Pallet routing strategies for a reconfigurable conveyor

by
Pieter-Jan Havenga

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

Supervisor: Prof Anton Herman Basson

March 2017

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.

Date: March 2017

Abstract

A reconfigurable manufacturing system is characterized by high customizability and a high throughput rate. The research presented in this thesis considers a reconfigurable approach to the path planning of a palletized conveyor. Reconfigurability here implies minimizing pallet transportation time by selection of the quickest routes.

Previous research by the Mechatronics, Automation and Design Research Group of Stellenbosch University developed a holonic controller for a palletized, modular conveyor. The focus in this thesis was to adapt this controller to include the functionality of path optimization.

Simulation software was used to simulate the palletized conveyor and construct a path optimization tool. Simio was the simulation software used. The virtual conveyor setup was constructed in Simio and, together with the Simio application programming interface (API), enabled predicting the travelling times of pallets on the conveyor. For the virtual conveyor the PLCs operating the conveyor were substituted by DLLs which were written in C#. This substitution of the hardware was to ensure that with only minor changes, the existing controller could also control the virtual conveyor. The predicted travelling times for the different routes were recorded, from which the quickest path was selected.

Experiments were conducted on an experimental setup of a reconfigurable assembly system to verify that reconfigurability was rapid and reliable. A traffic junction was set up to compare performances of the controller without path optimization with the controller which has the path optimization installed. The results confirmed that simulation can be considered as a way to add reconfigurability to the path planning of a conveyor in a manufacturing system. Latency, however, proved to be a concerning factor when utilizing a holonic controller in running simulations.

The holonic controller integrated with the simulations, was substituted with a controller that was developed in the simulation software itself. This approach alleviated the problem of latency found in the previous approach, and lent credibility to having a simulation controller as an optimizing tool to the path planning of a reconfigurable conveyor.

Uittreksel

‘n Herkonfigureerbare vervaardigingstelsel word gekenmerk deur hoë aanpasbaarheid en ‘n hoë deurset. Die navorsing in hierdie tesis oorweeg ‘n herkonfigureerbare benadering tot die roetebeplanning van ‘n vervoerband met pallette. Herkonfigureerbaarheid impliseer hier die minimering van die tyd wat dit ‘n pallet neem om van een punt na ‘n volgende te beweeg, deur gebruik te maak van die vinnigste roetes.

Vorige navorsing, gedoen deur die Megatronika, Outomatisasie en Ontwerp Navorsingsgroep van die Universiteit van Stellenbosch, was om ‘n holoniese beheerder te ontwikkel vir ‘n modulêre vervoerband met pallette. Die fokus in hierdie tesis was om hierdie beheerder aan te pas om die roete optimering funksionaliteit in te sluit.

Simulasie sagteware is gebruik om die vervoerband met pallette te simuleer en ‘n roete optimering strategie te ontwikkel. Simio is die sagteware wat gebruik was. Die virtuele vervoerband was opgestel in Simio en het, tesame met die Simio program se programmeerbare intervlak, dit moontlik gemaak om die vervoertye van pallette op die vervoerband te voorspel. Vir die virtuele vervoerband is die PLCs wat die vervoerband beheer het, met DLLs vervang wat in C# geskryf was. Hierdie vervanging van hardeware het verseker dat, met min aanpassings, die bestaande beheerder ook die vervoerband kan beheer. Die vervoertye wat voorspel is vir die verskillende roetes, is opgeneem en die vinnigste roete is gekies.

Eksperimente is uitgevoer op ‘n eksperimentele opstelling van ‘n herkonfigureerbare monteringstelsel om vas te stel dat herkonfigureerbaarheid vinnig en betroubaar geskied. ‘n Verkeersaansluiting is gebruik om die beheerder met roete optimering funksionaliteit te demonstreer. Die resultate het aangedui dat simulasie wel as ‘n moontlikheid oorweeg kan word om herkonfigureerbaarheid toe te voeg tot die roete beplanning van ‘n vervoerband met pallette in ‘n vervaardigingstelsel. Vertragings van programme was ‘n beduidende faktor wat geobserveer was gedurende die gebruik van ‘n holoniese beheerder in simulasies.

Die holoniese beheerder wat geïntegreerd was met simulasies, is vervang met ‘n beheerder wat ontwikkel is slegs in die simulasie sagtware. Hierdie benadering het die probleem van vertragings aangespreek, en kredietwaardigheid verleen aan die benadering van simulering tot die optimering van roetebeplanning vir ‘n herkonfigureerbare vervoerband met pallette.

Acknowledgements

Firstly I would like to thank Prof. A.H. Basson for your continued guidance, and being readily available. Your vast knowledge assisted in not only solving problems, but has deepened my interest in the field of mechatronics. Next, I'd like to thank fellow research group members M.M. Deacon and C.S. van den Bergh for providing the pleasant working environment, and K. Kruger and R. Rodrigues for the experience and knowledge shared. Lastly I'd like to thank my family and friends for whom I am so grateful for, and providing me with a sound support basis for completing this journey.

This is all due to the grace of our heavenly Father.

Table of Contents

List of figures

List of tables

Nomenclature

ABM	-	Agent based modelling
ADACOR	-	ADAptive holonic COntrol aRchitecture
API	-	Application programming interface
DLL	-	Dynamic link library
GUI	-	Graphical user interface
HLC	-	High-level control
HMI	-	Human machine interface
HMS	-	Holonic manufacturing system
LLC	-	Low-level control
MADRG	-	Mechatronics, Automation and Design Research Group
PLC	-	Programmable logic controller
PROSA	-	Product-resource-order-staff architecture
RMS	-	Reconfigurable manufacturing system
WCF	-	Windows Communication Foundation

1. Introduction

1.1 Background

The manufacturing world is an ever changing environment. As labour markets are becoming increasingly volatile, with strikes for higher remuneration occurring frequently, renewed interest is shown toward automating aspects of production.

The past was characterized by dedicated (DMSs) and flexible (FMSs) manufacturing systems, each participating in a trade-off of productivity versus variability. Production uncertainty is yet another challenge that has emerged and Keshavarzmanesh *et al.* (2010) evaluates dynamic responsiveness, i.e. capability to react in real-time, to be a more favourable approach than forecasting under these circumstances.

The reconfigurable approach to manufacturing is yet to be proven a sustainable option to the future of manufacturing solutions, hence industry's reluctance to implementing reconfigurable manufacturing systems (RMSs). The research presented in this thesis is a step towards the adoption of RMSs by considering an approach to the route planning of a modular conveyor. The concept of RMSs was developed by Koren and other researchers at the University of Michigan (Koren and Shpitalni, 2010). Designed at the outset for having low ramp-up time and producing fluctuating production volumes, RMSs could be the solution to the problem South Africa's manufacturing industry is faced with - the inability to efficiently produce small production volumes in response to fluctuating demand.

CBI Electric: Low voltage is a Lesotho-based manufacturer of trip switches and is an embodiment of the South African manufacturing climate. A concept RMS cell for the testing of the company's Q-frame assembly is used as context in this thesis. The RMS cell is an arrangement of manufacturing stations along a palletized conveyor and is used for various research projects by the Mechatronics, Design and Automation Research Group (MADRG) at Stellenbosch University. Only the conveyor will be considered in this thesis. Related research work includes Kruger and Basson (2016) evaluating Erlang for implementing a holonic cell controller and Graefe (2016) considering C# for implementing a station controller. Other work pertaining to the conveyor is Kotzé's (2016) development of a holonic conveyor controller within the ADACOR framework (Leitão *et al.*, 2005). The research conducted in this thesis builds on Kotzé's work by introducing a strategy to obtain optimal throughput within the ADACOR holonic structure. C.S. van den Berg is currently considering Erlang for implementing 'ants', as an alternative to ADACOR, in a controller for obtaining optimal throughput on an RMS conveyor.

1.2 Motivation

A reconfigurable system requires all aspects forming part of the system to be reconfigurable. Enabling a conveyor transportation system to be reconfigurable could therefore be essential to enabling an RMS to be reconfigurable.

A key aspect of a reconfigurable approach to a conveyor is the scheduling of pallet routes with the goal of avoiding delays and transporting products from starting position to destination in the least amount of time. Efficient pallet routing, taking other pallets into account, is required to alleviate throughput bottlenecks, ultimately resulting in increased revenue.

1.3 Objectives

The objective in this thesis is to develop a robust[1] pallet routing strategy for a palletized conveyor in an RMS cell. The routing should achieve the highest possible throughput rate by considering an optimal route for a single pallet, upon entering the conveyor, while taking the future movements of previously routed pallets into account. If the conveyor is reconfigured, it should be easy and quick to adapt the strategy to the new configuration.

1.4 Scope

From the outset, it was decided that the strategy will be implemented in a conveyor controller developed by Kotzé (2016) for an RMS cell, to facilitate validation and testing. This controller can easily be adapted if the conveyor is reconfigured due to the controller's holonic architecture. Further, it was also decided from the outset that the strategy will use existing simulation software for forecasting future pallet traffic, so that this approach can eventually be compared with the ants-based strategy being developed by C.S. van den Berg, as mentioned above.

In the work presented here, the strategy will only consider allocating a pallet an optimal route upon entering the conveyor. The pallet’s optimal route will be determined by taking into account what the current state of the conveyor is at that moment, including the previously determined routings for each pallet already in the system. The route allocated to the pallet entering the system remains unchanged for the remainder of a pallet’s journey along the conveyor. Future work might include affording pallets more agility in routing decisions - being able to alter routing decisions during the course of moving along the conveyor, in order to increase optimality without being oversensitive.

[1] In this thesis "robustness" is considered to include disturbance rejection

Reconfigurability is a key part of the context considered here. This thesis will primarily focus on the RMS property of scalability (RMS properties are discussed in Section 2.1.4), since the other RMS properties will then implicitly also be included. Kotzé's controller can easily accommodate this type of hardware changes which, in turn, requires software changes in the control setup.

'Robustness' in this thesis will be used to refer to the ability of routing strategies to accommodate predictable and unforeseen events that occur on the conveyor while in operation, such as traffic congestion and failures.

1.5 Research method

In formulating the pallet routing strategy, the possibility of controller software reconfiguration was considered. Therefore, initially, a pallet routing strategy was selected in which the controller developed by Kotzé (2016) controlled a virtual conveyor (represented by a simulation), with the virtual conveyor and its controller running faster than real time, to be able to predict future movements of pallets. The virtual conveyor and controller can therefore be used to evaluate alternative routes for a pallet to find the optimum path, taking into account the simulation's prediction of movements of pallets already on the conveyor. This approach is called the "Virtual holonic controller approach".

Subsequent to testing the routing strategy incorporating Kotzé's controller, which revealed a significant amount of latency, an alternative strategy, here called the "Simio controller approach" was also developed and tested. In the "Simio controller approach", Kotzé's controller is not used in the routing strategy (but is still used for controlling the physical conveyor).

The above two strategies are explained in greater detail in Chapter 3.

A significant limitation that had to be taken into account in the research method was that only a conveyor of a relatively simple RMS cell (Figure 1) was available to perform physical testing and validation of the pallet routing strategy. The research method therefore entailed two main phases: In the first phase, the pallet routing strategy was developed and tested in the RMS cell, using the control hardware and software developed by Kotzé (2016) for the "Virtual holonic controller approach". Although the conveyor in this cell allowed for some alternative routes between some stations, these alternatives were very limited.

In the second phase, the strategy was tested in a more realistic conveyor scenario with a much larger conveyor, but with the conveyor hardware replaced by a simulation that represented the real-time behaviour of a conveyor. It must be emphasised that the simulation referred to here for the second phase, representing the

real-time behaviour, was distinct from the simulation used in the routing strategy. The high-level controller developed by Kotzé was still used in the “Virtual holonic controller approach”, while in the “Simio controller approach”, the control logic was applied by the simulation software.

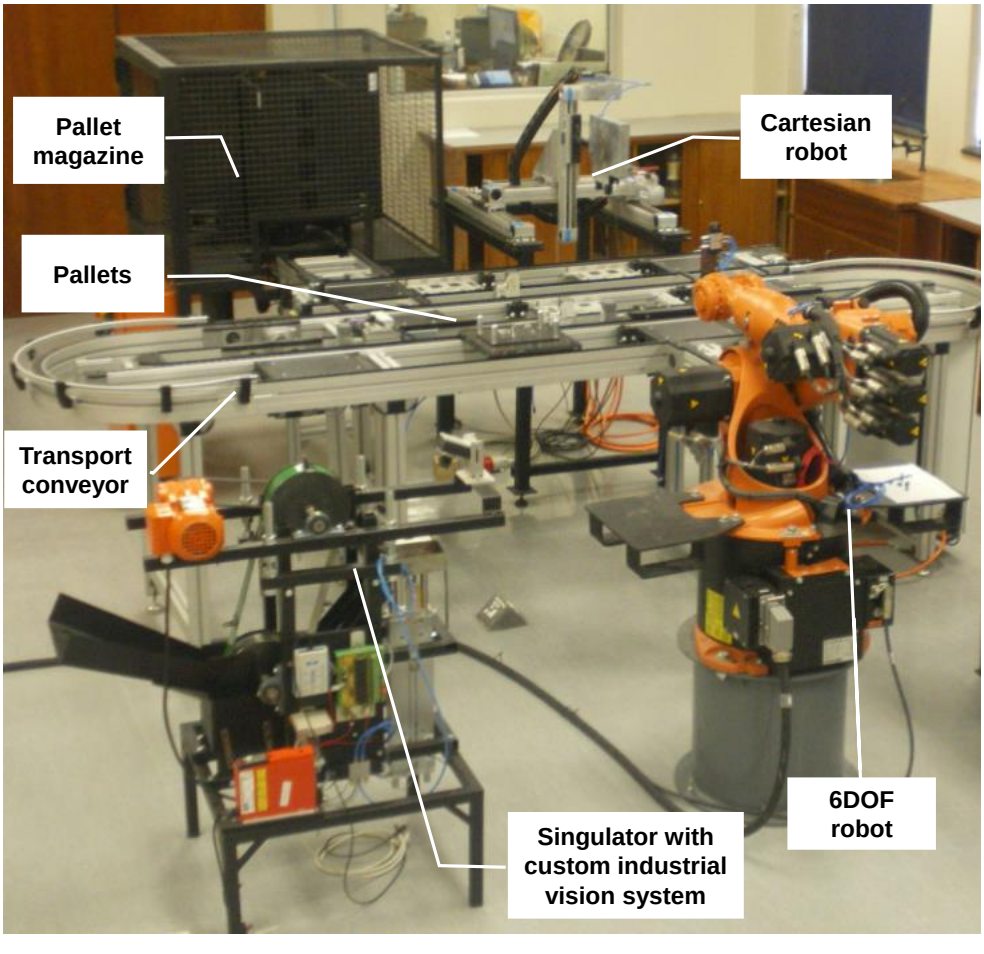

Figure 1 RMS cell setup

In the testing phase, it was deemed sufficient to prove that optimal decisions are indeed taken by both the “Virtual holonic controller approach” and the “Simio controller approach”. The two conveyor layouts mentioned above (referred to as ‘laboratory’ and ‘complex’ setups) were deemed suitable for proving the optimization capabilities of the two strategies. The laboratory setup contains two possible routes for a pallet to follow, whilst the complex setup has numerous options. The second, complex layout also provided the opportunity to demonstrate the scalability capability of the two strategies. Demonstrating scalability on additional

layouts would not add any generally applicable results, and only case-specific results.

Testing comprised first validating the model, by showing that the simulated conveyor's behaviour is close to the physical conveyor's behaviour (< 15% error). Then the ability of the routing strategies (respectively for the "Virtual holonic controller approach" and the "Simio controller approach") to find an optimum route was tested in a number of typical scenarios, including unpredictable circumstances.

1.6 Thesis overview

Chapter 1 provides an introduction to the problem and Chapter 2 contains a review of the relevant literature to address this problem. After having considered the literature, the "Virtual holonic controller approach" and the "Simio controller approach" are discussed in Chapter 3. The reader is introduced to the Simio concepts that were applied to the pallet routing strategies, in Chapter 4. Chapter 5 provides more detail on the pallet routing strategies discussed in Chapter 3. Chapter 6 reports on the tests that were conducted, whilst Chapter 7 provides closing remarks and considerations for future work.

2. Literature review

2.1 Manufacturing systems overview

Global economic competition and rapid technological advances in manufacturing have highlighted the importance of responsiveness in the manufacturing environment. Other challenges such as customization, cost-effectiveness, reliability, simplicity and non-obsolescence have also been identified but are predominated by responsiveness. Setchi and Lagos (2004) define responsiveness as being "the ability of a production system to respond to disturbances which impact upon production goals". Koren and Shpitalni (2010) add to this definition by stating that all manufacturing operations be done rapidly and cost-effectively in reaction to changes in market, product and system uncertainties such as failures. Generally, manufacturing uncertainty exist either internally or externally. Internal manufacturing uncertainty relates to equipment breakdown, job delay, rework, etc., while external uncertainty could be product price, product mix, product demand, etc. (Keshavarzmanesh *et al.*, 2010).

2.1.1 Dedicated manufacturing systems

Manufacturing paradigms originated as dedicated manufacturing systems (DMSs), being characterized by mass production, with products having good quality and low cost. DMSs see the operation of several tools simultaneously in machining stations. Profitability is only reached when these systems are used at full capacity and demand remains constant (Setchi and Lagos, 2004) - a rarity at present. Koren and Shpitalni (2010) therefore criticize DMSs as being too costly and requiring too much effort.

2.1.2 Cellular manufacturing systems

Cellular manufacturing systems (CMSs) utilise cells consisting of a group of machines arranged so that a product could be processed progressively and not waiting for a batch to be completed. CMSs are not suited to fluctuating market conditions due to limited flexibility in cases of machine breakdowns or changes in the product mix. Virtual cell formation and hybrid cellular layouts are possible solutions to this problem. A virtual cell is a dynamically formed cell in the system control software, in which machines are configured logically and temporarily, while a hybrid layout is a combination of functional and cellular layouts which do not influence the allocation of cells to part families. (Keshavarzmanesh *et al.*, 2010)

2.1.3 Flexible manufacturing systems

Flexible manufacturing systems (FMSs) realize quality products and economies of scope (i.e. economic when production volumes are low and a large variety of parts are produced). Although FMSs do not have the productivity of DMSs and require high initial investment cost, high flexibility is achieved by designing a FMS for a large set of operations. FMSs, such as CNCs, make use of propriety control systems

and therefore modifications to the system are either impossible or costly (Mehrabi *et al.*, 2000). Factors such as software complexity, lack of reconfigurability, investment and maintenance costs and rapid obsolescence have contributed to a decline in the use of FMSs (Mehrabi *et al.*, 2000). FMSs are often associated with wasted resources; supplemented by lean manufacturing principles, these waste forms might be resolved. Sallez *et al.* (2009) describe FMSs applied to routing problems as being stochastic and time-variable in nature.

2.1.4 Reconfigurable manufacturing systems

One of the earliest forms of an RMS was when Gerald Estrin proposed the idea of a 'fixed plus variable structure computer' which made use of reconfigurable hardware, in 1960 (Setchi and Lagos, 2004). Defined as the ability to rapidly change hardware and software components, reconfigurability allows for a swift adjustment of production capacity and functionality within a part family, in a cost effective way. Koren and Shpitalni (2010) identify the key features of an RMS as:

- Customization: Machine flexibility, limited to a single product family.
- Convertibility: Ease with which functionality is transformed, i.e. economically converting to a new set of production requirements by repositioning individual modules, without changing a system's topological characteristics.
- Diagnosibility: Diagnose root cause of output product defects and correcting the defects.
- Integrability: Integrate modules rapidly and precisely by a set of mechanical, control and informational interfaces.
- Modularity: Compartmentalization of operational functions into units.
- Scalability: Effortlessly modifying production capacity.

Modularity, integrability and diagnosibility reduce effort which translates to reduction in configuration time while customization, scalability and convertibility reduce cost (Koren *et al.*, 1999). Setchi and Lagos (2004) consider modularity to be the key enabler to reconfiguration; however, hardware modularity remains difficult to achieve.

Reconfiguration can be static or dynamic. Static reconfiguration is reconfiguration before operations start, while dynamic reconfiguration is reconfiguration of a portion of a device whilst other portions perform operations.

Open architecture controllers are associated with RMSs (Koren *et al.*, 1999). The benefit of having open architectures is that applications will be able to run on multiple platforms, interoperating with other system applications, and using a consistent style of interaction with the user.

RMSs' configurations are a function of reliability, machine speed, machine mix, desired volumes, part quality and cost (Koren *et al.*, 1999). The instalment of these configurations does however require large capital investments.

2.2 Control architectures

2.2.1 Centralized control architectures

Traditional manufacturing control systems are based on centralized and hierarchical control structures that present good production optimization, but weak flexibility (Leitão, 2009). This control structure focuses the processing power of shop-floor control under one central node. The resultant data complexity, however, requires a large amount of computing time (Borangiu *et al.*, 2015).

2.2.2 Decentralized control architectures

Decentralized control systems respond better to perturbations - failure of an isolated entity only affects part of the system, whilst other parts continue to operate without disruptions. Capabilities of this architecture include responding to change duly, without requiring external intervention (Leitão, 2009). A decentralized approach considers local data to be up-to-date, but leads to myopia which increases the need for global data whenever possible (Barbosa *et al.*, 2015).

2.2.3 Heterarchical control architectures

Heterarchical control architectures rely upon cooperation and negotiation among autonomous, intelligent entities which are capable of interacting with one another. This architecture achieves a combination of cost efficiency (associated with centralized control) and agility (associated with decentralized control). However, a drawback of this architecture is decision-myopia, which results in non-optimal long-term solutions (Borangiu *et al.*, 2015). Barbosa *et al.* (2015) reiterate that unexpected situations, such as resource malfunctioning and rush orders, have heterarchical architectures perform better.

2.2.4 Holonic control architectures

In the middle sixties Arthur Koestler (1969) introduced the word *holon* to describe the basic unit of organization in living organisms and social organizations. A holon can represent a physical or logical activity and has information about itself and the environment, containing an information processing part and a physical processing part, when the holon represents a physical device. Kruger and Basson (2015) define a holon as *any component of a complex system that, even when contributing to the function of the system as a whole, demonstrates autonomous, stable and self-contained behaviour or functioning.* This definition translates to a holon being an autonomous and cooperative building block for transforming, transporting, storing

and validating information of physical objects in a manufacturing environment. A holarchy is a hierarchically organized system populated with self-regulating holons which achieves system goals through cooperation between holons. A holon can belong to multiple holarchies where it obeys fixed rules and objectives, which differs from the conventional hierarchical control structure (Leitão, 2009). A holonic manufacturing system (HMS) is an example of a holarchy (Barbosa *et al.*, 2015). Agent software technology can be used to implement holonic manufacturing concepts.

Holonic architectures, for the manufacturing domain, include PROSA (product-resource-order-staff architecture) (Van Brussel *et al.*, 1998), ADACOR (adaptive component based architecture) (Leitão *et al.*, 2005) and HCBA (holonic component based architecture) (Chirn and McFarlane, 2000). Others, such as AARIA (Van Dyke Parunak *et al.*, 1997) and MetaMorph (Maturana *et al.*, 1999) are aimed at the industrial domain. The ants approach, derived from nature and biology, to manufacturing is also an example of holonic manufacturing systems (Van Belle *et al.*, 2009).

The architectures of holonic control distinguishes between control as low-level and high-level. Low-level control processes real time data from sensors and actuators while high-level control is embodied by a holon (Leitão *et al.*, 2013). Benefits to using a holonic control architecture in a reconfigurable manufacturing environment include the reduction in complexity and cost, modularity, reusability as well as increased maintainability and reliability. Leitão (2009) and Leitão *et al.* (2013) regard some of the disadvantages to implementing HMSs as being:

- required investment,
- uncertain relative merits of the technology and absence of proof of real applicability,
- fear of using emergent technology due to the inability of new technology to respect contemporary industrial requirements for real-time capabilities,
- distributed/decentralized approaches that are difficult to apprehend,
- not as flexible as manually reconfigurable systems,
- design and implementation of reconfigurable production systems, together with their supervisory control systems, are complex tasks,
- performance of these manufacturing control systems requires decoupling and benchmarks - required to provide realistic test cases for the research community to test their developed systems, allowing to compare different production control approaches.

2.2.4.1 Adaptive holonic control architecture

ADACOR (ADAptive holonic Control aRchitecture) (Leitão *et al.*, 2005) is a holonic control architecture which is suited to distributed manufacturing systems.

The adaptive control approach evolves over time to combine global production optimization parameters with the agile reaction to disturbances by balancing between a hierarchical stationary state and heterarchical transient state (Barbosa *et al.*, 2015). The supervisor entities, as well as self-organization and learning capabilities associated with holons, support the dynamic evolution and reconfiguration of the organizational control structure (Leitão, 2009). The four types of holons defined by the ADACOR structure are (Barbosa *et al.*, 2015):

- Product holon: A repository of the knowledge to produce a part.
- Task holon: Manages real-time execution of production orders on shop floor.
- Operational holon: Physical resources available on the shop floor such as robots, operators and numerical machines.
- Supervisor holon: Introduces optimisation into the system.

2.2.4.2 Product-resource-order-staff architecture

The product-resource-order-staff (PROSA) (Van Brussel *et al.*, 1998) architecture is another variant of holonic reference architectures for manufacturing systems. This architecture is based on three basic types of holons (Leitão, 2009):

- Resource holon: Contains the resource and information processing part that controls the resource. Resource holons aim to maximize the return on the execution of their services (Barbosa *et al.*, 2015).
- Product holon: Holds process and product knowledge and contains all information about the product, but no knowledge about the created instances.
- Order holon: Represents the tasks in manufacturing systems by using negotiation techniques to get a product produced (Van Belle *et al.*, 2009). This holon is also responsible for tracking production progress (Barbosa *et al.*, 2015).
- Staff holon (considered a 'special' holon): Assists and advises other basic holons to reduce workload and decision complexity.

Figure 2 depicts the different holons, as well as the interactions among the different types of holons. Holons interact with each other by exchanging information. These interactions, according to Figure 2, are:

- Product-resource: Represents the process knowledge which determines how a certain process can be achieved through a certain resource.
- Order-product: Production knowledge pertaining to the production of a certain product (by utilizing certain resources).
- Order-resource: Process execution knowledge, i.e. information about the process of executing processes on resources (Kruger and Basson, 2015).

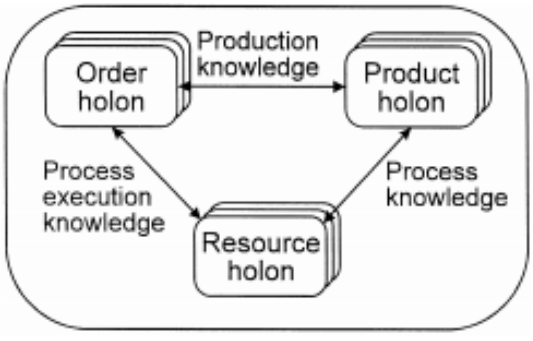

Figure 2 Basic holons of PROSA (Van Brussel *et al.*, 1998)

2.2.4.3 Multi-agent systems paradigm

Historically, the year 2005 is considered the end of the early stage of industrial applications of agent technology (Leitão *et al.*, 2013). Agent based computing (ABC) is the most popular approach to implement HMSs since software agents share many common properties with holons, such as a high degree of autonomy and cooperation capabilities. MASs (Multi-agent systems) are based on decentralized control employing these distributed, autonomous agents. Advantages of a MAS include scalability, reconfigurability and productivity but also robustness since it is not reliant on a centralized entity. Areas where agents have been successfully applied include production planning and scheduling, and logistics (Leitão *et al.*, 2013). Some of the disadvantages of MASs are questionable return on investment, lack of development tools and standards, lack in capacity to evolve, and the lack of skilled design, engineering and maintenance personnel (Leitão *et al.*, 2013). IEC 61499 is considered an alternative to agent technology for holonic manufacturing and seems a suitable solution for developing agent-based real-time and distributed control applications (Leitão, 2009).

2.3 Path planning algorithms

The purpose of this section is to introduce the reader to strategies that have been considered for path planning by literature. Most strategies proposed by literature seem to be a combination of existing methods, e.g. supplementing a potential field (PF) approach with the A* algorithm (Khuswendi *et al.*, 2011). The general tendency of these strategies is to have a map of the routes an entity is able to travel, and then finding the optimal route junction by junction. The progression in this section is from heuristic methods to metaheuristic methods, which can be applied to either time or distance domains.

2.3.1 Genetic algorithm

The genetic algorithm (GA) is a bio-inspired method based on the principle of natural selection. Each iteration of the method results in a "population" being more "fit" than the previous, thus a more optimal solution. This progression to becoming more "fit" may not always be the case when a randomness factor, referred to as "mutations", is introduced to the method.

The GA is considered by various papers on path planning, as a tool to supplement path optimization logic. A paper by Ulusoy *et al.* (1997) discusses how the GA was used in the routing logic of AGVs.

2.3.2 Backpressure routing strategy

According to Peng and McFarlane (2004), the backpressure routing algorithm (BPA) is a decision making strategy which can potentially provide good throughput performance when applied to a material handling system. McFarlane *et al.* (2001) consider the BPA to be specifically designed for congestion management. This algorithm is suited to handle arbitrary topologies, multiple traffic types and systems where looping of products is common, by making use of a pressure table/priority table. Ultimately, the highest pressure wins the bid at the intersection points.

The BPA is a class of algorithms from queueing theory, which does not require a set of parameters to be determined for each decision point in a plant (e.g. intersections). A publication by McFarlane *et al.* (2001) distinguishes among intersections as being one of four classes of base configurations (Figure 3), simplifying the process of finding the best route to follow. In this publication, the authors consider the achievement of high levels of throughput in an environment associated with highly flexible route selection. This objective is closely aligned with the objectives that have been outlined in Section 1.3.

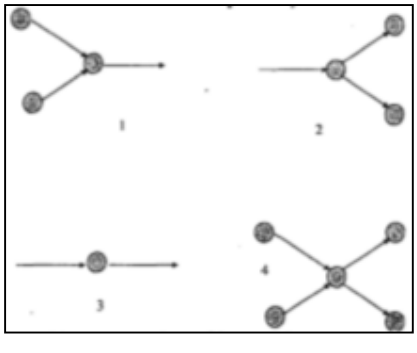

Figure 3 Basic routing configurations (McFarlane *et al.*, 2001)

2.3.3 Potential fields approach

The potential field (PF) method is modelled as a potential function consisting of an 'attraction field', pulling a vehicle towards a goal, and a 'repulsive field', ensuring a vehicle does not collide with obstacles. Attraction is directly proportional to angle to goal, and inversely proportional to exponentials of both distance and angle to the obstacle.

Mujumdar and Padhi (2011) identify the advantages of the PF method as being the efficiency with which destinations are reached, and the ease with which new obstacles can be added to the function. Disadvantages include getting trapped in local minima, and not providing the thoroughness associated with graph searching techniques (Warren, 1989).

Krogh and Thorpe (1986) and Tournassoud (1986) considered approaches employing geometrical solutions, for global path planning, and PF methods, for local path planning, around obstacles. Although this approach is more applicable to the UAV field of research, some of these concepts, such as navigating toward a goal whilst avoiding obstacles (e.g. path blockages), can be applied to the path planning of pallets on conveyors.

2.3.4 Dijkstra's algorithm

Dijkstra's algorithm maintains a set of vertices whose final shortest-path weights from the source have been predetermined. The algorithm repeatedly selects the vertex with the minimum shortest-path estimate. Due to Dijkstra's algorithm always choosing the "lightest" or "closest" vertex, it is often referred to as a 'greedy strategy'. Greedy strategies do not always yield optimal results (in general), however, Cormen *et al.* (2009) proved that Dijkstra's algorithm does indeed compute shortest paths. Kotzé's (2016) work, mentioned in Section 1.1, used Dijkstra's algorithm to do path planning.

2.3.5 Ants algorithm

Leitão *et al.* (2012) consider a bio-inspired method, ant colonies, in reconfigurable systems. The tendency of the ant species to exhibit swarm intelligence behaviour, and only using one (optimal) path, makes this an attractive approach. The stigmergy phenomenon is evident in ant colonies, enabling self-organization and decentralization. Self-organization is defined by Leitão (2008) as:

> *"The ability of an entity-system to adapt dynamically its behaviour to external changing conditions without external intervention."*

Bonabeau *et al.* (1997) are of the opinion that the basic ingredients to achieving a self-organized system are positive feedback, negative feedback and variability. Also, multiple interactions between individuals is key.

ACO is used for determining optimal paths to a goal. Agents, in the form of "ants", travel over a weighted graph randomly, leaving pheromones behind wherever they move. After an initial wandering phase, the "ants" make their routing decisions based on pheromone levels. Over time, the pheromone trails on less used paths dissipate, leaving the most used paths to prevail (Leitão *et al.*, 2012). The pheromone concentration is calculated as the reciprocal of the time it takes an entity to travel between machines.

Although Saidi-Mehrabad *et al.* (2015) claim that the ACO algorithm outperforms the GA, bio-inspired solutions do have drawbacks. Just like the PF approach, the ACO algorithm is limited to considering travelling entities with a local perspective, leading to myopia. As a result, a lack of future predictions exists.

A publication by Sallez *et al.* (2009) considered applying the ACO strategy to a functional architecture that was split into two levels: A physical level (PL) and virtual level (VL). At the PL, physically active products' routing decisions were made deterministically based upon the optimized results of the VL (which employed the ACO algorithm). This approach offered a good level of responsiveness and adaptability in reaction to environmental changes.

The ACO algorithm offers a decentralized approach to path planning and is therefore less compatible with the ADACOR architecture described in Section 2.2.4.1.

2.4 Simulation

With the dawn of Industry 4.0, increased interest is shown toward simulating what could happen in the future (Rüßmann *et al.*, 2015). Simulation significantly lowers cost by preventing the frequent, physical alterations that are associated with diagnosing manufacturing problems. With simulation packages it is possible to fast forward to a future point in time, from where the operating quality of a system could be evaluated, significantly helping with decision making surrounding reliability and, predominantly, diagnostic issues.

According to Kelton *et al.* (2010), 'Computer simulation' is *the imitation of the operation of a system and its internal processes, over time, and in appropriate detail to draw conclusions about the system's behaviour*. The objective for employing these simulation tools is speed. 'Speed' used in this instance translates to converting as much simulation time as possible to as little real-world time as possible. Simulation is therefore frequently used in the design, emulation and operation of

systems by having the ability to predict the effect of changes to existing systems, but also predicting the performance of new systems.

Generally, the nature of the states describing a simulated system can be either discrete or continuous, or both. Discrete-event simulation is characterized by having the simulation skip time to the next event as soon as the current event has been processed. Most industrial simulation tools only support this mode of simulation (Valckenaers and Van Brussel, 2016). Continuous states change continuously over time (e.g. temperature in an oven) by having defined differential equations that specify the rate of change. Simulation software uses numerical integration to generate a solution for these differential equations over time (Kelton *et al.*, 2010).

Another aspect to simulation is to decide whether to simulate in a stochastic or deterministic environment. Stochastic environments are most common; introducing randomness to represent the variation found in most systems (e.g. failure rates). Deterministic simulation has no variation and is therefore rare in design applications. Deterministic simulation is more common in model-based decision support such as scheduling and emulation applications (Kelton *et al.*, 2010).

Table 1 classifies the different benefits and drawbacks of simulation, according to Kelton *et al.* (2010).

Table 1 Simulation advantages and disadvantages

Advantages	**Disadvantages**
Useful to compute initial values for parameters of physical systems (e.g. time for a pick-and-place operation)	Too much (redundant) detail required, leading to extra effort and simulations being time-consuming
Allows for remedial action to be taken before implementing real-world systems by identifying potential challenges beforehand	Too large a discrepancy between simulated world-of-interest and simulation model (necessitating statistical validation)
Many resources available (e.g. online forums) to enable users to exchange ideas/come into contact with professionals	When used as ‘simulators’, delays might be encountered as a result of computing processes/communication across networks (or the Internet) taking up too much time

The following sections address aspects such as simulation model validation and verification, which are important factors to consider prior to implementing a simulation model. A brief introduction to an existing simulation software, Simio, is also provided.

2.4.1 Simulation model quality

Before implementing a simulation model, users need to be certain that the model and its output are accurate. These concerns are addressed by model verification and validation, and, often, accreditation and credibility (Sargent, 2013). These four measures are defined as:

- Model verification: Ensuring correct operation of the logic describing the simulation model, as well as correct implementation of this logic within the model.

  E.g. having the length of a conveyor in the model be equal to the actual length of the physical conveyor being simulated.

- Model validation: Ensuring the simulation model, within its domain of applicability, functions within a satisfactory range of accuracy which is consistent with the intended application of the model.

  E.g. ensuring the time it takes a pallet to travel between two nodes in the simulation model correlates well (e.g. < 5% error) with the time it takes a pallet to travel between the corresponding nodes on the physical conveyor.

- Model accreditation: Ensuring the simulation model satisfies specified model accreditation criteria according to the specified process.

- Model credibility: Establishing the required amount of confidence in users to use a model and its output.

When considering model validity, behavioural data is needed on the problem entity to compare the physical system's behaviour with the model's behaviour. This data usually takes the form of system input/output data. If behavioural data is unavailable, high model confidence is difficult to obtain as sufficient operational validation cannot be achieved (Sargent, 2013).

Generally, a model's output variables, which is of interest, should be identified and its required amount of accuracy specified. If the variables of interest are random variables, then properties and functions of the random variables such as means and variances are usually what is of primary interest and are what is used in determining model validity. A model should be developed for a specific purpose (or application) and its validity determined with respect to that purpose. If the purpose of a model is

to answer a variety of questions, the validity of the model needs to be determined with respect to each question. Some validation techniques are illustrated by Table 2.

Table 2 Validation techniques (adapted from Sargent (2013))

Technique:	**Description:**
Animation	Operational behaviour of the model is displayed graphically as simulation time progresses.
Comparison to Other Models	Results of the simulation model being validated are compared to results of other (valid) models.
Historical Data Validation	Using historical data to build the model and evaluate whether the model behaves as the system does.
Historical Methods	Rationalism – Assumes a common knowledge when determining whether the underlying assumptions of a model are true. Logic deductions are used from these assumptions to develop the valid model. Empiricism – Requires every assumption and output to be empirically validated. Positive economics – Only requires that the model be able to predict the future and is not concerned with the model's assumptions or structure.
Multistage Validation	Naylor and Finger (1967) proposed combining the three historical methods of rationalism, empiricism and positive economics into a multistage process of validation. This validation method consists of (1) developing the model's assumptions on theory, observations, and general knowledge, (2) validating the model's assumptions where possible by empirically testing them, and (3) comparing the input-output relationships of the model to the real system.
Operational Graphics	Values of various performance measures are shown graphically as simulation time progresses; i.e., the dynamical

	behaviours of performance indicators are visually displayed during a simulation model run to ensure they are correct.
Parameter Variability – Sensitivity Analysis	This technique consists of changing the values of the input and internal parameters of a model to determine the effect upon the model's output. The same relationships should occur in the model as in the real system. Sensitive parameters should be made sufficiently accurate prior to using the model.
Predictive Validation	The model is used to forecast the system's behaviour. Field tests are then used to determine whether the system's behaviour and the model's forecast are the same.
Traces	The behaviour of specific entities in the model are followed through the model to determine if the model's logic is correct and if the necessary accuracy is obtained.

Cost and time constraints are usually the factors preventing a model to be absolutely valid over the complete domain of its intended applicability. This is illustrated by the graph shown in Figure 4, showing that a high model confidence is only achieved at an exponentially high cost.

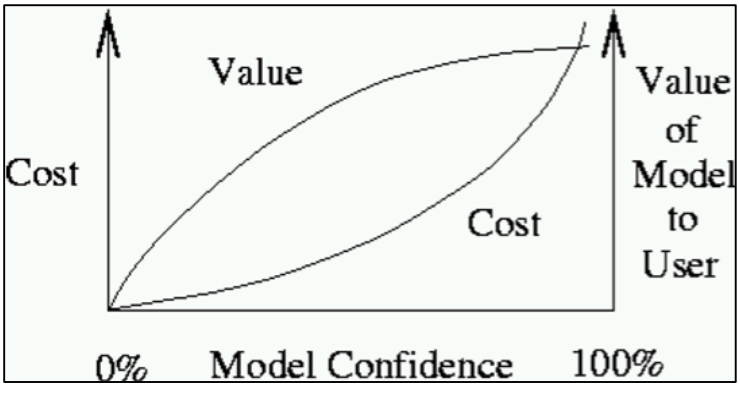

Figure 4 Model confidence (Sargent, 2013)

However, determining that a model has sufficient accuracy for numerous experimental conditions does not guarantee that a model is valid everywhere in its applicable domain. This concern encourages a simulation model only to be developed for a set of well-defined objectives.

2.4.2 Simio simulation software

According to Kelton *et al.* (2010), four discrete modelling paradigms have evolved over time:

- *Events* model the point in time when the system state can change (e.g. a customer arrival).
- *Processes* model a sequence of actions that take place over time (e.g. a processing station receives a pallet, processes it, and releases the pallet).
- *Objects* describe the model from the point of view of the facility.
- *Agent Based Modelling* (ABM) is a special case of objects – the system behaviour emerges from the interaction of a large number of autonomous intelligent objects (e.g. populations).

Simio is a multi-paradigm modelling tool that combines all the above paradigms into a single framework, enabling the user to use a single paradigm, or combine multiple paradigms in the same model.

2.5 Literature overview

The literature study for this thesis is structured to focus on the overarching concepts of manufacturing systems, and ends with the literature most relevant to the topic at hand, i.e. path planning algorithms. This funnel-like approach is applied to each subsection as well – discussing the alternatives before concluding with the most relevant literature.

The chapter starts with a discussion of manufacturing systems, and ultimately introduces the manufacturing systems most suited to the South African manufacturing climate, reconfigurable manufacturing systems. Next, the control architectures associated with these manufacturing systems are described. The basic architectures are first described followed by the holonic and agent approaches, which takes to both centralized and decentralized control. The holonic approach was selected for further discussion as this architecture is associated with great autonomy.

Having set the background to which the route planning strategies developed in this thesis will be applied, a discussion on path optimization algorithms follows. Algorithms, suited to finding the optimal path for a pallet on a conveyor in an RMS

cell, could not be found. Consequently, traditional algorithms, such as Dijkstra's algorithm, were considered. A limitation associated with Dijkstra's algorithm is the inability to account for future (and sometimes unpredictable) events on a conveyor, such as changing traffic and route blockages. The result of applying Dijkstra's algorithm to such circumstances would be pallets taking paths that were previously regarded the shortest, but are in reality slower, less optimal paths. The "ants" approach seems to be a more favourable solution, seeing as its autonomous properties fits well with a holonic control environment. The "ants" approach to path optimization is, however, not considered by this thesis since its application is investigated by a colleague C.S. van den Bergh.

Simulation is discussed next, as a possible approach to forecast route traffic. A possible solution is to have a simulator run simulations for all the possible routes a pallet could follow, at an accelerated speed. The route showing the quickest time is then booked on the physical conveyor.

The first step in adopting a simulation model is to be able to trust the simulation's results. Section 2.4.1 places special emphasis on simulation model validation, as this is the main factor determining model integrity in the thesis – is the simulated travelling times of pallets the same as the actual travelling times? Verification is regarded trivial, as having to ensure physical parameters, such as conveyor length and the velocity at which the conveyor moves, are replicated in the simulation model, is easily achieved.

Ultimately, the reader is briefly introduced to an existing simulation software package, Simio. Besides Simio being used by the MADRG, Simio has an Application Programming Interface (API) which allows for great customizability within simulation models. Simio's ability to have additional logic imported via user defined DLLs, coded in C#, makes it a favourable option for integration with Kotzé's existing C#-coded controller.

3. Overall approach

As mentioned in Section 1.3, where the objectives were discussed, the pallet routing strategy developed in this thesis was aimed at integration with the conveyor controller developed by Kotzé (2016). In this chapter, therefore, Kotzé's work is first described, and thereafter the overall approach adopted for the pallet routing, of which the details are described in Chapters 4 and 5.

The table provided in Table 3 summarises the terminology used in describing the overall approach and subsequent chapters.

Table 3 Terminology

Term	**Description**
Complex conveyor	Conveyor layout shown in Figure 20
Laboratory conveyor	Conveyor layout shown in Figure 11
Pallet	Pallet travelling on the physical conveyor
Physical conveyor	Conveyor shown in Figure 1, on which physical experiments are conducted
Physical conveyor module	Physical PLC with its associated inputs and outputs, e.g. a lifting unit
Physical HLC	Conveyor control application, as implemented by Kotzé, interfaced with the physical conveyor modules
Physical PLC	PLC (LS XEC-DR20SU) hardware used to operate conveyor
PLCStatus DLL	DLLs executing virtual conveyor module logic through Simio
Virtual conveyor	Conveyor hardware replaced by a simulation in Simio that represents the real-time behaviour of a physical conveyor
Virtual conveyor module	Physical conveyor module, e.g. a diverter unit, replaced by an entity in Simio which simulates the real-time behaviour of a that module
Virtual HLC	Conveyor control application, as implemented by Kotzé, interfaced with the virtual conveyor modules
Virtual pallet	Pallet travelling on the virtual conveyor
"Virtual holonic controller approach"	Initial route planning strategy, depicted by Figure 9
"Simio controller approach"	Second route planning strategy, depicted by Figure 13

3.1 Previous work at Stellenbosch University

The work proposed in this thesis can be regarded as a continuation of the research done by colleague Marcus Kotzé in fulfilment of his MEng degree. The work conducted by Kotzé (2016) focused on offering an alternative approach to what a former colleague, Anro le Roux, had considered. Le Roux's (2013) focus was on developing a holonic conveyor controller based on a centralized/hierarchical holonic architecture, whilst Kotzé focused on a more distributed/hybrid holonic architecture to control a conveyor. The main difference was the hardware design in which Kotzé allocated a PLC (LS XEC-DR20SU) module to each functional module, whilst Le Roux made use of only one PLC (Siemens S7) to control all functional modules. Kotzé's work consisted of configuring the low-level control (LLC), i.e. the different conveyor modules, and high-level control (HLC), responsible for controlling the conveyor. Figure 5, in accordance with the legend provided in Table 4, describes the setup Kotzé used for validating his controller.

Table 4 Annotations used in Figure 5 (adapted from Kotzé (2016))

Symbol	Description
Blue Square	Lifting Unit
Green Triangle	Stop Gate
Red Rectangle	Proximity Switch
Yellow Rectangle	Rocker Proximity Switch

The *Lifting Unit* is pneumatically operated and, in its low setting, lets a pallet pass unhindered, or, in its high setting, lifts a pallet slightly off the conveyor and holds it steady for a certain amount of time – regarded as the 'processing time'. The *Stop Gate* is also pneumatically actuated to stop a pallet or let a pallet pass through. Stop gates are used to control the flow of pallets and to stop pallets in the correct position above lifting units. The *Proximity Switch* generates a signal when a pallet passes over it, while the *Rocker Proximity Switch* has a rocking barrier which trips a proximity switch when a pallet pushes against the barrier. The transverse conveyor is pneumatically actuated and is used to move a pallet onto or off of the conveyor. The transverse conveyor has three possible settings: allowing pallets to pass unhindered; stopping a pallet precisely above the transverse conveyor, in position ready to be

transferred off the conveyor path; and lifting the pallet clear off the conveyor and transferring the pallet onto or off of the conveyor.

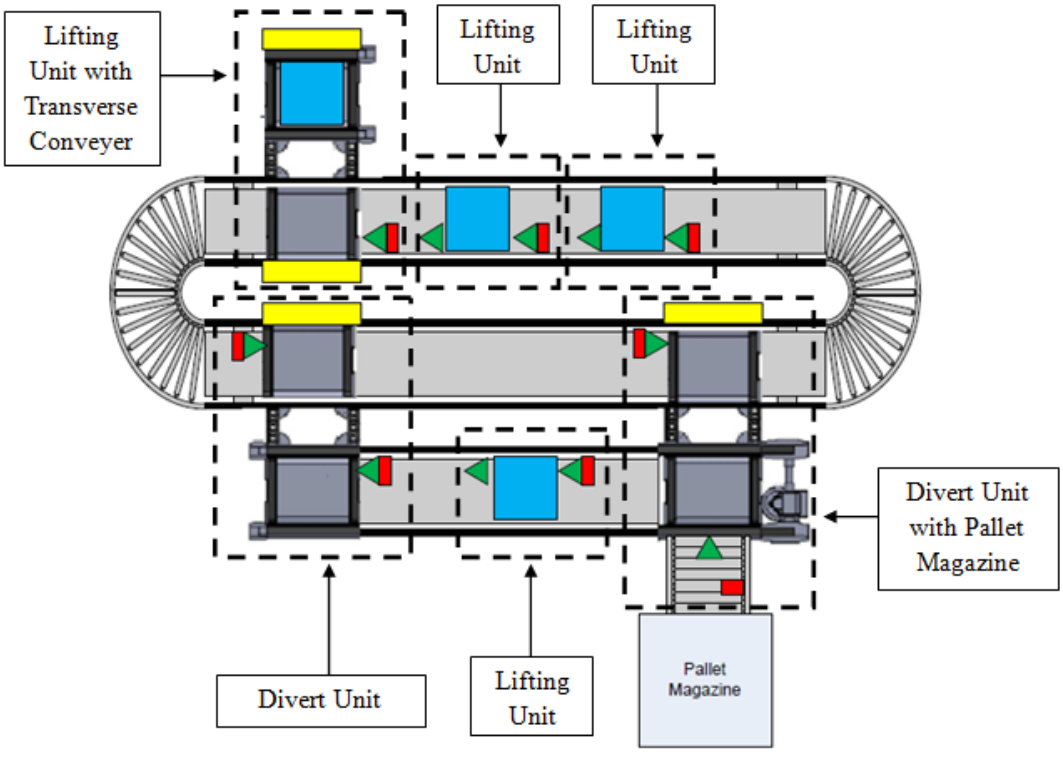

Figure 5 Conveyor modules (adapted from Kotzé (2016))

A reconfiguration experiment was conducted by Kotzé (2016) with the resultant setup as shown by Figure 6. As can be noted, the reconfiguration required modifying the hardware of the *Divert Unit* and *Divert Unit with Pallet Magazine* modules to handle the changed conveyor direction and change in proximity sensors. The PLC programs, as well as the operational holons in the C# program for the HLC, for both the *Divert Unit* and *Divert Unit with Pallet Magazine* conveyor modules were modified, due the physical structure of both modules having been changed. The author was obliged to neglect the *LU2* conveyor module, indicated by the solid outline box in Figure 6, for the research conducted in this thesis. This was due to the *LU2* conveyor module having been replaced by a microcontroller and used as another research project, during the period the research in this thesis was conducted. The physical conveyor module (*LU2*) was therefore removed from the conveyor and its associated software component from the physical HLC. The resultant conveyor schematic that this thesis considers, is shown by Figure 11.

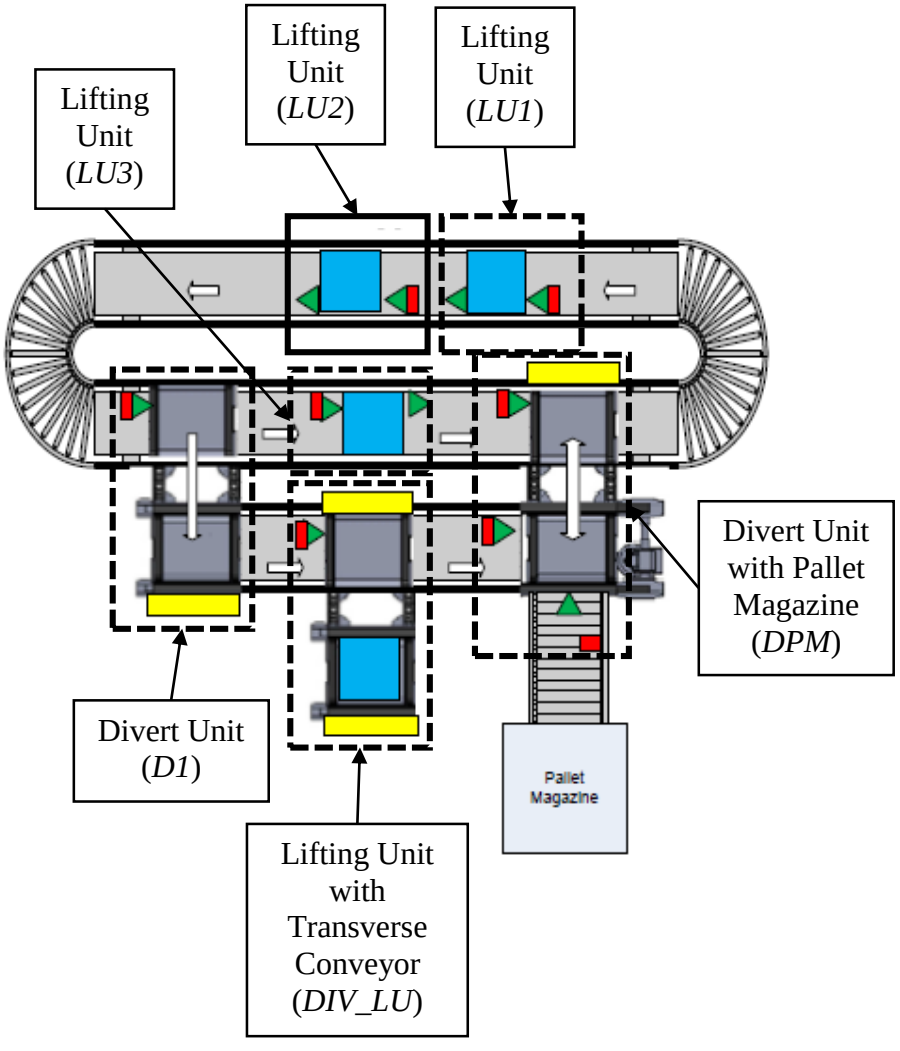

Figure 6 Conveyor setup after reconfiguration (adapted from Kotzé (2016))

The PLCs were programmed using the structured text language (STL) and interfaced with the HLC entity using TCP/IP messaging. The HLC, also known as the conveyor controller, was responsible for coordinating the actions of the different PLCs. Message payloads between HLC and LLC were in binary format. The HLC used a holonic architecture, based on ADACOR, with holons and connections between holons set up as shown in Figure 7. The different holons are set up according to the functional descriptions provided by Section 2.2.4.1. The HLC was programmed in C# using Microsoft Visual Studio, particularly due to the various inbuilt functions of C# and the user friendly GUI Microsoft Visual Studio provides. C# is object oriented, having encapsulation and inheritance properties, which compliments a holonic structure, such as ADACOR, well.

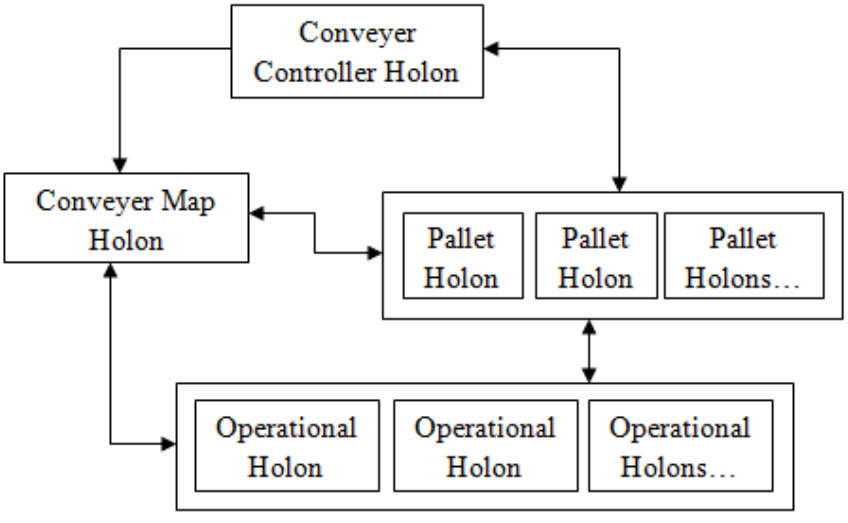

Figure 7 Holon communication and hierarchy (adapted from Kotzé (2016))

Herewith the functions of the different holons, shown by Figure 7:

- The *Conveyor Controller Holon* is responsible for starting up the entire holarchy and creating/terminating holons.
- The *Conveyor Map Holon* is responsible for calculating a route, based on the Dijkstra algorithm (in Kotzé's implementation), for a pallet, as well as keeping track of each pallet's movement along the conveyor.
- An *Operational Holon* is associated with every resource, mostly being PLC modules, of the conveyor and functions as the interface between HLC and LLC. When associated with PLC modules, each operational holon also keeps track of the pallets entering/exiting the modules' proximity.
- A *Pallet Holon* is associated with every pallet that is on the conveyor, i.e. every task that is created via the HMI shown in Figure 8, and is regarded as a combination between an operational holon and a task holon.

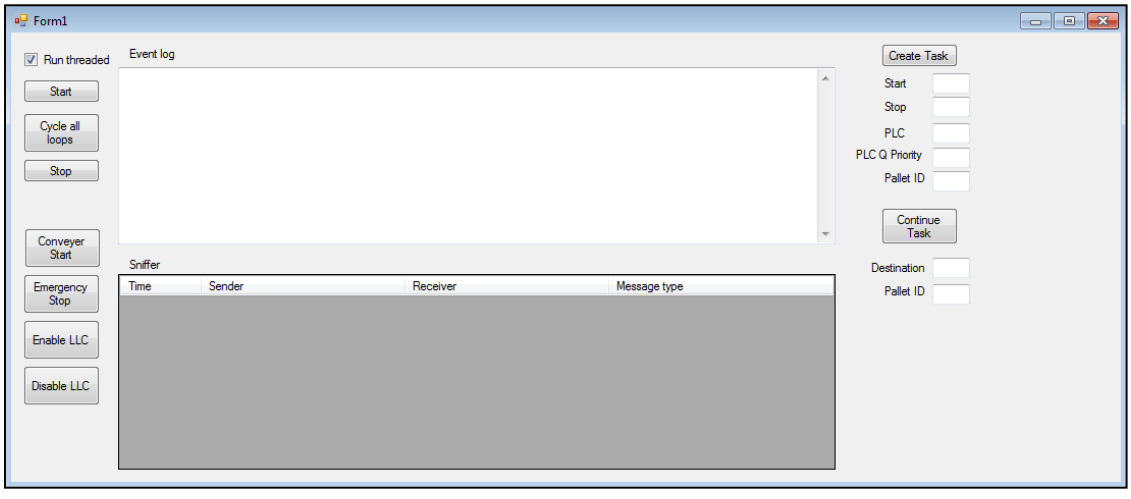

Figure 8 HLC human machine interface (adapted from Kotzé (2016))

3.2 Current work

The two route planning strategies developed in this thesis, the "Virtual holonic controller approach" and the "Simio controller approach", had to be integrated with the *Conveyor Map Holon* (see above) of Kotzé's HLC, since this holon performed the route planning function for pallets in Kotzé's case.

3.2.1 "Virtual holonic controller approach"

This approach entails having a physical instance and a virtual instance of Kotzé's controller running at the same time (refer to Figure 9). The physical instance is the HLC-LLC setup developed by Kotzé, as-is, whilst the virtual instance contains the same HLC Kotzé used, but an adapted LLC (as shown by Figure 9). The adapted LLC saw the development of entities *Interpreter*, *DLL* and *Simio conveyor* (virtual conveyor). The LLC, for the virtual case, was simulated, virtual conveyor modules, each housed by a DLL (named *PLCStatus*), and 'actuated' by simulation software Simio. The virtual HLC remained oblivious to the fact that it was communicating with the virtual conveyor modules, contained by Simio, instead of the physical conveyor modules (as LLC). A C# application, called *Interpreter*, was written to enable the communication between the DLLs and virtual HLC.

The "Virtual holonic controller approach" entailed having the physical HLC call the virtual HLC whenever an optimal route needed to be determined for a pallet that is about to enter the conveyor. The virtual HLC will be triggered with the state of the physical conveyor (utilizing the physical HLC's *Conveyor Map Holon*), after which this state is recreated on the virtual conveyor with virtual pallets. A simulation is

then run for each different route a virtual pallet is able to follow. Should the number of routes be very large (and require significant computational effort), a sample of the routes most likely to be selected by the routing strategy, could be drawn by employing Dijkstra's algorithm beforehand. The optimal route would eventually be selected from this sample.

The virtual conveyor is reset to its initial state every time a new simulation is run. After all the possible routes have been simulated, the virtual HLC identifies the optimal route and sends the appropriate routing sequence back to the physical HLC, resulting in a physical pallet being launched on this route. A brief explanation is provided to highlight some of the functions of the entities shown in Figure 9, in the following subsections.

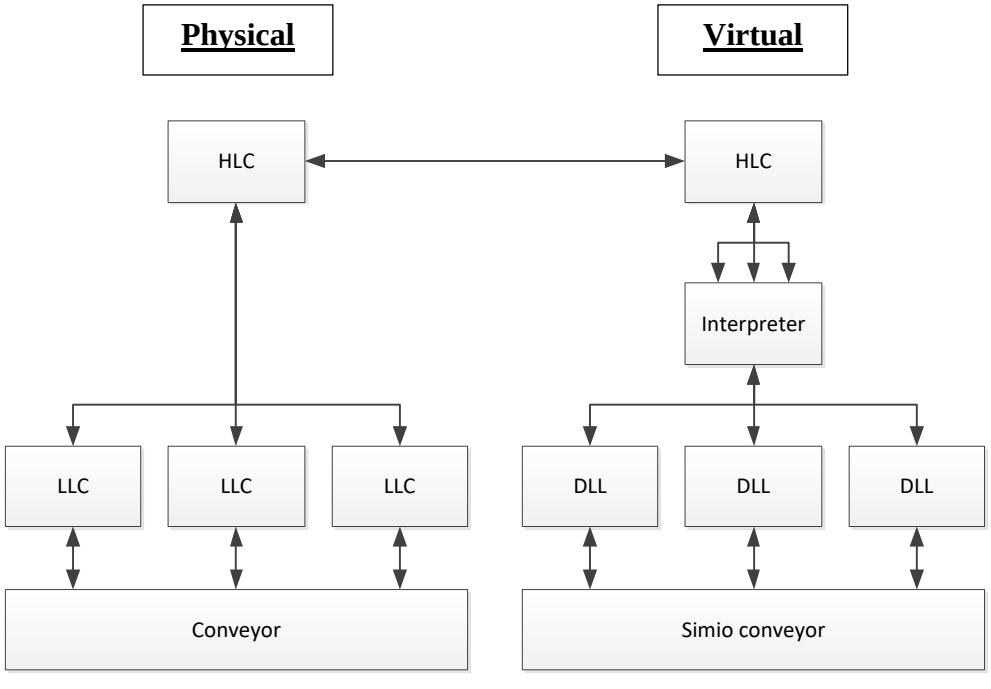

Figure 9 Optimization strategy: "Virtual holonic controller approach"

3.2.1.1 HLC

HLC is the high-level control application developed by Kotzé (2016), described in the first section of this chapter. The architecture for this controller is an adaptation to the ADACOR holonic architecture, as has been mentioned in Section 3.1, with a supervisor holon which serves as a global regulator of the dispersed operational holons (i.e. different conveyor modules). This supervisory role forms part of the

global path planning of the different routes pallets will be able to travel on the conveyor track.

3.2.1.2 Interpreter

The *Interpreter* application has three purposes:

- Maintain TCP/IP communication channels between the virtual HLC and the *PLCStatus* DLLs.
- Convert the byte messages received from the virtual HLC to actuation instructions of the virtual conveyor modules, via the DLLs.
- Send byte status messages of the different virtual conveyor modules, received via the DLLs, to the virtual HLC application.

3.2.1.3 Simio

Simio is the simulation software used to simulate the physical conveyor. A Simio representation of the MADRG's laboratory conveyor is shown in Figure 10, corresponding to the physical conveyor shown in Figure 6. Simio executes for the most part the logic determined by the *Interpreter* application and is allocated little intelligence (this however changes in Section 6.3 and Section 6.4). This approach promotes modularity.

Simio has *User-defined process steps* which can be used to link the virtual conveyor modules to the *Interpreter* application. These steps are C# encoded DLL (dynamic link library) files, and are triggered when one of the virtual conveyor nodes, controlled by a specific virtual conveyor module, is entered by a virtual pallet (called an *Entity* in Simio). The triggered step then awaits a message back from the virtual HLC application via the *Interpreter* application, before continuing with simulation model execution.

3.2.1.4 PLC states

It should be noted that all *Lifting Unit* modules, including the *Lifting Unit with Transverse Conveyor* module, acts as destination modules, i.e. they are the only modules able to hold pallets while manufacturing operations are executed on the pallets. The conveyor setup depicted by Figure 11 is that of a case study conveyor considered for this thesis, and is the second conveyor setup Kotzé used in his work (Figure 6). "PLC1" and "PLC5" are lifting units 1 and 3 respectively. "PLC3" and "PLC4" are the *Lifting Unit with Transverse Conveyor* and *Divert Unit* conveyor modules respectively. "PLC6" represents the *Divert Unit with Pallet Magazine*

module ("PLC7" is excluded for the purposes of this thesis and is replaced by a manually operated *Pallet Magazine* module).

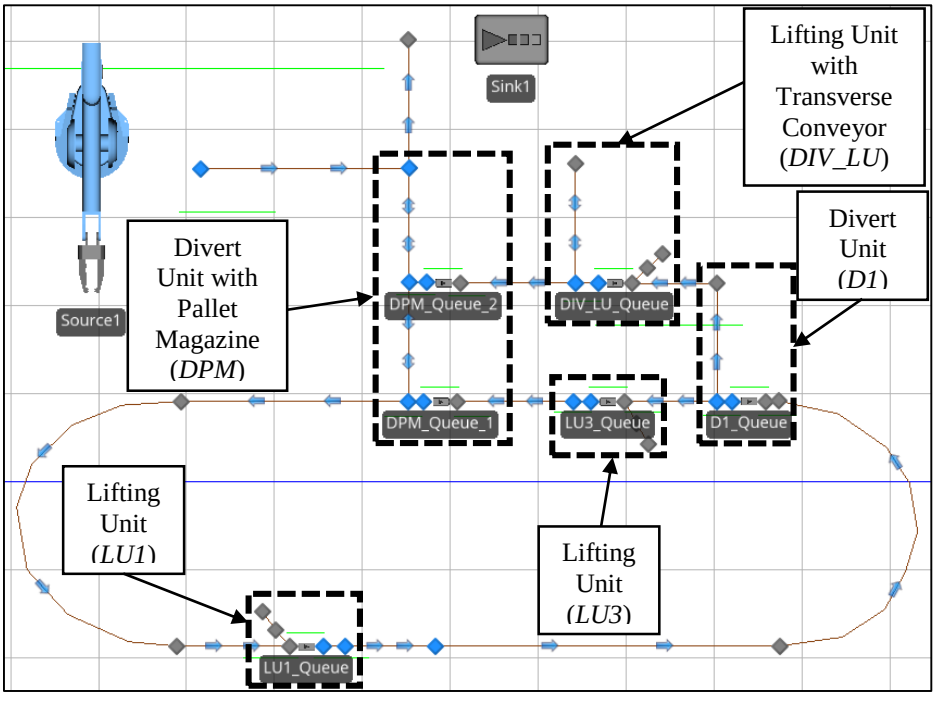

Figure 10 Laboratory conveyor (Simio representation)

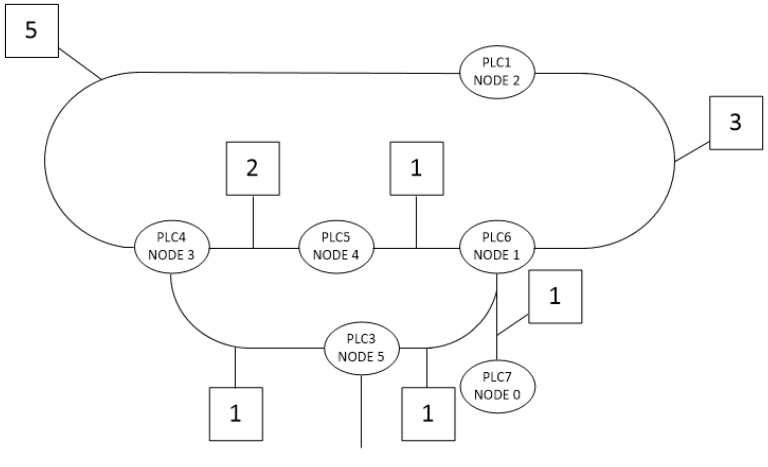

Figure 11 Laboratory conveyor setup

The state diagrams of the different conveyor modules, both in the physical and virtual setups, are illustrated by Figure 12 and Appendix A. Figure 12 depicts the state diagram of the first *Lifting Unit* (*LU1*).

- The square shaped boxes represent the status messages the respective PLCs are sending to the HLC.
- The diamond shaped boxes indicate where more than one sequence of events is possible.
- The rhombus shaped boxes indicate the commands sent by HLC to LLC.

Furthermore, the messaging sequence tables for the following tasks are given by Appendix B (being the same for both the physical and virtual setup):

- Moving a pallet from the *Divert Unit with Pallet Magazine* module (*DPM*) to the first *Lifting Unit* module (*LU1*)
- Moving a pallet from the *Divert Unit with Pallet Magazine* module (*DPM*) to the *Lifting Unit with Transverse Conveyor* module (*DIV_LU*)
- Moving a pallet from the *Divert Unit with Pallet Magazine* module (*DPM*) to the third *Lifting Unit* module (*LU3*)

Also, the return sequences, back to the *Divert Unit with Pallet Magazine* module, are included:

- Moving a pallet from the first *Lifting Unit* module (*LU1*) to the *Divert Unit with Pallet Magazine* module (*DPM*)
- Moving a pallet from the *Lifting Unit with Transverse Conveyor* module (*DIV_LU*) to the *Divert Unit with Pallet Magazine* module (*DPM*)
- Moving a pallet from the third *Lifting Unit* module (*LU3*) to the *Divert Unit with Pallet Magazine* module (*DPM*)

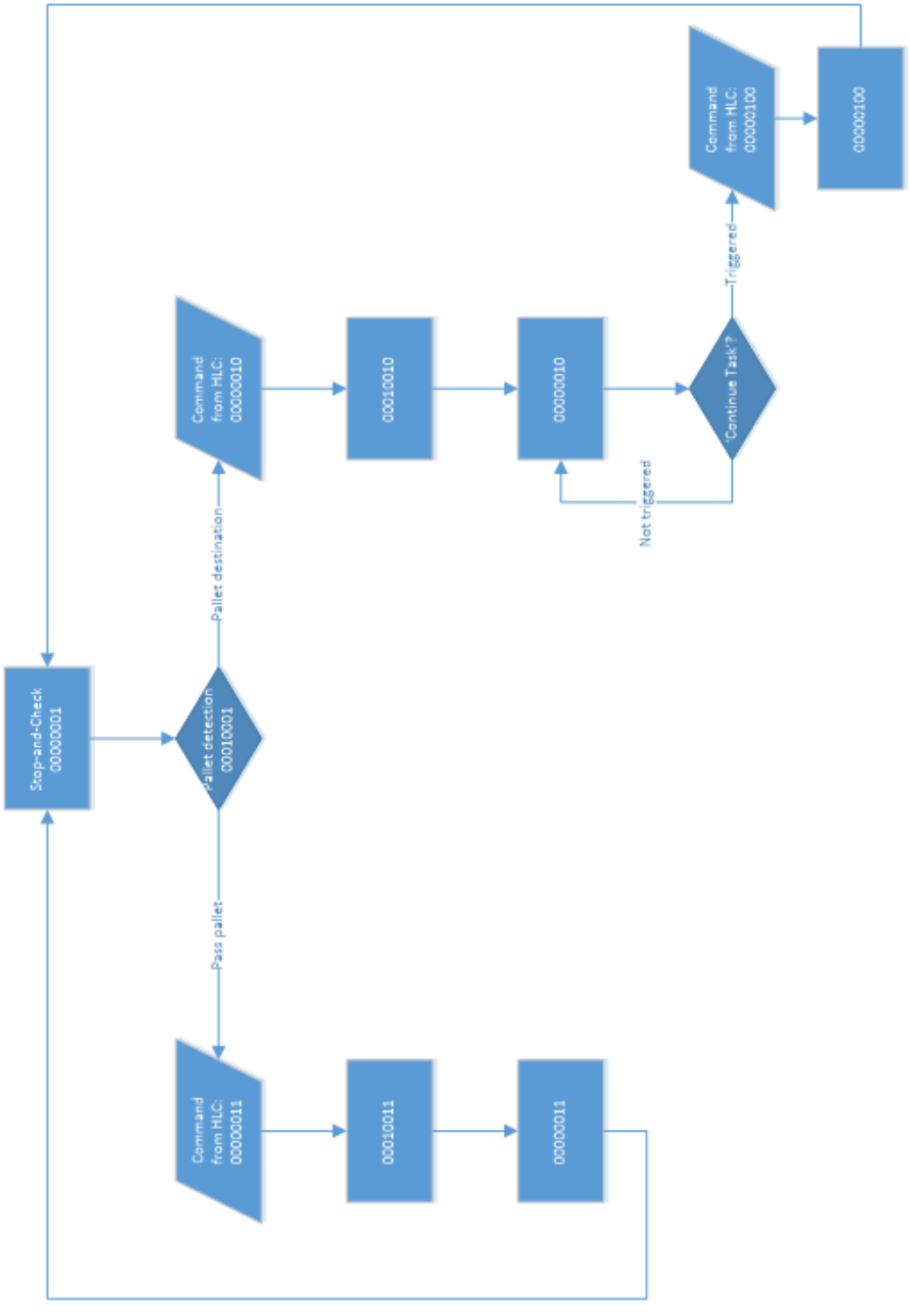

Figure 12 Lifting unit module (LU)

3.2.2 "Simio controller approach"

This approach substituted the virtual HLC, *Interpreter* and DLLs of the "Virtual holonic controller approach", shown in Figure 9, with only the Simio simulation software (see Figure 13). This meant that all routing logic had to be created within Simio, requiring that someone familiar with Simio implements and tests the control logic, to make any changes in the future. This approach was developed to address the latency issue that surfaced by employing the "Virtual holonic controller approach" (Figure 9). This latency concern prohibited the "Virtual holonic controller approach" from handling more than three virtual pallets, at the same time, on a virtual conveyor.

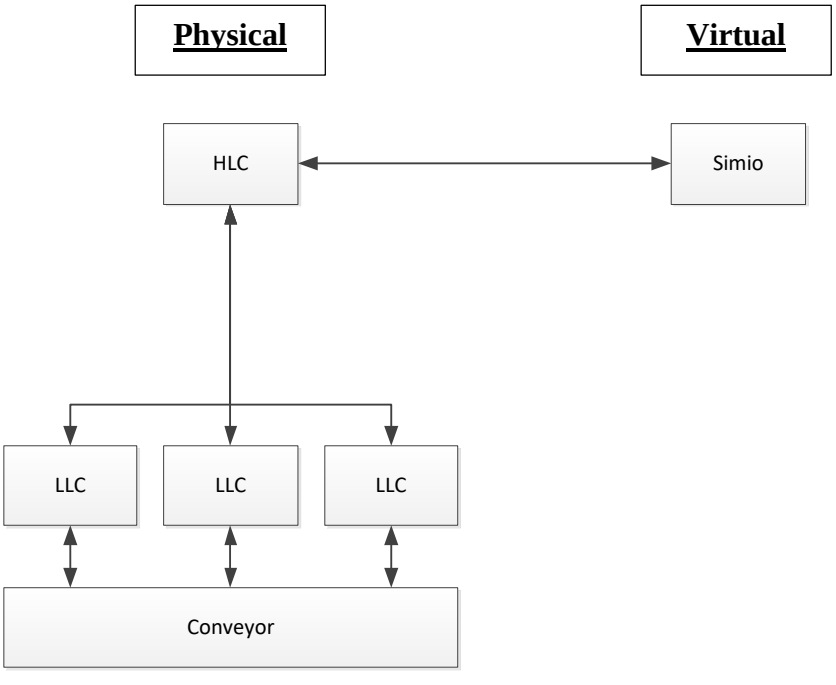

Figure 13 Optimization strategy: "Simio controller approach"

4. Simio integration

This chapter describes the aspects of the pallet routing strategies that are related to Simio. The reader will be introduced to Simio concepts that are used throughout the text. Only the essential Simio functions are discussed, as an elaborate discussion describing all of Simio's functionality is beyond the scope of a thesis (the Simio user manual can be found at https://dl.dropboxusercontent.com/u/106074191/Documentation.chm). Also, the test configurations, that relied on the "Simio controller approach", will be described.

4.1 Simio application

The Simio simulation platform (Version: 7.114.11720 (32 bit); License Type: University Enterprise) was the simulation software used for addressing the requirements stipulated in Chapter 1, due to the following reasons:

- Simio is C# (.NET) based, which is the object oriented language the MADRG uses (using Microsoft Visual Studio Premium Edition 2012, .NET Framework 4.5)
- The Simio API extension is convenient for adding logic to the Simio application
- Since Simio is also being used by colleagues K. Kruger (PhD candidate) and C.S. van den Bergh (fellow MEng student), it was the common simulation language used by the group and resources were easy to draw upon

The Simio application can be used as both a statistical tool for extracting results or a simulator as was mentioned in Chapter 1. Simio was predominantly used as a simulator throughout this thesis.

4.1.1 Facility view

The *Facility* view is where models are created. Models are constructed by dragging elements from the *Standard Library* to the window, and then specifying the operation logic in the element's properties, in the lower right hand corner of the screen. The *Add-On Process Triggers* is where *process logic* is referenced by the *Facility* view. Figure 14 depicts the *Facility view* of the laboratory setup (which is considered as one of the case studies by this thesis).

4.1.2 Processes view

The *Processes* view contains additional logic that can be added to elements contained by the *Facility* view. A *process* is created to which *process steps* can be added. The properties of a *process step* can be altered in the lower right hand corner of the screen. The created *process* only needs to be referenced in the *Facility* view, as was mentioned in Section 4.1.1, for the logic to be executed by an element. This logic will be triggered whenever an *entity* (e.g. a virtual pallet) arrives at the element. Figure 15 depicts the *Processes* view of the laboratory setup, indicating the sequence of *process step* execution as being from left to right.

4.2 Application programming interface

Simio has recently lend itself toward allowing more and more control to the user via the introduction of its API (application programming interface). To be able to reap the full benefits of using the Simio API, the Simio Visual Studio C# template just need to be installed, in order to use the *SimioAPI* library, which automatically includes the following Microsoft Visual Studio reference assemblies:

- SimioAPI
- SimioAPI.Extensions
- SimioAPI.Graphics
- SimioDLL
- SimioEnums

Mention should be made that the *PLCStatus* DLL (mentioned in Section 3.2.1) included a service reference (*ServiceReference*) which referenced the *NotifyService* service that was hosted by the *Interpreter* application (mentioned in Section 3.2.1) – this will be discussed in Section 5.5.3.

After the Simio C# template has been installed, a new project is created in order to build the *PLCStatus* DLL which could be applied to the *processes* of the *Processes* view in Simio. A project of type *User Defined Step And Element*, under the *Simio User Extensions* directory, needs to be used. Once a project is created, two classes are automatically generated – *UserElement* and *UserStep*.

These classes contain pre-set properties with comments making it easier for users to manipulate properties to their needs. Only the *UserElement* class was manipulated because these properties pertained to the *User-defined Process Step* in the *Processes* view. It was, however, compulsory to have a *User-defined Element* in the *Definitions* view for the *process step* to be used. This *element* pertained to the *UserElement* class, and was left unchanged in creating the DLL.

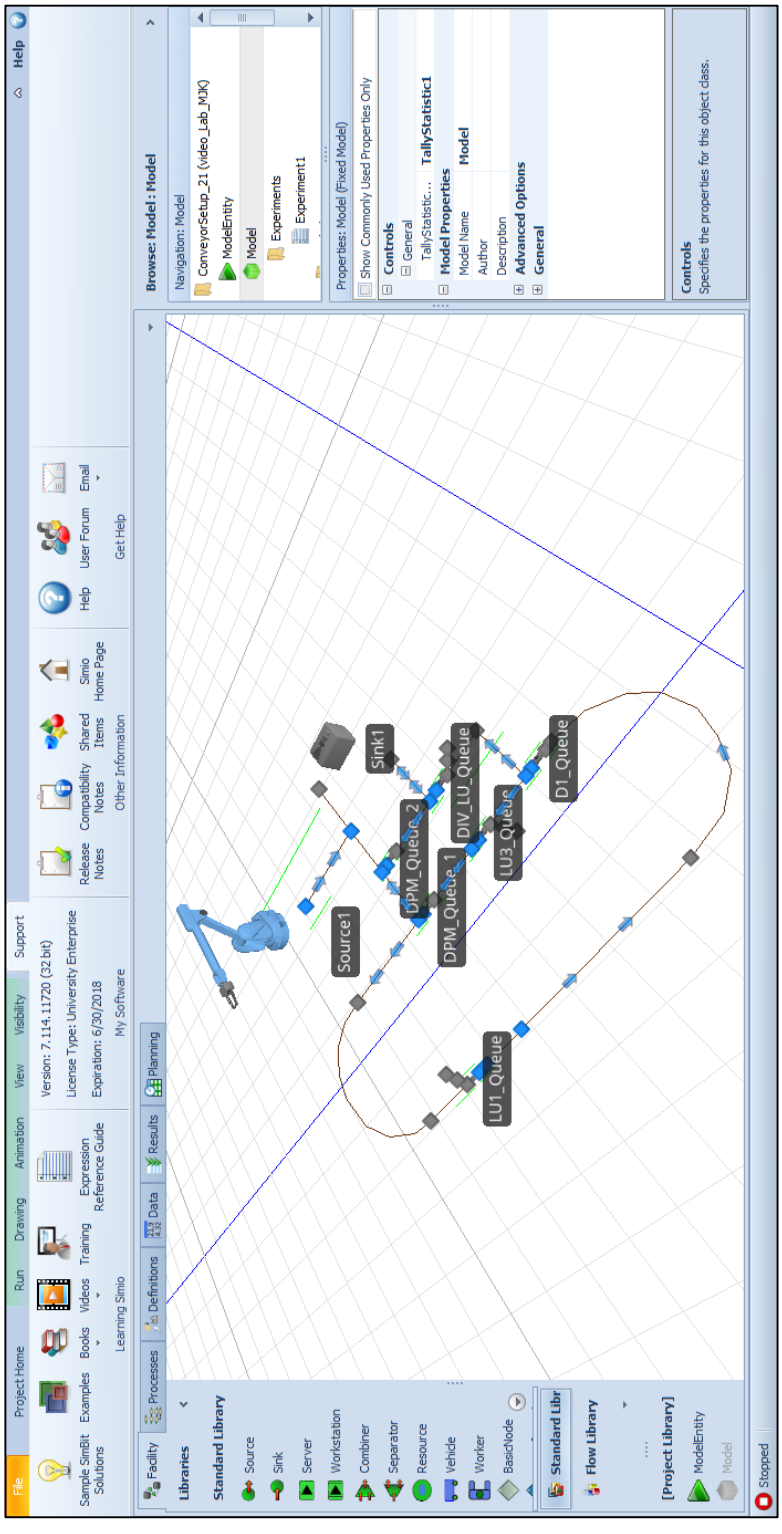

Figure 14 Laboratory setup: Facility view

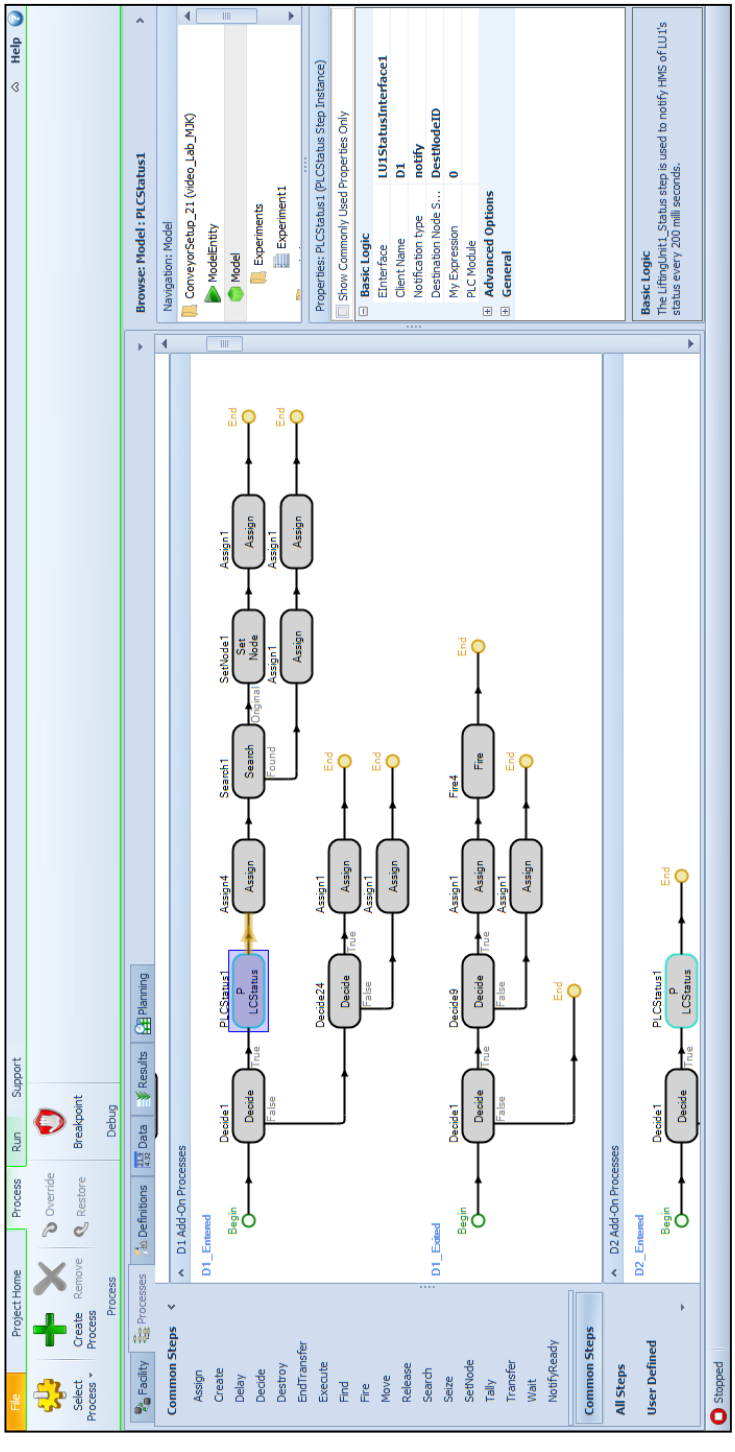

Figure 15 Laboratory setup: Processes view

Properties in the *UserStep* class that were changed, include:

- Name and description, which the user can alter to a string he/she prefers to be displayed in the *Processes* view when hovering over the *PLCStatus step* (see Figure 16).
- *Process step* properties, which are changed within the *DefineSchema* method. These properties are shown to the left of the colons in Figure 17.

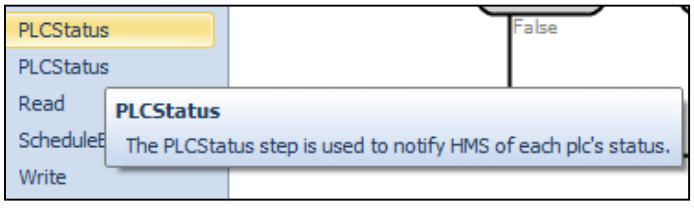

Figure 16 PLCStatus step description

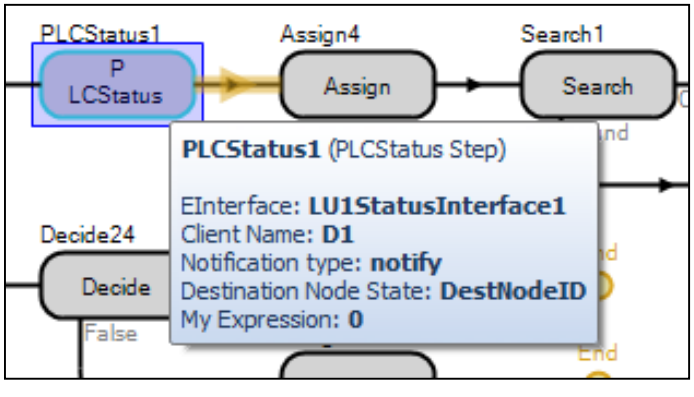

Figure 17 PLCStatus step properties (Simio representation)

The parameters to the right of the colons in Figure 17 are specified by the user when inserting a *PLCStatus step* into the process logic of a node. More specifically, the 'Client Name' parameter ('D1' in this case) and 'Destination Node State' ('DestNodeID' in this case) are the important properties. Communication between the *PLCStatus* DLL and the *Interpreter* application is instantiated when the *process step*, *PLCStatus*, is encountered during a simulation run, leading the DLL to subscribe to the *NotifyService* of the *Interpreter* application with its 'Client Name' parameter. The *Interpreter* will respond with a string (*dest_node*) that is stored as the 'Destination Node State' parameter. After having received the *dest_node* string, the

PLCStatus process step is exited and the *process token* (in Simio) continues to the next *step*.

4.3 Test configurations

This section describes the Simio representation of the case studies that were considered for testing in this thesis: A laboratory conveyor and a complex conveyor. "Simio representation" encapsulates constructing of the virtual conveyor in the *Facility* view, as well as the associated *process* logic determined by the *Processes* view, as mentioned in Section 4.1.2. The *process* logic depicting the "Simio controller approach" (described in Section 3.2.2) is also described for each case study, as these *processes* differ from the *processes* required by the "Virtual holonic controller approach" (described in Section 3.2.1); the main difference being the exclusion of the *PLCStatus process* step.

4.3.1 Laboratory conveyor

4.3.1.1 Facility view

The laboratory conveyor is that which was shown in Figure 6. A physical model of this is available in the laboratory of the MADRG. The reconstruction of this conveyor in Simio is shown in Figure 10, but is repeated in this section for ease of reference (Figure 18). *Server* elements (e.g. *LU1_Queue*, *D1_Queue*, etc.) are used for housing queues of 'pallets' at the different virtual conveyor modules. The *Source* element is responsible for launching virtual pallets onto the conveyor, whilst the *Sink* element removes virtual pallets from the conveyor.

4.3.1.2 Process view

Several process were created, of which four were of importance:

1. The *process* being executed whenever a simulation run is about to be started: This is an inherited *process*, a *process* that is created by default. The *Read*, *Create* and *Assign process steps* were inserted into this process. The *Read step* reads the physical conveyor's state from a file called *ConMap state*. The *Create step* creates the corresponding number of virtual pallets. The *Assign step* places the created number of virtual pallets at their respective destinations in order to recreate the physical conveyor's state in the *Facility view*.
2. The *process* associated with the *Source* element: This process is responsible for recording the time a virtual pallet is launched onto the virtual conveyor. An *Assign process step* was used for accomplishing this.

3. The *processes* associated with all of the virtual conveyor modules (i.e. nodes) shown in Figure 18: These *processes* made use of the *PLCStatus*, *Search* and *Assign process steps*. The *PLCStatus step* informs the *Interpreter* of a virtual pallet occupying a node, and awaits an integer value indicating to which node the virtual pallet needs to be routed next. The integer value is then used as index for the *Search step* to look up the corresponding destination node in a *List element*. After finding the node, the *Assign step* uses the node for directing the virtual pallet to its next destination. An example of one of these *processes* is shown in Figure 19.
4. The *process* associated with the *Sink* element. An *Assign process step* was used to record the time a launched virtual pallet exited the virtual conveyor.

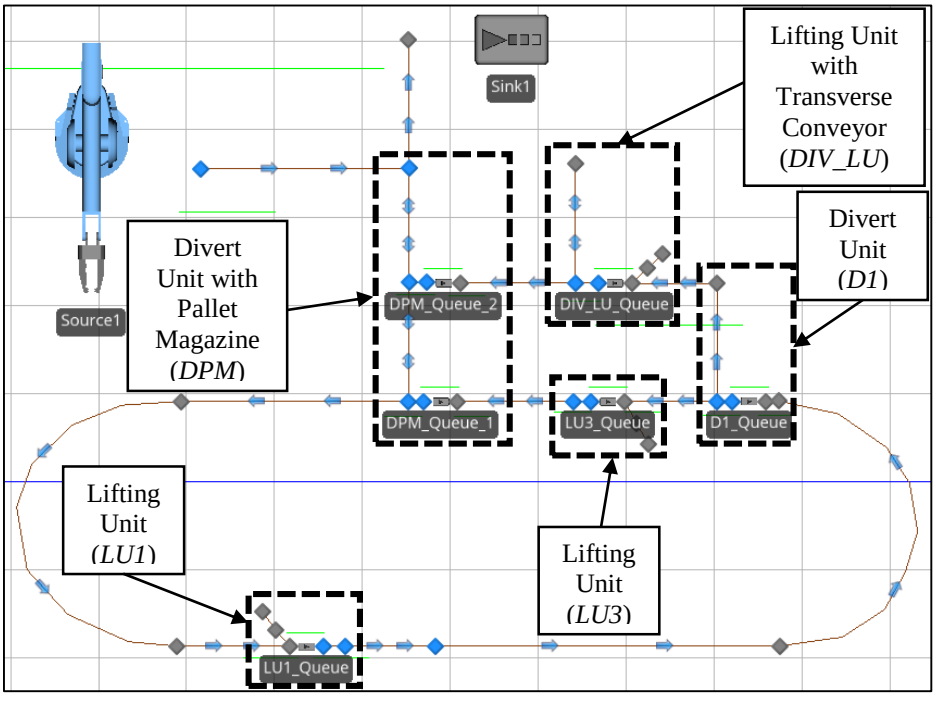

Figure 18 Virtual conveyor: Laboratory setup

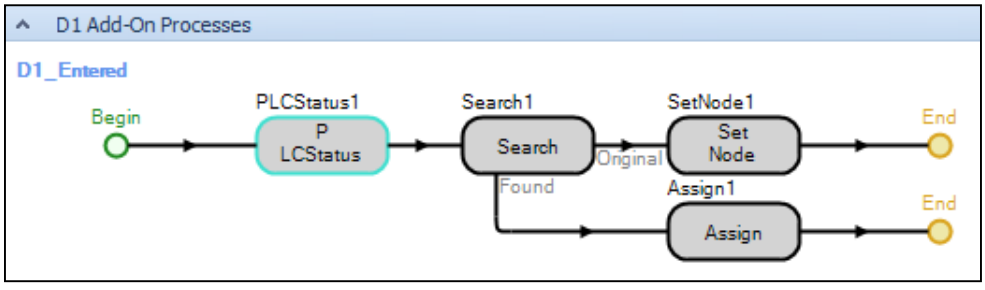

Figure 19 Diverter (D1) module's process

4.3.2 Complex conveyor

The complex conveyor setup was created to address the limitation of the laboratory conveyor. This limitation was the inability to sufficiently demonstrate the optimization capability of the strategy outlined by Section 3.2.1.

This conveyor had to be set up virtually. This implies that the physical complex conveyor was set up in Simio, and ran on a different computer as those running the physical HLC and virtual HLC, the virtual laboratory conveyor and its accompanying *Interpreter* application. Another *Interpreter* application was required to run on the same computer as this physical complex conveyor, to be able to run this conveyor with the physical HLC.

4.3.2.1 Facility view

The complex conveyor, which was used for conducting tests two and four, discussed in Sections 6.2 and 6.4, is shown by Figure 20. The elements used in this setup were the same as those used for the laboratory conveyor, mentioned in Section 4.3.1.1: *Source*, *Server* and *Sink*. The difference, apart from the conveyor structure, was the way in which the *Divert Unit with Pallet Magazine* module and some of the *Lifting Unit with Transverse Conveyor* and *Divert Unit* conveyor modules were set up. Modules *DPM*, *DIV_LU_2*, *DIV_LU_3*, *DIV_LU_6*, *DIV_LU_7*, *DIV_LU_8*, *DIV_LU_9* and *D2* were different to the *DPM*, *DIV_LU* and *D1* modules used by the laboratory conveyor (Figure 18). The *Divert Unit with Pallet Magazine* and *Lifting Unit with Transverse Conveyor* modules had two distinctly different routes entering and exiting the *Pallet Magazine* and *Lifting Unit* respectively, instead of having the same route enter/exit them. The *D2 Divert Unit* module was used as a joining junction in this setup.

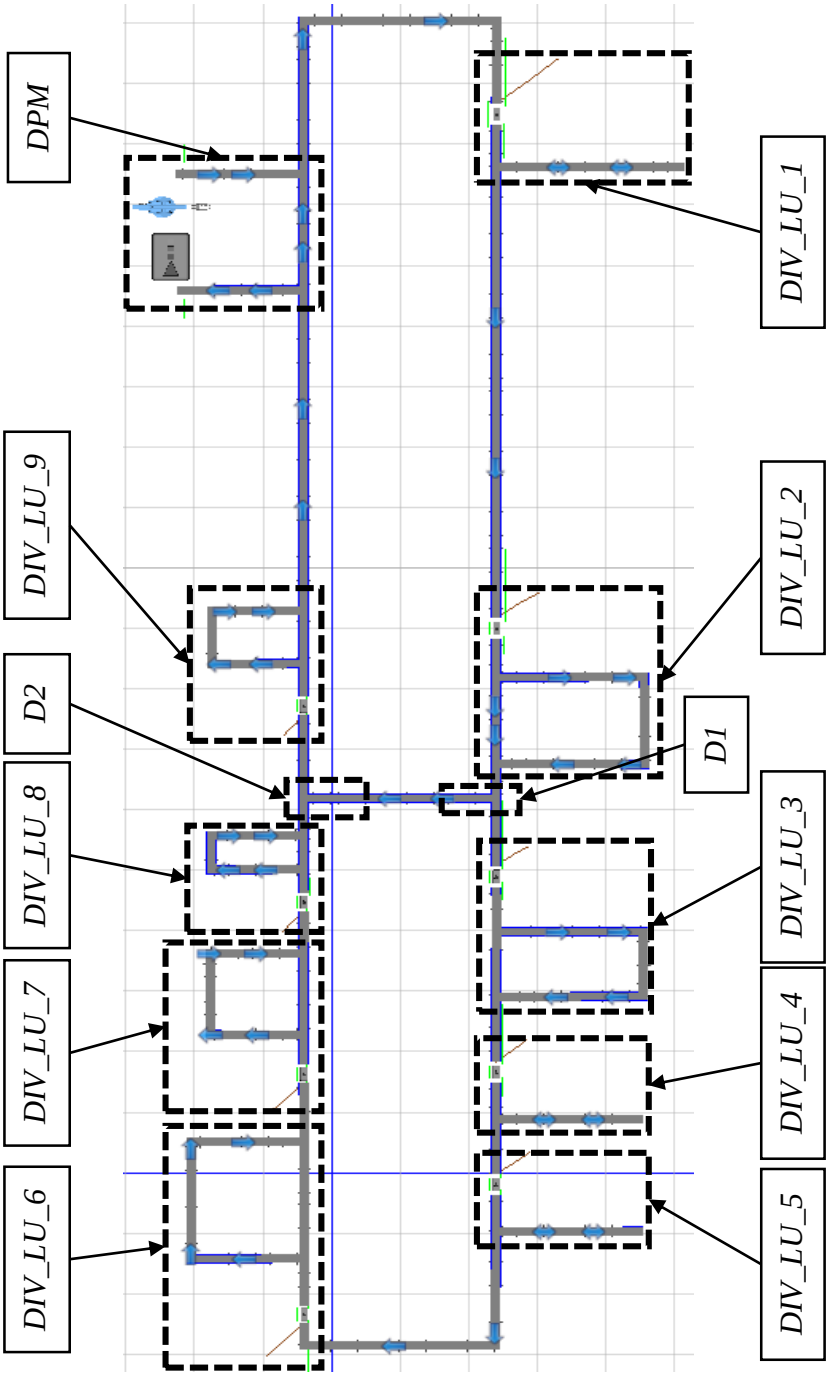

Figure 20 Virtual conveyor: Complex setup

4.3.2.2 Processes view

The same *process* logic that held for the laboratory conveyor, mentioned in Section 4.3.1.2, also holds for the complex conveyor setup.

4.3.3 "Simio controller approach"

A controller purely contained by Simio (shown by Figure 13) was created as an alternative to the "Virtual holonic controller approach", shown in Figure 9. The "Simio controller approach" was applied to both the laboratory and complex conveyors. This meant that the *PLCStatus process step* was no longer used and, as a result, significantly more *process steps* had to be included in the simulation model. The logic represented by the *PLCStatus step* was replicated by a host of *Assign*, *Delay*, *Decide*, *Wait* and *Fire process steps*. Table 5 describes the functions of the *Delay*, *Decide*, *Wait*, and *Fire process steps*.

Table 5 Simio controller *process steps*

Process step	**Function**
Delay	Delays the *process token*'s advancement for a specified amount of time.
Decide	Evaluates a condition to 'true' or 'false', and splits the *process* accordingly.
Wait	Holds a *process token* until an *Event* occurs. *Events* can be created in the *Definition* view, or chosen from a list of inherited (default) *Events*.
Fire	Launches a specified *Event*.

The difference between using the "Virtual holonic controller approach" and "Simio controller approach" as pallet routing strategy is in the *process* logic contained by the *Processes* view of the virtual conveyors. The virtual HLC and *Intrepreter* applications were still needed to convey and write the physical conveyor's state to the *ConMap state* file, mentioned in Section 4.3.1.2. The assumption was made that all pallets contained by the physical conveyor's initial state (when the pallet routing strategy was called) were stationed at their respective destination nodes when a simulation run was started. This concluded the roles of the virtual HLC and *Interpreter* for the Simio controller cases.

Due to the operator no longer being able to launch tasks from the HMI of the virtual HLC, a file, *Test_pallet_destination*, is used for storing the destination of a virtual pallet. The operator is required to save the destination of the virtual pallet to the file, prior to launching a new virtual pallet on the virtual conveyor.

5. Implementation

This chapter provides a detailed description of the different entities comprising the "Virtual holonic controller approach" pallet routing strategy. The operational procedure for the "Simio controller approach" is also described. Referring back to Chapter 3, the different entities forming part of the "Virtual holonic controller approach", included:

- Physical HLC
- Virtual HLC
- *Interpreter*
- *PLCStatus* DLL
- Simio

For this pallet routing strategy, the physical HLC and virtual HLC were run on a different computer than the *Interpreter*, *PLCStatus* DLL and Simio.

This chapter first describes the operational procedure associated with using the "Virtual holonic controller approach", after which some of the key aspects of Windows Communication Foundation (WCF) are described. The implementation details of each entity in the "Virtual holonic controller approach" are described next, followed by a description of the operational procedure associated with the "Simio controller approach".

5.1 Operational procedure for "Virtual holonic controller approach"

The laboratory conveyor is considered in this case for describing how the pallet routing strategy should be applied. This description is provided by Tables 6 and 7, with reference to the HMI shown in Figure 8.

Table 6 Physical conveyor operational procedure

Step	Instruction
1.	Start physical HLC, by clicking *'Start'* button, and wait until all connections between physical HLC and LLC are established
2.	Press '*Conveyor Start*' button
3.	Create tasks via '*Create Task*' button, and feed pallets to the conveyor accordingly

Conditions for deploying the pallet routing strategy:

- The physical conveyor is assumed to be in an idle state, i.e. pallets are awaiting new tasks or waiting in a queue, not being able to continue moving on the conveyor.
- The assumption was made that all processing stations/destination nodes perform the same service. Therefore, pallets entering the conveyor from the pallet magazine (both physical and virtual) are only going to stop at one node, after which it exits the conveyor, returning to the *Pallet Magazine* (signalling the pallet has been processed).

When the operator wants to launch a new pallet, i.e. create a new task using the physical HLC, and have this pallet spend the least amount of time on the conveyor before exiting again, the procedure shown in Table 7 is employed.

Steps 1-10 are repeated for every route to be simulated. Once all routes have been simulated, the operator evaluates the different travelling times that were recorded in the *'Time*' file (Step 10). The quickest route is then recreated as a new task on the physical conveyor.

Due to time constraints, a manual-based procedure was used in this thesis. The manual procedure was adequate to evaluate the route planning strategy. Implementing a more autonomous approach is expected to be feasible, due to the communication abilities already established between the different entities described in the following sections.

5.2 Windows Communication Foundation (WCF)

This section describes the key aspects of the framework used to build service oriented applications – a requirement to understanding the communication link between the *Interpreter* and the *PLC_Status* DLL. WCF is a framework based on service-oriented applications (Löwy and Montgomery, 2016). By making use of service endpoints, different applications are able to interface with one another, given the correct protocols are followed. A WCF service contract was used to establish a connection between the *PLCStatus* DLL and the *Interpreter* application (associated with the "Virtual holonic controller approach"). The developer can specify a WCF service within an application by right clicking the *App.config* file in the solution explorer window in Microsoft Visual Studio – refer to Figure 22.

Table 7 Virtual HLC operational procedure

Step	**Instruction**
1.	Start the virtual HLC application by clicking the '*Start*' button
2.	Start *Interpreter*, and wait until all connections between virtual HLC and *Interpreter* are established
3.	Click '*Conveyor Start*' button on virtual HLC
4.	Create a task via '*Create Task*' button on virtual HLC
5.	Click '*Run simulation*' button on *Interpreter* GUI (Figure 21)
6.	Click '*Refresh*' and '*GO!*' buttons on *Interpreter* GUI (Figure 21)
(Virtual pallet enters virtual conveyor, is processed and exits virtual conveyor again)	
7.	Click '*Stop*' button in Simio to end simulation
8.	Terminate connections between virtual HLC and *Interpreter* applications via clicking '*Emergency Stop*' button on the virtual HLC's HMI
9.	Stop virtual HLC and *Interpreter* applications
10.	Open '*Time*' file in which the travelling time, of the virtual pallet that was launched, was recorded

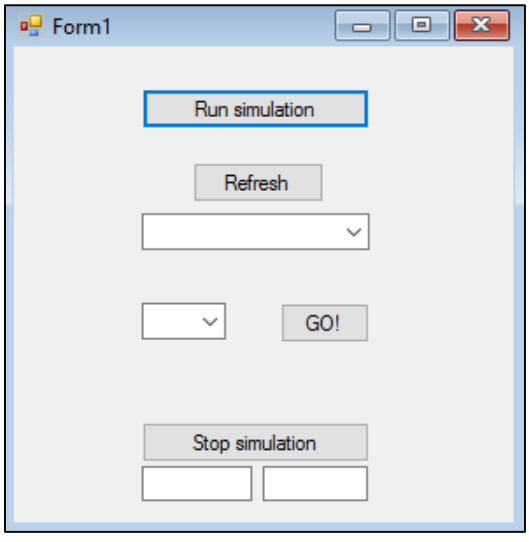

Figure 21 *Interpreter* GUI

As can be noted in Figure 22, the binding between the *PLCStatus* DLL and *Interpreter* application are specified, but, more importantly, the unique endpoints over which the two entities will be communicating, are specified. In summary, WCF (in this thesis) involves a service being hosted by some application (*Interpreter*) to which other applications (*PLCStatus* DLL) are able to subscribe, entering into a service contract with the host.

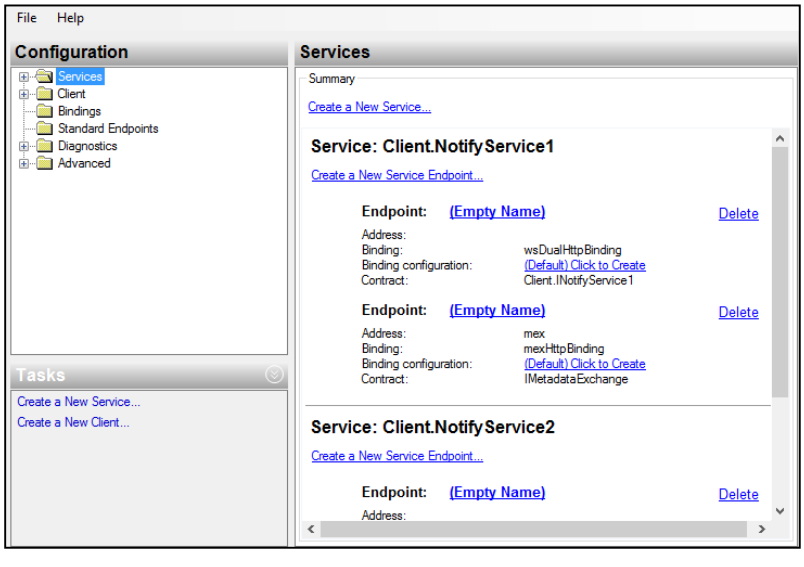

Figure 22 WCF GUI

5.3 Physical HLC

5.3.1 Role

The physical HLC application, as mentioned at the beginning of Chapter 3, is the controller Kotzé (2016) developed, with some minor adjustments as will be mentioned in the following two sections.

5.3.2 Information payload

Additional logic was added to the physical HLC to have the tasks that were created for the physical conveyor, and the resultant conveyor state recorded to separate files.

The information recorded in the files (*Phys_Commands*, *Queues* and *Occupy*) are passed to the virtual HLC. This information is used to duplicate the physical conveyor's state in the conveyor map holon of the virtual HLC.

5.3.3 Payload exchange

Static integer variables were introduced in the physical HLC application to each act as a counter for the queue waiting in front of each destination-type conveyor module. For example, the queue of pallets waiting in front of the *LU1* conveyor module (Figure 6) is stored in the *LU1_queue* variable.

Also, static integer variables were used to indicate whether the destination type conveyor modules were occupied (1) or not (0). For example, should the *LU1* conveyor module be occupied, a value of '1' is stored in the variable *LU1_state*.

Upon clicking the '*Create Task*' button (Step 3 in Table 6), the destination value is written to a file (*Phys_Commands*), with each subsequent new task being written on a new line. The *Phys_Commands* file is cleared every time the *'Start'* button of the physical HLC is clicked, and a particular task's destination value is set to '0' once the destination has been reached. The different *state* variables are written to a file (*Occupy*) whenever the physical HLC is informed that a pallet has reached its destination, and is now awaiting a new task. This logic resides in the *ServerListen* method of the *BServer* class, of the physical HLC application. The values stored in the different *queue* variables, mentioned above, are written to a file (*Queues*) whenever the Conveyor Map Holon is updated. A value of greater than zero indicates that a conveyor module is occupied by a pallet.

5.4 Virtual HLC

5.4.1 Role

This entity is responsible for extracting the state of the physical conveyor from the conveyor map holon of the physical HLC, i.e. the current positions of all pallets on the conveyor. This state is then passed on to the *Interpreter* via TCP/IP. The virtual HLC, via the *Interpreter*, is also responsible for directing virtual pallets on the virtual conveyor, and launching new tasks, should it be wished to launch new virtual pallets onto the virtual conveyor.

5.4.2 Information payload

The information the virtual HLC receives from the physical HLC is described in Section 5.3.2 and is accomplished by step number '3' in Table 7. The virtual HLC proceeds to pass the conveyor state information to the *Interpreter*. The *Interpreter* writes this information to a file named *ConMap state*. The other information passed between the virtual HLC and *Interpreter* is that which would typically occur between the physical HLC and the physical PLCs, illustrated in Appendix B.

5.4.3 Payload exchange

To have the virtual HLC remain oblivious to whether it was talking to virtual conveyor modules or physical conveyor modules, the communication between virtual HLC and *Interpreter* remained to be in TCP/IP. The virtual HLC messages sent to the *Interpreter* was the same as the messages sent from the physical HLC to the physical PLCs on the physical conveyor, only difference being other sockets and IP addresses used. Additionally, upon establishing connections between the virtual HLC and the *Interpreter*, the physical conveyor's status is sent from the virtual HLC to the *Interpreter*.

5.5 Interpreter

5.5.1 Role

The *Interpreter* application is the link between the virtual HLC and the virtual conveyor modules. This application made the commands received from the virtual HLC digestible for Simio to interpret and also translated what was happening on the virtual conveyor (in Simio) for the virtual HLC to understand.

5.5.2 Information payload

The information the *Interpreter* application receives and sends from/to the virtual HLC has already been described in Section 5.4.2.

The *Interpreter* also communicates with the *PLCStatus* DLL which links Simio to the *Interpreter* application, and ultimately the virtual HLC. This link can be regarded as the formulation of 'actuation' commands that the virtual conveyor modules would execute. The *Interpreter* conveys the commands received from the virtual HLC to the *PLCStatus* DLLs, whilst these DLLs inform the *Interpreter* of what is happening on the virtual conveyor.

5.5.3 Payload exchange

In order for communication from the *Interpreter* side to be the same as that coming from the physical PLCs, a class *SendStatusToHMS* with method *State_deter* was created in the *Interpreter* application. The purpose of this method is to generate and send the correct message payloads to the virtual HLC every 200 milliseconds, as expected by the physical HLC. Message payloads are determined by what is happening on the virtual conveyor, and the commands received from the virtual HLC.

The information exchanged between *Interpreter* and the *PLCStatus* DLL is achieved through WCF. The *Interpreter* hosts a service, *NotifyService*, to which the *PLCStatus* DLL subscribes. Each subscription is allocated a new thread in the *Interpreter* application. The *PLCStatus* DLL subscribes to the *NotifyService* service with a name representing a virtual conveyor module's designation (i.e. *DIV_LU*, *LU1*, etc.). The *NotifyService* service contains a class containing a method, *SubscribeIsReadyEvent*, which takes 'name' as input string argument.

The *NotifyService* class also contains a dictionary variable which takes as its value an action delegate, triggered with a string variable, *dest_node*. This delegate returns control back to the subscriber (*PLCStatus* DLL) whenever it is called. The call back invokes a method, *isReady*, which is contained by the *PLCStatus* DLL, with *dest_node* as input argument.

5.6 PLCStatus DLL

5.6.1 Role

The *PLCStatus* DLL was used to link the *Interpreter* application to the Simio application. This DLL is responsible for notifying the *Interpreter* of the whereabouts of the virtual pallets on the virtual conveyor, as well as ordering Simio where to move these virtual pallets on the virtual conveyor.

5.6.2 Information payload

The information that is received by the *PLCStatus* DLL from the *Interpreter* is described in Section 5.5.2. The communication from the *PLCStatus* DLL to Simio is regarded as the actual movement of virtual pallets in the *Facility* view (Section 4.1.1) of Simio. The communication from Simio to the *PLCStatus* DLL is the carryover of the virtual conveyor's status.

5.6.3 Payload exchange

The *PLCStatus* DLL is stored in Simio's *UserExtensions* directory. This DLL is read by the Simio application upon start-up. The DLL will be represented as a *process step* in the *Processes* view of Simio. For the *PLCStatus* logic to be executed, the *process step* needs to be included in a *process* associated with nodes in the *Facility* view. The *Client Name* property is associated with the *process step* and must be specified when inserting this *step* into a *process*. The *Destination Node* property of this *step* contains the *dest_node* string received by the *PLCStatus* DLL from the *Interpreter* (as mentioned in Section 5.5.3).

The *dest_node* string is only received back by the *PLCStatus* DLL when a *process step* has been triggered in Simio by a virtual pallet entering a node which has an *Add-On process* containing the *PLCStatus process step*. The entered node will instruct the virtual pallet where to move next, specified by *dest_node*.

5.7 Typical communication sequences

This section summarizes the "Virtual holonic controller approach" by providing the event sequence taking place when the physical HLC, but also the event sequence when the virtual HLC, is used for navigating pallets. Two examples are used to illustrate the event sequences, both navigating a pallet from *LU1* to *D1* (Figure 6).

5.7.1 Pallet launched by physical HLC

With reference to Table 6, the events shown in

Table 8 are associated with a pallet travelling on the physical laboratory conveyor from the first *Lifting Unit* module (*LU1*) to the *Divert Unit* module (*D1*).

Table 8 Physical HLC event sequence

1.	Pallet arrives at conveyor module *LU1*.
2.	Proximity switch is tripped and message (‘00010001’) is sent by *LU1* PLC to the physical HLC.
3.	Physical HLC responds with a message to *LU1* PLC (‘00000011’).
4.	*LU1* PLC executes a case statement corresponding to ‘3’ in its programmed logic.
5.	Pallet leaves *LU1* for *D1*

5.7.2 Virtual pallet launched by virtual HLC

The example illustrated by Table 9 correlates with Table 7, and depicts the events occurring when a virtual pallet is travelling from the virtual conveyor module, ‘*LU1*’, to the virtual conveyor module ‘*D1*’ (refer to Figure 18).

Table 9 Virtual HLC event sequence

1.	Virtual pallet arrives at node ‘LU1’ containing *PLCStatus process step*.
2.	*PLCStatus* DLL sends message containing client_name ‘LU1’ to *Interpreter*.
3.	*Interpreter* enters an if-statement based on client_name ‘LU1’.
4.	*Interpreter* sends a ‘proximity switch triggered’ message (‘00010001’) to the virtual HLC.
5.	Virtual HLC responds with a message to *Interpreter* (‘00000011’).
6.	*Interpreter* triggers an event based on the message received from the virtual HLC.
7.	Called event sends a string, *dest_node*, with value ‘3’ back to the *PLCStatus* DLL.
8.	*PLCStatus process step* passes ‘3’ to *Search process step* as index to search a list of nodes for directing virtual pallet, waiting at node ‘LU1’, where to go.
9.	Node ‘D1’, corresponding to ‘3’, is returned to the *Search step*, and passed on to the *Assign process step*.

10.	*Assign process step* sets node ‘D1’ as the destination for the virtual pallet.
11.	Virtual pallet leaves node ‘LU1’ and continues to node ‘D1’.

5.8 Operational procedure for “Simio controller approach”

The operational procedure for running the “Simio controller approach” pallet routing strategy is:

1. Run physical HCL as per usual (Table 6)
2. Run virtual HLC and *Client* as per usual and wait for connection to be established, and stop here (not proceeding to press ‘*Conveyor Start*’ buttons and creating tasks – this is only for conveying the physical conveyor’s state to the virtual conveyor)
3. Next, the operator records the destination of the virtual pallet he/she wishes to launch by saving the value to a file (*Test_pallet_destination*)
4. The operator then proceeds to press the ‘*Run simulation*’ button of the *Interpreter* GUI (Figure 21), which will then start the simulation and launch a virtual pallet onto the virtual conveyor containing the current state of the physical conveyor
5. Upon the launched virtual pallet exiting the virtual conveyor, the time this virtual pallet spent on the virtual conveyor is recorded in another file (*Time*), which the operator can consult, once the simulation has stopped

Should the operator wish to run another simulation, the steps (starting from step 2) above, will need to be repeated.

6. Testing and validation

This chapter reports the four tests that were conducted, as well as a comparison of the "Virtual holonic controller approach" and the "Simio controller approach". The first test was to validate virtual pallet travelling times against the physical pallet travelling times on the laboratory conveyor. The second test was to evaluate the route optimization capability of the route planning strategy by making use of the complex conveyor setup. The first two tests relied on the "Virtual holonic controller approach" to make routing decisions on the virtual conveyor. During these tests it was discovered that a significant amount of latency is encountered when employing the "Virtual holonic controller approach", making it possible to only have a maximum of three virtual pallets on the virtual conveyor at the same time.

Concern about the latency led to the emergence of tests three and four. These tests made use of the "Simio controller approach" discussed in Section 3.2.2, instead of the "Virtual holonic controller approach". Seeing as the virtual pallet travelling times found in the third test (Section 6.3) correspond to the travelling times of the first test (Section 6.1), and therefore not requiring validation, the third test evaluated the optimization capability of the "Simio controller approach" on the laboratory conveyor setup. The fourth test evaluated the optimization capability of the "Simio controller approach" used on the complex conveyor. Throughout tests three and four, the latency factor was observed.

6.1 "Virtual holonic controller approach" with laboratory conveyor

6.1.1 Objective

The purpose of this test was to validate the travelling times, according to Simio, against the travelling times of a pallet on the physical conveyor. Two experiments were conducted: the first being without *Delay process steps* and the second with *Delay process steps*.

6.1.2 Experimental setup

The setup used was that detailed in Section 4.3.1.

6.1.3 Results

Table 10 provides the various times it took the physical conveyor to complete tasks, whilst also indicating the time it took the virtual conveyor to complete the same tasks.

Table 10 “Virtual holonic controller approach”: Laboratory conveyor (without Delay steps)

		Time			
From	**To [s]**	**Lab (Actual) [s]**	**Simio (Simulated without *Delay process steps*) [s]**	**Diff [s]**	**Diff [%]**
DPM	DIV_LU	52	39.85	12.15	23
DPM	LU3	45	35.2	9.8	22
DPM	LU1	19	14.6	4.4	23
DIV_LU	DPM	12	8.8	3.2	27
LU3	DPM	9	8.45	0.55	6
LU1	DPM	37	29.05	7.95	21

As can be noted from columns three and four in Table 10, the physical pallet travelling times are greater than the virtual pallet travelling times. This discrepancy was presumably due to delays that physical pallets encounter when moving across junctions, while the simulation did not have such delays. Apart from having mirrored the physical conveyor in Simio (e.g. conveyor moving speed and conveyor lengths), the delays of the physical conveyor were accounted for by making use of numerous *Delay process steps* in the simulation. The *Delay process steps* were introduced on each path connecting two different virtual conveyor modules, bringing the virtual pallet travelling times as close as possible to the physical pallet travelling times, as is illustrated in Table 11. The delays incurred by the *Delay process steps* is represented by the *Diff* column in Table 10, calculated as the *Actual time* - *Simulated time*.

Table 11 "Virtual holonic controller approach": Laboratory conveyor (with Delay steps)

		Time			
From	**To**	**Lab (Actual) [s]**	**Simio (Simulated with *Delay process steps*) [s]**	**Diff [s]**	**Diff [%]**
DPM	DIV_LU	52	55.05	3.05	6
DPM	LU3	45	47.6	2.6	6
DPM	LU1	19	19	0	0
DIV_LU	DPM	12	12.7	0.7	6
LU3	DPM	9	7.99	1.01	11
LU1	DPM	37	38.75	1.75	5

6.1.4 Interpretation of results

As can be seen in Table 10, the largest percentage difference between the virtual conveyor times and the physical conveyor times, is 27% - moving a pallet from the first *Lifting Unit with Transverse Conveyor* module (*DIV_LU*) to the *Divert Unit with Pallet Magazine* module (*DPM*). The largest percentage difference after the *Delay process steps* were introduced, was significantly lower, at a value of 11% (Table 11) - having a pallet exit the conveyor (at *DPM*) starting from the second *Lifting Unit* module (*LU3*). Table 11 shows that, with the *Delay process steps*, the travel times predicted by Simio are close enough to the actual times to allow Simio to be used for evaluating alternative routes for pallets.

6.2 "Virtual holonic controller approach" with complex conveyor

6.2.1 Objective

The objective of this test was to prove the optimality and robustness of the "Virtual holonic controller approach" route planning strategy developed in this thesis. Three scenarios were considered in this test:

1. Finding an alternative route when a path is blocked (e.g. due to machine failure), implying that a virtual pallet has to find a route that bypasses the blocked processing station. This scenario was illustrated by the *DIV_LU_7*

module (refer to Figure 20) in a video clip that can be found at https://www.youtube.com/watch?v=Sbc8K5t3zQ8.

2. Have a virtual pallet follow a shorter route, which contains a processing station, although this is not the destination of the virtual pallet. The alternative route is longer and does not contain the processing station. The *DIV_LU_6* module (refer to Figure 20) was used to illustrate this in a video found at https://www.youtube.com/watch?v=Sbc8K5t3zQ8. The only time the longer branch would be used is when a virtual pallet is already travelling on the shorter branch.
3. Find a shorter route, utilizing a crossover branch, if there are no virtual pallets occupying the branch. This was illustrated by the path linking modules *D1* and *D2* in Figure 20. Due to the physical HLC determining a physical pallet's entire route upon launching the pallet, this scenario exhibited the same behaviour (as can be noted at 00:54 in a video found at the following link: https://www.youtube.com/watch?v=MkLpMtGVtRE). Another video found at the following link demonstrate the occasion when a virtual pallet does indeed follow the crossover route:
   - https://www.youtube.com/watch?v=Sbc8K5t3zQ8

   Note that this decision is taken upon launching a virtual pallet, i.e. a virtual pallet will only be assigned the crossover route, upon entering the virtual conveyor, should no virtual pallets be travelling on the crossover link at that point, otherwise the crossover route will not be considered.

6.2.2 Experimental setup

The setup illustrated by Figure 20 was used for creating two conveyor states. The first conveyor state was to illustrate the crossover branch not being used by a virtual pallet and had the following initial state:

- *DIV_LU_4*: 1
- *DIV_LU_5*: 1

A new virtual pallet was launched when the link connecting virtual conveyor modules *D1* and *D2* was unoccupied.

The second conveyor state was to illustrate the crossover branch being used by a virtual pallet and had the following initial state:

- *DIV_LU_1*: 1
- *DIV_LU_2*: 1

A new virtual pallet was launched when the virtual pallet which had occupied *DIV_LU_1* was travelling on the link connecting virtual conveyor modules *D1* and *D2*.

6.2.3 Results

The results of this test are best observed by considering the video clips mentioned in Section 6.2.1. The video found at the link https://www.youtube.com/watch?v=Sbc8K5t3zQ8 shows the shorter route, containing the crossover branch, leaving a virtual pallet destined for node '12' to finish its task and exit the conveyor first. This video also demonstrates scenarios 1 and 2 mentioned in Section 6.2.1.

The video found at https://www.youtube.com/watch?v=MkLpMtGVtRE shows the case of a virtual pallet not being able to take the crossover branch. This occurred due to another virtual pallet occupying the crossover branch, upon launching of a new virtual pallet destined for node '12' (*DIV_LU_9*).

6.2.4 Interpretation of results

The results that were obtained showed that all objectives were successfully met, qualifying the "Virtual holonic controller approach" as being both optimal and robust. However, the dead reckoning principle, on which the physical HLC relies for identifying pallets, could prove to be a problem in future experiments. The virtual HLC would occasionally jump to the conclusion that the first pallet to enter the conveyor is the first to exit the conveyor – a characteristic which does not complement optimization.

6.3 "Simio controller approach" with laboratory conveyor

6.3.1 Objective

The objective of this test was to, firstly, show that a significant decrease in latency is evident when employing the "Simio controller approach" instead of the "Virtual holonic controller approach". Secondly, some optimization capability of the "Simio controller approach" was evaluated to demonstrate the underlying principle of this thesis that 'the shortest route is not necessarily the optimal route'.

6.3.2 Experimental setup

The optimization logic in this test is illustrated by having a simulation run initialized with a virtual pallet destined for destination node '2' (*LU1*, Figure 18) (and have the virtual pallet continue to node '0' (*DPM*)) in the presence of the virtual conveyor already flooded with virtual pallets. A conveyor map state was created (by making use of the physical HLC) to have the following queues of virtual pallets at the different destination nodes (of which one virtual pallet is occupying the virtual conveyor module/node):

- *LU1*: 2
- *DIV_LU*: 6
- *LU3*: 9

En route to node '0', upon reaching the *D1* module, the virtual pallet is able to take either one of two routes:

- *D1* and *DIV_LU* route (geographically longer route)
- *D1* and *LU3* route (geographically shorter route)

Two simulations were run, one for each of the two routes: route *D1-DIV_LU* and route *D1-LU3*.

6.3.3 Results

The travelling times of the virtual pallet taking the two different routes were recorded in Table 12. As can be noted, the shortest route (as per Dijkstra's algorithm) is not necessarily the optimal route. Also, no latency/delays were evident in this experiment.

Table 12 Test 3: Results

D1, LU3	***D1, DIV_LU***
196 seconds	186 seconds

6.3.4 Interpretation of results

By substituting the "Virtual holonic controller approach" with the "Simio controller approach", latency was of no concern any longer. This qualified the approach discussed in Section 3.2.2, as a potential solution to the problem of route planning for conveyors. Furthermore, a first attempt at demonstrating the optimization

capability of the “Simio controller approach” proved to be successful, and gave credibility to this pallet routing strategy.

6.4 “Simio controller approach” with complex conveyor

6.4.1 Objective

The objective of this test was to evaluate whether the “Simio controller approach” could successfully address the same objectives as those mentioned in Section 6.2.1, also using the virtual conveyor shown in Figure 20. Also, the amount of latency associated with employing the “Simio controller approach”, was to be observed.

6.4.2 Experimental setup

For the first setup, use of the crossover branch between modules *D1* and *D2* was illustrated. The number of virtual pallets waiting at each node when the simulation run was initialized, was:

- *DIV_LU_5*: 1
- *DIV_LU_4*: 2
- *DIV_LU_3*: 1

For the second setup, illustrating a launched virtual pallet not being able to use the crossover branch between modules *D1* and *D2*, the initial state of the virtual conveyor was:

- *DIV_LU_5*: 1
- *DIV_LU_4*: 2
- *DIV_LU_3*: 1
- *DIV_LU_1*: 1

6.4.3 Results

Videos at the following web links illustrate, respectively, the first and second setup described above:

- https://www.youtube.com/watch?v=vBv1HuRavRU
- https://www.youtube.com/watch?v=zNwYaeQneIQ

Both videos illustrate the first and second scenarios described in Section 6.2.1. The third objective lead to the results in Table 13 being obtained.

Table 13 Test 4: Results

Longer route	Crossover route
266 seconds	185 seconds

6.4.4 Interpretation of results

As with the third test, no latency was observed and indicated that the "Simio controller approach" is scalable. The different objectives mentioned in Section 6.2.1 were also met, qualifying the "Simio controller approach" as being both optimal and robust. The "Simio controller approach" had the additional capability of identifying virtual pallets that were on the conveyor. This lead the "Simio controller approach" to believe that the first pallet entering the conveyor is not necessarily the first to exit the conveyor, which in contrast with the "Virtual holonic controller approach" (refer to Section 6.2.4).

6.5 Comparison: "Virtual holonic controller approach" versus "Simio controller approach"

The "Virtual holonic controller approach" is a viable strategy to the optimization of pallet routing decisions, and meets the requirements of being scalable. A latency factor, however, started to surface during the first test, leaving the virtual conveyor only being capable of handling a maximum of three virtual pallets at a time. The direct cause of this inherent latency was not investigated, as this was not the focus of the thesis. The possible cause was, however, narrowed down to the interfacing that occurs between either the virtual HLC and *Interpreter*, or the *Interpreter* and *PLCStatus* DLLs. Should more than three virtual pallets occupy the virtual conveyor, the latency seems to reach such a degree that it causes Simio to time out. The amount of latency experienced by Simio seemed to be directly proportional to the amount of virtual pallets occupying the virtual conveyor.

In tests 3 and 4 the "Simio controller approach" was applied to the same conveyor case studies used by tests 1 and 2. This approach appeared to be latency-free, and had no limitation put to the number of virtual pallets that can occupy the virtual conveyor at the same time.

Reconfiguration effort of the "Simio controller approach" was still in doubt – a property the "Virtual holonic controller approach" did not lack. It was, however, found that the effort associated with scaling the "Virtual holonic controller approach" is the same as the effort associated with scaling the "Simio controller approach" due to its modularity, and increased the credibility of the "Simio controller approach".

7. Conclusions and recommendations

The objective of this thesis was to develop a pallet routing strategy that supports a holonic conveyor controller. The basis of the strategy that was decided upon was simulation via the Simio simulation software package. This strategy substituted the Dijkstra algorithm that Kotzé (2016) implemented with the holonic controller. The main goal was to determine, having already committed and launched pallets on the conveyor, what the quickest route will be for a pallet that is about to enter the conveyor.

Two strategies were developed in addressing the above-mentioned objective: A "Virtual holonic controller approach" and a "Simio controller approach". The former consisted of an integrated approach containing the following entities: virtual HLC, *Interpreter*, *PLCStatus* DLL and Simio. The "Simio controller approach" neglected the use of these entities, and had its entire controller logic contained within the Simio application.

The results obtained show that simulation is indeed a possible solution in the face of 'the shortest route not being the optimal route'. Reflecting back on the characteristics of reconfigurability, mentioned in Section 2.1.4, the "Virtual holonic controller approach" fits well into this context. An important characteristic was that of integrability (with the physical HLC). Integrability was proved by not having changed the physical HLC significantly, and having the virtual HLC be oblivious to the fact that it was not communicating with the PLC hardware controlling the physical conveyor (as LLC), but instead with software modules exhibiting PLC behaviour.

An alternative approach to address the latency associated with the "Virtual holonic controller approach", and having the routing strategy applicable to large scale conveyors, was investigated. A controller, implemented in the simulation software package (Simio), was developed, which mimics the behaviour of the "Virtual holonic controller approach". This approach showed to be latency-free and also being able to exhibit optimization behaviour, such as rerouting in case of machine failures. This approach also proves to be more autonomous in that the operator is only required to click the simulation application's 'Start' button to run a scenario. This is opposed to the effort required, illustrated by Table 7, when using the "Virtual holonic controller approach".

Future work includes improving the diagnostic element of the HLC, e.g. by making use of RFID readers, and have this be added to the "Virtual holonic controller approach". The author however believes that simulation software, such as Simio, is capable of mimicking RFID-like behaviour in identifying different virtual pallets.

Ideally, the virtual HLC application and the rest of the "Virtual holonic controller approach" should be invoked autonomously, relying on as little operator input as

possible. Due to time constraints, improving the autonomy of this approach is considered to be future work.

Future work might also include having the route optimization strategy being applied not only when a pallet is about to enter the conveyor, but also during its course of moving between junctions on the physical conveyor.

8. References

Barbosa, J., Leitão, P., Adam, E. and Trentesaux, D., 2015. Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. *Computers in Industry.* Vol. 66: 99-111.

Bonabeau, E., Theraulaz, G., Deneubourg, J.L., Aron, S. and Camazine, S., 1997. Self-organization in social insects. *TREE*. Vol. 12, No. 5: 188-193.

Borangiu, T., Rəileanu, S., Berger, T. and Trentesaux, D., 2015. Switching mode control strategy in manufacturing execution systems. *International Journal of Production Research.* Vol. 53, No. 7: 1950-1963.

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

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., 2009. Introduction to Algorithms. Third edition. The MIT Press, Cambridge.

Graefe, R., 2016. *A C# implementation of a station controller for a Reconfigurable Manufacturing System*. MEng Thesis. Stellenbosch University.

Kelton, W.D., Smith, J.S., Sturrock, D.T. and Verbraeck, A., 2010. Simio and Simulation: Modeling, Analysis, Applications. McGraw-Hill, New York.

Keshavarzmanesh, S., Wang, L. and Feng, H.Y., 2010. A hybrid approach for dynamic routing planning in an automated assembly shop. *Robotics and Computer-Integrated Manufacturing.* Vol. 26, No. 6: 768-777.

Khuswendi, T., Hindersah, H. and Adiprawita, W., 2011. UAV Path Planning Using Potential Field and Modified Receding Horizon A* 3D Algorithm. *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*. No. 6021579.

Koestler, A., 1969. The Ghost in the Machine. Arkana Books, London.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. and Van Brussel, H., 1999. Reconfigurable Manufacturing Systems. *CIRP Annals - Manufacturing Technology.* Vol. 48, No. 2: 527-540.

Koren, Y. and Shpitalni, M., 2010. Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems.* Vol. 29, No. 4: 130-141.

Kotzé, M.J., 2016. *Modular Control of a Reconfigurable Conveyer System*. MEng Thesis. Stellenbosch University.

Krogh, B.H. and Thorpe, C.E., 1986. Integrated path planning and dynamic steering control for autonomous vehicles. *Proceedings - 1986 IEEE International Conference on Robotics and Automation*. pp. 1664-1669.

Kruger, K. and Basson, A.H., 2015. Implementation of an Erlang-based Resource Holon for a Holonic Manufacturing Cell. *Studies in Computational Intelligence.* Vol. 594: 49-58.

Kruger, K. and Basson, A.H., 2016. Erlang-based control implementation for a holonic manufacturing cell. *International Journal of Computer Integrated Manufacturing*. DOI: 10.1080/0951192X.2016.1195923

Leitão, P., 2008. Self-organization in Manufacturing Systems: Challenges and Opportunities. *Proceedings – 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. No. 4800673: 174-179.

Leitão, P., 2009. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence.* Vol. 22, No. 7: 979-991.

Leitão, P., Barbosa, J. and Trentesaux, D., 2012. Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence*. Vol. 25, No. 5: 934-944.

Leitão, P., Colombo, A.W. and Restivo, F.J., 2005. ADACOR: A Collaborative Production Automation and Control Architecture. *IEEE Intelligent Systems.* Vol. 20, No. 1: 58-66.

Leitão, P., Mařík, V. and Vrba, P., 2013. Past, Present, and Future of Industrial Agent Applications. *IEEE Transactions on Industrial Informatics.* Vol. 9, No. 4: 2360-2372.

Le Roux, A., 2013. *Control of a Conveyor System for a Reconfigurable Manufacturing Cell*. MEng Thesis. Stellenbosch University.

Löwy, J. and Montgomery, M., 2016. Programming WCF Services. O'Reilly Media, California.

Maturana, F., Shen, W. and Norrie, D.H., 1999. MetaMorph: An adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research.* Vol. 37, No. 10: 2159-2173.

McFarlane, D., Kollingbaum, M., Matson, J. and Valckenaers, P., 2001. Development of algorithms for agent based control of manufacturing flow shops. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Vol. 1: 146-151.

Mehrabi, M.G., Galip Ulsoy, A. and Koren, Y., 2000. Reconfigurable manufacturing systems and their enabling technologies. *International Journal of Manufacturing Technology and Management.* Vol. 1, No. 1: 114-131.

Mujumdar, A. and Padhi, R., 2011. Evolving Philosophies on Autonomous Obstacle/Collision Avoidance of Unmanned Aerial Vehicles. *Journal of Aerospace Computing, Information and Communication*. Vol. 8, No. 2: 17-41.

Peng, Y. and McFarlane, D., 2004. Adaptive agent-based manufacturing control and its application to flow shop routing control. *Production Planning and Control*. Vol. 15, No. 2: 145-155.

Rüßmann, M., Lorenz, M., Gerbert, P., Waldner, M., Justus, J., Engel, P. and Harnisch, M., 2015. *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries*. [Online]. Available: https://www.bcgperspectives.com/content/articles/engineered_products_project_business_industry_40_future_productivity_growth_manufacturing_industries/?chapter=2#chapter2_section4. [2016, February 9].

Saidi-Mehrabad, M., Dehnavi-Arani, S., Evazabadian, F. and Mahmoodian, V., 2015. An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. *Computers and Industrial Engineering*. Vol. 86: 2-13.

Sallez, Y., Berger, T. and Trentesaux, D., 2009. A stigmergic approach for dynamic routing of active products in FMS. *Computers in Industry*. Vol. 60, No. 3: 204-216.

Sargent, R.G., 2013. Verification and validation of simulation models. *Journal of Simulation*. Vol. 7: 12-24.

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

Tournassoud, P., 1986. A strategy for obstacle avoidance and its application to multi-robot systems. *Proceedings - 1986 IEEE International Conference on Robotics and Automation*. pp.: 1224-1229.

Ulusoy, G., Sivrikaya- Şerifoğlu, F. and Bilge, Ü., 1997. A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. *Computers and Operations Research*. Vol. 24, No. 4: 335-351.

Valckenaers, P. and Van Brussel, H., 2016. Design for the unexpected: From Holonic Manufacturing Towards a Humane Mechatronics Society. Butterworth-Heinemann, Oxford.

Van Belle, J., Saint Germain, B., Verstraete, P., Valckenaers, P., Ali, O., Van Brussel, H. and Cattrysse, D., 2009. A Holonic Chain Conveyor Control System: An Application. *4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. Vol. 5696: 234-243.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P., 1998. Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*. Vol. 37, No. 3: 255-274.

Van Dyke Parunak, H., Baker, A.D. and Clark, S.J., 1997. The AARIA agent architecture: An example of requirements-driven agent-based system design. *Proceedings of the 1997 1st International Conference on Autonomous Agents*. pp.: 482-483.

Warren, C.W., 1989. Global Path Planning Using Artificial Potential Fields. *IEEE International Conference on Robotics and Automation - 1989*. pp.: 316-321.

Appendix A: Conveyor modules

This appendix shows the message exchanges between the HLC and LLC, for each conveyor module associated with the laboratory conveyor. The same modules, except for the *Lifting Units*, were used to construct the complex conveyor.

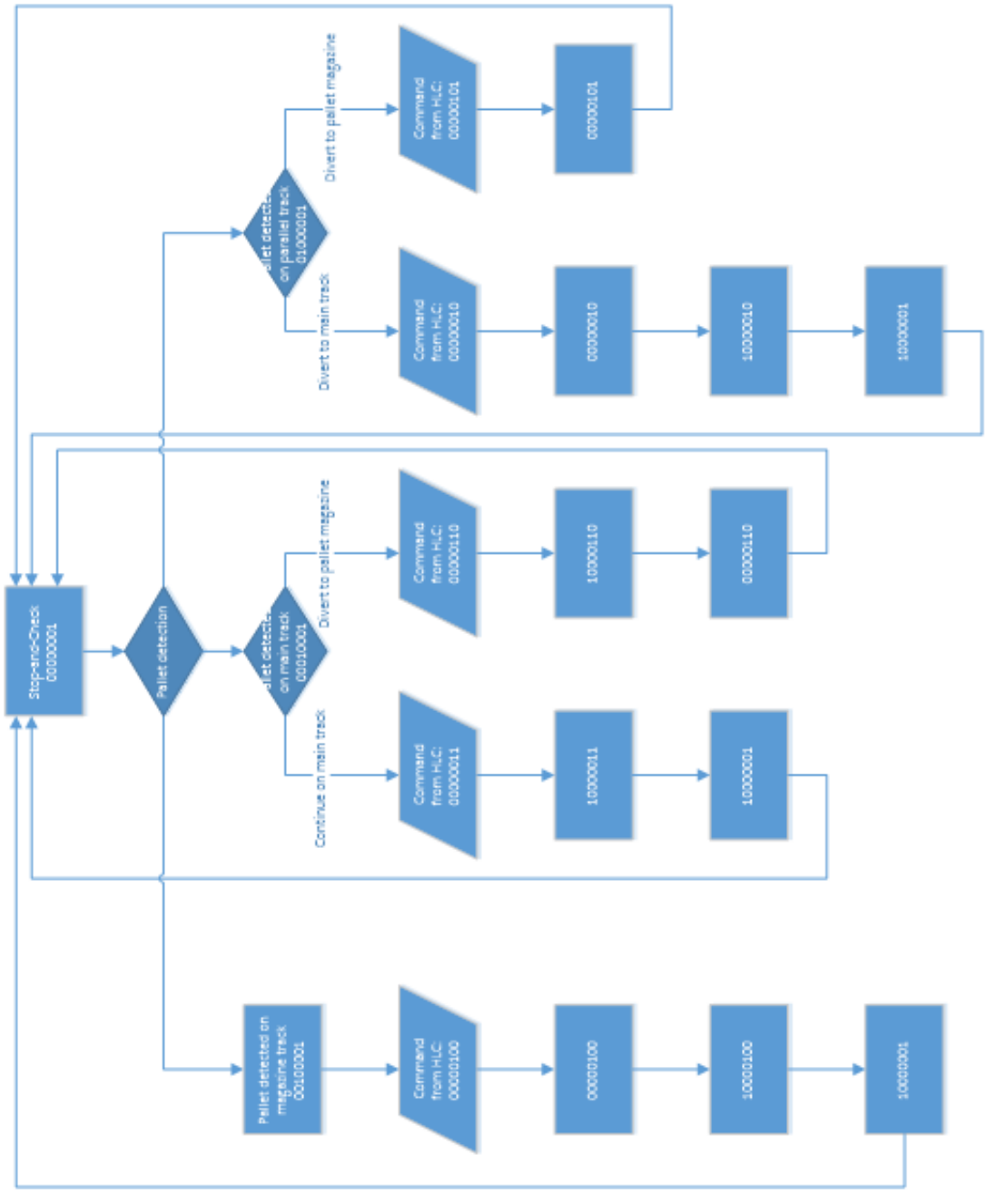

Figure 23 Divert Unit with Pallet Magazine (DPM)

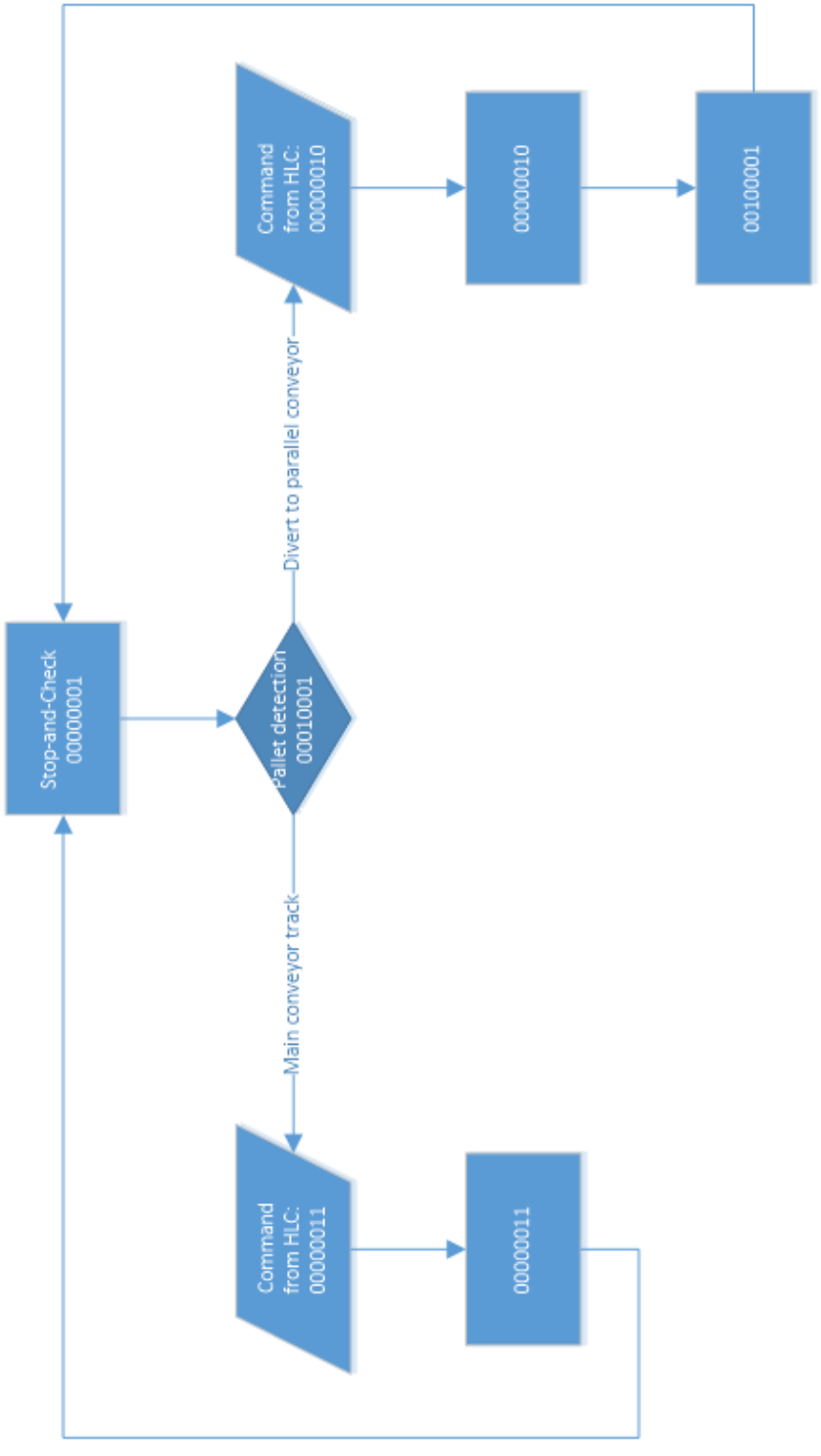

Figure 24 Divert Unit (D1)

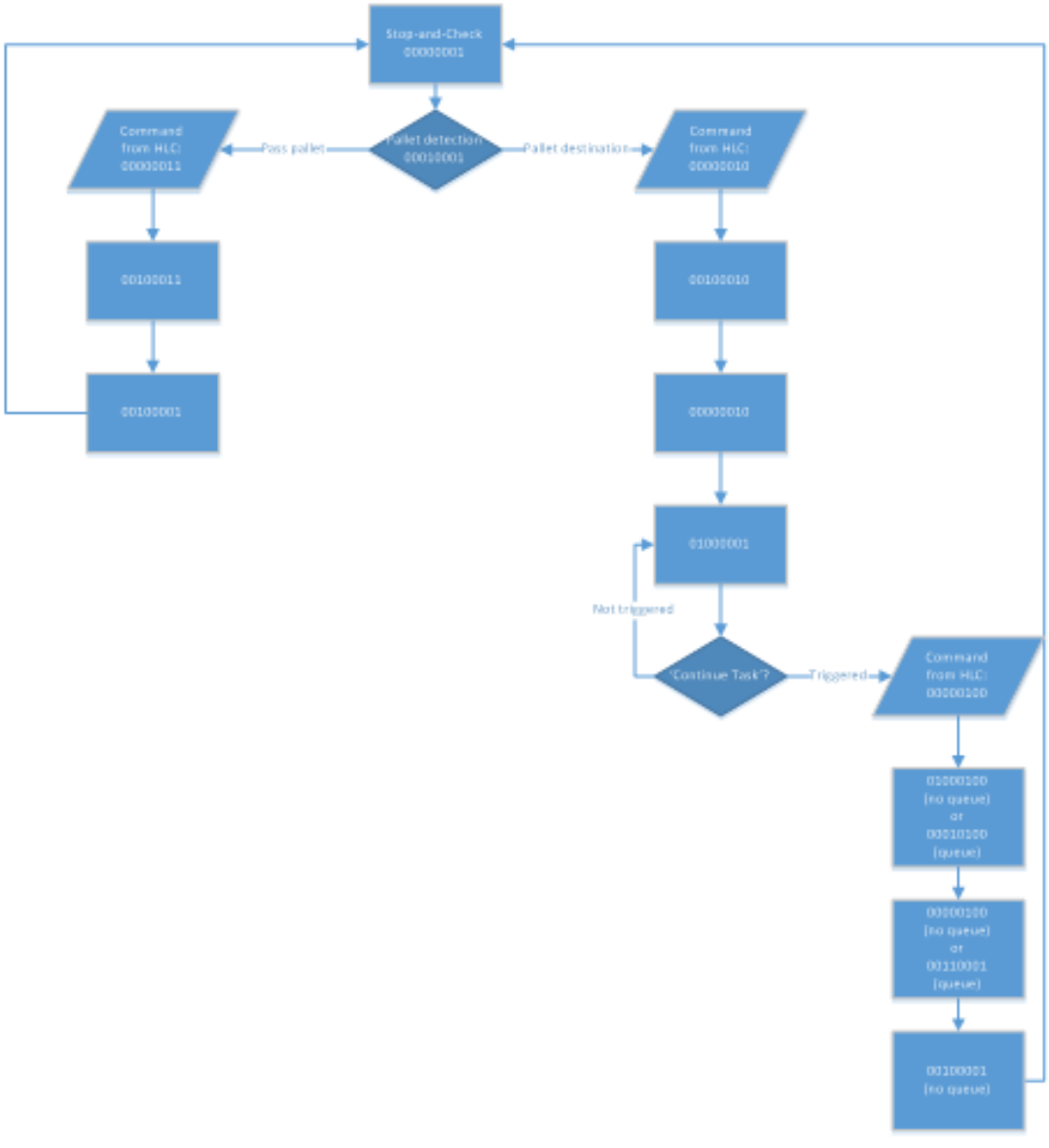

Figure 25 Lifting Unit (Unoccupied) with Transverse Conveyor (DIV_LU)

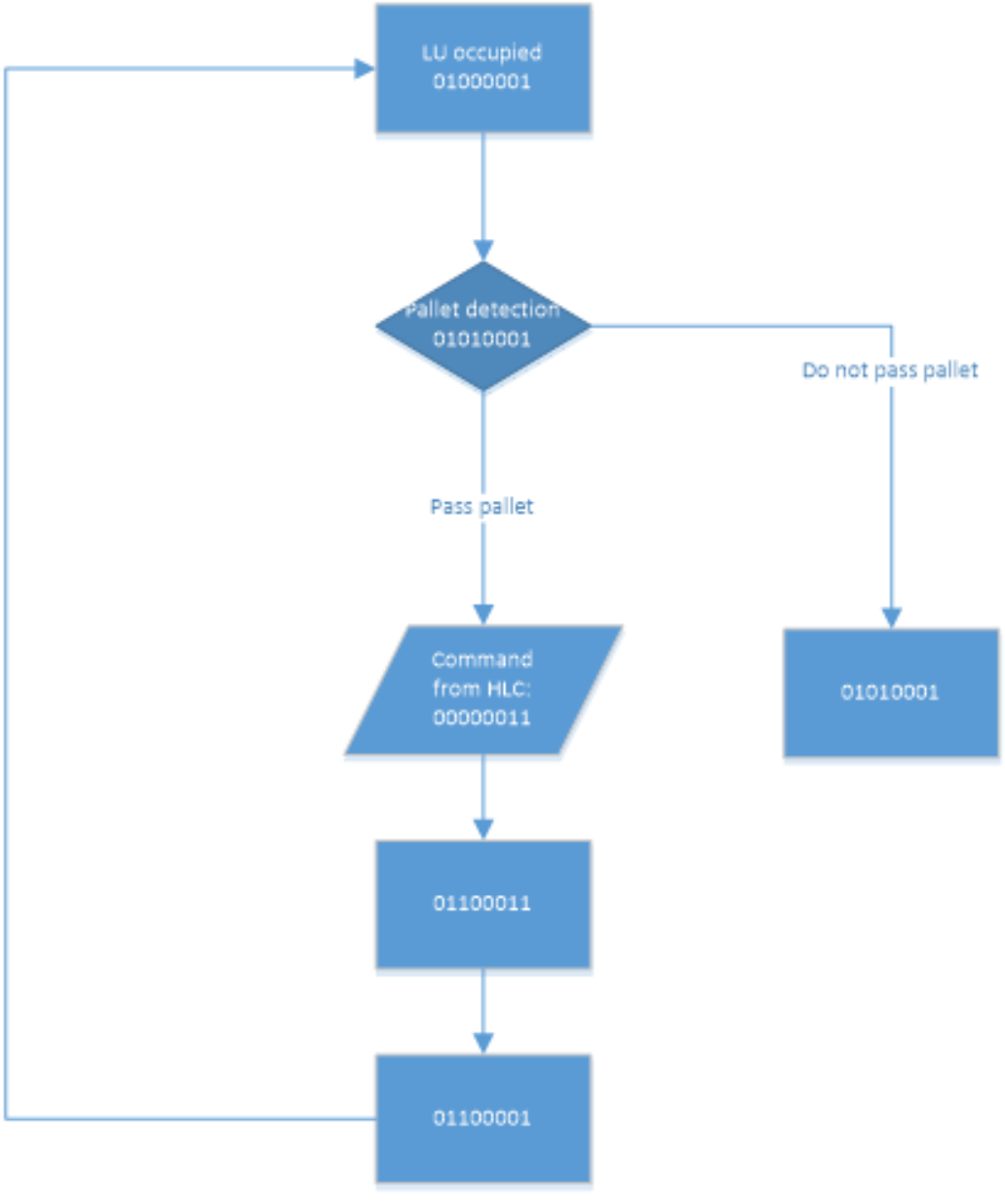

Figure 26 Lifting Unit (Occupied) with Transverse Conveyor (DIV_LU)

Appendix B: Routing sequences

This appendix provides some typical message sequences that are associated with completing certain tasks on the laboratory conveyor.

A legend to these tables:

> Each row with a thicker border, containing bold faced values, are messages that are sent by the HLC application.
>
> The thinner border rows, with non-bold values, are the messages that are sent by the different PLCs to the HLC application, at intervals of 200 milliseconds.

Commonalities among the different tables (with reference to Appendix A):

> Entry *1.* for every table is when the *Start* button is clicked by the operator, and after some delay (< 10 seconds) each PLC module (client) has established a connection to the HLC (server).
>
> Entry *2.* for every table is when the *Conveyor Start* button is clicked by the operator, which in turn prompts the HLC to send a command ordering each PLC to be in the 'stop-and-check' state.
>
> Entry *4.* for every table is after the user has clicked the *Create Task* button, followed by a pallet tripping the first proximity sensor upon entering the conveyor from the pallet magazine.
>
> Entry *5.* for every table is the command sent by HLC to the *DPM* module. This message indicates the case to be evaluated by the STL program on the PLC hardware, after it has been notified by the controller that a pallet is within its proximity (refer to 'Entry *4.*').
>
> The last two entries of every table is the termination of the entire system. The second to last entry occurs when the *Emergency Stop* button is clicked, whilst the last entry is the resultant 'passive state' status sent by each PLC to the HLC.

Appendix B.1: DPM to LU1

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000010**	**-**	**-**	**-**
12.	00000001	00010010	00000001	00000001	00000001
13.	00000001	00000010	00000001	00000001	00000001
14.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
15.	00000000	00000000	00000000	00000000	00000000

Appendix B.2: DPM to DIV_LU

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000011**	**-**	**-**	**-**
12.	00000001	00010011	00000001	00000001	00000001
13.	00000001	00000011	00000001	00000001	00000001
14.	00000001	00000001	00000001	00000001	00000001
15.	00000001	00000001	00000001	00000001	00000001
16.	**-**	**-**	**-**	**-**	**-**
17.	00000001	00000001	00000001	00000001	00000001
18.	00000001	00000001	00000001	00000001	00000001
19.	00000001	00000001	00000001	00000001	00000001
20.	00000001	00000001	00000001	00010001	00000001
21.	**-**	**-**	**-**	**00000010**	**-**
22.	00000001	00000001	00000001	00000010	00000001
23.	00000001	00000001	00000001	00100001	00000001
24.	00000001	00000001	00000001	00000001	00000001
25.	00000001	00000001	00000001	00000001	00010001
26.	**-**	**-**	**-**	**-**	**00000010**
27.	00000001	00000001	00000001	00000001	00100010
28.	00000001	00000001	00000001	00000001	00000010
29.	00000001	00000001	00000001	00000001	01000001
30.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
31.	00000000	00000000	00000000	00000000	00000000

Appendix B.3: DPM to LU3

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000011**	**-**	**-**	**-**
12.	00000001	00010011	00000001	00000001	00000001
13.	00000001	00000011	00000001	00000001	00000001
14.	00000001	00000001	00000001	00000001	00000001
15.	00000001	00000001	00000001	00000001	00000001
16.	**-**	**-**	**-**	**-**	**-**
17.	00000001	00000001	00000001	00000001	00000001
18.	00000001	00000001	00000001	00000001	00000001
19.	00000001	00000001	00000001	00000001	00000001
20.	00000001	00000001	00000001	00010001	00000001
21.	**-**	**-**	**-**	**00000011**	**-**
22.	00000001	00000001	00000001	00000011	00000001
23.	00000001	00000001	00000001	00000001	00000001
24.	00000001	00000001	00010001	00000001	00000001
25.	**-**	**-**	**00000010**	**-**	**-**
26.	00000001	00000001	00000010	00000001	00000001
27.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
28.	00000000	00000000	00000000	00000000	00000000

Appendix B.4: (DPM to LU1) to DPM

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000010**	**-**	**-**	**-**
12.	00000001	00010010	00000001	00000001	00000001
13.	00000001	00000010	00000001	00000001	00000001
(‘Continue Task’ button pressed by operator)					
14.	**-**	**00000100**	**-**	**-**	**-**
15.	00000001	00000100	00000001	00000001	00000001
16.	00000001	00000001	00000001	00000001	00000001
17.	00000001	00000001	00000001	00000001	00000001
18.	**-**	**-**	**-**	**-**	**-**
19.	00000001	00000001	00000001	00000001	00000001
20.	00000001	00000001	00000001	00000001	00000001
21.	00000001	00000001	00000001	00000001	00000001
22.	00000001	00000001	00000001	00010001	00000001
23.	**-**	**-**	**-**	**00000010**	**-**
24.	00000001	00000001	00000001	00000010	00000001
25.	00000001	00000001	00000001	00100001	00000001
26.	00000001	00000001	00000001	00000001	00000001
27.	00000001	00000001	00000001	00000001	00010001
28.	**-**	**-**	**-**	**-**	**00000011**
29.	00000001	00000001	00000001	00000001	00100011
30.	00000001	00000001	00000001	00000001	00100001
31.	01000001	00000001	00000001	00000001	00000001
32.	**00000101**	**-**	**-**	**-**	**-**
33.	00000101	00000001	00000001	00000001	00000001
34.	00000001	00000001	00000001	00000001	00000001
35.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
36.	00000000	00000000	00000000	00000000	00000000

Appendix B.5: (DPM to DIV_LU) to DPM

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000011**	**-**	**-**	**-**
12.	00000001	00010011	00000001	00000001	00000001
13.	00000001	00000011	00000001	00000001	00000001
14.	00000001	00000001	00000001	00000001	00000001
15.	00000001	00000001	00000001	00000001	00000001
16.	**-**	**-**	**-**	**-**	**-**
17.	00000001	00000001	00000001	00000001	00000001
18.	00000001	00000001	00000001	00000001	00000001
19.	00000001	00000001	00000001	00000001	00000001
20.	00000001	00000001	00000001	00010001	00000001
21.	**-**	**-**	**-**	**00000010**	**-**
22.	00000001	00000001	00000001	00000010	00000001
23.	00000001	00000001	00000001	00100001	00000001
24.	00000001	00000001	00000001	00000001	00000001
25.	00000001	00000001	00000001	00000001	00010001
26.	**-**	**-**	**-**	**-**	**00000010**
27.	00000001	00000001	00000001	00000001	00100010
28.	00000001	00000001	00000001	00000001	00000010
29.	00000001	00000001	00000001	00000001	01000001
(‘Continue Task’ button pressed by operator)					
30.	**-**	**-**	**-**	**-**	**00000100**
31.	00000001	00000001	00000001	00000001	01000100
32.	00000001	00000001	00000001	00000001	00000100
33.	00000001	00000001	00000001	00000001	00100001
34.	01000001	00000001	00000001	00000001	00000001
35.	**00000101**	**-**	**-**	**-**	**-**
36.	00000101	00000001	00000001	00000001	00000001
37.	00000001	00000001	00000001	00000001	00000001

38.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
39.	00000000	00000000	00000000	00000000	00000000

Appendix B.6: (DPM to LU3) to DPM

	DPM	***LU1***	***LU3***	***D1***	***DIV_LU***
1.	00000000	00000000	00000000	00000000	00000000
2.	**00000001**	**00000001**	**00000001**	**00000001**	**00000001**
3.	00000001	00000001	00000001	00000001	00000001
4.	00100001	00000001	00000001	00000001	00000001
5.	**00000100**	**-**	**-**	**-**	**-**
6.	00000100	00000001	00000001	00000001	00000001
7.	10000100	00000001	00000001	00000001	00000001
8.	10000001	00000001	00000001	00000001	00000001
9.	00000001	00000001	00000001	00000001	00000001
10.	00000001	00010001	00000001	00000001	00000001
11.	**-**	**00000011**	**-**	**-**	**-**
12.	00000001	00010011	00000001	00000001	00000001
13.	00000001	00000011	00000001	00000001	00000001
14.	00000001	00000001	00000001	00000001	00000001
15.	00000001	00000001	00000001	00000001	00000001
16.	**-**	**-**	**-**	**-**	**-**
17.	00000001	00000001	00000001	00000001	00000001
18.	00000001	00000001	00000001	00000001	00000001
19.	00000001	00000001	00000001	00000001	00000001
20.	00000001	00000001	00000001	00010001	00000001
21.	**-**	**-**	**-**	**00000011**	**-**
22.	00000001	00000001	00000001	00000011	00000001
23.	00000001	00000001	00000001	00000001	00000001
24.	00000001	00000001	00010001	00000001	00000001
25.	**-**	**-**	**00000010**	**-**	**-**
26.	00000001	00000001	00000010	00000001	00000001
(‘Continue Task’ button pressed by operator)					
27.	**-**	**-**	**00000100**	**-**	**-**
28.	00000001	00000001	00000100	00000001	00000001
29.	00010001	00000001	00000100	00000001	00000001
30.	**00000110**	**-**	**-**	**-**	**-**
31.	10000110	00000001	00000100	00000001	00000001
32.	10000110	00000001	00000001	00000001	00000001
33.	00000110	00000001	00000001	00000001	00000001
34.	00000001	00000001	00000001	00000001	00000001
35.	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
36.	00000000	00000000	00000000	00000000	00000000