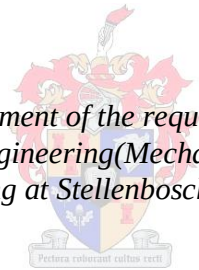
Evaluation of Control Strategies for Reconfigurable Manufacturing Systems

by
Chibaye Mulubika



Thesis presented in fulfilment of the requirements for the degree of Master of Science in Engineering(Mechatronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof Anton Basson

March 2013

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

25th February, 2013

Abstract

Evaluation of Control Strategies for Reconfigurable Manufacturing Systems

C Mulubika

Department of Mechanical and Mechatronic Engineering

Stellenbosch University

Private Bag X1, 7602 Matieland, South Africa

Thesis: MScEng (Mechatronic)

March 2013

The thesis evaluates control strategies for reconfigurable manufacturing systems by using a welding assembly cell as a case study. The cell consists of a pallet magazine, conveyor, feeder subsystem (comprising an articulated robot and singulation unit), welder subsystem (which uses a modular Cartesian robot), and inspection and removal subsystems. The research focuses on control strategies that enhance reconfigurability in terms of structure, hardware and software using agent-based control and the IEC 61499 standard, based on PC control. Reconfiguration may occur when a new product is introduced, as well as when a new subsystem is introduced or removed from the production cell.

The overall control architecture is that the subsystems retain no knowledge of the product, but product information resides in the cell controller, while services offered by the subsystems are registered with the directory facilitator of the Java agent platform. The control strategies are implemented on the modular Cartesian weld robot and the cell controller for assembly cell. A layered architecture with low-level control and high-level control is used to allow separation of concerns and rapid changes in both hardware and software components. The low-level control responds in hard real-time to internal and external events, while the high-level control handles soft real-time actions involving coordination of control related issues.

The results showed IEC 61499 function blocks to be better suited to low-level control application in distributed systems, while agents are more suited for high-level control. Modularity in software components enhances hardware and software scalability. Additionally, agents can support online reconfiguration of reconfigurable machines.

Abstrak

Evaluering van beheerstrategieë vir Herkonfigureerbare Vervaardigingstelssels

C Mulubika

Departement van Meganiese en Megatroniese Ingenieurswese
Universiteit van Stellenbosch
Privaat Sak X1, 7602 Matieland, Suid-Afrika
Proefskrif: MScIng (Megatronies)
March 2013

Die tesis evalueer beheerstrategieë vir herkonfigureerbare vervaardigingstelsels deur gebruik te maak van ’n sweismonteersel as ’n gevallestudie. Die sel bestaan uit ’n palletmagasyn, vervoerbande, voersubstelsel (bestaande uit ’n geartikuleerde robot en singulasie-eenheid), sweissubstelsel (wat gebruik maak van ’n modulêre Cartesiese robot), en inspeksie- en verwyderingsubstelsels. Die navorsing fokus op beheerstrategieë wat herkonfigureerbaarheid verhoog in terme van struktuur, hardeware en sagteware met behulp van agent-gebaseerde beheer en die IEC 61499 standaard, wat gebaseer is op PC-beheer. Herkonfigurasie mag voorkom wanneer ’n nuwe produk in-gestel word, sowel as wanneeer ’n nuwe substelsel bygevoeg of verwyder word van die produksiesel.

Die oorhoofse beheerargitektuur is dat die substelsels geen kennis van die produk hou nie, maar die produkinligting in die selbeheerder geberg, terwyl dienste wat aangebied word deur die substelsels wat geregistreer is by die gidsfasiliteerder van die Java agent platform. Die beheerstrategië is geïmplementeer op die modulere Cartesiese sweisrobot en die selbeheerder vir die monteersel. ’n Gelaagde argitektuur met ’n lae-vlak beheer en hoë-vlak beheer word gebruik om skeiding van oorwegings en vinnige veranderinge in beide hardeware en sagteware komponente toe te laat. Die lae-vlak beheer reageer hard intyds op interne en eksterne gebeure, terwyl die hoë-vlak beheer sag intyds die koördinering van beheerverwante kwessies hanteer.

Die resultate het getoon dat IEC 61499 funksie-blokke beter geskik is vir lae-vlak beheer toepassing in verspreide stelsels, terwyl agente meer geskik is vir hoë-vlak beheer. Modulariteit in sagteware komponente verhoog hardeware en sagteware skaleerbaarheid. Boonop kan agente ook aanlyn herkonfigurasie van herkonfigureerbare masjiene ondersteun.

DEDICATION

To
My wife Mutinta and my daughter Kabwe who

have endured separation and deprivation for a noble

Cause

May God continue to guide and bless us as we pursue our dreams.

ACKNOWLEDGEMENTS

I wish to acknowledge the effort of those who made this research process a success. First and foremost, I would like to thank Professor A.H Basson for his guidance and support during my research. I would also like to thank him for being a parent and organizing funding for my research. For your kind words and advice, God will bless you.

I would also like to thank all the able technicians from mechanical workshop for their speedy execution of the tasks. Without you the research would have taken forever. Maurisha, you are wonderful and I like to thank you a lot for your attitude and support. I wish you the best in life.

This research was going to be hell without the support of Taren Smith, who guided me with setting up of Festo equipment, and Coen Pretorius for making sure that shaft was the right size and selling us equipment to the right specification.

Ronaldo, I thank you for talking to me when you had the right not to, for giving your advice when it was needed. I would also like to thank my colleagues, Ruan for being kind and warm hearted. You made my stay in Stellenbosch enjoyable. Your encouraging words meant a lot to me. Am sure God will bless you for that.

I would further like to extend my thanks to Evan William to for his support during the installation of Beckhoff TwinCAT I/O.

To my colleagues, Anro and Karel, memories will last a long time. If I were asked to choose colleagues again for another research, I would choose you again.

Lastly but not the least, I would like to thank Advanced Manufacturing Technology Strategy (AMTS) for providing funding for this work, and the CBI for providing access to their plants.

Therefore, if there be any insufficiency in the quality of this work, the fault lay squarely on my head and not my supervisor.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ACL Agent Communication Language

AMTS Advanced Manufacturing Technology Strategy

CFP Call for Proposal

CNP Contract Net Protocol

DF Directory Facilitator

FB Function block

FMS Flexible Manufacturing System

FSM Finite State Machine

HLC High Level Control

IP Internet Protocol

JADE Java Agent Development Environment

LLC Low Level Control

MAS Multi-Agent System

PC Personal Computer

PLC Programmable Logic Controller

RM Reconfigurable Machines

RMS Reconfigurable Manufacturing Systems

TCP Transmission Control Protocol

1. INTRODUCTION

1.1 Background

This research considers control strategies for reconfigurable manufacturing systems (RMSs). It is a part of the research activities that have been undertaken to develop expertise in reconfigurable assembly systems in South Africa under the "Affordable Automation" theme of the Advanced Manufacturing Technology strategy (AMTS). AMTS is an initiative of the Department of Science and Technology aimed at developing technologies related to manufacturing industry.

For this research, the reconfigurable assembly cell for which control strategies are evaluated is a welding assembly cell used for production of circuit breaker components. Several students from the research group have worked on different aspects of the project. Sequeira (2008) developed a conceptual design of the welding assembly system. The design comprised five major subsystems, i.e. the pallet magazine, conveyor, feeder, inspection and removal station, and welder. The pallet magazine concept was designed and developed by Burger (2009), while Strauss (2009) designed a singulation unit, which is a part of the feeder subsystem. Kruger (2013) is developing a control system for the feeder subsystem, while Le Roux (2013) is developing the control system for the conveyor.

Students from Central University of Technology (CUT) in Bloemfontein developed a multi-agent control system which interfaces with the cell controller at the University of Stellenbosch. The CUT controller handles communication with other information systems in the factory, i.e. scheduling, security and high level human interfaces. Du Preez (2011), from the Department of Industrial Engineering at the University of Stellenbosch, developed a simulation procedure which determines, for a given product mix, an optimal assembly system configuration. The simulation also predicts the cost of production for a given product mix.

Work in this thesis evaluates control strategies for a modular Cartesian weld robot and the cell controller for the whole welding assembly cell. The cell controller will also interface with the multi-agent system developed by CUT.

1.2 Motivation

This work was motivated by the competition from the global manufacturing economy in which customers' and enterprises' preference for newer products have led to short product life cycles. The introduction of newer products would traditionally require changes to manufacturing system set-up, for instance, introducing new machines, as well as making changes to control programs.

In South Africa the situation is not different. South African companies have been forced to selectively replace labour for assembly so that manual labour and automatic operations may be combined and run concurrently in order to meet such

challenges. This selective replacement of manual labour has become increasingly necessary due to the direct and indirect cost of labour, as well as quality considerations.

For a manufacturing enterprise with high product variability and changeable volumes, RMSs offer a potentially attractive option to challenges faced with change in production capacity and functionality. Production volumes in South Africa are typically small and also vary substantially throughout the year. Furthermore, each product change may require a change in the manufacturing subsystems or adjustments to be made to the control program, especially when programmable logic controllers (PLCs) are used. Moreover, the time required to return the production to full capacity after a change in product occurs can be substantial. There is, therefore, a need for a control strategy which will enhance reconfiguration and reduce on the ramp up time.

In most industries, control of the manufacturing system is traditionally centralized. However, distributed control is used in the work presented here. The choice of distributed control is motivated by the difficulties associated with the traditional centralized control, for instance, any modification done to a centralized control will require a shutdown. However, any downtime is unproductive and may make the product costly, and the company less competitive.

Multi-agent systems and the IEC 61499 standard are some of the control standards that have been developed to effectively implement distributed control. Multi-agent systems and the IEC 61499 standard have both been used for holonic manufacturing systems (HMSs). This motivated the evaluation of the two standards to see which one would enhance reconfiguration in RMSs.

1.3 Objective

The objective of this research is to evaluate the ability of some distributed control strategies to enhance reconfiguration in terms of changes in structure, hardware and software components. Reconfiguration occurs when a new product is introduced, as well as when a new subsystem is introduced into a production cell.

The objective was approached by, firstly, designing a modular Cartesian weld robot using the six core characteristics of RMSs and the design principles of Reconfigurable Machines (RMs), and then implementing agent based control and IEC 61499 function blocks as alternatives to each other. Each approach is then evaluated in terms of ease of reconfiguration.

Secondly, a cell controller for the welding assembly cell, consisting of the pallet magazine subsystem, Bosch Rexroth TS2 Plus conveyor, feeder subsystem, welding subsystem, inspection and removal subsystem, is developed and tested using agent based control. All the control approaches are implemented using Personal Computers (PCs) since available (PLCs) do not support agent based control or IEC 61499 function blocks.

2. LITERATURE REVIEW

This section reviews manufacturing systems that have been developed in the past and the control strategies that have been used on them. It also considers the application of agents, holons, and IEC 61499 function blocks in distributed manufacturing systems.

2.1 Types of manufacturing systems

Manufacturing systems have evolved substantially from the time of their first inception. Stechi and Lagos (2004) highlights the evolution stages as: the dedicated manufacturing systems (DMS), the cellular manufacturing systems (CMS) and the flexible manufacturing systems (FMSs). They further make a case for the development of reconfigurable manufacturing systems (RMSs). The evolution was driven by various challenges. For instance, in the nineties, optimality, agility, waste reduction, quality and lean manufacturing were identified as key drivers and goals for ensuring survival in a globally competitive market (EIMaraghy, 2006). Similarly, Koren and Shpitalni (2010) identify cost, functionality, and capacity as the three features differentiating RMS, DMS and FMS. Each of the system types will be considered in more detail in the following sections.

2.1.1 Dedicated manufacturing systems

Dedicated manufacturing systems, sometimes called dedicated manufacturing lines (DML) or transfer lines, can produce a company's core products or parts at high volumes on dedicated machines. Koren and Shiptalni (2010) describes DML as comprising of inexpensive fixed automation and Koren et al (1999) further describes them as not scalable since they have fixed cycle times and capacity. These characteristics make them rigid in terms of product variation. Therefore, they cannot be globally competitive in a situation where product life cycles are ever changing and new products are frequently introduced.

However, a DMS is cost effective when the demand for a particular product exceeds the supply so that the DMS can operate at its full capacity (Koren and Shiptalni, 2010). Nevertheless, a DMS is at a disadvantage if the required production volumes change significantly or if the product is only produced for a short time.

2.1.2 Flexible manufacturing systems

FMS is described as "a manufacturing 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" (Mehrabi et al, 2000). The manufacturing system can produce a variety of products with changeable volume and mix on the same system (Koren et al, 1999). Koren et al (1999) further states that FMS hardware consists of expensive general purpose computer numerically controlled (CNC) machines and other programmable automations.

A flexible manufacturing system is designed to be flexible. Chryssolouris et al (2012) states that, flexibility of a manufacturing system is determined by its sensitivity to change and is evaluated by calculating the expected cost of accommodating possible changes in the operating environment. The lower the expected change cost is, the less sensitive the system is to changes in its operating environment and thus, the system is considered as being more flexible. Based on this definition, EIMaraghy (2006) identifies ten types of manufacturing flexibility and these are machine flexibility, material handling flexibility, operation flexibility, process flexibility, product flexibility, routing flexibility, volume flexibility, expansion flexibility, control program flexibility and production flexibility. While these promote better understanding of various types of flexibility, Chryssolouris et al (2012) observe that high flexibility or low sensitivity to a change provides a manufacturing system with three principle advantages, these are:

- *Product flexibility* enables a manufacturing system to make variety of part types on the same equipment. In the short term, this means that the system has the capability of economically producing small lot sizes to adapt to the changing demand for various products. In the long term, this means that the system's equipment can be used across multiple product life cycles, which increases investment efficiency.
- *Capacity flexibility* allows a manufacturing system to vary the production volumes of different products to accommodate changes in the volume demand, while remaining profitable. It reflects the ability of the manufacturing system to contract or expand easily. It has been traditionally seen as being critical for make-to-order systems, but it is also very important in mass production, especially for high value products such as automobiles.
- *Operation flexibility* refers to the ability of producing a set of products with the use of different machines, materials, operations, and sequences of operations. It results from the flexibility of individual processes and machines; that of product designs, as well as the flexibility of the structure of the manufacturing system itself. It provides breakdown tolerance – the ability to maintain a sufficient production level even when machines break down or humans are absent (Chryssolouris et al, 2012).

This manufacturing system, though flexible, is said to have a high initial cost and usually not all of its capabilities are utilized (Mehrabi et al, 2002).

2.1.3 Reconfigurable manufacturing systems

A RMS is designed at the outset for rapid change in structure, in both hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or in regulatory requirements (Koren et al, 1999). Therefore, the objective of RMSs is to provide capacity and functionality that is needed when needed. Proponents of this approach believe that it has the potential to offer a cheaper solution in the long run compared to FMSs, as it can increase the life and utility of a manufacturing system (Wiendahl et al, 2007).

Additionally, EIMaraghy (2006) mentions reconfigurability as being in line with the idea of expansion of flexibility. This implies that there are a number of similarities between reconfigurable systems and flexible systems which may make it difficult to differentiate between the two systems.

What then is the difference between flexibility and Reconfigurability? Wiendahl et al (2007) define flexibility as "the tactical ability of an entire production and logistics area to switch with reasonably little time and effort to new, although similar, families of components by changing manufacturing processes, material flows and logistical functions". Reconfigurability is defined as "the operative ability of a manufacturing or assembly system to switch with minimal effort and delay to a particular family of work pieces or subassemblies through the addition or removal of functional elements" (Wiendahl et al, 2007). From these two definitions two differences which are also key to differentiating between reconfigurability and flexibility can be deduced and these are:

- Diversity of work pieces handled: Reconfigurable systems may switch between different families of products, while flexible systems switch between similar products.
- Manufacturing system setup change: Reconfigurable systems may add or remove machine components, while flexible systems change the process or material flow.

Removing or adding machine components implies changes to both hardware and software components. Rooker et al (2007) categorize reconfigurations as basic and dynamic reconfigurations. Basic reconfiguration involves stopping the whole system in order to reconfigure, while dynamic reconfiguration does not involve stopping the whole system.

A reconfigurable system must have inherent features or properties such that the reconfiguration exercise is simplified. Wiendahl et al (2007) refer to these features as changeability enablers. Koren and Shpitalni (2010) identify changeability enablers, also known as six core reconfigurable characteristics as:

i. Customization (flexibility limited to part family) of system or machine flexibility limited to a single product family, thereby obtaining customized flexibility.
ii. Convertibility (design for functionality changes) being the ability to easily transform the functionality of existing systems to suit new production requirements.
iii. Scalability (design for capacity changes) being the ability to easily modify production capacity by adding or removing manufacturing resources, for instance machines, and /or changing components of the system.
iv. Modularity (components are modular) being the compartmentalization of operational functions into units that can be manipulated between alternate production schemes for optimal arrangement.

v. Integrability (interfaces for rapid integration) being the ability to integrate new modules rapidly and precisely by a set of mechanical, informational and control interfaces that facilitate integration and communication.
vi. Diagonisability (design for easy diagnostics) being the ability to automatically read the current state of the system to detect and diagnose the root cause of output defects, and quickly correct operational defects.

Bi et al (2008) identify reconfigurable machines (RMs) as the hardware systems of a RMS at the machine and device level, while the RMS is to be designed by using reconfigurable hardware and software (Koren et al, 1999). Two technologies which have been identified by Koren et al (1999) as necessary enablers for reconfiguration are: firstly, in software, modular, open-architecture controls that aim at allowing reconfiguration of the controller; secondly, in machine hardware, modular machine tools aiming at giving the customer more machine options. For the RMs, modularity, integrability and diagonisability allow rapid reconfiguration, but do not guarantee modifications in production capacity and functionality (Koren and Shiptalni, 2010).

Therefore, the core of the RMS paradigm is an approach to reconfiguration based on system design which encompasses the simultaneous design of open-architecture reconfigurable controllers and reconfigurable modular machines (Koren et al, 1999). The ultimate goal of the RMS is therefore to utilize a system approach in the design of the manufacturing process that allows simultaneous reconfiguration of the entire system, the machine hardware and control software (Koren et al, 1999).

2.1.4 Holonic manufacturing systems

Holonic Manufacturing Systems (HMSs) is an approach that combines the best features of hierarchical and heterarchical organizational structures (Blanc et al, 2006). Blanc et al (2006) further state that the concept can preserve the stability of a hierarchy, while providing the dynamic flexibility of heterarchies.

What is a holon? Van Brussel et al (1998) states that Koestler proposed the word holon. It is a combination from the Greek *holos,* which means whole with the suffix -on which, as in a proton or neutron, suggests a particle or part (Valckenaers et al, 1998;Van Brussel et al, 1998). The HMS consortium translated the concepts that Koestler developed for social organizations and living organisms into a set of appropriate concepts for manufacturing industries (Van Brussel et al, 1998). The goal is to attain manufacturing stability in the face of disturbances, adaptability and flexibility in the face of change, and efficient use of available resources (Valckenaers et al, 1998).

In order to understand and apply the concept of HMS to the manufacturing setting, the consortium developed a list of definitions. Valckenaers et al (1998) detail these definitions as:

- Holon: An autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon.
- Autonomy: The capability of an entity to create and control the execution of its own plans and/or strategies.
- Cooperation: A process whereby a set of entities develops mutually acceptable plans and executes these plans.
- Holarchy: A system of holons that can cooperate to achieve a goal or objective. The Holarchy defines the basic rules for cooperation of the holons and thereby limits their autonomy.

2.2 Control architectures in manufacturing systems

Centralized, hierarchical and distributed are the three control architectures identified by Meng et al (2006). These architectures differ in purpose and function, and are implemented in different manufacturing systems. This section looks at some of the control architectures used in manufacturing systems and how they are implemented.

2.2.1 Centralized control architecture

The centralized control architecture is one in which the whole system is controlled by one central controller carrying out all the automation processes. This control strategy has a number of shortcomings such as, difficulty of control system design, lack of flexibility, and a low level of fault tolerance. In order to reconfigure a centralized or hierarchical control system architecture, the whole system has to shut down and all data structures of the higher levels must be updated (Meng et al, 2006). Figure 2.1 gives an illustration of the architecture.

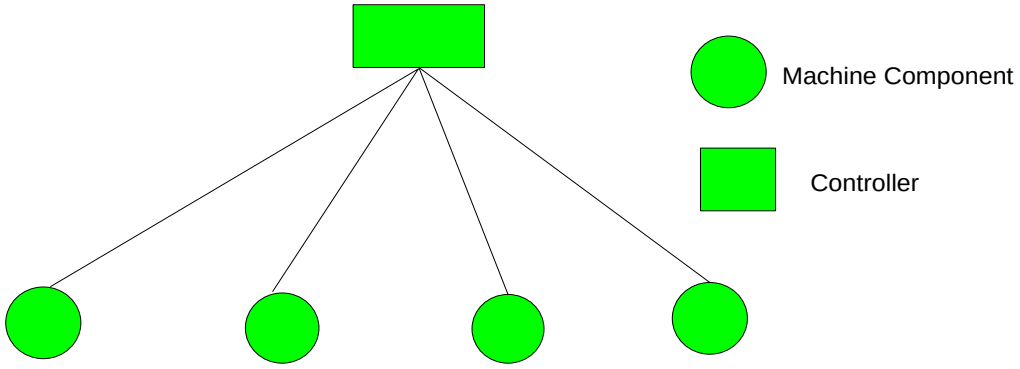


Figure 2.1 Centralized control architecture (Meng et al, 2006)

Furthermore, it is difficult to add, modify or delete resources. These reasons make centralized control strategy unsuitable for RMSs (Meng et al, 2006). However, Almeida et al (2010) state that where a centralised solution can be simply

implemented, maintained and changed, it will surpass a distributed solution in terms of conventional performance.

2.2.2 Hierarchical and heterarchical control architecture

Hierarchical control involves a command-response structure between high level and low level entities, while heterarchical control is achieved by allowing a high level of autonomy and decision making to be available to low level entities independent of the overall plant operations (Bongaerts et al, 2000).

In discrete manufacturing, developments in the information technology led to the realization of computer integrated manufacturing systems. With all its merits, integration resulted in rigid, hierarchical control architectures whose structural complexity grew rapidly with the size and the scope of the systems. Moreover, integration resulted also in complex decision problems (Monostori et al, 2006).

All hierarchical control architectures require a fixed structure while the system is running, and assume a deterministic behaviour of the components (Van Brussel et al, 1998). Figure 2.2 depicts heteracty and hierachy using graph theory.

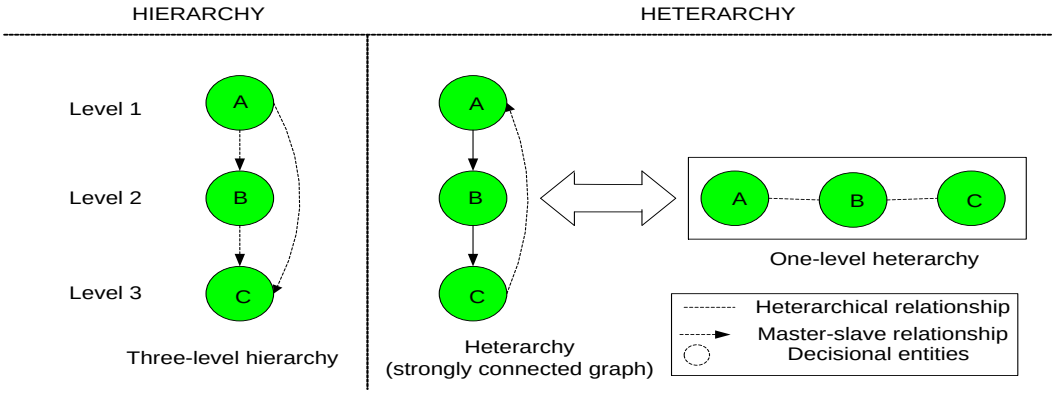


Figure 2.2 Formalization of heterarchy and hierarchy using graph theory (Trentesaux, 2009)

2.2.3 PROSA reference architecture

The name for Product-Resource-Order-Staff architecture (PROSA) refers to the composing types of holons (Van Brussel et al, 1998). Each of these holons is built on the basis that it is responsible for one aspect of manufacturing control. This can be logistics, technological planning, or resource capabilities.

In proposing the types of holons, Van Brussel et al (1998) state that in both the research community and manufacturing companies, three relatively independent manufacturing concerns do exist. These are:

- Resource aspects, such as driving the machine at optimal speed and maximizing its capacity.

- Product and process related technological aspects, such as which operations need to be performed to achieve a good quality product,
- Logistical concerns about the customer demands and due dates.

From this analysis, Van Brussel et al (1998) conclude that three basic holons exist in a holonic manufacturing system namely: product holon, order holon and resource holon. The three basic holons and their interactions in PROSA are shown in Figure 2.3.

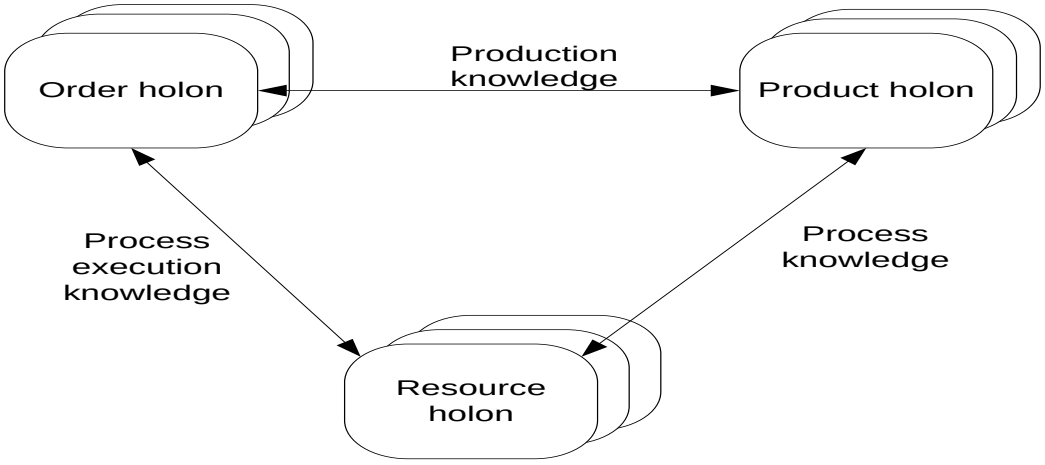


Figure 2.3 Basic building blocks of a HMS and their relations (Van Brussel et al, 1998)

The three key words used in defining relations between the three basic holons as shown in Figure 2.3 need explanation. According to Van Brussel et al (1998):

- Process knowledge contains information and methods on how to perform a certain process on a certain resource. It is knowledge about the capabilities of the resource, which processes it can perform, the relevant process parameters, the process quality, possible outcomes of a process, etc.
- Production knowledge represents the information and methods on how to produce a certain product using certain resources. It is knowledge about possible sequences of processes to be executed on the resources, data structures to represent the outcome of the processes, methods to access information of process plans, etc.
- Process execution knowledge contains the information and methods regarding the progress of executing processes on resources. It is knowledge about how to request the starting of processes on the resources, making reservations on the resources, how to monitor the progress of execution, how to interrupt a process, the consequence of interrupting a process, suspending and resuming processes on resources, etc.

The product holon holds the process and product knowledge to assure the correct making of the product with sufficient quality. This holon also contains consistent and up-to-date information on the product life cycle, user requirements, design, process plans, bill of materials quality assurance procedures etc. Therefore, this holon contains the "product model" of the product type and not the "product state model" (Valckenaers et al, 1998). The product holon also acts as an information server to the other holons in the Holonic Manufacturing System (HMS) (Van Brussel et al, 1998).

Similarly, a resource holon contains a physical part, namely a production resource of the manufacturing system, and an information processing part that controls the resource. It offers production capacity and functionality to the surrounding holons (Valckenaers et al, 1998). Each physical device of the manufacturing is incorporated in a resource holon (Blanc et al, 2006).

An order holon represents a task in the manufacturing system. It is responsible for performing the assigned work correctly and on time. It manages the physical product being produced, the product state model, and all logistical information processing related to the job. It also performs tasks traditionally assigned to a dispatcher, a progress monitor, and a short term scheduler (Valckenaers et al, 1998)

The name 'staff holon' is inspired by the difference between line functions in human organizations. Accordingly, the PROSA architecture allows the staff holons to assist the basic holons with information such that they can take better decisions. The basic holons are responsible for taking decisions; the staff holons are external experts giving advice (Valckenaers et al, 1998). Valckenaers et al (1998) also suggest that this manner of cooperation avoids the rigidity of conventional designs.

After comparing PROSA to other architectures, Van Brussel et al (1998) concluded that PROSA covers all aspects of hierarchical and heterarchical control architectures by incorporating relevant functions and control algorithms from centralized and distributed control approaches. Therefore, PROSA can be regarded as a generalized approach of centralized and distributed control approaches (Van Brussel et al, 1998).

The other two innovations introduced by PROSA are: decoupling of system structure from the control algorithm; and the decoupling of logistical aspects from technical ones. These innovations allow incorporation of more advanced hybrid control algorithms (Van Brussel et al, 1998). Decoupling is one of the main issues in the design of complex systems and therefore one of the important characteristic of PROSA architecture (Van Brussel et al, 1998).

2.2.4 ADACOR control architecture

Adaptive holonic control architecture (ADACOR) is based on the holonic manufacturing systems paradigm and in the following main foundations:

decentralized systems, supervisor entities and self-organisation (Lietao and Restivo, 2006). Like the PROSA architectures, it is built upon a set of autonomous and cooperative holons. These holons perceive their environment and responses to changes.

ADACOR architecture defines four manufacturing holon classes. Each holon is a representation of a manufacturing component that can be either physical resource or logic entity. Leitao et al (2005b), state that the four holons are: product holon (PH), task holon (TH), operational holon (OH), and supervisory holon (SH) and that the product holon, task holon and operational holon respectively represent, products, production orders and physical resources available in the shop (Leitao et al., 2005b). The PHs, THs and OHs resemble the product, order and resource holons defined in PROSA, while the SH is an ADACOR feature (different from the PROSA staff holon). The SH introduces coordination and global optimization in decentralized control and is responsible for forming and coordinating groups of holons (Leitao et al, 2005a).

In order to fully appreciate the semblance, difference and the significance of PH, TH and OH holons, Leitao et al (2005b) give high level Petri net models of these holons. Figure 2.4 gives an illustration of the product holon model.

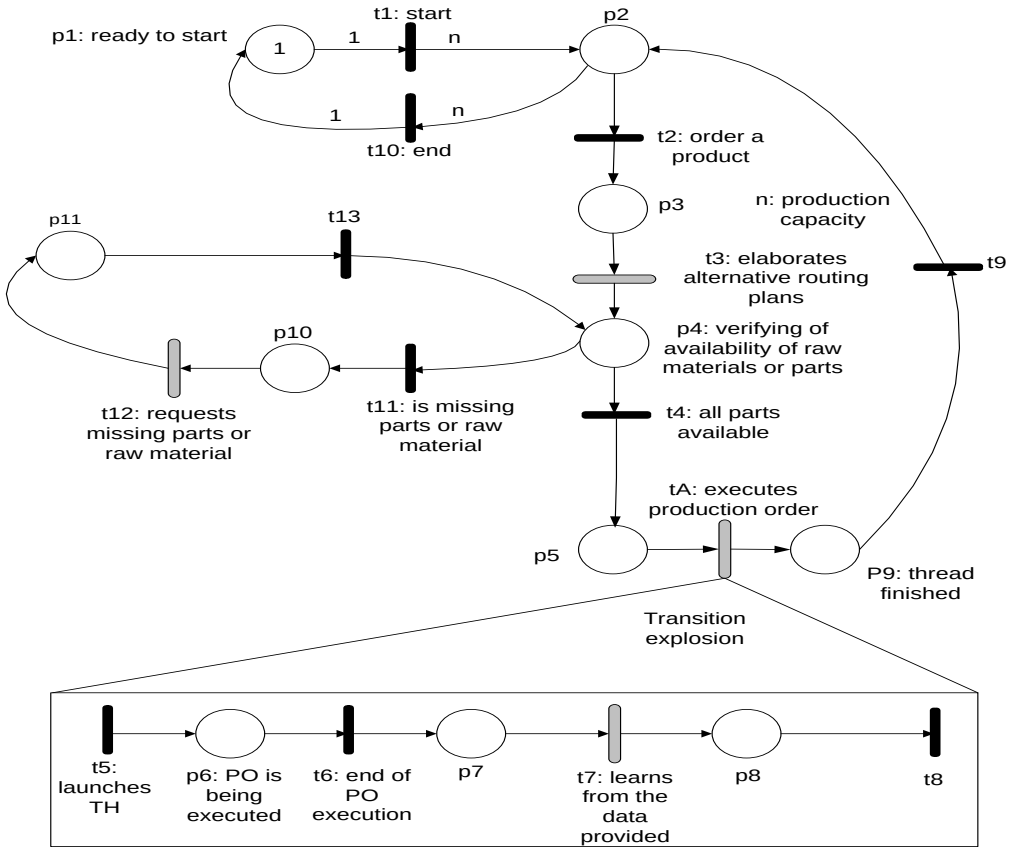


Figure 2.4 Petri net model of product holon (Leitao et al, 2005b)

Each PH is a representation of the product to be produced by the factory. When a new order is placed, it generates a new thread to handle its execution. The order comprises mainly of the short term process planning, management of the sub-parts and management of the production order execution (Leitao et al, 2005b). The PH continues to wait for new orders when it is finished. Moreover, it is able to simultaneously process several orders and is only limited by the production capacity n as depicted in Figure 2.4.

The transition *tA* in Figure 2.4, representing a production order, is exploded to show the activities that takes place when the PH launches a task holon (TH). Each available production order launched to produce a product is represented by a task holon. The behaviour of a TH comprises mainly the order decomposition, resource allocation planning and execution activities (Leitao et al, 2005b).

2.2.5 HCBA control architecture

Holonic Component-Based Architecture (HCBA) derives its concepts from component-based development (CBD) and HMS (Chirn and McFarlane, 2000). CBD is associated with a shift from statement-oriented coding to system building by plugging components together. The approach of CBD focuses much on developing reusability and reconfigurability at architecture level rather than individual software modules (Chirn and McFarlane, 2000). Physical objects of a manufacturing plant can be categorized into either a resource, performing the manufacturing operations, or a product which accepts manufacturing treatment (Chirn and McFarlane, 2000).

The resource component or resource holon is a self-contained system component which can perform operations on works in progress (WIP), such as fabrication, assembly, transportation and testing. Therefore, a resource holon contains these two main parts: a software part in the computing environment for control and decision making, and a physical part in the physical plant for actual fabricating (Chirn and McFarlane, 2000).

The product component or product holon also contains a physical part and a control part. The physical part may include raw material, parts and a pallet or fixture. A control part may contain routing control, process control, decision making and production information (Chirn and McFarlane, 2000).

HCBA is inherently distributed in terms of system structure and design philosophy.

2.2.6 Distributed control architecture

Because of the many difficulties faced with centralized control, one widely used solution to meet this problem has been distribution of decisional capabilities to decisional entities. It is important to note that sometimes “distributed” is used to refer to distribution of resources and not the control.

In the early 1970s, the first kind of control distribution was fully hierarchical and based on the Computer Integrated Manufacturing (CIM) paradigm (Trentesaux, 2009). However, since the 1990s, distribution of control decisions has been considered and has been characterized by the need for local reactivity (Trentesaux, 2009). In this arrangement, negotiation and cooperation between distributed resources is the main process of interaction apart from coordination (Marik and Lazanky, 2007).

There are challenges that come with designing distributed control architecture. Trentesaux (2009) highlighted that the dynamic behaviour of loosely linked autonomous decisional entities, as found in holonic and multi-agent systems, makes it hard to predict the behavior of the entire system.

A purely distributed control cannot be found in industry (Meng et al, 2006). Therefore, hierarchy will still be found in distributed contol. Bongaerts et al (2000) state that hierarchy in distributed control gives certain advantages. The three advantages cited are: firstly, a hierarchical centralised element, such as a scheduler, optimises the performance of the overall system; secondly, the ability to predict the behaviour of a distributed system particularly with respect to the progress of individual orders and loading on resources; and finally the ease of migration effort towards distributed (holonic) systems, as a means of support to a gradual shift from hierarchical systems to distributed systems (Bongaerts et al, 2000).

2.3 Agent based control in manufacturing systems

After considering the control strategies currently used in industry, Meng et al (2006) suggest that agent based control is the most natural way to implement schedule and control for RAS (Reconfigurable Assembly System). Meng et al (2006) further suggest that "Multi-agent systems are capable of changing the traditional architecture of manufacturing systems and overcoming the short comings of centralized and hierarchical architecture" (Meng et al, 2006). However, Meng et al (2006) do not mention why agents are best suited for this job. The following sections give a qualitative justification of why agent-oriented approaches are well suited to engineering complex control systems. The following sections also give a definition of an agent and explain how their properties can solve the control problems in manufacturing.

2.3.1 Characteristics of agent based control

There exist a number of definitions for agents. For instance, Monostori et al (2006) defines agents as a computational system that is situated in a dynamic environment, and is capable of exhibiting autonomous and intelligent behaviour, while Jennings and Bussman (2003) define an agent as follows: "An agent is an encapsulated computer system that is situated in some environment and can act flexibly and autonomously in that environment to meet its design objectives". However, Bellifemine et al (2007), note that all definitions agree that an agent is essentially a special software component that has autonomy that provides an

interoperable interface to an arbitrary system and/or behaves like a human agent working for some clients in pursuit of its own agenda.

Key words which elaborate agents are: autonomy, social, reactive and proactive. Bellifemine et al (2007) explain that agents are:

i. Autonomous because they operate without the direct intervention of humans or others and have control over their actions and internal states.
ii. Social because they cooperate with humans or other agents in order to achieve their tasks.
iii. Reactive because they perceive their environment and respond in a timely fashion to changes that occur in the environment.
iv. Proactive because they do not simply act in response to their environment, but they are able to exhibit goal-directed behaviour by taking initiative.

Monostori et al (2006) further highlight that four basic properties of an agent which are suitable to manufacturing systems control are:

- They are able to make observations about their environment.
- They have their own knowledge and beliefs about their environment.
- They have preferences regarding the state of the environment,
- They initiate and execute actions to change the environment

In manufacturing systems, complexity of a system often takes the form of a hierarchy. A major component of using agent based computing to solve such a problem is the decomposition of the problem into various autonomous entities (Jennings and Bussman, 2003). By decomposing the problem, a complex system is simplified in two ways and these are: firstly, it gives a natural representation for complex systems that are invariably distributed, which is typical of reconfigurable assembly systems; secondly, due to devolution of actions to autonomous entities, the actions performed by these entities (or agents) can be said to be responsive to the agents actual state of affairs, rather than some external entities perception of this state (Jennings, 2000).

Monostori et al (2006) state that, agents are individual problem-solvers with some capabilities of sensing and acting upon their environment, for deciding their own course of action, as well as for communicating with other agents. Depending on the actual problem and available technology at hand, agents can apply various faculties of problem solving, including searching, reasoning, planning, and learning (Monostori et al, 2006).

2.3.2 Agents versus objects

Although there are certain similarities between object-oriented and agent-oriented approaches to software engineering of complex systems, for instance both adhere to the principle of information hiding and recognize the importance of interactions (Jennings and Bussman, 2003), there are however fundamental differences between them and hence one approach is favoured above the other. The following paragraph outlines some of the differences.

Jennings and Bussman (2003), notes four differences between agents and objects.

- Firstly, objects are generally passive in nature, which means they need to be sent a message before they are active.
- Secondly, that although objects encapsulate state and behaviour realization, they do not encapsulate behaviour activation [action choice]. Although any object can invoke any publicly accessible method on any other object and the corresponding actions are performed, but objects do not initiate action by their own accord.
- Thirdly, object orientation fails to provide an adequate set of concepts and mechanisms for modeling complex systems. Recognition of these facts led to the development of more powerful abstraction mechanisms such as design patterns, application frameworks, and component-ware.
- Finally, object-oriented approaches provide only minimal support for specifying and managing organizational relationships (relationships are defined by static inheritance and hierarchies).

Agents also address autonomy and complexity. They are adaptive to changes and disruptions, exhibit intelligence and are distributed in nature (Monostori et al, 2006). They also may have an environment that includes other agents. The community of interacting agents as a whole operates as a multi-agent system (Monostori et al, 2006).

Bruccoleri (2007) mentions object-oriented modeling techniques as being widely proposed in scientific literature for the conceptual modeling phase of the control software development, because of their well-recognized features related to software modularity, rapid prototyping and re-use. The author, however, raises two concerns:

- The gap which exists between the object oriented conceptual model or design of the control software and its actual implementation. Unlike PC-based control software, PLC based control systems, for instance, do not have object-oriented features.
- Unconditional need of a simulation environment to test the effective operation of the new or the reconfigured control software to avoid unwanted effects (Bruccoleri, 2007).

On the other hand, Bellifimine et al (2007) note that multi-agent applications are in general quite complex. Every agent is composed of a single execution thread and all its tasks are modeled and can be implemented as behaviour objects (Meng et al, 2006). Furthermore, the patterns and outcomes of interactions are inherently unpredictable and therefore predicting the behaviour of the overall system on its constituent components is extremely difficult (Jennings, 2000).

2.3.3 Agent behaviour and interactions in JADE

A number of agent platforms exist which provide developers with support for programming and running agent applications. Examples of such platforms

includes: JADE, FIPA-OS, AGLOBE, MADkit or JACK (Vrba, 2012). Padgham and Winikoff (2004) classify agent platforms into, firstly, those that are optimized for agent reasoning and the development of agent plans, goals, etc. and, secondly, agent platforms that focuses (optimized) on inter-agent communication and provides means for the transfer of messages between agents. Examples of the former include PRS, JACK, JADEX, etc., while JADE and Zeus are examples of the latter. Grasshopper and Aglets are examples of agent platforms that focus on agent mobility.

The developer who wants to implement an agent-specific task should define one or more behaviours (Meng et al, 2006). In JADE, a behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends *jade.core.behaviour.Behaviour*. In order to make an agent execute the task implemented by a behaviour object, the object must be added to the agent by means of the *addBehaviour()* method of the agent class (Bellifemine et al, 2007).
For the JADE platform, three basic types of behaviours exist namely: 'One-Shot', 'Cyclic', and 'Generic' behaviour. Bellifemine et al (2007) explain that: 'One-Shot' behaviours are designed to complete in one execution phase and their *action()* method is executed only once; 'Cyclic' behaviours are designed to never complete and their *action()* method executes the same operations each time they are called; 'Generic' behaviours embed a status trigger, execute different operations depending on the status value, and complete when a given condition is met.

Furthermore, Bellifemine et al (2007) state that JADE also provides the possibility of composing behaviours together to create complex behaviours. The complex behaviours found in JADE are *ParallelBehaviour*, *SequentialBehaviour* and *FSMBehaviours* (Finite State Machine Behaviour).

Agents need to interact in order to achieve their intended objective. During their interactions, coordination protocols are used in order to reach common decisions. The Foundation for Intelligent Physical Agents (FIPA), an IEEE computer society standards organisation that promotes agent-based technology and interoperability of its standards with other technologies (FIPA, 2012), specifies standard interaction protocols which can be used as standard templates to build agent conversation. These protocols are: FIPA-Request, FIPA-query, FIPA-Request-When, and FIPA-Contract-Net (Meng et al, 2006). The FIPA website gives more details on the interaction protocols. The formulation of FIPA was inspired by the need for interaction between agents and led to the development of standards for agent development and communication.

2.3.4 Agent communication

Agents are fundamentally a form of distributed code processes and thus comply with the classic notion of distributed computing model comprising of two parts: components and connectors. Components are consumers, producers and mediators of communication messages exchanged via connectors (Bellifemine et al, 2007).

The most expressive model of an agent and its knowledge about the surrounding environment is the BDI model. Monostori et al (2006) state that the model assumes the agent has both certain and uncertain knowledge (Belief represented by B), regarding the state of its environment and also that states to be achieved are expressed in terms of goals, while states preferred in the long-term are represented by desires (D). Decisions concerning the future events have motivations and pre-arrangements in the past; these are expressed by the so called intentions (I) that represent the commitments of the agent made previously.

It is on the BDI model that the theoretical basis for agent communication language (ACL) is based. ACL was developed by FIPA based on the speech act theory (Monostori et al, 2006). Speech act theory views human natural language as actions, such as requests, suggestions, commitments and replies. It uses the term performative to identify the intended meaning of utterances, for instance verbs like request, promise, tell, etc. The first ACL was the Knowledge Query and Manipulate Language (KQML) that included many performatives, assertives and directives which agents use for telling facts, asking queries, subscribing to services and/or finding other agents (Monostori et al, 2006).

Effective communication between agents requires consensual knowledge. Consensual means that the whole community of agents has a common understanding both on the content and form of the expressed knowledge. This requires an explicit specification of the conceptual structures of a given domain called ontology. Ontologies can also facilitate machine processing, automated reasoning, as well as the interoperability of different agents (Monostori et al, 2006).

FIPA, despite having its own “language” called FIPA Semantic Language (SL) and three other subsets, does not prescribe a particular “language” to be used along with the communicative acts specified in the standard. The three subsets of FIPA SL (SL0, SL1 and SL2) differ in terms of which operators are supported. A FIPA-SL content expression may be used as the content of an ACL message (Bellifemine et al, 2007).

2.3.5 Directory facilitator

The directory facilitator (DF) is a specialized agent in the JADE platform which provides a “yellow pages” service to other agents within the platform. Agents can register (publish) services, deregister, modify and search for (discover) services in the “yellow pages” at any time during their lifetime (Bellifemine et al, 2007).

In accordance with FIPA agent management specification, every FIPA-complaint platform, like JADE, should host a default DF agent. Furthermore, Bellifemine et al (2007) state that other DF agents can be deployed if needed, and together with the default DF agent, can be federated to provide a single distributed yellow pages catalogue.

Since the DF is an agent, it is possible to interact with any other agent by exchanging ACL messages. JADE provides the *jade.domain.DFService* class with which it is possible to publish and search for services using a variety of method calls (Bellifemine et al, 2007). For instance, in order to publish services, agents must provide the DF with the service type, service name, the languages and ontologies needed to use the service and a collection of service-specific properties in the form of key-value pairs, and the *DFAgentDescription*, *ServiceDescription* and *Property* classes found in *jade.domain.FIPAAgentManagement* package represent these abstractions (Bellifemine et al, 2007).

In order to search for a service, an agent must provide the DF with a template description. The result of the search is a list of descriptions matching the provided template. A description matches the template if all the fields specified in the template are present in the description with the same values (Bellifemine et al, 2007).

2.3.6 Application of agents to control of manufacturing systems

There are a number of applications of agents in the control of manufacturing systems. Few are applied in real industrial environments while the majority, are proof-of-concept and trials established in laboratory conditions. For instance, shop floor components including two assembly robots, automatic warehouse, and a transport system are controlled by agents organized according to CoBASA architecture (Candido and Barata, 2007).

Brennan and Fletcher (2002) describe a distributed intelligent control system that is inherently adaptable and dynamically reconfigurable based on object-oriented and agent-oriented methods. Meng et al (2006) describe the development of agents for reconfigurable assembly system (RAS) using JADE agents. Vrba et al (2011) give a detailed analysis of research efforts towards realisation of an industrially accepted agent based control architecture. They also show how Rockwell Automation worked to integrate agents with PLC legacy control architectures by devising an holonic agent architecture comprising a low-level control, to process real-time data from sensors and actuators and a high-level control embodied by the software agent. Bussman and Child (2007) used agent technology to design a production system to meet rapidly changing operations in a factory plant of DaimlerChrysler used for production of cylinder heads.

Almeida et al (2010) state that common areas for the application of MAS in manufacturing operations include: when a real-time control of high-volume and discrete manufacturing operations are needed; when monitoring and control of physically highly distributed systems is needed; also when there is a necessity of information sharing and collaborative decision making between local autonomous units. Other areas of application are in transportation and material handling systems, production management of frequently disrupted operations, coordination of organisations with conflicting goals and frequently reconfigured environments (Almeida et al, 2010). When agent based control is the solution, they bring

robustness, flexibility, reconfigurability, redeployability and interoperability (Almeida et al, 2010).

However, there are challenges that agent based control systems face. Almeida et al (2010) highlight the challenges as security of agent execution and communication, complexity of the system, low level of scalability due to limitation in computational processing capabilities, and human-machine integration. Similarly, Wooldridge and Jennings (1999) highlight pitfalls in using agents based solutions as: assuming that an agent application solution for one testbed would solve all related problems, that agents can solve it all without use of other technologies, such as object-oriented technology, forgetting that agents are multithreaded and fail to plan for such things as synchronisation, mutual exclusion for shared resources and deadlocks.

2.4 IEC 61499 standard in distributed control

2.4.1 Introduction

The IEC 61499 standard was proposed by the International Electrotechnical Commission to design distributed control applications as well as the corresponding execution environments (Khalgui et al, 2011). The standard defines function blocks (FBs) as the main function encapsulation. In the IEC 61499 standard there is no global data and indirect data access is available. This implies that a FB can be developed and tested independently from the control devices and from the application they are used in (Lepuschitz et al, 2011). Lepuschitz et al (2011) state that this greatly increases reusability and further eases reconfiguration as the impact of changing or replacing a FB can directly be derived from the elements it is connected to.

A function block is an event-triggered software component composed of an interface and an implementation, such that the interface contains inputs and outputs of both data and events, and interacts with the environment using the same. Figure 2.5 shows a basic function block with the BOOL data type assigned to data input and output.

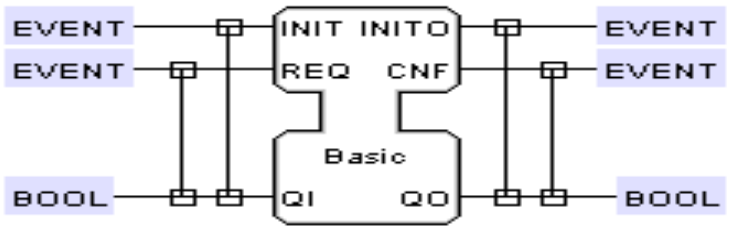


Figure 2.5 Basic function block

Each function block contains an algorithm and an execution control chart (ECC). By using both data and events, algorithms are executed when triggered by event inputs, by reading from data inputs and producing new data outputs.

The IEC 61131-3 standard currently used in PLCs is a predecessor to IEC 61499 standard. There are some drawbacks in the IEC 61131-3 standard which necessitated the introduction of the IEC 61499 standard. Rooker et al (2007) list some of the major drawbacks found in IEC 61131-3 standard which have been addressed in IEC 61499 standard as: the non-deterministic switching points in time (due to the cyclic execution policy), lack of fine granularity (i.e. reconfiguration at task level), jitter effect (i.e. task reconfiguration affecting other tasks) and the possibility of inconsistent states (which may lead to deadlocks). Lepuschitz et al (2011) also state that at the time of developing the IEC 61499 standard, the focus was much on HMS research; hence adaptability and reconfigurability were the main focus. However, dynamic reconfiguration is beyond the scope of this standard. For dynamic and real-time constrained reconfiguration, this interface is not sufficient, and an improved infrastructure is needed (Lepuschitz et al., 2011).

2.4.2 Execution environments for IEC 61499 function blocks

There are a number of execution environments for IEC 61499 function blocks. Hall et al (2007) list execution environments as: the function block run-time (FBRT) developed by James H. Christensen; the distributed controller operating system (DCOS), a fully functional distributed real-time operating system, developed by the University of Calgary; and Archimedes execution environment.

The first implemented IEC 61499 execution environment is the FBRT (Holobloc, 2012). It is implemented in Java and the IEC 61499 elements are presented as Java classes. DCOS provides services for integrated network management and location of transparent distributed services, while Archimedes' execution environment, which has two implementations: one designed for Linux and coded in C++, and the other implemented in Java targeting an enhanced Java virtual machine (Hall et al, 2007).

Additionally, there are a number of toolsets for function block design. The function block development kit (FBDK) is the most often used as it is the oldest and free for educational use (Black and Vyatkin, 2009). Black and Vyatkin (2009) also notes that commercial tool support is beginning to emerge and the example sited is the new version of ISaGRAF industrial design software which supports IEC 61499 function blocks.

For the FBs to become executable on a variety of hardware, hardware vendors must provide support for the standard. Currently, platforms which execute FBs are those which execute standard Java byte code. Therefore, FBRT can be used on such platforms. Examples of hardware where FBRT can run include desktop computers and PCs (Black and Vyatkin, 2009).

2.4.3 Deployment of IEC 61499 standard

Hussain and Georg (2007) in their work identify three factors that have to be considered before deploying the IEC 61499 standard. These are:

- Resource constraints: a distributed real-time application can be constrained by memory, utilization factors and network usage.
- Allocation constraints: the system architecture can impose the following constraints: residence; restricting deployment of software components on a subset of available hardware; co-residence i.e. forcing that certain components are to be placed on the same processing node; and exclusion, which is the opposite of co-residence and inhibits co-existence of software components.
- Time constraints: this is the most important constraint and is usually stated in terms of deadlines in the case of periodic tasks, or in terms of end-to-end response times for event-driven tasks.

For its implementation in industry, the IEC 61499 standard will have to overcome a number of challenges. Hall et al (2007) identify scalability, maintainability, predictability and extensability as some of the challenges. They argue that the focus of most reasearch has been on developing basic control programs with small number of devices, while a typical industrial application has a higher number of devices to be controlled. While the research community's primary focus is on how to develop and validate the initial control, Hall et al (2007) argue that maintaining the control system over the life of the system is a much larger challenge. Similarly, Hall et al (2007) highlight diagnostics for both the control devices and user's process as another corncern. " The challenge for IEC 61499 function blocks is the need to display both the execution sequence and data flow, since unlike scanned systems each FB's execution is controlled by event system" (Hall et al, 2007).

While IEC 61499 function blocks may enable faster typical response time over traditional scan-based IEC 61131-3 systems, Hall et al (2007) state that to predict the worst case response time may be a difficult task. Similarly, the need to extend an already existing control automation system also requires that the new process or control engineer understand the programs before modifying them. Therefore, "much effort is needed in the development of tools for debugging, operation, and maitenance of these [IEC 61499] systems" (Hall et al, 2007).

2.5 Evaluation criteria for control strategies

The core of the RMS paradigm is an approach to reconfiguration based on system design, combined with the simultaneous design of open-architecture reconfigurable controllers, having reconfigurable modular machine modules designed by synthesis of motion modules (Koren et al, 1999). Therefore, the ultimate goal of the RMS is the utilization of a system approach in the design of the manufacturing process that allows simultaneous reconfiguration of the entire system, machine hardware and control software (Koren and Shiptalni, 2010; Koren et al, 1999). Reconfiguration in structure, hardware and software are therefore some of the key areas for RMS.

Substantial research has been conducted in RMSs, for instance reconfiguration at structural level has been investigated by Koren and Shiptalni (2010). They looked at a number of possible reconfigurations for a given RMS. In hardware, Bi et al (2008) identified RM as the hardware systems at machine or device level. Furthermore, machine modules in RMS should have defined interfaces in: mechanical (e.g connectors, fasterners), power (hydraulics, pneumatics, electricity) and informational or control (control network) (Koren and Shiptalni, 2010; Bi et al, 2008; Koren et al, 1999). Additionally, technologies in hardware and software have been identified as reconfiguration enablers both at structural and hardware levels. In hardware, modular machine tools aiming at giving the customer more machine options, while in software, modular and open-architecture controls that aim at allowing reconfiguration of the controller (Koren et al, 1999).
For modularity to be supported in software, the control system must be based on the principles of an open architecture. IEEE defines open architecture as: "an open system providing capabilities that enable properly implemented applications to run on a wide variety of platforms from multiple vendors, inter-operate with other system applications, and present a consistent style of interaction with the user" (Pritschochw et al, 2001; Pritschow et al, 1993). Therefore, the overall emphasis in software is to enhance reconfiguration.

In this research therefore, evaluation of the control stategies at the HLC layer is based on establishing the properties of each control strategy to enhance the six core charactersistics of RMS. The six core characteristics are: customisation (flexibility limited to parts), integrability (interfaces for rapid integration), convertibility (design for functionality changes), modularity (components are modular), diagonisability (design for easy diagnostics) and scalability (design for capacity changes.

2.6 Conclusion

Agent based control has dominated research in the development of reconfigurable manufacturing systems, including areas such as production planning, resource allocation, distributed material-routing control etc. Other control strategies, such as the IEC 61499 standard, can still be applied depending on a given context, but the agent based approach to control of manufacturing systems appears to be more advantageous. Advantages include the decomposition of a complex control problem into small distributed autonomous entities capable of making their own decisions, while collaborating with others to meet certain goals. Agent based control has been used predominantly as a high-level control (HLC) layer, while IEC 61499 function blocks have been used as low-level control (LLC) layer.

3. CASE STUDY

The assembly system considered in the research work is a welding assembly cell. The concept was first developed by Sequeira (2008) for fixture-based reconfigurable spot welding. This section looks at the welding assembly system with its subsystems and particular attention is given to the design of a modular Cartesian robot. Section 3.1 describes the subsystems that make up the assembly cell and the part family considered for research, while Section 3.2 details the mechanical design of the modular Cartesian robot. Except for the cell controller and Cartesian robot, the assembly cell was developed by other researchers (Kruger, 2013; Le Roux, 2013). The development of the cell controller and the controller for the Cartesian robot are described in Chapters 5 and 4, respectively.

3.1 Assembly system overview

The welding assembly cell constructed at Stellenbosch University consists of six workstations i.e.: a pallet magazine, a Bosch Rexroth TS2 Plus conveyor, a feeder, a welder, an inspection station, and a removal station. Each subsystem takes a specific role in order to produce a circuit breaker and is organized around the conveyor as shown in Figure 3.1 and Figure 3.2.

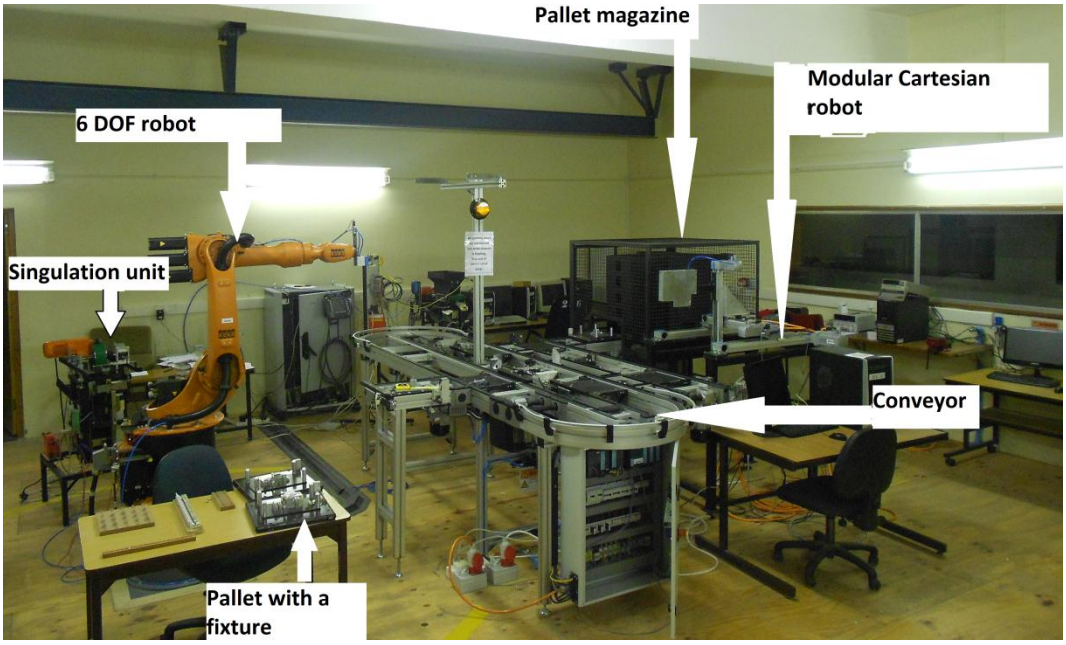


Figure 3.1 Weld assembly cell overview

The conveyor layout consists of a central round robin main loop with modular in-feed and out-feed conveyor units. The in-feed and out-feed from the conveyor units form outlets to subsystems. The outlets are used to convey a pallet with a fixture to the subsystem used in the production of the circuit breakers. Furthermore, each workstation is assigned a number which the conveyor uses to identify the station. These station numbers are used during communication when the product agent in the cell controller carries out its tasks. The tasks carried out

by the 6 degrees of freedom (DOF) robot are dual: it is used as the feeder station and is also used as the removal station.

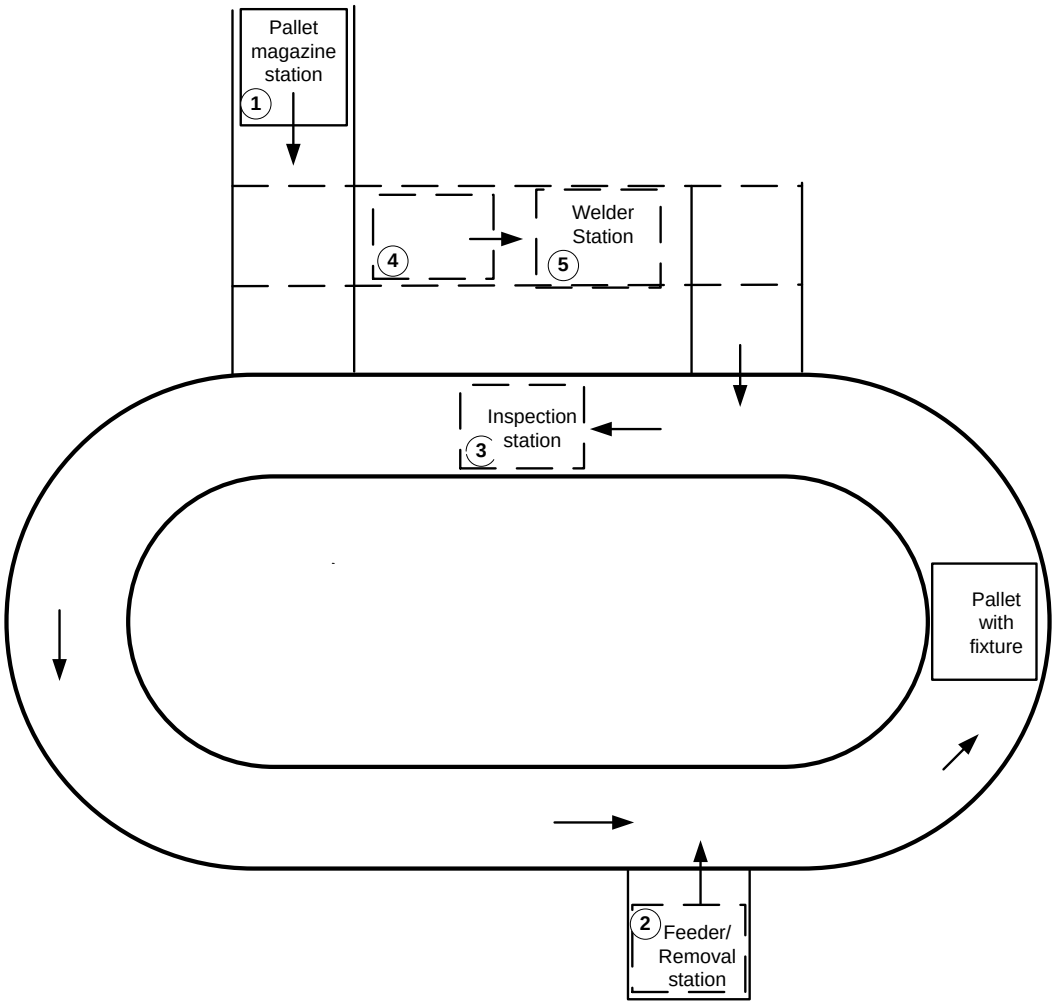


Figure 3.2 Welding assembly cell layout

Figure 3.2 shows the layout of the assembly cell with workstations for each subsystem around the conveyor. The pallet with a fixture, when offloaded from the pallet magazine, moves as indicated by the arrows in the figure to complete a production cycle. The following sections gives details of what happens at each station and control related issues.

3.1.1 Conveyor subsystem

The role of the conveyor is to take a pallet with a fixture to the workstations when requested. The request can come from the cell controller as discussed in Chapter 5 or direct from the operator. In order to manage traffic and diagnose errors, the conveyor has an RFID system with a number of sensors connected to an AS-i network using a PROFIBUS cable for communication.

By using the in-feed and out-feed conveyor units, shown in Figure 3.2, the conveyor can transport a pallet with a fixture to every station when commanded. Furthermore, the dimensions of the pallets used for different products are the same; therefore, no reconfiguration on the conveyor is needed when a new product is launched. However, when the workstations are changed around the conveyor, the cell controller has to be reconfigured accordingly to ensure consistency in messages and understanding between cell controller and the conveyor.

Messages sent between the conveyor and the cell controller has a pattern for the two systems to understand each other. Appendix D.2 gives the message format used between the conveyor and the cell controller.

3.1.2 Pallet magazine station

The pallet magazine station stores pallets, each with a fixture, needed for production. It has three magazines to store the pallets for different products. In order to offload or load a pallet, a command can be issued by the cell controller and the pallet is offloaded and loaded accordingly.

The pallet magazine controller is linked to the cell controller through the TCP/IP connection. Additionally, the pallet magazine and the conveyor exchange messages in order to synchronize the offload and loading of the pallet. The messages exchanged affect the cell controller control program and are addressed in Chapter 5.

3.1.3 Feeder station

The pallet with a fixture, unloaded from the pallet magazine, is placed on the conveyor and transported to the feeder station. At the feeder station, which comprises a 6 DOF robot, a singulation unit and part magazine, the circuit breaker components are placed on the fixture. For the robot to place parts on the fixture, it needs the pick-up coordinates for each part, as well as the coordinates of where to place the part on the fixture. Part coordinates are given by the singulation unit, while fixture coordinates are given by the cell controller. The singulation unit uses a vision system to detect a part and the coordinates. A typical command from the cell controller would request for a part and give coordinates of where to place the part on a fixture.

3.1.4 Inspection station

The inspection station plays two roles during the production cycle. Firstly, the fixture is inspected for the presence of parts and, secondly, to check for defects and proper welding of the parts. In both cases, the inspection station informs the cell controller of the test results, thus allowing the cell controller to decide on the appropriate action. Through the same messages used to inform the cell controller, the numbers of finished products are counted. The inspection station uses a vision system to detect parts.

3.1.5 Removal station

The role of the removal station is to remove welded parts from the fixture. Removing a welded part is done by the 6 DOF robot (Figure 3.2). These welded parts can be removed for either rework or for packaging as a finished product. The cell controller is responsible for instructing the 6 DOF robot to remove the product and, after successfully removing the product, the robot in turn informs the cell controller to take the pallet with a fixture for either re-use in the production or to storage in the pallet magazine.

3.1.6 Welding station

At the welder station, the components are simulated to be welded together on five points as shown in Figure 3.3. The work envelope for the weld station is 480 x 380 x 280 mm. Products to be welded at this station vary considerably, but fall within the work envelope. Therefore, parameters such as the clamping force, weld current and weld positions have to be given each time there is a product change. This requirement affects how the control program is implemented and is tackled in Chapter 4.

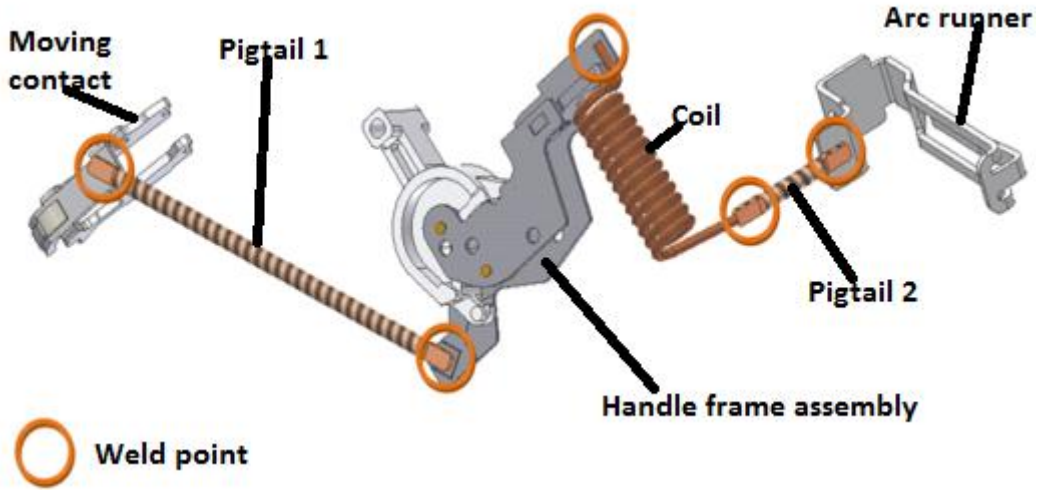


Figure 3.3 Components of a circuit breaker with welded points

3.1.7 Part family

There are a number of circuit breakers that can be produced at the weld station. One of the product families considered here is the Q-frame. The Q-frames differ in the sizes of their pigtails, among other things. For instance, the sizes of the two pigtails, shown in Figure 3.3, range from 10 mm to 60 mm long. Their diameters range between 2.5 mm and 4 mm, while the sizes of the moving contact and arc runner are 27 x 8 x 12 mm and 42.6 x 9.8 x 18.4 mm respectively.

These part variations have to be accommodated at each workstation. Therefore, at the design of the weld assembly cell, each station takes care of the variations.

3.2 Design of a modular Cartesian robot

The modular Cartesian robot was designed on the principles of reconfigurable machines (RMs), that is, modular structure and software components. The challenges to developing RMs, as cited by Bi et al (2008), are: developing one that takes into consideration the requirements of changes and uncertainties for a specific part family, and to have a control program that is not dedicated to a specific product. Figure 3.4 shows a rear view the modular Cartesian robot developed in this research.

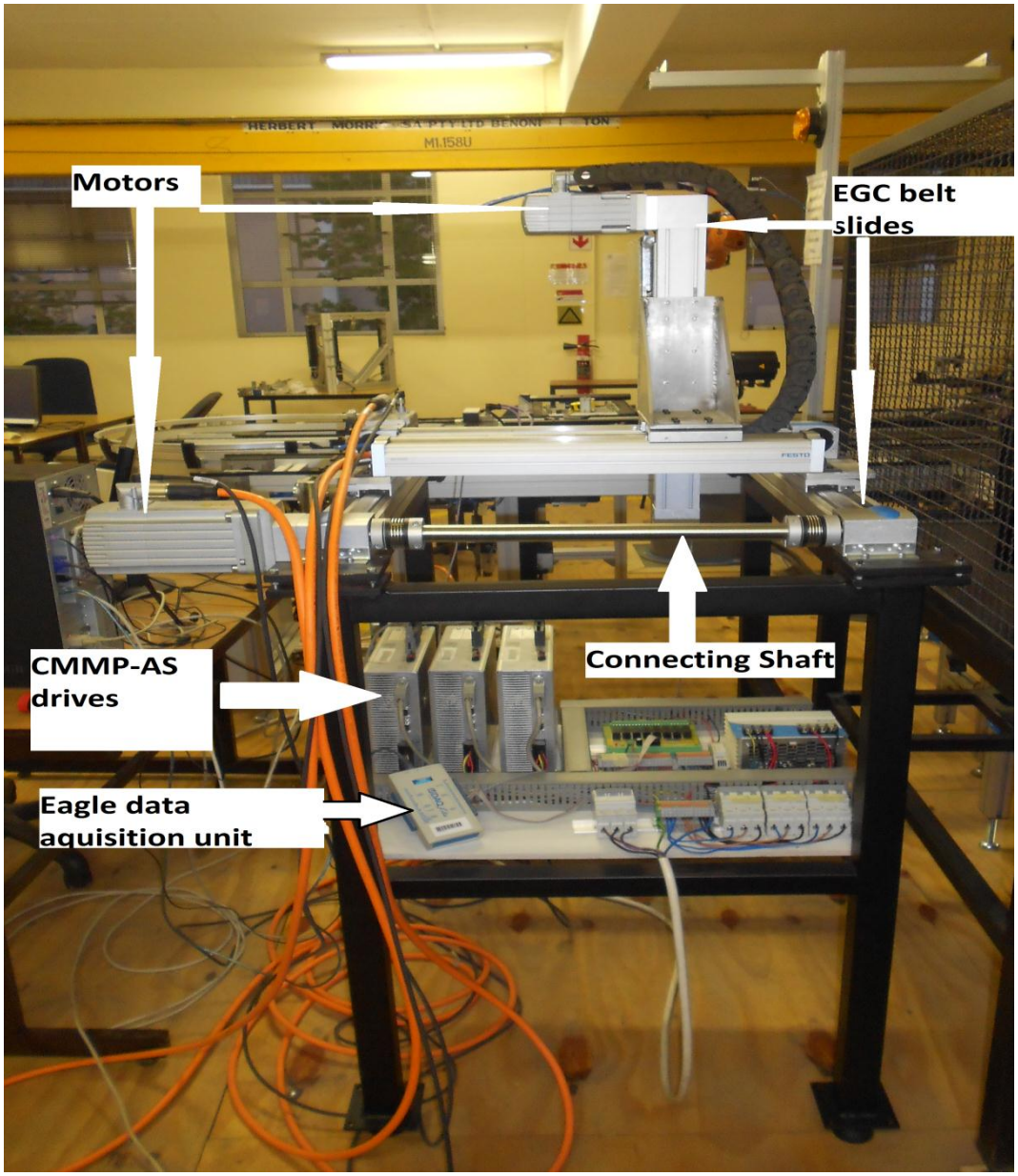


Figure 3.4 Rear view of modular Cartesian robot

There are many robots in industry that can be used for spot welding, for instance articulated robots, spherical robots, SCARA robots, cylindrical robots and Cartesian robots, classified according to their geometry. In this research, the choice of the weld robot was dependent on the following things: work envelope, geometry, use of different hardware and software vendor technologies and most importantly, facilitation of the use of IEC 61499 function blocks and agent based control in the control of the robot. Since the weld robot's required work envelope is rectangular and funding for a set of Festo linear drives was available, a Cartesian layout was chosen for the weld robot. Further, the inherent modularity when using Festo linear drives gives the potential of reconfiguration, that is not present with other robot geometries, and provides the opportunity to divide the controller into modules too.

In order to design the hardware and control program for the modular Cartesian robot, a functional analysis was done as shown in Appendix C.2. The axes of the modular Cartesian robot were designed using EGC belt drive axes from Festo (FESTO, 2012a). Details of each axis design are explained in the following sections.

The modular Cartesian robot has three degrees of freedom, which were sufficient for this research. However, a fourth axis (rotation about the vertical axis) can be added later, if required. Further, only point-to-point movement of the weld robot was required. Closed-loop control of each axis of the robot is provided by the servomotors' drives. By using Festo configuration tool (FCT) provided by Festo, parameters can be adjusted to meet the requirements for a given control. The "profile position control" option in FCT was used. To determine when a commanded motion has been completed, the motor drives were set up to give a digital signal when the "remaining distance" parameter was lower than a threshold value. Coordination and movement of axes is explained in Chapter 4. Repetition accuracy of the drives is ± 0.08 mm (FESTO, 2012) and was sufficient for reconfiguration investigation purposes.

3.2.1 X-axis hardware selection and configuration

The X axis is made of two EGC-80-500-TB-KF-OH-GV belt slides arranged in parallel whose size and length are 80 mm and 500 mm respectively. The two EGC belt slides are then coupled by a connecting shaft to synchronize the motion of the two slides (consider Figure 3.5).

Selection of the EGC belt slides was motivated by the cost considerations, loading forces, bending moments, work envelope, accuracy, repeatability and serviceability of the slide. The axis is designed to carry the Y and Z axes plus the weld head. It is mechanically linked to the Y axis by two metal pads screwed on the two parallel oriented X axis slides (consider Figure 3.4). Mechanical interfaces give the structure modularity needed during reconfiguration and it is also a crucial requirement for RMS (Koren and Shiptalni, 2010).

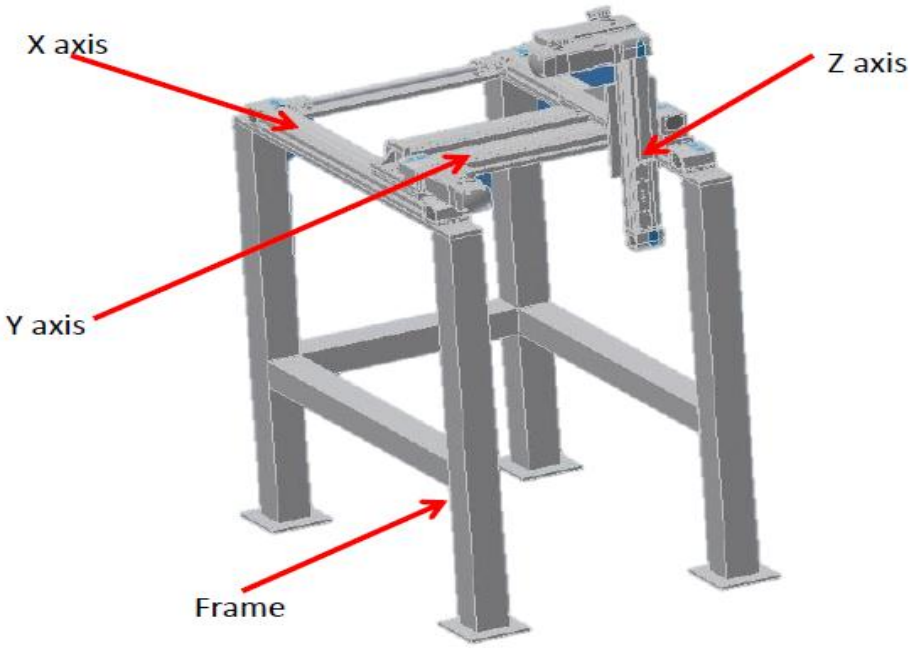


Figure 3.5 Mechanical structure of modular Cartesian robot

The X axis, with two parallel EGC slides coupled by a shaft, is driven by a CMMP-AS motor controller connected to an EMMS-AS-100-S-RM motor. The controller is powered by a 24V DC power supply and wired as shown in Appendix B.2. The motor controller has a number of control interface options, such as: digital I/O, Cable Area Network open protocol (CANOpen), DriveBus, RS-485, synchronization and analogue input. Not all the control interfaces have been used and Chapter 4 gives the control selection criteria for the control interface used. Connection to the mains supply is as shown in Appendix B.1. When the robot is not powered, the slides move freely, since the X axis motor has no brakes but are only engaged during operation.

To allow homing before operation or after reconfiguration, a proximity sensor is installed on one slide of the axis. The sensor is normally open. However, when closed, it sends a signal to the motor controller thus indicating a successfully homing.

3.2.2 Y-axis hardware selection and configuration

The Y axis is made from an EGC-80-400-TB-KF-OH-GK belt slide and an EGC-80-400-FA-GK guide axis. Their sizes and lengths are 80 mm and 400 mm respectively. Since the Y axis carries the Z axis and the weld head, one slide could not balance the mass of the weld head and inertial forces during motion and the Z

axis mass (see Figure 3.5). Therefore, EGC-80-400-FA-GK is used for balancing the structure and the Y axis.

With the structure connected as aforementioned, the EMMS-AS-70-M-RM three phase motor is then used to drive the Y-axis. Homing is done using a normally open proximity sensor during, after and before operation of the axis.

3.2.3 Z-axis hardware selection and configuration

The Z axis is an EGC-70-300-TB-KF-OH-GV belt slide with size and length 70 mm and 300 mm respectively. Unlike the other two axes, the motor for the Z axis has brakes to hold the weld head in position. The CMMP-AS is the controller used to control the controller, while an EMMS-AS-70-S-RMB motor with 11 kW braking power drives the axis.

4. RECONFIGURABLE CONTROL OF MODULAR CARTESIAN ROBOT

This section expounds on the control strategies applied on the modular Cartesian robot. The control strategies include the use of agents and IEC 61499 function blocks. They are applied on the modular Cartesian weld robot at high-level control (HLC) as alternatives to each other, while a Visual C# program is used as a low level control (LLC) layer. The CANOpen protocol DS 402 is also used.

The two control strategies that were compared have two different architectural philosophies. The IEC 61499 standard is an event-driven architecture, while each agent runs in its own thread, thus agents require much computing resources. This translates into a set of hardware requirements. For instance, it is not possible to run the IEC 61499 standard on a Programmable Logic Controller (PLC), which is mostly used in industry, because an event-driven PLC is not yet on the market. Similarly, to run agents on a PLC is also not possible. Therefore, control of the robot involved the use of a Personal Computer (PC) as a standard platform for the comparison of the control strategies as they can all run on this platform.

In implementing the control for the Cartesian robot, a layered architecture was utilized (Xuemei, 2009). Two layers, namely low-level control (LLC) and high-level control (HLC), were used to allow the separation of concerns. The LLC was used for real-time data acquisition and the HLC for negotiation and coordination. By separating the layers, the influence of each layer is distinguished and makes trouble shooting easier since the sphere of influence is clearly defined. Furthermore, the approach makes the software modular and easier to reconfigure if there are any changes to be made to any of the control layers. The two layers in the modular Cartesian robot are linked by a port number as assigned in Appendix D.1

4.1 Low-level control strategy for modular Cartesian robot

4.1.1 Hardware and software considerations

The LLC is a Visual C# program with a TCP/IP server accepting connections from the HLC, parsing messages and using the Eagle data acquisition unit as an interface to the CMMP-AS motor controllers. The Visual C# program further reads digital inputs from the CMMP-AS motor controllers and also writes digital outputs to the motor controllers via an Eagle data acquisition unit.

There are various data acquisition units which can be used for this purpose. In this set up, however, the choice of the Eagle data acquisition unit was motivated by cost considerations and the use of digital input and output to activate the CMMP-AS drives. The unit can be easily connected or disconnected to a computer with universal serial bus port.

The Eagle data acquisition unit has eight digital inputs and eight digital outputs and is then connected to the computer using the Universal Serial Bus (USB-2) port. In order to communicate with the Eagle data acquisition unit, the Visual C#

program references a dynamic link library DLL. The DLL is supplied by the suppliers of Eagle data acquisition unit. An Application Programming Interface (API) is then instantiated in the Visual C# program making available all the functions available to the LLC program. The functions that read from or write to the data acquisition unit, requires the serial number of the unit and a port. To address a port, hexadecimal format is used. For instance to write to the port, the snippet of the code would be:

```
static EDREApi eagleCard =new EDREApi ();
static int wPort = 0; //
static int lembaDAQCard = 0;
int value = 0;
int val = 0x71;//  0111 0001
value = value | val;
eagleCard.SerialNumber = 1000009424;
lembaDAQCard = eagleCard.DIOWrite(wPort, value);
```

4.1.2 Coordination of axes in the LLC layer

Coordination of the three axes of the modular Cartesian robot is achieved in the LLC layer using Visual C# programme. In order for the LLC layer to effectively achieve coordination of the axes, the HLC layer must pass messages in a consistent manner to the LLC layer. Extensible markup language (XML) is used to pass messages to LLC layer because of the consistent manner in which messages are presented. Additionally, the LLC layer also utilizes some capabilities of servo drives in order to coordinate motion of the robot using digital input and output control interface.

In the servo drives, weld coordinates of the product are saved in the position set table of each axis. The weld coordinates can be changed when a new product is introduced by using a Festo configuration tool (FCT). The FCT is software supplied by Festo used to configure drives using RS 232 cable.

The coordinates once saved to the drives can be used for welding when the digital input is activated. Activation of the path is achieved by a rising edge of digital input command from HLC layer. Once in operation, the weld movement uses point-to-point motion.

4.1.3 Movement of the X axis

The C# programme activates the digital input to the servo drive after receiving the command from the HLC layer. At the rising edge of the digital input from the Eagle relay board, the command responsible for enabling the reading from the position set table using is activated.

The servo drive traces the path in the position set table and for each position reached, gives a digital output to the C# programme. The C# programme in turn sends a message to the Y-axis. Similarly, when the Y axis completes movement, sends signal back to the X axis.

4.1.4 Movement of Y, Z axes and the weld head

When the Y axis moves to a position as assigned in the position set table, it delays for 3 s at each position. The time delay can be changed by the programmer. This delay, however, allows the movement of Z axis carrying the weld head to a weld point to do their task.

In a similar manner, Z axis sends a signal to the weld head in order to weld. The weld action demonstrated by in the modular Cartesian robot use compressed air. When a signal is received from the Z axis, the 5/2 valve is actuated to open the valve. The whole weld cycle of the weld robot is carried out in twenty seconds.

4.2 CANOpen configuration

The CANOpen protocol is used with both the agent and the IEC 61499 standard, and was chosen to enable the passing of values to the motor controllers during operation. In this approach, it is assumed that the subsystem should not store the product information, but will receive details from the cell controller. Product information will be in the form of coordinates of weld points, speed of operation, etc.

To support the CANOpen protocol on Festo motor controllers, the following items were incorporated from Beckhoff: FC5101 CANOpen master PC interface card with 32kbytes of NOVRAM, ZB5100 CAN 4-core cable fixed laying (2 x 2x 0.25 mm^2), four 9-pin D-sub connectors integrated with 120 Ω termination resistors and TwinCAT I/O software. The FC5101 PCI card with 32kbytes of NOVRAM from Beckhoff is used as a master while the CMMP-AS motor controllers from Festo are the slaves.

Each motor controller was set to the CANOpen DS 402 protocol using Festo Configuration Tool (FCT). DS 402 is the only CANOpen protocol available in CMMP-AS motor controllers. Using FCT, each axis is assigned a node number and baud rate. In this application, a baud rate of 500kBits/s is used and is sufficient for the application data requirements. The node numbers assigned to X, Y and Z axis are: one, two, and three respectively. Since the X axis carries the other two axes, it is assigned node number one so that it has the priority of receiving the command on the CAN bus. However, for the operation to take place, the product information, as operation parameters, must be passed to the robot using process data objects.

4.2.1 Process Data Objects (PDOs) assignment

The CANOpen protocol provides a simple and standardized possibility to access the parameters of the motor controller. In order to achieve this, a unique number (index and sub-index) is assigned to each parameter. As a rule, the motor controller is parameterized and also controlled via Service Data Objects (SDO). For the fast exchange of process data (e.g. target position), it is possible to use Process Data Objects (PDOs). Each message sent on the CAN bus will then have to contain a type of address which is used to determine the bus participant for

which the message is meant. For this reason, CANOpen protocol is suitable for the fast exchange of data during welding operations.

Festo CMMP-AS motor controllers have four transmit PDOs (TxPDOs) and four receive PDOs (RxPDOs) (FESTO, 2012b). The difference between the two types of PDOs is: TxPDO sends PDO when an event occurs, while RxPDO evaluates PDOs when a certain event occurs from the controller and host side respectively. Each PDO has a CANOpen bus identifier (COBId), an index and sub-index to which they must be mapped. FESTO (2012b) gives details of the two types of PDOs used in the modular Cartesian robot design.

The default COBId numbers that come with motor controllers are identical (FESTO, 2012b). However, if the three motor controllers are on the same CAN bus; conflicts in communication may arise rendering communication impossible. To avoid this situation, the first two COBIds for each axis were assigned in hexadecimal format as follows:

- X axis has 181_h and 281_h for the first two TxPDOs while the first two RxPDOs have 201_h and 301_h.
- Y axis has 182_h and 282_h for the first TxPDOs while the first two RxPDOs have 202_h and 302_h.
- Z axis has 183_h and 283_h for the first two TxPDOs while the first two RxPDOs were assigned 203_h and 303_h.

To deactivate the default COBIds for Y and Z axes, the 31st bit was deleted and then new COBIds assigned. For instance, to delete a default 181_h COBId and replace it with 183_h, use C0000181 and to activate, write a new COBId as 40000183 (all in hexadecimal format). The other method that could have been used is the use of electronic data sheet (EDS) files supplied by Festo. This, however, was not the best route because not all PDOs are used in the project. Hence the need to select the PDOs needed for the project.

4.2.2 PDO selection

The control program for the modular Cartesian robot, among other requirements, needs to pass data objects from the robot controller to the motor controller during operation. This passing of data objects is done through selected process data objects (PDOs) as needed in the program, and depending on an application, the PDOs can be selected from the Festo manual.

In order to select the appropriate PDOs, the guide was based on the size of the PDO, whether that PDO is a read or write type and whether it is defined and supported by Festo. PDOs which were selected for the application requirements were: mode of operation, mode of operation display, target velocity, target position, actual position, actual velocity, control word and status word.

In CANOpen, the entire regulation of the motor controller is achieved through two objects: the host can regulate the motor controller through a control word,

while the status of the motor controller is read back in the status word. Similarly, for the robot to operate, it has to be "instructed" in what mode to operate. The Festo manual provides a mode of operation (with object number 6060_h) to command the controller to a given motion profile and these profiles include homing, position profile, etc. When the command is successfully sent, the feedback is given by modes of operation display (6061_h). The sizes of the two PDOs can are given in FESTO (2012b).

After homing the axes, each weld coordinate in the form (X, Y, Z), must be supplied by the robot controller to all the three motor controllers on the bus in order to carry out an operation. Target position ($607A_h$) and actual position are the PDOs used to give the coordinate positions and the position arrived at respectively. The speed with which the axis must run is provided by target speed, while the parameter, actual speed, gives feedback to the controller.

4.2.3 TwinCAT I/O software configuration

TwinCAT I/O is a software environment from Beckhoff Company. It was used to set up variables that were then linked to CANOpen PDOs through mapping. The mapped PDOs can then be accessed in the HLC software. Another benefit of using TwinCAT I/O software is that it is a PC based software environment, therefore, it suits the requirement for the underlying framework in the evaluation of control strategies. Furthermore, the two vendors (Festo and Beckhoff) can be integrated through CANOpen protocol support in TwinCAT I/O.

Accessing and hence communication to the variables from the HLC was made possible by using TcJavaToAds.jar file and the DLL (adsToJava.dll) supplied by Beckhoff. The TcJavaToAds.jar file has a set of predefined methods to interface with the variables created in TwinCAT I/O. Accessing the jar file in Eclipse IDE was done by adding the TcJavaToAds.jar library to Eclipse IDE and importing classes from this file into the program. With the CANOpen as the LLC layer, the IEC 61499 function blocks and agents were built on top of this layer.

4.3 IEC 61499 control approach

In this approach, function blocks are used as a high-level control (HLC) with the Visual C# program as low-level control (LLC). Each motor controller for each axis as explained in Section 3.2 is modeled in a *frame-device* of function block and the control architecture modeled within resources. Hence, three *frame-devices* were used to control the three axes at HLC. The design tool used in the development of function blocks (FBs) is the function block development kit (FBDK). The choice of the FBDK was motivated by the fact that it is free and mostly used in developing IEC 61499 function blocks.

The IEC 61499 standard has two types of resources, the panel and embedded resources. A panel resource was used in this control approach in order to display activities running in the background to the operator for diagnostic purposes unlike the embedded resources. IEC 61499 function blocks (FBs) further defines three classes of function blocks namely: basic FBs, composite FBs and service interface

FBs. A FB is a building block that encapsulates a behaviour. Like a state machine, the FB has an execution control chart (ECC) which defines the reaction of a FB to an event. The reaction can consist of an algorithm within the FB taking data inputs with events and internal variables and giving output data and events. In an IEC 61499 architecture, the function performed by the system is specified as an application, which may reside in a single device or be distributed over several devices (Vyatkin, 2007).

4.3.1 Design methodology

FBs are object-oriented software elements. Therefore, as in other object-oriented software development, a model-view-controller (MVC) was used. In the MVC pattern, the system to be controlled is first visualized and simulated, then the control is tested, and later the model is substituted by interfaces to the real plant (Hirsh et al., 2007). In this work, MVC methodology is used as a framework for implementing object-oriented principles by using IEC 61499 FB types as classes which are normally used in a typical MVC implementation. Figure 4.1 shows a layered architecture, with the five layers, used in the development of the control program for the modular Cartesian weld robot.

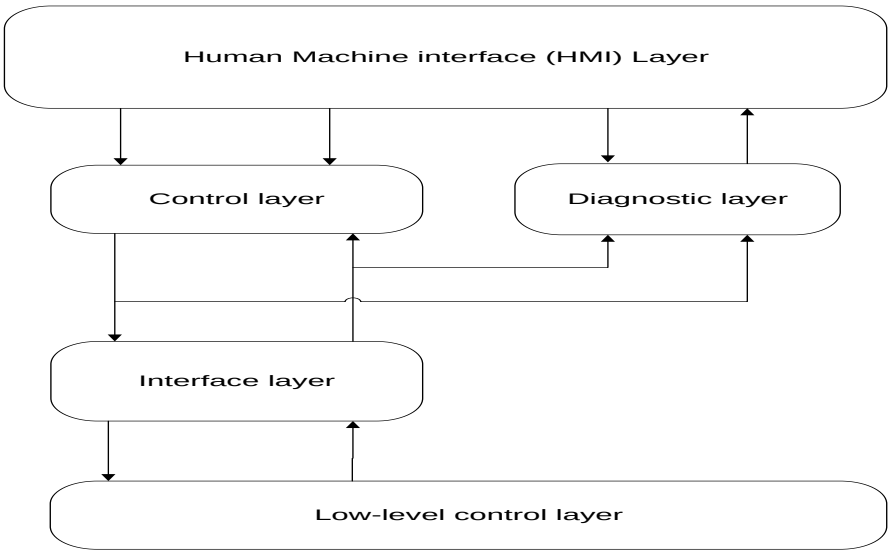


Figure 4.1 Layered architecture for control implementation of weld robot

The mechanism layer was implemented in Visual C# as the LLC layer and also as TwinCAT I/O as an alternative to Visual C#. Other layers are discussed in the following sections.

4.3.2 Human machine interface layer

The human machine layer provides a means for which manual operation of the weld robot is possible. The layer provides button and text fields which the operator can use. The layer is also used for diagnostic function blocks used to display error which would have otherwise happened.

In this layer, functionally similar elements for the X, Y and Z axes were identified from FBDK as: frame-devices and panel resources. These elements provide views where the user can interact with the program, unlike the remote-devices which do not have human machine interfaces. The frame-devices were used to model the system for each axis.

4.3.3 Control layer

In this control layer, message error checking is done to ensure consistent messages are passed between the cell controller and the HLC layer, and between the LLC layer and the HLC layer. Furthermore, message passing between resources is performed by layer, as well as message between the FB networks and other parts of the controller. Furthermore, the layer can be used for expanding the control programme when more axes are needed.

In the control layer, function blocks were embedded within panel resources and an application was formed using basic FBs, composite FBs and service interface FBs. For instance, to pass messages within an application, *publish* and *subscriber* service interface FBs from the net library of the IEC 61499 standard were used. *Server* and *client* FBs from the net library were also used to pass messages between the HLC to the cell controller. Interconnections of FBs were then combined in a composite function block. An example of a function block network in a composite FB is shown in Appendix C.1.

The function block responsible for the control of each axis was also developed. Figure 4.2 shows some of the FBs used in the control of the axis.

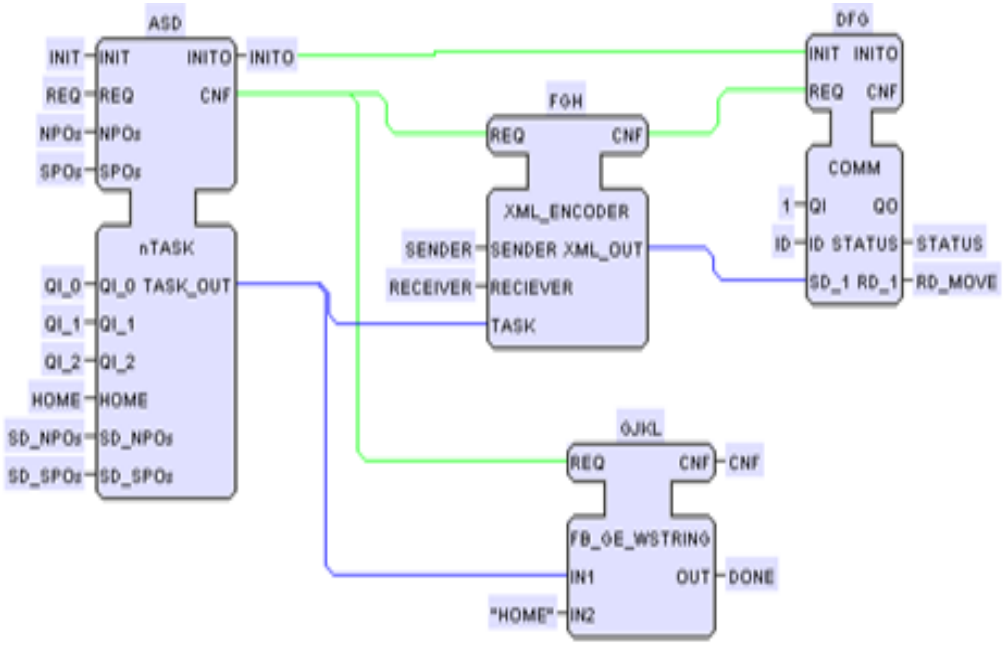


Figure 4.2 Composite function block for axis control

The nTask FB is responsible for commanding the axis to action. When the axis needs to home, the command is sent through the nTask FB. The home is encoded into XML format in the XML-ENCODER FB and then sent to the interface layer through the COMM FB. The XML encoding is used by the LLC layer to differentiate between which axes for which the command is intended.

To command all the three axes at once, *publish* and *subscriber* FBs are used. The *publish* FBs sends to all the axes, while *subscriber* FBs receives a message from the *publish* FBs. The two FBs can be found in the IEC 61499 standard library. To send a command from the operator to the axes, the HMI layer provides FBs through which a command is passed to the drives.

4.3.4 Interface layer

This layer is used for communication between the LLC layer and the HLC layer. Its primary purpose is to provide TCP/IP socket connection between the two layers and also handling of decoding of messages sent between them.

The layer is composed of the IEC 61499 function block shown and links with the LLC layer implemented as a Visual C# program. Figure 4.3 shows the function block used to interface HLC layer with the LLC layer.

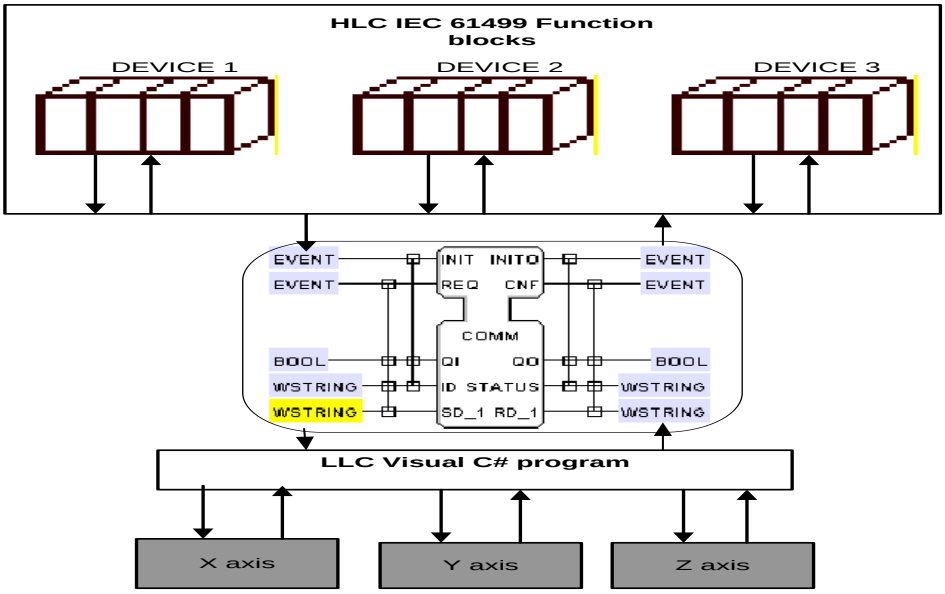


Figure 4.3 Interface between HLC and LLC using basic function block

FBs encode their messages using ANS.1 encoding. Therefore, the encoding must be understood by the LLC. However, this is not the case. Additionally, at the HLC, WSTRING format is used by service interface FBs of the net library to pass data to other FBs in the network of FBs. It is therefore imperative that the FB interfacing the LLC and the HLC, has its data input and output for receiving and

sending messages respectively in WSTRING format regardless of the aforementioned conflicts.

To solve the conflicts, an algorithm was developed and placed in the execution control chart (ECC) to determine how the function block will be executed when an event occurs. The algorithm in ECC is placed in the REQ state. Consider Figure 4.4

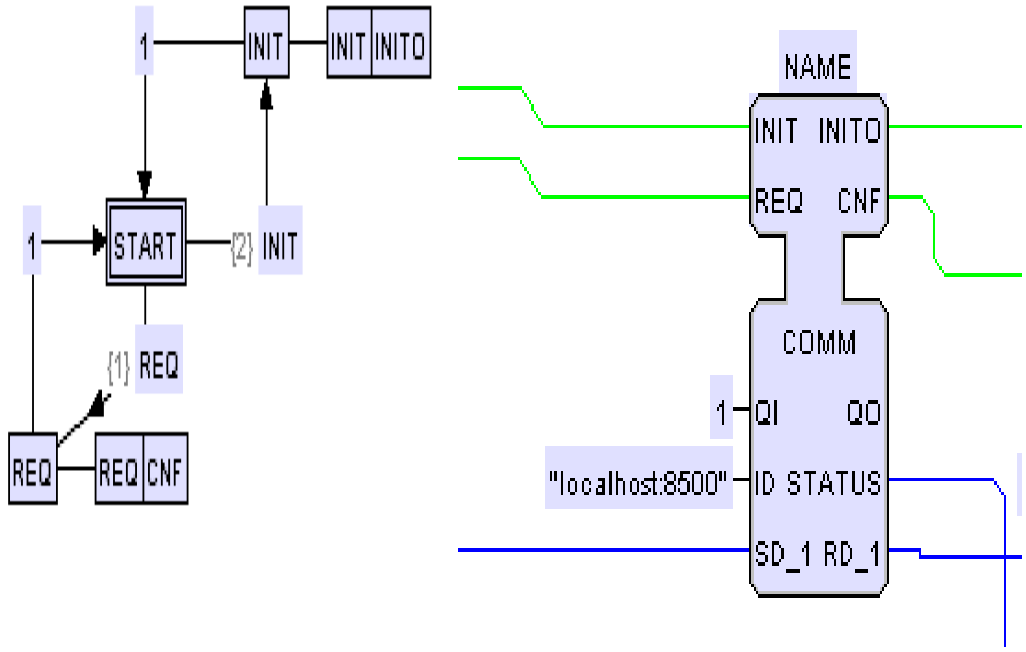


Figure 4.4 ECC and interfaces for COMM function block

For the function block to execute the algorithm effectively, it is assigned a port number, used by both the LLC and HLC, and host name as shown in Figure 4.4. When an event occurs, the REQ state executes the algorithm as shown in the transitions of the ECC.

Additionally, since FBs are Java compliant, they allow importing Java classes into the function block. For the algorithm developed in the COMM function block to interface HLC and LLC, Java.io.IOException and Java.net.* classes were imported and used to implement the algorithm. To get data from the WSTRING into the algorithm, the dot (.) *value* function was used. Dot *value* is a function block based method used to get data assigned to the FB for use in the algorithm. Figure 4.5 illustrate the algorithm used in the COMM FB.

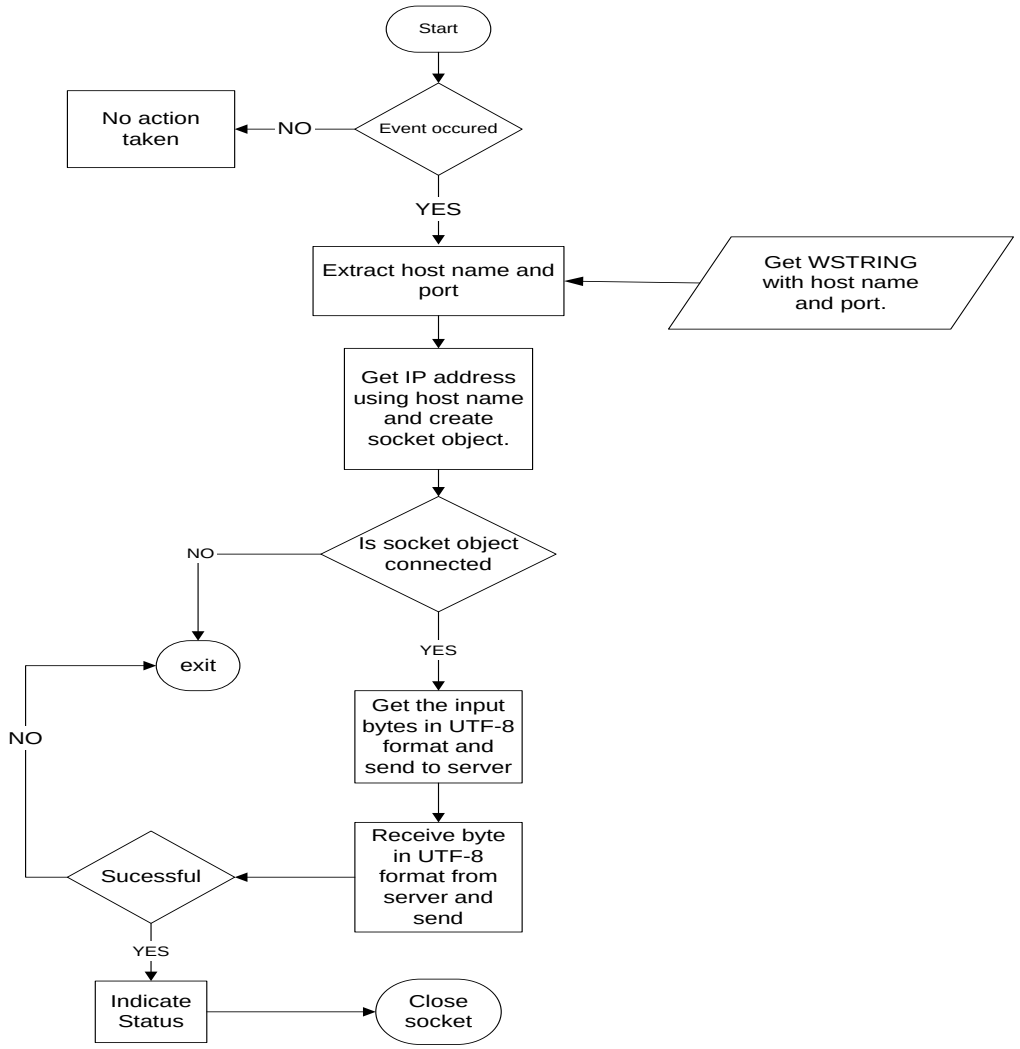


Figure 4.5 Algorithm for COMM function block

4.4 Agent based approach

In the agent based approach, the agent communicates with the LLC and the cell controller at a HLC layer. The LLC is a visual C# TCP/IP server program and the cell controller agents run as TCP/IP server program. Therefore, it is required that the HLC layer be a client to both LLC and the cell controller. To achieve this objective, a *CWelderAgent* agent was developed for the modular Cartesian robot with two ports; one port for connecting to the C# server program LLC and the other, also a client port, to connect to the *WelderAgent* residing in the cell controller (see Appendix D.1 for port assignment). Detailed design of the *WelderAgent* residing in the cell controller is given in Chapter 5.

In order to create two client connections in one agent, two *OneShotBehaviour* classes were used as inner classes of an agent. To invoke a behaviour without using the *reset()* method, the *OneShotBehaviour* class is extended and a constructor made. In this way, the behaviour is only invoked when a message is

passed to it, unlike using a cyclic behaviour where there is no control when it starts to run. Furthermore, if the *block()* method were to be used, with the cyclic behaviour, the whole agent would go to "sleep".

The constructors for the two extended *OneShotBehaviour* classes takes a string passed to it by the message received from the LLC server and a HLC server as ToInternalServer(String) and FromExternalServer(String) respectively. Then the Java socket communication in blocking mode is implemented in the *action()* method of each *OneShotBehaviour*. The *action()* and *done()* methods are the two abstract methods to be implemented for a class extending the behaviour class. The *action()* method defines the operations to be performed by the behaviour, while the *done()* method returns a boolean value indicating the state of the behaviour. To initiate communication with the LLC server, the operator clicks on the button of the GUI, passing the message to the LLC server. The response from the LLC server is then passed to the behaviour within the *action()* to the constructor serving the HLC server.

In order to add behaviours to the agent, the *setup ()* method of the agent class was used. The method is intended to include agent initializations, while the actual tasks are coded within behaviours. Typical operations that an agent performs in the *setup()* method include: registering services the agent provides to the DF, starting initial behaviours, showing a graphical user interface (GUI), and connection to a database. The two behaviours are added to the agent as:

addBehaviour (new ToInternalServer (message));

addBehaviour(new FromExternalServer(message));

4.5 Message transmission to the axes

In order to actuate the three axes of the modular Cartesian robot, messages received from the cell controller through the *ToInternalServer(message)* behaviour must be passed to the LLC layer for execution. One-to-many mapping of the agent was adopted to pass messages to the three axes of the robot. The one-to-many mapping is used when one agent is used to control similar hardware components. This mapping has the advantage of reducing the number of messages that would have been passaged among agents since one agent manages all the three messages used for control and communication. Moreover, the asynchronous nature of agent communication and non-deterministic operation of agents can be a concern during operation.

The messages received from the cell control are in an extensible markup language (XML) format. Each node of the XML message is parsed by the LLC and the intended axis is assigned the message. The feedback from the drives is received by the *ToInternalServer(message)* behaviour and passed to the *FromExternalServer(message)* behaviour for the cell controller to act on.

4.6 Modular Cartesian robot test results

Hardware reconfiguration tests where not conducted on the weld robot since CANOpen interface could not yield desired results. Tests carried out on the weld robot included aspects of the software elements of both IEC 61499 and agent based control to enhance reconfiguration of the assembly cell as whole.

Both IEC 61499 function blocks and JADE agents were found to be feasible technologies to implement the HLC for modular Cartesian robot. Since the axes of the robot were not required to be coordinated while moving, the demand on the HLC was quite moderate. Therefore, the focus of each approach was on their ability to support reconfiguration.

As discussed in Section 2.5, an open architecture is an important consideration for controllers. Both FBDK and JADE are open systems that are based on Java, thus meeting the open architecture requirement. However, it is an advantage from an interoperability and maintenance perspective if the HLC software is either IEC 61499 or FIPA compliant. Furthermore, the control strategies were investigated to establish if they exhibited the six core characteristics of RMSs.

IEC 61499 function blocks were found to be inherently modular. Furthermore, the IEC 61499 standard make no provision for global variables and therefore makes them superior to agents. Moreover, the design of a control device is more standardized in the IEC 61499 standard since the functionality of the different FBs, resources and devices are already specified

Communication is a central concern in integrability. In this respect, IEC 61499 function blocks suffered a setback when used as a HLC since the ASN.1 encoding it uses for string communication over Ethernet is not widely used by other high level languages. When used to connect to the cell controller or LLC, compatible encodings have to be used and therefore custom FBs had to be created.

Two phases of diagnosability were considered: firstly during development (including major reconfiguration that requires changes to the control software), and secondly during operation. With regards to the development phase, it was found to be very difficult to diagnose FB networks since FBDK's (and other available IEC 61499 platforms) debugging tools are rudimentary. To get debug output from a FB network, one has to include print statements in the algorithm or use another network of human machine interface FBs. Moreover, being an event driven architecture, the flow of events within the FB network is fast and difficult to visualize. On the other hand, agent platforms are much more mature and have good debugging tools.

Convertibility for RMSs was found to be more of a concern for hardware than software. Both FBDK and JADE allow for easy conversion of the HLC and the main concerns were found to be in terms of diagnosability, as described above.

An IEC 61499 implementation is easily scalable since it is modular. A FB can be re-used by assigning a unique name to each instance of the FB. This was used in the design of the three axes.

Furthermore, not many advanced features of the agent based control were used in the modular Cartesian robot. Therefore, the six properties of RMS for agents were not deemed conclusive for agents and a bigger platform (cell controller) was used. Chapter 5 explain most of the features of the JADE platform used to control the assembly cell bearing in mind the six core characteristics of RMSs.

5. CELL CONTROLLER FOR ASSEMBLY CELL

This chapter explains how the cell controller is designed to perform its tasks using agent based control strategy. The JADE framework and Eclipse IDE are used to design the agents which reside in the cell controller. The agents are then able to interact with the subsystems using TCP/IP connections. By interacting with the Directory Facilitator (DF), a product agent can carry out its production objectives. The details of the agents and their interactions are explained and the agents used in the control strategy are expounded in the following sections. Appendix A gives a functional analysis for the cell controller.

5.1 Cell controller architecture

The overall control strategy in designing the cell controller is based on two decisions: firstly, the product information will reside in the cell controller. This implies that a subsystem's control program is not tailored to a single product; new product introduction only affects the cell controller. By localizing software reconfiguration to one central point, both fault detection and diagnostics efforts are concentrated to one point thereby reducing time to trouble shoot and reconfigure. Secondly, during production execution, cell controller only needs to know the services offered by the subsystems in order to use them. This implies that the subsystems have to register their services to the Directory Facilitator (DF) agent of JADE platform. This also includes newly introduced subsystems.

The JADE agent platform was chosen to develop the cell controller because of various considerations including: maintenance, popularity, accessibility, evolution and it is fully distributed in nature. Furthermore, JADE is fully implemented in Java which is platform independent. Therefore, the system can be distributed across different machines with different operating systems. Additionally, from 1998 when JADE was developed by Telecom Italia (formerly CSELT) it has been updated from time to time and the latest version, JADE 4.2, was released in June 2012. It also complies with the FIPA standard and therefore the agent communication language (ACL) is FIPA complaint. The primary features of FIPA ACL, as used in this setup, offer an opportunity to use different content languages and manage conversations through predefined interaction protocols.

The Semantic Language (SL) used in the cell controller is Codec. The choice was motivated by the fact that it is human-readable, which is helpful when debugging and testing an application. It can also be adapted when there is a need for agents, produced on different platforms by different developers, to communicate.

The ontology used for communication between agents is the JADEManagementOntology. This ontology eases the message exchange between agents since any newly introduced agent will have to use an already existing ontology, thus saving much effort to develop a new ontology if the cell controller were to be reconfigured and new agents introduced.

With the cell controller set up in this manner, the strategy provides flexibility to the manufacturing system control. The control program has control of which machines it engages during production. Similarly, when more capacity is needed, the control program can engage more machines with similar services to produce the required product using the CNP as explained in Section 5.5.4.

To implement this control strategy, the cell controller design is based on the PROSA reference architecture. The choice of PROSA was motivated by the fact that it simplifies the design of the Multi-agent System (MAS) since the analysis step of all the roles of agents and their interaction has been largely completed. The other advantage which the PROSA reference architecture offers is that it decouples the cell controller control logic from the physical machines and therefore simplifies the distribution of control and resources. Decoupling control logic from hardware also considerably simplifies reconfiguration of the structure, hardware and software. Furthermore, Valckenaers et al (2011) state that reference architectures do not provide final solutions, but only a common basis from which to start. They further state that the aim of reference architectures is to be generic and widely applicable thereby leaving design and implementation to be done by the developer. Therefore, PROSA leaves detailed implementation to the application developer.

5.2 System partitioning

In a typical industrial setup, the structure of the hardware used for production is such that they are interconnected and the weld assembly cell is no exception. In order to use agent technology for the control of the hardware, agents are mapped to hardware which they control; and the best mapping would be based on a specific application. Ticky et al (2006) identified guidelines for the mapping of agent controls to their respective hardware and these are: one-to-one mapping, where one agent is responsible for one component, such as one agent controlling a component; one-to-many mapping, where one agent is controlling a set of equipment components; and many-to-one mapping, where more than one agent is operating a single equipment component.

In the partitioning of the weld assembly cell, one-to-one mapping was adopted. Ticky et al (2006) state that this mapping is very flexible since new components can be easily added to the system, together with the agents associated with them. Similarly, the mapping eases system development and debugging since the number of different agent types is lowered compared to many-to-one mapping. The one-to-many mapping, though efficient when several naturally related hardware components are grouped together, is only used in the control of the modular Cartesian robot, because it reduces the number of messages passed between agents. In the weld assembly cell, however, one agent mapped to all the systems can be a point of failure and also a bottle neck. The mapping can also reduce the robustness when one agent is controlling two different sections of a cell.

5.3 Control design and implementation for the assembly cell

With the assembly cell partitioned as described in the preceding paragraph, the agents can interact using the contract net protocol (CNP) with the DF as the link for the agents in question (consider Figure 5.1).

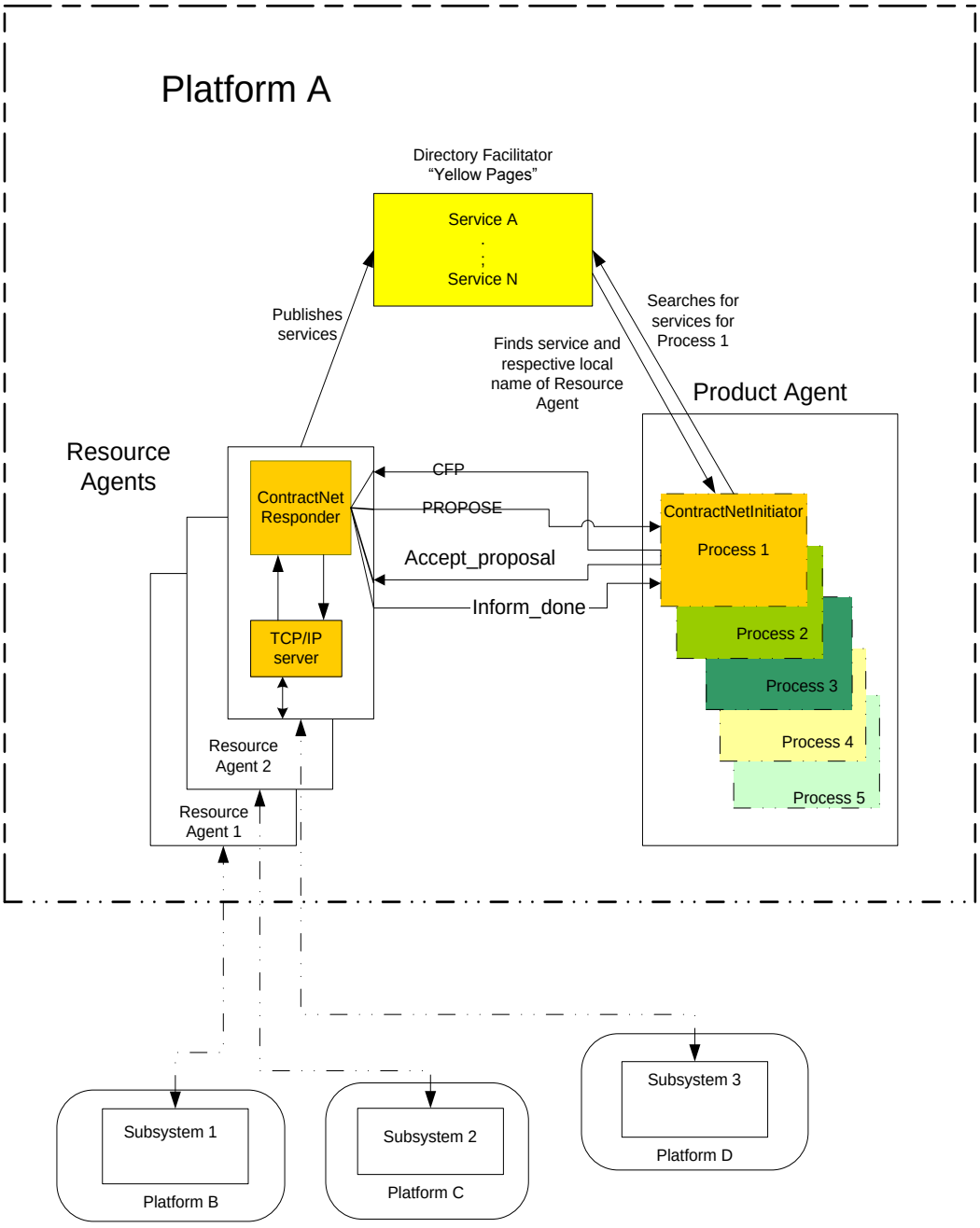


Figure 5.1 System partitioning and CNP based interaction of product and resource agents

In order to implement the control program, all the workstations explained in Section 3.1 are treated as holons, while resource agents represented in PROSA are treated as decision making entities for the holons. By using one-to-one mapping, each resource agent is mapped to a holon. Furthermore, all agents in PROSA reside in one agent platform (AP). In JADE terminology, an AP consists of machines, operating systems, FIPA agent management components, agents and any additional software (Bellifemine et al., 2007). However, the specific internal design of an AP is left to developers when more components are added.

With the assembly cell setup as explained in the preceding sections, two layers with different concerns are formed, namely execution and control layers. Each layer has a sphere of influence and to limit the sphere of influence for each layer, a Holarchy was formulated as shown in Figure 5.2. TCP/IP protocol is then used to connect the control and execution layers.

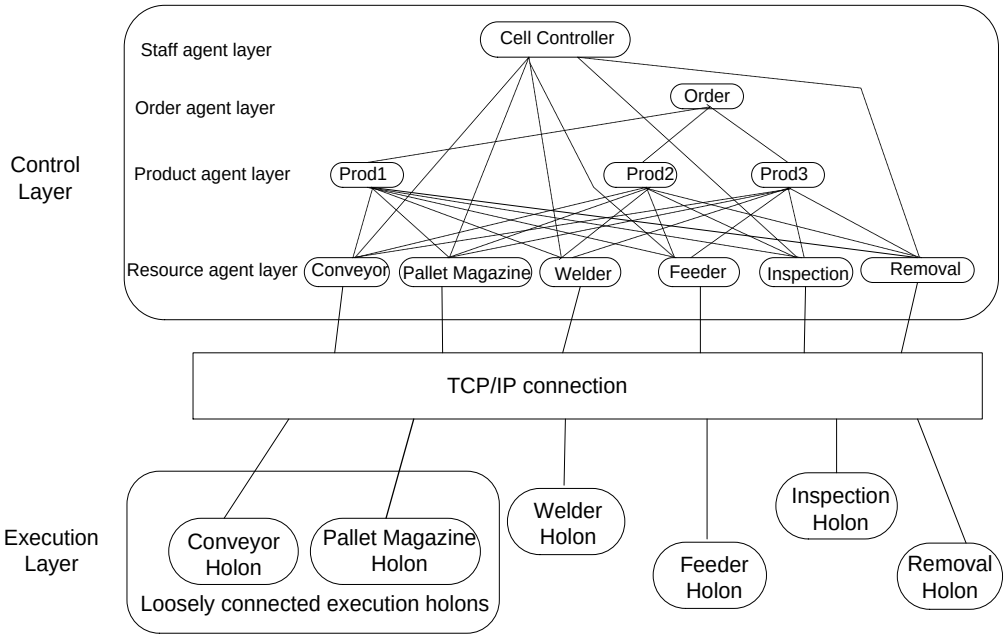


Figure 5.2 Weld assembly system holarchy

In the execution layer, holons execute commands received from the control layer and give their status during and after execution, while the control layer controls and makes decisions on behalf of the execution layer depending on their status.

Each holon in the execution layer is assigned a port through which it can listen for commands as well as indicate its status. Appendix D.1 gives the port numbers assigned to each holon. By using the TCP/IP protocol, each resource agent in the control layer is linked to a holon and exchange messages accordingly. The TCP/IP

protocol was selected as it is easy to use and fits well for a Windows 7 PC used to host the cell controller. The other advantage is that TCP/IP protocol can be used on wireless communication with the right hardware interfaces when the need arises. Moreover, wireless communication can also aid structural reconfigurations.

The six resource agents in the control layer have to publish services on behalf of their respective holons once they start running in the control layer. The services published by resource agents into the DF include:

- Unloading of the fixture from the pallet magazine to transport system
- Transportation of the fixture to different workstations.
- Loading of different parts on the fixture.
- Welding of the product on the fixture.
- Inspection of the welded product.
- Removing of welded part from the fixture.
- Storage of pallet in the pallet magazine.

To publish the agent name and its services in the DF, the *DFAgentDescription*, and *ServiceDescription* classes are used in the *setup()* method of the agent class. The service type is then added using *setType ()* method. Appendix D.3 gives the code used for publishing services to the DF. The aforementioned services are provided by; *PMAgent* representing the pallet magazine, *ConveyorAgent* representing the conveyor, *FeederAgent* representing the feeder subsystem, *WelderAgent* representing the modular Cartesian robot, *InspectionAgent* representing the Inspection subsystem and *RemovalAgent* representing removal subsystem. The pallet magazine holon and conveyor holon in the execution layer are loosely connected since the pallet magazine has to get a confirmation message from the conveyor each time a pallet is offloaded or loaded in order for the two subsystems to synchronize.

The product agents represented by Prod1, Prod2 and Prod3 in Figure 5.2, interact with resource agents by exchanging agent communication language (ACL) messages using the FIPA contract net protocol (CNP). With a CNP, in a call for proposal (CFP) message, the product agent can request for a service and the time to respond to a request. The *setReplyDate()* method was used to ascertain the time the response is expected.

The ability to set the time for an agent to reply, while at the same time searching for a service from the DF, using a CFP message, gives great benefit to the setup. The set time can be used in the evaluation of bids in CNP, while searching for a service allows using multiple agents matching a search description to be used at the same time. Furthermore, redundancy can be introduced when needed since the CNP provides for multiple agents to bid for a CFP message. For instance, when a new subsystem is introduced, a CFP message from a product agent is sent to all resource agents with a service needed at that particular time provided they have registered with the DF and their description matches that of the search template. Similarly, different product types can be produced concurrently on the same

production facilities using different product agents. This is because only the service is required and the resource agent providing a service can take orders as long as they are in the message queue and the subsystem does not breakdown. Section 5.5.4 explains this aspect in more detail.

5.4 Product agents

The product agents are a model of the actual product. They have knowledge of the procedure and processes involved in order to have the product made. For the product agents to have access to the services offered in the production process, they have to search for the services needed from the DF and can then interact with the resources using CNP. This process enables the agent to optimize a service characterized by the task through searching and discovering the appropriate service.

Product agents were implemented in two different approaches using the complex behaviours. In some industrial situations, products are simple and only need a few simple steps to make. Typically, they may need just a sequence of production stations with simple diagnostics. This sort of production set-up is modeled here using a *SequentialBehaviour*. Alternatively, complex products involving a larger number of workstations, and therefore more detailed diagnostics are modeled using the *FSMBehaviour*. Each product has a way of handling disturbances in the cell as explained in Section 5.4.3.

5.4.1 Design of the product agent using a sequential behaviour

The *SequentialBehaviour* class implements composite behaviour which schedules its children using a sequential policy. The behaviour starts with the first child, then moves to the next child and terminates when the last child is completed. This kind of implementation meets the basic design requirement of a product agent since in a typical discrete production setup a product is produced in a sequential manner. Additionally, using one agent to execute all the processes reduces computing resources since each agent runs in its own thread (Bellifemine et al, 2007). Furthermore, exchanging many messages before a task is done may lead to trading robustness for complexity (Ticky et al, 2006). Therefore, by using the sequential scheduling policy of the *SequentialBehaviour* class, each operation in the production line can be executed sequentially.

During production in the weld assembly cell, for instance, the sequence of operation requires unloading the pallet with fixture from the pallet magazine, placing parts on the fixture, welding and inspection of the product and subsequently removing the welded parts from the fixture. Each step in the production cycle requires the subsystem to inform the product agent whether it has successfully accomplished its task or the task was a failure. The information from the subsystem is sent to the product agent as an ACL message.

After receiving a message from the subsystem, the product agent makes a decision whether to continue production (i.e. when the task is successful) or inform the

order agent (i.e. when the task has failed). Figure 5.3 shows a *SequentialBehaviour* class implementation in a product agent.

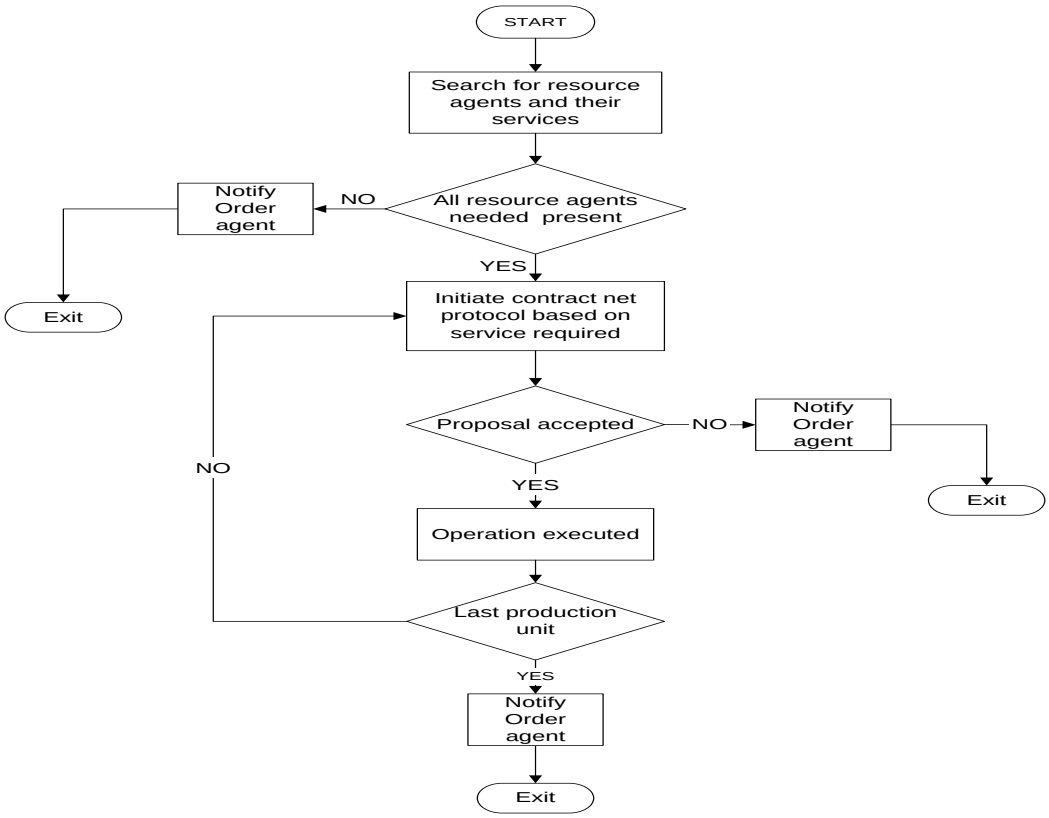


Figure 5.3 *SequentialBehaviour* class implementation flow in product agent

In implementing the *SequentialBehaviour* class in a product agent, the class is instantiated as an inner class of the extended agent class as:

```
public class  pProcess extends SequentialBehaviour{
public pProcess(Agent a){
super(a);
// code for searching from the Directory Facilitator agent
addSubBehaviour( new contractNetInitiator(a,msg){
                });
// code for searching from the Directory Facilitator agent
addSubBehaviour( new contractNetInitiator(a,msg1){
                });
                  }
              }
```

The sub-behaviours represent each production process and run one after the other until the agent terminates as shown in Figure 5.3. There is however a loose connection between the pallet magazine and the conveyor as shown in the

Holarchy (see Figure 5.2). When offloading the pallet and fixture from the pallet magazine, the conveyor receives a hardware interface message from the pallet magazine and sends one back again to confirm the operation was successful. Messages exchanged in this pattern enable synchronization of activities between the two subsystems. Details of message formats and impact on subsystem control are given by Le Roux (2013). This arrangement, however, has an influence on the product agent design since the agent has to ensure, through successful passing of messages, that the pallet with the fixture is present in the production cell. Section 5.4.4 gives details on the implementation in the product agent.

5.4.2 Product agent design based on FSM behaviour

The product agent design based on FSM behaviour utilized the architecture of the *FSMBehaviour* to mitigate unforeseen disturbances which might occur during production. The *FSMBehaviour* has a number of methods to use in order to achieve this goal. The behaviour also provides states to be registered for implementation.

In order to register a state, the FSM behaviour provides the *registerState()* method. The method accepts two arguments: a *String* defining the name of the state that is being registered and a *Behaviour* that will be executed in that state (Bellifemine et al, 2007). Further, the *FSMBehaviour* class provides two other methods for registering states and their transition during execution. The *registerTransition()* method accepts three arguments: two *Strings* defining the source state and the destination state of the transition and an integer value defining the label marking the transition. The other method, *registerDefaultTransition()* method, allows the definition of a default transition between two states. This method is not marked with any label and is only followed if and only if all other transitions from the same state are not followed (Bellifemine et al, 2007).

Both the *registerTransition()* and *registerDefaultTransition()* methods have an overloaded version which takes a further *String[]* parameter. The *String[]* parameter indicates a set of finite state machine states that must be reset when the registered transitions are followed.

To define which state will start first and which one will be the exit state in the execution process, *FSMBehaviour* class provides the *registerFirstState()* and *registerLastState()* methods respectively. While there can only be one state from where to start, a number of termination states can be defined.

Before a product agent design based on the FSM behaviour is implemented, normal transitions and anticipated disturbances which might cause unwanted transitions must be established. From the assembly cell view point, the normal transitions are: pallet magazine to feeder, feeder to inspection, inspection to welder, welder to inspection and finally to removal after which the pallet is loaded back to the pallet magazine.

The conveyor facilitates these transitions and is therefore the main link to all stations.

Having identified the normal transitions and the default transitions, the running and reactions of the product agent during production is then coded in the agent. Figure 5.4 illustrates possible transitions of pallets after being offloaded from the pallet magazine as explained in Section 5.4.4.

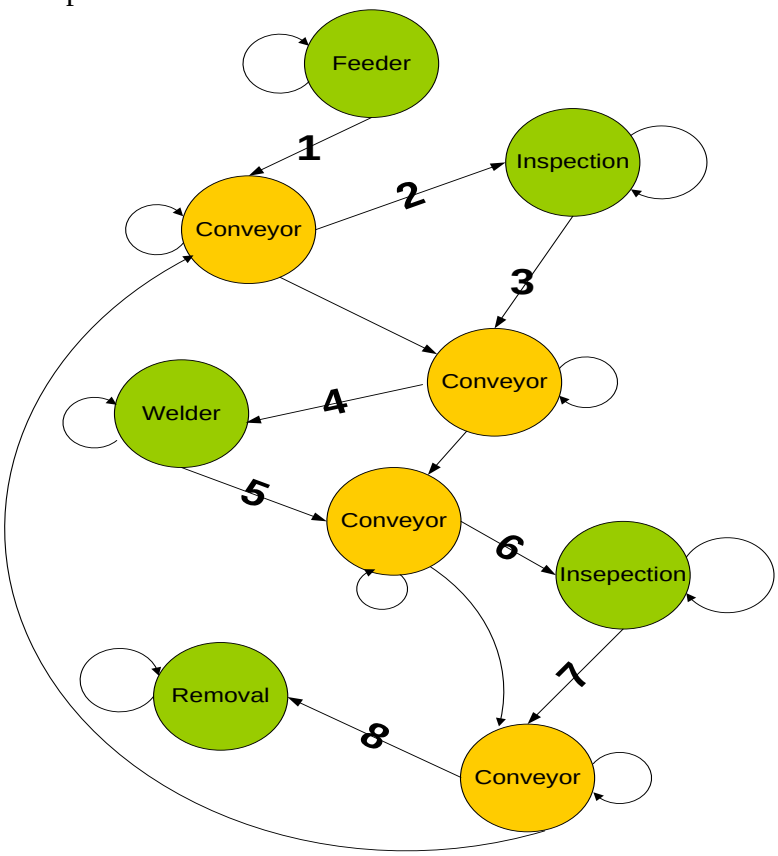


Figure 5.4 Transitions of the pallet after offloading

A normal transition would take the numbered route. This is when the product agent does not encounter any problems. However, for the states to be reused, they need to be reset. JADE provides a *reset()* method to achieve this. The resetting action is shown for every state the product agent uses.

The production setup in the assembly cell is sequential. Therefore, if at the feeder station there happens to be a problem, the best the product agent can do is to take the pallet round through the round robin of the conveyor so that the fault can be fixed. This route is shown is Figure 5.4. However, if the conveyor has a problem, the whole setup fails to run since all other activities depend on the transport system.

Since all the states are re-used by the product agent, when defining the transitions, the *registerTransitions()* method with four parameters is used as shown.

```
FSMBehaviour fsm=new FSMBehaviour(An agent){
public int onEnd(){
reset();
myAgent.addBehaviour(this);
}
 };
fsm.registerState(new Conveyor("ConveryorAgent", "CC_MOVING,2,1,3"),
STATE_B);
// Other states are put here including the starting and exit states
fsm.registerState(new Inspection("InspectionAgent", "INSPECT,2;"),
STATE_B);
fsm.registerTransitions(STATE_B, STATE_C, 1, new String[]{STATE_B,STATE_
C});
addBehaviour(fsm);
```

When the last state is executed, the behaviour which was added is removed from the pool of behaviours to be executed. In order to add again the same behaviour to the pool, when instantiating the FSMBehaviour, the *onEnd()* method is used. In this method, the behaviour is *reset()* and then added to the pool of behaviours as shown in the above snippet of code.

5.4.3 Handling of disturbances by product agents

Disturbances to production systems, such as machine breakdown, are a common feature of any manufacturing system. However, how these disturbances are handled, to some extent, guarantees survivability and competiveness of any manufacturing enterprise.

In one approach of the product agent design, a sequential behaviour was used. This complex behaviour schedules its children in a sequential manner. However, it cannot guarantee successful handling of disturbances. For instance, when a subsystem refuses a request to bid for a contract, the product agent continues with the sequence. Therefore, a behaviour must be monitoring the product agent in case disturbances occur. Other cases where disturbances may arise during production include when a new subsystem is introduced during production (since a product agent searches for services before execution begins) and also when the need arises to establish an alternative route when a subsystem fails during production. In these circumstances, the system must remain robust and resilient. The sequential behaviour, however, does not have the mechanisms to handle disturbances and must be therefore implemented independent of the sequential behaviour.

However, JADE provides a *FSMBehaviour* class, which is here exploited to meet this challenge. The *FSMBehaviour* class, which implements the composite

behaviour, schedules its children according to finite state machines (FSM) where each state corresponds to the FSM behaviour children. Like a sequential behaviour, the *FSMBehaviour* keeps a pointer to the current child until the child finishes when the *done()* method of the current child returns true. Furthermore, on the basis of the returned value, the *FSMBehaviour* checks its transition table in the form of integer labels as created by the developer. This enables selection of a new child to fire next time the new *action()* method of the new child is executed.

FSMBehaviour class provides an integer label as a means of setting transitions between children in the FSM behaviour. During execution, when a child is completed, the return value for that child's *onEnd()* method is taken as an exit value and is then matched against the labels of all the transitions exiting from the current child state. The first transition whose label matches the exit value is followed and its destination state becomes the new current child (Bellifemine et al, 2007). Using this *FSMBehaviour*, a more robust product agent was developed.

5.4.4 Pallet magazine and conveyor interaction during production

Offloading or loading a pallet with a fixture from the pallet magazine to the conveyor is of paramount importance. The success or failure of this activity during production largely determines the success of other production processes. Failing to offload a pallet means no production and failing to load means the production line runs without stopping, if it has already started, and therefore new orders which need different pallets are affected. This problem is compounded by the fact that the two subsystems (the pallet magazine and conveyor) are loosely connected by the hardware interface messages which must always be exchanged for offloading or unloading of a pallet to take place. Activities taking place at the conveyor station and the pallet magazine differ, but hardware interface messages enable the two subsystems to synchronize their activities during offloading or load of the pallet.

To tackle this problem, the *ParallelBehaviour* class is used. *ParallelBehaviour* invokes a current child and moves the pointer forward to the next sub-behaviour regardless of whether the former was completed or not. To ensure all operations in the *ParallelBehaviour* class are complete, the parallel behaviour provides a termination policy. The termination policy must be satisfied before termination can occur. Two termination policies are present in *ParallelBehaviour* class, i.e. WHEN_ALL and WHEN_ANY. WHEN_ALL termination policy ensures all operations in all the parallel behaviours are completed for the *ParallelBehaviour* class to terminate, while WHEN_ANY termination policy ensures termination when any of the behaviours completes. To ensure all the communication between the *PMAgent* and *ConveyorAgent* is completed, WHEN_ALL termination policy was used.

If a pallet is needed during production, the product agent requests a pallet from the pallet magazine by initiating a CNP driven conversation with the *PMAgent*. At the same, a CNP driven conversation between the conveyor (through the

ConveyorAgent) and the product agent must start. This simultaneous invoking of subsystems is done with a *ParallelBehaviour*. During the same period, the two subsystems must interchange hardware interface messages. The hardware message interchange happens when both the subsystem and the product agent each accept the proposal to offload a pallet and to transport a pallet by the pallet magazine and the conveyor respectively as commanded by the product agent.

At the point of sending interface messages, the pallet magazine sends the first message to the *PMAgent*. The *PMAgent* forwards the interface message received through its port to the *ConveyorAgent* using ACL message and the conveyor, upon receiving the message, sends it back again. This interchange of hardware interface messages between the conveyor and the pallet magazine also applies when the pallet is being loaded in the pallet magazine.

For a product agent based on *SequentialBehaviour*, for instance, the sub-behaviours in the agent with CNP, responsible for the *PMAgent* and *ConveyorAgent* were run in a dedicated thread to ensure other process within the behaviour do not interfere. By using a dedicated thread, the sub-behaviours within the agent can continue to run until they satisfy the termination policy. The snippet of the code thus implemented is shown below.

```
public class pProcess extends SequentialBehaviour{
public pProcess(Agent a){
super(a);
ParallelBehaviour a=new ParallelBehaviour(a,ParallelBehaviour.WHEN_ALL);
// code for searching from the Directory Facilitator agent
a.addSubBehaviour( new contractNetInitiator(a,msg){
                });
// code for searching from the Directory Facilitator agent
a.addSubBehaviour( new contractNetInitiator(a,msg1){
                });
addSubBehaviour(tbf.wrap(a));
            }
                }
```

Above *tbf* is an instance of *ThreadedFactoryBehaviour* class.

During the execution of the product agent, each production process, including offloading a pallet, in the agent is characterized by the search for a service. This approach is necessary to ensure that the service required is always present before a process commences. It further affords the product agent time to decide which of the available services it can pick from at a point in time. If however the service needed at a particular time during production is not present, the product agent informs the staff agent so that an appropriate action is taken. Appendix D.4 gives the code used to search for services by the agent and their respective behavior and the message sent to the staff agent if the resource agent is not found.

The interactions between the product agent and the resource agents can be depicted as shown in the Figure 5.5.

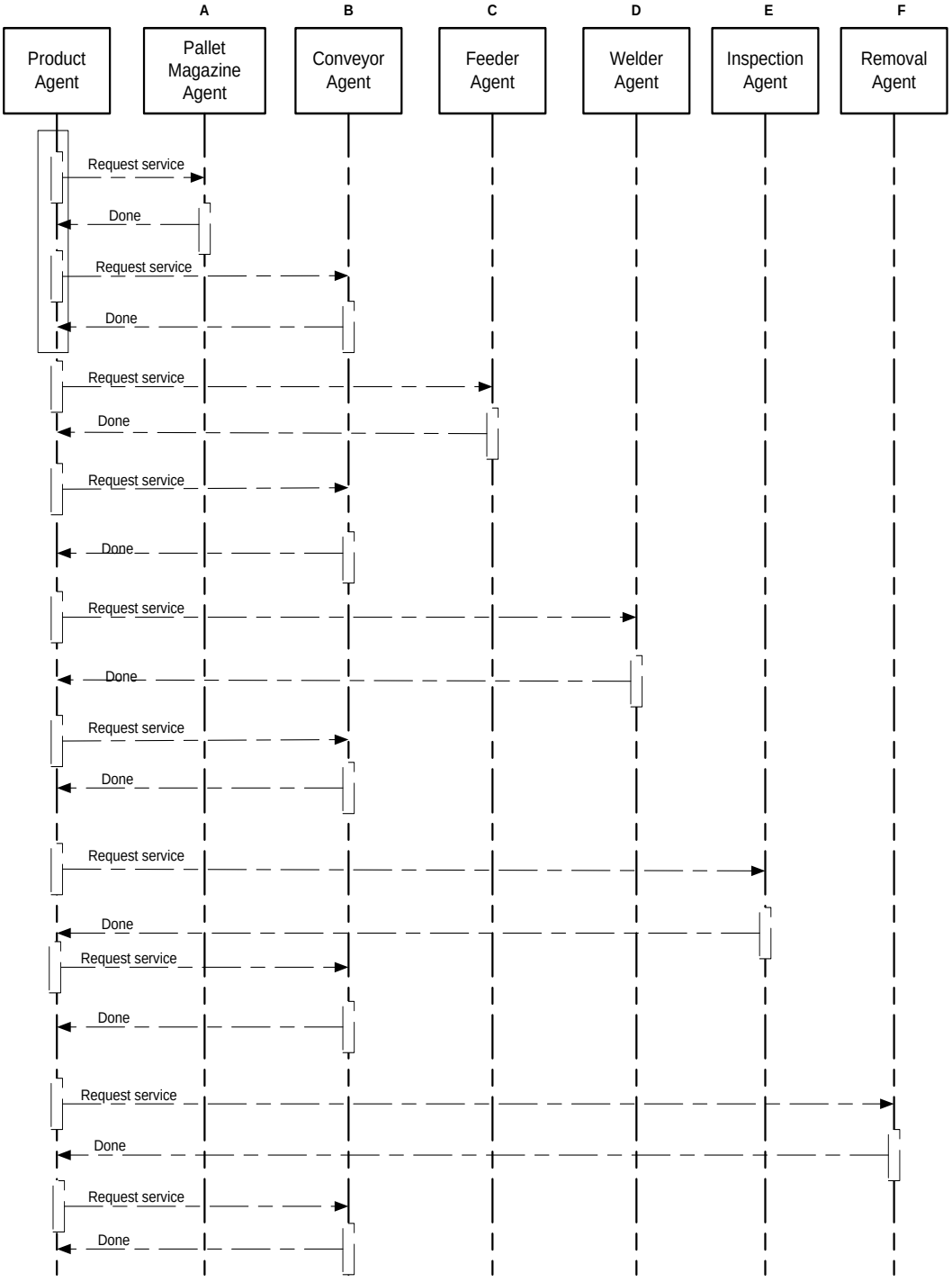


Figure 5.5 Sequence diagram of interactions for product and resource agents

JADE platform provides a developer with tools to monitor activities within the platform. To monitor ACL messages that are exchanged between agents during

offloading and loading a pallet, the sniffer agent was used. Figure 5.6 shows the interaction between the *PMAgent*, *ConveyorAgent* and the product agent as shown by the sniffer agent of the JADE platform during offloading of the pallet. Here the *ParallelBehaviour* is used in the product agent to facilitate interaction between the conveyor and the pallet magazine.

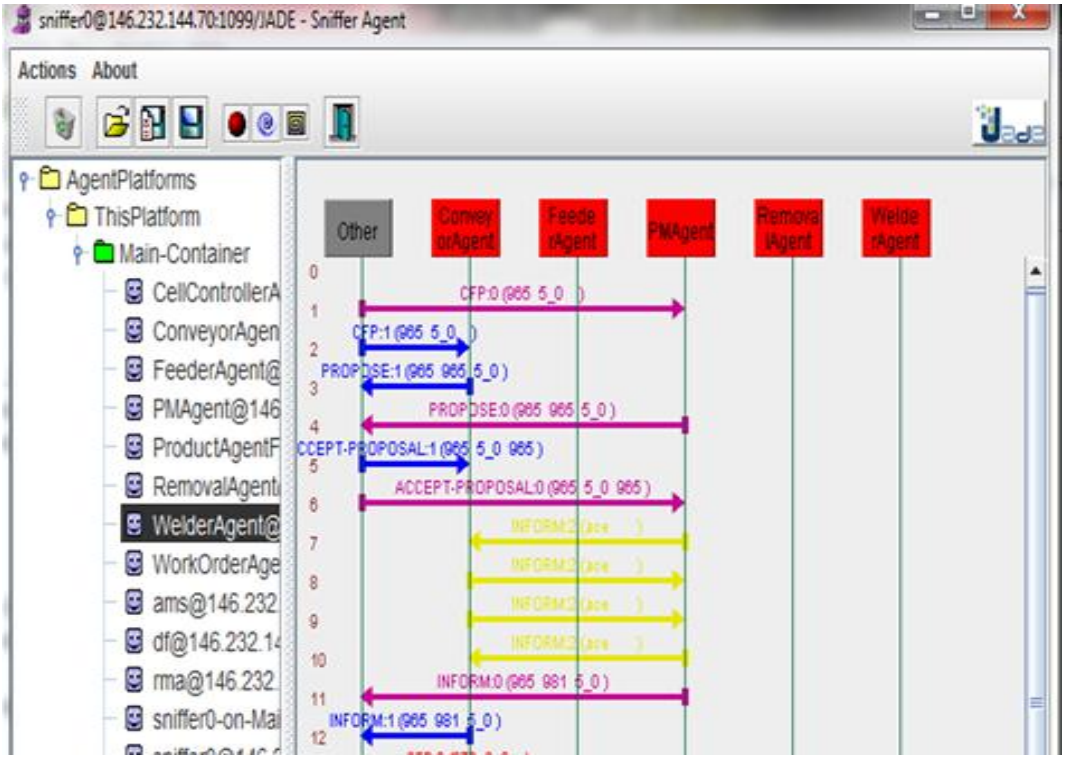


Figure 5.6 Interaction between product agent, *PMAgent* and *ConveyorAgent*

5.4.5 Product agent and pallet re-use

Pallets designed for use in the weld assembly cell are meant to be re-usable. In order for product agents to re-use them after a production cycle, a mechanism to successfully handle the situation is needed. Instances where re-using a pallet may be required is when there are insufficient pallets or the capacity of the transportation system is limited. In this scenario, the pallet with a fixture will have to be re-used until all the production requirements are met.

Unlike in a product agent implemented in a *SequentialBehaviour*, where all the production processes are handled by the product agent within the behaviour, the pallet is here assigned to an agent with the capabilities of responding to requests. Essentially, the pallet agent will carry out all the activities during offloading and loading a pallet as explained in Section 5.4.4. Therefore, it must communicate with the product agents using ACL messages. Through these messages, the pallet agent can indicate its status at any time when requested.

To request a pallet, the product agent sends an ACL message to the pallet agent. For product agents implemented in a *SequentialBehaviour*, the ACL message is sent by one of the sub-behaviours in the *SequentialBehaviour*, while for product agents using the *FSMBehaviour*, the state registered with the *registerFirstState()* method has a behaviour that requests for the pallet.

5.4.6 Introduction of a new product

There are instances when a new product has to be introduced into the system. This is necessitated by the frequently changing products needed by customers. Therefore, the need to introduce new products becomes inevitable.

The introduction of a new product impacts the assembly cell in different ways. This could be a change of control program, the addition of more subsystems or even reconfiguring the whole assembly cell. How the transition is managed makes a system worth investing in. In this work, introduction of a new product is assumed to be accompanied by either the introduction of more subsystems or reconfiguration of the entire structure with the same or new subsystems.

Based on the PROSA definition of a product agent and the product agent requirements, as explained in Section 2.2.3, firstly, the product agent should have production knowledge and the process knowledge. All the production knowledge and the process knowledge should be embedded in the software agent before it is launched. Secondly, the software agent must be able to search the DF for the service it requires when needed, and must interact with other agents using CNP. Therefore, the new product agent must be implemented with a *ContractNetInitiator* class and be capable of searching services from the DF at every moment of engagement with the resource agents.

The choice of the CNP for interaction with other agents, i.e. the resource agents, is to provide the product agent with capabilities to decide which resources it can engage if they are many of the same type. Furthermore, during production, if there are faults at the subsystem level, the agent can make a decision accordingly using its programmed evaluation method. The product agent can then be implemented as explained in either Section 5.4.1 or Section 5.4.2.

In order for the new product agent to engage with the resource agents, the product agent must first search for that particular resource from the DF. This enables the new product agent to engage the resource agent it needs. Furthermore, when the structure of the assembly cell is reconfigured, there is no need to change the control program since it can engage any subsystem by searching for the service the product agent needs. Moreover, the ontology used is common to all agents in the assembly cell i.e. JADEManagementOntology.

The product agent to be used by the new product can then be added to the cell controller by the staff agent. Addition of the agent to the cell can either be done online or when the system is shutdown. During production, the order agent can

send ACL messages instructing it to start production when needed. The ACL message must also be understood by both the product agent and the order agent. The messages to be exchanged during production must be established before launching the agent.

5.5 Resource agents

Resource agents are an abstraction of the actual subsystems residing on a different platform from that of the subsystem (consider Figure 5.1). The resource agents are representatives of the subsystems in the cell controller and they also publish their services in the "Yellow Pages" of the DF on behalf of their respective subsystems using code in Appendix D.3. This enables product agents to search for these services and use them during production. Furthermore, resource agents have to respond to product agents' requests using contract net protocol (CNP). The product agents are the initiators, while the resource agents are the responders and therefore use the contract net responder class for their implementation.

5.5.1 CNP responder selection

There are two types of contract net responder classes in JADE i.e: *SSContractNetResponder* and *ContractNetResponder*. The *SSContractNetResponder* is a single session contract net responder class and therefore carries out a protocol-driven conversation by the ACL message and subsenquently terminates when the session ends. The *ContractNetResponder* class on the other hand, after receiving a message with a predefined *MessageTemplate* parameter in its constructor, carries out the conversation and then goes back to wait for a new initiation message. For this reason the resource agents are implemented on the cyclic version of contract net responder class of JADE to ensure the resource agents are always present when needed since they do not terminate after executing a single message.

The other advantage for using the cyclic version of contract net responder is that the resource agents are required to handle as many messages from the product agents as the message queue can allow during production time. The messages are executed on first come first serve basis until all messages are served.

Each subsystem represented by a resource agent has its own form of semantic to communicate its status with the respective agent. This semantic is understood by both the subsystem and the agent. Appendix D.2 gives the semantics and meaning of each string for each subsystem. At any particular time, the state of the subsystem is known by the resource agent through the messages sent to it by the subsystem.

Furthermore, communication between the subsystem and the agent must always be present for resource agents to perform their duties. The TCP/IP protocol is used to connect the resource agent and the subsystem through an assigned port. The resource agent uses a TCP/IP server, while the subsystem uses a TCP/IP

client. Therefore, subsystems must log into the server before the resource agents are used.

5.5.2 Design of the Resource agent

The resource agent has two behaviours running during the agent life time. The first behaviour is responsible for accepting connection from the subsystem and also exchanging of messages using TCP/IP protocol, while the other behaviour is responsible for CNP driven conversation engagement. Since agents are cooperative rather than pre-emptive i.e.: one behaviour completes its task before another one starts. In order to overcome this hurdle, since the two behaviours must be running independent of each other, one of the two *CyclicBehaviour* classes used is run in a dedicated thread, while the other is run as a normal behaviour. The behaviour for the TCP/IP connection is run in a dedicated thread, while agent peer-to-peer communication using CNP, is run in a normal behaviour. This is to avoid one behaviour running all the time to the exclusion of the other. The *CyclicBehaviour* running in a dedicated thread uses the *ThreadedBehaviourFactory* class of JADE and is instantiated as:

```
ThreadedBehaviourFactory tbf=new ThreadedBehaviourFactory();
```

The CyclicBehaviour is then run as:

```
Behaviour b= new CyclicBehaviour(this){
public void action(){
}
  };
```

The behaviour object thus created is then wrapped in a wrapper class as:

```
addBehaviour(tbf.wrap(b));
```

In this way, the activities going on in one thread does not affect the other behaviour in the agent.

Information received from the TCP/IP sockets and the message queue of the agent is freely shared between the behaviours using a *DataStore* in order for the agent to carry out a given task. In JADE, *Datastore* is a behaviour whose function is that of storing data within the agent so that behaviours can share stored data.

Combining the behaviour running a contract net responder class and the *CyclicBehaviour* running a TCP/IP server in one agent has the advantages to the overall architecture of the cell controller. Firstly, this approach reduces the number of ACL messages that could have otherwise been sent between the two agents. Secondly, since each agent runs in its own thread (Bellifemine et al, 2007), there is a considerable reduction in the processing load which the PC is to handle.

The behaviour responsible for peer-to-peer communication of the agent uses a one-to-many *ContractNetResponder* class. This enables the agent to receive Call-for-proposal (CFP) messages from product agents. Figure 5.7 illustrates the resource agent architecture implemented here.

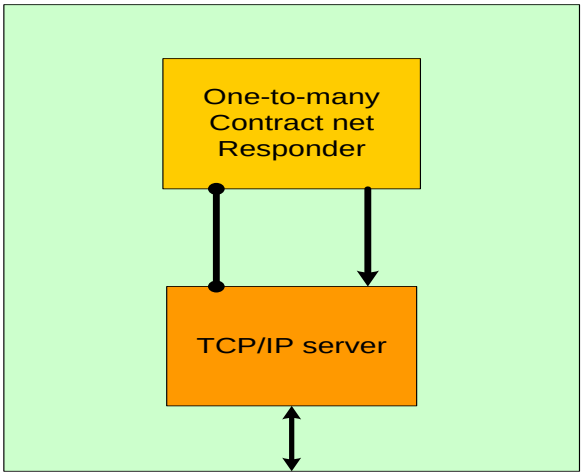


Figure 5.7 Resource agent model

In order to handle CNP driven conversations, resource agents implement two callback methods of the multi-session *ContractNetResponder* class. In the two callback methods, the control logic is implemented since JADE allows the developer to do so. The two callback methods are: *handleCfp(ACLMessage cfp)* and *handleAcceptProposal(ACLMessage cfp, ACLMessage propose, ACLMessag e accept)* and by using communicative acts, such as PROPOSE when the system is ready, REFUSE when the subsystem has a fault and INFORM when informing the product agent the status of the subsystem, the agent is able to pass relevant information to the product agent. This information is very important for diagnosing the subsystems as well as making a decision by the product agent.

In order for the *ContractNetResponder* class to receive protocol-driven messages from the product agent, a *MessageTemplate* is used to select messages from the message queue. The message template is declared as:

MessageTemplate template=MessageTemplate.and(MessageTemplate.MatchProt ocol(FIPANames.InteractionProtocol.FIPA_CONTRACT_NET), MessageTemplate.MatchPerformative(ACLMessage.CFP));

With the *MessageTemplate* created in this way, the behaviour responsible for receiving messages is added to the agent as:

```
    addBehaviour(new ContractNetResponder(this, template){
// Callback methods to execute.
    }):
```

The callback methods used to implement resource agent control logic, gives the programmer freedom to redefine them by customizing them with logic that relates to application domain (Bellifemine et al., 2007). In this application, the *handleCfp(ACLMessage cfp)* is invoked when a CFP message is received from the product agent. The CFP message is evaluated using the content of the message proposed. After evaluation, using the same message, a reply is created using *createReply()* method. The reply thus created will either propose or refuse depending on the status of the subsystem or the content of the CFP message. The status of the subsystem is communicated to the product agent using the *setContent()* method when the resource agent responds to the CFP message. The snippet of code is shown as:

```
ACLMessage propose=cfp.createReply();
propose.setPerformative(ACLMessage.PROPOSE);
propose.setContent("data to send to the product agent");
return propose;
```

In the *handleAcceptProposal(ACLMessage cfp, ACLMessage propose, ACLMessage accept)* method, the resource agent after proposing in the *handleCfp(ACLMessage cfp)* method, accepts the proposal from the product agent and sends the command for execution to the subsystem through the TCP/IP connection. The sending of data through TCP/IP is achieved by passing received data to the *Datastore* where the behaviour running the TCP/IP server collects the data and sends it to the subsystem. Furthermore, the resource agent waits for feedback from the subsystem after which it sends back the response to the product agent using the INFORM performative act, as demonstrated in the following snippet of code.

```
String data=accept.getContent().toString();
Received_msg_info.add(data);
ACLMessage inform=accept.createReply();
boolean result=false;
while(!result){
if(!RESULT.isEmpty()){
String status=RESULT.poll();
DATA=status;
inform.setContent(status);
result=true;
}
}
return inform;
```

5.5.3 TCP/IP server in a resource agent

Resource agents must be able to communicate with the order and the product agents, as well as the subsystems which they represent in the control layer (see Figure 5.2). Communication between agents residing in the control layer is a peer-

to-peer communication in which agents exchange ACL messages. However, in order for the resource agent to communicate with the respective holon, a TCP/IP server was adopted. Moreover, resource agents do not share the same platform with their holons. Therefore, through the TCP/IP connection, holons in the execution layer can log into the cell controller using their customized semantic messages as shown in Appendix D.2. With this setup, a resource agent can actively communicate with product agents, order agents and the subsystems.

The other option for implementing resource agent communication with subsystems would be the use of the remote monitoring agent (RMA) and federating all the DFs from different platforms. However, this method falls short of our test requirement since the platforms managed by the RMA must be FIPA compliant (Bellifemine et al, 2007). In this case study, however, the implementation of the subsystems' control does not necessarily need to be FIPA compliant.

The TCP/IP server depicted in Figure 5.7 for a resource agent model, uses the Java new I/O (Java NIO) package to implement the TCP/IP protocol. The Java NIO application programming interfaces (APIs) introduced in Java v1.4, provides new features and improved performance in the areas of buffer management, scalable network, file I/O, character-set support, and regular expression matching (Oracle, 2012). Unlike the Java IO package, Java NIO uses buffers to read and write to a socket channel. In addition, Java NIO is non-blocking. Non-blocking mode enables a thread to request data from a channel and only gets what is currently available, rather than be blocked until data is available as is the case with the Java IO blocking mode. Therefore, in non-blocking mode, a single thread can manage multiple channels of input and output. This aspect of Java NIO makes it possible for a resource agent with one-to-many mapping to control subsystems offering the same service; thereby introducing redundancy in the system when needed. Similarly, selectors can be used in Java NIO in managing the socket channels.

To implement Java NIO in the resource agent, the *ServerSocketChannel* and *SocketChannel* classes are created as fields of a *mysockets* inner class in the resource agent. The *mysockets* inner class is then declared as an object of a *Vector* field of the resource agent class. In this way, as many socket channels objects as needed can be instantiated in the server. The socket channel objects were then instantiated in the *setup()* method of the resource agent. Therefore, during the running of the agent, all the connections are accepted and binding to the respective ports is done. This further implies that the resource agent must be running before any subsystem can log into the server.

When a resource agent receives a message through the TCP/IP server socket, it passes the message to the contract net responder class running within the agent for the agent to make decisions. This is achieved through the use of a global variable shared in the agent by the behaviour running the contract net responder class and

the *CyclicBehaviour* running the server. Depending on the message type, the global variable is assigned a message and the same message is used by the contract net responder to make a decision.

When the contract net responder class finally wins a bid, based on the message received from the subsystem, the message is passed to the subsystem. To pass the message to the subsystem, the contract net responder class uses a Java *Queue* to pass the message. The server periodically checks whether a message is in the *Queue* and, when present, writes to the port connecting the subsystem.

5.5.4 CNP based interaction of resource agents

The contract net protocol (CNP) is the basis on which the resource and product agents interact. All the agents in the control layer begin to run when the cell controller is running. Resource agents, in particular, begin by publishing their services to the Directory facilitator (DF) at the same time the TCP/IP server begins running in the agent. Each subsystem can then log into the server when ready. The subsystems can either log in at the same time or in sequence. The logging in of one subsystem does not affect the other subsystems since each agent has its own TCP/IP server. However, when a subsystem logs in, the staff agent is sent an ACL message. This message enables the staff agent to give appropriate expert information to the order agent when information is needed.

When all the subsystems have logged in to their respective resource agents, the order agent can then take orders from the scheduler through the assigned port. Assigning a port to the order agent implies that the order agent conforms to the model on which the resource agent is built (consider Figure 5.7). The order agent therefore has a TCP/IP server running within the agent. Additionally, the order agent can also take orders from the operator through the graphical user interface (GUI).

5.5.5 Information interchange between resource agents and subsystems

Each subsystem has its own set of parameters that have to be sent to it for it to effectively carry out the task at hand. These parameters can also be used to optimize each operation undertaken. The parameters for each subsystem are explained here.

The modular Cartesian robot needs the coordinate positions for weld points, speed, weld current and time to weld. Instructions from the product agent to weld should therefore pass these parameters to the subsystem. The instruction is passed to the subsystem through an accept proposal message during resource agent and product agent CNP based interaction when the resource agent wins the bid on behalf of the subsystem.

On the feeder subsystem, comprising of the 6 DOF robot and singulation unit, the parameters are different. The 6 DOF robot has to pick and place the circuit breaker components onto the fixture. The singulation unit uses a camera vision in

order to identify a part and consequently gives the coordinates to the robot. The robot, upon receiving the coordinates, positions itself to pick that part. Therefore, the resource agent only passes, from the product agent to the subsystem, coordinates for placing the part on the fixture and the part types. The instruction to the subsystem includes: part number, X, Y, and Z coordinates and the pick-up angle. From this information the robot then finds the position where to pick that particular part and places it on the fixture.

The conveyor on the other hand needs to know the station where to take the pallet with the fixture. It does not need to distinguish between fixtures since the pallets and fixture have standard dimensions. Therefore, the parameters that need to be passed are the station number (shown in Figure 3.2) to take the pallet. The other parameter needed by the conveyor is the job number. The job number is used to differentiate between jobs that are running in the cell. Appendix D.2 gives the format of sending these parameters to each subsystem.

5.5.6 Fail-safe of resource agents

Since resource agents play a critical role in the control of the subsystems, in that they are representatives of the actual subsystems, it is important to take care of the misfortune of them failing during operation. This will enable their respective subsystems to react accordingly.

Therefore, there must be a mechanism to inform both the cell controller and the subsystem. As implemented here, if a resource agent fails, that resource agent takes advantage of the agent architecture to inform the subsystem and the staff agent. The architecture of the agent class is such that just before the agent terminates, it invokes the *takeDown()* method to perform clean-up operations, such as removing the agent subscription from the DF. It is in the *takeDown()* method where the code for sending a reset message to the subsystem using TCP/IP protocol is placed, as well as an ACL message to the staff agent when the agent terminates abruptly during operation. The message sent to the staff agent enables the staff agent to re-launch the agent again after eight seconds. For the affected subsystem to log in again, they also wait for a minimum of eight seconds after receiving a reset message before they can log in again. Details of the re-launching a failed resource agent are explained in Section 5.7.

5.5.7 Introduction of a new resource into the assembly cell

Introducing a new subsystem into the weld assembly cell may be done to increase the capacity of the cell and/or introduce more functionality into the cell. However, this task comes with challenges for which are specifically provided for in RMSs.

The procedure provided here for introducing new subsystems into the cell controller, matches with those suggested by Konrad et al (2012) as being key factors for a successful ramp-up. The key factors being: easy gathering of operator knowledge; flexible context-mapping of static and dynamic data, and extensibility and reusability. On the premise of the aforementioned ramp-up factors, the

introduction of a new subsystem into the cell controller is expounded in the following paragraphs.

The new subsystem to be introduced in the cell must have a representative resource agent in the cell controller. That resource agent for the new subsystem to be introduced must have an assigned TCP/IP port number through which the subsystem can login to the cell controller. Additionally, the resource agent must be able to publish its services to the DF and must have a *Behaviour* implemented with a one-to-many *ContractNetResponder* class according to the model in Figure 5.7. The subsystem and the agent must also have a messaging scheme between them which is understood by both parties. When the product agent begins its CNP driven messages with the resource agent, the bidding terms must be understood by both parties.

Furthermore, the service description to be used in registering the services to the DF must have already been classified for the category of the subsystems to be introduced. The service description must also be understood by the product agents which will be using the service. With all these parameters set, the staff agent can then launch the resource agent for the new subsystem.

This process reduces much effort which could have otherwise been spent in introducing a new subsystem into the cell, since the impact of introducing the new resource is restricted to the cell controller. Additionally, when a new resource agent is introduced, the new services offered by the new subsystem will only have to be known to the new products.

5.5.8 Removing a resource from the assembly cell

Removing a subsystem from the assembly is a part of reconfiguration activities. How the process is handled is as important as introducing a new subsystem into the assembly cell.

When a subsystem is removed from the assembly, the control program is affected in the following ways: firstly, the product agents which utilize that resource must be updated; and secondly, the resource agent which connects with the subsystem can be taken down from the cell controller if need arises. These are the only factors which affect the cell controller.

After the subsystem has been removed and the product agents updated, the product agent will only search for the service it needs from the DF before it engages resources. Even if the resource agent is still registered with the DF, during CNP driven conversation between the resource agent and the product agent, the resource agent will refuse the bid since no subsystem has logged into its server as illustrated in Figure 5.1 and explained in Section 5.5.3 and Section 5.5.4

5.6 Order agent

The order agent has a user interface to interact with the user and can also use a port through which it can receive orders from other systems. Other systems

include the scheduler, which in our case is developed by the CUT. The scheduler gives an order to the order agent for execution and awaits a response when the products have been made. The order placed describes what type of product and the quantity that is required. Only the product type needs to be specified since all the processes required to produce that product are already captured in the product agent.

Upon requesting for a product by the customer, the order agent establishes from the command received, whether the products are of the same type or not. If the product type is of the same kind, the order agent creates an instance of the product agent for each product ordered. The code used to create an instance of the product is shown in Appendix D.5. The order agent also provides the flexibility of removing an order from the queue. After completing the task, the product agent is removed from the cell controller. Removing an agent from cell controller is implemented using the *doDelete()* method provided by the JADE.

However, if the products are of different types, the order agent sends ACL messages to the respective product agents in order to commence production. Through the same message, the order agent indicates whether the pallet with fixture should be stored or kept available for a new order. When the orders have finished, the order agent will accordingly command the product agent to store the pallet with fixture. This arrangement allows multiple products of different kinds to run concurrently using the same production facilities. This is one of the advantages of an RMS. It must be noted here that handling of the traffic is taken care of by the transportation system.

5.7 Staff Agent

The staff agents assist the three primary agents in performing their duties. This responsibility qualifies them to have sufficient information to make better decisions (Valckenaers et al., 1998). In the cell controller, the staff agent has two tasks to perform and these are: launching all the agents and monitoring all the resource agents. The launching of agents includes the process of adding a new subsystem into the cell.

Further, if any of the resource agents which have been launched “dies”, the staff agent re-launches it after eight seconds. This is to ensure a fault tolerant cell controller which is able to handle disturbances and maintain production.

To re-launch the resource agent which abruptly terminates, the *TickerBehaviour* class of the *jade.core.Behaviour.behaviour* package is used. The *TickerBehaviour* class has the *action()* and *done()* methods pre-implemented to execute the *onTick()* abstract method repetitively after waiting for a given period. The period is specified in the constructor. The behaviour only stops when it is explicitly removed or its *stop()* method is called. In the staff agent, the *TickerBehaviour* class is implemented as an extended class as shown:

private class CreateAgent extends TickerBehaviour {

```
public CreateAgent(Agent a, String name,String Type){
super(a,8000);
String  AgentName=name;
String AgentType=Type;
   }
public void onTick(){
   // code to execute
   stop();
   }
}
```

When any resource agent "dies", it closes its TCP/IP server channel, as explained in Section 5.5.6, and sends a message to the staff agent indicating that it is no longer running. The ACL message is received and "read" by the staff agent. The same message is used to invoke the *CreateAgent* inner class, after eight seconds, and the agent is re-launched. The eight seconds is chosen arbitrarily and therefore can be changed to meet a designer's specific need. However, in the same eight seconds, the system should be able to deregister the failed agent from the DF. In this way, continuity of production is ensured.

In order to re-launch the failed resource agent, the FIPA request protocol is used in the staff agent to request the agent management service (AMS) to create the agent in question. The FIPA request protocol is used within the *TickerBehaviour* explained in the preceding paragraphs. The *AchieveREInitiator* class is used in the FIPA request protocol and two callback methods, namely *handleInform(ACLMessage inform)* and *handleFailure(ACLMessage failure)* are used to get feedback on the success or failure of the request respectively. When the failure message is received, the staff agent informs the order agent not to take any orders. The code used for re-launching a failed agent is given in Appendix D.6.

6. RECONFIGURATION INVESTIGATION

Reconfiguration investigations were carried out on the cell controller to evaluate the reconfigurability of the cell controller, based on the six core characteristics of RMSs (discussed in Section 2.5). Where possible, the available hardware in the cell was used for the investigation, while simulated resources were used where physical reconfigurations were not possible. Since the cell subsystems only interface with the cell controller through exchanging messages, a simulation of a subsystem could be performed by a computer program that reads the cell controller's messages and then returns appropriate replies.

Reconfiguration investigations could not be performed on the modular Cartesian robot since the CANOpen interface could not be established within the available time.

All reconfiguration investigations for the cell controller were done on a Dell laptop with the following specifications:

- Processor: Intel® core ™, i7-2670 QM with clock speed of 2.20GHz
- Installed RAM of 4.0 GB.
- Operating system: Windows 7, 64 bit
- The IP address is 146.232.144.70

The other PCs used in the investigations are: a Windows XP desktop which hosts the modular Cartesian robot control programs with IP address 146.232.145.145, the conveyor controller was hosted on a Toshiba laptop with Linux operating system and IP address 146.232.146.194, and the feeder station was hosted on a Dell laptop with IP address 146. 232.144.72.

6.1 Investigation 1: Introduction of a new subsystem in the assembly cell

The assembly cell was set up with subsystems arranged as shown in Figure 3.1. Once the cell controller, hosted on the Dell laptop with IP address 146.232.144.070 started running, all the subsystems logged into their resource agents, i.e. the conveyor to the *ConveyorAgent*, the pallet magazine to *PMAgent*, the feeder subsystem (with a 6 DOF robot and a singulation unit) to *FeederAgent*, and the welder (modular Cartesian robot) to *WelderAgent*.

A new resource agent (Section 5.5) to connect the new subsystem was developed and port number 9000 with IP address 146.232.145.21 were assigned. The new subsystem was simulated for station number four shown in Figure 3.2 and the control program used for the new subsystem was an adaptation of the agent-based control developed for the modular Cartesian robot. Using the graphical user interface (GUI) of the staff agent in Figure 6.1, whose local name in the cell controller is *CellControllerAgent*, the agent class, the agent name and the package where the control program is saved, were passed to the *CellControllerAgent*.

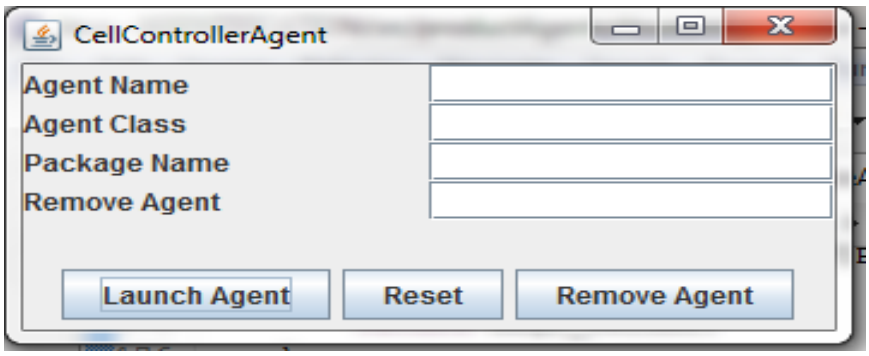


Figure 6.1 Graphical user interface for staff agent

By clicking the "Launch Agent" button, the new resource agent was added to the cell controller and the new resource then logged into this new resource agent.

The above procedure successfully simulated adding a new subsystem to the cell. The procedure also demonstrated that subsystems can be introduced in the production cell without shutting down the assembly cell.

6.2 Investigation 2: Introduction of a new product in the assembly cell

The introduction of a new product was simulated by a product which implements the *SequentialBehaviour* class. The product agent moves the pallet from the pallet magazine station to the feeder station via the newly introduced subsystem, without using the services of the weld station, and stores the pallet back into the pallet magazine before conducting this investigation. The weld station was removed from the assembly cell as explained in investigation 3. The cell controller was started and all the subsystems logged into their respective resource agents. The new subsystem was launched as explained in investigation 1.

The ACL messages to be passed between the order agent, whose local name in the cell controller is *WorkOrderAgent*, and the new product agent were established and programmed.

The new product agent was then launched in the cell controller using the staff agent with GUI as shown in Figure 6.1. To commence production of the new product, the *WorkOrderAgent* (through its GUI) was used. The quantity of products to be produced, the product type and the name of the product are the parameters that were passed to the order agent to invoke the new product agent to start production. The GUI for passing parameters of the new product agent to the order agent is shown in Figure 6.2

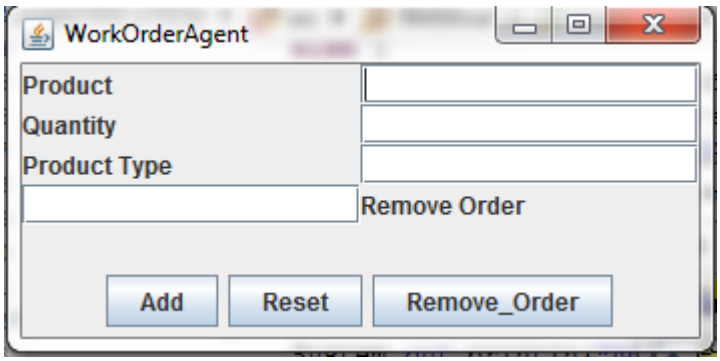


Figure 6.2 Graphical user interface for order agent

This procedure demonstrated the steps required to introduce a new product in the assembly cell. The procedure also demonstrated that the introduction of a new product can be done online.

6.3 Investigation 3: Removing a subsystem from the assembly cell

The assembly cell was set up with subsystems arranged as shown in Figure 3.1. In this investigation, the subsystem to be removed from the cell was the modular Cartesian robot which interacts with the *WelderAgent.*

With the cell controller running and the subsystems logged into their respective resource agents, test runs of a normal production cycle with the weld robot in the cell were conducted using a product agent designed using a *SequentialBehaviour*. Then the modular Cartesian robot was disconnected from the *WelderAgent* in the cell controller when the cell controller was still running.

Since the staff agent re-launches a failed agent after eight seconds, as explained in Section 5.7, the *WelderAgent* remained in the cell controller, but refused to take a bid during CNP driven conversations.

The *WelderAgent* could resume its services at any time if an operational subsystem logged into it.

This procedure completed the steps required to remove a subsystem from the assembly cell. It demonstrated that the system could continue functions even when a subsystem is removed from the assembly cell.

6.4 Investigation 4: Simulating disturbances in the cell when a product agent using a FSM behaviour is used in production

This investigation did not involve the actual conveyor, but a simulation program was developed to mimic disturbances that would potentially arise as a result of mishaps during production. The disturbances that were simulated are when the conveyor does not respond to commands, and when the conveyor generates a fault condition after taking an order from the product agent.

The LLC and HLC for the modular Cartesian robot (Section 4.4), with some changes, were used to represent the conveyor holon. The *TickerBehaviour* class, with a constructor that has a *String* parameter, replaced the *OneShotBehaviour* connecting the cell controller with the subsystem. The constructor for the *TickerBehaviour* requires the time interval to tick. This time interval is generated randomly in the agent's constructor, and passed to the parent class constructor. In this way, the time of response is uncertain for each operation.

To generate a fault condition or an error, the message received from the LLC through the *String* parameter in the constructor is overwritten and assigned an error message when the response time randomly generated is either ten seconds or thirty seconds.

In the *onTick()* method of the *TickerBehaviour* class, which is called after the time interval set in the constructor, the LLC message or error message is sent to the cell controller using a socket object. Then the *stop()* method is called to stop the *TickerBehaviour*.

A few tests using the simulated conveyor with randomly generated errors were conducted and the product agent implemented in a FSM behavior correctly handled the errors.

7. EVALUATION OF CONTROL STRATEGIES

Using the experience gain in implementing the controllers for the modular Cartesian robot and the cell controller, as well as the investigations described in Section 6, IEC 61499 function blocks were compared to agent based control as alternative strategies.

When comparing the control strategies, it should be noted that both control approaches were implemented using Personal Computers (PCs) since available Programmable Logic Controllers (PLCs) do not support agent based control or IEC 61499 function blocks. However, there are some differences between the two strategies that would help in choosing the strategy for a given application. JADE agents use an asynchronous messaging scheme, while IEC 61499 function blocks are event driven and use socket connections. These differences have been taken into account in the evaluations.

Evaluation of the two control strategies was based on the six core characteristics of RMSs (discussed in Section 2.5). For each characteristic, the two strategies are evaluated in terms of each one's ability to enhance reconfiguration in terms of hardware and software components.

7.1 Scalability of software components

The two control strategies both allow scalability of software components since they are software objects. In IEC 61499 function blocks, one function block can be instantiated multiple times by renaming that function block, while in agents, an agent code can be re-instantiated too. Examples of the applications are: when a product agent has to produce multiple products of the same type, the order agent creates multiple instances of the same product agent code; also in the IEC 61499 function blocks, when a function block representing an axis for the modular Cartesian was created for one axis, the same function block was reused for the other axes.

Furthermore, for MAS, there are two areas of scalability that can be exploited. These are hardware and software scalability as identified by Ticky et al (2006). Hardware scalability is where a system has to allow seamless utilisation of new computation units and networks that increase performance, robustness or capabilities of the system, while software scalability affords the possibility of adding and removing agents from the system at both design stage and run time.

In investigation 2, when a new subsystem was introduced to add capacity or functionality to the welding assembly cell, the agent offered a seamless utilisation by having an agent representing the subsytem integrated using the TCP/IP connection. However, there are limitations to software scalability for the Windows 7 platform. Since agents run in their own threads, the more they increase, the more computing resourses they need causing the CPU to increase its activity.

Before running agents on the Dell laptop with only background programs running, the CPU usage was at zero percent. When the cell controller agents where run, the CPU usage shot to twenty-five percent when monitored using the Windows Task Manager performance window. Such high CPU utilisation for the modest number of agents in the work presented here, demostrates that computer hardware limitations should be considered in practical implementations. Therefore, at the design stage, the maximum number of agents the system would need, if established, would give an estimation of the hardware and processing power needed to run an agent-based application. From the investigation, CPU usage demonstrated a linear relationship to the number of agents used on the PC.

Despite both IEC 61499 function blocks and agents being on par in terms of scalability, agents have an upper hand since they can be scaled up or down during runtime. This property of being able to be scaled up or down at runtime makes them suitable for applications were “objects” can be easily added or removed in a RMS.

7.2 Modularity of software components

IEC 61499 function blocks are modular. The standard is superior in this regard since the function blocks (FBs) do not have global variables nor indirect data access. Moreover, modeling of a control device is made easier since the functionalities of different FBs, for instance resources, devices, etc., are already specified. If the developer wants to develop a specific algorithm for a function block, the standard gives such possibilities. For example, in the experimental setup, a basic FB was developed for communication between the IEC 61499 control application in the HLC and the Visual C# program in the LLC.

Agents can be modularized to meet a specific control requirement. By partitioning the whole system which has to be controlled, and then mapping each partition to respective agents, the control problem can be made into manageable control modules which are simplified to control.

The three mappings can then be used namely: one-to-one, in which an agent controls a particular device or system as was used with resource agents in the cell controller in Figure 5.1 to control subsystems; one-to-many mapping, in which one agent is controlling a number of devices (this mapping was used in the control of modular Cartesian robot); and finally, the many-to-one mapping, in which multiple agents are controlling a single device.

The one-to-one mapping was used to modularize the assembly cell and such modules could be used when redundancy is needed in the assembly cell. Furthermore, interaction protocols such as the CNP, used by modularized resource agents, easies communication between modules.

Therefore, although IEC 61499 function blocks are more modular than agents, interaction protocols used by agents simplify the implementation of control in

which two or more modular components have to coordinate with each other. Modularity in software components also enhances scalability.

7.3 Integrability of software components

At the HLC layer, IEC 61499 function blocks suffer a setback. During the communication process, FBs encode their messages in ANS.1 encoding which may not be understood by the HLC layer. This scenario means that nearly every software component which needs to integrate with FBs will need to cater for the encoding. However, when used to integrate with other IEC 61499 function blocks or other IEC 61499 compliant platforms, the encoding is not a problem. Moreover, the library of the standard provides FBs for communication and interfacing with other IEC 61499 function blocks. Therefore, when FBs are used for control at HLC layer with other layers which are non-ANS.1 encoding compliant, message encoding can cause a potential problem during software integration and during communication.

On the other hand, agents are mostly used at HLC layer and can communicate with other layers without message encoding barriers. Since agents are more adapted to this layer, integrating with other HLC layers is not a problem.

Therefore, agents are more integrable at HLC layer than IEC 61499 function blocks. The IEC 61499 function blocks should be used at LLC for which they are best suited with their ANS.1 encoding. Moreover, they are event-driven with fast response time which is vital at the LLC layer.

7.4 Customization of software components

In IEC 61499 function blocks, algorithms can be developed for each FB. This aspect of a FB having its own algorithm makes customization of IEC 61499 function blocks relatively easier when the program is only needed for a specific application.

On the other hand, agents already have standard interaction protocols which allow developers to add their own logic in callback methods. Additionally, the exchange of messages between agents implementing a protocol-driven conversation is left to the interaction protocol. This was fully exploited in developing product and resource agents.

Therefore, IEC 61499 function blocks are more customizable than agents since the flow of events between FBs can also be customized unlike in agents where interaction protocols are already fixed.

7.5 Convertibility of software components

IEC 61499 function blocks are categorized and so are the interaction protocols in JADE agents. To develop an application, the choice of the IEC 61499 function blocks category to be used in application or the interaction protocol in agents is dictated by the control program to be implemented. In both control strategies, converting a software component to meet a new task for which it was not meant is

difficult. However, with agent interaction protocols, developers can add logic to be implemented in the callback methods. This also includes the use of *registerXXX()* methods (e.g *registerResultNotification()* used in FIPA request) in which a developer can add a behaviour to be included in the execution of an interaction protocol.

Therefore, both IEC 61499 function blocks and agents are on par in terms of convertability. To convert a control program used in an RMS that has been used for one application would require considerable time and effort.

7.6 Diagnosing a system using software components

It is difficult to diagnose a FB network in FBDK. For instance, in order to determine whether an event is triggering a FB and outputs are coming out, one needs to use other human machine interface (HMI) FBs, which might be time consuming. Furthermore, it is more difficult if a control program developed by a different developer has to be scaled up by a new developer who had not designed it.

On the other hand, agents have predefined classes and methods which, when implemented, help tracing where the problem might arise. In the experimental setup, a resource agent gets its information from the subsystems to make decisions. This aspect of agents being able to get data from the equipment they control to make decisions makes them suitable for the task.

Therefore, agents are more diagnosable than IEC 61499 function blocks.

7.7 Overview

From the investigations carried out in this research, the six core characteristics of RMSs demonstrated relationships among themselves: convertibility, customization and scalability were found to be influenced by the integrability, modularity and diagnosability of the components involved. For example, when modules are scalable due to their modularity and integrability, customization is achievable.

In terms of modularity and, thereby scalability, both IEC 61499 function blocks and agents are inherently modular, thereby facilitating easy scaling. However, agents' ability to be added or removed during run time (due to the architecture of the platform), can be a significant advantage. However, IEC 61499 function blocks have more clearly defined interfaces, which makes them inherently better in terms of integrablility.

8. CONCLUSIONS AND RECOMMENDATIONS

The thesis sought to evaluate control strategies that enhance reconfiguration in a reconfigurable manufacturing system. Control strategies are many and varied. However, in this research, the focus was on the use of JADE agents and FBDK's implementation of the IEC 61499 standard at high-level control layer. There is currently no commercially available PLC on which agent based control or IEC 61499 function blocks can run. Therefore, all the control strategies were implemented on a personal computer.

The reconfigurable manufacturing system for which the control strategies were applied is a welding assembly cell. The cell is intended to handle products with high variability and changeable volumes. It comprises a conveyor, a pallet magazine, a feeder subsystem with a 6 DOF robot and a singulation unit, a modular Cartesian weld robot, and inspection and removal stations. The IEC 61499 standard is only applied in the control of the modular Cartesian robot at HLC layer and its properties were evaluated in line with the six properties of RMSs (Koren and Shiptalni, 2010; Koren et al, 1999). JADE agents are applied to both the modular Cartesian robot and the cell controller.

It can be concluded that agents are more suited for control at HLC layer than IEC 61499 function blocks. IEC 61499 function blocks should be applied on the LLC layer because the architecture does not support dynamic reconfiguration (which is a crucial requirement to avoid downtime) and the ASN.1 encoding is suitable to that layer. The HLC layer has to negotiate and coordinate with other systems, which is more complex to implement in FBs. Since an IEC 61499 FB does not have provision for storage of events, it is difficult to be used at HLC for negotiation and decision making.

Modularity in software components makes software reconfiguration easy for both IEC 61499 function blocks and JADE agents. The modularity also aids in structural, software and hardware reconfiguration, since each software module has a specified component to control. In the IEC 61499 standard, the separation of events and data makes it more modular than the JADE agents.

When considering scalability, the same modularity in software can enhance addition and removal of subsystems from the cell. From the agent control perspective, by partitioning the system and then mapping agents to devices, agents simplify hardware scalability to the system. Moreover, agents are more scalable during runtime than FBs, since agents can appear and disappear without stopping the controller. The ability to disappear and re-appear without affecting the controller finds greater application during reconfiguration of the assembly cell since addition and removal or modifications to the cell can be done when the cell is running.

Furthermore, interaction protocols in agents make the implementation of complex control systems manageable since communication between agents using an

interaction protocol is already established. It is also possible with agents to implement a plug and produce system as suggested by Arai et al (2001).

From experience gained, the following recommendations for further work are made:

- Research should be conducted in combining the IEC 61499 standard and agents in one unit, since they are all implemented in Java.
- It should be assessed whether an ontology specifically developed for a manufacturing set up would aid reconfiguration with different vendor hardware components.
- More tests should be conducted on the use of multiple pallets using the FSM behaviours on the conveyor since the tests which were conducted on the FSM behaviour were mostly simulated.
- Research should be conducted for failure modes and effects analysis to prove reliability of the two control strategies.

REFERENCES

Almeida, F., Terra, B., Dias, P. and Goncalves, G., 2010. 'Adoption Issues of Multi-Agent Systems in Manufacturing Industry'. *IEEE Computer Society,* Volume 48, pp. 238-244.

Arai, T., Aiyama, Y., Sugi, M. and Ota, J., 2001. 'Holonic Assembly System with Plug and Produce'. *Computers in Industry,* Volume 46, pp. 289-299.

Bellifemine, F., Caire, G. and Greenwood, D., 2007. *Developing Multi-agent Systems with JADE.* West Sussex: John Wiley and Sons Limited.

Bi, Z. M., Lang, S. Y. T., Verner, M. and Orban, P., 2008. 'Development of Reconfigurable Machines'. *International Journal of Advanced Manufacturing Technology,* Volume 39, pp. 1227-1251.

Black, G. and Vyatkin, V., 2009. 'Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499'. *IEEE Transactions on Automation Science and Engineering*, Volume 7, No. 2, pp. 337-351.

Blanc, P., Demongodin, I. and Castagna, P., 2006. 'A Holonic Approach for Manufacturing Control: An Industrial Application'. *Information Control Problems in Manufacturing-INCOM 2006,* pp. 389-394.

Bongaerts, L., Monostori, L., McFarlane, D. and Kadar, B., 2000. 'Hierarchy in Distributed Shop Floor Control'. *Computers in Industry,* Volume 43, pp. 123-137.

Brennan, R. and Fletcher, M., 2002. 'An Agent-Based approach to Reconfiguration'. *IEEE transaction on Robotics and Automation,* Volume 18, No. 4, pp. 444-451.

Bruccoleri, M., 2007. 'Reconfigurable Control of Robotised Manufacturing Cells'. *Robotics and Computer- Integrated Manufacturing,* Volume 23, No. 1, pp. 94-106.

Burger, J., 2009. *'Reconfigurable Conveyor System and Pallet Magazine'*, BEng Final Year Project, Department of Mechanical and Mechatronic Engineering: University of Stellenbosch.

Bussmann, S. and Child, K., 2007. 'Self-organisation in Manufacturing Operations'. *Communications of ACM*, Volume 50, No. 12, pp. 74-79.

Candido, G. and Barata, J., 2007. 'A Multiagent Control System for Shop Floor Assembly'. *Proceedings of the 3rd International Conference on Industrial Applications of Holonic and Multi-agent Systems*, HoloMAS 2007, pp. 293-302.

Chirn, J. L. and McFarlane, D., 2000. 'A Holonic Component-Based Approach to Reconfigurable Manufacturing Control Architecture'. *11th International Workshop Database Expert Systems Application*, pp. 219-223.

Chryssolouris, G., Georgoulias, K. and Michalos, G., 2012. 'Production Systems Flexibility: Theory and Practice'. *14th IFAC Symposium on Information and Control Problems in Manufacturing*, pp 23-25.

EIMaraghy, H. A., 2006. 'Flexible and Reconfigurable Manufacturing Systems Paradigms'. *International Journal of Flexible Manufacturing Systems*, Volume 17, pp. 261-276.

FESTO, 2012a. *FESTO*. [Online]
Available at: http://www.festo.com
[Accessed September 2012].

FESTO, 2012b. *'Festo CMMP-AS Documentation Manual'*.

FIPA, 2012. *FIPA*. [Online]
Available at: http://www.fipa.org
[Accessed 12 July 2012].

Hall, H., Straton, R. and Zoitl, A., 2007. 'Challenges to Industry Adoption of the IEC 61499 Standard on Event-Based Function Blocks'. *5th IEEE International Conference on Industrial Informatics*, Volume 2, pp. 823-828.

Hirsh, M., Gerber, C., Hanisch, H.-M. and Vyatkin, V., 2007. 'Design and Implementation of Heterogeneous Distributed Controllers According to the IEC 61499 Standard-A Case Study'. *5th IEEE International Conference on Industrial Informatics*, Volume 2, pp. 829-834.

Holobloc, I., 2012. *Holobloc.* [Online]
Available at: http://www.holobloc.com
[Accessed 17 July 2011].

Hussain, T. and Georg, F., 2007. 'Deployment of IEC 61499 Compliant Distributed Control Applications. *IEEE*, pp. 502-505.

JADE, 2012. *Java Agent Development Environment.* [Online]
Available at: http://jade.tilab.com
[Accessed 22 May 2012].

Jennings, N., 2000. 'On Agent-Based Software Engineering'. *Artificial Intellingence, Volume* 117, No. 2, pp. 277-296.

Jennings, N. and Bussman, S., 2003. 'Agent-Based Control Systems:Why Are They Suited to Engineering Complex Systems?'. Volume 23, No. 3, pp. 61-73.

Khalgui, M., Mosbahi, O., Li, Z. and Hanisch, H., 2011. 'Reconfiguration of Distributed Embedded-Control Systems'. *IEEE/ASME Transactions on Mechatronics*, Volume 16, No. 4, pp. 684-694.

Konrad, K., Hoffmeister, M., Zapp, M., Verl, A. and Buss, J., 2012. 'Enabling Fast Ramp-Up of Assembly Lines through Context-Mapping of implicit Operator Knowledge and Machine-Derived Data'. *IFIP, International Federation for Information Processing*, Volume 371, pp. 163-174.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. and Van Brussel, H., 1999. 'Recongurable Manufacturing Systems'. *Annals of the CIRP*, Volume 48, No. 2, pp. 527-540.

Koren, Y. and Shiptalni, M., 2010. 'Design of Reconfigurable Manufacturing Systems'. *Journal Of Manufacturing Systems*, Volume 26, pp. 130-141.

Kotak, D., Wu, S., Fleetwood, M. and Tamoto, H., 2003. 'Agent-based Holonic Design and Operations Environment for Distributed Manufacturing'. *Computers in Industry*, Volume 52, pp. 95-108.

Kruger, K., 2013. *'Control of Feeder Subsystems in Reconfigurable Manufacturing Systems'*, MScEng Thesis, Department of Mechanical and Mechatronic Engineering: University of Stellenbosch.

Le Roux, A., 2013. *'Transportation Systems in Reconfigurable Manufacturing Systems'*, MScEng Thesis, Department of Mechanical and Mechatronic Engineering: University of Stellenbosch.

Leitao, P., Colombo, A. W. and Restivo, F., 2005b. *Formal Specification of ADACOR Holonic Control System: Cordination Models.* Seville, Spain, 44th IEEE Conference on Decision and Control, and European Control Conference, pp. 2137-2142.

Leitao, P., Colombo, W. and Restivo, J., 2005a. 'ADACOR: A Collaborative Production Automation and Control Architecture'. *IEEE Intelligent Systems*, Volume 20, No. 1, pp. 58-66.

Leitao, P. and Restivo, F., 2006. 'ADACOR-A Holonic Architecture for Agile and Adaptive Manufacturing Control'. *Computers in Industry*, Volume 57, pp. 121-130.

Lepuschitz, W., Zoilt, A., Vallee, M. and Merdan, M., 2011. 'Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents'. *IEEE*

Transactions on Systems, Man, and Cybernetics-Part C:Applications and Reviews, Volume 41, No. 1, pp. 52-68.

Marik, M. and Lazanky, J., 2007. 'Industrial Applications of Agent Techologies'. *Control Engineering Practice,* Volume 15, pp. 1364-1380.

Mehrabi, M., Ulsoy, A. and Koren, Y., 2000. 'Reconfigurable Manufactruirng and their Enabling Technologies'. *International Journal of Manufacturing Technology and Management,* Volume 1, pp. 113-130.

Mehrabi, M., Ulsoy, A., Koren, Y. and Heytler, P., 2002. 'Trends and Perspectives in Flexible and Reconfigurable Manufacturing Systems'. *Journal of Intelligent Manufacturing,* Volume 13, pp. 135-456.

Meng, F., Tan, D. and Wang, Y., 2006. *'Development of Agent for Reconfigurable Assembly System with JADE'.* Proceedings of the 6th World Congress on Intelligent Control and Automation, Dalian, China, pp. 7915-7919.

Monostori, L., Vancza, J. and Kumara, S. R. T., 2006. 'Agent-Based Systems for Manufacturing'. *Annals of the CIRP,* Volume 55, No. 2, pp. 697-720.

Oracle, 2012. *Oracle.* [Online]
Available at: docs.oracle.com
[Accessed 4 November 2012].

Padgham, L. and Winikoff, M., 2004. *'Developing Intelligent Agent Systems: A Practicle Guide'.* Sussex: John Wiley and Sons.

Pritschochw, G.,Altintas, Y., Jovane, F., Koren, Y., Mitsuishi, M., Takata, S., Van Brussel, H., Manfred, W. and Yamazaki, K., 2001. 'Open Controller Architecture: Past Present and Future'. *CIRP Annals-Manufacturing Technology,* Volume 50, No. 2, pp. 463-470.

Pritschow, G., Daniel, C., Junguhans, G. and Sperling, W., 1993. 'Open System Controllers: A Challenge for the Future of the Machine Tool Industry'. *Annals of the CIRP,* Volume 42, No. 1, pp. 449-452.

Rooker, M N.,Sunder, C., Strasser, T., Hummer, O. and Ebenhofer, G., 2007. 'Zero Downtime Reconfiguration of Distributed Automated Systems'. *Procceding of 3rd International Conference on Industrial Application of holoniv and Multi-agent system,* HOLOMAS 2007, pp. 326-337.

Sequeira, M., 2008. *'Conceptual Design of Fixture-Based Reconfigurable Spot Welding System',* MScEng Thesis, Department of Mechanical and Mechatronic Engineering: University of Stellenbosch.

Setchi, R. M. and Lagos, N., 2004. 'Reconfigurability and Reconfigurable Manufacturing Systems: State-of-the Art Review',. *2nd IEEE Conference on Industrial Informatics, INDIN '04,* pp. 529-535.

Shen, W., 2002. 'Distributed Manufacturing Scheduling Using Intelligent Agents'. *IEEE Intelligent Systems,* Volume 17, No. 1, pp. 88-94.

Strauss, R., 2009. *'Development of a Reconfigurable Parts Feeder for Automation',* BEng Final Year Project, Department of Mechanical and Mechatronic Engineering: University of Stellenbosch.

Ticky, P., Marik, V., Vrba, P., Macurek, F., Slechta, P., Straton, J R., Mautrana, P F. and Hall, H K., 2006. 'Deployment of Agent Technologies in Industrial Applications'. *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and its applications,* pp. 243-250.

Trentesaux, D., 2009. 'Distributed Control of Production System'. *Engineering Applications of Artificial Intelligence,* Volume 22, pp. 971-978.

Valckenaers, P., Van Brussel, H., Wyns, J., Bongaerts, L. and Peeters., 1998. 'Designing Holonic Manufacturing Systems'. *Robotics and Computer-Integrated Manufacturing,* Volume 14, pp. 455-464.

Valckenaers, P., Van Belle, J. and Ali, O., 2011. *'PROSA and Delegate MAS for Open-Air Engineering Processes'.* Leuven, Belgium, 2011 IEEE 16th Conference on Emerging Technologies and Factory Automation (ETFA).

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P., 1998. 'Reference Architecture for Holonic Manufacturing Systems: PROSA'. *Computers in Industry,* Volume 37, pp. 255-274.

Vrba, P., 2012. *'Past, Present and Future of Distributed Intelligent Control in Industrial Applications'.* Bucarest, INCOM 12.

Vrba, P., Ticky, P., Marik, V., Hall, H K., Straton, J R., Maturana, P F. and Kadera, P., 2011. 'Rockwell Automation's Holonic and Multiagent Control Systems Compendium'. *IEEE Transactions on Systems, Man,and Cybernetics,* Volume 41, No. 1, pp. 14-30.

Vyatkin, V., 2007. *'IEC 61499 Function blocks for Embedded and Distributed Control Systems Design'.* New Zealand: O3 neida.

Wiendahl, P. H., EIMaraghy, H A., Nyhuis, P., Zah, M F., Wiendahl, H H., Duffie, N. and Brieke, M., 2007. 'Changeable Manufacturing-Classification, Design and Operation'. *Annals of the CIRP,* Volume 56, No. 2, pp. 783-809.

Wooldridge, M. and Jennings N, R., 1999. 'Software Engineering with Agents: Pitfalls and Pratfalls'. *IEEE Internet Computing,* Volume 3, No. 3, pp. 20-27.

Xuemei, H., 2009. 'Intelligent and Reconfigurable Control of Automatic Production Line by Applying IEC 61499 Function Blocks and Software Agents'. *IEEE International Conference on Mechatronics and Automation,* pp. 1481-1486.

APPENDIX A: CELL CONTROLLER FUNCTIONAL ANALYSIS

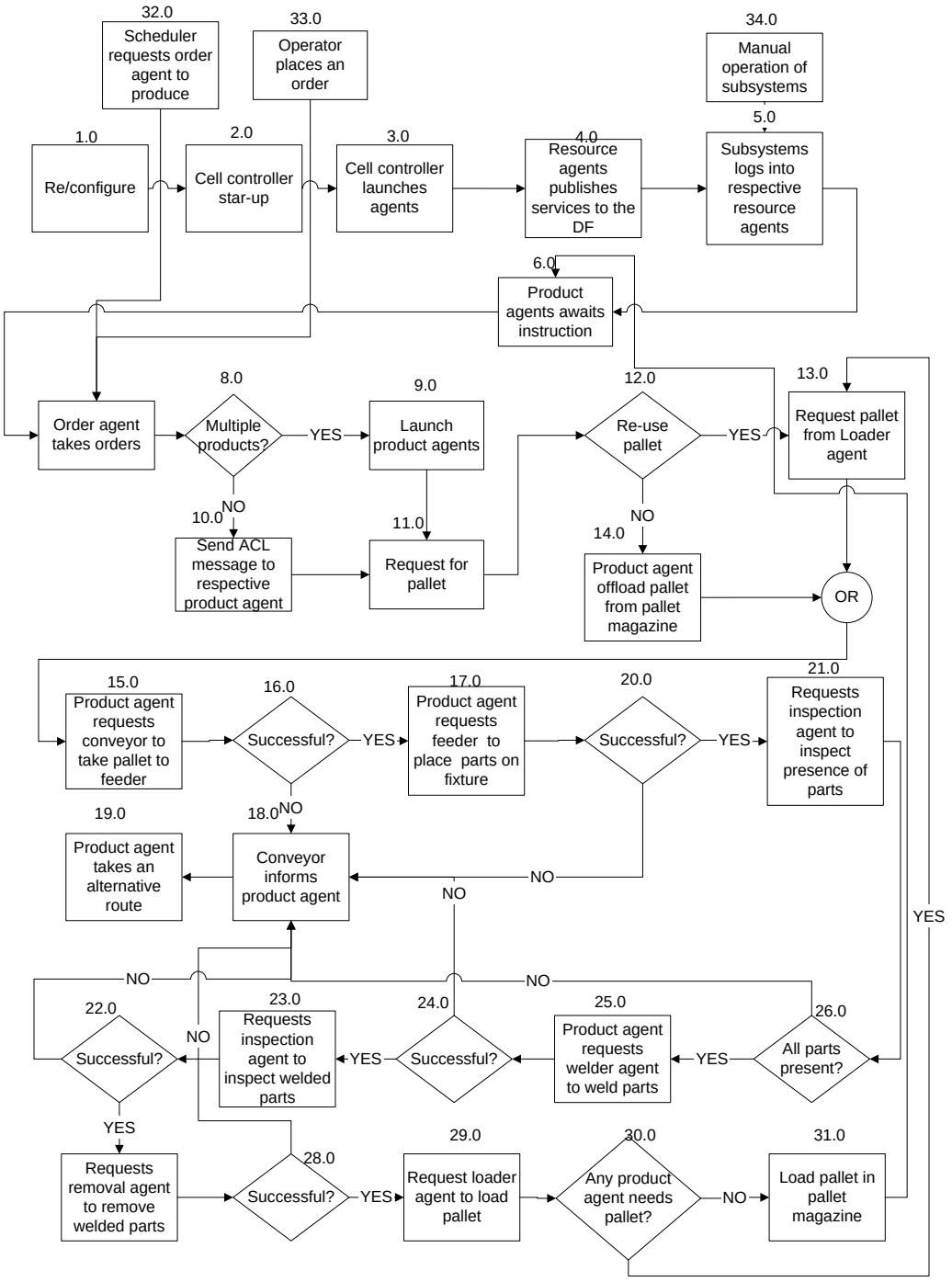


Figure A.1 Cell controller functional analysis

APPENDIX B: MODULAR CARTESIAN ROBOT CIRCUITS

B.1 CMMP-AS power connection pin

Pin no.	Description	Value	Specification
1	BR -	0V Brake	Holding brake (Motor), signal level depends on switching state, High side / Low side switch
2	BR+	24V Brake max. 1 A	
3	PE	PE	Cable shield for the stopping brake and the temperature sensor (not used in Festo cables)
4	-MTdig	GND	Motor temperature sensor, NC, NO, PTC, KTY...
5	+Mtdig	+3.3 V / 5 mA	
6	PE	PE	Protective conductor from motor
7	W	0...270 V_{eff}	Connection for the three motor phases
8	V	0...2.5 A_{eff} (CMMP-AS-C2-3A)	
9	U	0...5 A_{eff} (CMMP-AS-C5-3A) 0...1000 Hz	

Figure B.1 CMMP-AS Three phase power connection pin assignment **(**FESTO**,** 2012b**)**

B.2 Control circuit

The control circuit shows wiring for the three axes. NC pins to the relay board indicate signal input from the servo drives used as feedback to coordinate motion of the three axes. NO from the relay board is used for input to the drives from the control programme.

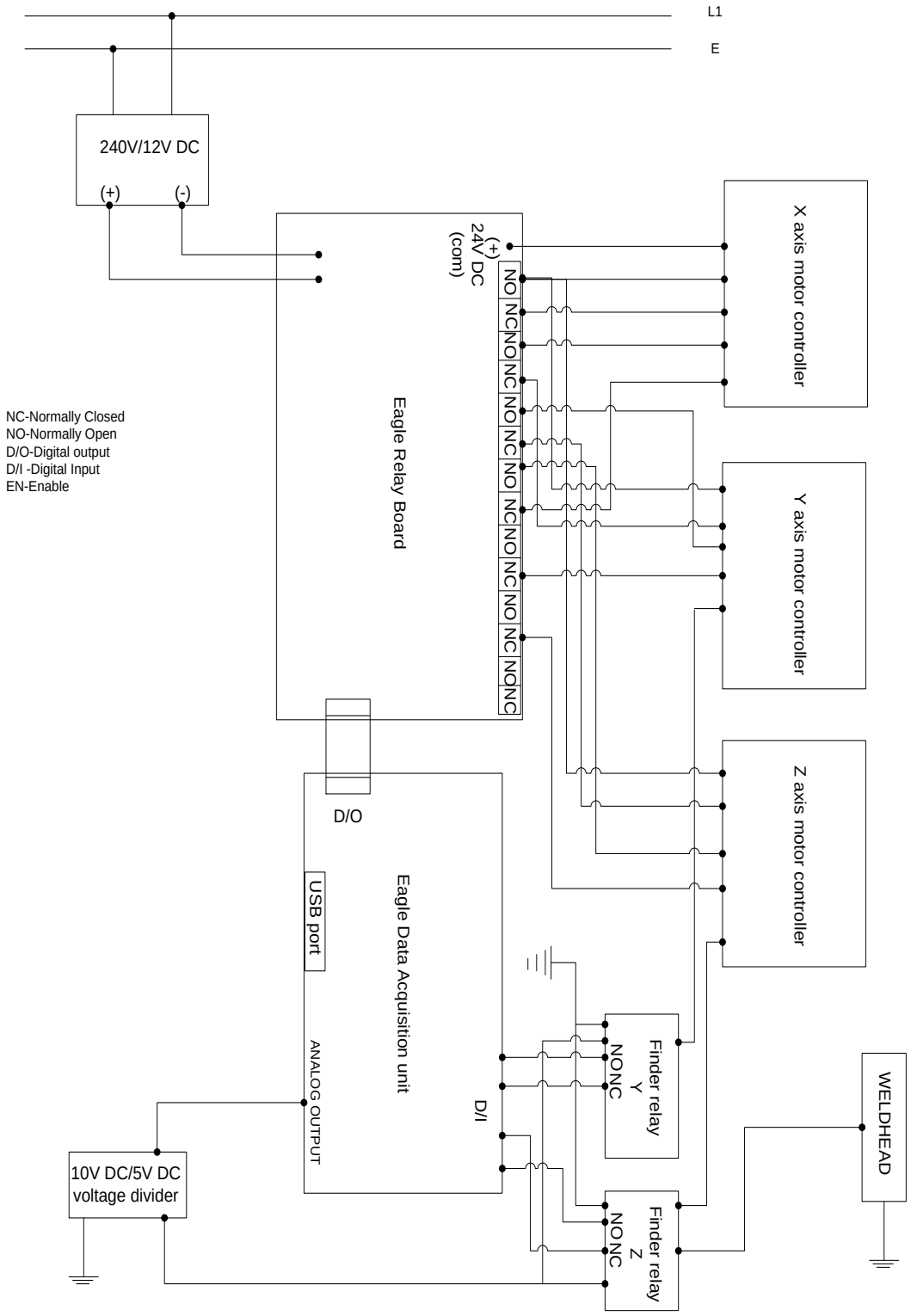


Figure B.2 Modular Cartesian robot control circuit

B.3 Mains supply

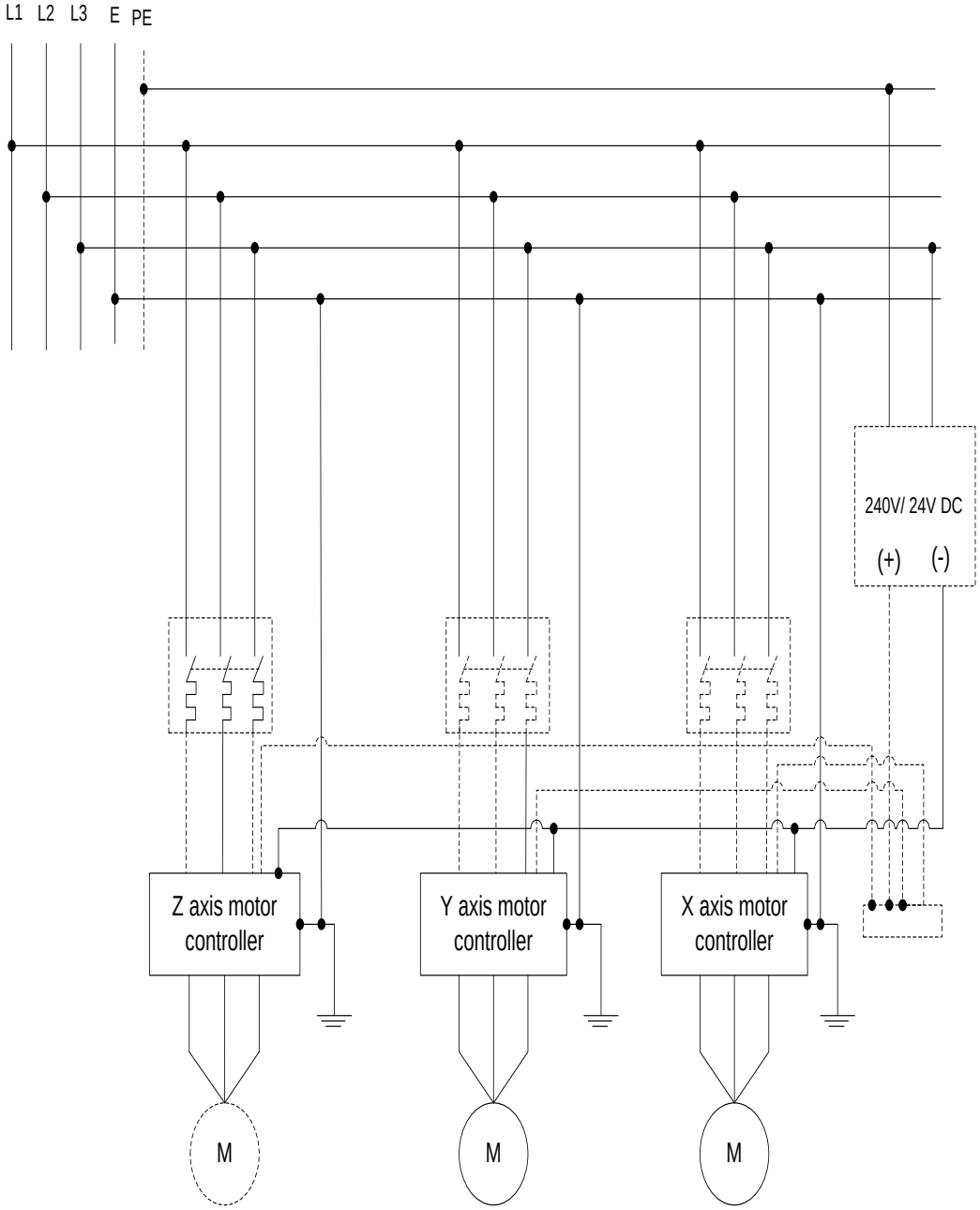


Figure B.3 Power circuit connection

APPENDIX C: MODULAR CARTESIAN ROBOT CONTROL

C.1Function block high level control

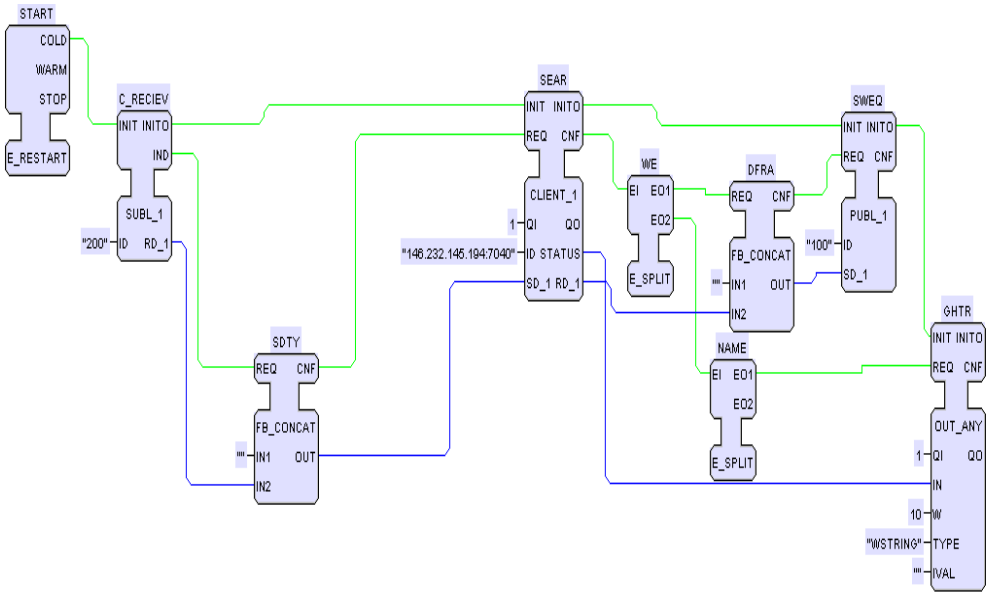


Figure C.1 Modular Cartesian robot function blocks

C.2Modular Cartesian robot functional Analysis

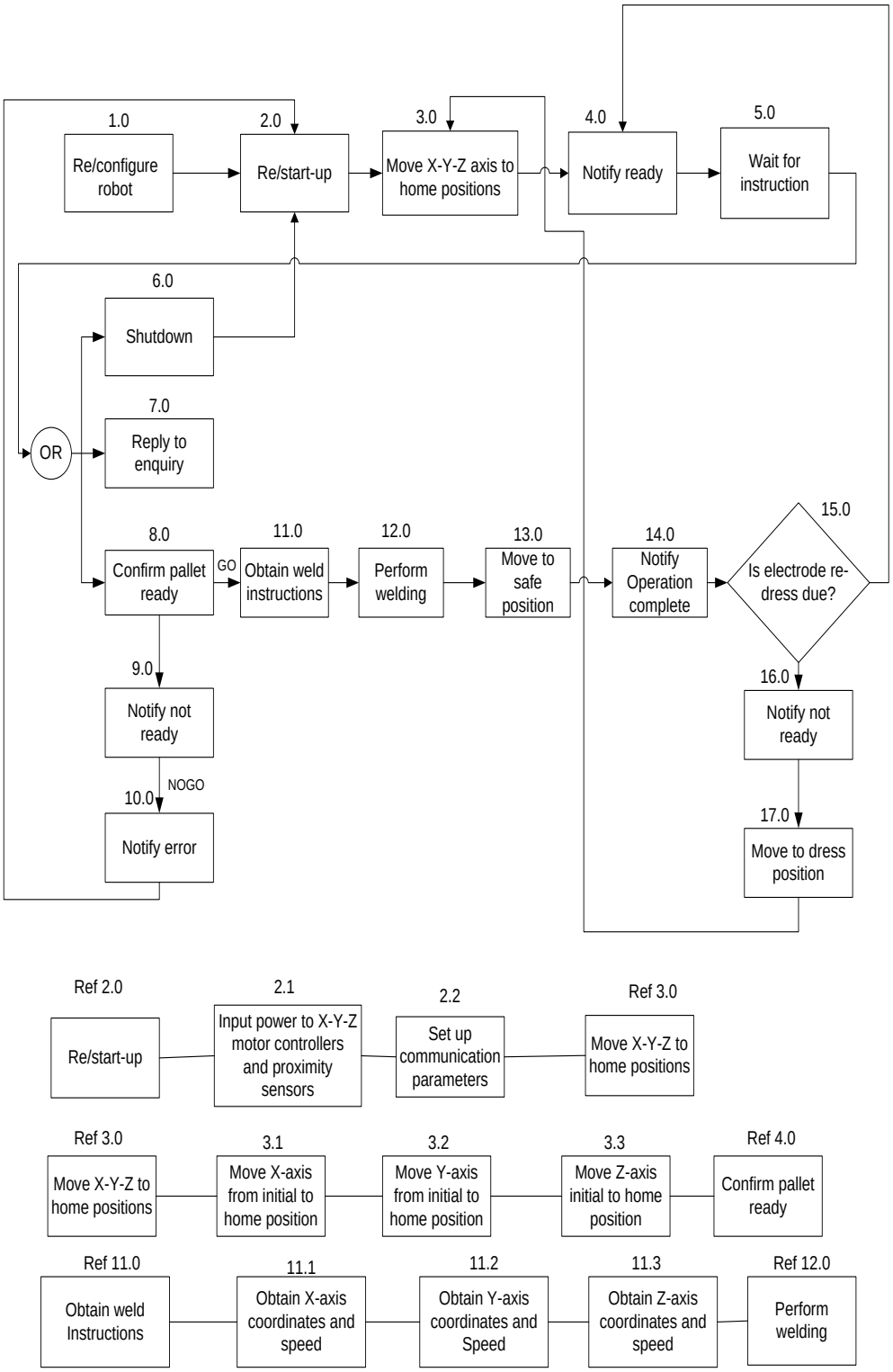


Figure C. 2a Modular Cartesian robot functional analysis

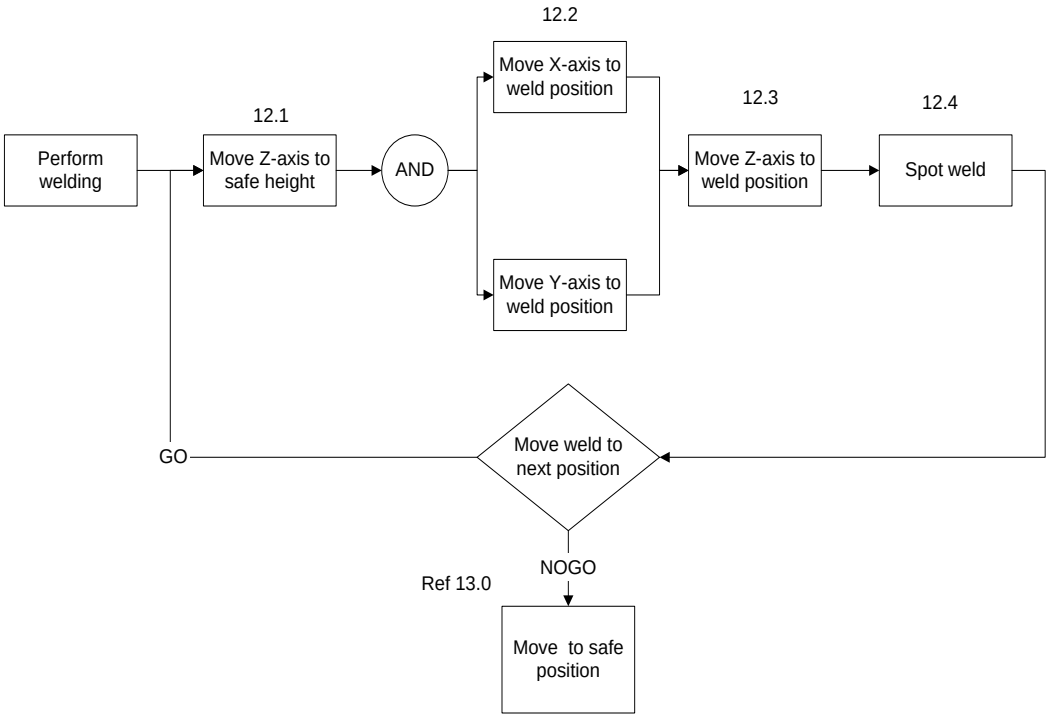


Figure C.2b Modular Cartesian robot functional analysis

APPENDIX D: CELL CONTROLLER PORTS AND DATA EXCHANGE FORMATS

D.1Port designation of subsystems

Port assigned to each Agent after a colon [:]

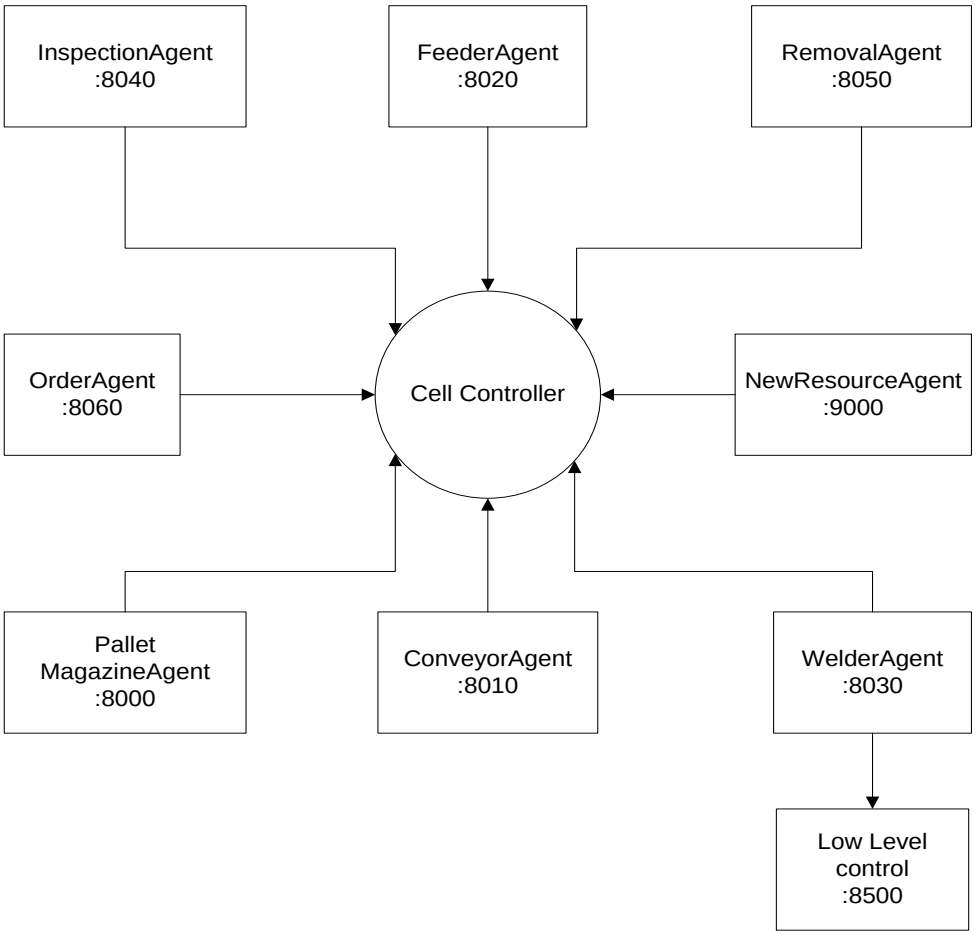


Figure D.1 Agent ports

D.2 Messaging formats for subsystems

The general messaging format between the cell controller and the conveyor is:
"Descriptor, Job ID, from which station, to which station;".

An example of a typical command to move a pallet from the pallet station to the feeder station is CC_MOVING, 4, 1, 2. Note that the semicolon is part of the command. The CC_MOVING is the descriptor, 4 is the Job ID, 1 is the station from where the pallet is taken from i.e. the pallet station, while 2 is the feeder station (where to take the pallet). The station numbers correspond with Figure 3.2

where each station is assigned a number as can be shown in the example as follows:

- pallet magazine is assigned one ,
- feeder station is assigned two
- inspection station is assigned
- removal station is assigned number two and
- welder station is assigned number five.

The commands are grouped into execution, diagnostics and startup/ shutdown. For instance, conveyor messages may start with CC_XXXX and for execution, HW_INTERFACE. All resource agents in cell controller understand the meaning of their respective subsystem semantics.

The welder also has a similar messaging scheme. For instance, in order to weld, when the modular Cartesian robot is in digital I/O configuration, the command "WELD, 1;" will trigger the whole wedding process. The "WELD" is an instruction, while the number one is the product type. The drives searches for the product type from the position set table and executes the command.

The feeder station uses the XML messaging format as:
<?xml version= "1.0" encoding= "UTF-16"?>
<CELLCONTROLLER><FEEDER><COMMAND>LOAD</COMMAND>
<PRODUCT>1</PRODUCT><NUMBOFTASKS>4</NUMBOFTASKS>
<TASK1>1</TASK1><X1>105.85</X1><Y1>150.6</Y1><Z1>27.77</Z1>
<A1>0.0</A1><TASK2>2</TASK2>……<TASK3>3</TASK3>…..
</FEEDER></CELLCONTROLLER>
All the coordinate positions are passed to the feeder for a particular product in this manner. Each task from the XML string has X, Y, Z and the angle denoted by A1 where 1 corresponds to the task number.

The pallet magazine has a messaging scheme with parameters that indicate the pallet type and where it must be offloaded. To offload a pallet, a command is of format "CM_UNLOADING, 0, 2;" and loading a pallet is of the format "CM_LOADING, 0, 2;"

D.3 Agent code for service description and publishing to the DF

```
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("Descrition of service type");
sd.setName(getLocalName() + "local name of the
agent");
dfd.addServices(sd);
try {
   DFService.register(this, dfd);
```

```
   } catch (FIPAException fe) {
      fe.printStackTrace();
   }
```

D.4 Agent code for searching for services

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(AgentType_to_search);
template.addServices(sd);
SearchConstraints sc = new SearchConstraints();
long maxDepth = 5;
long maxResults = 5;
sc.setMaxDepth(maxDepth);
sc.setMaxResults(maxResults);
try {
DFAgentDescription[] result = DFService.search(myAgent,
template, sc);
 ResourceAgents.clear();
for (int i = 0; i < result.length; ++i) {
     ResourceAgents.addElement(result[i].getName().getL
ocalName().toString());
 System.out.println(ResourceAgents.elementAt(i));
 }

} catch (FIPAException fe) {
          fe.printStackTrace();
                    }
if (!ResourceAgents.isEmpty()) {
     System.out.println(ResourceAgents.elementsAt(0)+":
Found");
}else {
System.out.println("Agent with service :" +
AgentType_to_search + ": was found");
ACLMessage sfd = new ACLMessage(ACLMessage.INFORM);
sfd.addReceiver(new AID("CellControllerAgent",
AID.ISLOCALNAME));
sfd.setContent("AgentType_to_create");
this.send(sfd);
}

ACLMessage msg = new ACLMessage(ACLMessage.CFP);
msg.setContent(Ready_msg);
Iterator<String> it = ResourceAgents.iterator();
while (it.hasNext()) {
String ResourceAgent = (String) it.next();
msg.addReceiver(new AID(ResourceAgent,
AID.ISLOCALNAME));
```

```
ResourceAgents.removeElement(it);
}
msg.setProtocol(FIPANames.InteractionProtocol.FIPA_CONT
RACT_NET);
```

D.5 Code for creating multiple agents

```
for (int j = 0; j < numberOfOrders; j++) {
CreateAgent ca = new CreateAgent();
ca.setAgentName(agentName + j);
ca.setClassName(agentType);
ca.setContainer(new
ContainerID(AgentContainer.MAIN_CONTAINER_NAME, null));
Action actExpr = new Action(getAMS(), ca);
ACLMessage request = new
ACLMessage(ACLMessage.REQUEST);
request.addReceiver(getAMS());
request.setLanguage(slCodec.getName());
request.setOntology(JADEManagementOntology.NAME);
request.setProtocol(FIPANames.InteractionProtocol.FIPA_
REQUEST);
try {
     getContentManager().fillContent(request, actExpr);
System.out.println("Request sent");
addBehaviour(new AchieveREInitiator(myAgent, request) {
private static final long serialVersionUID = 1L;
protected void handleInform(ACLMessage inform) {
System.out.println("Agent successfully created");
}
protected void handleFailure(ACLMessage failure){
System.out.println("Error creating agent.");
}
});
} catch (Exception e) {
   e.printStackTrace();
   }
}
```

D.6 Code for re-launching an agent

```
CreateAgent ca = new CreateAgent();
ca.setAgentName(agentName);
ca.setClassName(agentType);
ca.setContainer(new
ContainerID(AgentContainer.MAIN_CONTAINER_NAME, null));
Action actExpr = new Action(getAMS(), ca);
ACLMessage request = new
ACLMessage(ACLMessage.REQUEST);
request.addReceiver(getAMS());
```

```
request.setOntology(JADEManagementOntology.NAME);
request.setLanguage(FIPANames.ContentLanguage.FIPA_SL);
request.setProtocol(FIPANames.InteractionProtocol.FIPA_
REQUEST);
try {
      getContentManager().fillContent(request, actExpr);
       System.out.println("Request sent");
       addBehaviour(new AchieveREInitiator(myAgent,
   request) {
      private static final long serialVersionUID = 1L;
           protected void handleInform(ACLMessage
   inform) {
      System.out.println("Agent successfully created");
        }
      protected void handleFailure(ACLMessage failure) {
      System.out.println("Error creating agent.");
       ACLMessage msg=new ACLMessage(ACLMessage.INFORM);

   msg.addReceiver(new AID("WorkOrderAgent",AID.ISLOCAL
   NAME));
      msg.setContent(agentName);
      msg.setOntology(JADEManagementOntology.NAME);
       myAgent.send(msg);
       }
   });
} catch (Exception e) {
      e.printStackTrace();
   }
   stop();
}
```