Agent-Based Manufacturing Systems: A generic platform and its methodology for technological migration from FMS to RMS

by

Jorge Miguel Gamboa Revilla

Monterrey, N.L., June 2008
Agent-Based Manufacturing Systems: A
generic platform and its methodology for
technological migration from FMS to RMS

by

Jorge Miguel Gamboa Revilla

T h e s i s

Submitted to the Graduate Program in Mechatronics and Information Technologies

at the

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

in partial fulfillment of the requirements for the degree of

Master of Science

in

Automation

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L., June 2008
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
Division of Mechatronics and Information Technologies
Graduate Program in Mechatronics and Information Technologies

Thesis committee members recommend the present thesis of Jorge Miguel Gamboa Revilla, Eng. to be accepted in partial fulfillment of the requirements for the degree of Master of Science in Automation.

Thesis Committee

Sergio William Sedas
Gersey, Ph.D.
Thesis advisor

Miguel J. Ramírez Cadena, M.Sc.
Examiner

Manuel Eduardo Macías García, Ph.D.
Examiner

Graciano Dieck Assad, Ph.D.
Director of the Graduate Programs in Mechatronics and Information Technologies

June 2008
Abstract

More complex products in efficiency and quality, the necessity of diminish energy spending and investment reduction; amongst other things, are disturbances, and their occurrence may have severe impact in the performance of actual manufacturing systems. Manufacturing systems should be based in distributed and autonomous entities, being possible the addition of new components without stopping or re-starting processes. All these facilities point to the concept of agile manufacturing systems.

The main propose of this project is to design a multi-agent system platform on a Flexible Manufacturing Cell (FMC). Taking into consideration all levels of functional issues, cellular architecture, standardization, and technical configurations. LabView programming platform is selected for this development due to its easy applicability on holonic structures with features such as object oriented programming, distributed intelligence, and block function representation.

The approach is addressed to encourage the usage of holonic and multi-agent concepts in traditional production lines, with a friendly software upgrade and a minimum cost in hardware expansion. A methodology that includes the technological migration from a established flexible manufacturing structure (FMS) to intelligent and reconfigurable manufacturing system (RMS) is presented. There are several study cases, which are based on implementation examples that will be described in depth to show the viability of the proposed schema (reconfiguration, intelligence adoption, migration, integration, and tele-operation).
Dedicated

To my father Gerardo Gamboa
For his unconditional support and love

To my mother Silvia Revilla
For her love and affection
Acknowledgements

To my advisor, Miguel J. Ramírez Cadena, for your support, and for the encouragement of making this thesis more than a graduation requirement.

To Graciano Dieck Assad, Ph.D., for supporting in part the publication of two articles regarding this project.

To Sergio William Sedas Gersey, Ph.D., for taking the time to meet and discuss the topics related to this work.

To Manuel Eduardo Macías García, Ph.D., for your help, and advices, and for providing the necessary technology to launch the teleoperation system.

To Lyz Irene Flores Pierluisi, for your love and patience.

To my friends at the Mechatronics Department: Rodrigo Vargas, Celina Rea, Ariel Cano, Ángel Orozco, Jorge Claros, Geovanna Ruffo, Ignacio Barradas, Juan Carlos Tudón, Joel Castillo, Jorge Jiménez, Aldo Cedillo, José Ernesto Pérez, Pablo Ordoñez, Pedro Carstensen, Julio Ávila, Luis Carlos Félix, Christopher Ramírez, and all friends I made in this open chapter of my life.
# Contents

1 Introduction
   1.1 Motivation ............................................. 2
   1.2 The Problem ........................................... 2
   1.3 Objective ............................................ 2
   1.4 Related Works ........................................ 3
   1.5 Contributions ........................................ 4
   1.6 Thesis ................................................ 4
   1.7 Contents ............................................. 5

2 State of the art
   2.1 Agents and Holons ................................... 7
   2.2 Agile Manufacture (AM) ............................. 8
   2.3 Reconfigurability vs. Flexibility .................. 9
   2.4 Literature research ................................ 13

3 MAS Architecture and migration planning for GP
   3.1 MAS definition ...................................... 15
   3.2 MAS Design Considerations ......................... 16
   3.3 Migration planning .................................. 17
   3.4 The general model of interaction .................. 17
   3.5 Resources and technology reachable ............... 18
   3.6 Holonic implementation example ................... 20

4 The Methodology for GP
   4.1 The feature of migration ........................... 23
   4.2 The Methodology ..................................... 24
      4.2.1 Define Communication Structure ............... 26
      4.2.2 Isolate from global system .................... 27
      4.2.3 Convert from general to particular .......... 29
      4.2.4 Create relationships but not dependences .... 29

5 The study case
   5.1 The ITESM manufacturing cell description ....... 31
   5.2 Generic Platform design (GP) ..................... 33
5.3 Robot and camera agent ........................................ 37
  5.3.1 Agent Architecture for generic platform ................. 37
  5.3.2 Reconfigurability and agent implementation .......... 39
  5.3.3 Using LabView for intelligence adoption ............. 40
5.4 Adopting teleoperation facility ............................... 41
5.5 Contract Net protocol ........................................ 44
  5.5.1 Labview for AI ............................................ 44
  5.5.2 Contract-Net using LabView and Netlogo ............. 45

6 Guidelines for future works ................................. 49
  6.1 Adding new elements ........................................ 49
  6.2 Intelligence implementation ................................ 51
  6.3 Upgrading Teleoperation .................................... 51

7 Conclusion ...................................................... 55
  7.1 Conclusions and future works ............................... 55

A Robot Motoman UP6 .............................................. 61
  A.1 Using Visual C++ 6.0 ......................................... 61
  A.2 Creating *.dll from *.cpp ................................ 63
  A.3 Creating *.lsb from *.c .................................... 65

B OPC and OOP with LabView .................................... 67
  B.1 Connecting LabVIEW to RSLinx .............................. 68
      B.1.1 Configuring an Allen Bradley PLC in RSLinx .... 69
      B.1.2 OPC in LabView Example ............................. 73
  B.2 Object Oriented Programming ............................... 74

C ActiveX for Vision Systems .................................. 77
  C.1 Accessing an ActiveX-Enabled Application ................ 77
  C.2 Structure of connection, Vision system .................. 78

D State machines, an option for reactive intelligence ......... 79
  D.1 LabView State Diagram Toolkit ............................ 79

E Glosary .......................................................... 81
  E.1 Glosary of common terms ................................... 81

F Publications ..................................................... 83
  F.1 A Generic multi-agent architecture design in a FMC .... 83
  F.2 Intelligent Manufacturing Systems: a methodology for ... 83
List of Figures

1.1 Contribution Map ................................................. 4
1.2 Floor control design with agent-based coordination ........... 5
1.3 Thesis Schema .................................................... 6

3.1 Typical structure of a multiagent system. ...................... 16
3.2 Interaction general model. ...................................... 18
3.3 Object Oriented Programming ................................. 20
3.4 PetriNet production changes example .......................... 21

4.1 Topologies of manufacturing cells. .............................. 24
4.2 General View of traditional FMC operation flux. .............. 26
4.3 Communication HRA definition, from methodology 1st stage. 27
4.4 Multithreading example for isolation, from methodology 2st stage. 28
4.5 The methodology implemented to achieve migration feature. 30

5.1 Layout of ITESM Flexible Manufacturing Cell .................. 32
5.2 UML for Static operation standard cell representation. ....... 33
5.3 Dynamic operation of standard cell representation. ........... 34
5.4 The distributed platform cycle design and message structure. 35
5.5 First screen of HMI prototype platform ....................... 36
5.6 Second screen of HMI prototype platform ..................... 36
5.7 Third screen of HMI prototype platform ....................... 37
5.8 Multiagent abstract architecture for study case ................ 40
5.9 Dynamic study case representation ............................ 40
5.10 Real working of robot and camera, results .................... 41
5.11 LabView intelligence source code structure .................... 42
5.12 LabView intelligence application HMI .......................... 42
5.13 Teleoperation study case schema .............................. 43
5.14 Access Webpage for prototype .................................. 43
5.15 Client application first view .................................... 44
5.16 Products and parts for their assembly, example ................ 46
5.17 NetLogo and LabView interaction .............................. 46
5.18 Contract-Net vs. FIFO vs. Random ............................ 47
5.19 Experiments for Contract-Net Protocol ....................... 48
6.1 Prototype structure for adding new elements. .......................... 50
6.2 Teleoperation technical structure for clustering. ....................... 52

A.1 Robotic technical implementation structure. .......................... 61
A.2 NI external code nodes. ............................................. 62
A.3 Robot source code and schema of work. ............................... 62

B.1 OPC communication structure in platform. .......................... 68
B.2 Technical OPC implementation in source code. ....................... 69
B.3 OPC instruction 1. .................................................. 70
B.4 OPC instruction 2. .................................................. 70
B.5 OPC instruction 3. .................................................. 71
B.6 OPC instruction 4. .................................................. 71
B.7 OPC instruction 5. .................................................. 71
B.8 OPC instruction 6. .................................................. 72
B.9 OPC instruction 7. .................................................. 72
B.10 OPC instruction 8. ................................................. 72
B.11 OPC instruction 9. .................................................. 73
B.12 OPC instruction 10. ............................................... 73
B.13 OPC instruction 11. ............................................... 73
B.14 OPC instruction 12. ............................................... 74
B.15 Implementation of OOP in platform. ................................. 75

C.1 Vision System structure implemented. ................................. 78
List of Tables

2.1 Flexibility definition ................................................. 10
2.2 Manufacturing structures and objectives .......................... 12
2.3 Articles list .............................................................. 13
2.4 Articles Classification .................................................. 14
3.1 Comparition Table C++ vs. LabView ................................. 19
5.1 Communication channel selection ...................................... 34
Chapter 1

Introduction

Over the last 20 years, integrated information systems have undeniably helped to solve most of the issues concerning process management and control; reducing production costs; improving process capabilities and yields; monitoring and controlling process status; and enhancing overall production. Nevertheless, the management of an effective and modern manufacturing industry should not only invest in systems that perform better and provide more profitable ways to exploit business processes, but should also encompass wider horizons, embracing all of the critical business layers within and beyond the company’s boundaries.

Today’s marketplace is increasingly more demanding in terms of lower costs, faster time-to-market, and better quality, thus forcing companies to become ever more reactive and agile in performing their daily business management tasks. Some manufacturing industries have founded their businesses on shorter life-cycle products or have diversified into more competitive markets in different industrial sectors. The most direct implication of this evolution is that modern manufacturing companies should be able to act like cells in an organism (the market). In simple terms, the business model is changing from open competition to one in which, for the organism to survive, strong, effectively linked cooperation among businesses horizontally and vertically is fundamental.

Manufacturing Systems have had several redefinitions, in the eighties, the concept of flexible manufacturing systems (FMS) was introduced to develop a new family of products with similar dimensions and constraints. But nowadays, the capacity of reconfiguration has become a major issue for improving the functioning of industrial processes. Indeed, today a main objective is to adapt quickly in order to start a new production or to react in a failure occurrence [10]. Intelligent manufacturing systems (IMS)[24], has both flexibility and reconfigurability, in fact this concept brings more than a few ideas of software intelligence meanings, which contemplates characteristics such as autonomy, decentralization, flexibility, reliability, efficiency, learning, and self-regeneration, all of these facilities lead to the concept of agent-based manufacturing systems.
Chapter 1. Introduction

1.1 Motivation

On the past decade researchers have focused their investigations in the theory and design of Holonic Manufacturing Systems (HMS) and Agent-based Manufacturing systems (AMS), where can be found two principal aspects that at present are still being depurated. On one hand we have issues associated with the development of agent-based systems, on the other hand how the agent-based systems can be effectively deployed into manufacturing environments [12].

In spite of having a complete set of intelligent structures and algorithms, they still do not have the strength to displace the investments in hardware or anything that have an acceptable performance on existing companies, even though the companies know that in a brief time market will change and some actions have to be taken. Manufacturing control systems should be therefore be based in distributed and autonomous entities, expandable, being possible the addition of new components without stopping or re-starting the process. All these necessary facilities lead to the concept of agile manufacturing systems [27].

1.2 The Problem

More complex products, faster changing products, faster introduction of products, and the requirement of reduce investment, are disturbances; their occurrence may have severe impact in the performance of manufacturing systems, being also necessary to improve the system performance in terms of response to change [27, 3].

The lack of uptake is in the most part caused by the requirement of some metrics, for example; how to evaluate the design, construction and operation of a agent-based system vs. actual mass production systems, it is essential to deal with adequate software programming tools in order to map the agent-based concepts to the software design.

1.3 Objective

The main objective of this project is to implement the agent-based manufacturing concepts to permit the deployment of agent concepts, being able to acquire adoption of new elements and AI complex algorithms to encourage the usage of agents technology. In addition, final product should open the possibility of migration; from a
1.4 Related Works

Recent works based on this new tendency, have came up with sophisticated structures and algorithms that would satisfy some intelligent features on structures such as HMS, online scheduling reconfiguration, contract net negotiation [35], and plug and produce systems [39]. Some of these topics are not used on this project, however are mentioned in order to motivate and encourage the study of them, which could be used to follow this investigation developing new research lines.

Scheduling reconfiguration, this feature permits the online restructuration of the entire production set, considering that each element is assigned to interact with an specific product in order to achieve production, in other words when a new production set is demanded the system should be able to re-schedule tasks from each element of the cell, this schema could seem to be an easy implementation, however there are a complete set of different algorithms that are still depurated.

Plug and produce, an assembly system often needs to change its configuration of assembly devices such as inserting machines, robots, conveyers and parts feeders. Reconfiguration of the system is required in such cases as (1) breakdown or replacement of a device; (2) change of a product, and so on. When an assembly device is added (or removed, replace), the reconfiguration needs long start-up time in ordinary systems. This is partly because one device may affect the entire control system. To decrease the start-up time, the reconfiguration of the entire system should be separated from that of the device. The interface of the devices onto the control system must be established as easily as possible. This concept is called Plug and produce and was first introduced by T. Arai, Y. Aiyama, M. Sugi, and J. Ota, 2001 [39].

Each one of those approaches, are branches of the IMS topics, making possible the adoption of a new and agile manufacturing structures, being prepared for a more strict demand, consumers that won’t be conformed with poor quality products, and being also ready for a new generation of manufacturing. The argument of this work can be summarized by saying that intelligent manufacturing systems built along these agent-based or holonic lines are likely to demonstrate superior efficiency and effectiveness, through their ability to cut through the complexity that overwhelms system designed along lines of conventional organizational architecture.


1.5 Contributions

This work attempts to introduce a new concept, making possible the adoption and reconfiguration of actual mass production lines without dramatic changes in hardware, avoiding extra coding.

To say this in another way the final product should be seen as is illustrated on figure (See fig. 1.1).

![Figure 1.1: This figure pretends to show a map of contributions from global thesis product.](image)

1.6 Thesis

This scheme presents a novel approach to manufacturing floor control design with agent-based coordination, which should be ready to interact with superior enterprise levels such as a Manufacturing Execution System (MES), (See fig. 1.2).

From the structure presented on figure 1.2 which is based on a previous research [13, 14], we will base the development of the platform that throughout this work we shall describe in deeper. This scheme uses commercial software that includes a few mainly distinctive characteristics, such as the flexibility to scale platform capacities without missing structure concept. Although, this is the main idea of object-oriented programming, it is much more illustrative to visualize block functions than extended codes lines. Once an agent is developed it must be as general as possible in order to be adapted in a wide range of industrial equipment that falls into the same category depending only on a minimum of physical requirements and intelligence training. This project will refer to a multi-agent manufacturing
platform implemented at Monterrey Institute of Technology (ITESM), and along this project the agent-based coordination will be implemented on one sub-system, including a framework to evaluate its performance, addressing this topic for future system improvements. A methodology to convert conventional manufacture systems into new intelligent manufacturing, flexible and reconfigurable concept shall be explained in detail, to refer this deployment a **generic platform GP** shall be created.

### 1.7 Contents

From now on, next sections will guide the lector through this research, having a more general view of this scheme. Next illustration could be helpful to map this work, however next lines will describe how each part of this work will be built up (See fig. 1.3).

As a primary part all theoretical elements are presented (**Ch.2**), gather together latest research from journals, papers and oriented books, this section compiles the state of art.

Next step is migration planning where all available resources are studied to plan migration feature, and the most important part the methodology is presented, which is the core of the project. Migration from flexible manufacturing systems to reconfigurable manufacturing systems, is based on performing the methodology clearly explained on **Ch.3 and Ch. 4**.

Once theoretical concepts and methodology is given, reader will be enrolled with
the platform design, at this part each technical part will be explained on a general way, such as blocks diagrams and how conceptually the generic platform works. Besides contract net, which is an AI algorithm, teleoperation, intelligence adoption are study cases explained as well on (Ch. 5).

Further on three experimental examples of the platform deployment and their results shall be described, in order to show platform viability, on the other hand guide lines and technical details for future works are itemized on Ch. 6. However conclusions and guidelines for future works such as intelligent adoption, or reconfiguration, amongst other things, will be detailed on Ch. 7.
Chapter 2

State of the art

Future manufacturing systems may be quite different from those of today. Although mechanisation and automation have done much to increase efficiency, improve reliability, and reduce the cost of producing goods, traditional manufacturing systems tend to be inflexible and lack robust handling of disturbances. It can be said that they lack agility, in the sense that they are incapable of responding to change gracefully. An agile manufacturing system should be able to handle significant disturbances and reconfiguration changes.

2.1 Agents and Holons

Before continuing with theoretical aspects is important to define important concepts such as Agent and Holon meanings, those concepts belongs to fields of AI (artificial intelligence), however in last 10 years research have been tried to incorporate into manufacturing structures, researchers have demonstrated that their implementation would increase the overall performance of traditional systems. An agent is a computer system that is situated in some environment, and that is capable to act in an autonomous way in this environment in order to meet its design objectives. Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their goals, this characteristic is well known as reactivity. However an agent is also proactive, for it agent is able to exhibit goal directed behavior by taking the initiative. In addition, agents are social, having the ability to interact with other agents [43]. It worth to remember the definition of an ”Holon”, which its similarities with agent definition, brings up controversial meanings, nevertheless an holon is well recognized on manufacture applications with the distinctive of a more specific intelligence use, while an agent could have different levels of intelligence such as logical, reactive, layered or in a more advanced way, with beliefs, desires and intentions (BDI)[41].

The word ”holon”, first introduced by Koestler in 1967 [23], comes from the Greek holos that means whole, with the suffix on which, as in proton or neutron, suggests
Chapter 2. State of the art

a particle or part. Holons refer to any component of a complex system that, even when contributing to the functioning of the whole, demonstrates autonomous, stable, and self-contained behavior or function. In other words an Holon is an autonomous, co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. A system of holons that co-operate to achieve a goal or objective limited by rules of interaction is called holarchy [1].

It is easy to understand how concepts such as flexible manufacturing systems FMS [2] and computer integrated manufacturing CIM [34, 17], which characterized the approach to manufacturing in the 1980s and 1990s, evolved into the Holonic Manufacturing Systems HMS paradigm. CIM represented the introduction of a strong vertical integration of manufacturing subsystems into a hierarchical structure allowing information needed for production to flow from the top levels of an enterprise to the shop floor. HMS moves toward a more flexible integration of functionalities among distributed autonomous actors. Agile manufacturing AM is a more recent evolution of the previously mentioned concepts. It extends the concept of flexibility beyond manufacturing systems boundaries into the environment and the marketplace, where management of customer relations and cooperation among companies are even more important.

2.2 Agile Manufacture (AM)

Agility requires the efficient and effective utilization of internal and external resources to meet changing customer needs quickly and flexibly [15]. AM should lower manufacturing costs; increase market share; satisfy customer requirements; facilitate the rapid introduction of new products; eliminate non-value-added activities; and increase manufacturing competitiveness [6, 16].

The concepts of AM and Holonic Manufacturing System HMS are strictly intertwined, and many definitions highlight this fact. For Christensen [7] and Deen [9], an HMS is also depicted as "manufacturing system where key elements, such as raw materials, machines, products, parts, etc., have autonomous and cooperative properties." Fore Shen and Norrie [37], the integration of the entire range of autonomous and cooperative manufacturing activities (from order booking through design, production, and marketing) allows an enterprise to achieve agile manufacturing processes.

Several articles and books have attempted to explain how to make a company agile. According to the definition by Cho et al. [6], AM is "the capability of surviving and prospering in the competitive environment of continuous and unpredictable change by reacting quickly and effectively to changing markets, driven by customer-designed products and services." More specifically, Lee and Thornton believe that central to
being "agile" is the ability of a company to be "agile in design" [26]. The ability to understand a product and what is critical therein is key to success of a company and its ability to become more agile.

Some enabling technologies, such as the standards for product exchange; concurrent engineering; virtual manufacturing; component based hierarchical shop floor control systems; and information and communication infrastructures, etc., are critical to successfully achieving AM [6]. In general, innovation in information technology and in organizational models should shorten the path toward agility. In today’s business environments, information technology and organization are two strictly intertwined concepts, and it is not easy to assess, a priori, which of them is the engine driving the other to evolve. One thing is certain: information technology is expected to provide a means to allow the effective flow of information within an efficient organization and outside the organization boundaries.

### 2.3 Reconfigurability vs. Flexibility

The definitions of manufacturing flexibility found in literature either directly refer to the firm’s context or derive from general definitions of flexibility born in other disciplines (such as the biological-evolutionary one, the anthropological one, that of the theory of systems, etc.) [8]. The origin and application of the definitions of flexibility are summarized in table 2.1.

Manufacturing process and structures have suffered several redefinitions, with concepts such as Flexible, Agile, Reconfigurable, and Holonic. A Flexible system covers the system’s ability to be changed to produce new product types, it worth to remind that those different types of products, most full fit in a category limited by some physical aspects such as size, or weight, in other words a flexible manufacturing cell could be compared with an Crescent wrench that come in several sizes, 4”, 6”, 8”, and those are also adjustable to fit a family of sizes. However in spite of being a useful solution if the application change in a dramatic way, for example changing the type of screws heads, those wrenches becomes useless, while wrenches with a reconfigurability feature such as replacing heads, could maintain its utility. The adaptability to dramatic changes is one of the characteristics that can be found on reconfigurable systems over flexibility. Even both structures are little different, they point to the same direction Intelligent Manufacturing Systems (IMS), more complex products, faster changing products, faster introduction of products, and the requirement of reduce investment, make the necessity of improving the system performance in terms of response to change.

Most FMS comprise of three main systems. The work machines which are often automated CNC machines, robots, conveyors, transfers, etc. which are connected by a material handling system (conveyor or transfer) to optimize parts flow, and a
## Table 2.1: Origin and application of the definitions of flexibility.

<table>
<thead>
<tr>
<th>Other Disciplines</th>
<th>Manufacture Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>• As a characteristic of the interface between a system and its environment</td>
<td>Range of states reachable and time for moving (determined by demand variety and uncertainty)</td>
</tr>
<tr>
<td>• As a degree of homeostatic control and dynamic efficiency</td>
<td></td>
</tr>
<tr>
<td>• As a capability of adaptation or change</td>
<td></td>
</tr>
<tr>
<td>• Low cost for changing (economic approach)</td>
<td></td>
</tr>
<tr>
<td>• Changing without disorganization (organizational approach)</td>
<td></td>
</tr>
<tr>
<td>• Volume, mix, product, process changes (operational approach)</td>
<td></td>
</tr>
<tr>
<td>• Competitive priorities and businesses changes (strategic approach)</td>
<td></td>
</tr>
</tbody>
</table>
central control computer which controls material movements and machinery flow. Those sub-systems, are controlled by PLC controllers, frequency drivers, pneumatic actuators, amongst other things, it is important to remind that mostly all sub-systems are able to communicate data of the process itself being able information that actually could bring the upgrade of the system.

Each element work together in a sequential way to achieve a production set, which actually could be reprogrammed to create several kinds of products belonging to a family of products. Although it could be a stable application if for some reason, decreasing time of production for example, a new robot is attached to the FMC sequential algorithms would crash if a hard reconfiguration of the cell is not performed. RMS should be able to avoid crashing, in a general point of view the system should be similar to FMS, nevertheless reconfigurability would let the addition of hardware without corrupting the sequential process if it exist.

Changing manufacturing environment characterized by aggressive competition on a global scale and rapid changes in process technology requires creating production systems that are themselves easily upgradable and into which new technologies and new functions can be readily integrated. Table 2.2 taken from Mehrabi, Ulsoy, and Koren work [29] summarizes the major manufacturing paradigms and their definitions. Mass production systems were focused on the reduction of product cost. Lean Manufacturing places emphasis on continuous improvement in product quality while decreasing product costs. Flexible manufacturing systems make possible the manufacture of a variety of products (flexibility) on the same system. While this is an important objective, these systems have met with limited success. For instance, flexible manufacturing systems (FMSs) developed in the last two decades: (i) are expensive, since in many cases they include more functions than needed, (ii) utilize inadequate system software, since developing user-specified software is extremely expensive, (iii) are not highly reliable, and (iv) are subject to obsolescence due to advances in technology and their fixed system software / hardware.

The high risk of an expensive flexible production system becoming obsolete is one of manufacturers’ most troubling problems. Because advances in computers, information, processing, controls, optics, high-speed motors, linear drives, and materials sometimes occur in cycles as short as six months, today’s most efficient production system can become inefficient after a short time. Furthermore, the current customer-driven market and increased awareness of environmental issues lead to the ever-quicker introduction of new products. But adaptation of existing production systems to new products is slow and the launching of new systems can take a long time (up to two years for a machining system).
Table 2.2: Summary of definitions and objectives.

<table>
<thead>
<tr>
<th>Systems (machining/manufacturing)</th>
<th>Definition and Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machining System</td>
<td>One or more metal removal machine tools and tooling, and auxiliary equipment (e.g., material handling, control, communications), that operate in a coordinated manner to produce parts at the required volumes and quality.</td>
</tr>
<tr>
<td>Dedicated Machining Systems</td>
<td>A machining system designed for production of a specific part, and which uses transfer line technology with fixed tooling and automation. The economic objective of a DMS is to cost-effectively produce one specific part type at the high volumes and the required quality.</td>
</tr>
<tr>
<td>Flexible Manufacturing Systems</td>
<td>A machining system configuration with fixed hardware and fixed, but programmable, software to handle changes in work orders, production schedules, part-programs, and tooling for several types of parts. The economic objective of a FMS is to make possible the cost-effective manufacture of several types of parts, that can change overtime, with shortened changeover time, on the same system at the required volume and quality. Note: A part family is defined as one or more part types with similar dimensions, geometric features, and tolerances, such that they can be produced on the same, or similar, production equipment.</td>
</tr>
<tr>
<td>Reconfigurable Manufacturing Systems</td>
<td>A machining system which can be created by incorporating basic process modules, both hardware and software, that can be rearranged or replaced quickly and reliably. Reconfiguration will allow adding, removing, or modifying specific process capabilities, controls, software, or machine structure to adjust production capacity in response to changing market demands or technologies. This type of system will provide customized flexibility for a particular part family, and will be open-ended, so that it can be improved, upgraded, and reconfigured, rather than replaced. The objective of an RMS is to provide the functionality and capacity that is needed, when it is needed. Thus, a given RMS configuration can be dedicated or flexible, or in between, and can change as needed. An RMS goes beyond the economic objectives of FMS by permitting: (1) reduction of lead time for launching new systems and reconfiguring existing systems, and (2) the rapid manufacturing modification and quick integration of new technology and/or new functions into existing systems.</td>
</tr>
</tbody>
</table>
2.4 Literature research

In order to have a more complete view of literature which inspired this work and gave de basis for this deployment, next tables 2.3 and 2.4 illustrate a classification of works whit significant contribution on these fields.

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Cambridge Holonic Packing Cell, Auto ID Complete Paper</td>
<td>James Brusey, Martyn Fletcher, Mark Harrison, Alan Thorne, Steve Hodges, Duncan McFarlane</td>
</tr>
<tr>
<td>3</td>
<td>Holonic Assembly system with Plug and produce</td>
<td>T.Arai, J.Ota, Y.Aiyama, M.Sugi</td>
</tr>
<tr>
<td>4</td>
<td>holonic manufacturing systems in continuous processing: concepts and control requirements</td>
<td>Duncan C. McFarlane</td>
</tr>
<tr>
<td>5</td>
<td>Intelligent control for holonic manufacturing systems</td>
<td>S Balasubramanian, X Zhang and D H Norrie</td>
</tr>
<tr>
<td>6</td>
<td>Open Controller Architecture - Past, Present and Future</td>
<td>Yusuf Altintas, Francesco Jovane, Yoram Koren, Mamoru Mitsuishi, Shozo , Takata, Hendrik van Brussel, Manfred, Weck, Kazuo, Yamazaki</td>
</tr>
<tr>
<td>7</td>
<td>Organizational foundations of intelligent manufacturing systems the holonic viewpoint</td>
<td>John Mathews</td>
</tr>
<tr>
<td>8</td>
<td>Rationales for Holonic Manufacturing Control</td>
<td>Stefan Bussmann, Duncan c. Mcfarlane</td>
</tr>
<tr>
<td>9</td>
<td>The Ghost in the machine</td>
<td>Arthur Koestler</td>
</tr>
<tr>
<td>10</td>
<td>Concept of holonic Control</td>
<td>Shigeki Sugiyama</td>
</tr>
</tbody>
</table>

Table 2.3: This table present a list of all articles, which inspired this work.
Table 2.4: This table presents a list of all articles classified depending on their contribution.

<table>
<thead>
<tr>
<th>No.</th>
<th>Detailed research works</th>
<th>Holonic concept</th>
<th>HSM representation</th>
<th>MES</th>
<th>Research Applications</th>
<th>State of the art</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Chapter 3

MAS Architecture and migration planning for GP

3.1 MAS definition

Before introducing this chapter is important to define what does a Multiagent system (MAS) architecture mean, to do this we will refer to a typical structure given by Jennings [20], (See fig. 3.1). The system contains a number of agents, which interact with one another through communication. The agents are able to act in an environment; different agents have different spheres of influence, in the sense that they will have control over or at least be able to influence different parts of the environment. These spheres of influence may coincide in some cases. The fact that these spheres of influence may coincide may give rise to dependency relationships between the agents. For example, two robotic agents may both be able to move through a door but they may not be able to do so simultaneously. Finally, agents will also typically be linked by other relationships. Examples might be power relationships, where one agent is the boss of another.

The most important aspect of Multiagent system generally is that when faced with what appears to be a Multiagent domain, it is critically important to understand the type of interaction that takes place between the agents and their influence over entire system.

There are some important considerations before building a MAS, these considerations are mentioned in following paragraphs. In order to ensure functioning and generality of MAS design is essential to respect each of those considerations.
3.2 MAS Design Considerations

The design of any information system relies heavily on the architecture. Many authors accept agent and MAS architecture as the first step of design, using one of its instantiations to describe the problem-solving process. In this respect, Muller [33] listed the following aspects:

- Guidelining: An architecture specification of the agent is by definition a valuable general guideline for the MAS design, as well as for the implementation of the application.

- Structuring: An architecture specification of the agent generally provides the description of the system’s modules and layers, which are the classes of operational knowledge necessary to design and build the MAS.

- Re-use: An architecture is generally related to an implicit execution model which avoids programming from scratch.

- Standards: An architecture is generally related to a set of standards allowing many advantages, among which is the awareness of the possibility to communicate with other systems. In this respect, predefined application-independent mechanisms are usually directly available to the developers as a standard procedure.

- Predictability: The MAS behavior, through the basic patterns of interactions of the instantiated agents, can be predicted up to a certain level.
• Genealogy: An architecture generally allows strategic and functional extensions according to the evolutions requested by the environment in which an MAS acts.

### 3.3 Migration planning

The planning stage should be conformed by few considerations, to mention them; first programmer should understand how elements works before any change, in second place must evaluate every detail of the possible technology reachable in order to accomplish migration. Moving on to a different point, a model of interaction has to be evaluated, in other words how information will pass through every layer of the system. All this paragraphs brings more questions about this matters, however following paragraphs shall explain these issues.

The conventional architecture of a FMS consist on machinery, cells, and factory levels, their control functions are associated with each level ranging from real-time control and rigid operations to planning and scheduling functions at the shop-floor level. Its hardly assumption of changes makes traditional systems more centralized, in addition this structure is not compatible with the ideal holonic infrastructure [36].

This work defines a generic platform, which could be applied on traditional manufacturing systems, where the holonic expansion implies only the upgrade of a workstation with characteristics of cell coordinator. Thus, taking advantage of traditional shop floor interaction; such as OPC communications, dynamic libraries, ActiveX functions, in other words a system which software structure is based on an integrated control, then is possible to apply holonic concept into traditional manufacturing systems. It is essential for this development to preserve traditional hardware configurations, with just a few extra requirements. What we are trying to avoid is the complex implementation of new and costly hardware investments. Instead, the suggestion is to upgrade one or two workstations to integrate autonomy, co-operation, self-organization, and reconfigurability to the actual systems.

### 3.4 The general model of interaction

The suggestion is to concentrate this complexity in one processor, to improve data control transmission, likewise decreasing data communication errors. In addition, LabView bids the concept of object oriented programming, allowing one of the main characteristics of holonic systems, such as reconfigurability and easy expansion without redundant coding lines, the processing and communication model proposed on this work is illustrated on next figure (See fig. 3.2).
The model pretends to show how agents will interact, which are the main actions that have to be executed and by whom. As can be seen, management holons are strongly related with superior levels, its process comes directly from data processed by a manufacturing execution system, this layer is in charge of system-level management and services such as life-cycle services, collection services, and query services. Whereby, scheduler agent is comprehend at this layer and accepts orders from system manager (updated orders); conducts tasks, and dispatches jobs to equipment, material and labor managers (execution agents). Some temporary agents are created in order to accomplish some tasks; however those agents are unloaded once inspection or when quality agents releases the product manufactured [5].

![Interaction general model](image)

**Figure 3.2:** Interaction general model.

### 3.5 Resources and technology reachable

Since the multi-agent technology has been recognized as a key concept in building a new generation of highly distributed, intelligent, self-organizing and robust manufacturing control solution, the traditional concept of manufacturing flexible or adaptable, has became vulnerable to changes [40]. Adaptability to manufacturing volume and environmental changes, failure detection, reconfigurability, and expandability; are a set of capabilities that make an attractive option the idea to improve traditional FMSs. To achieve this project; the development of multi-agent system is based on an alternative commercial software platform which recently has been improved making possible the usage of OOP (object oriented programming), distributed intelligence processing, OPC data communication sockets, DLL interfacing, ActiveX function, and external code interfacing (See Tab. 3.1). The table presented shows few characteristics
### Table 3.1: This table shows the comparison between the programming languages; LabView and C++.

<table>
<thead>
<tr>
<th>Feature</th>
<th>LabView</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor/destructors functions</td>
<td>Don’t needed</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel processing</td>
<td>Yes</td>
<td>Line by Line</td>
</tr>
<tr>
<td>Ancestral class for every object</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Reference and parameters</td>
<td>Have just one syntax of value, and reference is inherent</td>
<td>Have both reference and parameters</td>
</tr>
<tr>
<td>Data mutation</td>
<td>Automatically, user could have access to old values, even though class has been edited</td>
<td>No, user has to track versions, and write extra mutation coding.</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Distributed coding</td>
<td>Yes, could be applied on one workstation.</td>
<td>No, it is possible to develop with some PC interconnected by a deterministic networking, for example.</td>
</tr>
</tbody>
</table>

that this alternative commercial software bids vs. C++ functions; moreover its data manipulation is under a friendly graphical environment.

Let’s take a general case of the project to explain how OOP would be implemented under this graphical environment. The Cell description, previously presented, refers to a belt-conveyor with three docking stations, even if there are three docking stations, it is essential to think on the most general application (a belt-conveyor with “n” docking stations).

To satisfy the general necessity we define methods, events, and characteristics that modify a general docking station class (See fig. 3.3). Applying this logical and to make the most of Inheritance, cell expansion not longer depends on how many docking stations are in a cell. Besides, the structure and coding organization could be sketched in certain way to represent the actual status of the cell. In the general case represented on figure 3.3, the tasks of methods are to take information about the actual statutes of docking station (Id of pallet traveling, stop order, release order, and time of pallet locked). Holonic Skills and knowledge should be attached to the inputs and outputs of these VI’s shown on figure 3.3, those entities could be developed on different coding resources, such as Matlab, C++ or JAVA, and this can be encapsulated as a external
code node in such a way that it can be used on LabView interface. Inspired on holonic models previously planned on earlier research projects, the design of intelligence holons shall be based on those works [40].

![Diagram of Object Oriented Programming mapped in LabView.](image)

**Figure 3.3:** Object Oriented Programming mapped in LabView.

Since the driving forces behind these studies are mainly oriented to develop a generic platform that bids the easy possibility to expand intelligence block functions in order to achieve migration from traditional FMS to IMS, we will not deal with complete holonic design in depth; however we shall present a single case to represent how holonic actions will be deployed.

The long established cells have solved production issues for years, however market is growing up so fast that conventional production lines, are close to be an option that suddenly will not satisfy market necessities; to mention some of those necessities we have: more complex products, high quality requirements, faster changing products, faster introduction of products, a volatile output, and the necessity of reduce investment. As we quoted previously on introductory part of this book, manufacturing control system should be based in distributed and autonomous entities, expandable [27], moreover being able to produce a family of products without large setup times. Each of these, are facts, that motivate migration and justify this schema, following chapters will discuss about methodology and its implementation, without losing the essential of traditional systems, keeping the generality for almost any kind of topologies and FMCs.

### 3.6 Holonic implementation example

As a general case we might mention different contingence cases that could be covered by artificial intelligence for example missing raw material at warehouse, the occupancy
of pallets or material transporting accessories, robot collisions, the disqualification of a sub-system because of failures or hardware problems, the lack of stability control in a sub-process or mechanism, and changes on production demand for several pieces or different products at the same time (See fig. 3.4).

**Figure 3.4:** Petri Net for holonic actions due to productions changes.

To have a brief description of the petri net presented on figure 3.4; the manufacturing execution system notify about changes on production to the central blocks, where multi-agent parallel processing takes place. At the same time this agents reprogram the schedule production to handle order task holons and supervise them in new production jobs. Task holons must have a record of any program needed on production to set them directly to executable holons once new changes have been established. It is vital to new concept of IMS, that current production do not result affected by those changes and the actual manufacturing process continues normally even when new production is being performed simultaneously.
Chapter 4

The Methodology for GP

Environmental changes, failure detection, reconfigurability, and expandability; are a set of capabilities that make an attractive option the application of migration feature. The facts about MAS architecture explained previously on chapter 3, seems to be the future of manufacture. Nevertheless it is imperative to evaluate the transition of actual mass production methods in order to become new and intelligent systems. Although researches and recent works about this topic have developed theoretical structures, with impressive benefits, they suggest a complete new equipment to perform in actual enterprises, making even more difficult the adoption of these new features. In this section a methodology shall be proposed, a methodology that would bring the opportunity to those elements of shop floor to become in new generation of manufacturing systems.

4.1 The feature of migration

The idea of a standard software platform including characteristics such as reconfigurability, flexibility and "holonic-ready" [21] concepts, is justified by the necessity of uptake on established systems, making easier to adopt new production infrastructures without dramatic hardware changes and long setup times. At the moment it is possible to find several topologies of manufacturing cells, such as centralized, hierarchical, and heterarchical structures (See fig. 4.1)[35]. Each topology could be considered as optimal and able to accept migration, taking into account that each block should be related without complete dependency, at least after migration is implemented, and well functioning shall not be compromised with any other element from the cell.

Summarizing migration features, a ordinary system must complete at least one of the following:

- flexible
- Reconfigurable
Chapter 4. The Methodology for GP

- Holonic-ready
- being part of a topology able to break apart
- Possibility of communication channels between elements

Figure 4.1: Topologies of manufacturing cells.

A generic platform was designed in order to apply multi-agent schema [21]. The platform design was implemented in such a way that any flexible manufacturing cell could be evolved into agent-based structure. The clue is to adopt the platform structure, and shape each element (robots, numeric control machinery, conveyors) of a cell to acquire agent personality. Once the problem or problems are identified the MAS design phase, starts, which is more oriented toward the implementation of the generic platform; however a methodology should be committed. The methodology includes definition of: 1st stage; capabilities of each single agent, and the inter-agent communication, 2nd stage MAS architecture planning.

4.2 The Methodology

Before any attempt can be made to implement agent societies effectively in a manufacturing system, an analysis of the industrial life cycle is pivotal. It therefore becomes important to introduce the environment in which an agent should act [31]. For it the information system of a manufacturing enterprise is crucial to be recognized, in order to clearly sketch how agents can be integrated and how data would be interchanged, as is depicted on figure 3.2 on chapter 3, wherein the three layers, that computer
systems in manufacturing management use, are illustrated.

The generic platform is toward from general to particular application, so before start working on developing intelligence, is crucial to make independent each element, which is supposed to emerge from a centralized and sequential architecture that actually shall be substituted by the new platform.

This section will start taking the hierarchy presented on figure 1.2 and model presented on figure 3.2, taken from earlier works on this research [13, 14, 21]. The superior part of this pyramid is performed by management layer, which are satisfied with a manufacturing planning level, and a manufacturing execution level. Both could be programmable holons, purely software based. In addition pyramid bottom is formed by executable holons, which has direct contact with machinery, and hardware systems, also this part of the pyramid frequently is the one with more constraints in manufacturing environments. The efforts on this section will be driven to get the pyramid base prepared to be adapted without neither hardware changes nor design, on the other hand ready to become reconfigurable, and holonic-ready [21], the methodology (See fig. 4.5) shall be explained in four steps, Define Communication Structure, Isolate from global system, Convert from general to particular, Create relationships but not dependences.

As is depicted on figure 4.5, each element from shop floor should be treated under same steps of methodology, Define Communication Structure, Isolate from global system, Convert from general to particular, Create relationships but not dependences, in order to accomplish transformation and acquiring Holonic Ready Agent (HRA) features.

An HRA, is a new concept brought with this research, HRA is an entity with characteristics and attributes necessaries to adopt intelligence, which could be contained in intelligent blocks software based (to become an agent). This extension could be oriented to achieve different functions or tasks.

To comprehend the subsequent subsections, is key to homogenize the overall view of a traditional FMS, as is quoted before, in chapter 2 a Flexible system covers the system’s ability to be changed to produce new product types, however this challenge comes with some limitations such as strict sequential systems, hard adoption of new elements or mechanism, in other words a general evolution, that actually is subject of a entire life of a product (See fig. 4.2). Figure 4.2 depicts how traditional systems use to perform before any migration methodology is applied.
4.2.1 Define Communication Structure

This answers the two important issues, how data acquisition will be performed and how data shall be shared between items on internal and distributed network, the last matter comprehends the interaction between future HRA blocks. On actual systems, this is an easy matter due the well-known kinds of communication protocols. Data interchange is available in several ways, for example ActiveX, data library functions (*.dll), OPC and data sockets, and those at the same time use different well established channels of communication, such Control-Net, Profibus, Ethernet, Device-Net, amongst others, making even easier this step on the methodology. An effortless data acquisition would depend only in how old fashioned equipment is, in other words this facility enclose elements relative modern PLCs, robots, vision systems, that actually have been functioning on traditional systems since 10 years ago, those elements are still useful, because they count with communication interfaces such the mentioned before.

Although it seems to be a relative simple selection of communication channels for data acquisitions, it is essential to have a plan for the interaction between HRA blocks, because will involve topics such as cooperation and coordination (See fig. 4.3).

Communication has long been recognized as a topic of central importance in computer science, and many formalisms have been developed for representing the properties of communicating concurrent systems [18, 32]. Such formalisms have tended to focus on a number of key issues that arise when dealing with systems that can interact with one another.
4.2. The Methodology

4.2.2 Isolate from global system

The well or bad functioning of one element should not compromise other elements. In other words, it is essential to dissolve dependences. However, isolation could be a difficult part on this methodology.

Each element in a traditional systems is related with a strong sequential logic, using strategies such as First-in-first-out (FIFO), Earliest Due Date (EDD), Shortest Processing Time (SPT), and Least Slack First (LSF), all of them have had an acceptable performance and their use have solved many industrial optimization problems, nevertheless they are sequence dependant, and sequential operations, are dependent, and rigid becoming in a structure that is not compatible with the ideal holonic infrastructure.

This stage of the methodology includes some important aspects from next stage, is essential for migration to scale system starting from a general part, and generality only can be achieved if each element of the cell is studied as a individual part, in other words isolated from global system.

Isolation should be achieved without interfering in a dramatic way with hardware changes, nor extra wiring, or unplugging materials from the manufacturing cell. Every detail has to be done separating from software structures; the best way to get this
done is using data parallelism. Data parallelism is a programming technique for splitting a large data set into smaller chunks that can be operated on in parallel. After the data has been processed, it is combined back into a single data set. With this technique, programmers can modify a process that typically would not be capable of utilizing multicore processing power, so that it can efficiently use all processing power available.

Task parallelism is the concurrent execution of independent tasks in software. Multiprocessor and multicore computer systems benefit from task parallelism by being able to execute two or more things at one time. Consider a single-core processor that is running a Web browser and a word-processing program at the same time. Although these applications are implemented with unique sets of threads, they still ultimately share the same processor. Now consider a second scenario in which the same two programs are running on a dual-core processor. On the dual-core machine, these two applications essentially can run independently of one another. Although they may share some resources that prevent them from running completely independently, the dual-core machine can handle the two parallel tasks more efficiently.

Multitasking is a more common term in computing that is related to task parallelism, where multitasking has a more broad definition of “multiple tasks running at the same time.” However, it’s important to note that multitasking does not necessarily imply true parallelism in software, it may just mean a computer is switching between tasks fast enough to give off the illusion that things are happening in parallel. On a multicore system, multitasking can occur in a truly parallel fashion. An application must be multithreaded so the system can execute the tasks in parallel. As an example, consider a measurement or control application that is composed of different tasks: Data Acquisition, User Interface, Network Communication, and Logging to Disk (See fig. 4.4).

\[\text{Figure 4.4: Multithreading example for isolation, from methodology 2st stage.}\]
4.2.3 Convert from general to particular

Once every element has been secluded by software from entire system, it is vital to keep generality in order to become those elements useful for a large amount of applications.

General and reusable software for language engineering is desirable for several reasons, first and foremost for shortening the way from idea to prototype and for reducing the effort needed in building language engineering systems utilizing existing components, thus allowing the system’s developer to focus on what is specific to the task at hand. When attempting to create such general and reusable language engineering software, there are a number of challenges that need to be tackled, varying along several dimensions such as the language and the domain to model, the characteristics of the task to be solved, and the type of users for which the system is intended [11].

While these issues seems to be reasonable and sound, it appears to be nontrivial to actually define and implement an open architecture that accommodate all the challenges they represent in all possible tasks such a system may face the challenges are simply too many ant too hard to deal with within the boundaries of one single system. A possible solution is to constrain the characteristics of the task to solve so that they form a set of related language engineering tasks that is small enough to facilitate the development of general and reusable software, yet large enough to justify the overhead that is involved in developing such software.

Scalability and generic features are the main topics on this methodology; we must conserve generality, thinking in advance to future hardware or software changes. Rigid and dedicated operation should be eliminated, to achieve different applications, making able to change its functions.

4.2.4 Create relationships but not dependences

Elements should be able at the end to share data, hence is necessary to establish a weak relation with messaging protocols such as FIPA or contract Net, that with fire actions in order to perform an application. It follows that relationships must be created without missing complete agent-based structure.

The result after this methodology would be what we call a ”holonic-ready agent” (HRA), which meaning contemplates an entity with characteristics and attributes necessaries to adopt intelligence blocks (to become a Holon) in order to achieve different functions or tasks.

An intelligent agent should be able to cooperate, coordinate, vote; in fact an agent could act as a selfish entity in order to optimize a specific task, all these behaviors
An example of relationship is a topic call *teams* is a language for representing large task hierarchies that contain complex constraints among tasks. You can think of it as a data structure for representing very complicated constraint optimization problems [28]. At its simplest, teams represents a goal hierarchy. As such, teams structures are roughly tree-shaped. The root is the top-level goals that we want the system to achieve. The children are the sub-goals that need to be achieved in order for the top goal to be achieved a form of divide and conquer. The leafs of the tree are either goals that can be achieved by a single agent or tasks that can be done by an agent. These goals and tasks, however, might require the use of some resources or data. teams represents these requirements with an arrow from the data or resource to the goal [41].

![Diagram](image_url)

**Figure 4.5:** The methodology implemented to achieve migration feature.
Chapter 5

The study case

For deployment of this methodology, the generic platform was used as an experimental case; however this schema was implemented in several study cases to demonstrate its viability for this application and to show the feasibility of developing software LabView (LV). The elements used to perform migration feature are completely real and common used on several traditional enterprises, elements such as Robots, CNC machines, conveyors, warehouses, vision systems, amongst others.

LabView developing platform was selected for its flexibility and well illustrated code features, such as block diagram functions and multiple usages on different branches of science. For this project is essential that the developing platform were able to perform algorithms of artificial intelligence, and modularity, because of it the initial part of this section shall explain the first application of this work that actually probes some results applying a complex algorithm of negotiation between agents based on Contract-Net protocol.

It is key to understand how elements use to perform before migration were implemented, in order to demonstrate how changes open new research lines and to emphasize the traditional environment in which actual mass production systems use to deploy.

In following sections a complete description of the main platform and its application at ITESM manufacturing cell shall be presented, including tele-operation case and RMS with intelligence migration, with a sample of application Robot and Camera interaction.

5.1 The ITESM manufacturing cell description

The laboratory installed at ITESM (Monterrey Institute of technology), consist of two identical cells equipped with one loop belt-conveyor, one robot (Motoman UP6), one ASRS (automatic storage retrieval system) installed in a warehouse of 2x12 storage
slots, a CNC machine (EMCO PC MILL 155), and an assembly table for each cell (See fig. 5.1).

The conveyors have three docking stations: robot, inspection and storage station. When a raw material is introduced by an operator production orders are delivered, so that each module is aware of their tasks and roles on production. When batches of raw material are deposited onto the belt-conveyor (Conveyor Holon), it must be aware at any time of which tasks are designated to each pallet that is navigating on the conveyor, and depending on the assigned task it can stop pallets at different docking stations in order to execute a process. When raw material is stopped at robot docking station, it could be delivered to CNC machine or assembly buffers, these tasks are performed by the manipulator (Motoman UP6). What to do and when has to be done, are examples of the information that order holons deal with the cell, more specifically to those executable holons in charge of that area or cell section.

Figure 5.1: Layout of ITESM Flexible Manufacturing Cell.

In spite of the flexible characteristics presented above, this structure is not compatible with the ideal holonic infrastructure, it is hard to adapt and centralized as well, any error condition stops production for a wide average of time. The habitual production concept will be presented to make clear how the cells used to perform manufacturing processes before the holonic ready-design. It is possible to define the dynamic and static operation of these cells in a graphical manner.

On one hand, we have the static operation, (See fig. 5.2); it just allows one product plan per operation cycle, wasting the most part of resources located at the cell and prolonging production. Even though, this is a good template to start the new holonic environment, it is still rigid in many aspects, such as the deployment of simultaneous production plans for the same product or different products.
5.2 Generic Platform design (GP)

In previous section a detailed description of the cell was presented, as well as its functionality before migration were implemented. In this section migration should be achieved using LabView platform due to its advantage against other development platforms, LabView bids some helpful characteristics such as parallelism, software distributed cycles, real time modules, and easy OPC communications, making a very simple task data acquisition, amongst other things.

On the other hand, dynamic operation is inflexible, even when each element installed on the cell has a lot of flexible functions, besides in any error occurrence whole production plan is stopped, without notifying the principal cause nor logging a record of it (See fig. 5.3).

All these aspects are constraints of established manufacturing cells. Taking up again the effort to develop a platform; we are aiming to create an easy way to adapt holonic and multi-agent concept, without crashing conventional hardware structures and trends, in order to avoid affecting in a representative way the infrastructure established.

![UML for Static operation standard cell representation.](image)

Figure 5.2: UML for Static operation standard cell representation.
Chapter 5. The study case

Figure 5.3: Dynamic operation of standard cell representation.

Table 5.1: This table shows different channels of communications for each element from the FMC.

<table>
<thead>
<tr>
<th>Element</th>
<th>Medium</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse</td>
<td>Datasockets RSLinx</td>
<td>OPC</td>
</tr>
<tr>
<td>Conveyor</td>
<td>Datasockets RSLinx</td>
<td>OPC</td>
</tr>
<tr>
<td>Vision</td>
<td>Library functions</td>
<td>ActiveX</td>
</tr>
<tr>
<td>Robot</td>
<td>Library functions</td>
<td>*.dll files</td>
</tr>
<tr>
<td>CNC</td>
<td>Datasockets RSLinx</td>
<td>OPC</td>
</tr>
</tbody>
</table>

The main objective is to apply each step of the methodology to migrate the FMC presented on figure 5.1 in order to renovate and shape it as a new Multiagent structure. Its generality would let fix the new system to become more flexible, and reconfigurable, adaptable to production changes or hardware evolution, being also possible the addition of new elements without complex software changes.

The first step of methodology says that we have to establish channel of communications, as it's quoted before, this should not be a problem due all the possibilities included on actual machinery. Next table 5.1 presents each communication assignment for case study.

An overall view of the resulting platform after methodology implementation (See fig. 5.4) emerges from figure 5.4, where is shown in a more oriented way the methodology applied on the commercial software used to develop the generic platform.
The methodology makes possible reconfigurability into the manufacturing cell, and at the same time the cell becomes ready to adopt second stage of the problem (refering to chapter 4, *The feature of migration*, lines 16-18), Multiagent architecture selection. An to explain how reconfigurability is done, the robot routine, which contains ethernet procedures and *.dll* functions to perform actions such as movements or execution of a specified routine. Lets imagine that we have to plug another identical robot to the cell, following this methodology procedure, it is just matter of duplicating robot cycle and change some kind of IP address to achieve plug and produce, like some authors have defined [39].

Holonic or intelligent agent skills and knowledge should be attached to the inputs and outputs of those isolated cycles, the intelligent entities could be developed on different coding resources, such as Matlab, C++ or JAVA, and these can be encapsulated as an external code node in such a way that they can be used on LabView interface.

![Figure 5.4: The distributed platform cycle design and message structure.](image)

Having as a result a highly reconfigurable platform, each of those elements software based, could be easily manipulated to achieve different tasks, including tasks for which the cell was not considered to develop, that characteristic is fundamental to promote, because even the cell was not build to adopt new integration of intelligence, or new hardware elements, or maybe radical production changes, the entire cell should be able to acquire reconfigurability with this new schema. For a detailed instruction for
Chapter 5. The study case

Figure 5.5: First screen of HMI prototype platform.

Figure 5.6: Second screen of HMI prototype platform.
5.3. Robot and camera agent

5.3.1 Agent Architecture for generic platform

Before continuing with the study case, it is fundamental to define some aspects about the agent architecture implemented, this refer to the second phase of migration problem. An agent could be categorized in purely reactive, in which do not consider historical data to react, and agents with state, this category contemplates past events, and contain internal states that describe the agent current situation, its perception of the world map to a set of possible actions to react in different manners. However these aspects still do not clarify how functioning might be implemented, functioning classes could be logical, reactive, intentional flexible (BDI) and layered, for this application case reactive agents is selected, in which decision making is implemented in some form of direct mapping from situation to action.
Chapter 5. The study case

For this implementation is necessary to create three different sets or vectors, in order to define the agent structure and its virtual environment. Equation 5.1 denote a representation of a set of discrete states $E$, which we can justify by pointing out that any continuous environment can be modeled by a discrete environment to any desired degree of accuracy, on the other side we have $Ac$ being a set of discrete actions. The basic model of agents interacting with their environments is represented on equation 5.2, as can be seen $r$ is a sequence with actions firing states, hence equations 5.3 and 5.4 are sequence terminated by actions or states respectively. The environment starts in some state, and the agent begins by choosing an action to perform on that state.

$$E = s_0, s_1, ..., s_u + 1; \text{ Ac } = \alpha_0, \alpha_1, ..., \alpha_u + 1;$$

$$r : s_0 \Rightarrow^{a_0} s_1 \Rightarrow^{a_1} s_2 \Rightarrow^{a_2} ... \Rightarrow^{a_u} s_u + 1$$

$$R^{Ac} : s_0 \Rightarrow^{a_0} s_1 \Rightarrow^{a_1} ... \Rightarrow^{a_u}$$

$$R^E : s_0 \Rightarrow^{a_0} s_1 \Rightarrow^{a_1} ... \Rightarrow^{a_u} s_u + 1$$

As a result of this action, the environment can respond with a number of possible states. However, only one state will actually result, and obviously the agent does not know in advance which it will be. The rules that govern environment are established by the state transformer, equation 5.5, at the same time each agent is defined by equation 5.6, in which an agent receives a run or sequence terminated by a state, an agent should map this situation to an action[43].

$$\tau(R^{Ac}) : R^{Ac} \Rightarrow \varphi(E)$$

$$Ag : R^E \Rightarrow Ac$$

Although architecture is designed by these abstract models, the following pseudo code, represent in a very general way how these models are implemented and the study case is developed:

1. function action(p:P):A
2. var fired:(R)
3. var selected:A
4. begin
5. fired:={((c,a)|((c,a)\not\in R and p\not\in c)}
6. for each (c,a)\not\in fired do
7. if !((c',a')\not\in fired such that (c',a')\not\in(c,a))
8. then return a
9. end-if
10. end-for
11. return null
12. end function action
Thus action selection begins by fires computing the set fired of all behaviors that fire (line 5). Then, each behavior \((c, a)\) that fires is checked, to determine whether there is some other higher priority behavior that fires. If not, then the action part of the behavior, \(a\), is returned as the selected action (line 8)\[42\].

### 5.3.2 Reconfigurability and agent implementation

Previous to this section, the ITESM manufacturing cell was described in detail in order to make a global view of how elements are initially set and how original operational flux is performed in a traditional environment. In this section the configuration of the cell will be altered physically in a non-dramatic way to ensure reconfigurability after implementing the "Holonic-ready" platform, also two elements with no previous interaction will have to cooperate in order to achieve a common goal. It is essential to avoid long setup times, extra physical wiring, or extra monetary investments, to demonstrate the feasibility of migration.

The study case begins with some physical changes, figure 5.1 shows the layout of the ITESM cell, for this application the camera from inspection station will be attached to robot assembly table, its image processing shall construct the perception of robot agent, in other words camera should be the medium that makes the robot able to observe its environment, whereas the robot agent’s performs decision making process (See fig. 5.8).

The tasks are defined as follows; raw material is delivered by an operator, and this material is formed by a pallet with geometrical figures, as shown in figure 5.8, these figures do not always conserve same patron of placing, thus the robot should perceive by the camera current state from environment, then the robot performs an specific routine or action to deliver each figure to another pallet with a specific location for each geometrical figure (See fig. 5.9).

The petri net demonstrate on figure 5.9, how actions and states modify environment from agent perspective, in a dynamic comportment. How often the robot agent performs a determinate route or path is established by the utility each path pays. The amount of utility given for each figure could be assigned by programmer. Nevertheless an agent always tries to maximize the utilities that it can obtain from a task \[41\], equation 5.7.

\[
A_{opt} = \arg\max_{Ag \in AG} \sum_{r \in R(Ag, Env)} u(r) P(r|Ag, Env)
\] (5.7)
Chapter 5. The study case

Figure 5.8: Multiagent abstract architecture for study case.

Figure 5.9: Dynamic study case representation.

5.3.3 Using LabView for intelligence adoption

LabView bids a useful toolkit for this application the state diagram, to have a more detailed view of this topic see appendix D. This instrument makes possible the literal drawing of state machine, which is very helpful for this kind of reactive intelligence on proposed study case. The following figure (See fig. 5.11), depicts the final structure for this study case, cycle number 1 have the role of perception giving information of actual states and environmental situations, the information captured on this stage is coming directly form care attached to robot, second cycle is the core of transformer states, here we deploy the state machine illustrated at the right top, finally number 3 is interaction with physical robot. Figure 5.12 is the final HMI for this study case camera and robot interaction.
5.4 Adopting teleoperation facility

Nowadays is fundamental to have an option of controlling and monitoring systems through Internet, at least to have this feature in a local network, justifying the fact or necessity of manage data, in order to manage information regarding production and/or hardware performance. Several industries concatenate all information in a centralized rooms with optical indicators, graphs screens, valves controls, etc. All this information let us to take critical decisions at the correct moment, in addition it is possible to record, in real time, all this information, which is also useful to manage production. Besides, if we take in consideration the schema of generic multi-agent platform, a feature like this could be integrated easily, working with real time information and taking decisions simultaneously in an autonomous manner, because all what agents need is correct information of current environment states.

The schema presented on figure 5.4, depicts at top cycle the teleoperation block. Currently, exist several ways of communication, for this study case TCP/IP protocol was chosen to reach teleoperation. The Internet protocol suite is the set of communications protocols that implement the protocol stack on which the Internet and most commercial networks run. It has also been referred to as the TCP/IP protocol suite, which is named after two of the most important protocols in it: the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which were also the first two networking protocols defined. The Internet Protocol suite like many protocol suites can be viewed as a set of layers. Each layer solves a set of problems involving the transmission of data, and provides a well-defined service to the upper layer protocols based on using services from some lower layers. Upper layers are logically closer to the user and deal with more abstract data, relying on lower layer protocols to translate data into forms that can eventually be physically transmitted.
The experimental implementation consist in running a server attached to the main FMC (See fig. 5.13), which would be the one to interact directly with generic platform described on Generic platform design section, all information collected is sent via ethernet to another server, which will carry data from LabView to a web service, that could be accessed by internet webpage. This application should be launched from a web page with scheduler manager access based on SQL [30], being possible the manage users and appointments to guaranty connection.
5.4. Adopting teleoperation facility

Figure 5.13: Teleoperation study case schema.

Figure 5.14: Access Webpage for prototype.
5.5 Contract Net protocol

5.5.1 Labview for AI

Artificial Intelligence (AI), is considered as the art of creating machines that perform functions that require intelligence, with capabilities of problem solutions that involves cooperation, negotiation, teams, voting, communication, selfish behavior, and every theme with the intelligent characteristic of maximizing utilities when the agent performs a task.

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds. There are two ways to do this: through introspection trying to catch our own thoughts as they go by, and through psychological experiments. Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program’s input/output and timing behaviors match corresponding human behaviors, that is evidence that some of the program’s mechanisms could also be operating in humans [38]. The fact is to concern not just in problem solution in contrast to worry more with comparing the trace of its reasoning steps to trace of human subjects solving the same problems.

Rational problem solutions engage cognitive behavior. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the working of the human mind [38]. An example of this could be clearly explained as follows, imagine there are two agents, obviously software based, that by reinforcement learning such as Q-matrix solutions, have concluded that the closest and well qualified "X"-elements are the ones that comes up with more utilities, therefore if they have to chose between a set of "X"-elements, both would like to have the same element, this situation becomes unstable and even more complicated when we increase number of agents on the game. For that we have some algorithms well established
such as *contract-net protocol*. Furthermore the platform should admit adoption of several protocols and standardization for using this kind of software based knowledge, prepared in advance for new algorithms and methods.

### 5.5.2 Contract-Net using LabView and Netlogo

Netlogo is a multi-agent programming language and integrated modeling environment. NetLogo was designed in the spirit of the Logo programming language to be ”low threshold and no ceiling,” that is to enable easy entry by novices and yet meet the needs of high powered users. The NetLogo environment enables exploration of emergent phenomena. It comes with an extensive models library including models in a variety of domains such as economics, biology, physics, chemistry, psychology and many other natural and social sciences. Beyond exploration, NetLogo enables the quick and easy authoring of models.

It is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of independent ”agents” all operating concurrently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals.

NetLogo has many thousands of active users. It is freely available from here. NetLogo is in use in a wide variety of educational contexts from elementary school to graduate school. Many teachers make use of NetLogo in their curricula. NetLogo comes bundled with a large library of sample models covering many domains in natural and social science.

NetLogo was designed and authored by Uri Wilensky, director of Northwestern University’s Center for Connected Learning and Computer-Based Modeling. Development has been funded by the National Science Foundation and other foundations.

The case study is expected to contemplate production of four different kinds of products (See fig. 5.16), this attribute represent those family of products that in real world could be deploy in production while demands is changing at the same time. The mass production line shall be represented in a graphical environment to show user the disposition of production and the way manufacture elements are installed, in actual enterprises exist several topologies. As first place on this solution, user must consider the interaction between both platforms, (LabView and Netlogo), the next schema (See fig. 5.17) represents how this applications would be playing their roles to achieve the correct analysis of the entire system.

Netlogo will generate a file that should contain info about the corresponded scenario, previously created by Netlogo. At the same time LabView is waiting for file
information to fill a data table, in order to perform contract net protocol to establish the correct pairs, once this pairs are created, LabView should enclose information to send it back to Netlogo in a text file. At this point of the process Netlogo would compile all necessary data to compare between three different protocols; CNet vs. FIIFO vs. Random, displaying times of victory of each protocol. The results are given by NetLogo, nevertheless Contract-Net protocol is performed by LabView, following graphs are samples of several runs, and those results demonstrate that contract-net has a better rational solution, which achieves the closest and well qualified element (See fig. 5.18).

In one hand we could succeed to implement the contract net protocol, is difficult to see an important deference with the first 10 iterations, however it is important to emphasize that contract-net is taking into account selections of better qualified machinery and the closest ones, on graph 5.19 includes much more iterations, with contract-net and random comparison, demonstrating a better performing by contract Net over
traditional protocols. On the other hand, we demonstrate that LabView could handle in a satisfactory way, implementations of intelligence and communication architectures.
Figure 5.19: Experiments for Contract-Net with Several iterations.
Chapter 6
Guidelines for future works

This section shall include important tips to consider before adding code for new developments, is recommended that the reader takes a look to appendices before continuing, because technical terms will be assumed in order to focus on new expansion.

As a first recommendation we can find common suggestions, before creating any kind of software is better to maintain clear and well organized, be sure of defining new variables without losing track of their existence. LabView uses the wiring technique for programming, which could be very useful, however it could transform code in a mess of wires, so is important to keep straight lines without overlapping them, this will help to always be sure of what each element does and to which elements is connected. Other useful aspect is, when you are coding always think in general, start programming with generality, will let you reuse code and saving a lot of time, LabView have the option to encapsulate code in order to generate small blocks, similar to subroutines.

6.1 Adding new elements

In order to keep flexibility is essential to maintain actual structure, as can be seen from entirely source code; each element is isolated with different cycles and controlled by a variable time of setup, those elements results from methodology, explained on chapter 4. Programmer is able to add new components, assuming that each element has been shaped with the methodology, and is supposed to have its own new timed loop cycle somewhere around the entire code, inside second phase of sequence (See fig. 6.1).

The overall view of entirely code, presented on figure 6.1, besides presenting the addition of new elements, shows as well how the structure is performed in order to keep flexibility.
Figure 6.1: Prototype structure for adding new elements.
6.2 Intelligence implementation

Before running intelligent implementation, programmer should consider some facts of agent architecture, intelligent block should interact with inputs and outputs of the element which is becoming in agent, before creating an intelligent agent, programmer should consider how agent will have perceptions of the media, how this agent will interact with environment, user as well has to define different environment states.

Intelligence could be developed in several ways depending on how complicated should be the algorithm it could be created as a state machine, or an external code node which involves using matlab, java, C++, amongst others, both methods could be attached exactly the same as previous section, sometimes programmer could use the same cycle of the element, which is becoming intelligent or create a parallel cycle that interacts with the element cycle in transformation. The fact is that either one option or the other, is essential to evaluate the algorithms before adding them to entire system.

If an extra timed cycle loop is selected, the only issue that has to be kept on mind is timing, which means period of processing information.

6.3 Upgrading Teleoperation

First we present definition of upgrading teleoperation, for this is assumed that programmer wants to add some extra information into teleoperation schema, such as new buttons, indicators, etc. for example, a new robot is been installed at main cell, and size of information such as controls and indicators have increased and are not considered in previous version. Till now TCP/IP just mange closed package information that does not include new terminals. The solution of this issue, includes some manipulation that comprise refreshing information in basically 8 clusters but only in 2 of them the containing is modified and then copies will replace the 6 originals left.

All the information prepared to be sent through internet, clearly quoted on corresponding appendices, is encrypted in a LabView element called Cluster, a cluster is analogous to a struct in C programming, combines one or more data types into one data type. The cluster may contain different data types such as boolean, string and integer. The wire on the block diagram that carries data from a cluster can be thought of as a bundle of smaller wires like a telephone cable. This reduces clutter on the bock diagram. So, for previous versions of teleoperation system, the amount of data encapsulated in a cluster is fixed, if some extra information such as controls or indicators are created, they must be part of the cluster to be sent, and for clusters that are received.
Let's mention which would be the 8 clusters previously mentioned, first programmer must identify which cluster send information server to client, such as alarms, ends of operation, names of files, etc. we take as information to be sent all answers from the physical system, on the other hand we have cluster that receives information such as clicking of a button, file selection by user, orders, etc. everything that has to be with the final user, that is supposed to try getting control of the cell by internet. Once each cluster is identified new elements have to be added in clusters, the refresh of this new connections, will crash with error connections whole system, but it is just matter of open the subVi’s in charge of handling clusters to TCP protocol and replace inside old clusters for new ones (See fig. 6.2).

Figure 6.2: Teleoperation technical structure for clustering.
In figure 6.2, is depicted a technical structure for teleoperation, is shown how server and client interchange package of data by passing clusters with different kind of information, note that for server stage C1 and C2 are connected via subVi to CA and CB, as well as C3 and C4 are connected to CC and CD at client stage.

• 1.- Modify on server clusters C1 and C2, with new information regarding new components to teleoperate.

• 2.- Once C1 and C2 have been modified, the wires to CA and CB should not be compatible unless programmer gets into the corresponding SubVi and replace CA for a copy of C1 and CB for a copy of C2, refresh and wires will be reestablished.

• 3.- For client C3 must be a copy of C2 from server, because everything that is received by server is supposed to be sent by client and vice versa for C4 and C1.

• 4.- CC is a copy of C3, and CD is a copy of C4.
Chapter 7

Conclusion

7.1 Conclusions and future works

This prototype reveals that the usage of a graphical environment that actually bids common functions of coding that could be an option to displace traditional manufacturing process and encourage the holonic-ready concept. This work is geared towards direct applicability in real production line, also we present approaches to developing a methodology for construction of holonic systems, starting from a conventional FMC without making representative changes on infrastructure, to set the trend of a new and easier holonic execution. Furthermore, the applicability of a MES (Manufacturing Execution System) is easier due to distributed intelligence facilities to make the most of the data concentration (timing, production, failures, and quality) in a flexible system. The work on this development will continue to improve the design and performance of multi-agents layers; moreover the presented approach opens several research perspectives.

The physical changes were successfully achieved, there are several ways to qualify this characteristic such as time and hardware adaptation, even both aspects were optimized with the usage of the holonic-ready platform, if there were dimension changes on assembly table for example, collision of work space would be also an important problem to solve, nevertheless collision could be avoided adding some extra collision avoidance algorithms, it always will depend on how old-fashioned integrated systems are in the FMC to migrate and its ability to interact. In other words solutions for different elements, will depend on how flexible or communicable they are, as a result we could have several solutions. However preparation of a generic platform that actually could adopt different solutions seems the most urgent issue.

The platform shows sufficient flexibility to accomplish the unexpected request of assembling products, as well as showing flexibility in removal, addition and reconfiguration of assembly devices.

We could succeed to implement a holonic-ready platform in a generic mode showing
its capability for migration to convert common FMSs into RMS agent-based systems.

As future work a scheduler of assembly devices shall be developed, and interaction with superior levels such manufacture execution system, both have to be developed in a generic schema to be adopted on the platform, opening different research lines such as logistics and planning for intelligent manufacture, and technological migration.
Bibliography


[38] Stuart J. Russell, Peter Norving, Artificial Intelligence, a modern approach, Second edition, Prentice Hall, pp. 3-20, 2004


Appendix A

Robot Motoman UP6

Almost every robotic system is equipped with a dynamic library function, this possibility makes a great option because exist on different robotic companies such as Yasnak, ABB, PUMA, etc. Now days manufacturing enterprises are controlled with a large amount of robotic mechanisms, however it always would be a better option to monitor, control, and reprogram them by HMI.

Following steps are would give a complete idea of how robot cycle perform on actual platform, at the same time reader shall be ready for new robots implementations.

Figure A.1: Robotic technical implementation structure.

A.1 Using Visual C++ 6.0

routines, in this case Motocom32.dll functions, which has a large amount of subroutines to interact with physical robot.
First is user should select if the project will be created on C++, which data type is defined as *.cpp, or C with definition type of *.c. Difference between them is on which kind of data, elements, or operations are needed for the development. C++ has different options for using files for example, in contrast C is a lower level of code so there exist a more unrestricted path for developments, but at the end either both could be helpful. If user selects a C++ project the target would be create a new *.dll function, on the other hand if user selects a C project the goal would be create *.lsb file, both are usable on LabView platform, however they use different interface node.

Figure A.2: NI external code nodes.

Figure A.3: Robot source code and schema of work.
A.2 Creating *.dll from *.cpp

The following steps help you to create a *.dll starting from a *.cpp file (C++), this new *.dll will work as a bridge between LabView and Motocom32.dll, to use this new file launch in LabView the block named Call Library Function Node:

1.- In Visual C++ 6.0, select File / New / Projects / Win32 DLL. Give it a name, choose a location, select CREATE NEW WORKSPACE, and then OK. On the next page, select CREATE EMPTY DLL. (this creates a folder with a skeleton project in it).

2.- In LabVIEW, create a VI with the inputs and outputs corresponding to the DLL’s inputs and outputs. Put a CALL LIBRARY function on the diagram.

3.- Pop up on the CALL LIBRARY node and select CONFIGURE to get a dialog box. Leave the LIBRARY NAME field blank for now. Enter a function name and select C as the calling convention.

4.- Change the words "return type" to "error", or some descriptive name. Set the TYPE selector to NUMERIC, and the DATA TYPE selector to SIGNED 32-BIT INTEGER. This establishes the return type of your C function (in the DLL).

5.- Add parameters, set their TYPES and DATA TYPES and pass BY VALUE or BY REFERENCE as needed. For XX, I chose: errorCode/Numeric/Signed 32-bit integer, MachineName/String/StringHandle, ProgramName/String/StringHandle, ErrorMessage/String/StringHandle, ChannelTags/AdaptToType/HandlesByValue.

6.- Check the FUNCTION PROTOTYPE field to verify it looks correct.

7.- Click OK, and the CALL LIBRARY node changes to match your settings.

8.- Wire the inputs and outputs to the CALL LIBRARY node.

9.- Save the VI in the project directory.

10.- Pop up on the CALL LIBRARY node, and choose CREATE C FILE. Change the extension to .CPP if needed, and save the file in the project directory. Quit LabVIEW if you want.

11.- Open the DSW file in Visual C++.

12.- Select PROJECT - ADD FILES - and add the C (or CPP file) you just created.

13.- Select PROJECT - ADD FILES - and add LABVIEW.LIB (from the LabVIEW CINTOOLS folder), if you need manager functions.
14.- Choose PROJECT - SETTINGS - ALL CONFIGURATIONS - C++ - PREPROCESSOR and enter the full path to the CINTOOLS directory in the ADDITIONAL INCLUDE DIRECTORIES field.

15.- Choose PROJECT - SETTINGS - ALL CONFIGURATIONS - CODE GEN and select STRUCT ALIGNMENT = 1 BYTE.

16.- Choose PROJECT - SETTINGS - DEBUG CONFIGURATION - choose DEBUG MULTITHREADED DLL in the USE RUNTIME LIBRARY selector. If you use DCOM (OLE between machines), enter the following after the last /D entry in PROJECT OPTIONS:

```bash
/D "_WIN32_DCOM" |(not including the | marks)
```

17.- Choose PROJECT - SETTINGS - RELEASE CONFIGURATION - choose MULTITHREADED DLL in the USE RUNTIME LIBRARY selector. If you use DCOM (OLE between machines), enter the following after the last /D entry in PROJECT OPTIONS:

```bash
/D "_WIN32_DCOM" |(not including the | marks)
```

18.- Open the CPP file in Visual C++. If you're using C++, put the text

```c
extern "C" { | just before the declaration of your function, and the text |} | immediately after it. (not including the | marks)
```

19.- Put the text

```c
__declspec(dllexport) | immediately before both the declaration and the definition of the function. (not including the | marks)
```

20.- Erase the line YOUR CODE GOES HERE, and enter your code.

21.- Build the DLL. Set the BUILD - ACTIVE CONFIGURATION to RELEASE and use the DLL in the RELEASE folder once all bugs are fixed.

22.- Open the VI again in LabVIEW and pop up on the CALL LIBRARY node to select CONFIGURE. Enter the file name of the DLL in the LIBRARY NAME field,
or BROWSE to the version in the DEBUG (release) folder.

23.- Run the VI.

24.- The DLL is loaded when the VI is loaded, and unloaded when the VI is unloaded. If you go back to C to make revisions, close the VI before rebuilding, or you’ll get a CAN’T WRITE TO XX.DLL FILE error.

A.3 Creating *.lsb from *.c

To build CINs using the Visual C++ Integrated Development Environment, complete the following steps:

1.- Create a new DLL project. Select File/New and select Win32 Dynamic-Link Library as the project type. You can name your project whatever you like.

2.- Add CIN objects and libraries to the project. Select Project/Add To Project"Files and select cin.obj, labview.lib, lvsb.lib, and lvsbmain.def from the Cintools \Win32 subdirectory. These files are needed to build a CIN.

3.- Add Cintools to the include path. Select Project/Settings and change Settings for: to All Configurations. Select the C/C++ tab and set the category to Preprocessor. Add the path to your cintools directory in the Additional include directories: field.

4.- Set alignment to 1 byte. Select Project/Settings and change Settings For: to All Configurations. Select the C/C++ tab and set the category to Code Generation. Choose 1 Byte from the Struct member alignment: tab.

5.- Choose run-time library. Select Project/Settings and change Settings for: to All Configurations. Select the C/C++ tab and set the category to Code Generation. Choose Multithreaded DLL from the Use run-time library: tab.

6.- Make a custom build command to run lvsbutil. Select Project/Settings and change Settings for: to All configurations. Select the Custom Build tab and change the Build commands field to

<your path to cintools>\win32\lvsbutil $(TargetName) -d $(WkspDir)$(OutDir) and the Output file fields to $(OutDir)$(TargetName).lsb

.
Appendix B

OPC and OOP with LabView

OLE for Process Control (OPC) which stands for Object-Linking and Embedding (OLE) for Process Control, is the original name for an open standards specification developed in 1996 by an industrial automation industry task force. The standard specifies the communication of real-time plant data between control devices from different manufacturers.

After the initial release, the OPC Foundation was created to maintain the standard. Since then, standards have been added and names have been changed. Currently (June, 2006), "OPC is a series of standards specifications". (Seven current standards and two emerging standards.) "The first standard (originally called simply the OPC Specification") is "now called the Data Access Specification", or (later on the same page) "OPC Data Access", or OPC Data Access Specification.

OPC servers provide a method for many different software packages to access data from a process control device, such as a PLC or DCS. Traditionally, any time a package needed access to data from a device, a custom interface, or driver, had to be written. The purpose of OPC is to define a common interface that is written once and then reused by any business, SCADA, HMI, or custom software packages.

Once an OPC server is written for a particular device, it can be reused by any application that is able to act as an OPC client. OPC servers use Microsoft’s OLE technology (also known as the Component Object Model, or COM) to communicate with clients. COM technology permits a standard for real-time information exchange between software applications and process hardware to be defined.

LabVIEW itself is not an OPC Client, but it can be used to communicate to OPC Servers through the DataSocket VIs. DataSockets have an OPC layer, allowing you to read and write to an OPC Server from LabVIEW (or LabWindows/CVI).

DataSocket is a technology based on TCP/IP composed of two elements: the DataSocket API and the DataSocket server. The API provides an interface for
different data types, converting the data into a stream of bytes that can be sent across a network via TCP/IP (See fig. B.1).

![Diagram of OPC communication structure in platform.](image)

**Figure B.1:** OPC communication structure in platform.

### B.1 Connecting LabVIEW to RSLinx

The National Instruments Industrial Automation Servers CD has an Allen Bradley (RSLinx) driver that can communicate with RSLinx through a dynamic link library (DLL). With the release of Industrial Automation Server CD 1.1, this server became an OPC server. Any OPC client, such as Lookout, BridgeVIEW, or other packages on the market, can connect to this server. Rockwell Automation developed their own OPC Server in version 2.0 of RSLinx. This version has been upgraded to version 2.20.

Rockwell Automation RSLinx 2.10 is an OPC server, meaning that any OPC Client should be able to exchange data with this server. The OPC server is available only with the OEM version or the full version, so the lite version of RSLinx does not have the OPC Server included. RSLinx OPC server is an in-process type of server, which means that you have to load a DLL to communicate with it. RSLinx 2.0 to 2.20 OPC servers only support the browsing capability on Control Logix 5000 series PLCs. Other Allen Bradley PLCs can be used on the OPC server by manually adding the addresses to be seen. For this application there are three PLC interacting, which means that generality is ensured with this utility, to have a clear overview of how LabView communicates with the CNC, Assembly table, Conveyor, Warehouse (See
Another mechanism of communication between applications and RSLinx is through the DDE server that comes with RSLinx. The DDE server is only available for Rockwell Automation DDE clients in the OEM version. The drawback of DDE connection is that it is slower than OPC. If you decide to use the LabVIEW DDE VIs, then you need to have the full version of RSLinx.

B.1.1 Configuring an Allen Bradley PLC in RSLinx

1.- Start RS Linx and click on Communications-Configure Drivers.

2.- The Configure Drivers dialog box appears. Click on the Available Driver Types pull-down menu and choose your communication driver from the list. For instance, if you have the regular serial connection, choose RS-232 DF1 devices from the list.

3.- Click Add New.

2.- Enter any name you want for the driver name in the Add New RSLinx Driver dialog box. Click OK when you are done.
Appendix B. OPC and OOP with LabView

Figure B.3: OPC instruction 1.

4.- Configure the device. Click the Auto-Configure button and RSLinx will auto-configure the device for you. If the auto-configure failed, contact Allen Bradley technical support for help. Click Help if you need more information about device configuration. Click OK to go to the next step.

Figure B.4: OPC instruction 2.

6.- Now you should be able to see the driver that you just created in the Configured Drivers section of the Configure Drivers dialog box. Make sure there is no error showing and that the Status indicates Running. Click Close. Now you should be able to see your device when browsing the tree.

7.- If you do not see your PLC in the list, uncheck the Autobrowse checkbox and click Refresh. When you highlight the items in the window, you should see the little squares in the communication icon blinking, meaning that there is a communication link between the PC and the PLC.

8.- Right-click your PLC icon and click Configure New DDE/OPC Topic.

9.- A new access path is added for your server, so when you define the access path
in your SCADA software, you have to use the following syntax: [Access Path]Item. You can enter any name you want for the access path. In this example the access path is ML1500.

10.- Click Done after you are finished, and click Yes when RS Linx prompts you to update the topic.

Now you should be able to access all the available registers on your PLC from any OPC client software. With the access path ML1500, the syntax to access the register N7:0 would be [ML1500]N7:0. Next, test the OPC server just created using RS OPC Test Client.

11.- Launch the OPC Test Client from Start-Programs-Rockwell Software-OPC Tools-OPC Test Client. Then select File-New.
12.- Select RSLinx OPC Server and click OK.

13.- At this point you should see a blank screen called OPC Test Client - [RSLinx OPC Server]. Select Group-Add Group.

14.- Enter anything for the group name.

15.- Click OK. Now you should be able to see the group in the previous window.
16.- Select Item-Add Item.

17.- For the access path, enter the topic name you defined previously and type the register name you are trying to get data from or writing to. Click the Add Item button and the item name should appear in the left-most field, as shown in the illustration. Add all the items you want, and click OK when you are done. You should see the items and value that you just connect to under ItemID.

B.1.2 OPC in LabView Example

The following is a brief example of how to connect to RSLinx OPC Server from LabVIEW using our example VI.
1.- Launch LabVIEW and open opc.llb in the ...
\National Instruments\LabVIEW 6\examples\comm
folder.

2.- Open NI Demo OPC Client.vi.

3.- This VI can connect to any OPC Server using DataSocket connection. Next, connect it to RSlinx OPC Server. Set the Server Name to be RSlinx OPC Server, and the Item Name in the form [Access Path]Item. In this example we are trying to connect to register O0:0 using access path ML1500 defined in RSlinx, therefore the item name would be [ML1500]O0:0.

4.- Click the run button when you are ready.

You should see the register data that you just connected to.

![Image](image.png)

**Figure B.14:** OPC instruction 12.

### B.2 Object Oriented Programming

The idea behind object-oriented programming is that a computer program may be seen as comprising a collection of individual units, or objects, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions, or simply as a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine or actor with a distinct role or responsibility.

Object-oriented programming is claimed to promote greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering. Furthermore, proponents of OOP claim that OOP is easier to learn for those new to
computer programming than previous approaches, and that the OOP approach is often simpler to develop and to maintain, lending itself to more direct analysis, coding, and understanding of complex situations and procedures than other programming methods.

The benefits of working with OOP makes able the generality of instance such as number of docking stations, for the specific case of FMC, for this work a conveyor with three docking stations is taken (RB station, VIS station, WH station), however in order to achieve generality as is depicted on the methodology proposed, is essential to use some OOP to assume same diagrams of code for almost any size of conveyors. For the case study object oriented programming is represented in figure B.15, inside those blocks, as is depicted on figure B.15, keep the same communication methodology OPC.

**Figure B.15:** Implementation of OOP in platform.
Appendix C

ActiveX for Vision Systems

You can use LabVIEW as an ActiveX client to access the objects, properties, methods, and events associated with other ActiveX-enabled applications. LabVIEW also can act as an ActiveX server, so other applications can access LabVIEW objects, properties, and methods.

ActiveX-enabled applications include objects that have exposed properties and methods that other applications can access. Objects can be visible to the users, such as buttons, windows, pictures, documents, and dialog boxes, or invisible to the user, such as application objects. You access an application by accessing an object associated with that application and setting a property or invoking a method of that object.

Events are the actions taken on an object, such as clicking a mouse, pressing a key, or receiving notifications about things such as running out of memory or tasks completing. Whenever these actions occur to the object, the object sends an event to alert the ActiveX container along with the event-specific data. The ActiveX object defines the events available for an object.

C.1 Accessing an ActiveX-Enabled Application

To access an ActiveX-enabled application, use the automation refnum control on the block diagram to create a reference to an application. Wire the control to the Automation Open function, which opens the calling application. Use the Property Node to select and access the properties associated with the object. Use the Invoke Node to select and access the methods associated with the object. Close the reference to the object using the Close Reference function. Closing the reference removes the object from memory.

For example, you can build a VI that opens Microsoft Excel so it appears on the user’s screen, creates a workbook, creates a spreadsheet, creates a table in LabVIEW,
and writes that table to the Excel spreadsheet.

To insert an ActiveX object on the front panel, right-click the ActiveX container, select Insert ActiveX Object from the shortcut menu, and select the ActiveX control or document you want to insert. You can set the properties for an ActiveX object using the ActiveX Property Browser or property pages, or you can set the properties programmatically using the Property Node.

Use the Invoke Node to invoke the methods associated with the object.

For example, you can display a Web page on the front panel by using an ActiveX container to access the Microsoft Web Browser control, selecting the Navigate class of methods, selecting the URL method, and specifying the URL.

If you use the ActiveX container, you do not have to wire the automation refnum control on the block diagram to the Automation Open function or close the reference to the object using the Close Reference function. You can wire directly to the Invoke Node or Property Node because the ActiveX container embeds the calling application in LabVIEW. However, if the ActiveX container includes properties or methods that return other automation refnums, you must close these additional references.

C.2 Structure of connection, Vision system

The structure presented on figure C.1, shows how interaction is implemented on platform. Almost every complex machinery at actual enterprises have the option of operation through active-X, as was quoted before, to use active-x functions in LabView is just matter of refreshing methods and invoke nodes corresponding to the new activeX functions, this is achieved only by plug-in the refnum terminal into a invoke node.

Figure C.1: Vision System structure implemented.
Appendix D

State machines, an option for reactive intelligence

Taken from contribution of Jeanne Sullivan F. [19], the LabVIEW State Diagram Toolkit can be used to design and implement state machines within the LabVIEW development environment. The toolkit includes a State Diagram Editor so the user can draw the logic that defines the state machine. As this visual representation of the logic is created, the State Diagram Editor generates the LabVIEW code that executes the state machine. The logic is represented in code by a series of graphical while loops and case statements. The generated LabVIEW code can run on multiple operating systems and multiple real-time targets including NI programmable automation controller (PAC) platforms such as PXI/cPCI, CompactRIO, and Compact FieldPoint.

D.1 LabView State Diagram Toolkit

LabView is both a graphical programming language and development environment created by National Instruments (NI). It has been used extensively for automation and test applications on the factory floor. The LabView State Diagram Toolkit is an add-on toolkit that can be used to design and implement state machines within the LabView development environment. When a State Diagram structure is placed on the LabView block diagram, LabView automatically opens the State Diagram Editor. This is a tool for drawing state machine diagrams quickly and easily.

The editor includes a graphical user interface for the addition of new states and transitions. The states can be dragged by the user and placed in any location on the diagram. The transitions arrows can be dragged and shaped.

As states and transitions are added to the state machine diagram, LabView generates code to match each change. The logic for the state machine is represented in the LabView code by a series of graphical while loops and case statements. The toolkit also inserts comments in the LabView code to indicate where additional code
can be added. This additional code can be used to define the behavior of the states and the conditions for the transitions. The type of state is indicated by the color of the oval. The initial state is colored green. States can be configured as terminal states and are then colored red. Intermediate states are yellow.

Once the state machine has been described in the editor, the LabView code is ready to receive its state and transition logic from the developer. At any time, the application can easily be modified by reopening the State Diagram Editor and making changes to the state diagram. Consequently, the generated LabView code is also updated. Parts of the generated code that are tied to the editor are initially locked to prevent changes. The user can choose to unlock these parts of the code from the editor if desired. Once the code is unlocked from the editor, however, the editor cannot be used to edit it.

A useful feature for debugging the operation of the state machine is the ”highlight execution” option in the State Diagram Editor. The execution of the state machine can be visualized directly within the editor using this feature. The LabView code also has execution highlighting.

The user interface for the overall application can be created on the front panel of the LabView program. LabView does not require any programming to create a customized user interface. The user simply selects among various predefined graphical user interface (GUI) objects for program inputs (controls) and outputs (indicators). Controls include sliders, knobs, dials, and push buttons. Indicators include fill bars, gauges, LEDs, and graphs.
Appendix E

Glosary

E.1 Glosary of common terms

AI— Artificial Intelligence  
AM— Agile Manufacture  
AMS— Agent Manufacturing Systems  
ASRS— Automatic Storage Retrieval System  
BDI— Believes Desires and Intentions  
CIM— Computer Integrated Manufacturing  
CNET— Contract Net Protocol  
EDD— Earliest Due Date  
FIFO— First Input First Output  
FMC— Flexible Manufacturing Cell  
FMS— Flexible Manufacturing System  
GP— Generic Platform  
HMI— Human Machine Interface  
HMS— Holonic Manufacturing System  
HRA— Holonic Ready Agent  
IMS— Intelligence Manufacturing System  
ITESM— Instituto Tecnológico de Estudios Superiores de Monterrey  
LSF— Least Stack First  
LV— LabView  
MAS— Multiagent System Architecture  
MES— Manufacturing Execution System  
OOP— Object Oriented Programming  
OPC— Ole for Process Control  
RMS— Reconfigurable Manufacturing System  
SPT— Shortest Processing Time
Appendix F

Publications

F.1 A Generic multi-agent architecture design in a FMC

Abstract— The main propose of this project is to design a multi-agent system platform on a Flexible Manufacturing Cell (FMC). Taking into consideration all levels of functional issues, cellular architecture, standardization, and technical configurations, we take advantage of features released on a graphical programming commercial software, such as object oriented programming, distributed intelligence, and block function representation. The approach is addressed to encourage the usage of holonic and multi-agent concepts in traditional manufacturing systems, with friendly software based upgrade and a minimum low cost hardware expansion. LabView programming platform is selected for this development due to its easy applicability on holonic structures, where the structure is the programming code itself. The intention is to develop a system that will fit into different levels of an enterprise, including shop-floor and manufacturing execution systems, and simultaneously we will deal with technological migration from conventional manufacturing structure to a holonic-ready FMC.


F.2 Intelligent Manufacturing Systems: a methodology for

Abstract—More complex products in efficiency and quality, the necessity of diminish energy spending and investment reduction; amongst other things, are
disturbances, and their occurrence may have severe impact in the performance of actual manufacturing systems. Manufacturing systems should be based in distributed and autonomous entities, being possible the addition of new components without stopping or re-starting processes. All these facilities point to the concept of agile manufacturing systems. The approach is addressed to encourage the usage of holonic and multi-agent concepts in traditional production lines, with a friendly software upgrade and a minimum cost in hardware expansion. A methodology that includes the technological migration from a established flexible manufacturing structure (FMS) to intelligent and reconfigurable manufacturing system (RMS) is presented. An example of implementation will be described in depth to show the viability of the proposed schema.

A Generic multi-agent architecture design in a FMC, implementing distributed intelligence

J.M. Gamboa, Member IEEE, and M. J. Ramirez, Member IEEE

Abstract—The main propose of this project is to design a multi-agent system platform on a Flexible Manufacturing Cell (FMC). Taking into consideration all levels of functional issues, cellular architecture, standardization, and technical configurations, we take advantage of features released on a graphical programming commercial software, such as object oriented programming, distributed intelligence, and block function representation. The approach is addressed to encourage the usage of holonic and multi-agent concepts in traditional manufacturing systems, with friendly software based upgrade and a minimum low cost hardware expansion. LabView programming platform is selected for this development due to its easy applicability on holonic structures, where the structure is the programming code itself. The intention is to develop a system that will fit into different levels of an enterprise, including shop-floor and manufacturing execution systems, and simultaneously we will deal with technological migration from conventional manufacturing structure to a holonic-ready FMC.

Index Terms—Artificial intelligence, Computer integrated manufacturing, Distributed control, Holon, Intelligent manufacture, Manufacturing automation, Multiagent Systems, Parallel algorithms, Parallel architectures, Parallel processing, Petri nets, Reconfigurable architectures.

I. INTRODUCTION

The word "holon" comes from the Greek holos that means whole, with the suffix on which, as in proton or neutron, suggests a particle or part [1]. Holon is an autonomous, cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. A system of holons that cooperate to achieve a goal or objective limited by rules of interaction is called holarchy [2].

On the past decade researchers have focused their investigations in the theory and design of holonic manufacturing systems, where can be found two principal aspects that at present are still being depurated. On one hand we have issues associated with the development of holonic systems, on the other hand how the holonic systems can be effectively deployed into manufacturing environments [3]. In spite of having a complete set of holonic structures and holonic algorithms, they still do not have the strength to displace the investments in hardware or anything that have an acceptable performance on existing companies, even though the companies know that in a brief time market will change and some actions have to be taken. Manufacturing control systems should be therefore be based in distributed and autonomous entities, expandable, being possible the addition of new components without stopping or re-starting the process. All these necessary facilities lead to the concept of agile manufacturing systems [4]. More complex products, faster changing products, faster introduction of products, and the requirement of reduce investment, are disturbances; their occurrence may have severe impact in the performance of manufacturing systems, being also necessary to improve the system performance in terms of response to change [4], [5]. Nevertheless, the lack of uptake is in the most part caused by the requirement of some metrics, for example; how to evaluate the design, construction and operation of a holonic system vs. actual mass production systems, it is essential to deal with adequate software programming tools in order to map the holonic concepts to the software system design [3]. This paper presents a novel approach to manufacturing floor control design with holonic coordination, including the interaction through a MES with manufacturing planning level (See fig.1).

From the structure presented on figure 1 which is based on a previous research [6], [7], we will base the development of the platform that throughout this work we shall describe in deeper. This scheme uses commercial software that includes a few mainly distinctive characteristics, such as the flexibility to scale platform capacities without missing structure concept. Although, this is the main idea of object oriented programming, it is much more illustrative to visualize block functions than extended codes lines. Once a Holon is developed it must be as general as possible in order to be adapted in a wide range of industrial equipment that falls into the same category depending only on a minimum of physical requirements and intelligence training. This project will refer to a multi-agent manufacturing platform implemented at Monterrey Institute of Technology (ITESM), and along this project the holonic coordination will be implemented on one sub-system, including a framework to evaluate its performance, addressing this topic for future system improvements. During this paper the design of a multi-agent manufacturing platform and the technical adjustment to implement it on an ordinary FMC shall be explained.
II. A MULTI-AGENT MANUFACTURING PLATFORM DESIGN

The conventional architecture of a FMS consist on machinery, cells, and factory levels, their control functions are associated with each level ranging from real-time control and rigid operations to planning and scheduling functions at the shop-floor level. Its hardly assumption of changes makes traditional systems more centralized, in addition this structure is not compatible with the ideal holonic infrastructure [8].

Fig. 1. Floor control design with holonic coordination, including the interaction through a MES with manufacturing planning level.

This work defines a generic platform, which could be applied on traditional manufacturing systems, where the holonic expansion implies only the upgrade of a workstation with characteristics of cell coordinator. Thus, taking advantage of traditional shop floor interaction; such as OPC communications, dynamic libraries, ActiveX functions, in other words a system which software structure is based on an integrated control, then is possible to apply holonic concept into traditional manufacturing systems. It is essential for this development to preserve traditional hardware configurations, with just a few extra requirements. What we are trying to avoid is the complex implementation of new and costly hardware investments. Instead, the suggestion is to upgrade one or two workstations to integrate autonomy, co-operation, self-organization, and reconfigurability to the actual systems.

There are two main reasons of why we decide to apply HMS and multi-agent technology using LabView platform; parallelism and distributed systems. In terms of software coding "parallelism" use to be a hard challenge, the parallel execution is a critical subject on automation systems, where multiple task and elements are in real time completing duties and tasks, closing control cycles while data is being communicated simultaneously to a human machine interfaces or embedded applications. This kind of system is called distributed systems [9], whereby with a single control workstation it is possible to control in real time several devices. Conventional distributed systems demand several processor units taking care of one centralized system (robot, conveyor, inspection, processes…), and those processor units at the same time must be interconnected in a complex network vulnerable to noise and data alteration. The suggestion is to concentrate this complexity in one processor, to improve data control transmission, likewise decreasing data communication errors. In addition, LabView bids the concept of object oriented programming, allowing one of the main characteristics of holonic systems, such as reconfigurability and easy expansion without redundant coding lines, the processing and communication model proposed on this paper is illustrated (See fig. 2).

Fig. 2. A multi-agent model, functions and layers.

Referring to figure 2, the model pretends to show how holonic and agents will interact, which are the main actions that have to be executed and by whom. As can be seen, management holons are strongly related with superior levels, its process comes directly from data processed by a manufacturing execution system, this layer is in charge of system-level management and services such as life-cycle services, collection services, and query services. Whereby, scheduler holon is comprehend at this layer and accepts orders from system manager (updated orders); conducts tasks, and dispatches jobs to equipment, material and labor managers (execution holons). Some temporary holons are created in order to accomplish some tasks; however those holons are unloaded once inspection or when quality Holon releases the product manufactured [10].

Although next level is an intermediate layer, its application runs on the same processor unit, therefore this is called "multi-agent parallel processing", here is where distributed intelligence takes place. A deterministic network could be applied in software level to separate cycle process loops from other cycle process inside the cell production. Thus, control and monitoring systems are integrated in a solid platform; nevertheless everything is isolated to thwart effects caused by individual errors or hardware problems.Executable holons, are the base of the manufacturing model, on one hand this executable holons receive more specific orders, (e.g. movements, and execution of production programs). On the other hand these holons are in charge of real world communication (status of product handling, starting information and finalizing information of production, maintenance order, and timing).
III. TECHNOLOGICAL MIGRATION FROM STANDARD FMC TO HOLONIC-READY FMC

The long established cells have solved production issues for years, however market is growing up so fast that conventional production lines, are close to be an option that suddenly will not satisfy market necessities; to mention some of those necessities we have: more complex products, high quality requirements, faster changing products, faster introduction of products, a volatile output, and the necessity of reduce investment. As we quoted previously on introductory part of this paper, manufacturing control system should be based in distributed and autonomous entities, expandable [4], moreover being able to produce a family of products without large setup times. Next sections show how a commercial FMC (See fig.3) is adapted to be a FMC with holonic features; we name this new FMC as "holonic-ready FMC".

IV. THE ITESM MANUFACTURING CELL

The laboratory installed at ITESM (Monterrey Institute of technology), consist of two identical cells equipped with one loop belt-conveyor, one robot (Motoman UP6), one ASRS (automatic storage retrieval system) installed in a warehouse of 2x12 storage slots, a CNC machine(EMCO PC MILL 155), and an assembly table for each cell (See Fig.3). The conveyors have three docking stations: robot, inspection and storage station. When a raw material is introduced by an operator production orders are delivered, so that each module is aware of their tasks and roles on production.

When batches of raw material are deposited onto the belt-conveyor (Conveyor Holon), it must be aware at any time of which tasks are designated to each pallet that is navigating on the conveyor, and depending on the assigned task it can stop pallets at different docking stations in order to execute a process. When raw material is stopped at robot docking station, it could be delivered to CNC machine or assembly buffers, these tasks are performed by the manipulator (Motoman UP6). What to do and when has to be done, are examples of the information that order holons deal with the cell, more specifically to those executable holons in charge of that area or cell section. In spite of the flexible characteristics presented above, this structure is not compatible with the ideal holonic infrastructure, it is hard to adapt and centralized as well, any error condition stops production for a wide average of time. The habitual production concept will be presented to make clear how the cells used to perform manufacturing processes before the holonic ready-design. It is possible to define the dynamic and static operation of these cells in a graphical manner. On one hand, we have the static operation, (See fig.4); it just allows one product plan per operation cycle, wasting the most part of resources located at the cell and prolonging production.

![Fig. 4. UML for Static operation standard cell representation](image)

![Fig. 5. The Object Oriented Structure implemented and the analogous conveyor sketching with real conveyor holonic control coding.](image)
All these aspects are constraints of established manufacturing cells. Taking up again the effort to develop a platform; we are aiming to create an easy way to adapt holonic and multi-agent concept, without crashing conventional hardware structures and trends, in order to avoid affecting in a representative way the infrastructure established.

V. “HOLONIC-READY” FMC

Since the multi-agent technology has been recognized as a key concept in building a new generation of highly distributed, intelligent, self-organizing and robust manufacturing control solution, the traditional concept of manufacturing flexible or adaptable, has become vulnerable to changes [11]. Adaptability to manufacturing volume and environmental changes, failure detection, reconfigurability, and expandability; are a set of capabilities that make an attractive option the idea to improve traditional FMSs. To achieve this project; the development of multi-agent system is based on an alternative commercial software platform which recently has been improved making possible the usage of OOP (object oriented programming), distributed intelligence processing, OPC data communication sockets, DLL interfacing, ActiveX function, and external code interfacing (See Tab.1). The table presented shows few characteristics that this alternative commercial software bids vs. C++ functions; moreover its data manipulation is under a friendly graphical environment.

### TABLE I

<table>
<thead>
<tr>
<th>Feature</th>
<th>LabView</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor/destuctors functions</td>
<td>Don’t needed</td>
<td>✓</td>
</tr>
<tr>
<td>Parallel processing</td>
<td></td>
<td>Line by Line</td>
</tr>
<tr>
<td>Ancestral class for every object</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Reference and parameters</td>
<td>Have just one syntax of value, and reference is inherent</td>
<td>Have both reference and parameters</td>
</tr>
<tr>
<td>Data mutation</td>
<td>Automatically, user could have access to old values, even though class has been edited</td>
<td>✓, user has to track versions, and write extra mutation coding.</td>
</tr>
<tr>
<td>Inheritance</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed Intelligence coding</td>
<td>✓, could be applied on one workstation.</td>
<td>✓, it is possible to develop with some PC interconnected by a deterministic networking, for example.</td>
</tr>
</tbody>
</table>

Let’s take a general case of the project to explain how OOP would be implemented under this graphical environment. The Cell description, previously presented, refers to a belt-conveyor with three docking stations, even if there are three docking stations, it is essential to think on the most general application (a belt-conveyor with "n" docking stations).

To satisfy the general necessity we define methods, events, and characteristics that modify a general docking station class (See fig.6). Applying this logical and to make the most of Inheritance, cell expansion not longer depends on how many docking stations are in a cell.

Besides, the structure and coding organization could be sketched in certain way to represent the actual status of the cell. In the general case represented on figure 6, the tasks of methods are to take information about the actual statutes of docking station (Id of pallet traveling, stop order, release order, and time of pallet locked). Holonic Skills and knowledge should be attached to the inputs and outputs of these VI's shown on figure 6, those entities could be developed on different coding resources, such as Matlab, C++ or JAVA, and this can be encapsulated as a external code node in such a way that it can be used on LabView interface. Inspired on holonic models previously planned on earlier research projects, the design of intelligence holons shall be based on those works [11].

Since the driving forces behind these studies are mainly oriented to develop a generic platform that bids the easy possibility to expand intelligence block functions in order to achieve migration from traditional FMS to IMS, we will not deal with complete holonic design in depth; however we shall present a single case to represent how holonic actions will be deployed.

VI. HOLONIC IMPLEMENTATION EXAMPLE

As a general case we might mention different contingence cases that could be covered by artificial intelligence for example missing raw material at warehouse, the occupancy of pallets or material transporting accessories, robot collisions, the disqualification of a sub-system because of failures or hardware problems, the lack of stability control in a subprocess or mechanism, and changes on production demand for several pieces or different products at the same time (See fig.7).
To have a brief description of the petri net presented on figure 7; the manufacturing execution system notify about changes on production to the central blocks, where multi-agent parallel processing takes place. At the same time this agents reprogram the schedule production to handle order task holons and supervise them in new production jobs. Task holons must have a record of any program needed on production to set them directly to executable holons once new changes have been established. It is vital to new concept of IMS, that current production do not result affected by those changes and the actual manufacturing process continues normally even when new production is being performed simultaneously.

VII. CONCLUSIONS AND FUTURE WORKS

This prototype reveals that the usage of a graphical environment that actually bids common functions of coding that could be an option to displace traditional manufacturing process and encourage the holonic-ready concept. The work in this paper is geared towards direct applicability in real production line, also we present approaches to developing a flexible system. Furthermore, the applicability of a MES (Manufacturing Execution System) is easier due to distributed intelligence facilities to make the most of the data concentration (timing, production, failures, and quality) in a flexible system.

The work on this development will continue to improve the design and performance of multi-agents layers; moreover the presented approach opens several research perspectives that are currently pursued by the authors.

VIII. REFERENCES


IX. BIOGRAPHIES

Jorge M. Gamboa received his BS degree in electronic and communications engineering, at Monterrey Institute of technology (ITESM), Monterrey, Nuevo León in 2005. Since 2006 he has been part of the Automation Graduate Program at ITESM, campus Monterrey, México. Since then, he has been engaged in teaching in the areas of automation, manufacturing systems and control systems in Mechatronics laboratories. His employment experience included the Across Whirlpool Company (electric appliance systems manufacture), México, Vitroflex (windshield manufacture field). His special fields of interest integrated control process, automation, intelligent manufacture systems and robotics. At the moment he is finishing his master degree in automation and control.

Miguel J. Ramírez received his Master degree in electronic systems, at Monterey Institute of technology (ITESM), Monterrey, with a certification in Manufacturing Technologies (CMfgT) by SME. Since 2002 he is professor at the campus in Mechatronics department. He teaches and coordinates areas such as manufacture systems automation and integrated manufacturing systems. His professional experience incorporated the development of training and consult programs related with numeric control technologies, robot systems, and design of automated production lines oriented to metalworking enterprises. At the moment he is candidate to Ph.D. in Mechatronics systems at the ITESM, having participated on international research stays in Spain, England and Germany.
Intelligent Manufacturing Systems: a methodology for technological migration

Jorge Gamboa-Revilla and Miguel Ramírez-Cadena.

Abstract—More complex products in efficiency and quality, the necessity of diminish energy spending and investment reduction; amongst other things, are disturbances, and their occurrence may have severe impact in the performance of actual manufacturing systems. Manufacturing systems should be based in distributed and autonomous entities, being possible the addition of new components without stopping or re-starting processes. All these facilities point to the concept of agile manufacturing systems. The approach is addressed to encourage the usage of holonic and multi-agent concepts in traditional production lines, with a friendly software upgrade and a minimum cost in hardware expansion. A methodology that includes the technological migration from a established flexible manufacturing structure (FMS) to intelligent and reconfigurable manufacturing system (RMS) is presented. An example of implementation will be described in depth to show the viability of the proposed schema.

Keywords: Computer integrated manufacturing, Distributed control, Holon, Intelligent manufacturing systems, Multiagent Systems, Parallel architectures, Parallel processing, Reconfigurable Manufacturing Systems

1 Introduction

In the last twenty years manufacture concepts have had several redefinitions, in the eighties, the concept of flexible manufacturing systems (FMS) was introduced to develop a new family of products with similar dimensions and constraints. But nowadays, the capacity of reconfiguration has become a major issue for improving the functioning of industrial processes. Indeed, today a main objective is to adapt quickly in order to start a new production or to react in a failure occurrence [1]. Intelligent manufacturing systems (IMS) [2], has both flexibility and reconfigurability, in fact this concept brings more than a few ideas of software intelligence meanings, which contemplates characteristics such as autonomy, decentralization, flexibility, reliability, efficiency, learning, and self-regeneration, all of these facilities lead to the concept of agent-based manufacturing systems. An agent is a computer system that is situated in some environment, and that is capable to act in an autonomous way in this environment in order to meet its design objectives. Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their goals, this characteristic is well known as reactivity. However an agent is also proactive, for it agent is able to exhibit goal directed behavior by taking the initiative. In addition, agents are social, having the ability to interact with other agents [3]. It worth to remember the definition of an "Holon", which its similarities with agent definition, brings up controversial meanings, nevertheless an holon is well recognized on manufacture applications with the distinctive of a more specific intelligence use, while an agent could have different levels of intelligence such as logical, reactive, layered or in a more advanced way, with beliefs, desires and intentions (BDI)[4]. The word "holon" comes from the Greek holos that means whole, with the suffix on which, as in proton or neutron, suggests a particle or part. A system of holons that co-operate to achieve a goal or objective limited by rules of interaction is called holarchy [5]. On the past decade researchers have focused their investigations in the theory and design of holonic manufacturing systems (HMS), wherein can be found two principal aspects that at present are still being depurated. On one hand we have issues associated with the development of multi-agent systems (MAS), on the other hand how the MAS can be effectively deployed into manufacturing environments [6]. In spite of having a complete set of agent architectures and algorithms, they still do not have the strength to displace established manufacturing systems, even though the companies know that in a brief time market will change and some actions have to be taken. This paper presents a novel approach to manufacturing floor control design with agent coordination, including the interaction through a manufacturing execution system (MES) with manufacturing planning level (See fig.1) structure taken from previous researches [7, 8, 9]. This scheme uses commercial software that includes a few mainly distinctive characteristics, such as block oriented programming, parallelism for distributed structures, and the flexibility to scale platform capacities without missing the structure concept. This article will refer to a multi-agent manufacturing platform imple-
Figure 1: Floor control design with holonic coordination, including the interaction through a MES with manufacturing planning level.

2 The feature of migration

Since the multi-agent technology has been recognized as a key concept in building a new generation of highly distributed, intelligent, self-organizing and robust manufacturing control solution, the traditional concept of manufacturing systems, has become vulnerable to changes [10]. Environmental changes, failure detection, reconfigurability, and expandability; are a set of capabilities that make an attractive option the application of this feature of migration. The idea of a standard software platform including characteristics such as reconfigurability, flexibility and “holonic-ready” [9] concepts, is justified by the necessity of uptake on established systems, making easier to adopt new production infrastructures without dramatic hardware changes and long setup times. At the moment it is possible to find several topologies of manufacturing cells, such as centralized, hierarchical, and heterarchical structures [11]. Each topology could be considered as optimal and able to accept migration, taking into account that each block should be related without complete dependency, at least after migration is implemented, and well functioning shall not be compromised with any other element from the cell. A generic platform was designed in order to apply multi-agent schema [9]. The platform design was implemented in such a way that any flexible manufacturing cell could be evolved into agent-based structure. The clue is to adopt the platform structure, and shape each element (robots, numeric control machinery, conveyors) of a cell to acquire agent personation. Once the problem or problems are identified the MAS design phase, starts, which is more oriented toward the implementation of the generic platform; however a methodology should be committed. The methodology includes definition of: 1st stage; capabilities of each single agent, and the inter-agent communication, 2nd stage MAS architecture planning.

3 The Methodology

Before any attempt can be made to implement agent societies effectively in a manufacturing system, an analysis of the industrial life cycle is pivotal. It therefore becomes important to introduce the environment in which an agent should act [12]. For it the information system of a manufacturing enterprise is crucial to be recognized, in order to clearly sketch how agents can be integrated and how data would be interchanged (See fig. 2), wherein the three layers, that computer systems in manufacturing management use, are illustrated. The generic platform is toward from general to particular application, so before start working on developing intelligence, is crucial to make independent each element, which is supposed to emerge from a centralized and sequential architecture that actually shall be substituted by the new platform.

Figure 2: Hierarchy model of communication and interaction

This section will start taking the hierarchy presented on figure 1 and model presented on figure 2, taken from earlier works on this research [7, 8, 9]. The superior part of this pyramid is performed by management layer, which are satisfied with a manufacturing planning level, and a manufacturing execution level. Both could be programmable holons, purely software based. In addition pyramid bottom is formed by executable holons, which has direct contact with machinery, and hardware systems, also this part of the pyramid frequently is the one with more constraints in manufacturing environments. The efforts on this section will be driven to get the pyramid base prepared to be adapted without neither hardware changes nor design, on the other hand ready to become
reconfigurable, and holonic-ready [9], the methodology (See fig. 3) shall be explained as follows.

- **Define Communication Structure:** This answers the issue, how data acquisition will be performed and how data shall be shared between items on internal and distributed network. On actual systems, this is an easy matter due to the well-known kinds of communication protocols. Data interchange is available in several ways, for example ActiveX, data library functions (*.dll), OPC and data sockets, and those at the same time use different well established channels of communication, such Control-Net, Profibus, Ethernet, Device-Net, amongst others, making even easier this step on the methodology.

- **Isolate from global system:** The well or bad functioning of one element should not affect to the other elements functioning. In other words is essential to rupture dependences. Isolating, could be a difficult part on this methodology, understanding that each element in traditional systems is related with a strong sequential logic, using strategies such as First-in-first-out (FIFO), Earliest Due Date (EDD), Shortest Processing Time (SPT), and Least Slack First (LSF), all of them has an acceptable performance and their use have solved many industrial optimization problems, nevertheless are sequential dependent, sequential operations, and dependency are rigid and that structure is not compatible with the ideal holonic infrastructure.

- **Convert from general to particular:** Scalability and generic features are the main topics on this methodology; we must conserve generality, thinking in advance to future hardware or software changes. Rigid and dedicated operation should be eliminated, to achieve different applications, making able to change its functions.

- **Create relationships but not dependences:** Elements should be able at the end to share data, hence is necessary to establish a weak relation with messaging protocols such as FIPA or contract Net, that with fire actions in order to perform an application. It follows that relationships must be created without missing complete agent-based structure.

The result after this methodology would be what we call a "holonic-ready agent" (HRA), which meaning contemplates an entity with characteristics and attributes necessary to adopt intelligence blocks (to become a Holon) in order to achieve different functions or tasks. An overall view of the resulting platform (See fig. 4) emerges from figure 4, where is shown in a more oriented way the methodology applied on the commercial software used to develop the generic platform. The methodology makes possible reconfigurability into the manufacturing cell, and at the same time the cell becomes ready to adopt second stage of the problem, Multiagent architecture selection. An to explain how reconfigurability is done, the robot routine, which contains ethernet procedures and *.dll functions to perform actions such as movements or execution of a specified routine. Lets imagine that we have to plug another identical robot to the cell, following this methodology procedure, it is just matter of duplicating robot cycle and change some kind of IP address to achieve plug and produce, like some authors have defined [13]. Holonic or intelligent agent skills and knowledge should be attached to the inputs and outputs of those isolated cycles, the intelligent entities could be developed on different coding resources, such as Matlab, C++ or JAVA, and these can be encapsulated as an external code node in such a way that they can be used on LabView interface.

4 Agent Architecture for generic platform

Before continuing with the study case, is fundamental to define some aspects about the agent architecture implemented, remember this refer to the second phase of migration problem. An agent could be categorized in purely reactive, in which do not consider historical data to react, and agents with state, this category contemplates past events, and contain internal states that describe the agent current situation, its perception of the world map to a set of possible actions to react in different manners. However these aspects still do not clarify how functioning might be implemented, functioning classes could be logical, reactive, intentional flexible (BDI) and layered, for this application case reactive agents is selected, in which decision making is implemented in some form of direct...
mapping from situation to action. For this implementation is necessary to create three different sets or vectors, in order to define the agent structure and its virtual environment. Equation 1 denote a representation of a set of discrete states $E$, which we can justify by pointing out that any continuous environment can be modeled by a discrete environment to any desired degree of accuracy, on the other side we have $Ac$ being a set of discrete actions. The basic model of agents interacting with their environments is represented on equation 2, as can be seen $r$ is a sequence with actions firing states, hence equations 3 and 4 are sequence terminated by actions or states respectively. The environment starts in some state, and the agent begins by choosing an action to perform on that state.

$$E = s_0, s_1, ..., s_u + 1; Ac = \alpha_0, \alpha_1, ..., \alpha_u + 1;$$

$$r : s_0 \Rightarrow \alpha_0 \ s_1 \Rightarrow \alpha_1 \ s_2 \Rightarrow \alpha_2 \ ... \Rightarrow \alpha_u \ s_u + 1$$

$$R^{Ac} : s_0 \Rightarrow \alpha_0 \ s_1 \Rightarrow \alpha_1 \ ... \Rightarrow \alpha_u$$

$$R^{E} : s_0 \Rightarrow \alpha_0 \ s_1 \Rightarrow \alpha_1 \ ... \Rightarrow \alpha_u \ s_u + 1$$

As a result of this action, the environment can respond with a number of possible states. However, only one state will actually result, and obviously the agent does not know in advance which it will be. The rules that govern environment are established by the state transformer, equation 5, at the same time each agent is defined by equation 6, in which an agent receives a run or sequence terminated by a state, an agent should map this situation to an action[3].

$$\tau(R^{Ac}) : R^{Ac} \Rightarrow \nu(E)$$

$$Ag : R^{E} \Rightarrow Ac$$

Although architecture is designed by these abstract models, the following pseudo code, represent in a very general way how these models are implemented and the study case is developed:

1. function action(p:P):A
2. var fired:(R)
3. var selected:A
4. begin
5. fired:=\{(c,a)|l(c,a)^R and p^c\}
6. for each (c,a)^fired do
7. if !(c^c,a^c)^fired such that (c^c,a^c)<(c,a))
8. then return a
9. end-if
10. end-for
11. return null
12. end function action

Thus action selection begins by fires computing the set fired of all behaviors that fire (line 5). Then, each behavior (c, a) that fires is checked, to determine whether there is some other higher priority behavior that fires. If not, then the action part of the behavior, a, is returned as the selected action (line 8)[14].

5 The ITESM manufacturing cell

The laboratory installed at ITESM, consist of two identical cells equipped with one loop belt-conveyor, one robot (Motoman UP6), one ASRS (automatic storage retrieval system) installed in a warehouse of 2x12 storage slots, a CNC machine(EMCO PC MILL 155), and an assembly table for each cell (See fig. 5). The conveyors have three docking stations: robot, inspection and storage station. When a raw material is introduced by an operator production orders are delivered, so that each module is aware of their tasks and roles on production. When batches of raw material are deposited onto the belt-conveyor (Conveyor agent), it must be aware at any time of which tasks are designated to each pallet that is navigating on the conveyor, and depending on the assigned task it can stop pallets at different docking stations in order to execute a process. When raw material is stopped at robot docking station, it could be delivered to CNC machine or assembly buffers, these tasks are performed by the manipulator (Motoman UP6). What to do and when has to be done, are examples of the information that order agents deal with the cell, specifically to those executable agents in charge of that area or cell section.

6 Validating reconfigurability and agent implementation

Previous to this section, the ITESM manufacturing cell was described in detail in order to make a global view of how elements are initially set and how original operational flux is performed in a traditional environment. In
In this section the configuration of the cell will be altered physically in a non-dramatic way to ensure reconfigurability after implementing the "Holonic-ready" platform, also two elements with no previous interaction will have to cooperate in order to achieve a common goal. It is essential to avoid long setup times, extra physical wiring, or extra monetary investments, to demonstrate the feasibility of migration. The study case begins with some physical changes, figure 5 shows the layout of the ITESM cell, for this application the camera from inspection station will be attached to robot assembly table, its image processing shall construct the perception of robot agent, in other words camera should be the medium that makes the robot able to observe its environment, whereas the robot agents performs decision making process (See fig. 6). The tasks are defined as follows; raw material is delivered by an operator, and this material is formed by a pallet with geometrical figures, as shown in figure 6, these figures do not always conserve same patron of placing, thus the robot should perceive by the camera current state from environment, then the robot performs a specific routine or action to deliver each figure to another pallet with a specific location for each geometrical figure (See fig. 7). The petri net demonstrate on figure 7, how actions and states modify environment from agent perspective, in a dynamic comportment. How often the robot agent performs a determinate route or path is established by the utility each path pays. The amount of utility given for each figure could be assigned by programmer. Nevertheless an agent always tries to maximize the utilities that it can obtain from a task [4], equation 7.

$$A_{opt} = \arg \max_{A \in AG} \sum_{r \in R(A, Env)} u(r) P(r|A, Env)$$

(7)
7 Conclusions and future works

The physical changes were successfully achieved, there are several ways to qualify this characteristic such as time and hardware adaptation, even both aspects were optimized with the usage of the holonic-ready platform, if there were dimension changes on assembly table for example, collision of work space would be also an important problem to solve, nevertheless collision could be avoided adding some extra collision avoidance algorithms, it always will depend on how old-fashioned integrated systems are in the FMC to migrate and its ability to interact. In other words solutions for different elements, will depend on how flexible or communicable they are, as a result we could have several solutions. However preparation of a generic platform that actually could adopt different solutions seems the most urgent issue. The platform shows sufficient flexibility to accomplish the unexpected request of assembling products, as well as showing flexibility in removal, addition and reconfiguration of assembly devices. We could succeed to implement a holonic-ready platform in a generic mode showing its capability for migration to convert common FMSs into RMS agent-based systems. As future work a scheduler of assembly devices shall be developed, and interaction with superior levels such manufacture execution system, both have to be developed in a generic schema to be adopted on the platform, opening different research lines such as logistics and planning for intelligent manufacture, and technological migration.

8 Acknowledgements

This work is a contribution for the Rapid Product realization for Developing Markets Using Emerging Technologies Research Chair, Registration No. CAT077. The authors wish to acknowledge the support of the Tecnologico de Monterrey, Campus Monterrey, Graduated Program department and the Mexican Science and Technology Council (CONACYT) sustaining under Grant J110256.

References


