennings, N., N.R. Jennings, and M.J. Wooldridge, *Agent technology: foundations, applications, and markets*. 1998: Springer.
3. Ferber, J., *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. 1999: Addison-Wesley Reading.
4. Onori, M., et al., *The IDEAS project: plug & produce at shop-floor level.* Assembly Automation. **32**(2): p. 124-134.
5. *Ribeiro, L., J. Barata, et al. (2011). Self-organization in automation-the IDEAS pre-demonstrator. IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society, IEEE.*
6. *Ribeiro, L., R. Rosa, et al. (2012). IADE – IDEAS Agent Development Environment: Lessons Learned and Research Directions. CIRP Conference On Assembly Technologies And Systems (CATS 12), Ann Arbor.*
7. *Ribeiro, L., A. Rocha, et al. (2012). A product handling technical architecture for multiagent-based mechatronic systems. IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society, IEEE.*
8. *IDEAS, "Instantly Deployable Evolvable Assembly Systems".* 2010.
9. Nagel, R., R. Preiss, and K. Goldman, *Agile competitors and virtual organizations: strategies for enriching the customer*. 1995, New York, Van Nostrand Reinhold.
10. Arai, T., et al., *Holonic assembly system with Plug and Produce.* Computers in Industry, 2001. **46**(3): p. 289-299.
11. Babiceanu, R.F. and F.F. Chen, *Development and applications of holonic manufacturing systems: a survey.* Journal of Intelligent Manufacturing, 2006. **17**(1): p. 111-131.
12. Bi, Z., et al., *Reconfigurable manufacturing systems: the state of the art.* International Journal of Production Research, 2008. **46**(4): p. 967-992.
13. Koren, Y., et al., *Reconfigurable manufacturing systems.* CIRP Annals-Manufacturing Technology, 1999. **48**(2): p. 527-540.
14. Bi, Z., L. Wang, and S.Y. Lang, *Current status of reconfigurable assembly systems.* International Journal of Manufacturing Research, 2007. **2**(3): p. 303-328.
15. *Onori, M. (2002). Evolvable Assembly Systems - A New Paradigm? 33rd International Symposium on Robotics Stockholm.*
16. *Barata, J., M. Onori, et al. (2007). Evolvable Production Systems: Enabling Research Domains. International Conference on Changeable, Agile, Reconfigurable and Virtual Production. Toronto, Canada.*
17. *Ribeiro, L., J. Barata, et al. (2009). Evolvable Production Systems: An Integrated View on Recent Developments. 6th International Conference on Digital Enterprise Technology. Hong Kong, IEEE.*
18. Arai, T., et al., *Agile assembly system by â€œplug and produceâ€•* . CIRP Annals-Manufacturing Technology, 2000. **49**(1): p. 1-4.
19. Martin, M.V. and K. Ishii, *Design for variety: developing standardized and modularized product platform architectures.* Research in Engineering Design, 2002. **13**(4): p. 213-235.
20. Bi, Z. and W. Zhang, *Concurrent optimal design of modular robotic configuration.* Journal of Robotic systems, 2001. **18**(2): p. 77-87.
21. Ulrich, K., *The role of product architecture in the manufacturing firm.* Research policy, 1995. **24**(3): p. 419-440.

22. Blackenfelt, M. and R.B. Stake. *Modularity in the context of product structuringâ€“a survey*. in *The Second Nord Design Seminar, KTH, Stockholm*. 1998.
23. Bi, Z., et al., *Development of reconfigurable machines.* The International Journal of Advanced Manufacturing Technology, 2008. **39**(11-12): p. 1227-1251.
24. Alsterman, H. and M. Onori. *Process-oriented assembly system concepts: the Mark IV approach*. in *Assembly and Task Planning, 2001, Proceedings of the IEEE International Symposium on*. 2001. IEEE.
25. Gaugel, T., M. Bengel, and D. Malthan, *Building a mini-assembly system from a technology construction kit.* Assembly Automation, 2004. **24**(1): p. 43-48.
26. BoÃ«r, C., et al., *Integrated computer aided design for assembly systems.* CIRP Annals-Manufacturing Technology, 2001. **50**(1): p. 17-20.
27. Chen, I.-M., *Rapid response manufacturing through a rapidly reconfigurable robotic workcell.* Robotics and Computer-Integrated Manufacturing, 2001. **17**(3): p. 199-213.
28. Pedro Ferreira, N.L., Margarita Razgon, Piero Larizza, Giuseppe Triggiani, *Skill Based Configuration Methodology for Evolvable Mechatronic Systems.* 2012.
29. M. Kratochvíl and C. Carson, *Growing Modular.* Berlin: Spinger, 2005.
30. *Bolton, William. Programmable logic controllers. Access Online via Elsevier, 2009.*
31. Beckhoff, *Beckhoff Documentation. Available: http://www.beckhoff.com/english.asp?download/.* 2012.
32. Vyatkin, V., *IEC 61499 function blocks for embedded and distributed control systems design*. 2007: ISA-Instrumentation, Systems, and Automation Society.
33. Tichem, M., *Position report on flexible assembly automation.* Laboratory for Production Engineering and Industrial Organisation, Delft University of Technology, Landbergsstraat, 2000. **3**.
34. Nwana, H.S., *Software agents: An overview.* Knowledge engineering review, 1996. **11**(3): p. 205-244.
35. *Wooldridge, M. and N. R. Jennings (1995). "Intelligent Agents - Theory and Practice." Knowledge Engineering Review 10(2): 115-152.*
36. *Wooldridge, M. J. and N. R. Jennings (1994). Agent Theories, Architectures, and languages: A Survey. ECAI-Workshop on Agent Theories, Architectures and Languages, Amsterdam.*
37. Bernon, C., et al., *Engineering adaptive multi-agent systems: The adelfe methodology.* Agent-oriented methodologies, 2005: p. 172-202.
38. Cossentino, M., *From requirements to code with the PASSI methodology.* Agent-oriented methodologies, 2005. **4**: p. 79-106.
39. Henderson-Sellers, B. and P. Giorgini, *Agent-oriented methodologies*. 2005: IGI Global.
40. Onori, M. and J.B. Oliveira, *Outlook report on the future of European assembly automation.* Assembly Automation. **30**(1): p. 7-31.
41. Oliveira, J.A.n.B.d., *Coalition based approach for shop floor agilityâ€“a multiagent approach.* 2003.
42. Barry, J., et al. *NIIIP-SMART: An investigation of distributed object approaches to support MES development and deployment in a virtual enterprise*. in *Enterprise Distributed Object Computing Workshop, 1998. EDOC'98. Proceedings. Second International*. 1998. IEEE.
43. Fox, M.S., J.F. Chionglo, and M. Barbuceanu, *The integrated supply chain management system*. 1993, Citeseer.
44. Sadeh, N.M., D.W. Hildum, and D. Kjenstad, *Agent-based e-supply chain decision support.* Journal of Organizational Computing and Electronic Commerce, 2003. **13**(3-4): p. 225-241.
45. Shen, W., F. Maturana, and D.H. Norrie, *MetaMorph II: an agent-based architecture for distributed intelligent design and manufacturing.* Journal of Intelligent Manufacturing, 2000. **11**(3): p. 237-251.
46. Yen, B.-C. and O. Wu, *Internet scheduling environment with market-driven agents.* Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2004. **34**(2): p. 281-289.

47. Shen, W., et al., *Applications of agent-based systems in intelligent manufacturing: An updated review.* Advanced engineering INFORMATICS, 2006. **20**(4): p. 415-431.
48. Ferreira, P., *An agent-based self-configuration methodology for modular assembly systems*. University of Nottingham.
49. *Barata, J. (2003). Coalition Based Approach for Shop Floor Agility. Electrical and Computer Science Engineering. Monte da Caparica, Universidade Nova de Lisboa. PhD.*
50. *Van Brussel, H., J. Wyns, et al. (1998). "Reference architecture for holonic manufacturing systems: PROSA." Computers in Industry 37(3): 255-274.*
51. *Leitao, P., A. W. Colombo, et al. (2005). "ADACOR: a collaborative production automation and control architecture." Intelligent Systems, IEEE 20(1): 58-66.*
52. *Vrba, P., P. Tichý, et al. (2011). "Rockwell Automation's Holonic and Multiagent Control Systems Compendium." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 41(1): 14-30.*
53. Butler, J. and H. Ohtsubo, *ADDYMS: architecture for distributed dynamic manufacturing scheduling.* Artificial Intelligence Applications in Manufacturing, 1992: p. 199-214.
54. Bussmann, S. *An agent-oriented architecture for holonic manufacturing control*. in *Proccedings of the 1st international workshop on Intelligent Manufacturing Systems, IMS-Europe*. 1998.
55. Fischer, K. *The design of an intelligent manufacturing system*. in *Proc. of the 2nd Int. Working Conf. on Cooperating Knowledge-Based Systems, University of Keele, Dake Centre*. 1994. Citeseer.
56. Shen, W. and J.-P.A. BarthÃ¨s, *An experimental multi-agent environment for engineering design.* International Journal of Cooperative Information Systems, 1996. **5**(02n03): p. 131-151.
57. Kraus, S., *Strategic negotiation in multiagent environments*. 2001: The MIT press.
58. Smith, R.G., *The contract net protocol: High-level communication and control in a distributed problem solver.* Computers, IEEE Transactions on, 1980. **100**(12): p. 1104-1113.
59. Shen, W. and D.H. Norrie, *Dynamic manufacturing scheduling using both functional and resource related agents.* Integrated Computer-Aided Engineering, 2001. **8**(1): p. 17-30.
60. Ouelhadj, D., C. Hanach, and B. Bouzouia. *Multi-agent system for dynamic scheduling and control in manufacturing cells*. in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. 1998. IEEE.
61. Lin, G.Y.-j. and J.J. Solberg, *Integrated shop floor control using autonomous agents.* IIE transactions, 1992. **24**(3): p. 57-71.
62. Finin, T., et al. *KQML as an agent communication language*. in *Proceedings of the third international conference on Information and knowledge management*. 1994. ACM.
63. Fipa, A., *FIPA ACL Message Structure Specification.* Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004), 2002.
64. None, S., *FIPA Abstract Architecture Specification.* 2000.
65. Lesser, V.R., *Reflections on the nature of multi-agent coordination and its implications for an agent architecture.* Autonomous agents and multi-agent systems, 1998. **1**(1): p. 89-111.
66. *Fatima, S.S., M. Wooldridge, and N.R. Jennings, An agenda-based framework for multi-issue negotiation. Artificial Intelligence, 2004. 152.*
67. Matos, N., C. Sierra, and N.R. Jennings. *Determining successful negotiation strategies: An evolutionary approach*. in *Multi Agent Systems, 1998. Proceedings. International Conference on*. 1998. IEEE.
68. *F. Bellifemine, G. Caire, T. T. S. p. A. Trucco, formerly CSELT) , Rimassa G. (FRAMeTech s.r.l.), and R. M. P. GmbH), "Jade Administrator's Guide," JADE 4.0 ed. Boston, MA, 2010.*
69. Bruno, E.J. and G. Bollella, *Real-Time Java Programming: With Java RTS*. 2009: Pearson Education.

70. *Ferreira, P. and N. Lohse, Configuration Model for Evolvable Assembly Systems, in 4th CIRP Conference On Assembly Technologies And Systems 2012: Ann Arbor, Michigan, USA.*
71. *Tanenbaum, Andrew S., and David Wetherall. Computer networks. Prentice Hall, 2011.*
72. *CAVIN, SHIRLEY, FERREIRA, PEDRO and LOHSE, NIELS, 2013. Dynamic Skill Allocation Methodology for Evolvable Assembly Systems In: Proceedings of the IEEE 11th International Conference on Industrial Informatics, Bochum, Germany, 29-31 July 2013. (In Press.).*