Rogério Paulo Guerreiro Rosa

Licenciado em Ciências de
Engenharia Electrotécnica e de Computadores

Assessing Self-Organization and Emergence in Evolvable Assembly Systems (EAS)

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

Orientador :	José António Barata de Oliveira, Professor Auxiliar, FCT-UNL
Co-orientador :	Luís Domingos Ferreira Ribeiro, Doutor, UNINOVA

	Júri:
Presidente:	Doutor Pedro Alexandre da Costa Sousa
Arguente:	Doutor Tiago Oliveira Machado de Figueiredo Cardoso
Vogais:	Doutor José António Barata de Oliveira Doutor Luís Domingos Ferreira Ribeiro

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

Assessing Self-Organization and Emergence in Evolvable Assembly Systems (EAS)

To my mother, father, sister, brothers and my friends.

Acknowledgements

I would like to acknowledge several people that helped me through the development and writing of my thesis.

I would like to thank to Prof. José Barata for being my supervisor and for the possibility of developing such an interesting work as well as for all the opportunities that he has provided to me, in special the participation on the IDEAS project.

A special recognition to Prof. Luis Ribeiro. More than a supervisor, he is a friend. The great success of the end of my journey in the university is thanks to him, he always motivated me, guided and helped me when necessary. His professionalism, personality, ambition, had a high influence on me. It was a truly honour to work with him, and I really hope to have that honour again.

I would like to thank my lab colleagues, Andre Cavalcante, Steffen Schutz, Andre Rocha, Ângelo Veiga and Mauro Dias, who have provided a great environment during the development of this thesis.

To my colleagues from FCT-UNL that give me their support and friendship since the beginning of this journey: João Virote, Luis Sousa, Hugo Silva, Filipe Correia and Hugo Lopes, I would like to thank them for their friendship and support during this journey.

For last but not the least, I would like to show the most kind and special gratitude to my family. To my mother and my father, for their support during this journey and for being the best examples that I could have. To my brothers and sister, for being the best persons to grow with. To my niece and my nephew, no matter how bad a day is, they always bring happiness to it.

Abstract

There is a growing interest from industry in the applications of distributed IT. Currently, most modern plants use distributed controllers either to control production processes, monitor them or both.

Despite the efforts on the last years to improve the implementation of the new manufacturing paradigms, the industry is still mainly using traditional controllers. Now, more than ever, with an economic crisis the costumers are searching for cheap and customized products, which represents a great opportunity for the new paradigms to claim their space in the market.

Most of the research on distributed manufacturing is regarding the control and communication infrastructure. They are key aspects for self-organization and there is a lack of study on the metrics that regulate the self-organization and autonomous response of modern production paradigms.

This thesis presents a probabilistic framework that promotes self-organization on a multiagent system based on a new manufacturing concept, the Evolvable Assembly Systems/Evolvable Production Systems. A methodology is proposed to assess the impact of self-organization on the system behavior, by the application of the probabilistic framework that has the dual purpose of controlling and explaining the system dynamics.

The probabilistic framework shows the likelihood of some resources being allocated to the production process. This information is constantly updated and exchanged by the agents that compose the system. The emergent effect of this self-organization dynamic is an even load balancing across the system without any centralized controller.

The target systems of this work are therefore small systems with small production batches but with a high variability of production conditions and products.

The agents that compose the system originated in the agent based architecture of the FP7-IDEAS proejct. This work has extended these agents and the outcome has been tested in the IDEAS demonstrators, as the changes have been incorporated in the latest version of the architecture, and in a simulation and more controlled environment were the proposed metric and its influence were assessed.

Keywords: Manufacturing, Distributed, Self-Organization, Emergence

Resumo

Existe um crescente interesse de parte da indústria na áreas das aplicações distríbuidas em IT. Actualmente, a maioria das fábricas faz uso de controladores distribuidos tanto para controlo de processos de produção, para os monitorar ou para ambas as aplicações.

Apesar dos esforços nos últimos anos para melhorar a implementação dos novos paradigmas de manufactura, a indústria ainda usa principalmente controladores tradicionais. No entanto, agora mais que nunca, em ambiente de crise económica os consumidores procuram por produtos baratos e customizados, representando uma grande oportunidade para os novos paradigmas reclamarem o seu espaço.

A maioria da investigação nos sistemas distribuídos é focada na infra-estrutura de comunicação e controlo. São ambos aspectos fundamentais para a auto-organização, no entanto existe uma lacuna no estudo de métricas que regulam a auto-organização e a resposta autónoma dos paradigmas modernos de manufactura.

Esta tese apresenta um enquadramento probabilístico que promove a auto-organização num sistema multiagente baseado num conceito de manufactura novo, os Sistemas Evolutivos de Assemblagem/Sistemas Evolutivos de Produção. É proposta uma metodologia para avaliar o impacto da auto-organização no comportamento do sistema após a aplicação do enquadramento probabilistico que tem um duplo propósito, que consiste em controlar e explicar a dinâmica do sistema.

O enquadramento probabilístico mostra também a probabilidade que alguns recursos têm de ser alocados a um processo de produção. Esta informação é constantemente actualizada e trocada pelos agentes que compõem o sistema. O efeito emergente da dinâmica auto-organizada é um balanceamento nivelado da carga no sistema sem qualquer controlo centralizado.

Os sistemas alvos deste trabalho são pequenos sistemas, com pequenos lotes de produção mas com alta variedade nas condições de produção e nos produtos.

Os agentes que compõem o sistema têm origem na arquitectura agente de base do projecto FP7-IDEAS. Este trabalho estende esses agentes e o resultado foi testado nos demonstradores do IDEAS, assim como as alterações foram incorporadas na última versão da arquitectura, e em simulação num ambiente controlado a métrica proposta e a sua influência foi avaliada.

Palavras-chave: Manufactura, Sistemas Distribuidos, Auto-Organização, Emergencia

Acronym

AGV	Automatic Guided Vehicle
AMI	Agent Machine Interface
ASk	Atomic Skills
BA	Broker Agent
BMS	Bionic Manufacturing Systems
CLA	Coalition Leader Agent
CSk	Composite Skills
EAS	Evolvable Assembly Systems
EPS	Evolvable Production Systems
DA	Deploy Agent
DPWS	Devices Profile for Web Services
DSk	Decision Skills
FMS	Flexibile Manufacturing Systems
HMS	Holonic Manufacturing Systems
HUA	Handover Unit Agent
IADE	IDEAS Agent Development Environment
MA	Mechatronic Agent
MAC	Mechatronic Agent Class
MMAS	Mechatronic Multi-Agent Systems
MRA	Machine Resource Agent
PA	Product Agent
PSiA	Product Sink Agent
PSoA	Product Source Agent
RFID	Radio-Frequency Identification
RMS	Reconfigurable Manufacturing Systems
RTT	Round Trip Time
SOC	Self-Organized Criticality
TEA	Transport Entity Agent
TS	Transport System
TSA	Transport System Agent
YPA	Yellow Pages Agent

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In a continuous changing world with a more demanding, both from a product customization and sustainability points of view, with changing consumer habits and the subsequent decrease at the products life-cycle, a change in the established productions paradigms is required to maintain competitiveness in a global market.

In 1990's some competitive priorities as responsiveness, flexibility, quality, concern for the environment and international competitiveness have emerged [1], forcing the industry to search for new approaches on manufacturing control. Self-diagnostics, self-repair, self-organization, flexibility, extendibility are some of properties desired for the new generation of manufacturing control systems. To meet these requirements several new approaches have emerged, the Holonic Manufacturing Systems (HMS) [2], inspired on the holonic concept introduced by the philosopher Arthur Koestler [3], the Bionic Manufacturing Systems (BMS) based on biological organisms [4, 5], Reconfigurable Manufacturing Systems (RMS), designed for a rapid change in the system structure [6], and the Evolvable Assembly Systems (EAS) [7] and Evolvable Production Systems (EPS) [8], that closely relate to this work.

Despite the continuous efforts, the prototype implementations based on these approaches are not yet mature, facing some technical challenges, implying that it is not yet feasible to readily apply these concepts in an industrial context.

In fact, and despite the adoption debate [9, 10], the small scale and the limitations of the prototypes do not allow the proper validation of the reference architectures. Furthermore, although most of the architectures are conceptually simple, their implementation is technically challenging. It is often the case that the implementation often fails to meet industrial standards or its applicability is narrowed to extremely simple cases. For this reason, there are only a few cases of industry-oriented prototypes. These emerging paradigms are extremely promising. Hence the prototyping at a pre-industrial stage is of paramount importance. This is the direction taken by the present work which was integrated with the FP7 IDEAS project.

1.2 Research Problem and Contributions

The present work, supported by the scientific background of the FP7 IDEAS project [11], has a strong implementation and prototyping contribution. In particular, it stands as a proof-of-concept of the feasibility of some Evolvable Production Systems related concepts, namely emergence and self-organization. Self-organization is a central topic in the dynamics of modern production paradigms. It is also normally linked with unpredictability and abnormal behaviour. More conservative industrialists perceive it as "the system doing all sort of dangerous actions (undesired emergent behaviour) on its own".

In this context, the present implementation oriented work aims at showing how self-organization can be introduced in a mechatronic stack. Furthermore, and with the purpose of clarifying how (un)predictable these systems may be, the interactions between some of the agents, in the IDEAS stack, were studied, in simulation, to uncover the hidden dynamics of self-organization in the context of the IDEAS project.

For this purpose the stack was first applied to three pre-industrial demonstrators to assess the validity of the implementation.

Afterwards, in simulation, a decision metric was designed and applied to the system in order to quantify how likely some agents are to take some decisions. The same agent stack was considered.

In this context, the results of this work contributed to the IDEAS project, as part of the official implementation, and to raise the awareness that self-organizing mechatronic systems are not necessarily unpredictable as their dynamics can be explained.

1.3 Thesis Outline

This thesis is composed by six main chapters: *Introduction*, *State-of-the-Art*, *Architecture*, *Implementation*, *Results and Validation* and *Conclusion and Future Work*.

The first and present chapter, *Introduction*, gives a short introduction of the research problem, the contributions and research activities wherein this thesis has been developed.

The *State-of-the-Art* chapter briefs the context of this thesis. Initially presents a short resume on the new paradigms for manufacturing systems and multi-agent systems, and finally briefly discusses about emergence and self-organizing in the context of mechatronic systems.

On the *Architecture* chapter, the IDEAS reference architecture is presented, including the supporting concepts and reference agent interactions. The decision metric used, later on, to assess the self-organization response is also detailed.

The fourth chapter, *Implementation*, presents the implementation of the architecture detailed in the previous chapter. The entities that compose the system, the interactions between them and the developed algorithm that implements the suggested decision process are also fully detailed.

The *Results* chapter presents three real physical layouts where system has been tested and validated, and details the results obtained through simulation when assessing the proposed decision metric.

Finally, the chapter *Conclusion and Future work* is a critic overview of the work developed for this thesis and its potential future research directions.

2 State-Of-The-Art

2.1 Manufacturing Paradigms

In the 70's people started to become more fashion-conscious, demanding up-to-date products incorporating the latest gadgets, leading to a new trend of raising products variety with a smaller life-cycle [12]. However, the industry questioned this trend, promoting campaigns to limit the product diversity and low-priced quality products. This strategy proved to be wrong since the profits did not increase, the markets were composed of many niches and the costumer tastes kept changing, being necessary to promote flexibility to face the new markets trends without increasing the production costs [12]. All this culminated with the emergence of the Flexible Manufacturing Systems (FMS). The manufacturing flexibility concept means the ability to produce different products efficiently through a manufacturing resources reconfiguration [13].

The Agile Manufacturing concept introduced by Nagel and Dove at the Iacocca institute [14] implies that agility is the ability to thrive and prosper in an environment of constant and unpredictable change [15]. The agile paradigm differs from the FMS by being more than flexible covering different areas from management to shop-floor control.

The Holonic Manufacturing Systems (HMS) [2, 16, 17, 18] are inspired on the holon concept of Arthur Koestler's work [3] which describes the living systems and their social organization. The "holon" word is a combination of "holos", meaning whole, and the suffix "on", meaning part. The Holonic concept is hierarchically divided and a Holon can be an organism composed by minor organisms and be part of a bigger organism simultaneously. The application of this concept in manufacturing results in a system with several subsystems that can be decomposed to exhibit the holonic behaviour [16]. The interaction between the parts is, however, easier to describe and contextualize than it is to implement. It is, nevertheless, a central concept in most modern production paradigms as it stands as the first step towards complexity encapsulation. This means the holonic approach, and others that follow, can conceptually support a multilevel description of a mechatronic system.

The Bionic Manufacturing Systems (BMS) [4], similarly to HMS, take inspiration in living organisms focusing on organs. In this context, a manufacturing system is a composition of cells and organs. The BMS relies on a kind of product agent that is able to produce itself by enacting a self-organization system response. The main concept is that the shop-floor components are able to dynamically interact with each other exchanging DNA like data. Later an extension of BMS with learning capabilities was proposed [19]. Although the philosophical background differs from HMS, the same base ingredients are shared: complexity encapsulation, self-organization, decentralization of the decision problem(and also of mechanisms that jointly contribute to solve a production problem, whose scope exceeds the one of individual parts).

The Reconfigurable Manufacturing Systems (RMS) [6, 20] are designed to have capabilities for a fast reconfiguration and integration of the system components. To achieve these capabilities, a RMS system should be designed with some characteristics such as: *Modularity*, *Integrability*, *Convertibility*, *Diagnosability*, *Customization* and *Scalability* [21]. The RMS do not envision a new specific paradigm, but they set guidelines on how technology should be to support such dynamic shop-floors. The main enabler of the successful application of these emerging paradigms, probably does not lie in paradigms nor technology alone, but rather when both are conveniently combined. In [22], the author identify three fundamental gaps that frequently limit the correct path from conceptualization to implementation.

2.1.1 Evolvable Production Systems

The EAS/EPS paradigms [7], recently introduced, shares some particularities with the HMS, BMS and RMS, which, not surprisingly, are the idea of encapsulation and self-organization.

An EAS/EPS system should comprise two characteristics: The ability to evolve and adapt. Adaptation in the sense that the system needs to be able to propose an alternative configuration to minimize the adverse effects of disturbances. Adaptation is short-termed and normally entails self-reconfiguration typically in the form of parameter's adjustments. Regarding the evolution, the system has to be able to allow the introduction of new modules or removal of existing modules [8]. Evolution is, therefore, a long-term process that involves strategic decisions and cost evaluation. The EAS/EPS follow three fundamental guiding principles [23, 24, 8, 25]:

- Principle 1: "The most innovative product design can only be achieved if no assembly process constraints are posed. The ensuing, fully independent, process selection procedure may then result in an optimal assembly system methodology".
- Principle 2: "Systems under a dynamic condition need to be evolvable. i.e., they need to have an inherent capability of evolution to address the new or changing set of requirements".
- Principle 3: "EPS systems are based in intelligent, process oriented, self-contained, self-organizing modules that can aggregate to deliver different functionalities on demand".

To achieve this an EAS/EPS system must be designed taking into consideration the following features [26, 27]:

- Modularization: Similarly to HMS, BMS and RMS paradigms the notion of independent modules should be present on an EPS/EAS system.
- Granularity: An EPS/EAS should allow several levels of granularity (i.e. one module can be a simple gripper or the whole robot).
- Plugability: The ability to handle the introduction of new modules while the system is running. The system must be able to redesign the internal dynamics in order to keep its efficiency.
- Reconfigurability: The system needs to be able to handle the redesign of the layout without compromising any functionality.

All these features bring the EAS/EPS very close to RMS, being mistakenly considered the same concept with different interpretations, however, some differences between the concepts can be spotted. The RMS focuses on the reconfigurability of system components while EAS is focused on the adaptability. As stated before the EAS/EPS allow several levels of module granularity, the RMS follow an approach of "transport-handling-assembly-finalisation" block, which implies a limitation on granularity [28].

EPS/EAS have been developed along a successful series of multidisciplinary international research projects. The latest of these projects, the FP7 IDEAS project, was the background environment supporting the development of this work. IDEAS focused not only on the scientific development of EPS related concepts as well in the development and pre-industrial validation of supporting technology from a mechatronic point of view (software and hardware). The agents later detailed, whose implementation is the backbone of this work, are a fundamental part of the software concept as EPS envision a multiagent based system.

2.1.2 Multi-Agent systems

Although there is not a consensus about the definition of an Agent, in this work the adopted definition is the given by Wooldridge [29, 30, 31]:

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives."

Autonomous action is indeed the most important characteristic when defining an agent. Other important characteristics include **Reactivity**, **Rationality**, **Proactiveness**, **Social Ability**, **Adaptability**, **Continuity** and **Mobility** [30, 32, 33].

Although, these characteristics affect most the individual architecture of an agent, there are at least two, social-ability and mobility, that can only be expressed in a multiagent context.

The prominence of the others normally defines the agent's behaviour. Reactive agents normally require less computational effort in their decision process. However, they may be sensitive to decision myopia. On the other hand, Rational agents consume more computational resources and, although decision myopia may not constitute a problem, the time required to enact a decision can be too long. In this context, the perception of the world that led to that decision may not be valid any more when the decision becomes effective.

It is reasonable to envision an agent as a problem solver. In a multi-agent context one is therefore considering a distributed set/network of problem solvers. Furthermore, the components of this network can make use of their social-ability to collectively solve problems.

From a technological point of view, the implementation of such system implies the existence of an agent platform upon which the reference architecture can be instantiated.

Regardless of the agent characteristics the architecture normally defines generic classes of agents and their interactions. The focus on this generic design is fundamental to support the encapsulation, self-organization and scalability requirements of modern paradigms.

The Figure 2.1 [26] represents one of such generic architectures for a MAS framework regarding the EAS/EPS concepts. It is derived from the CoBASA architecure [32]. The architecture contains the following agents:

- **Mechatronic Agent (MA)**: It is a threefold entity constituted by an equipment, its controller and the active logical part (the agent). The mechatronic agent is, in this context, a self-containing pluggable unit.
- **Broker Agent (BA)**:The BA is an intermediary agent that keeps the information

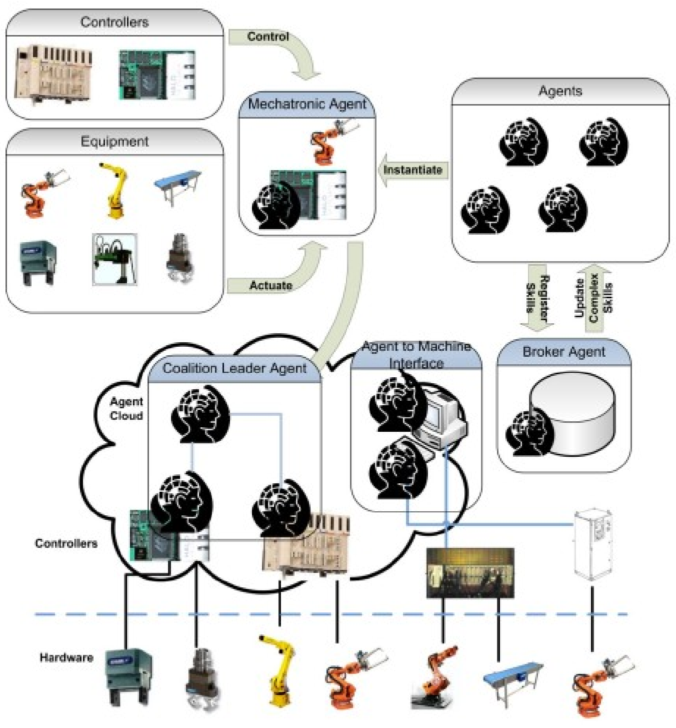


Figure 2.1: MAS generic architecture

about the other agents present in the system. It also facilitates their interactions.

- **Coalition Leader Agent (CLA)**: The function of this agent is to promote the organization between the other agents in order to aggregate the functionalities provided by them. It is therefore paramount in promoting the emergent behaviour that leads to an organized structure.

- **Agent Machine Interface (AMI)**: Is the agent that controls the hardware modules. This agent works as an intermediary between the mechatronic equipment and the agent when the full integration of the mechatronic agent cannot be attained.

The purpose of the architecture is also a fundamental point that needs to be considered as it definitely influences the implementation decisions and supporting technologies. For instance, the generic architecture depicted in Figure 2.1 is focused on system re-engineering. The IDEAS architecture detailed in the next chapter also entails fast re-engineering, but includes other aspects such as control and material handling in an emergence and self-organization framework.

In this context, it is crucial to clearly understand the meaning of these concepts in a mechatronic scenario. The next subsection presents a brief discussion of both concepts. This is highly relevant since EPS/EAS evolve around the idea of Adaptation and Evolvability supported by self-organization and emergent behaviour but do not specify the mechanisms to attain so. Having stated so there have been preliminary implementations, specially in the academic domain of EPS/EAS in particular:

- MAS - oriented: [34, 35] preliminary implementations on a didatic kit based on the CoBASA architecture but featuring material handling.
- Web Service - oriented: [36] a similar implementation exploring web service technology in particular the emerging Devices Profile for Web Services (DPWS) stack.

Some basic self-organization mechanisms were explored in these papers that eventually led to the solution presented in this work.

2.2 On Emergence, Self-Organization and Mechatronic Systems

2.2.1 Emergence

Emergence and Self-organization are central concepts in recent production paradigms and architectures. They are also frequently used in an loosely way. The concept of emergence is attributed to G. H. Lewes. Lewes observed that "although each effect is the resultant of its components, we cannot always trace the steps of the process, so as to see in the product the mode of operation of each factor" [37]. Holland supports roughly the same definition: "there are regularities in system behaviour that are not revealed by direct inspection of the laws satisfied by the components" [38]. If one inspects other definitions of emergence the notion that the whole is more than the sum of the parts dangerously emerges to "raise the spectre of illegitimately getting something out of nothing" [39].

There appears to be several ingredients to identify emergence. Whether or not the observer is one of these decisive ingredients is one of the most relevant discussion topics. While some authors deem the observer necessary, see for instance [40] that claims that "the concept of emergence is much better captured through the sound analysis of the observer's structure", others reject the observer as condition to identify emergence [41]. There is a very important qualitative, phenotype-like, dimension of emergence that conveys significance to the observer which is generally rejected by pure reductionists. This debate revolves around the "how" and the "why". While pure reductionists contend that emergence is associated with a certain degree of ignorance from the observer in respect to the "how", they often neglect the "why" which, due to the observer's inability to fully grasp the causal matrix, hinders pure determinism and prevents him from, in most cases, relating the mechanisms that explain how a set of parts lead to useful phenomena as an whole [42]. The why normally relates to a set of observable properties which can be more or less salient according to [37]:

- "Radical novelty: emergents have features that are not previously observed in the complex system under observation. This novelty is the source of the claim that features of emergents are neither predictable nor deducible from lower or micro-level components. In other words, radically novel emergents are not able to be anticipated in their full richness before they actually show themselves."

- "Coherence or correlation: emergents appear as integrated wholes that tend to maintain some sense of identity over time. This coherence spans and correlates the separate lower-level components into a higher-level unity."
- "Global or macro level: since coherence represents a correlation that spans separate components, the locus of emergent phenomena occurs at a global or macro level, in contrast to the micro-level locus of their components. Observation of emergents, therefore, is of their behaviour on this macro level."
- "Dynamical: emergent phenomena are not pre-given wholes but arise as a complex system evolves over time. As a dynamical construct, emergence is associated with the arising of new attractors in dynamical systems (i.e., bifurcation)."
- "Ostensive: emergents are recognized by showing themselves, i.e., they are ostensively recognized. (...)Because of the nature of complex systems, each ostensive showing of emergent phenomena will be different to some degree from previous ones."

In [43] the following properties are additionally considered as pertaining to emergence:

- "Interacting parts - The parts need to interact - parallelism is not enough. Without interactions, interesting macro-level behaviours will never arise. The emergents arise from the interactions between the parts."
- "Decentralized control - Decentralized control is using only local mechanisms to influence global behaviour. There is no central control. i.e. no single part of the system directs the macro level behaviour. The actions of the parts are controllable. The whole is not directly controllable (...)"
- "Two-Way Link - In emergent systems there is a bidirectional link between the macro-level and the micro level, the parts give rise to an emergent structure. (...) In the other direction, the emergent structure influences the parts"
- "Robustness and Flexibility - (...) Emergents are relatively insensitive to perturbations or errors (...)"

If one considers the observer a fundamental part in the identification of emergence then time and scale of the observation are also fundamental ingredients of emergence. In this context Castelfranchi [44] proposes a threefold classification of emergence:

- Diachronic emergence - is a time based process requiring a favourable convergence of factors, typically the critical accumulation of "components, or ingredients and forerunners of that phenomena" that in the right mix can trigger a disruption in the system leading to different system phases.
- Synchronic emergence - is related to the point and scale of the observation process itself. From a specific point of view correlations between otherwise uncorrelated entities may become evident.
- Descriptive emergence - relates to the description of emergence properties in the macro level where they are visible and measurable. It is a form of emergence that suits the observers descriptive purposes in respect to the observed system.

It is however difficult to devise rules on how to decide on the time frame and scale of the observations. The link between emergence and self-organization has been widely debated.

2.2.2 Self-Organization

A general definition of self-organization is: "a system is self-organizing if it acquires a spatial, temporal or functional structure without specific interference from the outside. By specific we mean that the structure or functioning is not impressed on the system, but that the system is acted upon from the outside in a non specific fashion"[45].

Self-organization is often observed in natural systems yet in most cases the organization of these systems happens at critical state. This sort of organization is typically cited in the literature as Self-Organized Criticality (SOC). The concept is normally attributed to Bak, Tang and Wisenfeld when they proposed the sandpile model as an example of the quantification of SOC [46]. As detailed in [46] "the system naturally evolves to the state without detailed specification of initial conditions (i.e. the critical state is an attractor of the dynamics). Moreover, the critical state is robust with respect to variations of parameters, and the presence of quenched randomness".

As detailed in [43] this ability to retain order is fundamental in the characterization of self-organization and further, in addition to the previously described properties, in the identification of pure emergent systems, pure self-organizing systems or systems that denote both.

Context seems to decisively influence the identification of emergence and self-organization. In the domain of emerging production paradigms both concepts have become buzz words and are often loosely applied to describe mechatronic systems that can adapt to changing conditions very often by adjusting processes' parametrization or, in extremely rare cases, their physical/mechanical configuration. There is generally a scarcity of metrics that determine the degree of qualitative novelty (arguably emergence) or organization when these systems take such an actions. Yet most current research is pointing in the direction of developing increasingly distributed and pluggable component-based systems. One of the main challenges is therefore overcoming the absence of general indications and guidelines on evaluating the state of the system which renders very difficult its guidance through a trajectory including the best possible states as opposed to a path that, leading to the same final result, may hinder the overall response.

2.2.3 Challenges in a mechatronic context

The successful implementation of modern production paradigms is currently less a technological issue and more a matter of modelling. If the current development trend continues it is fundamental to define a methodology to design the next generation of production systems. Emerging approaches propose a radically different way of building systems where global models are replaced by open system architectures that emphasize the generic interactions between the system's components and how these should influence it has an whole. These systems are designed for emergence and indeed it can be the most fruitful way of conceiving and managing complex production systems that result from the composition of thousands, maybe millions of intelligent components.

It is therefore necessary to understand how the existing notions of emergence and self-organization may apply to a mechatronic system. Bedau proposes the notion of Weak Emergence [39, 41], which may be useful for this purpose, that determines that the system's macro-states can be derived from its micro-states and its micro-dynamic. Bedau also sustains that this link must be supported by a simulation process given the potential complexity of the causal matrix. Bedau's definition appears has a reaction to the more traditional conception of Emergence (Strong Emergence) that is more restrictive in the two-way link between the whole's behaviour and the parts. In the context of this work, the author shall follow Bedau's notion of Emergence as, to a great extent, the purpose of the paper is to uncover the inner dynamics that govern a Mechatronic Multi-Agent system (MMAS) as an whole.

As for the concept of self-organization the author perceive it as a natural property of the mechatronic systems discussed. These systems make massive use of autonomous behaviour in an attempt to continuously adjust to changing production conditions while maintaining a productive organizational state. In this respect the author′ view is aligned with the characteristic properties of self-organization as defined in [43].

2.3 Integrated discussion

The research in application of multi-agent systems in an industrial context is vast and multidisciplinary. It goes from technological considerations and development to more philosophical issues.

There has been a wide debate relating with the small implantation of these emerging production approaches in concrete industrial scenarios. One of the main difficulties is that most of these concepts are easy to explain and understand but they are difficult to implement. The EPS/EAS paradigm is an example of such efforts. In particular it has been undertaking development and refining activities for almost one decade and, only recently, in the scope of the IDEAS project, the first pre-industrial prototypes have emerged.

The project has also been voted as an FP7 success story [47], and part of this success has to do with the focus on concepts but also on the instantiation of those concepts in a technological framework. As stated before, one of the biggest contributions of this work lies in the implementation and refinement of part of the IDEAS architecture.

This architecture strongly relies in emergence and self-organization. These concepts are normally perceived as the sources of disturbances and undesired behaviour. It is therefore also important to start to formalize and understand the dynamics of such systems in order to show that, although they create production environments that are very different from the current ones, there is a considerable added value in adopting them.

In this context, the IDEAS architecture and a metric to study self-organization are detailed in the next chapter.

3 Architecture

This chapter starts by defining two supporting concepts, namely skill and area, which are required to explain the generic interactions between the IADE agents. The agents and their interactions are then defined and finally the adaptations considered for the study on self-organizing as well as the metric are presented.

3.1 IADE

The IDEAS Agent Development Environment (IADE) is a library developed under the IDEAS project and based on the JADE platform.

The results of this thesis are supported by IADE and, simultaneously, this work contributed to its implementation. The principal contributions consist in the partial development of the Mechatronic Agents (MA), tools and Yellow Pages services. The implementation of the Transport System is out of the scope of this work.

The final results presented on this thesis are an important study in order to support the system configuration, which also provided feedback to the IADE agent stack.

3.1.1 Skill Definition

The generic and dynamic functionalities provided by the agents in IADE are expressed as skills. There are three skill types in the architecture. The **Atomic Skills (ASk)** are the link between the agent and the implementation of a library for a specific controller (I/O mapping and low level functions). In this context, an atomic skill encloses the code required to interact with the physical world. The **Composite Skills (CSk)** implement processes which can contain any other skills including others CSks. A CSk contains a workflow of skills that defines the process. Unlike ASks the CSks never directly interact with the hardware. In this context, they simply mediate the execution at agent level. **Decision Skills (DSk)** are a special skill type which performs decisions during the process execution and, according to the result of that decision, the process will take different paths.

The Figure 3.1 depicts a potential composite skill. Important points about the picture include the fact that, as stated before, a CSk can contain any other skill. In particular, a CSk can define a sequence, as in the main CSk portrayed in Figure 3.1, or a parallel execution on the top branch of the DSk. The decision skill is based on the evaluation of a user defined expression. If the expression evaluates to true, the top branch is executed. If it evaluates to false the other branch executes. Only ASks can interface the hardware and each different skill may be controlled by a different agent. A skill can exist even if there is not an agent to control it. In this sense, skills are just descriptions and only come to live when associated with specific instances of agents.

Skills are also comparable entities. In this context, two agents can exchange skill information and verify if the skills are the same even if their instantiation, on the shop-floor, is different. In this context, agents can allocate and replace comparable skills. This fact leads to the notion of area defined next.

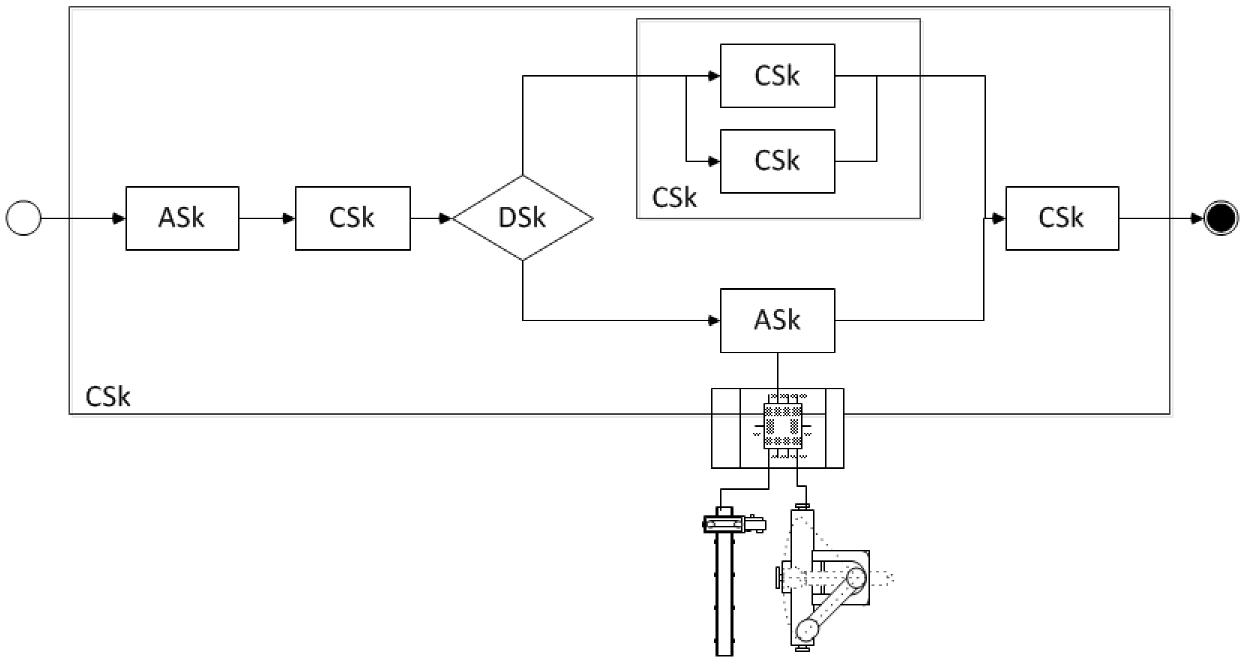


Figure 3.1: Composite Skill Example

3.1.2 Area Definition

In the proposed architecture, the scope of some particular agents is limited by areas which are a definition of physical delimitations of the action scope of one agent. In the Figure 3.2

is exemplified a system with two distinct physical areas, both with similar modules.

Considering the example on Figure 3.2 a MA deployed on area A can only allocate skills of other agents that are physically located on that area, the same rule is applied for a MA deployed on area B.

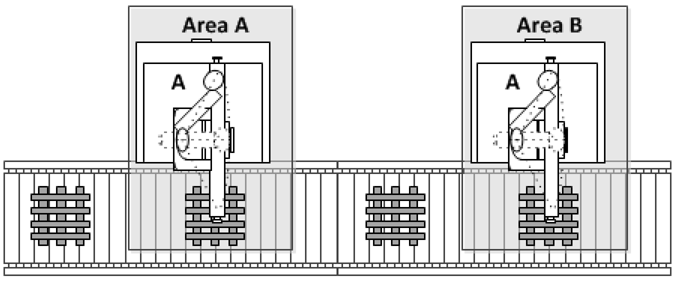


Figure 3.2: Area Example

This definition prevents agents from allocating skills of other agents that may match the requirements of a CSk, but that are not physically accessible. This restriction is specially important when dealing with ASks (and hardware manipulation). For this reason, different agents can belong to areas of different sizes, so that in case we have CSks that are composed by others CSKs the controlling agents are able to allocate skills in a wider area of the system boosting self-organization.

3.1.3 IADE Agent Architecture

The IADE architecture (Figure 3.3) is designed to support self-organizing, robust and reconfigurable systems. The IADE architecture explores and extends JADE's agent structure and infrastructure in particular: FIPA compliant communication support, basic agent architecture, service publishing and subscribing functionalities.

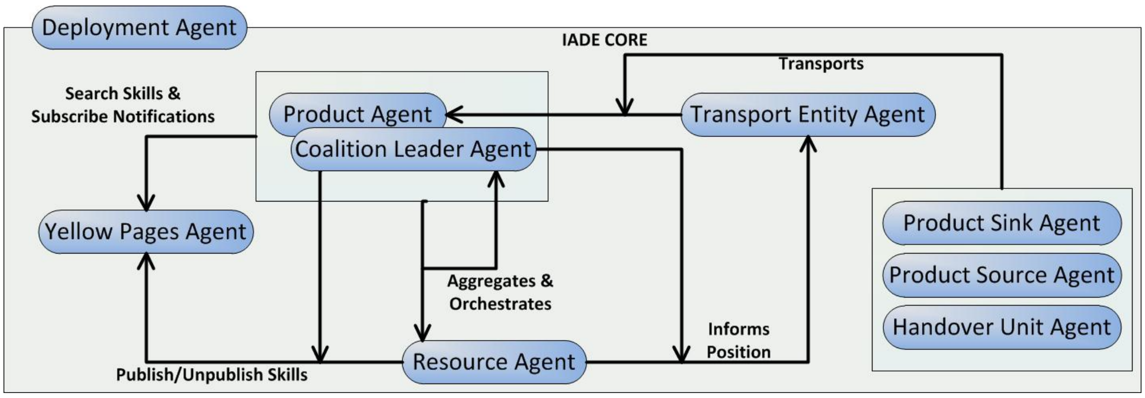


Figure 3.3: IADE Agents

The agents that compose the IADE core are:

- **Machine Resource Agent (MRA)** is the simplest agent present on IADE core. This agent controls the hardware modules that can be plugged and unplugged from the system. The MRA only contains atomic skills that are indivisible and directly related with hardware. As a consequence, any reconfiguration attempt of functionalities provided by MRA will require some changes at the controller level. Although these changes are out of the agent control, the MRA provides the required mechanisms to handle them in a generic way.

- **Coalition Leader Agent (CLA)** is the agent that supports the composition of skills. This means that it is possible to build and execute process using the skills available in the system. The CLA is able to react to disturbances in the environment such as the addition or removal of functionalities and faults under its coalition. Adding functionalities does not necessarily lead to a reaction by the CLA, since the agent can choose not to reconfigure itself if everything is stable. In case of removal or fault, whenever possible, the CLA will negotiate a replacement in order to keep the system running. The ability of negotiation places the CLA as a pillar of self-organization in the system.

- **Yellow Pages Agent (YPA)** is responsible for managing the information provided by others agents. It provides an infrastructure that allows other MAs to register their information or query about others, it also offers a subscription service which notifies the subscribers each time an agent informs the YPA about a change on its state. The system is composed by several YPAs and each one stores the corresponding information of the agents that belong to a specific area.

- **Deploy Agent (DA)** is an auxiliary agent and is able to receive a serialized IADE agent, rebuild it and deploy the agent on the controller where the DA is running. This agent allows the system to be configured through external tools.

- **Transport Entity Agent (TEA)** is the agent that abstracts the transport entities (i.e. conveyors or automates guided vehicles). It is responsible for the computation of transporting costs between locations in the system.

- **Handover Unit Agent (HUA)** is the agent responsible for controlling the physical handover of a product from one TEA to another. This agent processes the transport cost from TEAs and interacting with other HUAs, computes the transport spanning tree, that contains the best routes between each pair of HUAs/TEAs on the system.

- **Product Source Agent (PSoA)** this agent is a special case of HUA and is responsible for introducing new products in the physical environment.

- **Product Sink Agent (PSiA)** is the responsible agent for retrieving products from the environment, similarly to PSoA this agent is a special case of HUA.

- **Product Agent (PA)** is an extension of the CLA agent. The PA represents a product and its production process. From an execution point of view the principal difference between PAs and CLAs is the interaction with the TEA, the PA should combine the information given by the TEA, CLA and RA to optimize the production process.

3.1.4 Agent interactions functional view

Table 3.1 [48] details all the possible interactions between the agents described.

Table 3.1: Possible Interaction between the IADE Agents, x denotes interaction

	MRA	CLA	PA	TEA	HUA	PSoA	PSiA	YPA
MRA	-	x	x	x	-	-	-	x
CLA	x	x	x	x	-	-	-	x
PA	x	x	-	x	-	x	x	x
TEA	x	x	x	-	x	x	x	x
HUA	-	-	-	x	x	-	-	x
PSoA	-	x	x	x	-	-	-	x
PSiA	-	x	x	x	-	-	-	x

These agents include all the shop-floor entities, although they can be extended to incorporate new functionalities without changing their internal behaviours and the communication protocols, as is the case of the PA. When launched, the agents have to perform some configuration routines which include communicating with other agents for the purpose of setting up the environment and synchronize the start of some activities. The Figure 3.4 depicts the interaction flow that occurs when the agents are launched. It also encovers part of the engineering methodology.

The system setup starts with the deployment of the HUAs. After all HUAs have been started the TEAs are added to the system. There is one HUA between two or more TEAs and together they form the transport network. This implies changing transport costs (the cost of traversing each TEA), and the construction of the routing tables (at HUA level).

Sources and Sinks are added afterwards. From this point, the transport system is fully operational and is ready to route any product agents. However, without any resources, the PAs would not have anywhere to go.

The next step is therefore the addiction of resource agents and the bootstrap of their atomic skills. From this moment on PAs, whose process description only includes the available skills at the initialized MRAs, can start producing.

Nevertheless, the purpose of the architecture is to enable skill composition, so that more complex processes can be supported by the system. In this context, these processes can be supported by defining the adequate CLAs. Simultaneously as these agents are plugged on the system, they register their skills in the YPA that is associated with the TEA where the specific MRA or CLA executes its skill.

Finally with the adequate processes defined and their execution locations know by the system, the PAs can be launched and start producing. Their entrance on the system is always mediated by the PSoA and their exit is controlled by the PSiA.

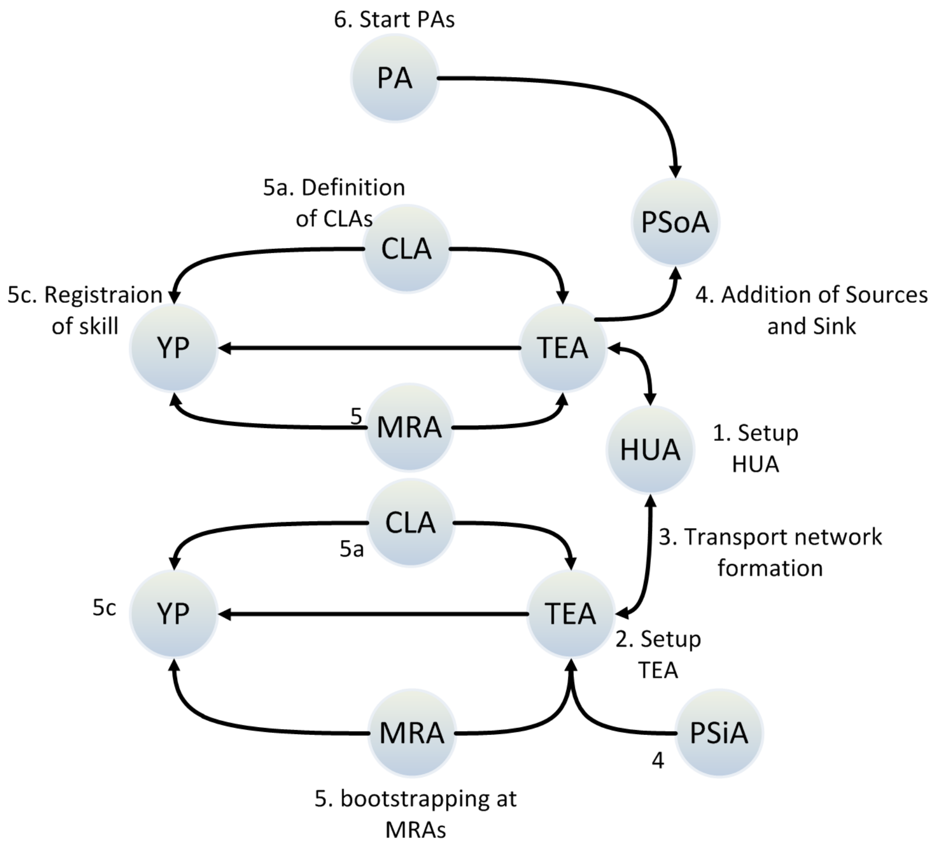


Figure 3.4: IADE - Interaction flow at the start-up

The Figure 3.5 shows the interactions flow between the PAs, CLAs and MRAs during the process execution. The PAs, unlike CLAs cannot respond to incoming messages, because the PAs are autonomous from an execution point of view and are a top level entity, on the other hand, the MRAs only react when they receive an incoming message. The CLAs have the ability to receive and send execution requests. The PAs, during their execution, are attached to a TEA, this TEA changes according to the PAs physical location. When a PA needs to be transported, it performs a transport request to the corresponding TEA. The TEAs when are transporting a PA they displace it (physically and logically) to one of its neighbours through a HUA. However the TEAs neighbours are only HUAs which do not communicate with the PAs. The HUA, when receiving a PA, re-routes it to the neighbour TEA, that is part of the best path for the PA destination. Finally, when the PA reaches the destination the target TEA informs the PA and that TEA becomes responsible to handle the future transport requests from the PA.

Figure 3.5 shows that from a functional point of view the main interactions as well as the decision process is centered in the product agent.

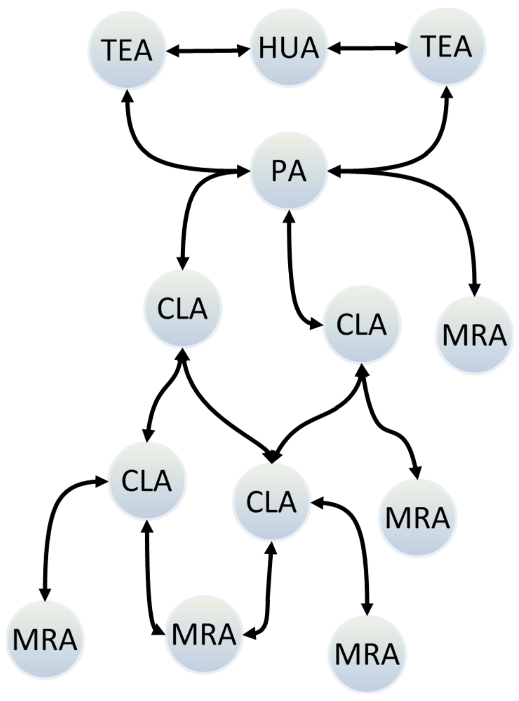


Figure 3.5: IADE - Interaction during runtime

In fact the PA is, indeed, the top level decision maker and the ultimate responsible for the execution of its process plan. Although these interactions appear simple in this high level architectural view, there is a considerable number of actions happening to support them.

In particular, before deciding where to execute next, the PA contacts the TEA, that is currently associated with it, and queries it for all the agents, and transport cost associated with their location, where the next skill in the process plan can be executed. It then contacts the corresponding MRAs or CLAs for an estimation of the execution cost. With both these values it computes a combined cost and commits to an execution location. The PA then requires the transport (following the logic already described).

When at the execution location, the PA will require the execution to the corresponding agent. If this agent is a CLA it may, correspondingly, trigger other agents in cascade.

As shown later in the results chapter, performance is the price paid for the liberty of reconfiguring the system in an had-oc way. The depth of the composition should be carefully regulated and understood to maintain the desired system response.

3.1.5 Simplified IADE stack and self-organization assessment

3.1.5.1 Reduced Agent Stack

In order to focus on the study of self-organization, during runtime, and to isolate it from the influence of the transport system, whose dynamics are out of the scope of the present work and have been studied in [49], a simplified version of the IADE stack was considered (Figure 3.6).

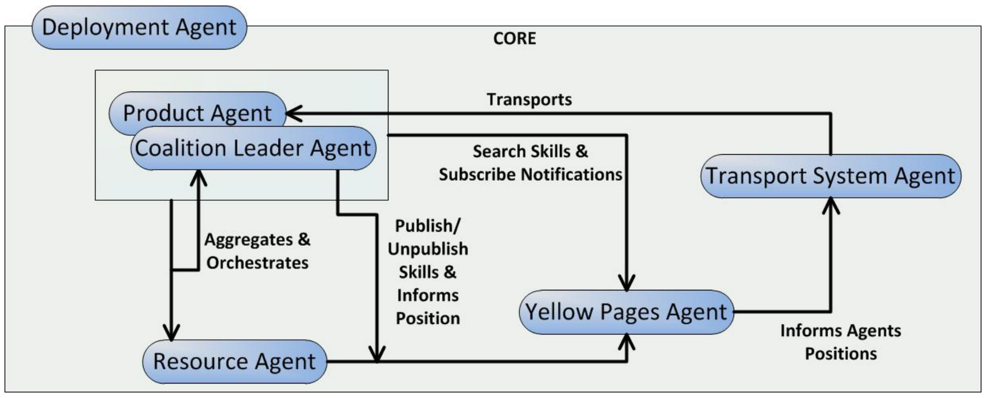


Figure 3.6: Simplified IADE Stack agents

As detailed in the picture, the number of classes considered to model the transport system has been reduced. From an instantiation point of view, there is only one instance of the transport system agent. This instance controls the entire transport system.

The re-mapping of these interactions is detailed in Table 3.2. In comparison with Table 3.1 one of the main differences, aside on the type of agents, is that in the reduced version the CLAs and MRAs interact directly with the area YPAs to inform about their skills. In the IADE stack this interaction is always mediated by the TEAs.

Table 3.2: Possible Interaction between the Simplified IADE Agents

	MRA	CLA	PA	TSA	YPA
MRA	-	x	x	-	x
CLA	x	x	x	-	x
PA	x	x	-	x	x
TSA	-	-	x	-	x

The system is started in a fairly similar way (Figure 3.7). First the transportation agent is started, then the area YPA is initialized and, finally, the MRAs and CLAs are added to the system. Upon entrance they register in the corresponding YPA.

Once the entire system is setup, the PAs can be launched. Note that the reference interactions between PAs, CLAs and MRAs remain the same and that the self-organization assessment will focus on these specifically.

The runtime dynamic is also slightly different (Figure 3.8). In particular, the PAs request from the single instance of the TSA a list of the locations where the desired skill can be executed. Instead of replying with the transport cost, and the associated agents, the TSA returns the addresses of the YPAs.

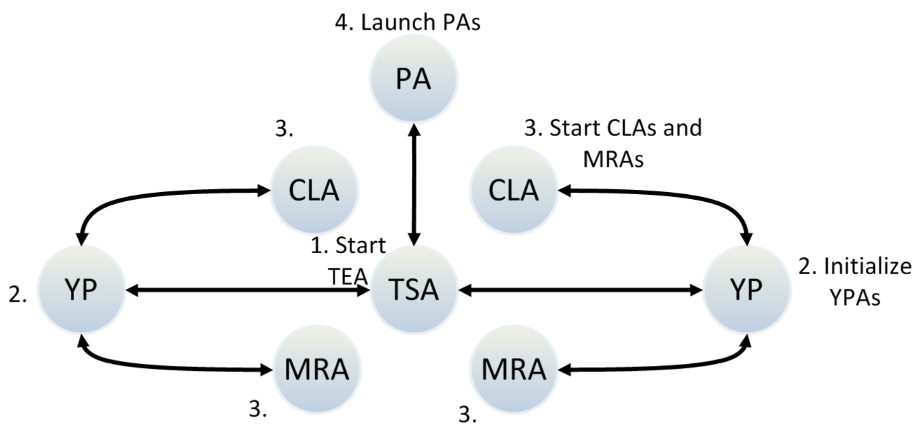


Figure 3.7: Simplified IADE Stack - Interactions flow during configuration

The PA will subsequently contact the YPAs to set the execution on the different locations. A consequence of such set of interactions, is that the transport cost is not considered in the decision process.

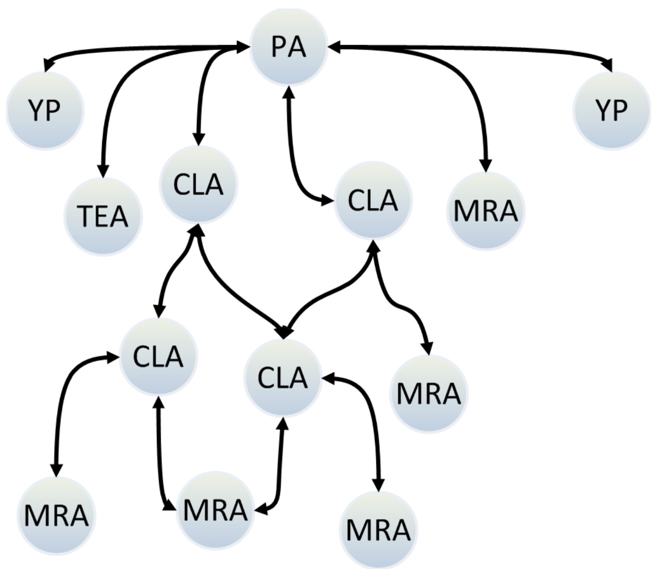


Figure 3.8: Simplified IADE Stack - Interactions flow during execution

With the set of interactions fixed it is now important to define the decision metric that is involved in the decision of where the execute and frame the whole process under the scope of emergence and self-organization.

The architecture sections defining IADE and its simplified form, only provide a high level functional perspective. The more technology-oriented details are provided in the implementation chapter.

3.2 Emergence and Self-Organization in IADE

3.2.1 Contextualizing Emergence and Self-Organization

In characterizing the emergent and self-organizing aspects of the system one needs first to consider it under the framework of the emergence and self-organization. In respect to emergence the author follows Bedau's notation [39]: "Macrostate P of S with microdynamic D is weakly emergent if P can be derived from D and S's external conditions but only by simulation.". In the present case S denotes an IADE system.

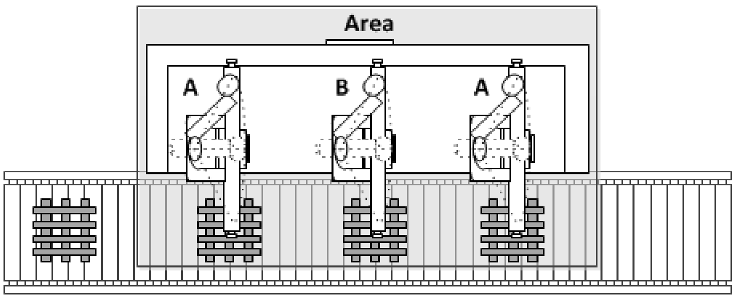


Figure 3.9: System Example

A brief example on how to apply the concept of weak emergency shall be provided now. In this case, Figure 3.9 shows a system S that is composed by a conveyor and a area with three stations. From an agent point of view, the station would be abstracted and controlled by a CLA. Under its coalition each one of the robots would be considered as an MRA with skills A, B and A respectively. The interactions between the IADE agents that compose this system defines the microdynamic D. The microdynamic has been fully characterized from a functional point of view in the previous section.

It is important to remind, at this point, that Bedau's definition of weak emergence is completely generic and detached from any context.

In the mechatronic example provided, or in any IADE system, the purpose is not to attain a specif macrostate P but rather ensure, by promoting self-organization, that the physical system is in a set of desired states.

The size of this set of states depends on the size of the system and the enumeration of all the elements in the set defeats the purpose of using self-organization as the driving force to keep the system under control. In fact if the state space of the system can be easily enumerated then probably the EPS approach is not suitable for that specific system.

The main concept is, therefore, to ensure that the system remains in a productive state when it faces disturbances even with a performance penalty. At the same time, when the disturbing elements are removed, the system should try to improve its efficiency, for instance, by balancing the utilization of resources.

For instance in Figure 3.9, if one of the MRAs implementing A breaks down the other may re-parametrize itself to, not only do its job with the native parametrization, but also do the other agent's job.

The loss of performance is obvious but the overall system remains operational. This is fairly different from current shop-floors where the same scenario would crash the entire process. This is definitely added value that can only be considered in an emergent and self-organizing perspective.

From a architectural point of view, the process is still controlled by the several PAs (which are the decision makers). This results in a wide range of decisions that need to come together as a self-organizing whole.

Choosing the right decision metric can on the one side improve this process and, on the other side, help explaining the self-organizing dynamics.

3.2.2 A metric to promote and assess self-organization and emergence

As previously stated the PAs dynamically choose the agents that will execute the skills in their process plan. This feature occurs through negotiation. In the simplified stack library the negotiation process is slightly different on the way the decision is made. With or without the transport system, when a PA commits to executing in a specific location/agent that agent can have an estimate of its future workload and can use this knowledge to attract or repel more PAs. Such kind of look ahead metric can dynamically capture the joint future intentions of all the PAs on the system. It can also justify and explain why some resources are more attractive than others. In this context, the proposed metric follows this philosophy and is focused on the interactions between the PAs and the first layer of agents that provide a direct match for the PA's required skills (Figure 3.10).

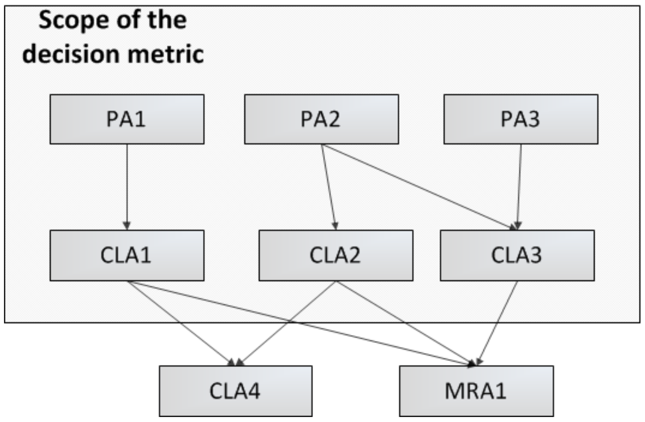


Figure 3.10: Scope of the decision metric

The individual metric of each CLA is calculated under the probability (Equation 3.1) of the CLA to be invoked, where this probability is related with the CLAs and PAs that have a history or can in the future invoke the given CLA.

The P_{Pi} term in Equation 3.1 is related with the mix of PAs in the system, and its value is reduced when a skill execution of CLA is performed, since the agent knows that there is one agent less to request the invocation of the skill. The $P(CLAx \mid P_{Pi})$ term is related with the history of successful negotiations for the skill. A successful negotiation will increase the value of $P(CLAx \mid P_{Pi})$ since the agent considers now that there is a certain agent that will require the execution of its skill. The purpose is to, as previously stated, capture what is likely to be the future state of the system.

$$P_{CLAx} = \sum_{i=0}^{P_{types}} P_{Pi} P(CLAx \mid P_{Pi}) \tag{3.1}$$

The Equation 3.2 is the decision metric itself. By using this metric the agent attempts to minimize the execution cost for a set of skills in its process plan. In fact, what the metric is computing is the joint cost of executing the process plan over the available resources. As the dynamic of the system changes the value of $P(CLAx)$ so changes the cost for subsequent PAs trying to execute in the system.

$$D_m = \sum_{i=0}^{CLA_{required}} P_{CLAi} \tag{3.2}$$

The main restriction of this metric is that the computation of $P(CLAx)$ requires some global knowledge about the system, in particular the current mix of PAs on system. In the conventional version of IADE this is controlled by the Product Sink and Sources Agents, and in the reduced version of the stack it is available as shared data.

The expected outcome of this metric is that the system will be able to balance the load of the PAs on the available CLAs on first tier of resources.

It is also important to stress that the metric is not meant to be an optimized metric for load balancing. The scope of the present work is not to perform scheduling based on an agent systems. For this purpose, there are several specialized papers [50].

The focus of the work is clarifying self-organization and emergence in a mechatronic context. Hence the dual purpose of the metric, showing load balancing and explaining the dynamics that led to it.

As detailed before implementation is still a significant barrier. The next chapter provides the main implementation details supporting this work.

4

Implementation

This chapter describes the implementation of the concepts used on this thesis as well as an additional tool used to support testing and validation.

4.1 Mechatronic Agents

4.1.1 IADE Data Model

As it was stressed before the IADE library is composed by six agents, the CLA and MRA agents are particular cases of the Mechatronic Agent Class (MAC). The MAC is an abstract class (Figure 4.1) that contains the common behavioural logic and communication protocols for those agents. The most relevant data fields that compose this agent include:

- **OMACState:** this field is used to fulfil a conventional automation concept and track the machine status. It is applicable or used when the agent is attached to a physical resource.

- **Skills:** is the list of skills provided by an agent. Each skill present on this list contains the required information for the local execution and publishing purposes.

- **ExecutorEngine:** is the engine responsible for skill execution. This class is abstract, and each mechatronic agent subclass must have its own implementation.

- **NegotiatiorEngine:** is the engine responsible for the negotiation of skills to be executed by other agents. This class, like the ExecutorEngine, is an abstract class and each mechatronic agent subclass must provide its own implementation. This engine is in charge of the proposal evaluation during negotiation and to do the proposal when a negotiation request is received.

- **MyType:** this field contains information about the agent subtype, such as MRA, CLA, TSA, PA or any new type defined in a customized solution. It is used to identify the agent type on the system during the deployment.

- **YellowPages:** is a library that provides the mechanisms to access the YPA services. This library contains the mechanisms that facilitate a parametric querying for agents and skills.

- **ExecutorResponder:** this field is a class that enables the server logic for the MAC agents to be able to receive and reply to execution requests. The protocol supported is the FIPA Request Protocol [51] (appendix 1).

- **NegotiatorResponder:** is the server side of negotiator. Provides the mechanisms that enable mechatronic agents to handle negotiation requests. The protocol supported is the FIPA Contract Net [52] (appendix 2).

- **ExecutorInitiator:** is the client side of executor. This class enables the MAC agents to do execution requests. The protocol supported is the FIPA Request [51] (appendix 1).

- **NegotiatorInitiator:** is the client side of negotiator, like the ExecutorInitiator it enables the MAC agents to perform negotiation requests. The protocol supported is the FIPA Contract Net [52] (appendix 2).

- **ExecutionScheduler:** is the execution manager. This class manages the execution queue ensuring that the skills are executed by the correct order (the order in which they were requested).

As stated before, the CLA and MRA agents are extensions of the MAC. The main difference between them lies in the execution logic. The CLA execution consists in an logic of composite skills and the CLA orders other agents to execute the skills present on its composite skill. The MRA only contains atomic skills. Each skill is related with a hardware function that is executed by a specific controller.

The PA is a special case of a CLA, since it is autonomous from the execution point of view. When launched, the PA starts by the execution of its process and is not able to process other execution or negotiation requests. This agent also includes mechanisms to interact with TSAs, which are supported by the following fields:

- **MoveRequest** is a field that provides the mechanisms for PAs to order transportation requests to TSAs for a given position where the PA desires to execute a skill.

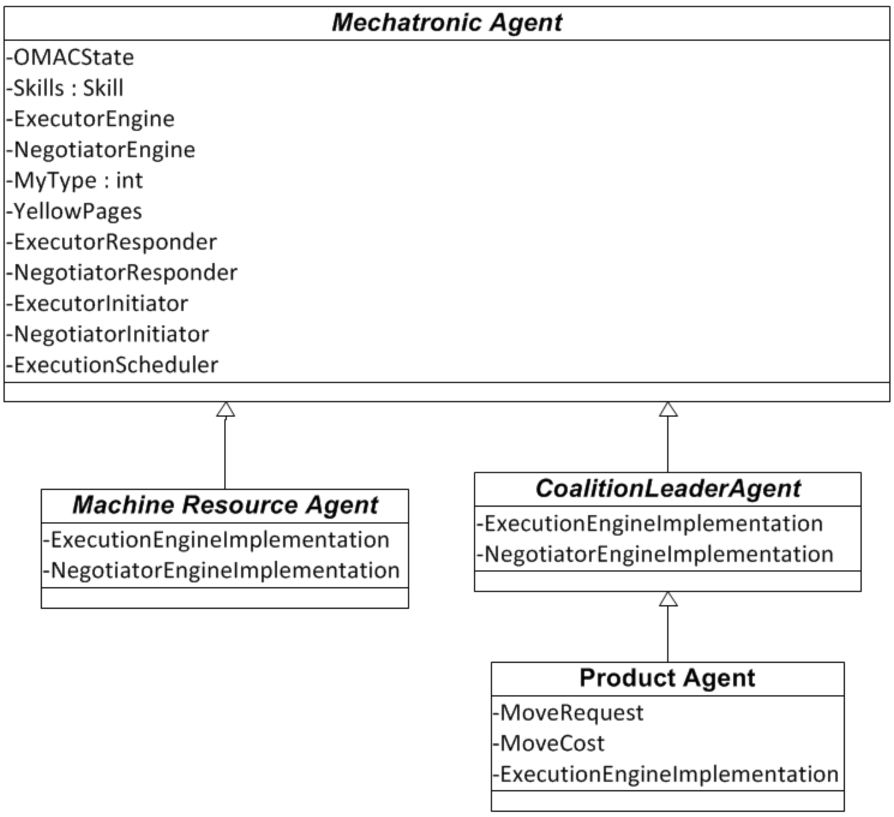


Figure 4.1: IADE - Mechatronic Agent Class

- **MoveCost:** is a field that enables the PA to request the transport costs to various positions. This cost together with the proposals of agents (MRA or CLA) helps the PA to decide which are the best agents to execute the skills on its process.

One of the most significant data structures present in the IADE library is the **Skill** abstract class (Figure 4.2). The skill is a representation of the functionalities that agents provide to the system.

The main fields that compose the skill abstract class are:

- **Name:** this field works from a query point of view as the skill id, however it is not unique since an agent can have more than one skill with the same name but with a different parametrization.
- **Type:** identifies the skill type (CSk, ASk or DSk). It is particularly important for agents to identify the skills on the execution queue and know how do execute it.
- **Owner:** this field is mostly used on skills that are to be executed by other agents. Defines the agents (pre-defined or negotiated) that can execute the skill. When the list is empty the agent will proceed with a negotiation process to allocate the owner.
- **Parameters:** is a list of parameters. The parameters behave like I/O lines, where the values of a parameter on one skill can be affected by the values on another skill.

The parameter is composed by the *name* which is the identifier of the parameter, the *nativeType* identifies the data type of the parameter, *value* is the field that keeps the parameter's value, *lowerBound* and *upperBound* define the lower and upper values acceptable for this parameter, the *type* indicates if it is an input or output parameter, the field *unit* stores the unit type (i.e. kilogram, meter, litre, etc) and the *enumValues* field is a list of enumerated values. The parameter is an important part of the skill parametrization.

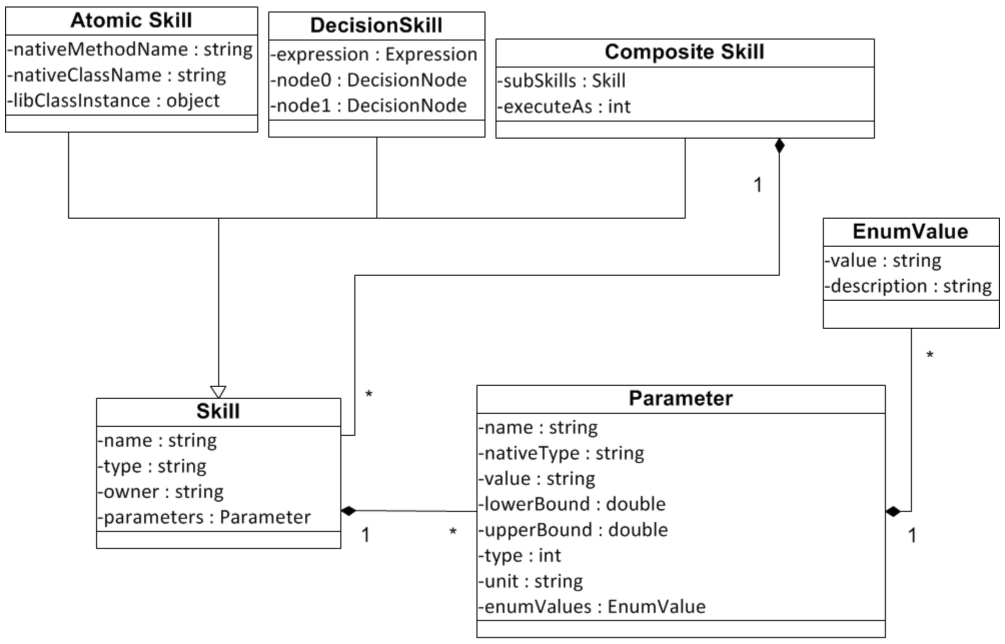


Figure 4.2: IADE - Skill Class

The skill abstract library is extended by three different skill classes. The **Atomic Skill (ASk)** represents an indivisible functionality and is directly related with a hardware resource, the **Composite Skill (CSk)** is a skill that groups others skills. The formed group of skills can be executed in a sequential or parallel logic, and the **Decision Skill (DSk)** is a special skill with an *expression* where the values of process parameters are evaluated.

In order to link the ASk to a hardware resource it contains two additional fields, the *nativeMethodName* that stores the method to be invoked when this skill is to be executed and the *nativeClassName* that has the class name where the method is implemented. The CSk includes a list with *subSkills* and a field *executeAs* that indicates which is the execution logic of this skill (sequential or parallel). The DSk is composed by an *expression* field that stores the condition to be evaluated, the *node0* and *node1* are classes that store the skills to be executed according to result of the evaluated expression, the *node0* is executed when the expression evaluates to true, and *node1* is executed otherwise.

The agents require an infrastructure to expose their skills so that other agents can access them and decide if the skills available fit their needs. The YPA is designed to fulfil the desired requirements for information management on IADE. It supports non blocking queries, which means that, an agent can perform a request without being blocked while is waiting for the answer. It contains a subscription service where an agent can subscribe for events related to others agents and receive notifications when the status of the subscribed agent changes. Each YPA only stores the information relative to its area, and consequently it is possible to reduce the load on YPA and improve the response time.

The Yellow Pages is divided in two parts (Figure 4.3), the server side (YPA) and the client side. The YPA is composed by the following fields:

- **DBControl:** this class is responsible for the management of the skill database. It provides the infrastructure to perform the requests from the others agents.
- **SubscriptionManager:** is responsible to manage the lists of subscriptions. When an agent changes the internal state, this manager sends a message to all subscribers notifying them about the changes.
- **SubscriptionRegister:** this field is a behaviour that receives requests to subscribe or unsubscribe the subscription service.
- **MYAreaName:** identifies the area for which that YPA is responsible.
- **QueryYPResponder:** is a behaviour that responds to query requests. It uses the protocol FIPA Request to do non blocking queries.

The client side is a class that contains the mechanisms to perform the requests and the subscriptions:

- **MyArea:** stores the name of the area where the client is located.
- **SubscriptionResponder:** is a behaviour that receives and processes notifications about agent changes.
- **SubscribedAgents:** this field is a list with the agents for which the notification service has been subscribed.
- **YellowPagesAccess:** is the behaviour responsible to perform queries to the server which includes the operations for registering and de-registering the agent.

The communication between the server and the client uses a special structure. The *Service Template* defines which service is being required (query or data change operation). This is stored on the *service* field. The *AgentInfo* field is a class which stores the basic information about a mechatronic agent (name, type and status) and the *SkillTemplate* field is similar to Skill class but only with the relevant information that is used to perform queries to Yellow Pages.

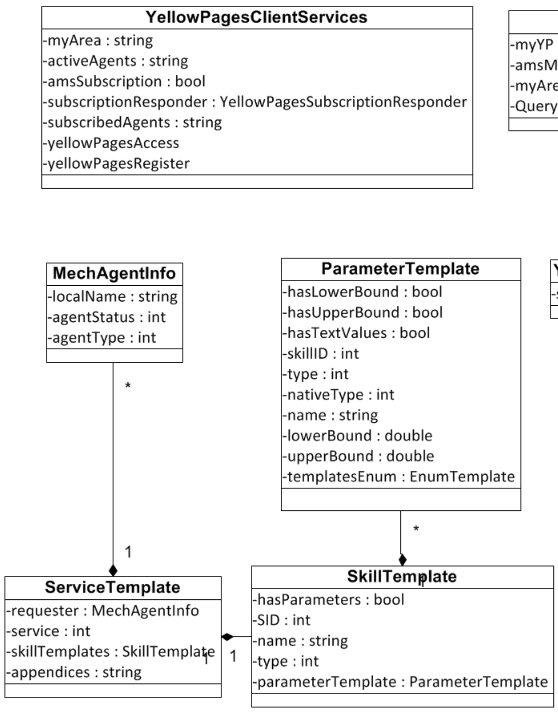


Figure 4.3: IADE - Yellow Pages classes

The AgentInfo and SkillTemplate behave like hybrid fields, they can contain input information to the YPA or output information returned by the YPA.

The TSA (Figure 4.4) is an abstract class and is the core of the Transport System (TS). The TS is composed by four agents that are extensions of the TSA. They are the *Transport Entity Agent (TEA)*, *Handover Unit Agent (HUA)*, *Product Source Agent (PSoA)* and *Product Sink Agent (PSiA)*. The PSoA and PSiA are particular cases of the HUA, they allow PAs to enter or leave the system.

The TSA comprises the most fields that are used by all agents in the TS:

- **Area:** this field stores the area name where the TSA operates.
- **Capacity:** this field is a constant related with the physical limitations of the transport system. It stores the amount of products that fit in the part of the system that is controlled by the TSA.
- **Queue:** is a list with products, sorted by arrival time, that are present on the TSA.
- **MoveRequest:** this field is a behaviour that receives requests to move products. This behaviour can receive the request directly from the product or by an intermediary (i.e. the case when the product destination is not located on the same TEA where the PA currently is).
- **TablesReceiver:** is a behaviour that is responsible to handle the tables with the costs sent by its TSA neighbours.

The TEA comprises a set of locations that define the docking points where the skills are executed. It is also responsible for computing the traversing costs between the docking points and inform its neighbours. Like the MRAs, the TEAs have generic mechanisms that allow them to control the equipment. The TEA by itself cannot handle the points where the products enter or leave the system. For that purpose, the system contains two additional agents. From a data model point of view, the additions of a PSiA and a PSoA imply that the TEA must have a behaviour that receives a request to put a new product (*NewProductResponder*) on the system and another one to take out a product from the system (*ExitProductResponder*).

The TEA has the following fields:

- **MoveCost:** is a behaviour that responds to requests about the transport cost to a given skill and replies with the cost for all known agents that are able to perform the execution of that skill. This cost will be fundamental for the requester agent to decide where the skill will be executed.
- **AgentRegisterResponder:** this field is a behaviour that handles requests from agents that register on docking points.
- **YellowPagesClient:** is the client of Yellow Pages. After receiving a register request, the TEA needs to know which skills that agent provides, and that information is stored on the YPA.

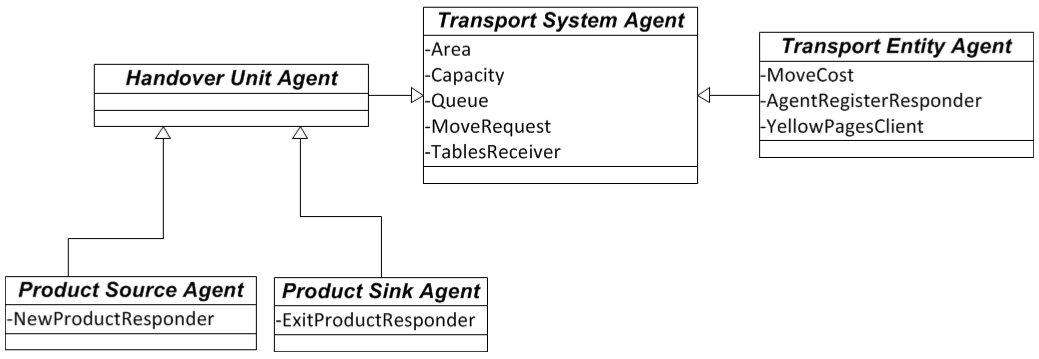


Figure 4.4: IADE - Transport System classes

The HUA is responsible for controlling the points where routing decisions are taken, that means the HUAs are responsible for doing the transitions of products between TEAs. The HUAs like the TEAs and MRAs are hardware dependent which means that they have to be able to handle different physical systems. The generic mechanisms that enable these agents to manipulate any hardware are out of the scope of this work and have been documented in [53].

4.1.2 IADE Main Interactions

In order to ensure the desired requirements such as, self-organization, emergence, reconfigurability and robustness, the system demands a refined communication protocol. In this subsection, the main interactions present on IADE are described. The IADE interactions are designed to minimize the amount of information and messages exchanged, and therefore reduce their impact in the overall performance of the system. This is fundamental since agent-based systems tend to be extremely expensive from a communication point of view.

Figure 4.5 illustrates the message sequence that occurs during the launch of CLAs and MRAs. The CLA1, when launched, will perform its registration on the YPA and if necessary will subscribe for notifications about specific agents. When the MRA registers the CLA1 receives the notification message about a change. Finally the CLA2 starts the registration process and, like the CLA1, subscribes for notifications about MRA. In this case the MRA is already registered so the notification message is sent upon agent registration. All these interactions use the FIPA Request Protocol [51] (appendix 1).

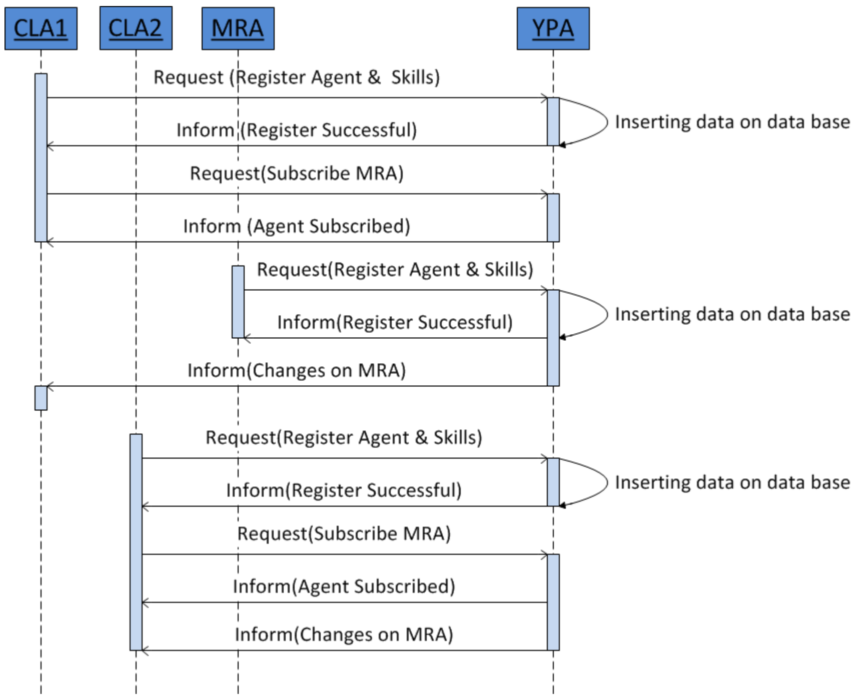

CLA1
CLA2
MRA
YPA
Request (Register Agent & Skills)
Inform (Register Successful)
Inserting data on data base
Request(Subscribe MRA)
Inform (Agent Subscribed)
Request(Register Agent & Skills)
Inform(Register Successful)
Inserting data on data base
Inform(Changes on MRA)
Request(Register Agent & Skills)
Inform(Register Successful)
Inserting data on data base
Request(Subscribe MRA)
Inform(Agent Subscribed)
Inform(Changes on MRA)

Figure 4.5: IADE - Registering and Subscribing for Agents

The following interaction (Figure 4.6) takes place when a CLA or MRA is plugged into a docking point associated with a TEA. The first step is to send a message to the TEA informing which docking point will be used. Once the plugging process is done the TEA requests to the YPA the information about the skills that the plugged agent provides. Again, the FIPA Request protocol is used.

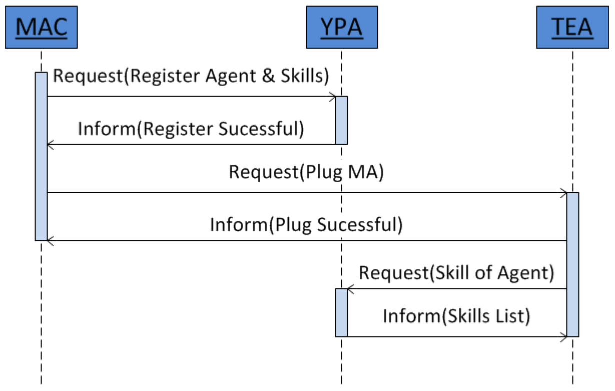


Figure 4.6: Plugging Agents in docking point

As soon as the agents are registered and plugged they are ready to execute their process. At this point, the CLA can directly request the execution of a skill or may need to negotiate.

It is important to recall from the architecture and data model that, in a composite skill (and the CLAs always manage composite skills), the owner field defines if negotiation needs to occur. If the owner has a value then the CLA will immediately choose that agent for the skill execution. If, otherwise, the value is not defined, the CLA will start the negotiation procedure to allocate an owner to the open skill description. The negotiation (Figure 4.7) follows the FIPA Contract Net Protocol [52] (appendix 2).

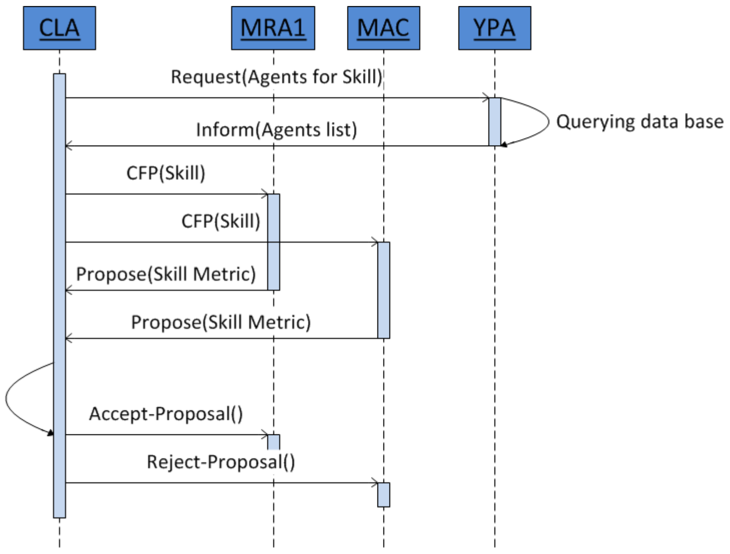


Figure 4.7: IADE - CLA Negotiation

The CLA negotiation process (Figure 4.7) consists in requesting a proposal from all the agents that can execute the skill and belong to the same area. As a one-to-many process, the negotiation implies that the CLA has a list of all the agents in the area that have a skill with the proper parametrization. The agents receiving the call for proposals message will return a proposal based on their operational condition, which in the present case is related with their workload. The CLA will finally allocate the agent that has issued the best offer and reject all the other proposals.

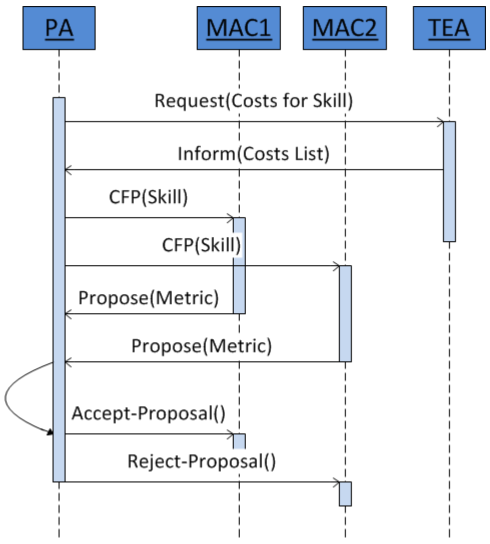


Figure 4.8: IADE - PA Negotiation

The PA negotiation process (Figure 4.8) follows the same logic as the one described for the CLA, but the PA is able to negotiate with all agents present in system. In order to find agents that can execute a skill, the PA requests to the TEA that is currently associated with it a list of all the transport costs and, the associated agents that can implement the desired skill. The following process is the same as for CLA, however, in the evaluation process the PA takes in consideration the transport cost as well. The PA will therefore combine both costs and commit to an execution location.

As soon as the negotiation is successful the PA requests the TEA currently associated with it (Figure 4.9) to transport it to the desired agent. The transport process triggers a set of interactions between the agents in the transport system to where those agents will drive and handover the PA until reaches the desired destination. When the transport process is done the PA receives an Inform with its location. This location corresponds to the new TEA that is logically and physically responsible for the PA. The PA is now ready to request the skill execution.

All these interactions use the FIPA Request Protocol, except for the last Inform message, that is sent straightforwardly without following any protocol. It is important to recall that in JADE, the message exchange never fails and that, delivery is ensured therefore the last message can simple be a one way message.

The execution interaction illustrated on Figure 4.10 exemplifies a situation where concurrency occurs. The first agent to perform the request will receive an agree by the requested agent, if there is a second agent performing a request before the execution is

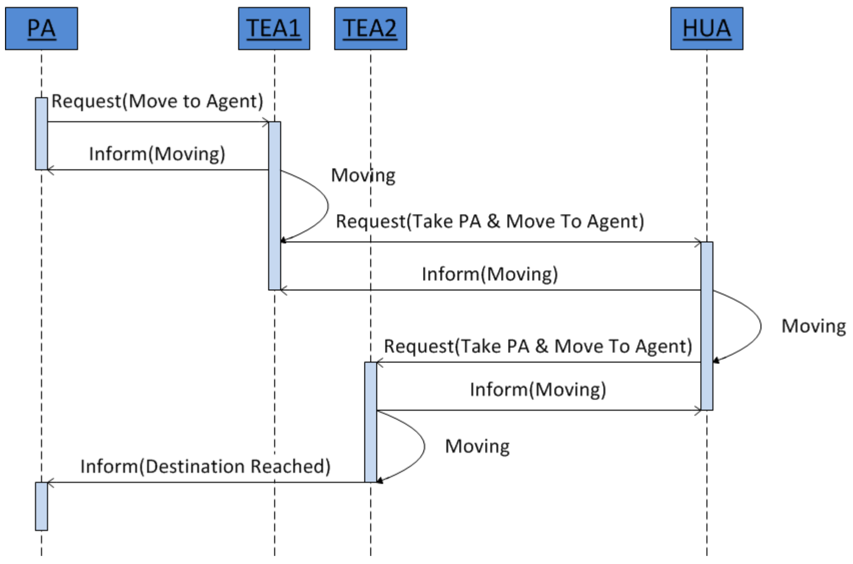


Figure 4.9: IADE - PA Moving

done, that agent receives a refuse and should try again or look for another agent to execute the skill. This applies to docking points where the agents executing therein can buffer PAs.

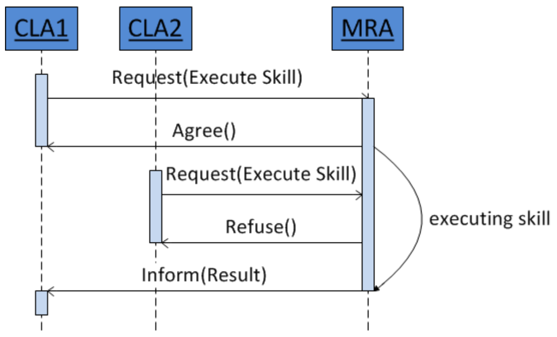


Figure 4.10: IADE Execution

These interactions and implementation enclose the contribution of this thesis for the IADE stack and, as later detailed, have been tested in the FP7 IDEAS project prototypes. The implementation described in the next sub-section relates with the simplified version of the stack and other supporting tools for the emergence and self-organization assessment.

4.1.3 Reduced Stack - Data Model

The work described in this sub-section relates with the more scientific component of this work.

The main differences between the IADE stack and the reduced stack reside almost only at PA, TSA and YPA level. The CLA and MRA remain almost without any changes

(Figure 4.11). The main changes at CLA and MRA level have to do with the introduction of the proposed metric, and subsequent adjustments to the negotiation procedure. To maintain the integration with IADE, two separate classes CLA* and MRA* were created. The new PA is also derived from the CLA*. Since the transport agents are out of the scope of this thesis but they are an important part on physical systems, it was necessary to adapt them in the form of a unique class called Transport System Agent (TSA).

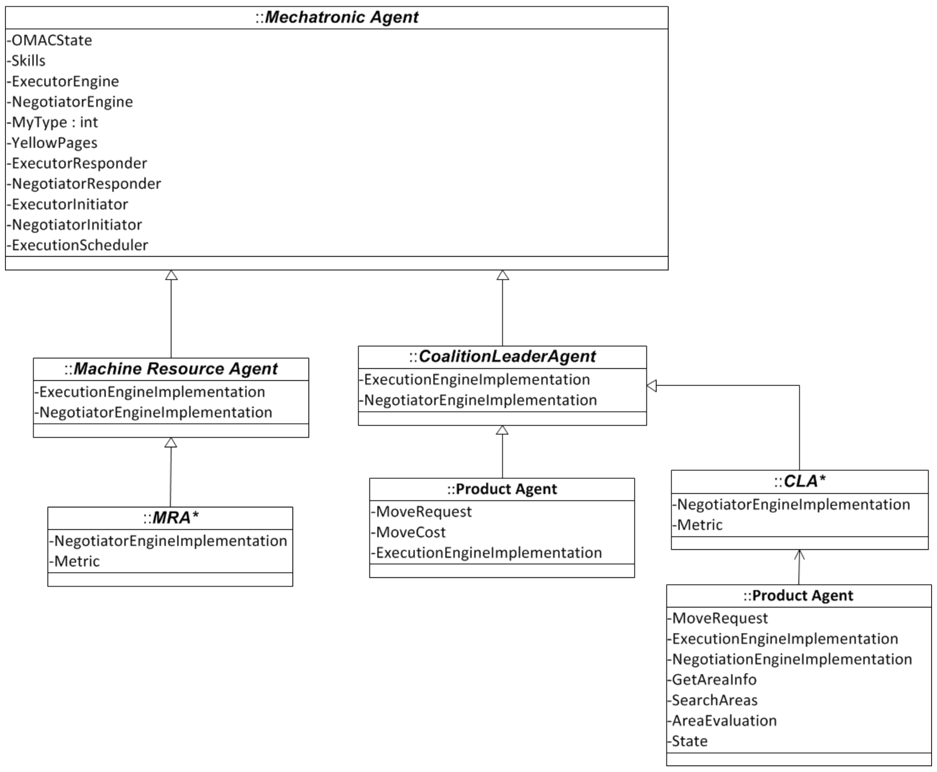


Figure 4.11: Simplified IADE Stack - Mechatronic Agent Class Diagram

Like in IADE, the TSA is responsible for managing several stations (abstracted as agents) which execute in specific docking points. Each station is associated with an YPA. Instead of CLAs and MRAs plugging into TSAs, they register themselves on the associated YPA. The TSA is responsible for seeking skills on the YPA and to make the PA know about the stations that can provide the skills required by the PA, however, that information is only available when requested. The fields that ensure the main operations by the TSA are:

- **Areas:** is a list with the areas (in this particular case, a list of YPAs) that are attached to the TSA.
- **PAQueue:** is a list where the products are ordered by entrance, the products are only allowed to do moving requests when they are the at first position of the queue.
- **AgentLocations:** this field stores the location of products. It only stores the location of products that are not on the PAQueue list.

- **GetAreaInfoResponder:** this is a behaviour that returns a list of areas that can handle a set of skills.
- **MoveProductResponder:** this field is a behaviour that replies to transport requests from products.
- **NewAreaResponder:** this field is a behaviour that handles requests for plugging new areas and their agents.
- **UpdateAreaResponder:** is a behaviour that is triggered when the agent receives a notification about any change on agents. The TSA subscribes for all agents on the YPA.

One of the main differences in the PA is the negotiation procedure. Instead of negotiating the skills one by one, as occurs in IADE, it negotiates sets of skills to optimize a wider set of executions. That allows the PA to consider the execution of a set of skills on the same area without negotiating in between.

The negotiation process is performed for a set of skills, and it only takes into consideration the largest subset of skills that can be ensured by at least one area. The areas that cannot execute the largest subset are considered, but the execution cost of the missing skills is replaced by the best computed for the other areas.

To better explain the process used to select the optimal area, the example in Figure 4.12 is considered. The system illustrated in Figure 4.12 is composed by three areas, **Area A**, **Area B** and **Area C**. Each area has three skills, provided by three top level CLAs. The Table 4.1 summarizes the system. A PA named PA-1 with the following process plan *Pick and Place, Glue, Screw, Test and Tag* is assumed. For this example, it is also assumed a capacity to handle three PAs simultaneous at area level. Two different cases will be considered:

- **Case 1**: All areas have at least one free spot.
- **Case 2**: Area A is running at full capacity.

The first step to select which agents will execute the PA skills, is to gather all the agents in the system that provide the required skills. However, at each instant, the system does not consider agents in an area that is running at full capacity. This process is described in Algorithm 1.

In case 1, all areas will be returned when the method *getFreeAreas()* is invoked, in case 2, only B and C areas are returned. The algorithm will then proceed with a search through the returned areas for skills that match with the PA-1's requested skills. The Algorithm 1 will return all the matching skills.

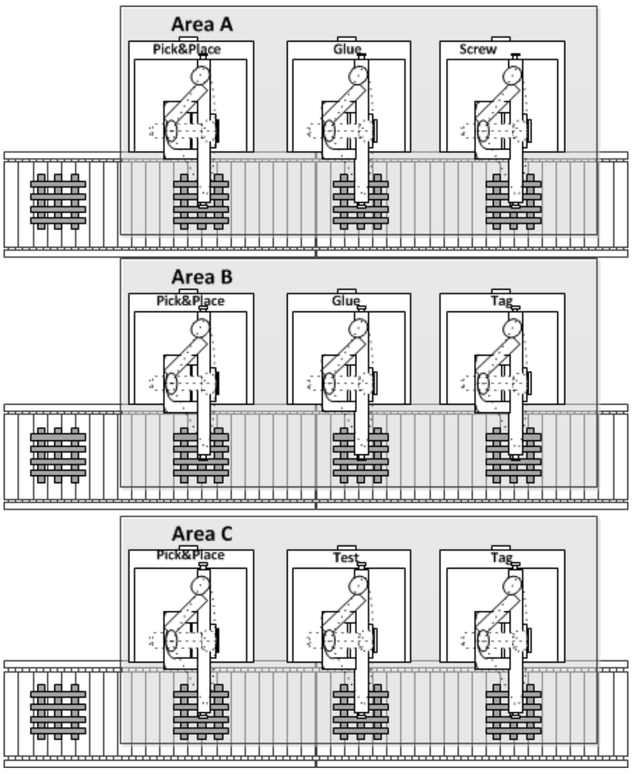


Figure 4.12: System example

In the case 1, all skills in the pairs, CLA and skill, are returned, since all of them have a positive match with the requested skills. In case 2, the Area A is not considered, but all the pairs, CLA and skill, in area B and area C are returned.

Table 4.1: Metric Values Example

CLA	Skill	Metric Value	Area
CLA-A1	Pick and Place	0.5	Area A
CLA-A2	Glue	0.8	Area A
CLA-A3	Screw	0.3	Area A
CLA-B1	Pick and Place	0.5	Area B
CLA-B2	Glue	0.6	Area B
CLA-B3	Tag	0.6	Area B
CLA-C1	Pick and Place	0.8	Area C
CLA-C2	Test	0.3	Area C
CLA-C3	Tag	0.5	Area C

The second step to select the optimal area is processed at PA level, the Algorithm 2 describes this step. It consists in negotiating with all the agents returned by the TSA, and computing which area offers the best metric. Although the PA requested the agents for all the remaining skills on its process, not all skills will be taken into consideration when the metric is computed. For instance, in case 1, only the skills *Pick and Place*, *Glue* and *Screw*

```
Data: $requiredSkills$
Result: $areasInfoList$
$availableAreas \leftarrow getFreeAreas();$
for $skillIndex \leftarrow 0$ to $requiredSkills.size()$ do
    for $i \leftarrow 0$ to $availableAreas.size()$ do
        $area \leftarrow availableAreas[i];$
        $areaInfo \leftarrow newareaInfo(area.name);$
        $areaSkills \leftarrow area.getSkills;$
        for $j \leftarrow 0$ to $areaSkills.size()$ do
            $skill \leftarrow areaSkills[j];$
            if $skill == requiredSkills[skillIndex]$ then
                $agent \leftarrow area.getAgentForSkill(skill);$
                $areaInfo.add(skill, agent);$
            end
            else
                $breakfor;$
            end
        end
        $areasInfoList.add(areaInfo);$
    end
end
```

Algorithm 1: Reduced Stack - Areas Selection at TSA level

can be executed in the same area at once, without the need for the PA to switch to another area, so only those skills will be considered. In case 2, only *Pick and Place* and *Glue* skills can be sequentially executed in one area. The excluded skills, will be postponed to future negotiations.

After selecting the skills to be negotiated, the negotiation with the agents that own that skills is initiated. When the PA receives all the responses, it will proceed with the evaluation of the metric value for each area regarding the negotiated skills. It is important to note that even the areas that do not have all the negotiated skills will be included in the evaluation.

When an area is missing skills, to perform a fair comparison between all the areas, the value used for the missing skills is the best computed value for the missing skill in the entirely system. In the given example, in the case 1, the *Screw* is missing in B and C areas, and the *Glue* skill is missing in area C. In the case 2, only the *Glue* skill is missing in area C.

To simplify, in the given example, it is assumed that the overall metric value for each area is a simple arithmetic addition of all individual agent metric values. The Table 4.2 and Table 4.3 illustrate the metric values returned on the negotiation process for each considered case. These metrics are presented on Table 4.1. The **x** value is related to a missing skill.

```
Data: $requiredSkills$
Result: $\{CLA_1...CLA_n\}$
$CLAs \leftarrow TSA.search(RequiredSkills)$;
$sortedCLAs \leftarrow sortByArea(CLAs)$;
$skills \leftarrow slctLargeSetOfSkill(sortedCLAs, requiredSkills)$;
$CLA_{neg} \leftarrow slctCLAs(sortedCLAs, skills)$;
$areas \leftarrow TSA.getAllAreas()$;
for $i \leftarrow 0$ to $CLA_{neg}.size()$ do
    $CLA \leftarrow CLA_{neg}[i]$;
    $startCNET(CLA)$;
end
$result = asynchWaitForNegotiation()$;
for $i \leftarrow 0$ to $areas.size()$ do
    $mSkills \leftarrow MissingSkills(areas[i], skills)$;
    if $mSkills.size() > 0$ then
        $addMissingSkills(mSkills, results[i])$;
    end
    $results[i] \leftarrow ComputeD_m(areas[i])$;
end
$\{CLA_1...CLA_n\} \leftarrow EvaluateBestPerArea(results)$;
```

Algorithm 2: Reduced Stack - Decision Process at PA level

As it was previously stated, for the missing skill metric values, the best metric value in the system regarding the missing skill will be used. In case 1, in Area C, the *Glue* skill uses the metric value returned by the CLA-B2, which is the best value in the entirely system for that skill. For the skill *Screw*, the best value is provided by the CLA-A3, and that value is considered to compute the overall metric value in area B and C.

In case 2, only area C has a missing skill. The used value in area C for the *Glue* is the one provided by the CLA-B2.

Table 4.2: Metric values - Case 1

Area/Skill	Pick and Place	Glue	Screw
Area A	0.5	0.8	0.3
Area B	0.5	0.6	x
Area C	0.8	x	x

Table 4.3: Metric values - Case 2

Area/Skill	Pick and Place	Glue
Area B	0.5	0.6
Area C	0.8	x

The final computed values for the overall metric and for each area are illustrated at the Table 4.4 and Table 4.5. In case 1, the area that provides the best metric is the area B, although this area does not have all the skills, the system considers that is best to execute the *Pick and Place* and *Glue* in area B, and then negotiate again the *Screw* skill. It is important to note that if the transport cost would have been considered, the results in this case could be different and the three skills could have been executed in the same area.

In case 2, the area B offers the best metric value, and it will be chosen to execute the negotiated skills. As soon as the PA-1 executes these skills, the entirely process will be repeated for the remaining skills. It is important to note, that each successful or unsuccessful negotiation will affect the individual metric value of the agents.

Table 4.4: Total Metric values - Case 1

Area/Skill	Pick and Place	Glue	Screw	Total
Area A	0.5	0.8	0.3	1.6
Area B	0.5	0.6	0.3	1.4
Area C	0.8	0.6	0.3	1.7

Table 4.5: Total Metric values - Case 2

Area/Skill	Pick and Place	Glue	Total
Area B	0.5	0.6	1.1
Area C	0.8	0.6	1.4

To handle the negotiation process described above, some modifications (Figure 4.11) have been introduced in the PA. The following fields are the main fields that compose the PA:

- **GetAreaInfo:** is a behaviour that requests a list of areas that can execute a given subset of skills.
- **MoveRequester:** this field is a behaviour that performs the transport requests.
- **SearchAreas:** is a behaviour that is triggered when the agent is the first of the TSA's queue. Is responsible for preparing a list with the remaining skills to be executed and launch the behaviour that performs the communication.
- **AreaEvaluation:** this field is responsible for the area evaluation.
- **State:** this field stores the agent's state.

The operating logic of the PA follows a finite state machine (Figure 4.13). On the initial state the agent requests the TSA to introduce it on the system. When the introduction process is done the agent transits to the *IDLE* state, on this state the agent is on idle waiting to reach the first position of TSA's queue. In the following state, *Requesting Areas*, the agent will search for areas that can execute the largest sub skill set.

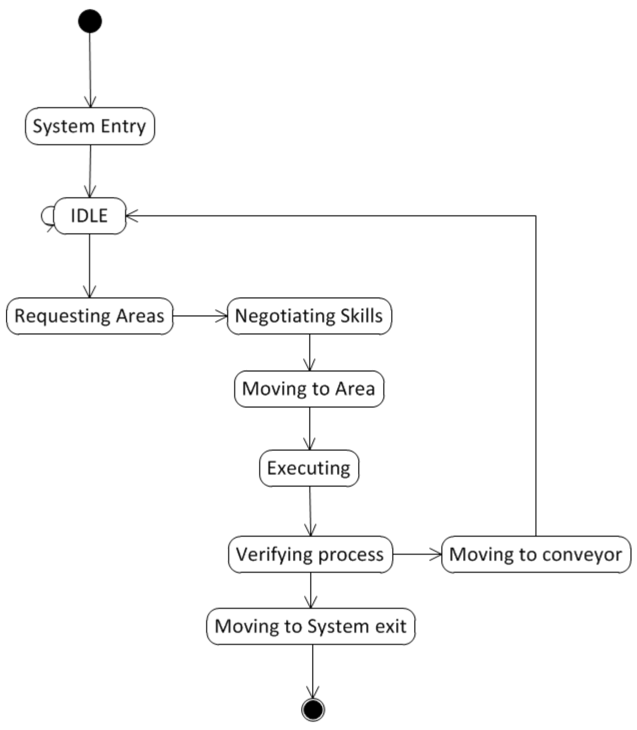


Figure 4.13: Simplified IADE Stack - Product Agent state machine

The *Negotiating Skills* state is triggered after the agent receives the areas information, and it is on this state that the agent proceeds to negotiate and evaluate the areas in order to choose the best solution for executing. As soon as the areas are evaluated and chosen, the agent transits to *Moving to Area* where the agent requests to the TSA a transport and waits until that transport is finished.

The PA state transits to the *Execute* state when the TSA finishes the transport. On this state the PA is able to perform the execution requests for the agents that are in that area. Once the execution is done, the PA triggers the *Verifying Process* state where the agent executes a verification if the process contains more skills to execute, in this case the PA will trigger the *Moving to Conveyor* state where the agent requests the TSA to transport it back to conveyor. The agent then returns to *IDLE*. In case the PA process is finished the agent transits to *Moving to System Exit*. On this state the PA requests the TSA to transport it out of the system.

The YPA (Figure 4.14) inherits the model and functionalities present on IADE and includes a *Capacity* and *NewArea* fields, where the *Capacity* stands for the number of products that can fit on that particular area and the *NewArea* is a behaviour to do the registration on the TSA.

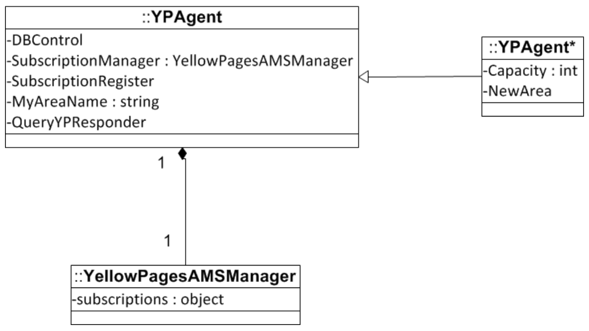


Figure 4.14: Simplified IADE Stack - Yellow Pages Agent Class Diagram

4.1.4 Reduced Stack - Interactions

Unlike IADE where the YPA and the TEA have a one to one relation, in the reduced stack, as stressed before, the YPA stands for an area and encloses the stations and their agents. The Mechatronic Agents also have the capacity of handling several PAs. In Figure 4.15 is illustrated the protocol for YPA registration.

The YPA starts by requesting a registration and informing the TSA about its capacity. It then receives a inform to confirm the registration. The TSA then subscribes for all the agents that can be part of an area, in order to receive a notification for all the changes.

As soon as the MAC performs the registration, the TSA receives the subscription notice and proceeds to request the information about the new agent that operates on that station.

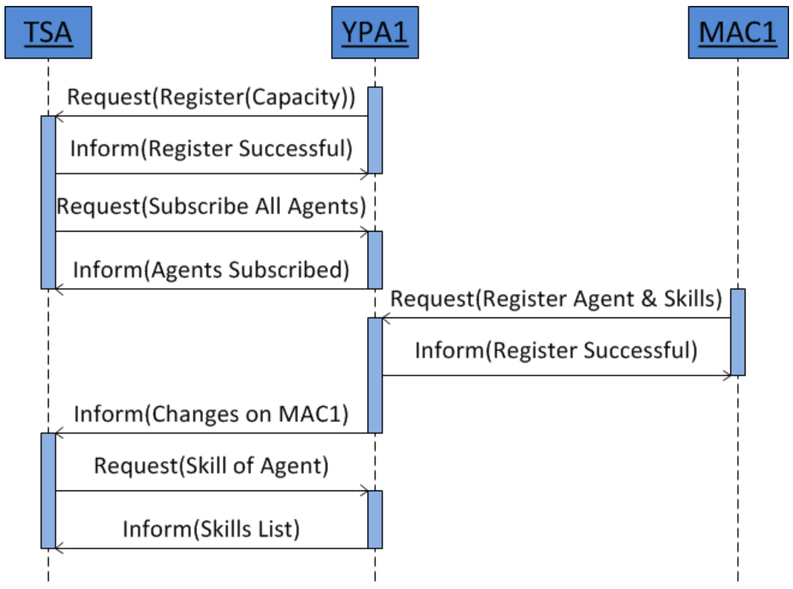

TSA
YPA1
MAC1
Request(Register(Capacity))
Inform(Register Successful)
Request(Subscribe All Agents)
Inform(Agents Subscribed)
Request(Register Agent & Skills)
Inform(Register Successful)
Inform(Changes on MAC1)
Request(Skill of Agent)
Inform(Skills List)

Figure 4.15: Simplified IADE Stack - Interactions between YPA and TSA

Each interaction between PAs and TSAs is established through a simple FIPA request protocol. There are two different interactions that can occur between those agents. The

Figure 4.16 represents the particular case where two PAs attempt to request information about a station.

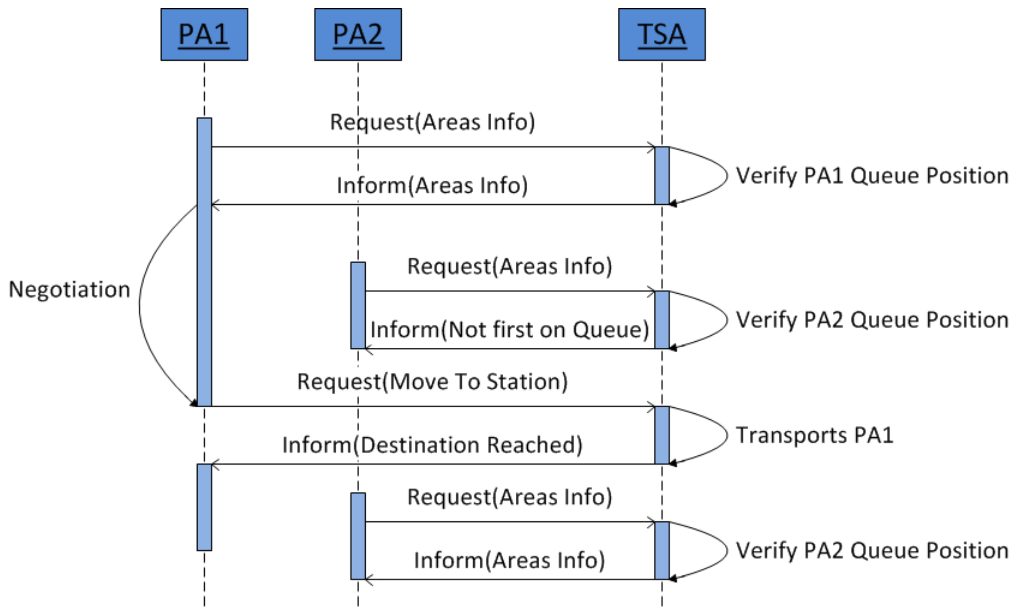


Figure 4.16: Simplified IADE Stack - Interactions between PA and TSA

A positive response results in an authorization for a PA to do the negotiation and order a transport into a station. The case illustrated assumes that PA1 is the first agent on the queue and PA2 is the next. When PA1 requests the information about the areas, the TSA replies favourably with the information about the stations.

The PA2 attempts to do a request about the area info, but receives a negative response since it is not the first in the queue. After receiving a negative response the PA2 waits a certain time before issuing the same request.

The PA1 then starts to negotiate with all agents of interest. As soon as the negotiation process is complete the agent requests a transport to a certain station and receives an inform when the process is done. Afterwards the PA2 jumps to the first position on the queue and is able to do requests and negotiate.

The Figure 4.17 illustrates the interactions between the PAs and MACs at negotiation time. It is a simple Contract-Net Protocol, where the PA requests the metric values for several MACs and waits for each response. As soon as it gets all the responses the PA proceeds with the metric evaluation as previously described.

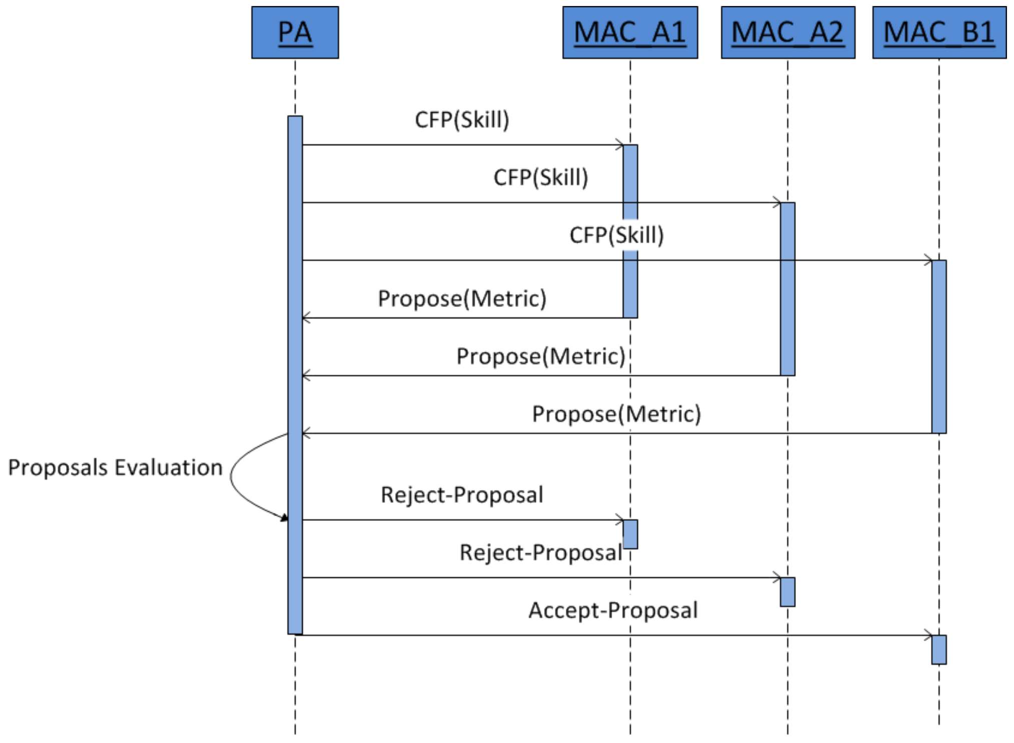


Figure 4.17: Simplified IADE Stack - PA Negotiation Interactions

4.2 A tool to generate networks of skills

In the next chapter a set of tests and results are detailed. One of the tests relates with the performance degradation when skill composition is considered in a loosely way. It is therefore important to be able to carry out testing in distinct skill trees with different characteristics. In this context, this subsection details a support tool, that was implemented as part of this work, that creates random skill trees, and enables their execution on the reduced IADE stack, for the purpose of improving the analyses of the results.

4.2.1 Concept

The Figure 4.18 examples a potential system implemented through the IADE library. In this example the system is composed by several entities that are able to interact among them.

The system exemplified can be decomposed into indivisible entities and a mapping between skills and agents is possible until the MRA level is reached (i.e. tools, the magazine, manipulater, etc). The relation between an indivisible entity and a MRA will be exposed as an atomic skill (i.e. moveto, grip, dispense, glue, etc). The pallets in Figure 4.18 represent the PAs. The PA's process, as already detailed, is a composition of the skills available in the system. In this particular example the composition could be: dispense glue, pick and place, glue, welding, etc.

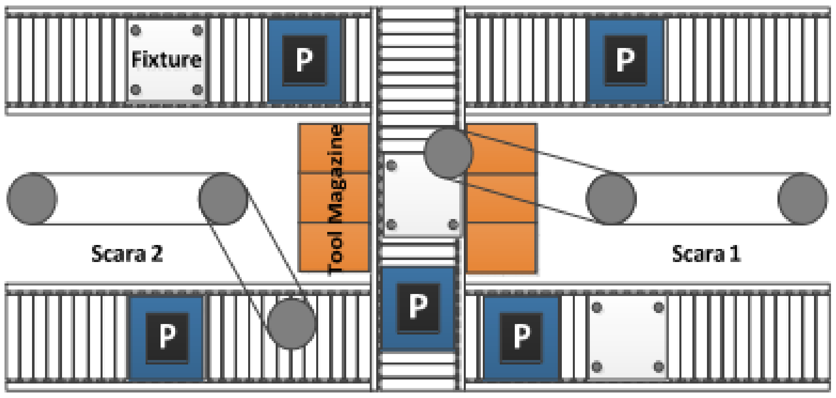


Figure 4.18: System Example

The link between the PAs and the MRAs is mediated through several layers of CLAs where gradually the lower level skills are composed as higher compositions, as illustrated on Figure 4.19. In this example a competition between the PAs to grab the CLAs in order to execute their skills implies that the PAs will inevitably have to wait for the CLAs to become free and then complete their process.

This very brief example shows that, if not done properly, composition can have an enormous effect on performance. Fundamental aspects that affect the execution's efficiency include:

- depth levels: each dept level considered in the skill tree contributes to encapsulate the complexity of the underlying process. However, it also means that an instance of a CLA will be deployed to manage that subskill. This obviously entails a new communication link being established and the associated computing and message round trip time (RTT) overheads.

- level degree: level degree is a measure of resource sharing in a the layer immediately below. If one level's degree is three, this means that each agent in that level connects on average to three agents on the layer below. If the layer below does not have the required number of agents to support a one-to-one relation, this implies that the agents in the layer above are sharing resources and this resources have to be managed and used to ensure mutual exclusion. In this context, a lot of resource sharing entails waiting times.

The design of the skills is therefore fundamental in balancing the trade-off between performance and the ability of the system to reconfigure itself. That is, from a self-organization point of view, an implicit definition of the desired macrostates.

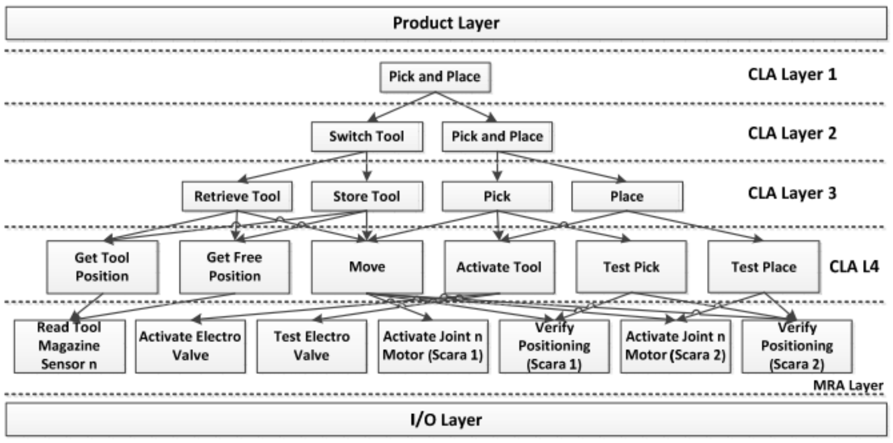


Figure 4.19: Decomposition of the Pick and Place Skill

4.2.2 Network Generator Algorithm

The Algorithm 3 generates random networks which follow the structure of skills composition exemplified on Figure 4.19. The proposed algorithm takes as inputs the network *depth* which means the number of agent layers. The first layer corresponds to PAs and the last layer are MRAs. The middle layers stands for CLA. The algorithm also receives as input data the number of nodes and connection degree for each layer (*nodesPerDepth* and *degreeDepth*).

```
Data: $depth, nodesPerDepth[], degreeDepth[]$
Result: $Network$
$nodes[][] \leftarrow initiliazeNodes(depth, nodesPerDepth)$;
for $i \leftarrow 0$ to $depth - 1$ do
    for $j \leftarrow 0$ to $nodesPerDepth[i]$ do
        $node \leftarrow nodes[i][j]$;
        while $node.degreeSize() < degreeDepth[i]$ do
            $connectingNode \leftarrow randomNode(node, nodes[i+1])$;
            $prob = randomProb(nodes[i+1], connectionNode)$;
            $chance = random()$;
            if $prob > chance$ then
                $node.addConnection(connectingNode)$;
            end
        end
    end
end
```

Algorithm 3: Random Agents Network Generator

After initializing the nodes the next step is to interconnect the layers. In this case it is only possible to connect two nodes that are in adjacent layers. The node's connections are chosen by a random process. The *randomNode* method returns a random node that is in the layer below of the node that is creating the outgoing connections. The method ensures that the nodes always receive at least one incoming connection by returning first the nodes without any connection before returning nodes already with incoming connections.

As soon as all the nodes have at least an incoming connection, and if is still needed to create more connections between nodes, all the nodes will be iterated until all the incoming connections are established. However, if a node already has a connection, it only receives another connection under a certain probability. The probability is higher if the number of incoming connections of the current node is lower than the average of incoming connections of the remaining nodes. This ensures different network topologies but with homogeneous in degrees.

5 Results and Validation

This chapter details the tests considered for the validation of the implementation. The chapter is divided in two main sections that cover two different aspects of the validation procedure.

The first subsection describes the pre-industrial demonstrators in which the MAS infrastructure was deployed and ran. It details the agents and the skills considered in the different scenarios. This section shows that the architecture can be efficiently implemented and can be applied to different systems without requiring changes to the agent stack.

The second subsection studies and shows evidence of the emergent properties and self-organizing ability of the system. This section is mainly supported by simulation and provides a quantitative view on the architecture / implementation response.

5.1 Industrial Demonstrators

In the IDEAS project the IADE stack was tested under three demonstrators. In this subsection the demonstrators are described.

5.1.1 Festo Cell

The Festo Cell (Figure 5.1) is a physical test system, but is not equipped with a real transport system. From an IADE stack point of view it is only possible to test the interactions between the PAs and CLAs, PAs and MRAs, and finally between CLAs and MRAs.

This system is composed by a feeder and two cylinders (Figure 5.2). The purpose of

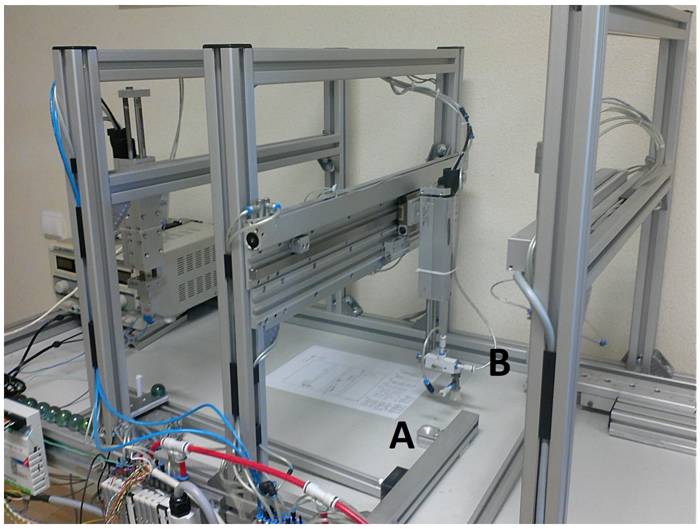


Figure 5.1: Festo Cell

this system is the transportation of a product between the feeding position into positions A or B. In order to perform the transportation *cylinder 1* is equipped with a gripper and an axis that move vertically along the Z dimension. This cylinder grabs the product from the *feeder* and waits for it to change position to release the product into the gravity feeder. The *cylinder 2* is similar to *cylinder 1*. However, it contains one more axis that allows the horizontal movement along the Y axis. The system also contains two breaks located on positions A and B. The *feeder* has two positions, the position 1 (Figure 5.3(a)) where an operator or other machine should load a product, and the position 2 (Figure 5.3(b)) for the pick and place operation.

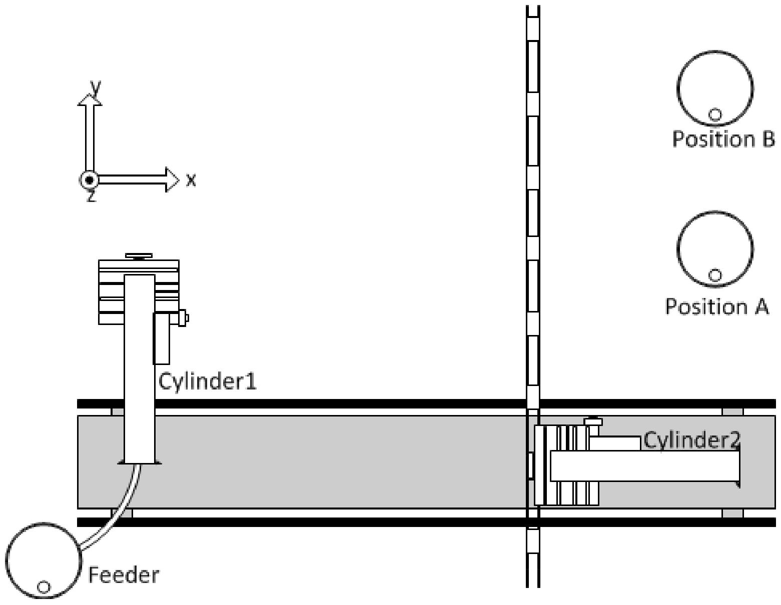


Figure 5.2: Festo Cell Layout

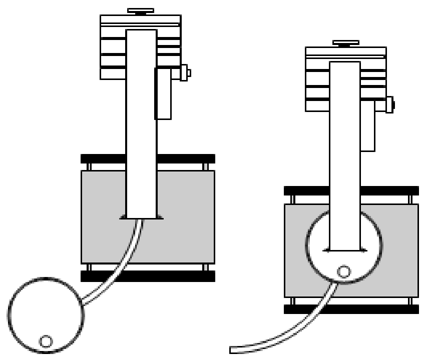

(a) Position 1 (b) Position 2

Figure 5.3: Festo Cell - Feeder positions

The IADE hierarchy for this system comprises several MRAs, such as:

- **Feeder:** this agent controls the feeder resource and provides two skills. The *load* skill, changes the feeder position to position 1. The *unload* skill, changes the feeder position to 2.
- **Gripper1, Gripper2:** are the agents responsible to control the gripper of both cylinders (cylinder 1 and cylinder 2 respectively). They contain the *openGripper* skill and the *closeGripper* skill. As it is possible to deduct, these skills open or close the corresponding mechanical grippers.
- **AxisZ1, AxisZ2:** control the axis of the cylinders (cylinder 1 and cylinder 2 respectively), the skills provided by these agent are the *moveUp* and *moveDown*.
- **AxisY2:** like the AxisZ2 this agent is attached to cylinder 2 and it is responsible to control the Y axis containing the *moveLeft* and *moveRight* skills.
- **BreakA, BreakB:** these agents are respectively for the breaks on position A and B. They provide *break* and *releaseBreak* skills.

The following configuration step consists in the designing of the processes that make use of the agents skills described. In this architecture, is not desired that the product agents access directly to lower order skills, unless in cases where a bigger composition is not possible. Figures 5.4 and 5.5 illustrate a possible skill composition for this particular system.

As shown in Figure 5.4 the *pick and place* skill is placed on the higher CLA layer, this is the skill that should be part of the process of the PAs. This skill is a composition of two skills on the layer below. The *pick* and the *place* skills are built as a composition of skills located on the MRA layer. The *pick and place* also uses directly two skills located on MRA layer, because those skills are not adequate for a skill composition on the layer above.

The composition of skills from the agents that compose the second parf of the system, also leads to a *pick and place* skill at the higher layer. The *pick* and the *place* skills have a

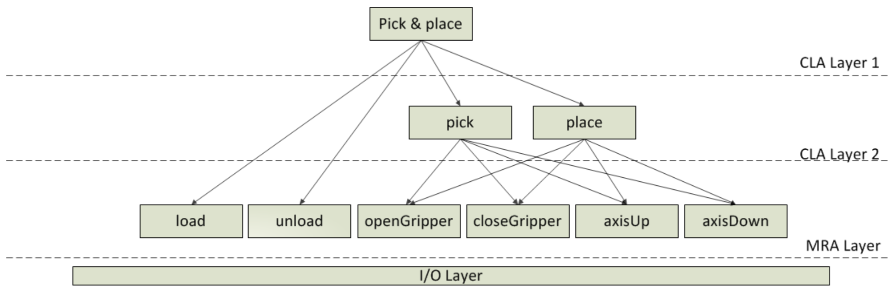


Figure 5.4: Festo Cell - Gripper 1 - Pick and Place

similar composition to the considered for cylinder 1. In this case, the *pick and place* also requires the skills that perform the movement into the desired position (A or B). These skills are built through a composition between the skills of the AxisY2 and BreakA(B) agents.

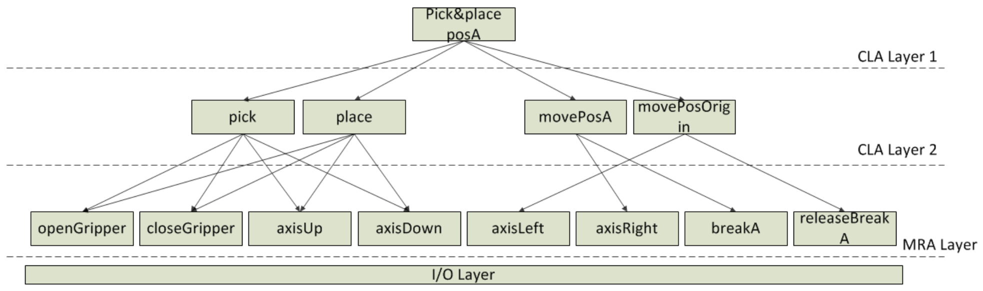


Figure 5.5: Festo Cell - Gripper 2 - Pick and Place

This system has some limitations, at first it does not have an active transport system between the two cylinders making it impossible to test the TSA. It also limits the redundancy tests and the plug and play of new physical resources that allow to test the self-organization promoted by the IADE stack. Despite the limitations, this system shows the correctness of the interactions between PAs, CLAs and MRAs.

The Figure 5.6 illustrates the controllers's scheme for this systems. The PAs and the CLAs are deployed on a home PC, and the MRAs are deployed on an ARM controller running Windows CE. The communication is through an ethernet network.

5.1.2 IDEAS Pre-Demonstrator

The IDEAS Pre-Demonstrator schematized in Figure 5.7 consists of a table with two Automated Guided Vehicles (AGVs) that ensure the transport of products to the several working stations. A *stacker* unit is responsible for placing the products onto each AGV or store them on the pallet repository. The table is also equipped with five slots. On each slot is possible to plug modules. In the tested system, three slots were occupied by a

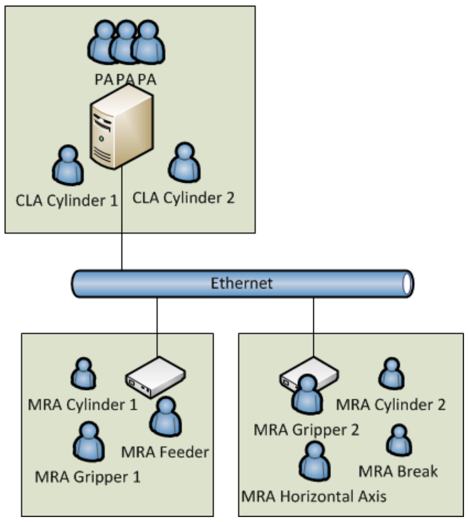


Figure 5.6: Festo Cell - Controllers

glueing unit, a *testing* unit and a *pick and place* unit.

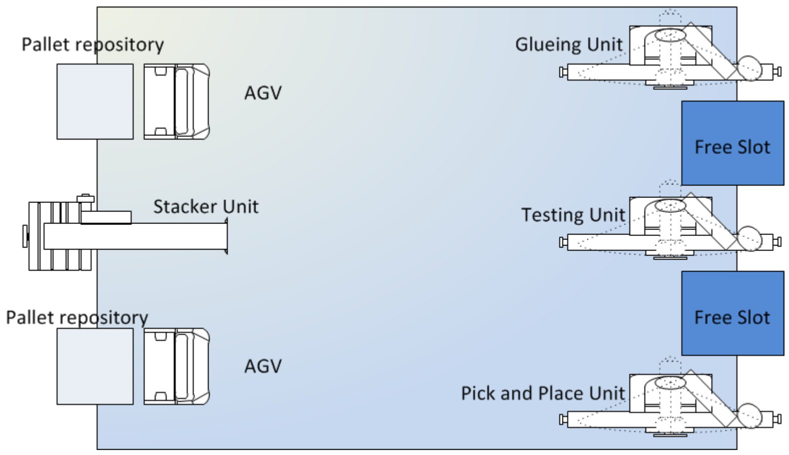


Figure 5.7: IADE Pre-Demonstrator Layout

Like the Festo Cell this system is divisible into several MRAs. From an IADE stack point of view, the only change is in the libraries that interface with the hardware. To ensure the lower level functionalities, the following MRAs were deployed on the system:

- **Dispenser**: controls the dispensing of glue, it provides only one skill, *dispense*.
- **Tester**: is the agent that controls the module that the provides the *test* skill that assesses the state of the product.
- **PickAndPlaceGripper, StackerGripper**: this agent controls the gripper of the pick and place and stacker units respectively. It provides the *openGripper* and the *closeGripper* skills.

- **GlueAxis, TesterAxis, PickAndPlaceAxis**: these agents controls the axis of the glue, test and pick and place units. They provide the *axisUp* and *axisDown* skills.
- **StackerAxis**: is the agent that controls the stacker axis. The main difference in repesct of the axis units is that this agent also allows motion along the horizontal axis. It provides three skills, *axisUp* and *axisDown* for the vertical motion, and *move* for the horizontal motion.

The *stacker* unit provides two top level skills (Figure 5.8), the *retrieve* and the *store* skills. The *retrieve* skill brings a new pallet into the system. The *store* skill, stores a finished pallet in the repository. Both skills are similar from a network point of view as they connect to the same MRAs, the difference lies on the parameter values passed by the higher level skill.

The Figure 5.9 illustrates the three possible horizontal positions of the stacker axis. In the position A (Figure 5.9(a)) the stacker is positioned to grab a pallet from the AGV, or put the pallet in the AGV. To retrieve a pallet from the repository the position B, illustrated in Figure 5.9(b) is used. Finally, to store a pallet into the repository, the stacker should be positioned at position C (Figure 5.9(c)).

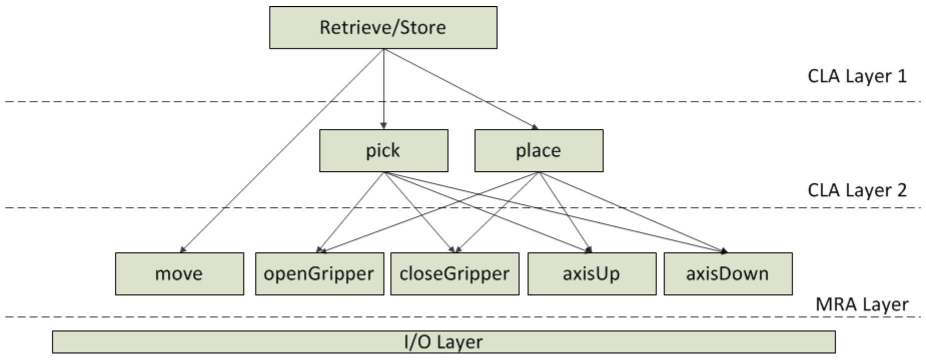
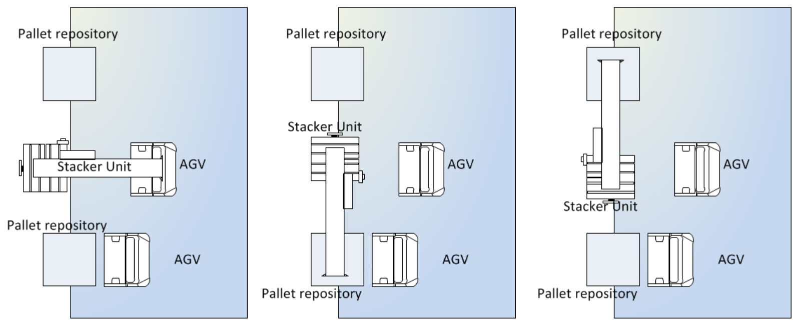


Figure 5.8: IADE Pre-Demonstrator - Stacker Skills



(a) Position 1 (b) Position 2 (c) Position 3

Figure 5.9: IADE Pre-Demonstrator - Stacker Axis horizontal positions

The *glue* unit and the *test* unit provide one high level skill each, the *glue* skill (Figure 5.10) and the *test* skill (Figure 5.11) respectively. They are both similar from a network point of view, comprising only two layers.

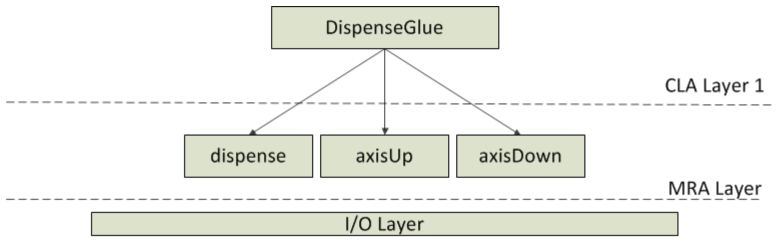


Figure 5.10: IADE Pre-Demonstrator - Glue Skill

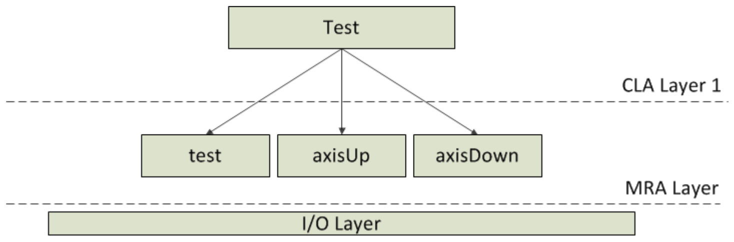


Figure 5.11: IADE Pre-Demonstrator - Test Skill

The *pick and place* skill (Figure 5.12) is provided by the *pick and place* unit. This skill is a three layer skill.

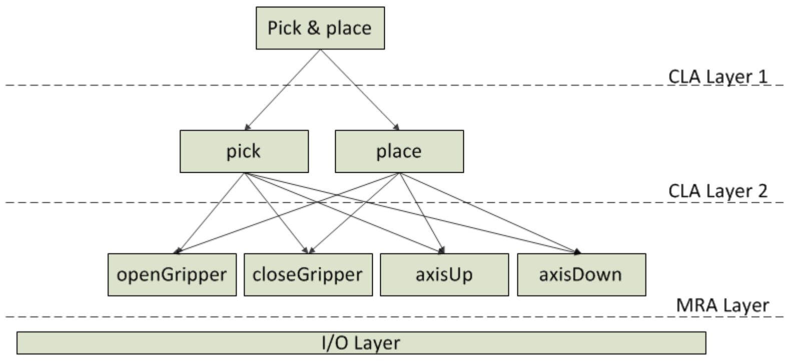


Figure 5.12: IADE Pre-Demonstrator - Pick and Place Skill

One of the biggest particularities of this system, is the possibility of plugging or unplugging stations on it. With this characteristic is possible to test the IADE stack's response to disturbances, in this case, a layout modification at run time. The IADE stack proved to be robust enough to handle this disturbance. In this system two tests to prove this functionality were performed:

- **First test:** The system started without the station (*Test Unit*), which will be plugged at run time. The PAs executed their process until they reached the missing skill and, after some time, the missing station was plugged.

- **Second test:** The system was booted with all stations plugged. During run time one station was unplugged and plugged in a different location.

In both cases the PA detected the absence of the required skill on the system, and performed some safe routines. In this case, the safe routines consists on move the PA to a position where it do not interfere with the other PA. This is important, because the PA will not give any order to the transport system until it has an agent owner for the next skill on its process plan.

The same controllers used on Festo Cell, where present on this system, and all the MRAs are deployed on them (Figure 5.13). The TEAs ran on a home PC with Windows XP, but the agents in this case do not actuate directly on the hardware. For the transport system, a software PLC is running on the PC, the agents have an interface where they can communicate with this PLC. The PAs and the CLAs as it is in Festo Cell are deployed on a home PC running Windows XP. The communication between agents is made through an ethernet network.

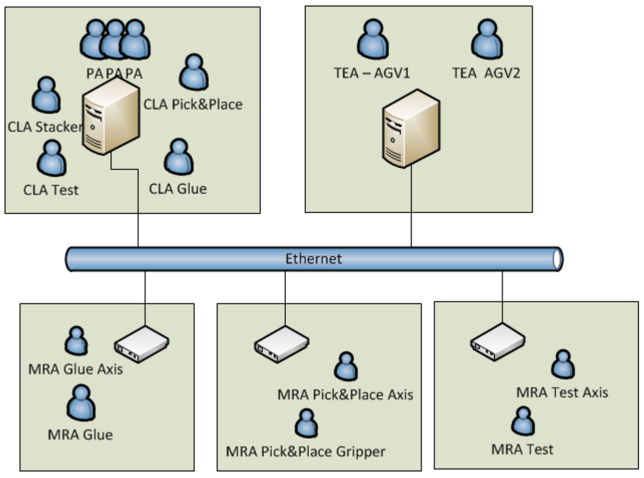


Figure 5.13: IADE Pre-Demonstrator - Controllers

5.1.3 Masmec Demonstrator

The most important demonstrator in the IDEAS project is illustrated in Figure 5.14. As opposed to the IDEAS Pre-Demonstrator, the transport system in this demonstrator is composed by a conveyor system with no limitations regarding the number of products on the line. An Radio-Frequency Identification (RFID) system is also present in order to support the actions performed by the transport system. This characteristic allowed a wider study about the capabilities of the IADE stack.

The initial setup of this system comprises the following stations: *Manual Load Station*, two *Testing Stations*, *Labelling Station*, *Manual Unload Station* and *Automatic Unload Station*.

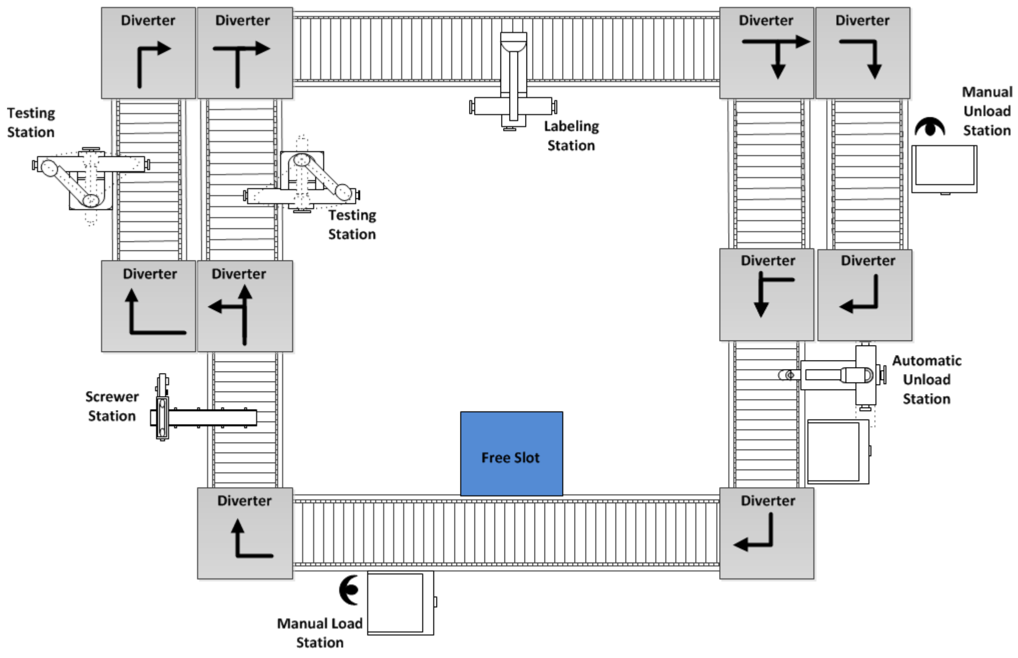


Figure 5.14: Masmec Demonstrator Layout

The *Manual Load Station* is in charge of introducing new pallets into the system. This station is operated manually. At the time the operator introduces a new pallet, it should inform the system about this new product. This functionality is available to the operator on a tool specifically designed to support the operation of loading products. Although this characteristic was not predicted in the original design of the IADE stack, the possibility to introduce it with a very little effort proves how easy is to re-configure a system based on the IADE stack.

This system offers two alternatives to unload a pallet. The *Manual Unload Station*, is similar to the *Manual Load Station*. An operator is present to take off the products from the system. The second alternative is the *Automatic Unload Station*, which is a mechanical gripper with an axis that allows vertical and horizontal motion. In this system the PA should always choose the *Automatic Unload Station* rather than the *Manual Unload Station*, if the conditions are similar on both stations.

The *Screwer Station* is a composition of a pick and place unit and a screwer unit, it is on this station thtat the assembly operations are performed. The *Testing Station* is in charge of testing if the product was correctly assembled. In case of a manufacturing defect that result is written on the RFID tag of that product. Depending if the product is correctly assembled or not, the *Labelling Station* will stamp the product with an **Ok** or **NotOk** label.

In comparison with the Festo Cell and the IADE Pre-Demonstrator, this demonstrator is composed by a wider diversity of skills compositions. The *Manual Unload Skill* composition is illustrated in Figure 5.15. In present case, both skills, at MRA level, are supplied by the same agent. The *WriteProductInformation* skill registers the pallet on the RFID system. The *WaitStartingProduct* skill works as a signal for the PA to know if it can proceed with its process plan or not and is dependent of the operator. When the operator finishes loading the pallet he should press a button. The MRA intercepts the signal and returns it to the PA agent so that it knows that the execution was successful.

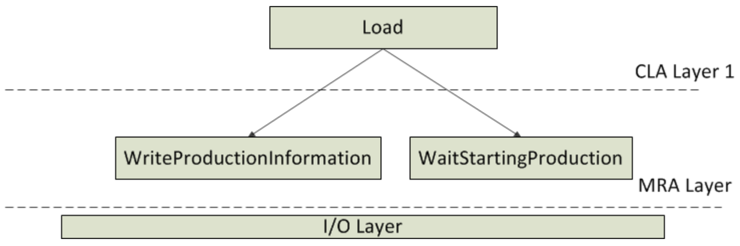


Figure 5.15: Masmec Demonstrator - Manual Load Skill

The *Screwer Station*, as it previously referred is a composition of two units. The pick and place unit, is a composition of a robotic arm and a gripper, and they are controlled by two different MRAs. The top level skill of this station is the *Screw* skill (Figure 5.16). The *assemblyScrew* is an ASk provided by the screwer unit, this unit is only abstracted by one MRA. The robotic arm provides the *moveTo* and the gripper provides the *openGripper* and *closeGripper* skills. The composition of these three skills (*moveTo*, *openGripper*, *closeGripper*) constitute the *pick and place* skill.

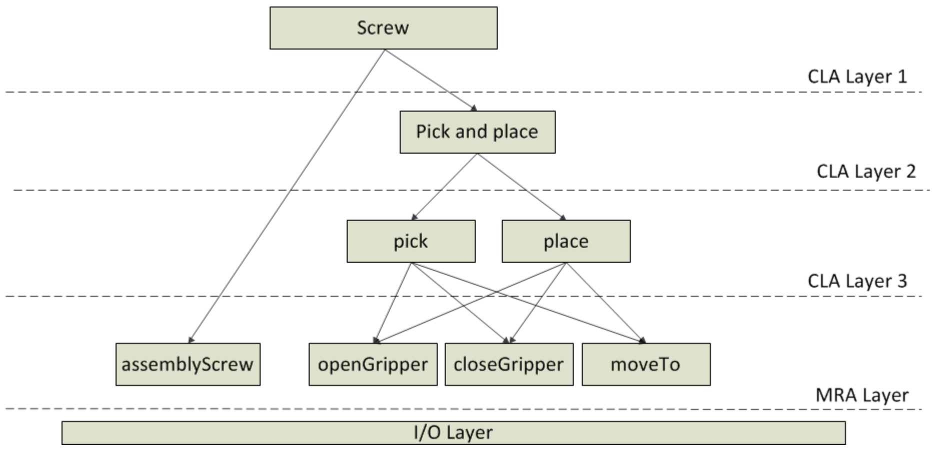


Figure 5.16: Masmec Demonstrator - Screw Skill

The *Testing Station* is constituted by a vertical cylinder and a test unit. The *Test* skill (Figure 5.17) is fairly simple from a network point of view. The *cylinderUp* and the *cylinderDown* skills are provided by the same MRA agent, while the *test* and the *writeTestResult* skills are provide by another MRA. The *writeTestResult* writes in the RFID tag the test result.

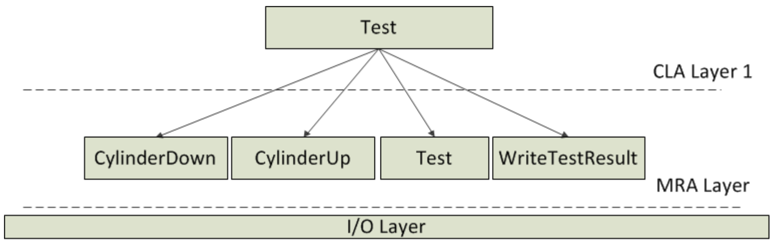


Figure 5.17: Masmec Demonstrator - Test Skill

The *Labelling Unit* is constituted by a vertical cylinder, a horizontal axis, and a vacuum gripper. The skill supplied by this unit is illustrated in Figure 5.18. From a network point of view it is the most complex skill in this system. This station comprises the following skills at MRA level: *readProductInfo*, *moveH*, *cylinderUp*, *cylinderDown*, *suckerOn* and *suckerOff*.

The skill *GetTag* is particularly interesting because its process plan includes a decision skill to support the selection of the label will be grabbed. The layout on Figure 5.20 illustrates the Labelling unit. Since there are two possible positions to grab a label, the parameter on the *moveH* skill will be dependent on the result of the decision skill. The Figure 5.19 depicts a possible process plan for this skill. The first step is to retrieve the test information from the RFID system, which is possible to be obtained through the *readProductInfo* skill, then the test result is evaluated on the decision skill. If the test result is **Ok** the *moveH* is invoked with a parameter *Py=Ok*, which corresponds to the position for the **Ok** label. In case of a **NotOk** result, the *moveH* is invoked with a parameter *Py=NotOk*.

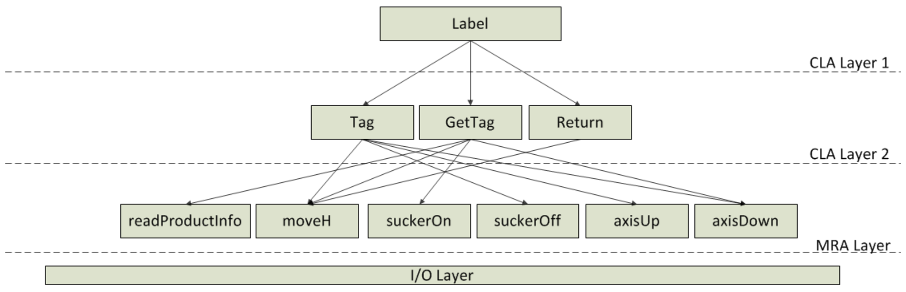


Figure 5.18: Masmec Demonstrator - Label Skill

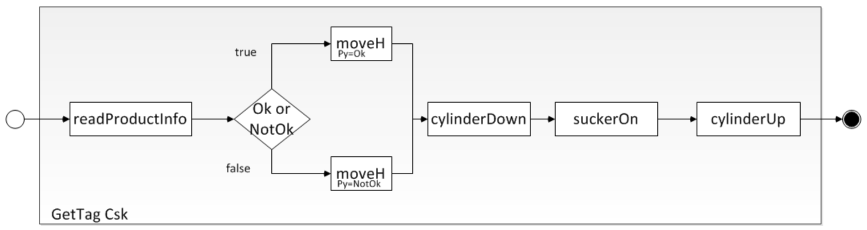


Figure 5.19: Masmec Demonstrator - Label Skill

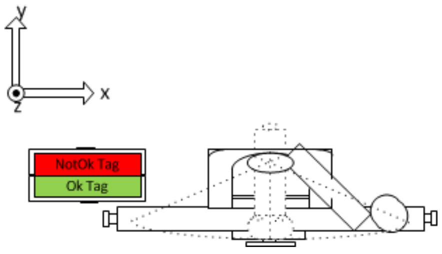


Figure 5.20: Masmec Demonstrator - Labelling Station - Label Unit

The *Manual Unload Station* and the *Automatic Unload Station* offer the same skill, however, they have different compositions. In Figure 5.21 is depicted the network structure of the *Unload* skill provided by the *Manual Unload Station*. Figure 5.22 illustrates the *Unload* skill of the *Automatic Unload Station*.

The *Unload* skill offered by the *Manual Unload Station* is fairly simple. The *readProductInfo* skill reads the information about the test performed at the *Testing Station* and the *DisplayProductInformation* skill displays the result to the operator. The operator, depending if the product has a manufacturing defect or not, will store the pallet in different repositories. Finally, the *WaitPalletUnload* skill is dependent on a signal given by the operator which should occurs as soon as the operator removes the pallet.

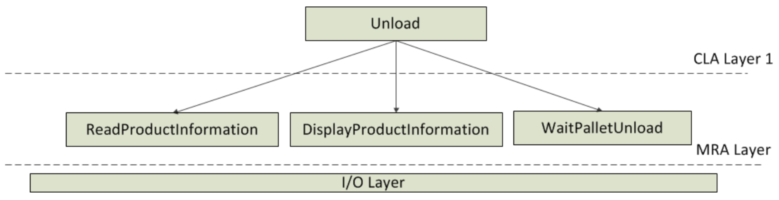


Figure 5.21: Masmec Demonstrator - Manual Unload Skill

The *Unload* skill, provided by the *Automatic Unload Station* (Figure 5.22), is composed by a *pick* skill and a *place* skill. Similarly to the *GetTag* skill of the *Labelling Station*, the process of this skill has a decision skill since, depending on the result of the test at *Testing Station*, the destination repository changes, splitting the pallets with defect from the good ones. The *Unload* skill shows that it is possible to have the same top level skill provided by different stations with different internal compositions.

The **pluggability** and **reconfigurability** were tested on this system. One of test cases included the unplug of the *Labelling Station* and its re-plug it on the free slot available. Another concept, the **granularity** was also tested with success on this demonstrator, stations like the *Manual Unload Station* and the *Automatic Unload Station* have different internal processes with different granularity levels, although, from a PA agent point of view, the provided skill is exactly the same.

The controllers schematic of this system is illustrated on Figure 5.23. Similarly to the

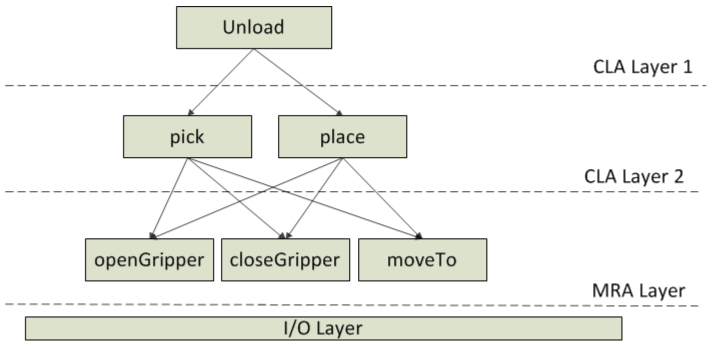


Figure 5.22: Masmec Demonstrator - Automatic Unload Skill

Festo Cell and the IADE Pre-Demonstrator, the PAs and CLAs are deployed on a home PC running Windows XP. The main difference, on the controllers of this system is the introduction of a new controller for some MRAs and the transport system. All the agents of the transport system run on a controller with an embedded version of Linux. Apart from the transport system agents, the agents from the *Screwer Station*, *Testing Station* and *Automatic Unload Station* are deployed on these controllers. The remaining MRAs are deployed on the same controllers used on the other demonstrators, which are running Windows CE. Regarding the communication channel, similarly to the systems previously described, is made through an ethernet network.

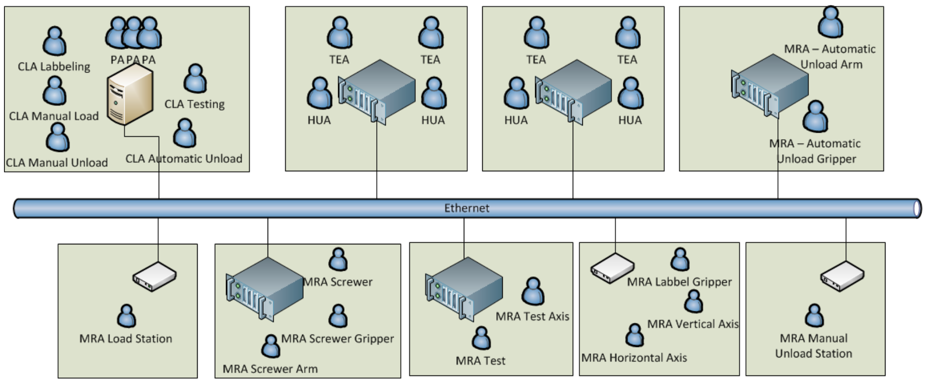


Figure 5.23: Masmec Demonstrator - Controllers

5.2 Simulations

From a simulation point of view, two tests were considered. In the first test, the impact of concurrency on the performance of the system was assessed. In this context, several compositions of skills where simulated for a virtual system and executed. The performance of the system was evaluated and analysed.

In the second test the proposed metric was put to the test. The purpose is to evaluate emergence and the self-organizing response of the system. It has also the purpose of showing that self-organizing systems are not necessarily unpredictable but their behaviour must be clarified and understood.

5.2.1 Skill Composition Tests

The central focus of these tests is to analyse the time required to execute processes on the top level entities (products) in concurrent situations.

5.2.1.1 Theoretical Performance Limits

The network structure of the decomposition of a top level skill into several layers enables the estimation of the execution time of an individual skill as well as for an entire process.

For a composed skill decomposed in n layers with an average number of branches per node per layer (bl_n) and a number of nodes per layer (N_n), the number of skills executed is given by the following equation:

$$SkillExec_{untilN} = 1 + \sum_{n=0}^{i} N_n * bl_n \tag{5.1}$$

Disregarding the communication delay, the total time to execute one skill would be proportional to the number of skills activated ($N_{Skill_{Resources}}$) (Equation 5.2) which is given by the difference between the number of skills executed in the layer n and layer $n-1$. The average execution time of skills ($ExectTime_r$) is give in Equation 5.3.

$$N_{Skill_{Resources}} = SkillExec_{untilN} - SkillExec_{until(N-1)} \tag{5.2}$$

$$ExectTime = N_{Skill_{Resources}} * ExecTime_r \tag{5.3}$$

However, there is some processing time between the higher level skill and the skill at the resource level that cannot be despised. In the CLA level, actions like marshaling , un-marshaling and orchestration have to be computed and should be taken into consideration. The estimation for the global execution time is detailed in Equation 5.4.

In Equation 5.4, the interactions (I) is the number of the connections between the skills. The multiplier stands for the fact that it is a bidirectional link. The travel time between agents (TTB) is the estimated time since the message is sent until it reaches the destination (this time includes the marshaling and un-marshaling operations), and finally the processing execution time on MRAs ($ExecTime_R$) and CLAs ($ExecTime_{cla}$) is multiplied by the number of the corresponding agents.

$$ExecTime = 2 * I * TTB + N_{SkillResources} * ExecTime_R + N_{clas} * ExecTime_{cla} \tag{5.4}$$

5.2.1.2 Testing Conditions

For testing the system was ran 103 times. In each run a new network has been generated, which leads to different topologies regarding the connections between the agents. The variables that characterize the network are however maintained. The tests run under a network composed by five layers. The first layer belongs to PAs, the second, third and fourth layers are for CLAs and the bottom layer is composed by MRAs.

The first layer is populated with five agents where each one will connect to five CLAs. The second layer, has five CLAs and each one connects to two CLAs on the layer below, which is populated with ten CLAs. Each of the ten CLAs of the third layer will individually connect to two of the ten CLAs that compose the fourth layer. The CLAs on fourth layers connects to four agents on MRA layer.

In these trials, the negotiation is not allowed, so the initial structure of the network does not suffer any changes during the execution. The PAs start the execution of their process plan at the same time to promote competition. When the agents (PAs and CLAs) receive a refuse message, which happens when the requested agent is busy, the agents enter an idle state for two seconds before trying the execution. The MRAs simulate the hardware and each MRA take approximately 500 milliseconds to execute a skill.

In each trial, the agents run on the same computer, and there are no significant delays on communication due to network performance. In these circumstances, the TTB is negligible since it is less than 1 millisecond.

For each skill requested by a PA a total of 23 skills are executed (16 MRA skills and 7 CLA's), which leads to the following optimal (no concurrency) execution time:

$$ExecTime = 0 + 16 * 500ms + 7 * 2ms = 8014ms \tag{5.5}$$

5.2.1.3 Results

The Table 5.1 contains the data produced by the execution of all the PAs. Since the system runs under a simulated environment, it is not expected the appearance of failure messages. The number of agree message is exactly the same as the number of skills that are on the processes. The number of received refuses is a consequence of the competition in the system. The delay introduced on the agents between a refuse message, and a new request by a PA, has a high influence on this number. A high number of refuses could also be an indicator for a bad dimensioned system regarding the number of PAs that are competing for the skills.

The Tables 5.2 and 5.3 detail the data recorded by the CLAs. The layers 3 and 4 correspond to agents with one input connection (in degree 1) and the first layer stands for agents with five input connections (in degree 5).

Table 5.1: Messages Processed at PA level

	Number PA	Total Messages	Messages/Agent	Standard deviation
Requests Sent	515	9418	18.28	8.79
Refuses received	515	6843	13.28	8.79
Agrees Received	515	2575	5	0

Table 5.2: Requests Received by CLAs

In Degree	Number CLA	Total Messages	Messages/Agent	Standard deviation
all	3605	24868	6.9	6.53
1	3090	15450	5	0
5	515	9418	18.29	12.13

As it is expected the results on Tables 5.2 and 5.3 are a consequence of the conditions at the PA layer. As it is possible to observe, on Table 5.2 the amount of requests received by the CLAs on layer 3 and 4 is precisely 5, which matches the number of skills present on PA's processes, and it is due the fact of the degree (in degree = 1) of these agents. The agents on the first layer are directly connected PAs so the amount of messages received by each exceeds the number of skills because they handle competition on this layer.

Table 5.3: Refuses Sent by CLAs

In Degree	Number CLA	Total Messages	Messages/Agent	Standard deviation
all	3605	6843	1.9	6.53
1	0	0	0	0
5	515	6843	13.29	12.13

On Table 5.3, it is possible to observe that the competition is almost filtered by the agents on first layer and does not happen between the layers 1 and 2, and between the layers 2 and 3.

The Table 5.4 shows that the concurrency is also present at the interactions between CLAs and MRAs.

The out degree 4 on Table 5.4 corresponds to CLAs on layer 4, and they are directly connected to MRAs. On this level the concurrency between MRAs leads to CLAs receiving refuses, but since the average degree (the in average degree is 4) is less than in the second layer and the competition is almost filtered between the first and second layers, the number of refuses per agent is low.

The Table 5.5 confirms that the MRAs with a higher degree are the most solicited agents at this layer.

This results give a clear idea on how the concurrency is handled by skill networks that follow a semi-hierarchical structure. It was observed that the competition almost only happens on the first and last layers. It is necessary to note that is possible to change

Table 5.4: Refuses Received by CLAs

Out Degree	Number CLA	Total Messages	Messages/Agent	Standard deviation
all	3605	1101	0.31	0.65
2	1545	0	0	0
4	2060	1101	0.53	0.79

Table 5.5: Execution Requests Received by MRAs

In Degree	Number MRA	Total Messages	Messages/Agent	Standard deviation
all	4120	42301	10.26	4.40
1	1175	5875	5	0
2	1943	19802	10.19	0.44
3	848	13223	15.59	0.81
4	136	2902	21.33	1.35
5	17	467	27.47	1.81
6	1	32	32	0

the network structure in order to affect where the competition is handled, since this is related with the in degree of the agents.

Table 5.6: Execution Times for CLAs

	CLA1	CLA2	CLA3	CLA4	CLA5
Average	17880 ms	18640 ms	18420 ms	17620 ms	17820 ms
Standard deviation	11110 ms	11460 ms	10880 ms	9580 ms	10210 ms
Worst	83925 ms	81979 ms	81791 ms	61012 ms	66690 ms
Best	8062 ms	8092 ms	8157 ms	8118 ms	8064 ms

The executing times of CLAs on the second layer are detailed on Table 5.6. These times are measured by the PAs, and correspond to the time that the PAs need to wait for the execution of the requested CLA. It is possible to observe that at least one of the CLAs was able to perform their execution on a time very close to the optimal theoretical limit described in the previous sub-section.

The worst case reached approximately 84 seconds. Considering a scenario where the competition only occurs at this first layer, the worst case for a PA is when the agent is the last of the PAs present on the system to pick the CLA. In this case it has to wait for the others to perform their execution. For this hypothetical situation, the theoretical time to execute that skill is $8014 * n$,where n is the number of PAs including itself. In this particular case $n = 5$, and the worst theoretical case time is $40070ms$. However, the worst measured case took more $43855ms$ than the theoretical, which leads to the conclusion that the competition on MRA layer had an important contribution for the execution delay.

5.2.2 Assessing Emergence and Self-Organizing tests

For the second test the purpose is to identify two emergent effects, the first is the distribution of the agents workload through the system, and the second emergent effect is an increased performance on the system.

Those two effects come as a consequence of the metric defined in section 3.2. The purpose of the metric is to ensure that the PAs choose the best option when there are multiple options available. It is important to note that the PAs do not have an entire vision regarding the system, and their view is based on the negotiation process with the CLAs.

5.2.2.1 Testing Conditions

The Figure 5.24 illustrates the system (S) designed for this test. This system comprises two generic areas and three different tasks, namely "A", "B" and "C". The system also contains a conveyor system that is in charge of transporting the pallets. The transport system also allows a product to choose between leaving the system or returning to a station.

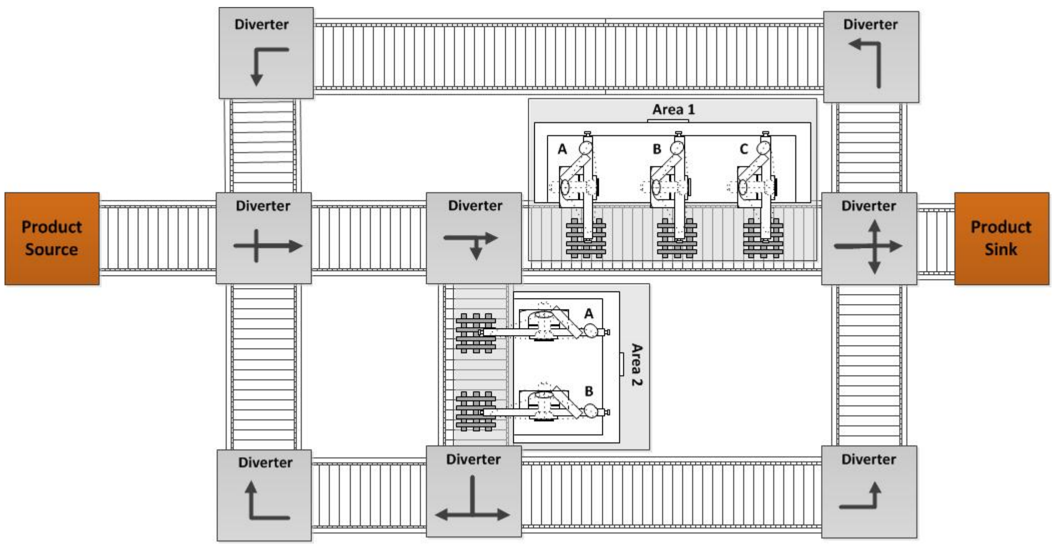


Figure 5.24: Simulated System

The areas are not equal. The *Area 1* provides the three tasks, and the *Atation 2* only provides "A" and "B". In this simulation, there are three different products, P_1, P_2 and P_3, each one with different processes. P_1 executes "A","B" and "C". P_2 executes "A" and "B". Finally P_3 only executes "A". For this simulation the amount of products in the system is fixed and its value is 20 products of each type. There is a total of 10 MRAs in S and 5 CLAs (each one managing 2 MRAs).

In order to measure the emergent effects, the following tests were performed:

- Negotiation with increased processing capacity (T_1) - the products have the freedom to choose the stations where to perform the tasks and each area has a capacity for three simultaneous products.
- Negotiation with decreased processing capacity (T_2) - the products have the freedom to choose the stations where to perform the tasks and each area has a capacity to handle only one product.
- No Negotiation and increased process capacity (T_3) - the products are pre allocated to the stations and should execute their skills there. The negotiation is disabled. In this case each area has a capacity for three simultaneous products.
- No Negotiation and decreased process capacity (T_4) - the products are pre allocated to the stations and should execute their skills there. The negotiation is disabled. In this case each area handles only one product.

The tests T_3 and T_4 have the purpose of setting the base values for the assessment of the self-organization metric. On these simulations each test was performed 100 times.

5.2.2.2 T_1 Results

The results of the work balance emergent are depicted in Figure 5.25. This figure clearly details that the system takes some time to stabilize the value of the metric which denotes the convergence of S to a steady state. In this first moment the behaviour of S is clear. Since all the PAs, regardless of their type, start by executing the skill A there is an increase in the demand of B and C as the value of A drops after being executed. This effect is more evident on *Area 1* since the largest skill set that can be allocated to this area is $\{A, B, C\}$ which is directly implemented by the CLAs therein. As *Area 1* becomes more heavily solicited the PAs requiring $\{A, B, C\}$ will tend to choose *Area 2* and postpone the execution of C. S then reaches a steady state with the values of the metric for each CLA tending to increase.

This shows two distinct self-organization periods. The first one, until $Time = 1 \times 10^5$, that shows the start of the system and its effort to improve its organization in respect to the location where PAs are processed and the second one, from $Time = 1 \times 10^5$ to $Time = 2.7 \times 10^5$, that shows only minor adjustments as the systems stabilizes into an organizational state that meets its design purposes.

It is noteworthy the fact that the values of the metric for the second station are slightly higher. This is a clear indication that the station is being more heavily requested probably due to the fact that it implements a reduced number of skills and therefore is faster in processing PAs. The only exception to this trend is the CLA implementing C in *Area 1*. This can be explained with the start up of the system where P_1s have opted for *Area 2* and need to loop around the system before executing C in *Area 1*. This behaviour clearly

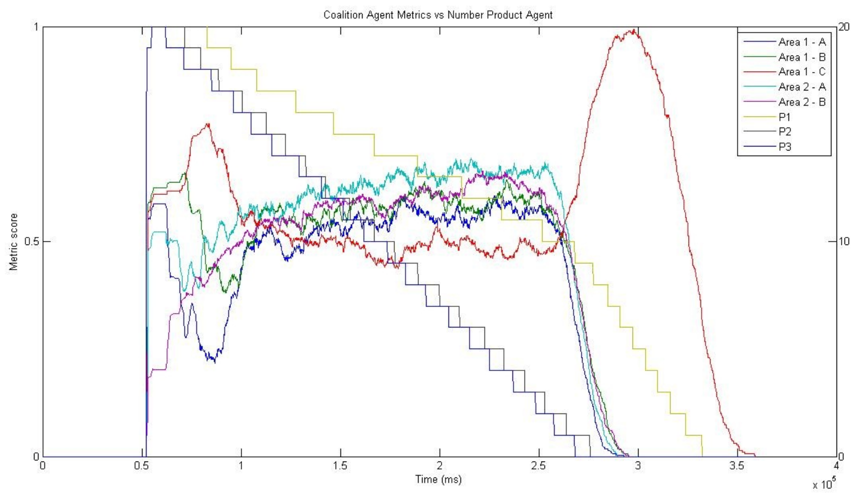


Figure 5.25: Average Metric Evolution and Product Number Evolution for T_1

justifies the final section of the graphic that depicts an increase in demand for C in *Area 1*. This observation is consistent with the number of PAs per type (Figure 5.25) as it can be verified that PAs of type P_2 and P_1 leave the system earlier causing the cost for the execution of A and B to drop as C rises.

This third organizational period, after $Time = 2.7 \times 10^5$, shows the system struggling has the PAs systemically require the same skill. The system is processing at its maximum speed to meet all the product requests in station one as station two cannot offer C.

The analysis of the Table 5.7 complements the behaviour observed in the Figure 5.25. In fact, Table 5.7 shows that the self-organization metric was successful in distributing the load between both areas. *Area 2* has processed roughly 9 P_1's. This means that in balancing the system at a specific time it became convenient to re-route P_1s (this was attained by improving the organization of the system). This behaviour is also an indication that the negotiation strategy that allocates the largest subset of skills performed by a station tends to privilege areas, with acceptable performance, that implement that subset since most of the P1s have been processed by *Area 1*. This can be clarified by evaluating the contribution of P_1s, P_2s and P_3s to the overall metric of the CLAs implementing the skill A in areas 1 and 2 (Figures 5.26 and 5.27 respectively). Both figures clearly depict a preference of P1 for *Area 1* in the initial phase where P2 and P3 were mainly processed at *Area 2*.

These results show how the proposed metric can influence the organization of the system without any centralized point of control and using mainly negotiation. One can speculate out of these results that the runtime introduction of a new CLA which would bring redundancy could lead the system yet to a new organizational level. In the specific case of this system the simulation shows that it would benefit from the addition of another

Table 5.7: Average PAs requests processed by CLAs (T_1)

	Station 1						Station 2			
PA/CLA	A		B		C		A		B	
	avg	stdev	avg	sdev	avg	sdev	avg	sdev	avg	sdev
P1	10.69	1.78	10.69	1.78	20	0	9.31	1.78	9.31	1.78
P2	8.94	1.78	8.94	1.78	0	0	11.06	1.78	11.06	1.78
P3	9.13	2.36	0	0	0	0	10.87	2.36	0	0

CLA supporting the execution of C to decrease the demand peak after $Time = 2.7 \times 10^5$.

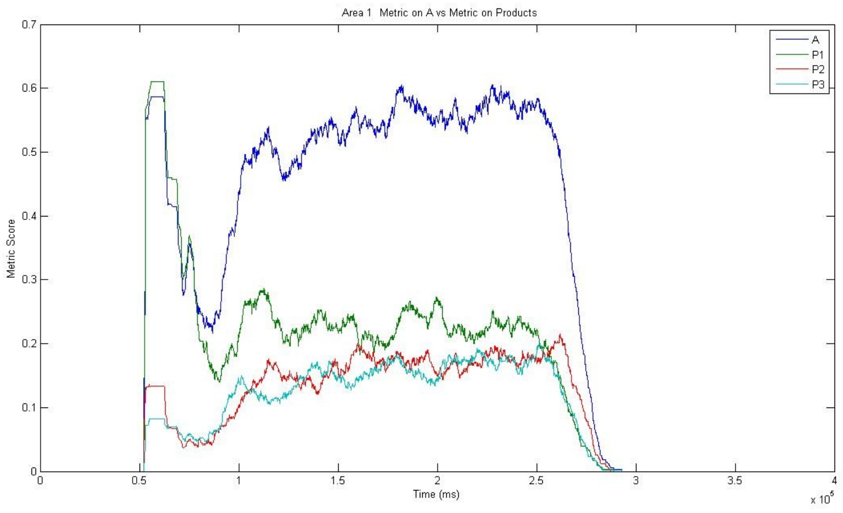


Figure 5.26: Contribution of PA types for the CLA implementing A in Area 1 (T_1)

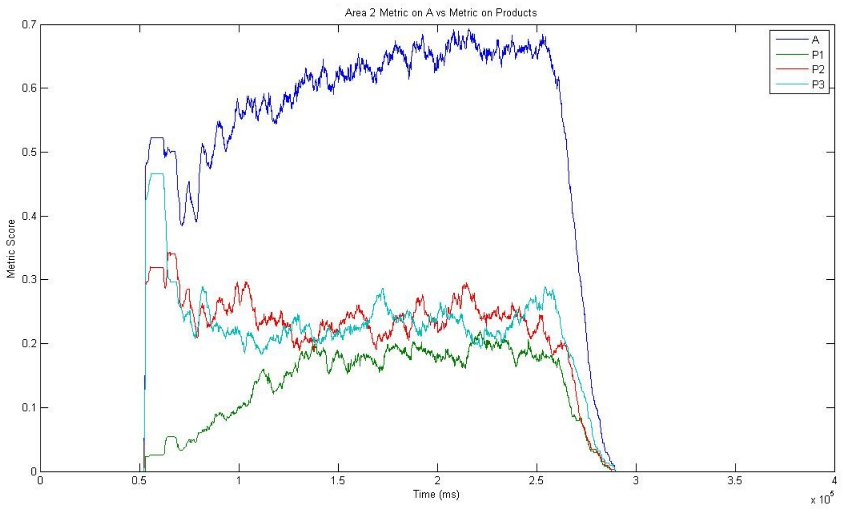


Figure 5.27: Contribution of PA types for the CLA implementing A in Area 2 (T_1)

Another important result that uncovers the inner dynamics driving the system is the evolution of the average of the metric computed by each CLA and seen by the PAs when negotiation takes place (Figures 5.28 and 5.29). Each point on the graphic denotes de average value on time. This reinforces the importance of simulation in studying emergent phenomena since in a system with a deterministic dynamic the metric would evolve in steps and would remain constant for longer periods of time. However, Figure 5.28 shows that the value seen by the PA and used in negotiation is always lower than the value computed by the CLA. This difference can be explained since upon negotiation the PA is positively contributing to the cost increase in the metric computed by the CLA. In addition the metrics seen from the PA retain the latest value considered for negotiation until the PA leaves the system. The metrics computed from the CLA point of view are much more dynamic in this respect as they react to the number of PA's on system and their types as explained in section 3.2.2.

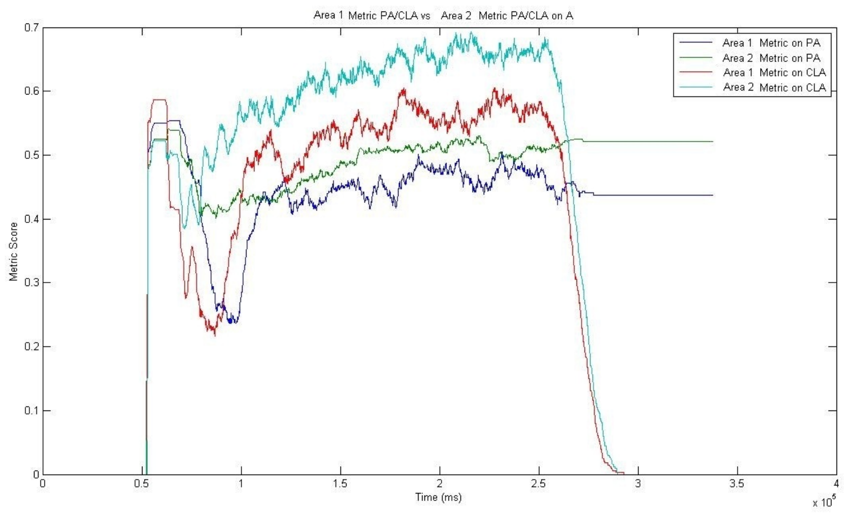


Figure 5.28: Comparison of the metric values computed by the CLAs and seen by PAs over time in both areas for A (T_1)

5.2.2.3 T_2 Results

The second test restricts the capacity of each station to one PA. Although the micro-dynamic of the system produces similar results (illustrated on appendix 3) when compared to the results already detailed it is worth analysing the impact of this change in the variation of the metric Figure 5.30. It is important to mention that the main difference induced by this change is in the variation of the metric. In this context, since only one execution at a time is allowed, given that a finished execution affects the value of P_{Pi} decreasing it and successful negotiations increase the value of $P(CLAx \mid P_{Pi})$ the evolution of the metric smooths with the increase in station capacity.

It is also interesting to compare the number of average requests. The $T2$ conditions

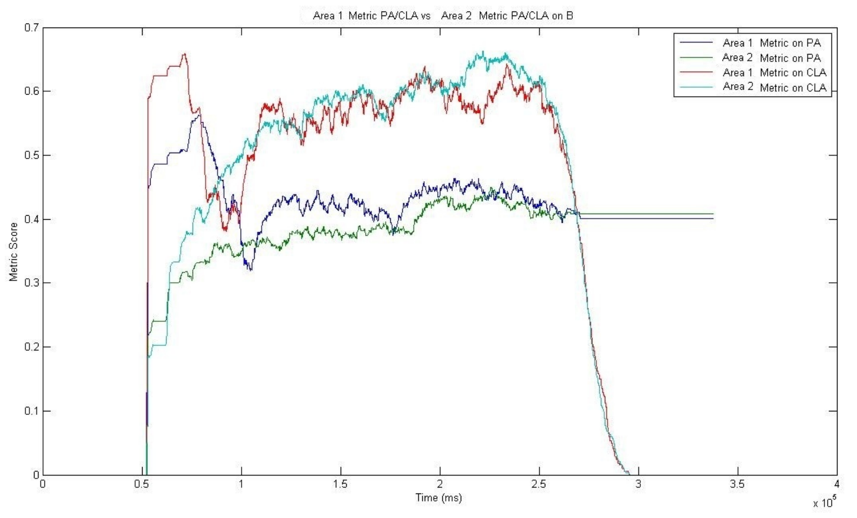


Figure 5.29: Comparison of the metric values computed by the CLAs and seen by PAs over time in both areas for B (T_1)

have not significantly impacted the overall behaviour of the system (Table 5.8) when compared to $T1$ (Table 5.7). The main difference lies in the number of P_2 requests processed by each station. In comparison to $T1$ more P_2 requests have been processed in *Area 2* than in the $T2$. However, as previously noted for the variation of the metric, there is also an increase in the standard deviation of P_2 which confirms the less stable behaviour of S as the capacity per station decreases.

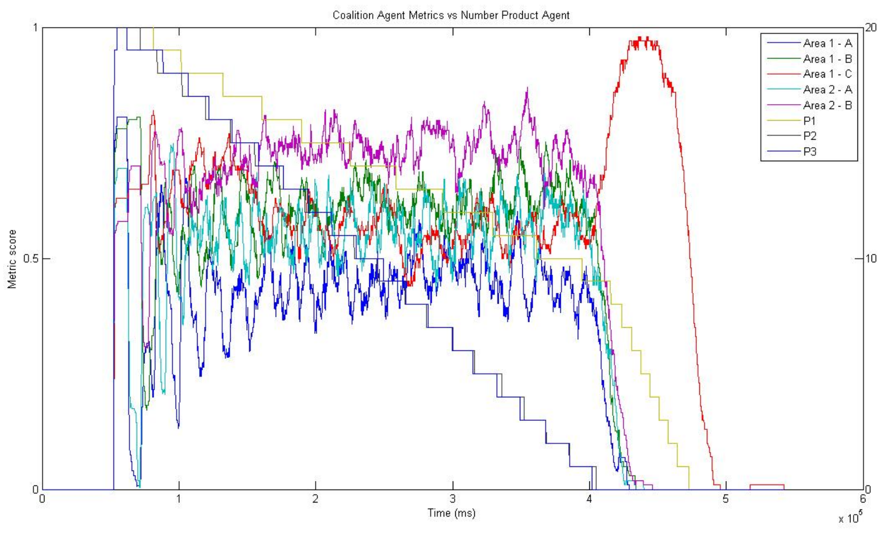


Figure 5.30: Average Metric Evolution and Product Number Evolution for T_2

Table 5.8: Average PAs requests processed by CLAs (T_2)

	Station 1						Station 2			
P/CLA	A		B		C		A		B	
	avg	stdev	avg	sdev	avg	sdev	avg	sdev	avg	sdev
P1	10.86	1.98	10.86	1.98	20	0	9.14	1.98	9.14	1.98
P2	7.17	2.09	7.17	2.09	0	0	12.83	2.09	12.83	2.09
P3	8.09	2.29	0	0	0	0	11.91	2.29	0	0

5.2.2.4 T_3 and T_4 Results

As control tests the same systems under similar conditions were ran without negotiation. In this context all the PAs instantiated as P_1 will be solely processed by *Area 1*, the P_2 and P_3 will be handled in *Area 2*. This scenario corresponds to a system with fixed job allocation.

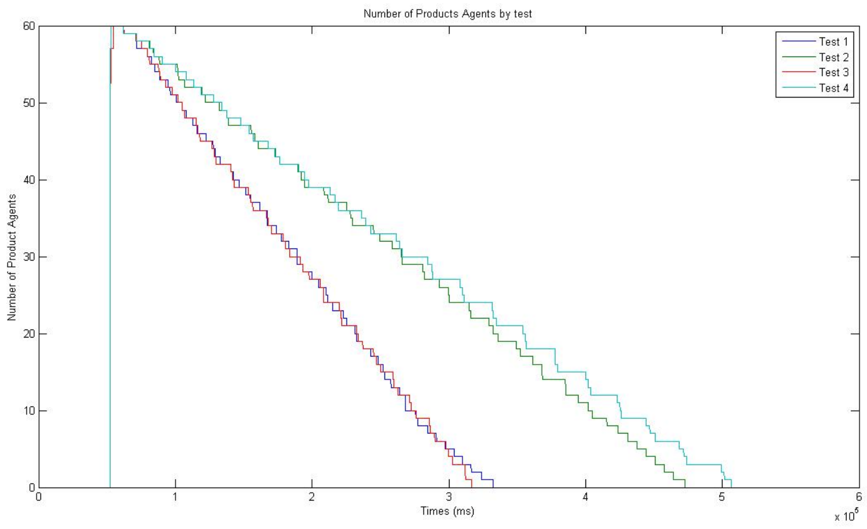


Figure 5.31: Comparasion of the Speed of the System with negotiation and without negotiation

The analysis will focus on the control tests with increased capacity since $T4$ produced similar results. As can be seen by Figure 5.31 negotiation stimulates the system's performance and yet maintains the organization of the system. It is interesting to observe that for the worst testing conditions (tests 2 and 4), where each area can only process one product at a time, the impact of negotiation is more noticeable than in the tests where each area can process more products ($T1$ and $T3$). However, even if the gain in time is not substantially in $T1$ and $T3$ the workload distribution is confirmed, Table 5.7 shows that the requests to each area are much more even in test 1 than in test 3.More importantly these emergents have been attained by the system through continuous adjustments to its organizational state.

6 Conclusion and Future Work

6.1 Conclusions

This thesis meets several purposes. Firstly it attempts to investigate the execution performance of a production system considering the impact of the network structure. The obtained results intend to point out that a proper design of the system can be crucial to achieve a considerable level of performance. The results also illustrate that the competition in self-organized systems has a significant contribution to the performance.

In this context, the performance of the network of agents was discussed in respect to execution times and compared with the maximum theoretical values. In the calculation of the theoretical values, the message transport cost has been neglected since the simulation ran on a single machine. The execution time at MRA level has been arbitrarily chosen and the processing time at the CLA has been measured and includes all the orchestration and message processing operations per skill execution. The results are necessarily influenced by the performance of the pc, the implementation details and the quality of the agent's code. However the important aspect to retain is that they reflect consistently the operation in a network of agents that constitutes a system.

The results also attempted to provide some quantitative meaning to the concept of Emergence in the context of a mechatronic system. The analysis is restricted to a special case of Emergence denoted as Weak Emergence which by definition assumes that the whole can be derived by simulation from the causal interactions between its parts and that, as in the Strong Emergence concept the whole exerts influence on the parts. The results detail all these effects. In particular, it was shown that a coherent metric can be used to promote different stages of organization that contribute to the emergence of some measurable aspects of the whole.

In the present case, it has been shown that a metric capturing the degree of usage of the components can improve the system performance and its workload distribution if the main decision-makers (the product agents in the present case) attempt to minimize the value of the metric when making a decision. In this context all the results reflect the bottom up approach of a system based on an architecture designed to promote self-organization and emergence. The system is driven by a set of autonomous components and its micro-dynamic is responsible along with the external conditions for the inner self-organizing processes that result in the emergent qualities under study.

The work hereby detailed also reinforces the importance of simulation in assessing new control systems and methodologies. As modern control approaches and production paradigms increasingly evolve in the direction of system design for emergence it is fundamental to understand and model the principles that may guide these systems. It is worth recalling that the advantages frequently attributed to modern production approaches, namely plug-ability, reconfigurability, robustness, sustainability; are only true if the underlying system, its micro-dynamic and context are understood and tamed. Unlike most computational systems, that were the origin of resource distribution, mechatronic systems have specific constraints that influence its behaviour.

This work stands therefore as a first step toward the development of a methodology to assess systems that are designed for emergence. Given that most of the technological barriers that haunted modern production approaches for years have been removed by recent advances in IT it is the author belief that the analysis and modelling of the inner dynamics of these system is the necessary next step if they are to become a reality other than elusive prototypes.

6.2 Future Work and Scientific Contributions

One of the main challenges to be tackled as future work is the assessment of the system dynamics when the effects of the transport system are not neglected as well as the study of the system behaviour when is affected by faults and layout re-designs.

As the technology that can support such systems is starting to consolidate it becomes increasingly important to explore their organizational aspects. Approaching this problem from a network point of view is extremely important since the domain of complex networks, has a wide set of theoretical results that can improve the behaviour of self-organizing mechatronic systems.

Furthermore these systems can be envisioned from multiple perspectives. In the present context the approach was to explore the networks of skills yet, it would be equally important to explore the network of devices themselves as well as the network of control interactions. Finally, the work documented in this thesis has produced the following scientific publications.

- Self-organization in automation - the IDEAS pre-demonstrator [54]
- IADE – IDEAS Agent Development Environment: Lessons Learned and Research Directions [55]
- A Structural Analysis of Emerging Production Systems [56]

Bibliography

[1] Y.Y. Yusuf, M. Sarhadi, and A. Gunasekaran. Agile manufacturing:: The drivers, concepts and attributes. *International Journal of Production Economics*, 62(1):33–43, 1999.

[2] Hendrik Van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37(3):255 – 274, 1998.

[3] A. Koestler. The ghost in the machine. 1968.

[4] K. Ueda. A concept for bionic manufacturing systems based on dna-type information. In *Proceedings of the IFIP TC5/WG5. 3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*, pages 853–863. North-Holland Publishing Co., 1992.

[5] N. Okino. Bionic manufacturing systems. In *Conference on Flexible Manufacturing Systems, Past, Present-Future (Ed: J. Peklenik), Ljubljana: Faculty of Mechanical Engineering*, 1993.

[6] Y. Koren, G. De Gersem, U. Heisel, H. Van Brussel, F. Jovane, T. Moriwaki, G. Pritschow, and G. Ulsoy. Reconfigurable manufacturing systems. *Manufacturing Technologies for Machines in the Future*, pages 627–665, 2003.

[7] M. Onori. Evolvable assembly systems-a new paradigm? In *33rd Int. Symposium on Robotics (ISR)*, pages 617–621, 2002.

[8] L. Ribeiro, J. Barata, G. Cândido, and M. Onori. Evolvable production systems: an integrated view on recent developments. In *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, pages 841–854. Springer, 2010.

[9] Danny Weyns, Alexander Helleboogh, and Tom Holvoet. How to get multi-agent systems accepted in industry? *International Journal of Agent-Oriented Software Engineering*, 3(4):383–390, 2009.

[10] Vladimir Marik and Jiri Laznsky. Industrial applications of agent technologies. *Control Engineering Practice*, 15(11):1364 – 1380, 2007.

[11] IDEAS 2013. Instantly deployable evolvable assembly sys. *From http://www.ideas-project.eu/*, 2013.

[12] PT Bolwijn and T. Kumpe. Manufacturing in the 1990s -productivity, flexibility and innovation. *Long Range Planning*, 23(4):44–57, 1990.

[13] Andrea Krasa Sethi and Suresh Pal Sethi. Flexibility in manufacturing: A survey. *International Journal of Flexible Manufacturing Systems*, 2:289–328, 1990. 10.1007/BF00186471.

[14] R.N. Nagel and R. Dove. *21st century manufacturing enterprise strategy: An Industry-Led View*, volume 1. Lehigh University Press, 1991.

[15] SL Goldman, RN Nagel, and K. Preiss. Agile competitors and virtual organizations: strategies for enriching the customer. 1995.

[16] H. Van Brussel, P.H. Hannover Germany, and V. Brussel. Holonic manufacturing systems, the vision matching the problem. In *First European Conference on Holonic Manufacturing Systems*. Citeseer, 1994.

[17] S. Bussmann and D.C. McFarlane. Rationales for holonic manufacturing control. In *Proc. of Second Int. Workshop on Intelligent Manufacturing Systems*, pages 177–184, 1999.

[18] Radu Babiceanu and F. Chen. Development and applications of holonic manufacturing systems: A survey. *Journal of Intelligent Manufacturing*, 17:111–131, 2006. 10.1007/s10845-005-5516-y.

[19] K. Ueda, I. Hatono, N. Fujii, and J. Vaario. Reinforcement learning approaches to biological manufacturing systems. *CIRP Annals-Manufacturing Technology*, 49(1):343–346, 2000.

[20] M.G. Mehrabi, A.G. Ulsoy, and Y. Koren. Reconfigurable manufacturing systems and their enabling technologies. *International Journal of Manufacturing Technology and Management*, 1(1):114–131, 2000.

[21] Hoda A ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International journal of flexible manufacturing systems*, 17(4):261–276, 2005.

[22] Luis Ribeiro and Jose Barata. Self-organizing multiagent mechatronic systems in perspective. In *Proceeding of the 11th IEEE International Conference on Industrial Informatics (INDIN 2013)*, 2013.

[23] D. Semere, J. Barata, and M. Onori. Evolvable assembly systems: Developments and advances. In *Assembly and Manufacturing, 2007. ISAM'07. IEEE International Symposium on*, pages 282–287. IEEE, 2007.

[24] M. Onori, D. Semere, and J. Barata. Evolvable assembly systems: From evaluation to application. *Innovation in Manufacturing Networks*, pages 205–214, 2008.

[25] L.D.F. Ribeiro. Diagnosis in evolvable production systems. 2012.

[26] Luis Ribeiro and Jose Barata. Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging it based production paradigms. *Computers in Industry*, 62(7):639 – 659, 2011.

[27] J. Barata and M. Onori. Evolvable assembly and exploiting emergent behaviour. In *Industrial Electronics, 2006 IEEE International Symposium on*, volume 4, pages 3353–3360, 2006.

[28] Mauro Onori, Jose Barata, and Regina Frei. Evolvable assembly systems basic principles. In *Information Technology For Balanced Manufacturing Systems*, volume 220 of *IFIP International Federation for Information Processing*, pages 317–328. Springer Boston, 2006.

[29] Michael Wooldridge, Nicholas R Jennings, et al. Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152, 1995.

[30] Michael Wooldridge and Nicholas R Jennings. Agent theories, architectures, and languages: a survey. In *Intelligent agents*, pages 1–39. Springer, 1995.

[31] Michael Wooldridge. *An introduction to multiagent systems*. Wiley. com, 2008.

[32] J.A.B. de Oliveira. *Coalition based approach for shop floor agility–a multiagent approach*. PhD thesis, Universidade Nova de Lisboa, 2003.

[33] Luis M Camarinha-Matos and Walter Vieira. Intelligent mobile agents in elderly care. *Robotics and Autonomous Systems*, 27(1):59–75, 1999.

[34] Luís Ribeiro, José Barata, Bruno Ferreira, and Jorge Pires. An architecture for a fault tolerant highly reconfigurable shop floor. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 1194–1199. IEEE, 2008.

[35] José Barata, Luís Camarinha-Matos, and Gonçalo Cândido. A multiagent-based control system applied to an educational shop floor. *Robotics and Computer-Integrated Manufacturing*, 24(5):597–605, 2008.

[36] Luis Ribeiro, Jose Barata, and Armando Colombo. Supporting agile supply chains using a service-oriented shop floor. *Engineering Applications of Artificial Intelligence*, 22(6):950–960, 2009.

[37] J. Goldstein. Emergence as a construct: History and issues. *Emergence*, 1(1):49–72, 1999.

[38] J. Holland. *Holland, Emergence: from chaos to order*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1998.

[39] Mark A. Bedau. Weak emergence. *Nous*, 31:375–399, 1997.

[40] E. Bonabeau and J.L. Dessalles. Detection and emergence. *Intellectica*, 25(2):85–94, 1997.

[41] M.A. Bedau. Is weak emergence just in the mind? *Minds and Machines*, 18(4):443–459, 2008.

[42] Peter A. Corning. The re-emergence of emergence: A venerable concept in search of a theory. *Complexity*, 7(6):18–30, 2002.

[43] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. In Sven Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 77–91. Springer Berlin, Heidelberg, 2005.

[44] C. Castelfranchi. The theory of social functions: challenges for computational social science and multi-agent learning. *Cognitive Systems Research*, 2(1):5–38, 2001.

[45] H. Haken. *Information and self-organization: A macroscopic approach to complex systems*, volume 40. Springer Verlag, 2006.

[46] P. Bak, C. Tang, K. Wiesenfeld, et al. Self-organized criticality. *Physical review A*, 38(1):364–374, 1988.

[47] IDEAS 2013. Modular production system boosts in-house assembly. *From http://ec.europa.eu/research/industrial_technologies/success-stories_en.html*, 2013.

[48] L. Ribeiro, A Rocha, and J. Barata. A product handling techincal architecture for multiagent-based mechatronic systems. 2012.

[49] Andre Rocha. An agent based architecture for material handling systems. 2013.

[50] Weiming Shen. Distributed manufacturing scheduling using intelligent agents. *Intelligent Systems, IEEE*, 17(1):88–94, jan/feb 2002.

[51] FIPA. Fipa request interaction protocol specification, 2002.

[52] FIPA. Fipa contract net interaction protocol specification, 2002.

[53] Luis Ribeiro and José Barata. Deployment of multiagent mechatronic systems. In *Industrial Applications of Holonic and Multi-Agent Systems*, pages 71–82. Springer Berlin Heidelberg, 2013.

[54] L. Ribeiro, J. Barata, M. Onori, C. Hanisch, J. Hoos, and R. Rosa. Self-organization in automation - the ideas pre-demonstrator. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pages 2752 –2757, nov. 2011.

[55] Luis Ribeiro, Rogerio Rosa, Andre Cavalcante, and Jose Barata. Iade - ideas agent development environment: Lessons learned and research directions. In *Proceedings of the 4th CIRP Conference on Assembly Technologies and Systems*, 2012.

[56] L. Ribeiro, R. Rosa, and J. Barata. A structural analysis of emerging production systems. In *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, pages 223 –228, july 2012.

7

Appendix 1 - FIPA Request

The FIPA Request, is a protocol that allows an agent to request to other agent to perform an action. The Figure 7.1 illustrates the protocol. In this protocol, the *Initiator* agent starts to do a request to the *Participant* agent. The *Participant* can accept that request or refuse it. In case of a refusal, it sends a refuse message and the conversation ends. If it agrees it sends an agree message. The agree or refuse message are optional so the *Participant* can choose to bypass those messages and send an inform or failure.

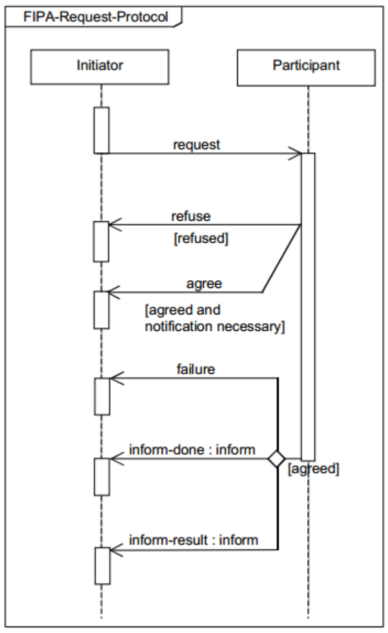


Figure 7.1: Protocol FIPA Request

In case of an agree, the *Participant* proceeds with the processing of the requested task. As soon as the task is processed, the *Participant* sends an inform message in case of success, or a failure message in case of a fault.

8

Appendix 2 - FIPA Contract Net Protocol

The Contract Net protocol (Figure 8.1), allows an agent to start a negotiation with several other agents. This protocol begins with the *Initiator* agent sending a message requesting a proposal to *m Participant* agents. Each one of the *Participants* will respond with a refusal (e.g. if the *Participant* cannot handle the requested task), or with a proposal.

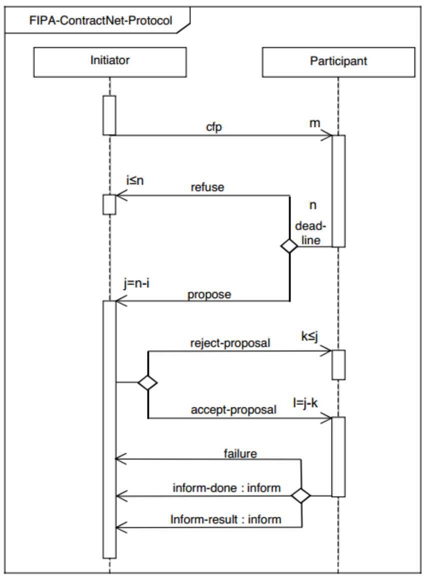


Figure 8.1: Protocol Contract Net

As soon as the *Initiator* receives all the responses (proposals and refusals) it will proceed with an evaluation of all the proposals. In case of rejected proposals, the *Initiator* will send a refuse-proposal to the corresponding *Participants*, and the interaction between those agents ends here.

To all the *Participants* that have been evaluated with success, the *Initiator* will send an accept-proposal. After receiving an accept-proposal, the *Participants* will internally process the requested action by the *Initiator*. As soon as the action is done, the *Participants* should respond with an inform message in case of success, otherwise with a failure message.

9

Appendix 3 - T2 Results

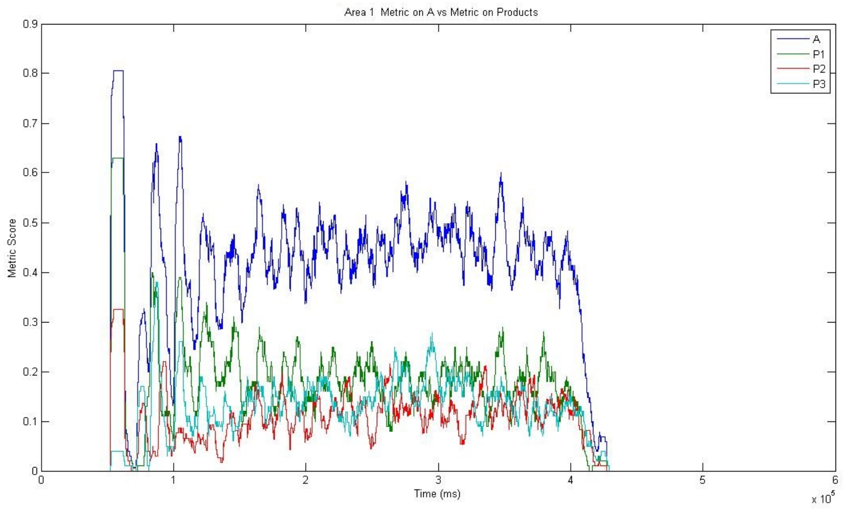


Figure 9.1: Contribution of PA types for the CLA implementing A in Area 1 (T_2)

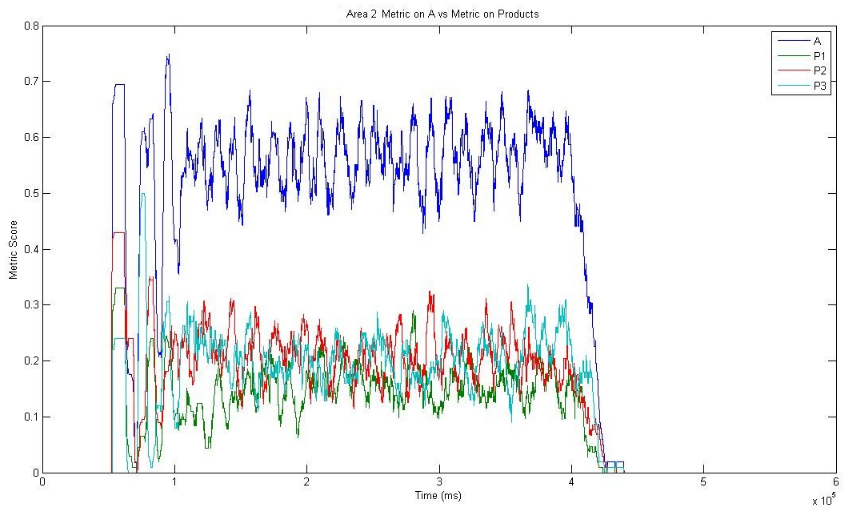


Figure 9.2: Contribution of PA types for the CLA implementing A in Area 2 (T_2)

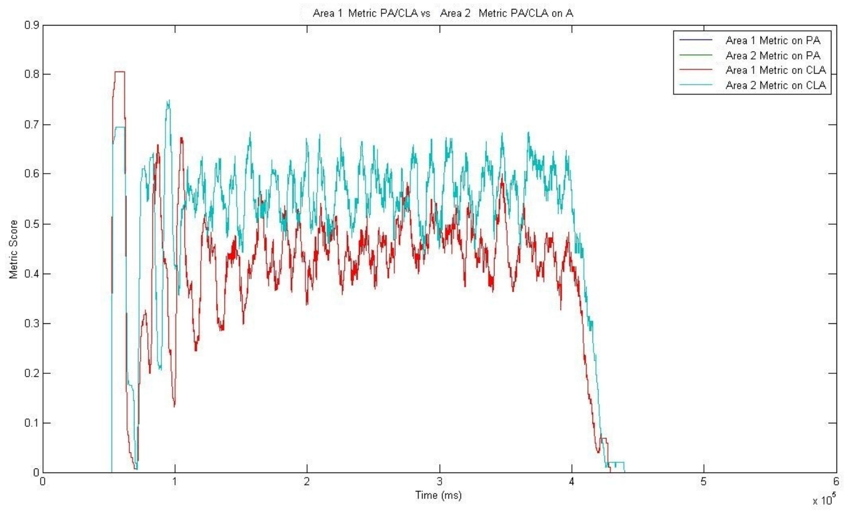


Figure 9.3: Comparison of the metric values computed by the CLAs and seen by PAs over time in both areas for A (T_2)

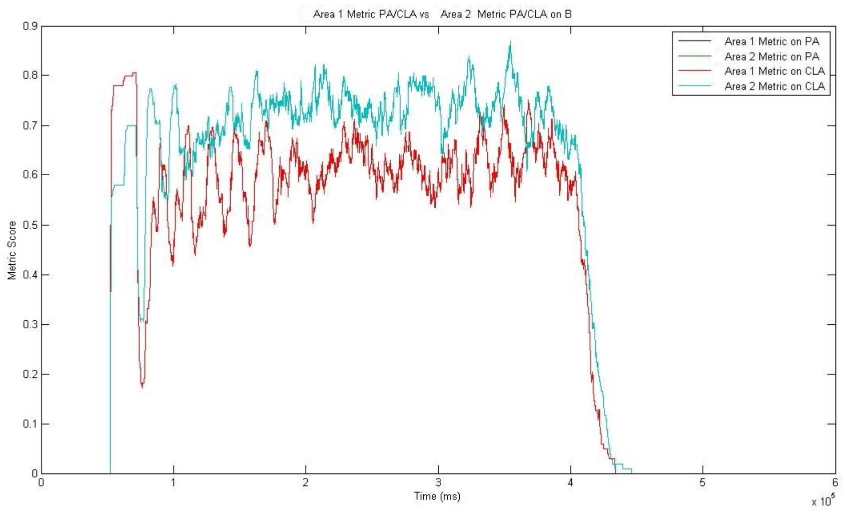


Figure 9.4: Comparison of the metric values computed by the CLAs and seen by PAs over time in both areas for B (T_2)