A Component-Based Virtual Engineering Approach to PLC Code Generation for Automation Systems

By

Bilal Ahmad

DOCTORAL THESIS

Submitted in partial fulfilment of the requirements

for the award of Doctor of Philosophy of

Loughborough University

January 2014

Wolfson School of Mechanical and Manufacturing Engineering

Loughborough University

ABSTRACT

In recent years, the automotive industry has been significantly affected by a number of challenges driven by globalisation, economic fluctuations, environmental awareness and rapid technological developments. As a consequence, product lifecycles are shortening and customer demands are becoming more diverse. To survive in such a business environment, manufacturers are striving to find a cost-effective solution for fast and efficient development and reconfiguration of manufacturing systems to satisfy the needs of changing markets without losses in production.

Production systems within automotive industry are vastly automated and heavily rely on PLC-based control systems. It has been established that one of the major obstacles in realising reconfigurable manufacturing systems is the fragmented engineering approach to implement control systems. Control engineering starts at a very late stage in the overall system engineering process and remains highly isolated from the mechanical design and build of the system. During this stage, control code is typically written manually in vendor-specific tools in a combination of IEC 61131-3 languages. Writing control code is a complex, time consuming and error-prone process. The lack of effective tools for offline verification of control code further reduces the reliability of the current programming practice. As a result, a large number of errors remain undetected until the commissioning phase, which results in a significant increase in the cost and the lead-time of a project.

The work presented in this thesis focuses on addressing the limitations of the traditional PLC programming practice by proposing a novel approach for generation of control code and HMI screens based on the control behaviour of the component-based virtual model of an automation system. The main contributions of this research are 1) a method for the definition of control logic within 3D-based virtual engineering tools to enable direct deployment of the complete control code, 2) the design and implementation of a target PLC control software architecture that complies with the current industrial best practice used in the automotive industry and 3) the automatic generation of PLC code and HMI screens for this architecture utilising the control information defined in virtual models of manufacturing cells.

The approach presented integrates the controls engineering with manufacturing process planning and mechanical engineering using a collaborative 3D-based Virtual Engineering (VE) tool and thus enables the definition and validation of the control logic of a system within VE tools before the physical build of a machine. The approach has been experimentally evaluated from various perspectives to identify its strengths and limitations during the development, reconfiguration and operational phases of automation systems.

Keywords: Component-based automation, automatic code generation, PLC programming, virtual commissioning, control systems, assembly automation.

Acknowledgements

In the name of Allah the most gracious and the most merciful

I would like to acknowledge and express my gratitude to my supervisor Professor Robert Harrison. I am deeply grateful for his support, encouragement and invaluable guidance throughout this research.

I would also like to acknowledge my colleagues at the University of Warwick and Loughborough University who helped and taught me so much, especially Dr. Young Saeng Park, Dr. Xiangjun Kong, Dr. Izhar Ul Haq, Dr. Daniel Veara and Dr. Stuart Charles McLeod for their involvement in this research work and constructive discussions that contributed to my knowledge in the area of automation systems.

My gratitude also goes to Dr. Leslie Lee and Keith Hills from Ford Motor Company UK, Werner Viebrock from ThyssenKrupp Krause Systems Engineering GmbH, Martyn Bromage and Paul Sear from Schneider Electric, and Ian Lindsay from Siemens for their invaluable support, arranging trainings, constructive discussions and feedback during this research.

I would also thank University of Engineering and Technology (Peshawar, Pakistan), Engineering and Physical Sciences Research Council (EPSRC) UK and Technology Strategy Board (TSB) UK for their financial support to conduct this research.

Finally, I passionately express gratitude to my beloved parents, sister Azra and brothers Aftab and Ilyas for their support, patience, love and prayers.

To My Parents

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Due to the changing business environment, manufacturing industry is facing greater challenges and risks than ever before [1]. Globalisation, economic fluctuations, innovations in technology, environmental concerns, and demanding customers are just some of the factors that are triggering frequent changes in market requirements [2-5]. It is becoming evident that the era of mass-production of one-of-a-kind product is now being replaced by the era of market niches. As a consequence, product lifecycles are constantly shortening and introduction of new products is becoming more frequent [6, 7].

The automotive industry is considered as a backbone of the European economy [8]. The effects of the changing manufacturing landscape are also evident within the automotive industry [9]. Automotive manufacturers are facing severe risks due to economic downturns, overcapacity and volatility of demand. The ability of a company to quickly respond to changes by offering innovative, customised and competitively priced products to meet customer demands on a timely basis is fundamental to maintaining market share in both existing and emerging markets [10].

Due to ever-increasing need to introduce new products frequently, the traditional mass-production model of the automotive industry is under direct attack [11]. The automotive industry is under immense pressure to cope with rapid product changes (such as low-emission internal combustion engines, hybrid drive systems and electric power packs) to fulfil changing customer demands and local market requirements all over the world [12]. Reducing the time-to-market while maintaining high quality and low cost is becoming a challenge. It has been reported that a few months delay in the launch of products, such as motor vehicles or large sub-assemblies, can cause a significant loss in the market share [12, 13]. For instance, a delay in production of a powertrain assembly line at Ford Motor Company results in a loss of £20,000/hour. This together with increasing pressure to achieve lower manufacturing costs and high quality, motivates vehicle manufacturers to find efficient and cost effective solutions for fast adaption of their resources to respond to market requirements without losses in production [14]. As a consequence, new visions and technologies are urgently required to support fast and cost-effective development, reconfiguration and reuse of manufacturing systems.

1.1 Problem Statement

1.1.1 Motivation

To capture short windows of opportunity, meeting customer demand by introducing innovative

vehicles in short time is increasingly becoming critical to automotive manufacturers. As a consequence, the responsiveness of automation systems to adapt quickly and economically to produce new generations of products is now fundamental for a company's success.

Despite the improvements made through just-in-time and lean production strategies, manufacturing industry struggles to respond efficiently and effectively to the changing business requirements [15]. This is in-part due to the fixed configuration of the manufacturing systems which cannot support reconfigurability and reuse with changing market needs. The current approach of built-in flexibility to perform a number of machining/assembly operations in the production systems to handle the responsiveness addressed the issue to some extent. However, such flexibility comes with high initial investment and may suffer from obsolescence when significant changes are required in the manufacturing process that cannot be attained with the built-in flexibility [16].

To realise reconfiguration, one of the approaches is the use of component-based modular production systems. Modular systems are designed at the onset to be reconfigurable by standardising the interfaces and created from basic hardware and software modules that can be re-arranged with changing requirements. Using a modular approach, machine builders are able to build a standard library of pre-validated components. A new system can be built by reconfiguring and assembling the required components without the need to understand their internal complexity [9, 16]. The use of component-based modular systems addressed the requirements of hardware reconfiguration to a great extent. However, reconfiguration at the logical level is still a challenging task. This is because the current modular approach at the logical level is typically based on a parameterisation approach, which involves changing parameters of the standard software components. Such solutions (e.g., FOM, section 2.5.2.3) typically lead to large amount of monolithic control code able to cover a wide range of system configuration variants. This increases the complexity of a system and of each component's internal code, and thus making them more difficult to develop and maintain individually.

It has been established that a major obstacle in realising an efficient and reconfigurable approach for control code development is the currently fragmented approach to the engineering of manufacturing systems [16]. The current engineering approach is well established and follows a classical sequential model. The mechanical design and build, electrical design and installation, and control engineering occur independently in a sequence and integration occurs only during commissioning. This is due to the use of heterogeneous tools, proprietary data-structures, department-specific engineering methods and a lack of cross engineering domain and cross organisation collaboration tools, which leads to repetition of work and loss of information [12, 17].

Control engineering normally starts at a late stage of the machine development process and remains highly isolated from mechanical design and build. Control logic is coded by interpreting process charts and timing diagrams. This interpretation is typically carried out in an ad-hoc manner that requires extensive manual effort, expertise, and time [18]. In addition, the control code cannot be verified until the machine is assembled and ready for commissioning. This leads to inconsistencies

which are only discovered at a very late stage of the machine development process, which in turn results in high unexpected cost and prolonged commissioning and ramp-up phases [19-21]. In such an engineering approach, commissioning and production ramp-up becomes the most challenging phases [22].

Figure 1-1 shows the time and cost profiles that might be associated with the lifecycle phases of an automotive powertrain assembly system. It can be seen that the peak cost incurs at the commissioning and ramp-up phase. This is because of the issues that arise when fragmented solutions are integrated at these stages and the unseen errors translate into non-working machines, unmatched control functionality, catastrophic failures and additional redesign phases. It has been reported that the correction of defective control software consumes up to 60% of commissioning time and accounts for 15% of time-to-delivery [23].

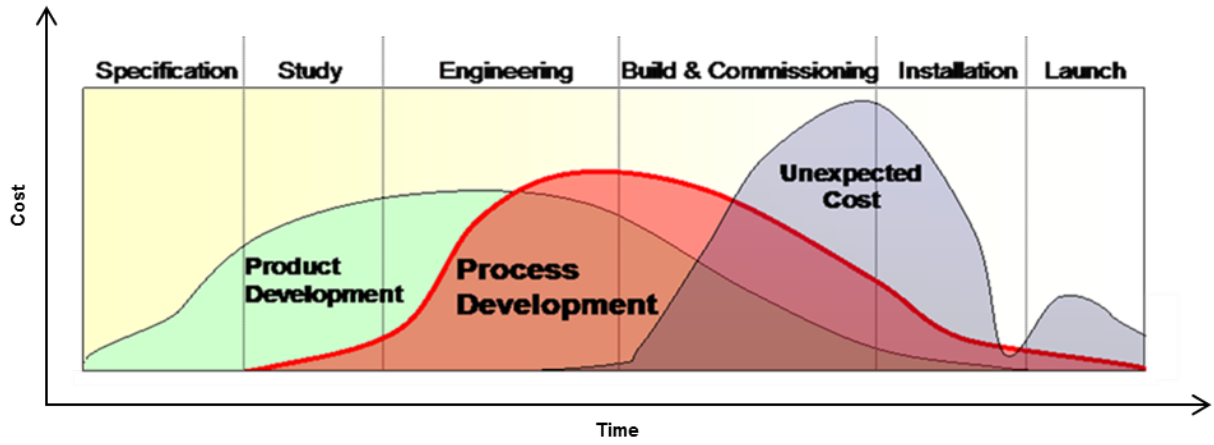


Figure 1-1 Cost and time profiles associated with the lifecycle phases of powertrain assembly systems [24]

In order to improve engineering practice, both academia and industry are conducting extensive research to facilitate an integrated engineering approach that can enable efficient information reuse and dynamic reconfiguration. Over the past decade, a most promising research outcome is the virtual engineering of automation systems which provides a 3D-based collaborative engineering environment. Industry has shown significant interest in virtual prototyping and commissioning of manufacturing systems using virtual engineering tools to validate and optimise manufacturing resources before the physical build [25]. The objective of virtual commissioning is to ensure design robustness and check for inconsistencies by mimicking the actual shop-floor setup before investing significant resources to implement the physical prototype build [26]. With virtual commissioning, the engineers are able to mock-up a manufacturing system to simulate component assembly, mechanical movements to determine clashes, human operator work processes and overall station layout and processes. The immediate effect and benefit is a substantial reduction in the machine build time and significant savings, realised by virtual validation before the physical build [27].

It can be seen that the inherent attributes of component-based modular systems combined with virtual

commissioning can potentially compress the development time, and thus relieve the challenges of the manufacturing industry. Over the past decade, there has been a considerable progress in the development of virtual engineering tools to provide a coherent collaborative environment to assess and optimise the performance of component-based reconfigurable automation systems [28].

1.1.2 Justification of Research

The recent trend of the use of IT tools allows engineers to carry out commissioning activities in a 3D virtual environment without having real production facilities. Thus providing a platform to check inconsistencies and optimise the performance of a manufacturing system before physical commissioning. Virtual commissioning does not eliminate the need for physical commissioning; however, it can significantly compress the machine build time by identifying structural defects and inconsistencies in control behaviour before the physical build [21, 29]. This creates a new parallel process and lessens the time pressure that exists in the classical sequential approach for machine development [30].

Virtual commissioning can be categorised into Full Simulation of Machinery (FSM) and Hardware-In-Loop (HIL) [23]. FSM is essentially a 3D simulation study of a manufacturing system for process validation. FSM is a mature research area and a growing number of industries are now using it to validate the manufacturing processes. On the other hand, HIL commissioning is used for testing of the control software by connecting the virtual prototype of a machine to real control hardware, thereby avoiding making changes to the control software afterwards on the shop-floor. However, due to poor integration of 3D simulation tools and control engineering tools, the control logic defined within the simulation tools cannot be reused [31]. As a result, the current HIL commissioning methods rely on classical manual programming practice to translate the specified control behaviour into control code. Any change in the configuration of a machine requires corresponding modification of both the virtual model as well as the control code. This results in repetition of work and many technical problems inherent in the manual programming approach remain unresolved while machines are programmed. Andersson et al. [20] emphasise the need for control information reuse and the direct deployment of control code with the help of the following phrase:

"The control function is first defined by the mechanical engineer and then implemented in a robot simulation tool where robot paths are added. After that the mechanical design is verified by a 3-D simulation, and finally the same control function is implemented in the PLC program. In addition, manually programmed control programs often suffer from inadequacies and errors, and the result is a rather rigid and inflexible control function that does not allow work to proceed other than in the specified sequence."

To overcome these problems, there is an increasing interest in the reuse of control information from the 3D-based manufacturing process simulation tools to automatically generate a deployable control

code. Such approach of generating control code will eliminate the manual programming, and thus enables the dynamic reconfiguration of the control software.

Automatic generation of control code is an integrated and seamless IT solution but very little research is found in this area. In recent years, a number of academic researchers and control vendors have introduced methods and tools to generate the control code from 3D-based manufacturing process simulation tools. However, none of these tools and methods is as yet adopted by industry. There are a number of reasons for this. According to Bergert [32], solutions based on commercial tools have problems with tool-specific inflexibility and open programming interfaces, which leads to simplified program with limited control functions. According to Andersson et al. [20], the current research outcomes mainly focus on nominal production operation of a machine (i.e. automatic cycle) only. Nevertheless, the nominal production cycle is the most prominent but is only a small part of the required control functionality. In reality, the code for the automatic cycle only represents 10-20% of the required control code. In addition, control functions, such as manual mode control, interlocking of mechanisms, fault diagnostics, and HMI screen generation are often neglected [17]. Such incomplete programs require manual rework before deployment, and thus make them prone to errors. Moreover, most of the new proposed methodologies are carried out with minimal understanding of the actual logic design methods used in industry. The generated programs are often unstructured, and thus are difficult to understand and debug. This lack of compliance with the current industrial practice and software standards results in a poor reception for these approaches from industry.

Given the demanding requirements of industrial control systems, this research area is still in its infancy and a complete practical solution simply does not exist. It is very unlikely that industry will agree to utilise tools which provide partial and non-standard solutions for the development of control logic for their manufacturing systems. Research is therefore needed in order to further explore the potential of the virtual engineering for complete control code deployment which is acceptable to industry and meet the demand for efficient and cost-effective manufacturing system engineering.

1.2 Research Description

1.2.1 Research Background

Since mid-nineties, Automation Systems Group (ASG) at the University of Warwick[1] has conducted research on the component-based approach for the development of manufacturing automation systems. This research has been conducted in collaboration with Ford Motor Company, ThyssenKrupp System Engineering, Bosch-Rexroth and Schneider Electric. The primary objective of the group's ongoing research is to enhance the collaborative engineering approach and enable the robust launch of manufacturing automation systems within the automotive industry by developing engineering tools to

[1] Formerly known as Manufacturing System Integration (MSI) Research Institute based at Loughborough University

support the efficient and integrated development and deployment of production system.

Key aspects of the group's research are i) the development of a 3D-based virtual engineering tool to virtually construct and commission production facilities out of generic set of components to validate and optimise them before the physical build and ii) the deployment of runtime control systems based on the validated behaviour of a production system defined within virtual engineering tools.

The development and implementation of the 3D-based virtual engineering tool and the deployment of the runtime system has been achieved via a number of EPSRC, TSB and FP7 funded research projects, such as COMPAG (COMponent-based Paradigm for AGile automation) and COMPANION (Common Model for PArtNers in automatION), RI-MACS (Radically Innovative Mechatronics and Advanced Control Systems), SOCRADES, BDA (Business Driven Automation) and 3Deployment (Direct Digital Deployment).

Figure 1-2 shows an overview of the research and development work of the group as a whole. The boxes with a blue background indicate the specific areas of the author's work. The focus of the author's contributions was realising a new approach to code generation for the deployment of PLC-based runtime control systems.

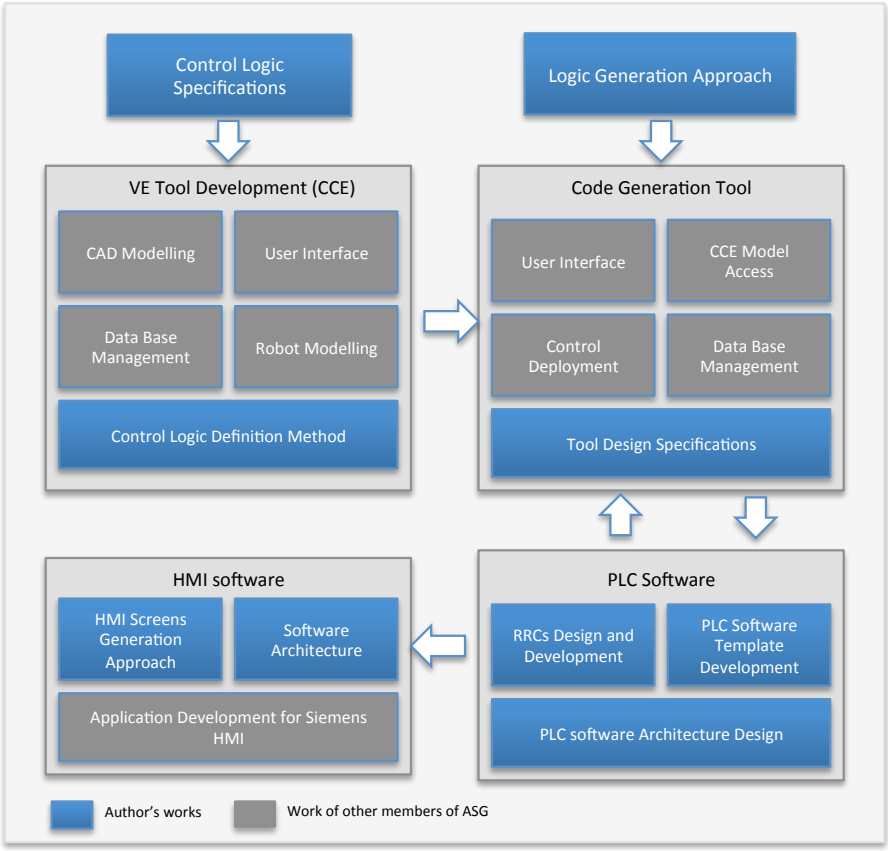


Figure 1-2 Development work conducted by the author and other members of the research group

1.2.2 Research Hypothesis

The principle hypothesis of this research is that if the logic control behaviour defined within virtual process planning tools to carry out virtual commissioning is in accordance with the current best practice and fulfils all the runtime control systems requirements then the same control information can be reused to automatically generate the complete control code for the target PLC.

1.2.3 Research Aim and Objectives

The aim of this research is to enable the generation of control code from the control behaviour defined within 3D based virtual manufacturing process simulation tools. Once a machine is virtually validated by simulating it in a process simulation tools then by reusing the same control information, bug-free control software can be generated automatically. This will enable a dynamic and efficient response to changes by eliminating the time-consuming and error-prone method of manually writing low-level control code in vendor specific tools. As a result, a significant reduction in the cost and time associated with the commissioning and ramp-up phases can be achieved.

The objectives of this research are:

- **Objective 1:** To review existing PLC programming practice within the automotive manufacturing sector, to identify the limitations of existing programming practices, and to recognise the control software structural and functional requirements.
- **Objective 2:** To enhance control information and specify a method for the definition of control logic within a virtual process simulation tool, the CCE, to enable direct deployment of the complete control code.
- **Objective 3:** To design a software architecture that complies with the current PLC software structures used in automotive industry production machines and to design an approach to automatically generate PLC control code according to this software architecture by utilising control information defined in virtual models of manufacturing cells developed in the manufacturing process planning tool, the CCE.
- **Objective 4:** To design and develop and an approach for integrated and automatic generation of HMI Screens.
- **Objective 5:** To implement a prototype system to validate the research hypothesis, evaluate the approach and show the achievement of research objectives.

1.2.4 Research Methodology

In order to undertake this research effectively and systematically, the research work was conducted in five phases. Various research methods [33] such as exploratory, explanatory, descriptive, grounded theory and experimental were adopted in these research phases. A brief description of the work involve at each phase is given below:

Phase 1 – Literature and Technology Review

The objective of this phase was to explore the research area and understand the problem. During this phase current research, technologies, engineering practices and tools were critically studied in order to recognise their strengths and limitations to help identify research gaps.

Phase 2 – Studying Industrial Practice and Developing Specifications

In this phase, several visits were made to end-user and machine builder sites to study the engineering process for developing powertrain assembly systems. The requirements of end-user and machine builders from control engineering perspective were documented. The outcome of this research phase was in the form of documentation of the functional and architectural specifications for implementation of the PLC control software.

Phase 3 – Synthesis of a New Control Software Deployment Method

In this research phase, the concept of the control software deployment method was developed, exploiting information reuse from virtual engineering tools. The control logic definition method within VE tool was devised to enable the direct deployment of the control software. A novel control software architecture was proposed to enable direct deployment and to fulfil the industrial requirements. In addition, an engineering tool was designed and developed for the generation of control code using the proposed software architecture. Its applicability within the automotive powertrain assembly systems was then verified against the requirement specifications developed in the previous phase.

Phase 4 – Pilot Application

The objective of this phase was to show a proof-of-concept application. In this phase, the proposed control engineering method was applied to the commissioning and programing of a Festo test rig. After the successful implementation of this table-top system additional experiments were carried out using two industrial demonstration machines; the Automation System Demonstrator (ASD) at the University of Warwick and automotive engine assembly line test loop from ThyssenKrupp System Engineering GmbH located at Manufacturing Technology Centre (MTC), UK.

Phase 5 - Evaluation and Future Developments

This research phase aimed to evaluate the potential benefits of the proposed research concept. In this phase, the pilot applications were tested by creating various scenarios to evaluate the performance of the proposed and implemented approach. A number of factors were considered during this evaluation, such as ease of system design, reconfigurability and runtime performance. The test results were then compared with the traditional manual programming to appraise the advantages of the proposed approach. Based on the evaluation, a number of gaps to be solved by future developments are suggested to enhance this new approach and make it useable in the industry.

1.2.5 Research Scope and Limitations

This thesis demonstrates the applicability of the proposed concept within the automotive powertrain

assembly sector. The control system specifications are based on the requirements of Powertrain Operations, Ford Motor Company. However, the proposed concept is applicable in a wider context, such as body-in-white manufacturing, airport baggage handling, and warehouse automation systems. Whilst this research has only targeted generating code for the Siemens SIMATIC STEP 7, Schneider Electric Unity Pro and PLCopen platforms, the method is potentially applicable for other platforms implementing distributed systems and Service Oriented Architecture (SOA).

1.3 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 provides a background study and reviews the state-of-the-art technology and practices to identify their limitations in the context of business requirements. To address these limitations, Chapter 3 proposes and implements a methodology for logic definition within virtual prototyping tools and a framework for logic generation to enable the direct deployment approach. For proof of concept and evaluation of the proposed approach, Chapter 4 describes the conducted experimental work and analysis of the empirical results. Based on the evaluation of the experimental work, Chapter 5 summarises the main contribution of the work, the lessons learned from the research and presents potential future research work.

2 Literature and Technology Review

2.1 Introduction

The manufacturing world has undergone a number of major paradigmatic shifts in the past century [34]. In the early twentieth century, the focus was to increase productivity and decrease cost by using mass-production techniques. By the middle of the century, the focus shifted towards product variants and quality. In the eighties, the debate on the restructuring of the manufacturing resources started due to the success of Japanese companies adopting lean production strategies [35]. In the nineties, the end of the cold war, introduction of free trade, and advances in Information and Communication Technologies (ICT) resulted in business globalisation [35]. The world progressively became a global village and as a result new business strategies were adopted to penetrate into new markets. Outsourcing, offshoring, and migration of production to developing countries became common business norms [36].

Over the past decade, exceptionally dynamic and unprecedented market demands are forcing the manufacturing landscape to undergo a transformational shift once again [37]. In contrast with the past, product variants are increasing while product lifecycles are decreasing [2]. These changes have a direct impact on the business model of the automotive industry. The traditional mass-production model of the automotive industry is now under direct attack [11].

To achieve long-term business goals, manufacturing industry not only needs to deal with the dilemmas of today but also has to prepare itself for completely different future requirements [1]. It is becoming increasingly important to devise cost effective and efficient mechanisms to enable manufacturers to instantly respond to any change by fast adaptation of manufacturing resources to cope with the new market requirements [14].

2.2 Evolution of Manufacturing Paradigms

A manufacturing paradigm can be defined as a philosophy that underpins the techniques and practices required for companies to sustain their internal and external business environment. This is commonly associated with the phrase “paradigm shift” that refers to the significant change in these principles [38].

Since the birth of modern manufacturing, industries have gone under several paradigm shifts due to changing market conditions, economic trends and technology. In response, manufacturing philosophies have evolved over time to fit the business requirements. This section briefly reviews the traditional manufacturing paradigms and systems. The aim is to provide an understanding of why the previous manufacturing philosophies can no longer cope with the current business requirements of industry.

2.2.1 Mass Production

Mass-production refers to the production of large amounts of standardised products. The concept of the mass production paradigm emerged in the nineteenth century as an outgrowth of the industrial revolution brought by Henry Ford, also known as Fordism. Henry Ford's intention was to manufacture the largest number of cars with lowest possible cost to produce affordable vehicles for the general public [39].

The production of the Ford Model-T is the first and most famous example of the mass production paradigm. Henry Ford modernised the manufacturing process of the Model-T with his concepts of production of a single model, interchangeable parts, work specialisation and the moving assembly line. This significantly reduced the average cycle time (from 514 to 1.19 minutes) required for car assembly. The moving assembly line and breaking the assembly process into simple tasks increased the capital cost and the worker ratio. Nevertheless, tremendous reduction in the cost per unit was achieved [40].

The reduction in the unit cost resulted in the increased sales as more people could afford the products. Due to this, the Ford Motor Company was able to increase its production from slightly over 10,000 automobiles per year to 500,000 in 1916, and to 2 million in 1923. This achievement brought by Fordism caught the attention of other manufacturers and soon gained worldwide acceptance in all manufacturing sectors [41].

To further increase the productivity, Taylor of the Bethlehem Steel Company introduced the principles for *Scientific Management,* which led to specialisation and task allocation by breaking down the production process into smaller and simple tasks with a set target time for each. The principle also included the payment of bonuses to those that achieved them [34, 42]. Breaking down production process resulted in simple and smaller tasks, which were not only easy to specialise by humans but also to automate them to achieve higher production rates. This resulted in replacing the general-purpose machines with special-purpose dedicated manufacturing systems to achieve high repeatability of operations. The dedicated systems abolished the demand for highly skilled versatile labour. As a consequence, the labour became interchangeable and in some cases eliminated by automation [43].

Taylor's approach resulted in significant improvement in the production rate. Nevertheless, this resulted in many negative consequences in regard to the human aspects in the industry. A number of disadvantages of the *Tayloristic* approach were reported, including high turnover of employees, health issues and poor motivation [34].

2.2.2 Lean Production

The mass production era continued for more than half a century, with its peak around 1955, when six models from GM, Ford, and Chrysler accounted for 80% of all cars sold in the US. However, in 1970s the increasing intensity of competition, increase of consumer power and the decrease in the demand of

products due to a growing number of industries resulted in a need to optimise production resources by eliminating waste. The consequence was a paradigm shift from mass production to lean production [44].

The philosophy of lean production originated at Toyota Motor Inc. by Taiichi Ohno and Eiji Toyoda. After studying the mass production model of the Ford Motor Company, Taiichi Ohno and Eiji Toyoda came to the conclusion that the mass production model would never work in Japan. This was because of the smaller Japanese market, strong tradition of craftsmanship and the post World War starved economy that could not afford the Western technology. In contrast to the mass production model, a new philosophy was introduced that emphasised "*less of everything"* [40, 45].

Lean Production is essentially a management philosophy that targets the achievement of operational efficiency through elimination of non-value adding activities (muda, i.e. wasted effort, material and time) and perfect workflow in every area of production. The principles behind lean production included: produce only what is pulled by customer, perfect first-time quality, waste minimisation, reduced inventory, continuous improvement, flexibility, design for rapid changeover, empowering workers, building and maintaining a relationship with suppliers, load levelling and maximising production flow and visual control [40].

Lean production is now having a repercussion across all enterprise functions. According to a lean manufacturing survey, the lean production practice has proven instrumental. Only around 0.02% of respondents described it as not very important to the prosperity of their organisation [46]. According to this survey, UK manufacturers found lean principles beneficial in: improving efficiency, removing waste, reducing costs, reducing lead times, reducing inventory and reducing the workforce.

2.2.3 Agile Manufacturing

Since the 1990s, the business environment has started changing dramatically. The manufacturing sector is facing tremendous challenges due to the growing consumer influence and continuing globalisation. The results of this could be seen in the form of demand for product variety and reducing product lifecycles. In the past, the product lifecycles were long and demand was quite stable with a steady increase and then decrease at the beginning and the end of the lifecycle respectively. Today, the product demand climbs to its highest peak almost immediately after product launch. Shortly the demand starts decreasing momentarily but increases again to a second peak with promotion activities. A sudden reduction in the demand then occurs that leads to discontinuation of the product, mainly due to launch of a more competitive rival product [47]. In such a situation, due to the rigid configuration and limited flexibility, companies are now required to modify and renew their manufacturing systems much earlier than their useful life. This earlier obsolescence of the manufacturing systems is tremendously increasing the production cost and time to market [48].

In response to the challenges faced by industry, a study conducted at the Iacocca Institute at Lehigh

University in 1991, coined the term *agile manufacturing* [49]. The study emphasised on growing need for the ability to adapt quickly and profitably to continuous and unexpected changes in the manufacturing environment [50].

The agile manufacturing philosophy aims at addressing the responsiveness dimension that is not evident in lean manufacturing. Lean implies high productivity and quality. Whereas agile stresses the importance of being highly responsive to meet the customer demands while simultaneously striving to be lean [51].

Agile manufacturing is defined by a number of researchers in different ways from different perspectives. Nevertheless, all definitions of manufacturing agility insist on the capabilities of industry to quickly respond to changes. Some common definitions are given below.

- According to Gunasekaran agile manufacturing is *"the capability to survive and prosper in a competitive environment of continuous and unexpected change by reacting quickly and effectively to changing markets, driven by customer-designed products and services"* [52].
- Kidd defines manufacturing agility as *"the ability to thrive and prosper in a competitive environment of continuous and unanticipated change, to respond quickly to rapidly changing markets driven by customer-based valuing of products and services"* [53].
- Gupta and Mittal define it as a "*concept that integrates organisations, people and technology into a meaningful unit by deploying advanced information technologies and flexible and nimble organisations structures to support highly skilled, knowledgeable and motivated people"* [54].

The key enablers of agile manufacturing include virtual enterprise formation, distributed manufacturing architecture, rapid partnership formation, concurrent engineering, integrated product, production and business formation tools, rapid prototyping and electronic commerce [51].

It is worth noting that agile manufacturing is not only concerned with being flexible and responsive to current demands but also requires an adaptive capability to be able to respond to future changes. The concept of agility is often confused with flexibility. Though closely related, there are distinct differences between them. Agility is the ability of an enterprise to adapt to unpredicted changes in the external environment whilst flexibility is the ability of companies to respond to a variety of customer requirements that exist within defined constraints [55].

2.3 Need for Reconfigurable Manufacturing Systems

Since the birth of modern manufacturing, industries have gone under several paradigmatic shifts due to changing market conditions, economic trends, and technology. In response, manufacturing systems have evolved over time to fit the business requirements. In the early 20th century the lack of competition and the high-volume demand resulted in the demand of *dedicated manufacturing systems*

to attain a high production rate and tight tolerances [56]. In the latter half of the twentieth century, the regression of the industrial growth and changes in the customers' attitude demanded for high quality, low cost and greater product mix. During this period, *flexible manufacturing systems* became a central theme of competitiveness. The built-in flexibility to perform a variety of manufacturing processes was a promising way to produce small batches of mixed products.

Dedicated and flexible manufacturing systems are still dominant in the automotive industry. However, due to the fixed configuration and rigid structures, these systems cannot be easily modified to cope with changing production requirements and thus face a constant threat of obsolescence [16]. As a consequence, companies are required to modify or renew their production facilities much earlier than their lifetime. To address this failing it is necessary to reconsider current manufacturing systems and find solutions to aid their rapid design and reconfiguration to respond to the changing market requirements in an efficient manner [14].

Following the initiative of *agile manufacturing* by the Iacocca Institute, the concept of the *Reconfigurable Manufacturing Systems* (RMS) was introduced at the University of Michigan to address the need for agility within manufacturing systems. The aim of the concept was to enable cost effective and rapid system changes to "*exact functionality and capacity as needed and when needed*" [57]. This aimed to allow an RMS to evolve over time to changes in required functionality and production capacity with changing market demands.

Koren [58] defines RMS as a system that is "*designed at the outset for rapid changes in the structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family*". Such a system must have an open ended architecture and modularity that possess plug-and-play capability to allow the addition, removal or modification of machine hardware and software components with minimum effort [59]. The components may include sensors, actuators, conveyor systems and their underlying control algorithms.

According to Koren et al. and Bi et al. to achieve true reconfigurability, reconfigurable systems have six core characteristics: modularity, integrability, customised flexibility, scalability, convertibility, and diagnosability [58, 60]. A brief description of these characteristics is given in Table 2-1. Of these six characteristics, modularity and integrability are the most essential characteristics and sufficient to enable reconfiguration [44]. When possessing these characteristics, the responsiveness of a manufacturing system to unpredicted events increases, such as changes in product demand or unexpected machine breakdown.

Table 2-1 Core characteristics of RMS [44]

Attribute	Description
Customisation	The ability to apply a customised flexibility to production or inspection machines to meet new requirements within a part family
Scalability	The ability to efficiently change the machines' production throughput by altering or augmenting the components in the machine
Convertibility	The ability to efficiently redirect the functionality of the machine and it controls to suit new production requirements
Modularity	The compartmentalisation of operational functions and hardware into units that can be manipulated between alternate machine configurations
Integrability	The ability to integrate machine modules rapidly and precisely by a set of mechanical, informational, and control interfaces
Diagnosability	The capability of monitoring the current state of a machine and controls so as to detect and diagnose the root cause of output product defects

2.4 Manufacturing Automation Systems

Manufacturing automation systems refer to the technology that utilises control system to manage machines and processes to reduce the need for human intervention. Control systems are used to send commands to physical devices (such as actuators) to perform specific manufacturing tasks. These commands depend upon their software based control logic and the input data from physical devices such as sensors. Control systems are generally application-specific devices, so these are associated with many definitions depending on their applications. However with regard to automation systems in manufacturing industry, a control system can be broadly defined as "*a device or set of device(s), which can be used to manage, command, direct or regulate the behavior of other devices or systems"* (Lee, 2004).

2.4.1 Automations System Architecture

Industrial automation systems comprise of a hierarchy of levels with specific control functionalities at each level. These hierarchical levels based on ISA-95 reference model are shown in Figure 2-1 [61].

The corporate management level is the highest in the hierarchy. This level controls mainly managerial activities, such as human resource planning, manufacturing resource planning, sales and distribution.

The operations management level is primarily concerned about the scheduling and monitoring of the production facility such as manufacturing resource allocation, detailed scheduling and manufacturing data acquisition.

The control level is concerned with the actual control of production processes i.e. runtime control of

manufacturing systems. This level typically employs controllers and human machine interface devices to operate and monitor individual workstations. The Programmable Logic Controllers (PLCs) are often considered as the main work-horses at this level.

The field level is the lowest level is hierarchy. This level is comprised of devices (such as sensor and actuators) that perform basic data acquisition and manufacturing processes. They are typically connected with the PLCs via remote I/Os modules that communicate with the PLC via field area networks.

The scope of research in this thesis is limited to the lowest two levels of automation systems.

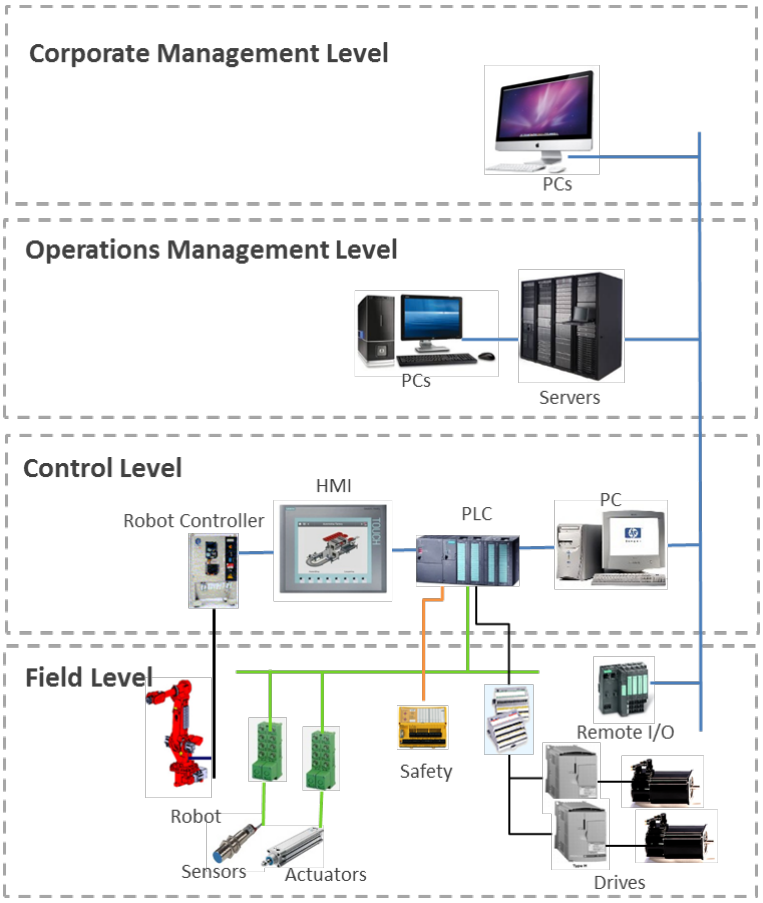


Figure 2-1 ISA-95 control architecture of factory floor automation systems

2.4.2 Example Shop-Floor Architecture of an Assembly Automation System

The typical shop-floor control architecture of an engine assembly line is shown in Figure 2-2. Engine assembly is a highly automated process within the automotive industry. An assembly line typically consists of a long S-shaped conveyor, divided into zones. Each zone consists of a number of workstations, built around the conveyor. Engine blocks are loaded on to pallets and transported from station to station. At each station a set of assembly operations are performed, such as nut running. Radio frequency identification (RFID) tags, also known as data-tags, are installed on the pallets for storing product and process information associated with the engine block.

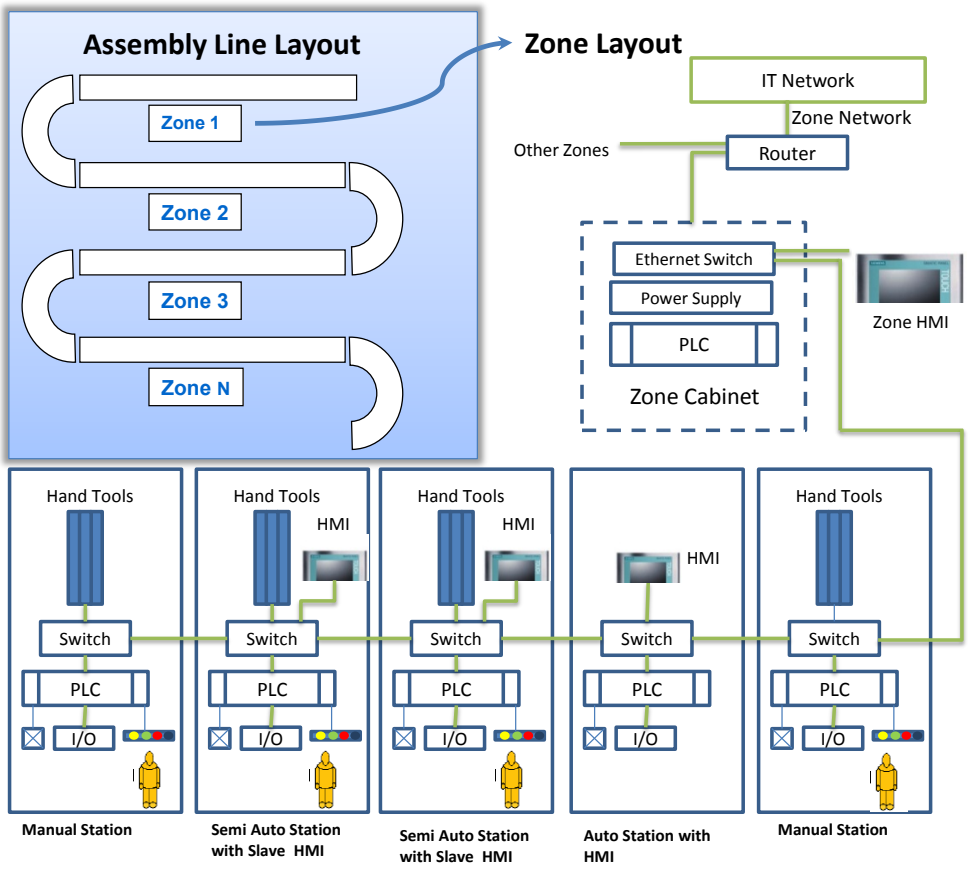


Figure 2-2 A typical layout of engine assembly line (Courtesy of Ford Motor Company, UK)

Typically, at each station, an engine pallet waits at a pre-stop. Once allowed to enter, the pallet then moves to the machine-stop. At the machine-stop the controller reads information from the data-tag to determine the operations to be carried out. The required operations are then performed and the information about the operations is written back to the data-tag to keep the history of the operations carried out.

On the basis of the degree of automation, stations are typically categorised into three types: automatic, semi-automatic, and manual. Automatic stations represent the highest degree of automation and do not require operators for their normal operation. On the other hand, manual stations represent the lowest degree of automation and an operator is required to perform operations manually. The operations performed at manual stations are often difficult to automate (such as spark plug assembly) or the labour cost is much lower than the cost to automate the process. For example, in countries such as India and China, semi-automatic and manual stations are most commonly used because of low labour cost.

An assembly line is controlled by a number of PLCs; typically known as Resources1. As shown in Figure 2-3, a Resource (PLC) can support a number of Areas of machine control; an Area can consist of a number of Stations; a Station is based on a number of processes performed by a number of mechanisms. An Area operates autonomously and has its own power supply but reports back to a

[1] The terms Resource, Area and Station are adopted from ThyssenKrupp System Engineering GmbH

centralised controller, zone PLC. Stations of an Area shares power supply but have their own operating mode control, dedicated HMI screens[1], RFID and stack lights.

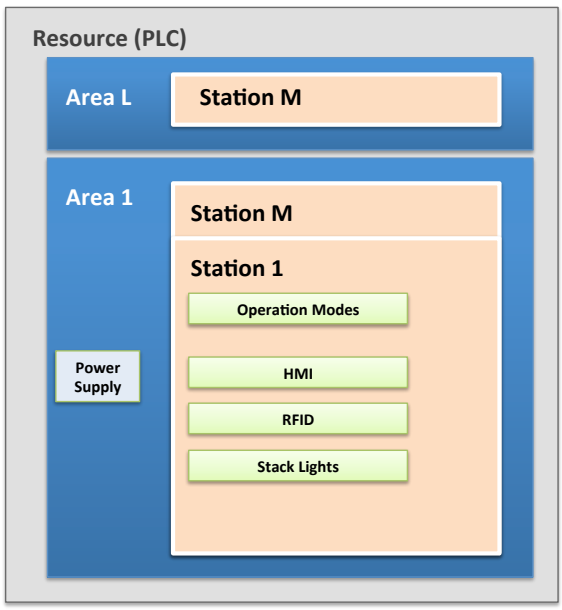


Figure 2-3 A typical architecture of machine control area controlled by a PLC (Courtesy of ThyssenKrupp System Engineering GmbH)

The purpose of the Resource PLCs is to coordinate all the machine operations in a sequential manner. The centralised controller, known as zone PLC, synchronizes the work between all stations for integrated plant operation. Semi-automatic and automatic stations have their own local HMIs, but can also be controlled from zone HMI. However, manual stations are typically controlled via hard pushbuttons and have no local HMIs. All PLCs, HMIs and remote I/Os are today usually interconnected via industrial Ethernet cables. Ethernet is also used for connections to the corporate IT network for communicating productivity information.

2.4.3 Automation Systems Lifecycle

The design and build of automation systems is one of the key competitive areas of automotive manufacturing. The lifecycle engineering of automation systems involves geographically distributed teams of end-user, machine builders and control vendors [62]. End-users are the automotive production companies. Machine builders, also known as Original Equipment Manufacturers (OEMs), are the tier-1 suppliers to the end-users. They are responsible for the design and build of manufacturing systems. Machine builders may also sub-contract some parts of the machine construction to specialist component builders or system designers. The control vendors are tier-2 suppliers and provide control hardware to the end-user. Collectively all these partners are responsible for the implementation and lifecycle support of the manufacturing systems [24].

[1] An HMI panel can host HMI screens for one or more Stations. The number of stations hosted by HMI depends on the complexity of Stations.

The lifecycle model of automation systems development within Ford Motor Company is shown in Figure 2-4. The lifecycle model consists of three primary phases: planning, realisation and installation & production. Each phase is composed of a number of sub-phases that are typically shared by many actors across organisations within the supply chain. A brief description of the activities within each phase is given below.

2.4.1 Planning

2.4.1.1 Study

The planning phase starts with a study, where the end-user defines the strategic intent of the programme, sets objectives, defines the sourcing strategy and creates a supplier shortlist.

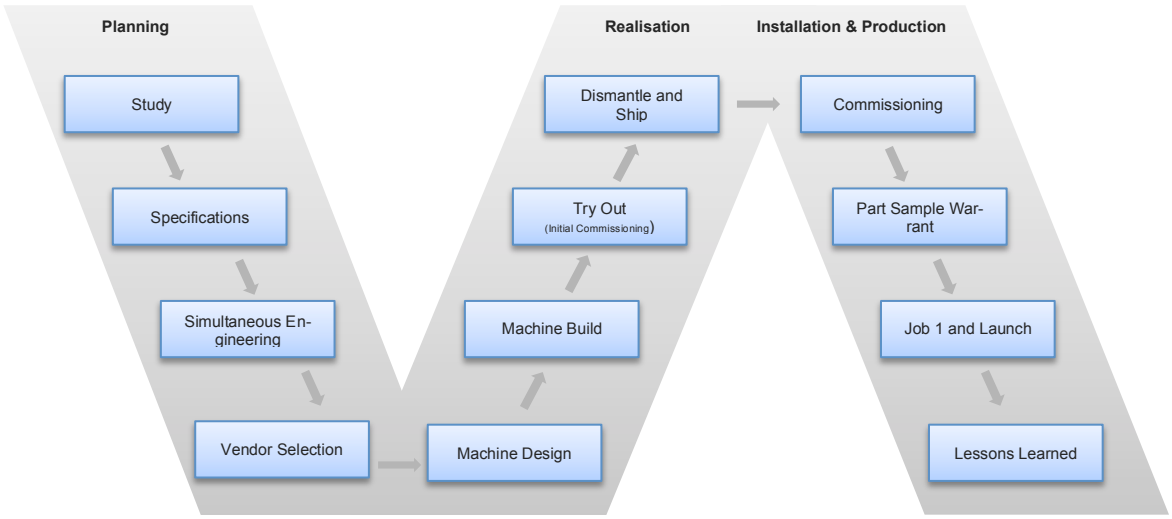


Figure 2-4 Lifecycle model of manufacturing automation systems development [63]

2.4.1.2 Specifications

The study phase is followed by the definition of product and manufacturing process specifications. At this phase, the controls vendors specify appropriate control technology and ensure the necessary control components are available to cover the scope of the programme. In addition, the work-plan and milestones are specified.

2.4.1.3 Simultaneous Engineering

A simultaneous engineering team typically consists of the end-user, control vendors and machine builder representatives. Simultaneous engineering enables identification of technical and economic issues associated with the new project at an early stage. During this stage, details such as the overall design, part lists, purchase lists, layout drawings are prepared.

2.4.1.4 Vendor Selection

The last phase of the planning is the machine builders and control vendors selection by the end-user.

The selection is typically based on the cost, technology, engineering solution and lifecycle support.

2.4.2 Realisation

2.4.2.1 Machine Design

Machine builders are responsible for the design of machines. The design stage starts with the mechanical design of the machine followed by electrical design and the design of the control system architecture. To reduce the time and cost the reuse of previous machine designs are widely considered. Typically CAD tools and virtual engineering tools are used to support these activities.

2.4.2.2 Machine Build

The machine build phase starts after the approval of the machine design by the end-user. Typically, the mechanical build, electrical cabinets build and the control software engineering are carried out in this phase. All the related activities of three engineering disciplines are carried out independently. Once the machine is assembled then the electrical wiring and installation of electrical cabinets are carried out. Finally, the control engineers download their program to the controllers and conduct the testing.

2.4.2.3 Commissioning at Machine Builder

This stage is also known as try out. At this stage end-user's engineers visit the machine builder site to test all the individual stations. Various checks are carried out to prove the machine robustness and production rate.

2.4.2.4 Dismantle and Ship

Once the machine is approved, the machine builder dismantles and ships the machine to the end-user site.

2.4.3 Installation and Production

2.4.3.1 Commissioning at End-User

This phase starts with the installation of the entire production line at the end-user site by machine builder engineers. The production line is integrated with the surrounding facilities, and sections of the system are tested one by one and a series of tests are conducted to validate the proper operation of the production line. The commissioning phase ends with the production of a first good product.

2.4.3.2 Part Sample Warrant

The commissioning phase is followed by part sample warrant. At this stage, the quality of the produced product is tested. The system remains under the inspection of machine builder to ensure that the machine is in stable working order. Usually, the line further undergoes modifications, optimisation of cycle time and further testing to achieve the desired product quality and the specified production

rate. Typically, production of 250 parts is carried out during this stage.

2.4.3.3 Job 1

Once the desired production rate and quality is achieved, Job 1 is announced. Job 1 denotes the official beginning of production. During production machines are monitored for their production rate to ensure that the production targets are being met.

2.4.3.4 Lessons Learned

After Job 1, a number of scheduled meetings are held for up to a year to formally identify and capture the lessons learned in the project. These meetings involve personnel from all of the supply chain partners. The information is documented and used to bring improvements to future projects.

2.5 Control System Engineering

2.5.1 Programming Languages

The control system programming is typically carried out in IEC 61131-3 languages. This is a worldwide recognised standard for the programming and configuration of industrial control devices and is used in most industrial applications [64]. IEC 61131-3 consists of three graphical and two textual languages. All of these languages can be used separately or in combination to form a complete application. The choice of programming language depends on the programmer's skill, the nature of the programming task, the level and structure of the problem, and the need for future modifications [65, 66]. A brief description of the IEC 6113-31 languages is given below.

Ladder Diagram

Ladder Diagram (LD) is based on graphical symbols laid out in networks in a similar way to the rungs of a relay ladder. A network represents the flow of the power from left to right between two rails. The elements contained by a network are contacts, timers, counters and logical blocks [67]. Ladder logic is a widely used language for sequence and interlock programs but has no inherent structure. The complexity of the program increase tremendously when used for designing large and complex systems [68].

Function Block Diagram

Function Block Diagram (FBD) provides a mechanism to enhance the reusability of code by encapsulating functionality in a black box with a common external interface [69]. It is used for programming complex procedures with graphical objects or blocks that represent functions, function blocks or programs, similar to electronic circuit diagrams. The direction of signal flow between function blocks is always from left to right, except in the feedback paths [67].

Sequential Function Chart

Sequential Function Chart (SFC) was derived from Grafcet, a graphical language based on a French national standard and itself an evolution of the Petri Net [70]. Unlike LD and FB, an SFC has inherent structure to better organise and visualise the control program flow [65]. A sequential function chart consists of two main elements, steps and transitions. A transition is fired when the step above it is active and the transition condition is true. A set of action is associated with each step and a condition is associated with each transition. Both conditions and actions can be programmed by using any of the IEC standard programming languages, including SFC itself [71].

Instruction List

Instruction List (IL) is a low level textual language, similar to the assembly code. This language corresponds to the programming technique that has traditionally been used in embedded controllers [71]. It consists of a chunk of text lines where each line describes an operation instruction. An instruction consists of an operator or a function and an operand. Labels are necessary in order to enable jumps in the program.

Structured Text

Structured Text (ST) is a high level textural language with syntax similar to Pascal and is designed to make PLCs more accessible to programmers familiar with traditional programming languages [69].

2.5.2 PLC Software Structure Standards

The ever-growing complexity and size of control software in automotive manufacturing has led to considerable research into structured programming methods. Today, almost every automobile manufacturer uses its own structured programming standard. However, all contain the same basic principles. The purpose of these standards is to ensure the consistency and quality of the programs. Automotive manufacturers force their machine builders to be compliant with their programming standard and often provide training courses to machine builders prior to writing control code for their machines.

These programming standards have tremendously helped to overcome many past problems such as consistent software quality from different vendors, standardised and efficient diagnostics, coding minimisation, reduced training costs, and easy program modification and reusability. However, these standards limit the flexibility of the programmer as the structure of the PLC code is determined by the end-user rather than the machine builder [69]. An overview of some of the structured programming methods used in Ford Motor Company is given below.

2.5.2.1 Error Diagnostic Dynamic Indication

Error Dynamic Diagnostic Indication (EDDI) is the first known structured programming method used

in the automotive industry. EDDI originated in the 1980's to address discrepancies in PLC control and diagnostic code. Traditionally, sequential control code and diagnostics code were implemented as separate entities. It was a common practice to add the diagnostic part of the code at the end of the programming process. As the diagnostic part was not integrated with the machine control code, this often led to inaccurate and misleading diagnostic messages being presented to the operator. In addition, it was a very common mistake to modify the machine control code without updating the diagnostic code [72].

EDDI was a European initiative led by engineers of Ford Motor Company from the body and assembly division. EDDI is essentially a design template and a set of directions to programmers and can be applied to a variety of PLC and PC based platforms. It incorporates the diagnostics by default as a result of the logic for controlling the machine. As a result, the diagnostics are accurate from the very start and remain aligned throughout the machine's lifecycle. EDDI is the most widely used vendor independent programming structure.

The Mondeo assembly project in 1989/90 at Genk, Belgium, was the first occasion when Ford Motor Company insisted that all OEMs must use Fords' specific programming structure for all machines in the plant. According to Ford engineers that was the most successful launch they had ever achieved till that time [73].

The EDDI philosophy pioneered a number of major achievements including [74]:

- First non-proprietary software structure for use with PLC systems,
- A mapped sequence making the process apparent to the operator,
- Fully integrated diagnostics, i.e. the diagnostics are integral part of the sequence control program,
- The realisation of manual diagnostic capability known as manual cross-interlock checking.

Over the last decade, Ford Motor Company has released a number of control software standards. However, the basic principles of these standards are still based on EDDI. Other examples of these standards are STEPS (Structured Transfer-Machine EDDI Programming System) and FAST (Ford And Siemens Transline). STEPS is the most widely accepted standard at Ford Motor Company and has been used for more than two decades in a number of plants across the world. STEPS consist of a control logic framework made up of ladder logic code and function blocks [68]. So far, numerous versions of STEPS have been released, such as Ladder STEPS and Function Block STEPS. FAST is the most recent control software standard and was released in 2013 for powertrain manufacturing operations [75, 76].

2.5.2.2 Zone Logic

Zone Logic originated as a result of the joint efforts of Septor Electronics and Lamb Technicon to

make the control system intelligent in response to unforeseen failures. Zone Logic is essentially a control scheme that could analyse the condition of a machine, automatically trap machine faults and compose error messages based on the allowable machine state matrix. This eliminates the need for a programmer to have anticipated the fault. In addition the system was able to indicate the actions available to the operator and give the reason why the other actions were inhibited.

Instead of programming the operation of devices in a conventional manner, the allowed conditions of each device are entered into the system. The operating system then continually compares the current conditions of each device to its possible valid conditions. In case of any non-matching condition an error message is automatically generated. Thus any condition that is not an allowed state of the device is an error state. This tremendously reduces the amount of coding when a programmer has to code each and every fault condition.

Several other features made Zone Logic an important milestone in machine control including: distributing the control to a station level, connecting the controllers via a fibre optic bus and integrating numerical motion control into the architecture were all innovative concepts at the time [77].

Unfortunately, Lamb Technicon's controlling interest in the system restricted the market with the other rival machine tool builders refusing to use the system. In June 1988, a subsidiary of Daimler-Benz gained a controlling interest in Septor which reduced appeal to the US automotive sector [74].

2.5.2.3 Function Oriented Modularity

Function Oriented Modularity (FOM) is a control software structure introduced by ThyssenKrupp System Engineering (TKSE) GmbH in 2007 for programming assembly automation systems. The purpose of the introduction of this structured programming is to enhance the reusability of the control code and avoid end-user specific standards by offering a common solution to their customers. The reuse of the control code is handled by encapsulating generic code in function blocks for a family of mechanisms. Instead of cutting and pasting sections of the code, a function block can be instantiated and configured as required.

The overall software structure is shown in Figure 2-5. Unit and Process Step are the basic building blocks of the software structure. Unit is the smallest and lowest level of enclosed working functionality and represents a single, or combination of, mechanical, electrical, and software elements. The software is encapsulated in a blackbox that can be parameterised. A Unit not only controls the behaviour of mechanisms but also includes integrated fault diagnostics and generates the required HMI screens for manual mode control. A Process Step is a function for defining sequence of operations of a specific task (such as nut running) and typically consists of one or more Units. A Process Step can also be used as an enclosed object, which reads the RFID data-tag at the start of an operation and writes the status back to the tag when the operation is complete. A Process Step consists of a number of sub-tasks (such as open clamp, close clamp) knows as 'Process Single Step'. All subtasks are coor-

dinated via a process coordinator and implemented in a combination of FBs and LD [78].

FOM has been used on production lines by a number of automotive companies such as Ford and Volvo. One clear advantage of FOM, from a programming perspective, is ease of use for the OEM. However, FOM acceptance amongst end users was mitigated. Feedback from end-user (i.e. Ford Motor Company) collected for the purpose of this research has shown that FOM function blocks are written to be generic (i.e. small number of FOM blocks can cover a wide variety of device control) and covers a wide range of functions. This practically results in: a) increased program scan time because of large number of lines of code and hence lowers performance and b) an excessive code complexity that makes it difficult to interpret and debug compared to more conventional programming structures based on LDs and SFCs.

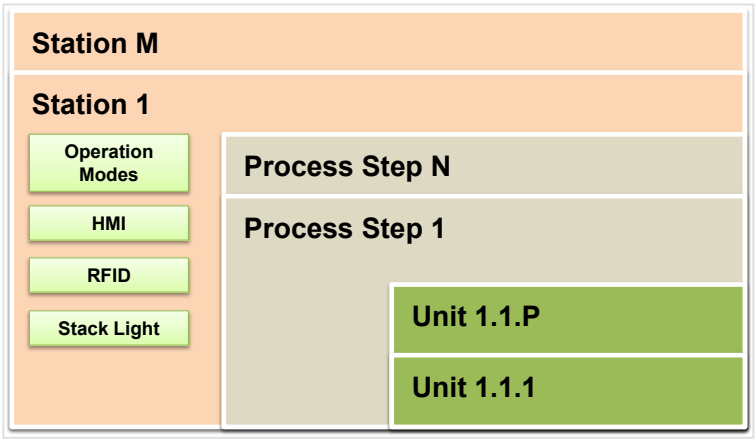


Figure 2-5 FOM software structure (Courtesy of ThyssenKrupp Krause System Engineering GmbH)

2.5.3 The Current Controls Software Development Practice

As witnessed in the automotive industry, the development of current automation system follows a classical sequential model, see Figure 2-6. The development of a system involves process engineering, mechanical engineering, electrical engineering and control software engineering processes in sequence. In the current workflow, control software engineering starts at a very late stage in the overall system engineering process and remains highly isolated from the mechanical design and build of the system.

The control software usually provides a wide range of control functions. The control logic must warrant that production is carried out by sequencing the required manufacturing operations in smallest cycle time. In addition to the nominal production behaviour of the machine, functions are provided to allow the operators to run the machine in various operating modes and cycles, such as automatic, manual, step, dry-run, to provide operators with full control of the machine in various situations. It is critical to provide the ability to restart the machine safely by returning to its initial position from all possible positions that the machine can have during operation without any human intervention. To ensure the safety of both human and machine, it is necessary to envisage and program machine

behaviour for every possible situation that could occur during machine operation. These include sensor failures, mechanical breakdown, power loss and operator mistakes. Typically, interlock logic is added to inhibit the movements of mechanisms to react in a safe manner in case of operator error or hardware faults. Finally, diagnostic code is needed to generate fault alarms. Text messages are usually associated with each alarm to clearly define the fault and the required corrective actions to get the machine back into production in the shortest possible time [79].

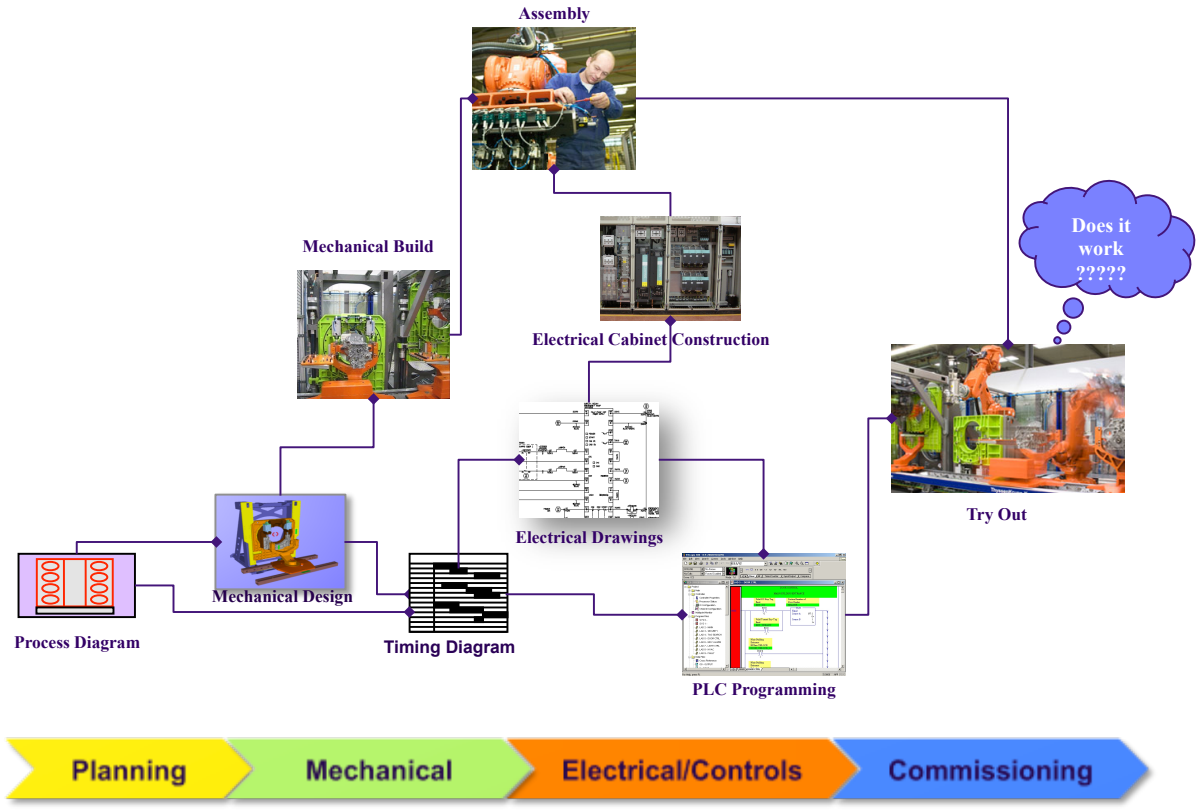


Figure 2-6 Overview of the current engineering workflow

A flow diagram of control software development process is shown in Figure 2-7. The end-user typically provides controls specification to the machine builders, which define the structure of the control software, the required control functionality and the control hardware. In order to write control logic, timing diagrams, assembly drawings and electrical drawings are usually provided to control engineers. Timing diagrams typically describe the sequential behaviour of a machine in a time-dependent manner. Assembly drawings are used to identify the positions of components (e.g., sensors and actuators) as well as schematics of the whole machine. Electrical drawings define the electrical wiring, the physical addresses of the components (such as sensors and actuators) and the hardware configuration. In addition to this, the control engineers typically have access to the control software from previous similar projects and will often reuse the relevant sections of this code.

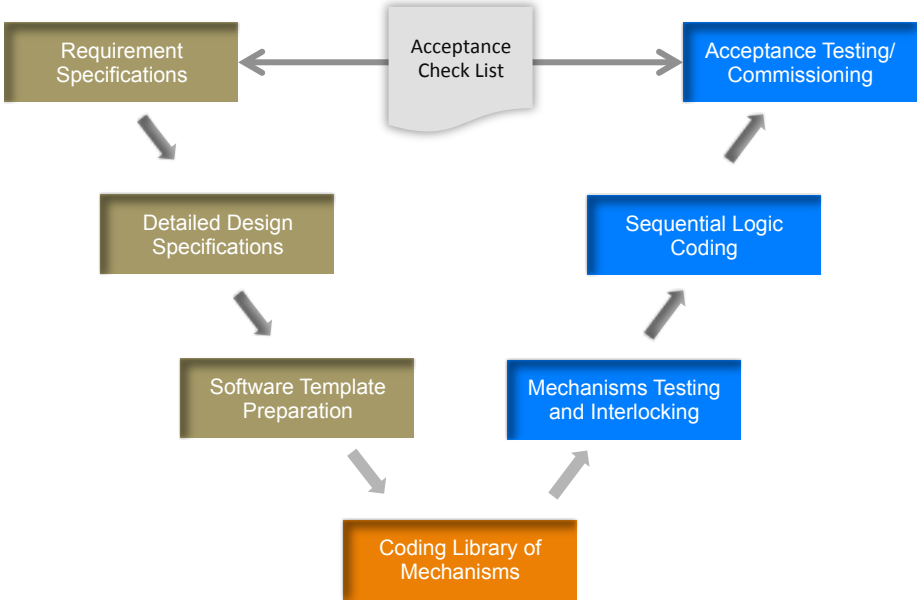


Figure 2-7 Control software development V-model

Coding of the control logic is typically achieved manually using proprietary engineering tools and is based on the interpretation of timing diagrams and process charts. This coding process requires extensive resources i.e. considerable time and skilled engineers, and is prone to misinterpretation and errors [18]. The control code is usually implemented in a combination of IEC 61131-3 languages. Typically the main structure is based on ladder logic and the sequence of operations is based on SFC or LL-based step logic with the mechanisms control based on LL, ST or IL encapsulated in FBs.

The software development starts with the preparation of a generic template usually based on software structure specifications provided by the end-user. The template determines the overall structure of the software, the required control functionality and the languages to be used for each segment of the program. The template also consists of some common functions such as control modes, electrical cabinet faults, and bus faults. After template preparation, the memory map is defined. The memory map is the allocation of the memory locations to the physical I/Os and data for program execution as well as for communication with other devices. Memory map management is usually considered to be the most painstaking, time-consuming and error prone task. The software is typically written in a modular way in small segments. The code for mechanisms is added one by one to the template and necessary interlocks and fault messages are added and tested. The code for mechanisms is encapsulated in Function Blocks (FBs), which are written and tested individually. FBs for mechanisms are written once and stored in a standard FB library for reuse. Finally, the logic for the automatic mode machine behaviour (i.e. sequence of operations) is added by interpreting the timing diagram. To operate and monitor the machine, an HMI is created in parallel to the PLC coding. The HMI design is usually based on user-defined template.

After the program has been written, it is important to verify that it works as intended before it is used in production. Currently, due to the lack of suitable tools and methods for doing so, testing and de-

bugging is not carried out until the machine is assembled and ready for commissioning. During commissioning the machine is operated in different ways to ensure the fail-safe behaviour. Sections of the machine are tested one by one and a series of predefined tests are conducted to validate the control behaviour [12]. The final stage of software testing is running the machine for a long period of time, known as dry run. The dry run is considered essential because some errors will only appear until the machine has run for a several thousand cycles. The commissioning process takes more time than the actual control software development. This is because of the unforeseen errors that will be discovered during commissioning, resulting in non-working machines, unmatched functionality, and sometimes even catastrophic failure [80].

2.5.4 Challenges in the Controls Engineering Approach

In the current business environment, manufacturing systems must be designed to adapt rapidly and reliably with changing market demands. The current approach to control system implementation whilst well established and based on well-proven methods, still follows a classical rigid sequential model and uses an ad-hoc collection of poorly integrated tools and mechanisms to take customer requirements and translate them into desired systems. From extensive literature review and discussions with engineers from ThyssenKrupp Krause, Ford Motor Company, and Schneider Electric, a number of limitations within the current engineering approach were identified. A brief description of these limitations is presented in this section. To align with the scope of this research, the discussion is mainly from the control software development perspective.

2.5.4.1 Lack of Integrated Engineering

The design and development of automation systems involves a number of engineering disciplines. In the current engineering approach, the engineering process for each discipline is carried out independently from the others in a sequential fashion. The engineering activities are performed using well established but department specific and proprietary engineering tools [81]. At each development stage, a set of specifications and information are documented, and passed onto other engineering departments further down the process chain as required. Due to the use of department specific tools, data types, formats and structure, information exchange often becomes a bottleneck. As a consequence, information exchange is largely paper-based and requires constant re-implementation at various stages to suit the individual requirements [20]. This re-implementation is typically carried out via manual translation and interpretation, which are highly prone to errors and costly in terms of time and resource utilisation.

2.5.4.2 Late Verification of Control Logic

The correctness of the control logic plays an important role in the proper functioning of manufacturing systems [82]. The current practice shows that the testing of the control logic is not possible until machines have been built. This is because of the lack of means to check the consistency

of mechanical, process, and control system designs collectively during the design stage. As a consequence, the control programs are usually only tested during commissioning when a machine has been constructed and all mechanical, electrical and control parts have been assembled.

According to Harrison and Colombo 80% of the software engineering is performed at machine-builder, with the remaining 20% carried at the end-user site [83]. Haq [48] reported that out of this 80% only 20% is validated during implementation while the remaining 80% validation is carried out during commissioning.

At the commissioning stage, a large number of errors and discrepancies are typically discovered. These results in often unforeseen delays (typically up to several days for an automatic station) that eventually lead to hold-ups in system delivery. It has been reported that the control software engineering accounts for a large number of errors that occur during commissioning and run-up phases of machine implementation [84-86].

Figure 2-8 shows the time consumed by the control software correction in the overall project duration. Typically, the control software engineering takes up to 60% of commissioning and accounts for up to 15% of time to deliver [23]. According to Harrison and Colombo [83] there is a pressing need to optimise the commissioning phase as it has a direct impact on the production ramp-up times. Projections show that on a typical European automotive engine production line installation project, a reduction of the ramp-up time of 50% could translates in a financial saving of €20 million.

Figure 2-8 Contribution of control software errors to project delay [23]

2.5.4.3 Lack of Reuse

Discussions with the machine builders during this research have shown that about 3/4 of automotive engine assembly projects are re-designs of previous similar projects because of the similarity of assembly operations and machine components. However, the re-design of existing systems requires major rework of the underlying control logic. Reuse of the control code is carried out via copy-paste of fragments of the control software from previous projects and altering them to fit the existing scenario. Such reuse is not only prone to errors and time consuming, but it also requires engineers to have knowledge of the legacy system in order to be able to effectively interpret and possibly rewrite

the control software of the legacy system.

Another common problem is the “not invented here” syndrome. The software developers often prefer to reuse solutions from previous projects developed by themselves and do not trust common solutions from other engineers [87]. Moreover, highly experienced engineers often leave companies without their knowledge being passed on to other relatively less experienced engineers.

2.5.4.4 Fragmented Control Software Development

Although structured programming techniques have the potential to increase the robustness of the process, the chances of errors are still high since the control functions, fault diagnostic and HMI screens are often treated as separate entities. Changes conducted in the control function require extensive re-work of the other related parts of the control code. It is a very common mistake that control engineers modify the control functionality of the machine but forget to conduct the corresponding modifications in diagnostic and HMI codes. This results in incorrect and misleading diagnostic messages being presented to operators. For this reason, the ability to view the control code at machines on the shop-floor is still considered as an essential feature by the end-users. In order to address such problems, high-level and process oriented programming techniques are required that focus on defining system behaviour with integrated diagnostic and screen generation rather than coding individual parts and checking for consistency at a later stage [69].

2.5.4.5 Lack of Reconfiguration

The objective of system reconfiguration at control software level is to enable the quick and cost-effective modification of a system from its current configuration to another configuration without being taken offline or disrupting production when changes are required or unpredictable events occur [58]. If a software component is not designed at the onset to be reconfigurable then any change in the system will require major rework to tailor to specific needs.

Currently, reconfiguration is usually achieved by manually modifying the control code offline. Any change in the control code also requires updating respective sections of the fault management and HMI systems. The change in the configuration often results in severe instability and disruptions in the production [88].

2.5.4.6 Lack of Interoperability of Control Programs

The choice of control hardware is often one of the most difficult decisions to be made during a new automation system project for end-users. End-users tie themselves to a specific control vendor at a project level. However, their choice of control vendor often changes in future projects for a number of reasons, such as the cost of developing control programs from machine builder, the cost of the hardware itself, the expertise of its technicians and global support from control vendor. This makes interoperability of the control code or vendor neutral control logic one of the important issues.

Because of vendor specific programming environments, interoperability of PLC programs does not exist i.e. a program written for one PLC cannot be executed on another PLC and thus results in a painstaking reusability of control code. In addition, due to entirely different programming environment and user-interfaces, even expert logic designers are often unable to use other systems and development tools efficiently [79].

2.6 Enabling Technologies for Reconfigurable Automation Systems

Current market constraints are forcing industry to rapidly develop and reconfigure manufacturing systems in order to reduce the delivery time of new products to the market. The limitations of the traditional engineering methods and tools and the increased complexity of manufacturing systems have made it difficult to develop reliable manufacturing systems that can be changed rapidly. In particular, unforeseen delays often occur during machine development and installation, which consequently hold-up the system delivery date. Even a small change in a system could drastically increase the lead-time [89], and thus can result in a major loss in revenue.

There is still significant potential to improve the efficiency of the engineering process by enabling the efficient reuse of existing solutions, dynamic reconfiguration of the control systems and providing a collaborative engineering platform to support seamless integration across various engineering disciplines. In the past decade, a number of technologies and methods have been introduced to realise this. Those that have had a significant impact on the system design and reconfiguration are discussed below.

2.6.1 Modular Mechatronic Engineering

Modularity is the application of the standardisation principle to create components that can be configured into a wide range of products. The principles of modularity have been discussed for decades and have remained mainly focused on the product design to satisfy the wide range of customer needs. However, the concept has recently gained great attention within manufacturing systems engineering due to the resulting reconfigurability and reusability the approach can offer, which are required in order to cope with frequent production systems' re-design and changes.

Modularity is a general concept applied in many areas. At the most abstract level, it is defined as the capability of a system's components to be separated and re-combined [90]. Baldwin and Clark [91] define a module as a unit in a large system that is functionally integrated but structurally independent. The prime purpose of making modular system architecture is to enable heterogeneous inputs to be re-combined into a variety of heterogeneous configurations. Thus, a modular architecture allows system's components to be disaggregated and recombined into new configurations with minimum loss of functionality [92].

In the context of manufacturing systems, modularity allows manufacturing systems to be decomposed

into functional sub-systems, which interact with each other to perform a task. In the past, the functionality of a system has been determined to a large extent by the mechanical design, and thus mechanical aspects of a system are mainly considered during decomposition of a machine into modules while the control system has been given little attention. However, to enable effective reconfiguration, a mechatronic-oriented approach is required, which requires that all three facets of an automation system (i.e. mechanical, electrical and control software) should be decomposed and integrated at the same level of granularity in order to achieve encapsulation of complete mechatronic functionality.

Conforming to the above, a module in a manufacturing system can be defined as a generic reusable and reconfigurable mechatronic device consisting of mechanical, electrical and control elements with a well-specified interface that carries out a specific process-oriented function. A module may be used alone or combined with other modules to perform a specific manufacturing process. This definition of modularity is complemented by the Figure 2-9 [93].

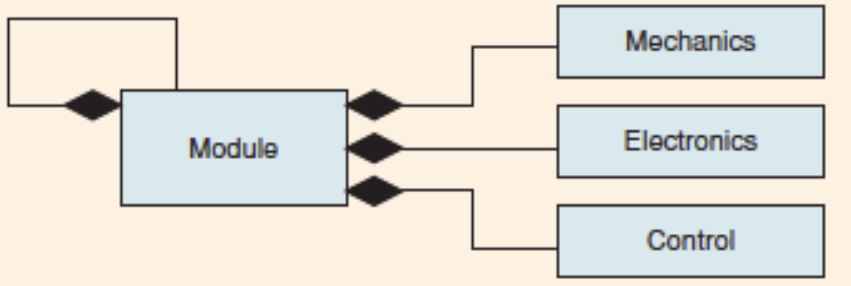


Figure 2-9 UML representation of mechatronic module, adopted from [93]

In modular systems, the most important characteristics are the interface and the granularity level of a module. To achieve true reusability and reconfigurability the interface should be standardised and universal. Standardisation of interfacing implies that all three interfaces (i.e. mechanical, electrical and control software) must possess plug and play capabilities [93]. This enables quick changes in the structure of a machine to allow alternative functionality or change in capacity by simply adding or removing modules without affecting the functionality of other modules [94].

Figure 2-10 portrays how different types of manufacturing systems can be developed using standard mechatronic modules on the basis of product requirements. This allows manufacturers to buy a simpler machine that can be altered with changing requirements rather than investing in highly complex and expensive general-purpose machines. A typical example of modular systems is the Modular Production Systems (MPS) manufactured by Festo. The modular stations of an MPS can be easily arranged in various combinations to significantly change the desired operations [2].

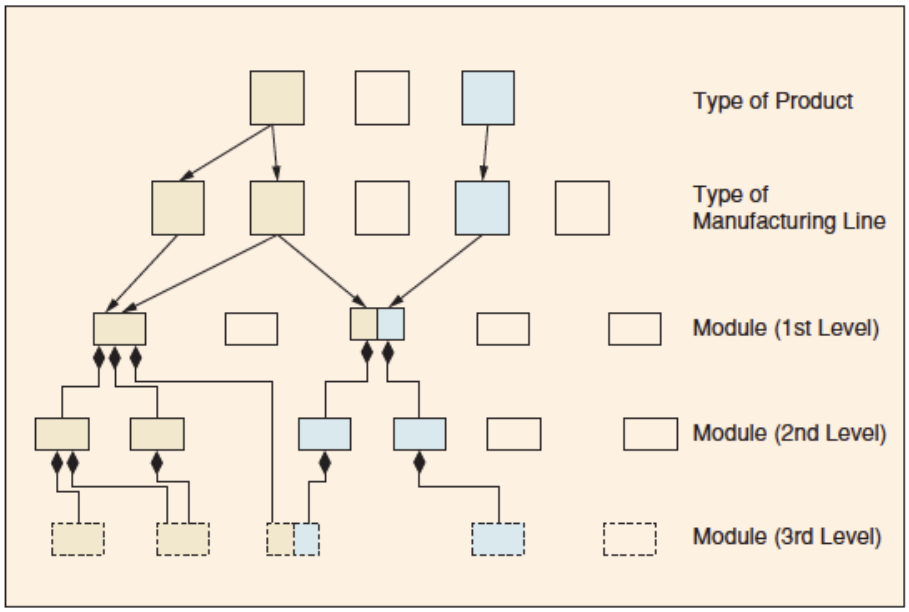


Figure 2-10 Development of manufacturing systems from mechatronic modules, adopted from [93]

In addition, it is important to carefully investigate how much functionality a given module should provide. A pragmatic approach is to create a system of coarse granularity that still offers the ability to provide the necessary system variants and avoid making integration over complicated. Correct modularity can make system complexity easier to manage [9, 16]. Figure 2-11 illustrates the trade-offs of selecting the granularity level.

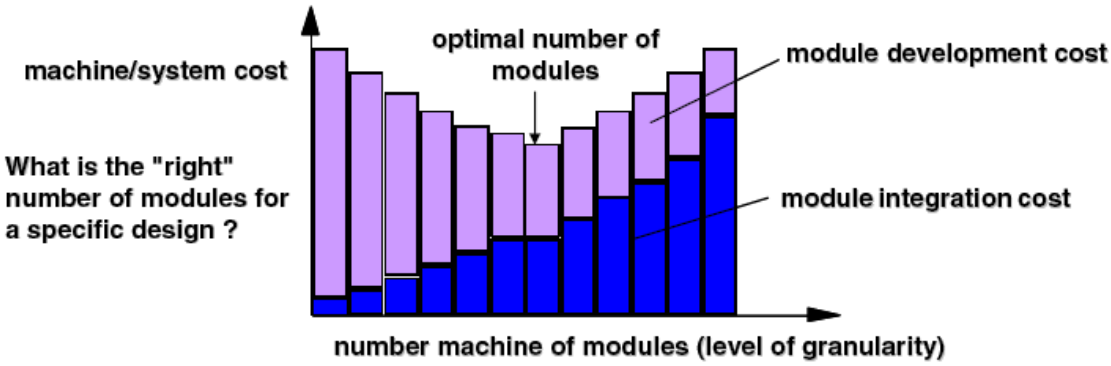


Figure 2-11 Comparison of trade-off between cost and the granularity of modular systems, cited from [16]

Unlike conventional dedicated or customised machines, modular systems offer several benefits. The advantages of modular approaches have been reported by a number of researchers in the literature. These include reusability, re-configurability, pre-testability, reduced development time and better forecast of production cost and lead-time [2, 83, 93, 95]. However, due to the high optimisation of mechanical configuration for a specific production process, the productivity of traditional standard machines tends to be higher than that of modular systems [93]. This is mainly due to finely adjusted mechanical configuration of traditional machines.

2.6.1.1 Existing Modularity Approaches

In the past two decades, extensive research has been conducted to enable the reconfigurability of manufacturing systems using modular approaches. A number of modular system design approaches can be found in the literature to match the control system modularity with the mechanical modularity of a machine. The research approaches either fall in the category of utilising intelligent system design tools to reduce the manual programming tasks or deploying intelligent control software units that can accommodate changes when change in system is required.

Multi-agent systems have been recognised as a promising paradigm for implementing distributed and reconfigurable automation control systems [96]. The agent-based approach in manufacturing control derives from the Distributed Artificial Intelligence (DAI) field, being characterised by decentralisation and parallel execution of activities based on autonomous entities, called agents. An agent is typically defined as an autonomous component that represents physical or logical objects in a manufacturing system, capable of acting in order to achieve its goals, and being able to interact with other agents, when it does not possess the knowledge and skills to reach alone its objectives (i.e. requires collaboration). An agent can represent physical manufacturing resources and logical objects. The application of agent technology in the manufacturing field has been carried out by several research teams, in different application domains, such as enterprise integration and manufacturing planning and control [97].

To ensure real-time responsiveness and extend the agent-based approach to field-level control, a so called holonic agent architecture has been introduced. At the lowest level of manufacturing control, the agents are typically known as holons. This holonic vision has resulted in the development of the IEC 61499 standard. This standard represents the key results of the HMS initiative and has a great potential to enable dynamic reconfiguration of runtime control [98]. A holon is a compound object embedded with a low-level control (LLC) part that processes the real-time data from sensors and actuators and a high-level control (HLC) part that coordinates the manufacturing tasks. LLC is implemented in languages of the IEC 61131-3 or IEC 61499 standards for the programming of PLCs [98-100].

A number of prototypes have been developed and reported in the literature. Most of these approaches are however aimed at addressing manufacturing execution systems. On the other hand, modular approaches that focus on low-level logic control are rare. Of these approaches, the ADACOR, RIMACS and COMPAG research projects are briefly presented below.

ADACOR (ADAaptive holonic Control aRchitecture for distributed manufacturing systems) has been researched at Ploytechnic Institute of Braganca. ADACOR aimed to provide modularity, decentralisation, autonomy, scalability, and re-use of manufacturing resources. The ADACOR architecture is built upon autonomous and cooperative units, known as holons. The term holon refers to an identifiable part of a manufacturing system that has a unique identity, yet is made up of sub-ordinate parts. An

ADACOR holon comprises of the physical resource and the logical control device needed to perform a manufacturing task autonomously. The logical control device is composed of three components: decision, communication and physical interface [101].

RIMACS, a European FP6 research project, proposed a collaborative automation paradigm based on an autonomous and modular component-based approach for flexible and agile manufacturing systems to enable mass customisation with reduced lead-time. The RIMACS approach uses open architecture standards, modular mechatronics devices and virtual engineering environments. The approach considers the set of production entities as a conglomerate of distributed, autonomous, intelligent and reusable units, which operate as a set of cooperating entities at production runtime. Each entity is typically constituted from hardware and control software with embedded intelligence (such as fault diagnostics) and provides common communication capabilities [94].

The COMponent-based Paradigm for AGile Automation (COMPAG) developed at Loughborough University allows an automation system to be decomposed into a set of distributed control components. The control components comprise actuators and sensors that contain embedded sequence and interlock capabilities. The control functionality can be constructed to match the physical modularity of the machine. The component-based paradigm can eliminate the traditional centralised PLC controller, and the system is not programmed using conventional relay ladder logic [16, 102, 103]. The control software is embedded into the individual components supporting the control behaviour, error checking, diagnostics and lifecycle data acquisition. The application logic that defines the desired state behaviour for a specific manufacturing resource is configured via parameters rather than reprogramming. The machine configuration and application logic is designed and generated via 3D virtual process engineering tools. It should be noted that the COMPAG project focused on low-level realtime machine control systems that exhibit predominantly reactive behaviour rather than proactive high-level agent-based systems [83].

2.6.2 Virtual Commissioning

In the past decade, virtual engineering has gained a great attention and is recognised as a major driver of productivity and competitiveness in a number of engineering domains. The use of the virtual engineering tools allows the continuous planning of changes in product, process and resources in a virtual environment, thus eliminating the need for physical prototyping and testing. As a consequence, development time can be radically reduced and several design alternatives can be tested. Such capabilities of the VE system are very valuable in automotive and aerospace industries, where the physical models are expensive and require long development times.

A recent trend in the automation systems engineering domain is the use of virtual engineering environments to virtual prototype manufacturing systems including 3D CAD of the systems. This is often referred to as virtual commissioning or virtual manufacturing. Virtual engineering tools

typically provide a digital collaborative engineering environment in which mechanical design, process engineering and control engineering can be integrated into a three-dimensional dynamic model of a manufacturing system [80]. As a result, the engineering process becomes concurrent because of valuable intra-discipline collaboration and integration between mechanical, process, electrical and control engineering over the lifecycle of manufacturing systems, see Figure 2-12. In particular, it allows control engineers to work closely with mechanical engineers to validate and optimise the control behaviour at an early stage [30].

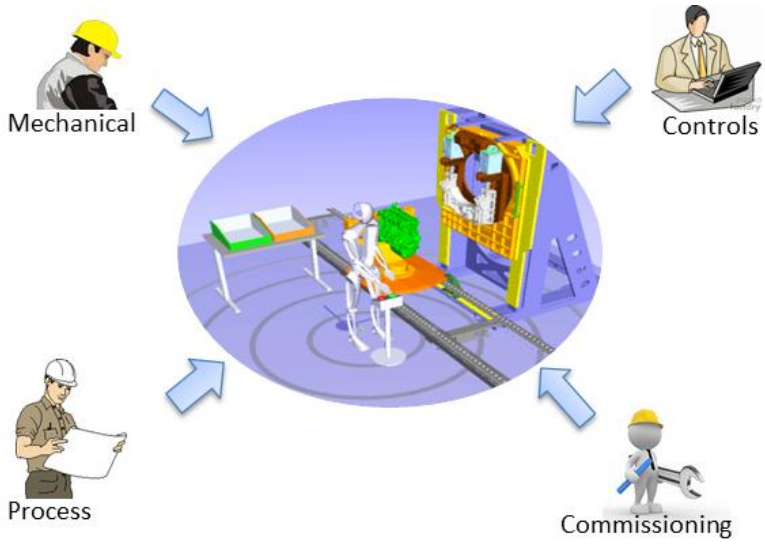


Figure 2-12 Virtual collaborative engineering environment

Like the application of the CAD/CAM technologies in product design, virtual prototyping of manufacturing systems has similar effects on the design and development of manufacturing systems. With the help of virtual prototyping tools, manufacturing systems can be visualised, optimised and validated before the physical build, thus ensuring "*the right first time"* build of automation systems. Once a machine is virtually built, the validation of a system can be carried out via visual inspection of a 3D CAD model of a machine, which is executed against the actual control logic [6]. A number of designs, configurations, and "what-if?" scenarios can be easily simulated that are otherwise difficult and time consuming if performed on the shop-floor [104].

A typical workflow of virtual commissioning is shown in Figure 2-13. The workflow is composed of four stages: virtual modelling and process planning, PLC and HMI programming, OPC (Open Platform Communications) link creation and testing. The virtual modelling and process planning involves the use of 3D CAD models of manufacturing cells and humans to mock-up the plant layout, production lines and manufacturing cells. The 3D models are typically composed of pre-developed and reusable standard mechatronic units. These mechatronic units consist of geometry, kinematic and control behaviour. Once the machine is modelled then the sequence of operations is defined to simulate the manufacturing processes in a required sequential manner. The simulation of a 3D model allows visual inspection to analyse component assembly, mechanical clashes and validate the

systems' behaviour. At this stage the sequence of operations and the position of actuators and sensors can be edited to adjust the mechanical inconsistencies and optimise manufacturing processes and cycle time.

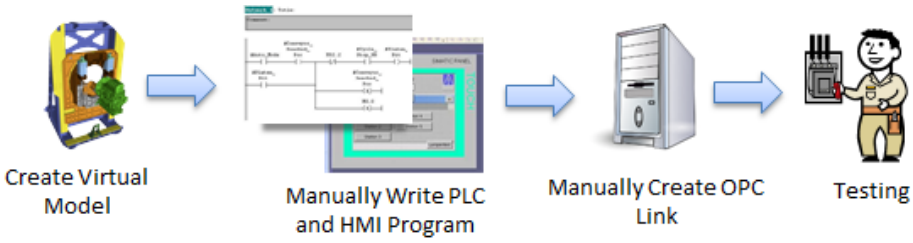


Figure 2-13 Virtual commissioning workflow

In process planning the intent is only to simulate the manufacturing process and therefore the focus is a sequence of operations. The sequence of operations is typically defined at a higher level of abstraction and often lacks the requirements of runtime controls engineering (such as safety interlocks). Such generalised control logic definition allows the testing of runtime behaviour only on a pseudo-code basis.

The second stage is offline-programming. Offline-programming allows testing of the actual control code against a virtual 3D model of the system to validate the runtime control behaviour. At this stage PLC program is manually written in the vendor specific programming tools (as described in section 2.5). The program is then downloaded into the real control hardware (or soft-controller) and connected to the virtual model of the manufacturing cell via OPC (Object linking and embedding for Process Control) link. The OPC enables the communication of the I/O signals between the virtual modelling environment and the PLC. The virtual model of the cell is then operated using the actual runtime controller thus enabling the testing of a cell under more realistic conditions [23], as shown in Figure 2-14.

Figure 2-14 Virtual commissioning of a manufacturing cell

Using the offline programming approach, machine behaviour can be thoroughly tested to check the robustness of the control code. This provides the opportunity for controls engineers to debug and optimise the control logic before the physical build. Testing of actual control code in a virtual environment offers a number of significant benefits including [105, 106]:

- Efficient control code debugging,
- Reduced commissioning time and accelerated ramp-up,
- Decreased downtime during production due to validated control logic,
- Testing without risking both man and machine, and
- Operator training before physical build

Commissioning of manufacturing systems in a virtual environment does not eliminate the need for real commissioning [107] but creates a new parallel process to conduct planning, validation and optimisation outside the project critical path. This removes much of time pressure that exists in the classical sequential approach [30]. As a consequence, the commissioning time is significantly compressed by identifying structural defects and verifying the control behaviour in the early stage of a machine build process [21, 29]. Figure 2-15 compares the classical machine build process with the one that involve virtual commissioning. As virtual commissioning is able to move a number of commissioning activities from the project critical path, a significant reduction in the overall project lead-time can be achieved.

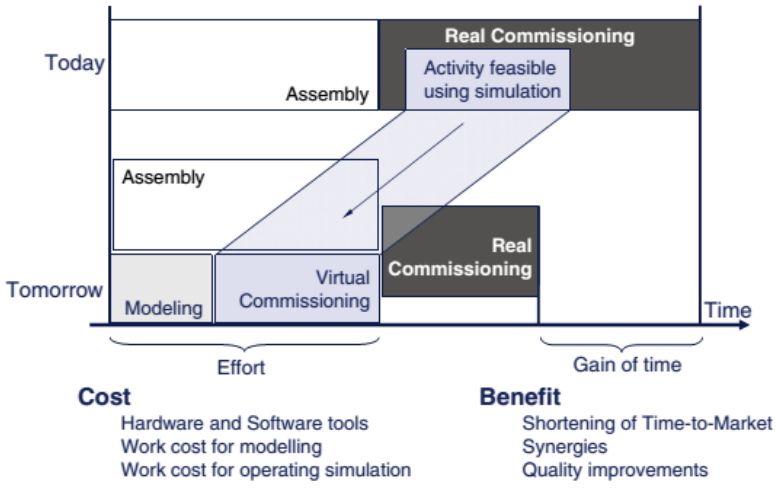


Figure 2-15 Impact of virtual commissioning on the machine development time [23]

In academia, virtual manufacturing has been acknowledged for more than a decade as an approach for the identification of the design flaws and inter-domain problems. Recently industry has also recognised the significance of the virtual manufacturing solutions. Especially in the automotive industry, virtual commissioning is becoming more common in a number of production areas, such as body-in-white and engine assembly. According to a study conducted by Reinhart [23] the

commissioning time was reduced to 75% using virtual commissioning. David [30] reported the use of virtual commissioning in a number of robotic welding and assembly operations that results in a saving of two to three man-weeks in a project lead-time. One of the main hindrances of a wide acceptance of VE in the industry is the additional effort of creating the simulation models. However, the use of pre-defined library components can significantly reduce the model development time.

2.6.3 Automatic Logic Generation

Automatic logic generation refers to the information reuse and transformation of machine control behaviour data defined at a higher level of abstraction into executable control code. Automatic logic generation tools aim to make the programming of the control systems intuitive so that the control behaviour can be defined at higher level of abstraction during the process planning phase without considering the complicated low-level control code [68]. Depending on the source of information to generate the control programs, the existing research work can be generally categorised into 3D virtual engineering based automatic logic generation and formal modelling languages based automatic logic generation. The research efforts focusing on logic generation using formal modelling languages are reported in section 2.6.4.

To enable seamless integration between mechanical and controls engineering, the concept of automatic logic generation based on the 3D virtual engineering has emerged in the past decade. The virtual machine models used for virtual commissioning have embedded control behaviour; therefore, the same data can potentially be reused and converted into control code [20]. This can not only reduce the efforts to manually write the control code but can also ensure consistency in the structure and quality of the control software [31]. As the control code for both the HMI and PLC could be generated from the same model, discrepancies between HMI and PLC programming can also be avoided. Automatic logic generation based on 3D virtual models is considered as a very promising way to significantly compress the development and commissioning time of control programs.

Automatic generation of programs from virtual engineering tools is a relatively novel research area and very little implementation work can be found in literature. Bergert [32] has presented a framework for the automatic generation of PLC programs from digital process information extracted from manufacturing cells modelled in DELMIA Process Engineer. In the case study presented, a cell-specific process plan is developed that consists of all process information related to a cell, such as human, robots, and PLC driven activities. To generate a PLC program, the process plan is filtered for PLC-relevant information to remove the data which is not required for PLC program generation. The filtered data is converted into an SFC for Unity Pro. Within Unity Pro, the SFC is then connected to resource-specific library function blocks. The resource-specific function blocks describe the behaviour of the manufacturing resources and contain all I/O signals from the resource. The research presented by Bergert is however mostly conceptual and the implementation is very limited [32]. Manual editing of the generated program is required to map SFCs with resource-specific function

blocks to make the program deployable. In addition, the work mainly discusses the automatic cycle and does not cover diagnostics, mode control, and integration with the HMI. These usually form a large part of the control code.

Researchers at Chalmers University have presented a framework to enable reuse of information from the mechanical design of a manufacturing cell to generate control programs. The framework identifies the part of the control program that can be generated by using control information extracted from robot simulation and design of a cell. The control information is synthesised using formal methods (such as Petri Net modelling) to ensure the work in the cell is well coordinated and no operational condition is violated. The control information is then used together with standard components to generate the control program. The components are essentially function blocks, which represent the devices and the basic cell functions. These component blocks are instantiated from a standard library. The concept mainly focuses the automatic control and does not include other parts of control programming, such as manual mode control, HMI integration, and integration with business execution systems. The details of this work can be found in [6, 7, 108, 109].

Steinegger [110] presented a general paradigm for the generation of PLC program by integrating related data from manufacturing process simulation tools. However, the proposed method is conceptual and no further practical solution has as yet been presented yet.

A number of PLC vendors have also launched tools to generate executable PLC programs from virtual engineering tools. Two such tools are discussed below:

SIMATIC Automation Designer

SIMATIC Automation Designer builds on the Siemens Tecnomatix tool, Process Simulate, to enable the reuse of information from the planning phase CAD design to develop the control software; thus integrating the real and digital factory. It allows integrated engineering of the mechanical, electrical, and control aspects of a component and enables modular configuration of a system [111].

Automation Designer includes tools for the automatic generation of PLC code for Siemens Step7 and HMI screens for Siemens WinCC Flexible. The PLC code and HMI screen generation is essentially based on the use of standard templates. A template in Automation Designer represents a real world object, such as a robot, and contains information about the object, such as hardware information, PLC code, and HMI screens. The templates for the generation of the PLC programs can either be written inside Automation Designer or be imported from a STEP7 library.

Enterprise Controls and RS TestStand

Enterprise Controls and RS TestStand, from Rockwell Automation, were designed to improve the efficiency of the logic development process in the automotive industry. The concept was essentially based on visual verification of the manufacturing process and reusable control libraries. RS TestStand and Enterprise Controls do not use common database, therefore an application has to be written twice,

i.e. once for the virtual model and again for code generation.

RS TestStand allows simulation of the behaviour of a machine in a virtual environment using animation elements or importing CAD models. Once verified, the logic can then be developed in Enterprise Control by creating device templates to control a particular class of mechanisms. Each device template includes integrated HMI generation, error handling and diagnostic capabilities. Once tested, these generic templates are then stored in a library for re-use and automatically translated into ladder logic code.

A control application is prepared by creating a required sequence of operations. The sequence of operation calls relevant actions predefined in the device template. The inputs and outputs can be associated with physical inputs and outputs or a virtual model created in RS TestStand. Once the application definition is complete, the control code is automatically generated for an Allen-Bradley HMI and PLC, which can be connected back to the virtual model in the RS TestStand via OPC for virtual verification.

Pilot demonstrator projects at Loughborough University, University of Michigan, and the University of Warwick have shown the potential benefits and limitations of the Rockwell software suite. The details can be found in [68, 70, 112]. Enterprise controls did not generate sufficient attention from industry and was eventually discontinued in 2007.

eM-PLC

eM-PLC aimed to provide an integrated virtual environment to streamline the engineering process and to provide a seamless path from process design to shop-floor automation. eM-PLC provided functions to import CAD models, assemble them to make components and define their kinematic and control behaviour. The sequence of operations is defined in a Gantt chart. eM-PLC is integrated with Siemens Step 7 Professional and can automatically create Step 7 project mainly consisting of a number of SFCs [113]. The SFCs can be tested against an in-house simulated PLC, and verified using the virtual plant model. The PLC program can also be tested using a real PLC and HMI against the virtual plant via open connectivity (OPC) [68]. eM-PLC has been discontinued and replaced by Automation Designer from Siemens following the takeover of UGS by Siemens in 2007.

2.6.4 Formal Methods in Controls Engineering

Formal methods refer to the mathematical reasoning about system model properties [6]. The complexity of programming and verifying large systems has resulted in interest in the possibility of using formal modelling and analysis techniques [19, 112]. The key benefits of formal methods in controls engineering are to define the control logic in a graphical way at a higher level of abstraction and then to authenticate the control code by performing mathematical analysis, e.g., to check stability, reachability and deadlock of a controller over all possible operating conditions. These methods

typically comprise the formalisation of informal specifications (such as timing diagrams, sequence charts and interlocks) followed by automatic synthesis and implementation of the PLC code [114]. The most common languages used for formal modelling are Petri Net (PN) and finite state automata [114].

The use of PN gained the interest of many academic researchers as a potential tool for designing PLC programs [68]. A Petri Net is essentially a graphical method of defining discrete event systems, consisting of places, tokens, transition, and arcs. A number of PN-based methods have been reported. Uzam et al. [115] proposed the use of Petri Net to synthesise a supervisor. This supervisor can be converted to Ladder Logic via a token passing logic controller. Frey presented Signal Interpreted Petri Nets (SIPN) to model a controller using a graphical description. The control algorithm was then verified and translated into IEC standard PLC code written in Instruction List. Feldmann et al. presented ordered colored PN to design and implement logic control for PLCs. The approach allowed combination of advantages of formal validation and as well as traditional PLC programming for the development of PLC control code. The PN model is then automatically converted into standard IEC 61131-3 code [116].

The collective view of researchers about the use of Petri Nets seems to be divided. Some researchers, such as Lee [117], refer to PN as a flexible method, easier to use than Ladder Logic. This statement is based on comparing the number of logical elements or conditions in Ladder Logic and PN programs. However, Ljungkrantz [113] states that the number of logical conditions and elements does not reflect the work required to configure a control system. Hajarnavis [118] states that such comparison method is “questionable and not fair”. Industrial practitioners have shown very little enthusiasm for the use of Petri Net [118].

Finite state automata have been considered by many researchers to model and analyse manufacturing systems. However, finite state automata (as well as Petri Nets) suffer from the problem of state explosion when the reachability analysis is conducted for a complex system with a large number of reachable states. To avoid state explosion, Endsley et al. [119] used an extension to finite automata called modular finite state machines to generate a verifiable controller. The control system is divided into modules. From the modules control behaviour can be built and verified. However, the control behaviour is not translated into standardised IEC language. Thapa [19, 89, 120] presented a model-based architecture. The presented approach is very similar to the SIPN approach. A system is modelled using formalism using timed-MPSG (Message-based Part State Graph), an extended version of finite state automata. The model is then converted into textual specification for formal verification using a model checker tool (SMV). The formal model of the system can also be interfaced with a 3D model based simulator for validation. The simulator matches the formal model with the corresponding 3D model and then executes the motion in the virtual environment to validate the system. After validation, the input and outputs of the formal model are then mapped to the I/O addresses and

executable PLC code is generated for Siemens STEP7. Before downloading into a PLC, the program requires minor manual modifications.

The use of formal methods has received considerable attention from academia, however there has been very little interest has been shown by industry. As a consequence, these methods are mainly still confined to research laboratories [121]. The modelling complexity and non-familiarity of the modelling languages to control engineers is one of the main reasons for the lack of interest in formal methods from industry. According to Thapa et al. [19] the PN approach is new to control engineers and technicians and thus does not fit well within the current engineering practices. Logic design using PN is quite different and complex when compared to the existing approaches used in industry. For example, enabling/firing of transitions can be quite a cumbersome task. According to Danielsson et al. [108] formal methods require users to learn new skills (such as new modelling languages and computer programming methods), which are complex when compared to conventional PLC programming methods. Some researchers have also developed tools for the formalisation of existing IEC 61131 PLC code but this still requires the user to learn new languages and tools for the specification development [122]. However, Lucas argues that the benefits of these new methods over the current practice have not been well demonstrated [123].

2.7 Review and Discussion

Due to a high degree of automation in the automotive industry, control systems are critical to the operations of their production lines. However, due to highly rigid and fragmented process of machine development, control engineering remains highly isolated from mechanical design and starts at a very late stage of the engineering process. Control engineers typically rely on paper-based machine specifications and use proprietary engineering tools. The control logic is typically written by interpreting timing diagrams and process charts. As the complexity and the size of the production machines increases, the task of writing control code becomes difficult and relies heavily on the experience of control engineers. The control codes developed for such systems are often monolithic and unstructured, making them difficult to understand, modify, maintain and reuse. Due to this, alterations to the automation system software are potentially time consuming, complex, error prone and expensive. In addition to this, the unavailability of tools to verify control programs further reduces the reliability of the whole process. As a consequence, a large number of errors are only detected during commissioning. This results in a long ramp-up phase and hence leads to a loss of potential revenue.

As industries are trying to adopt changeable and reconfigurable systems, the limitations in the conventional practice of logic development for programmable controllers are becoming more apparent. To overcome these limitations, both academia and industry are conducting extensive research to develop new methods and tools for the more efficient engineering of automation systems. The component-based modular engineering approach has become an established method to bring

modularity to systems. The industry has also noticed that potential improvements can be achieved through the use of IT tools to virtually develop and commission automation systems. A number of automation suppliers have launched virtual engineering tools for the more efficient development of automation systems. The advent of virtual engineering tools has made it possible to engineer automation systems concurrently and in an integrated manner. This engineering concept is essentially based on smart components, which consist of graphical representation, kinematic modelling, and control information that describe the behaviour of the required machine or cell.

The current virtual engineering approach is mainly oriented to process modelling and optimisation. The approach is not yet fully exploited from the mechatronic perspective [124]. There is a lack of tool integration which can be clearly seen in virtual engineering environments and in control software development for runtime controllers (such as PLCs and Robot controllers) [125]. Due to this lack of integration, control logic must often be defined more than once in different engineering tools. The control behaviour is first defined within the virtual engineering tools to simulate the machine behaviour. However, the same control logic must then be re-implemented manually in the proprietary control software development tools [20]. In order to cope with these limitations, new methodologies are necessary to fill the gap between mechanical and controls engineering [19].

In recent years, the potential for information reuse from virtual engineering tools for the automatic generation of control logic has been recognised as a promising area in control engineering practice. In contrast to the existing approach, automatic generation can significantly reduce the manual coding work by automatically translating machine specifications into control code from the previous engineering phases. By reusing information created in early phases of an engineering project, not only can the overall engineering efficiency be improved, but the gap between mechanical and controls engineering can also be bridged.

The generation of control logic from virtual engineering tools is a novel research area. Very few tangible results from research in this field have been found. Most of the available approaches are focused on generating the source code for part of the required program and target only automatic operation of a machine. In reality, this only represents a small percentage of a machine control program. Furthermore, most of these methods do not support fault diagnostics and integration with the HMI and manufacturing execution systems.

3 Methodology and Implementation

3.1 Introduction

A current trend in manufacturing engineering is the use of IT tools, referred to as virtual engineering (VE) tools, to support virtual prototyping and validation of automation systems design. Most VE software solutions provide 3D CAD based simulation environment, which allows the control configuration of a system to be visualised and validated against its physical layout. The simulation helps to identify discrepancies in the control behaviour and overall process at an early stage, and thus provides the capability to significantly decrease the cost and the lead-time of a project. The ability to directly reuse modelling data (i.e. from 3D machine layout and process plans) to generate deployable control software is potentially very promising to replace the error-prone and time-consuming manual coding of the control software.

As reported in Chapter 2, the reuse of information for control software generation has been applied already in the manufacturing sector and a number of prototype systems have been developed. The approach can significantly compress the project lead-time and bridge the gap between mechanical and control engineering by working concurrently. However, the current implementations only present partial and non-standard solutions and thus cannot be deployed in real production systems in the industry.

There are two main reasons for the current partial implementation of the control software. The first reason is that the current solutions do not address the necessary functional and structural requirements of control software. The second reason is the unavailability of the required control information within VE tools to generate the complete control software. This is because most of the modelling, simulation and analysis capabilities of VE tools are primarily intended only for manufacturing systems process planning only. Such VE tools often focus on a high-level description of the intended production process of a machine to control the sequence of operations and simulate the 3D machine without taking into account the requirements of the logic definition for the runtime control systems. A direct consequence is that VE tools lack specialised control engineering functions (such as machine safety), and the control behaviour definition methods often do not support the definition of the runtime control logic. As a result, the control behaviour is of limited use for control logic generation.

Given the above facts, this chapter proposes an approach for the generation of the control code aimed at addressing the aforementioned limitations. The required enhancement of control information within VE tools is discussed. Finally, the proposed control software architecture and the methodology for control software generation are presented.

3.2 VE Based Control Software Generation Framework

The proposed framework for generation of control code from VE tools is shown in Figure 3-1. The framework is aimed at addressing the limitations of the current control software engineering approaches by using 3D-based manufacturing process planning and simulation tools referred to here as Virtual Engineering (VE) tools to generate control code, while fulfilling all the industrial control system requirements.

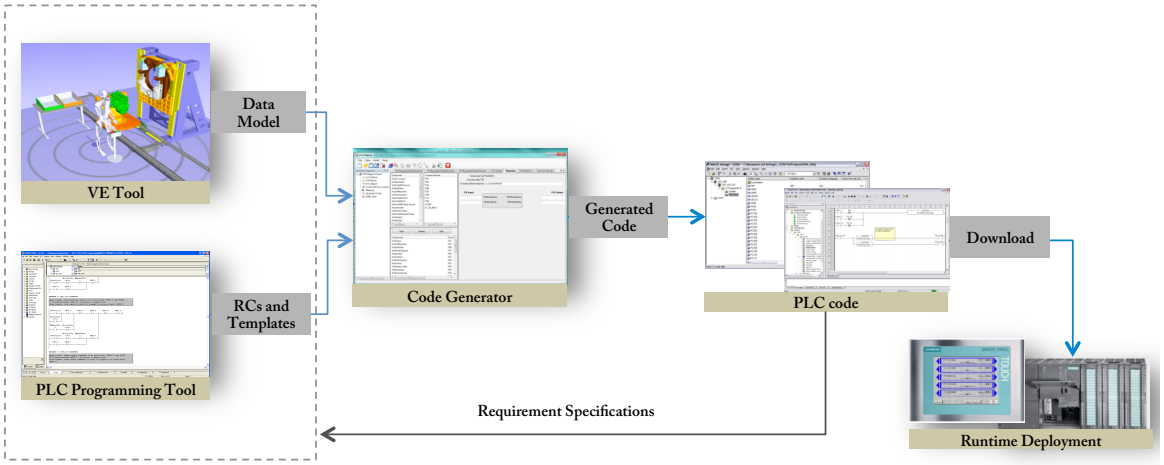


Figure 3-1 Proposed framework for logic generation

The work carried out in this research by the author is shown in Table 3-1. To propose and implement an acceptable approach, the control software requirements of the automotive industry are documented. The existing approach for defining the control behaviour of a manufacturing cell within a VE tool (the CCE software product was used) is analysed to identify the limitations from the control engineering perspective. To enhance the control information within VE tools for the direct deployment of the control software, a new method for the control behaviour definition is proposed.

Table 3-1 Summary of the author's contributions

Author's Contributions	Description
Control Software Requirements	Documentation of functional and non-functional requirements of control systems.
Enhancement of Control Information within VE Tools	A new logic definition method for defining control behaviour of a manufacturing system within virtual engineering tool to enable the direct deployment of control software.
Deployable Control software architecture	A new PLC and HMI control software architecture, which supports the direct deployment approach as well as architectural requirements of the end-user.
Software Generation Approach	Methodology of generating the PLC control code and the HMI screens automatically according to the proposed control software architecture by reusing the control behaviour defined within VE tools

The proposed code generation framework adopts the template-based approach used within the auto-

motive industry to write control software, wherein the template is populated from the machine specific control data imported from the virtual model of a manufacturing cell. A control software architecture that facilitates the direct deployment of control code from the virtual model of manufacturing cell is proposed. The methodology for automatic code generation is then designed and implemented.

3.3 Control Software Requirements

The purpose of this section is to identify control software requirements. The control software requirements presented in this section are based on the powertrain assembly systems of Ford Motor Company. The requirements can be categorised into software structural requirements and functional requirements. The purpose of the structural requirements is to promote commonality and common best practices in software development. The software structure requirements are based on the Ford Motor Company's FAST (Ford And Siemens Transline) PLC programming standard for Powertrain Operations Manufacturing Engineering. The functional requirements of assembly automation systems are presented in the remainder of this section.

3.3.1 Machine's Modes of Operation

In order to fulfil a wide range of activities (i.e. commissioning, normal operation, and fault diagnostic) the following three modes of operations are required:

Automatic Mode allows execution of a complete sequence of operation of a machine without any operator intervention. Automatic mode allows a machine to start auto cycle if all the safety and interlock conditions are met. During auto-cycle, respective automatic interlocks should are checked just before starting the move and continually checked until end of the movement. Any machine fault will drop-out the machine form automatic cycle. The automatic cycle can be terminated by requesting stop-end-of-cycle or run-out via respective pushbuttons.

Manual Mode allows manual control of machine movements (motors, cylinders, stops, nut runners etc.). This mode is typically used during commissioning and maintenance for testing and error recovery. In this mode, actuators can be moved between home and work positions regardless of the sequence of operations via pushbuttons provided on HMI screens. In this mode, the movement of actuator remains active as long as the manual movement button is being pressed. Manual interlocks and safety conditions are checked before initiating any movement and are continually checked as long as the movement is active. Any violation of interlock or safety condition should inhibit the movement. For example, if a part is clamped then pushing the transfer button (to transfer the part) should not cause an action as the movement would cause a clash.

Semi-Automatic Mode allows manual execution of a group of motions in the required sequence of

operations. A specific button is provided for each group of motions.

3.3.2 Cycle Types of Machine Operations

A brief description of the required cycle types of machine operations is given below:

Continuous Cycle is used for normal production mode of operation. Continuous cycle is requested through a 'start cycle' command initiated from the HMI.

Runout Cycle does not allow introduction of new parts in a machine. Machine remains in auto cycle until it finishes processing parts that are already in the station.

Single Cycle is used to process one part at a time. In single cycle mode, the machine processes part that is currently in the machine or the next part if machine is empty. The machine allows introduction of a new part, but operation on the new part are not performed.

The machine will process part currently (or the next part if machine is empty) in the machine and release, allowing introduction of a new part. Operation on the new part shall not be performed.

Dry Cycle is used to test the robustness of mechanical equipment and control software. In dry cycle machine is typically operated in 'automatic cycle' without parts for about 24 hours. To execute dry cycle, the part present sensors are bypassed. Dry cycle is typically activated by setting up a 'dry run flag' in the PLC program.

Return to Initial Position enables the automatic return of all actuators of a machine back to their initial positions. During the return it must be ensured that all actuators follow a safe return path to avoid any mechanical clash.

3.3.2.1 Machine Safety

Machine safety functions enable the safe operation of a machine. The safety function can be categorised into interlock checking, constantly monitored zone and general safety checks. A brief description of these is given below:

Interlock Checking is required to prevent mechanical clashes during machine operation. An interlock is a relationship between two or more functions such that one must be maintained whilst the other function is performed, e.g. a workpiece must be clamped during the drilling operation. To ensure safe machine operation, all movements are interlocked to ensure that machine damage cannot be inadvertently made using machine controls.

Interlocks are provided both in manual and automatic mode, usually known as manual interlocks and automatic interlocks respectively. Manual interlocks are to prevent operator's mistakes during the manual mode of operation, while automatic interlocks are required to avoid any unexpected movement during the automatic sequence, e.g. a clamp moving off its limits due to low air pressure.

Constantly Monitored Zone (CMZ) is a set of input signals that should be present throughout the ma-

chine operation to ensure human and machine safety. These include emergency stops, safety gates, air pressure, overload thermal trips etc. All CMZs have the highest priority and immediately stop a machine if activated.

General Safety Checks are required to ensure that:

- There should be no machine movement machine as a result of switching on power.
- No unexpected movements or hazard should exist after starting up a machine from a stopped condition or in recovering or returning a machine to a required position.
- Selection of a mode shall not initiate movement, and there must be visual indication that a particular mode has been selected.
- A machine shall prevent automatic restart of any motion or motor when power is restored after power failure.
- Where memory devices are used then correct operation, retention and recovery must be ensured in case of power supply interruption to prevent a hazardous situation resulting from a shutdown.
- Loss of supply voltage shall stop the machine without damage and require a manual restart with at least these two deliberate acts by the operator to:
    1. first select the machine mode, and
    2. then press a button to initiate a particular manual function or automatic cycling

3.3.2.2 Machine Diagnostics

Machine diagnostics is one of the most important aspects of the machine control logic. The purpose of the diagnostics functions are concerned with finding faults as well as waiting for states arising in machines. Diagnostics should occur by default of the logic for controlling a machine. The control software must make certain that the machine operation is safe and should present a correct diagnosis of problems in case of any hardware failure. This part of the control logic requires significant time and can increase the cycle time substantially.

Machine diagnostic is usually classified into two categories:

Machine Fault is defined as any non-waiting state that is not manually initiated, stops the machine from producing and normally requires intervention for recovery. A machine fault is raised if a machine voids a predefined safe operation procedure. A machine needs to be stopped when a machine fault is raised. A number of standard fault checks are usually performed, e.g., pairs checking, supervisory time, position monitoring etc.

Each machine fault is assigned a priority number such that more critical faults are displayed in preference to less critical errors. When a machine enters the fault mode it latches the fault that has highest priority. This highest priority fault is most likely to be the primary fault responsible for the machine

breakdown (for example, an emergency stop discharges air and can trigger low air pressure fault message as well).

Machine warning is defined as any event that does not require the machine operation to stop but needs to be corrected. In general, a machine warning is an event or condition detectable by the machine logic that could indicate a reduction in the overall performance of the machine, e.g. bad PLC batteries, low lubrication level, no workpiece to process. Warnings may occur while a machine is running or not. Warning messages do not have any priority numbers associated with them.

Warnings are mostly machine specific but there may also be some warnings that apply across an entire plant. Typically the controls, process and productivity groups will produce a set of warnings lists after a careful review of specific needs and capabilities. Warnings are usually categorised as plant, programme, line or machine-specific warnings.

3.3.2.3 Operator Messages

Presenting messages on an HMI in a meaningful way has always been an essential part of machine control. The messages must be presented in a way to enable the detection of the machine state and any faults with minimum effort. Operator messages must not only reflect the status of a machine but also should guide the operator during error recovery.

3.4 CCE, Virtual Engineering Environment

In this research, the Core Component Editor (CCE) VE software has been used to implement manufacturing process simulation models from which the basic information required to generate control code is derived. The CCE is a set of engineering tools, developed at Loughborough University, that provide 3D and process modelling functions for automation systems. The CCE software was designed based on the requirements of the automotive industry to support engineering of assembly automation system over their entire lifecycle.

The CCE toolset is a lightweight modelling and simulation package based on the use of non-proprietary and open standards data formats. The CCE tool set uses standard VRML formats for 3D modelling and a generic State-Transition Diagrams definition to support the control logic editing and visualisation. The CCE software architecture includes interfaces to external environments, such as OPC client/server interface that allows control-related events to be sent/received. A so-called Broadcaster enables linking to Web-based applications (e.g., Web-based HMI, Web-based VRML models).

In the CCE, machine geometry (CAD), kinematic behaviour and control behaviour are integrated around a common data structure, referred as common model architecture that describes a hierarchical system as a composition of components. A set of system representations (3D visualisation, state-transition diagrams and timing diagrams) provide a variety of specialist and non-specialist views that

are designed to support both detailed engineering tasks and general collaboration between project partners and engineers from different domains.

The CCE tools development was driven by the concept of component-based system architecture [12, 126], which seeks to enable re-usability and re-configurability of basic modelling constructs. The concept of a "Component", which is defined as a re-usable, reconfigurable unit providing the data integration mechanisms for control, 3D modelling, kinematics and other data types describing a particular resource (e.g. component faults), is central to the CCE tool development [127].

Using the CCE tool, the overall modelling and simulation workflow can be broken down into two main stages, which are the "component modelling" and "system modelling". A brief description of these stages is given below.

3.4.1.1 Component Modelling

Component modelling is the first step for modelling a manufacturing system. In CCE, Component Builder Module is provided to create components and store them in a component library. The Component Builder Module functions cover three domains, namely 3D geometry modelling, kinematic modelling and control behaviour modelling. A component could consist of one or a combination of these three domains. An outline of the component modelling is provided in Figure 3-2.

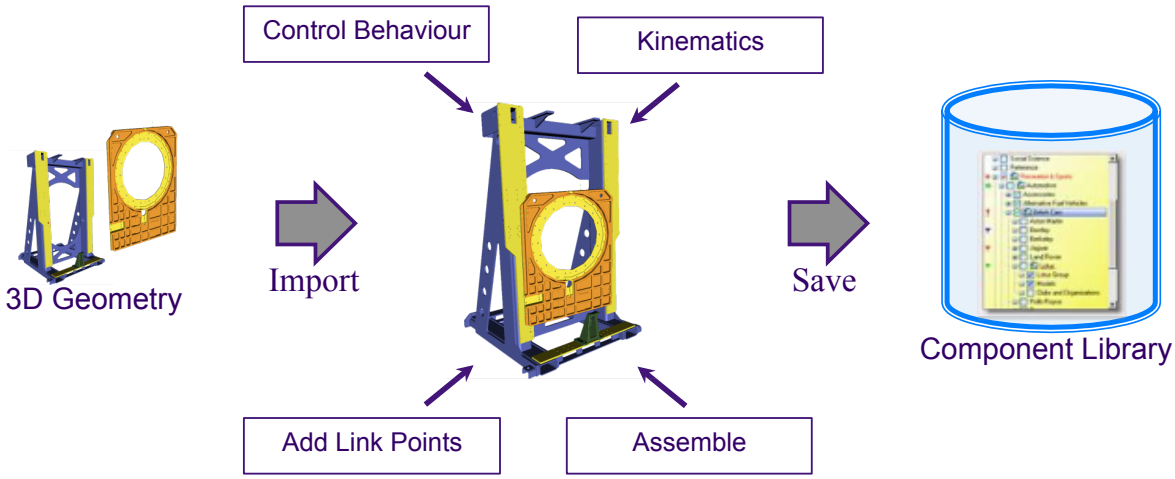


Figure 3-2 An overview of component modelling in CCE

3D geometry facilitates visualisation and description of the physical attributes of machine components, such as dimensions and shape. CCE does not provide any CAD modelling functions. Instead, 3D CAD geometry in a VRML (Virtual Reality Modelling Language) format can be imported into the CCE Graphic Library. The imported VRML models are surface models only, which greatly reduce the memory size of the simulation models. The Component Builder Module uses the geometry in the Graphic Library to enable Lego-like assembly of components with the help of Link Points. Link

Points are location points within the model space of the parent geometry. Example link points created for assembling a gripper are shown in Figure 3-3.

Once a component is assembled, kinematic behaviour can be defined to animate the 3D geometry. In CCE, two types of kinematic joints, i.e. translational and rotational, can be modelled. The Component Builder Module provide functions to define the complete kinematic behaviour of joints required for simulation, such as type of joint, displacement, acceleration and velocity.

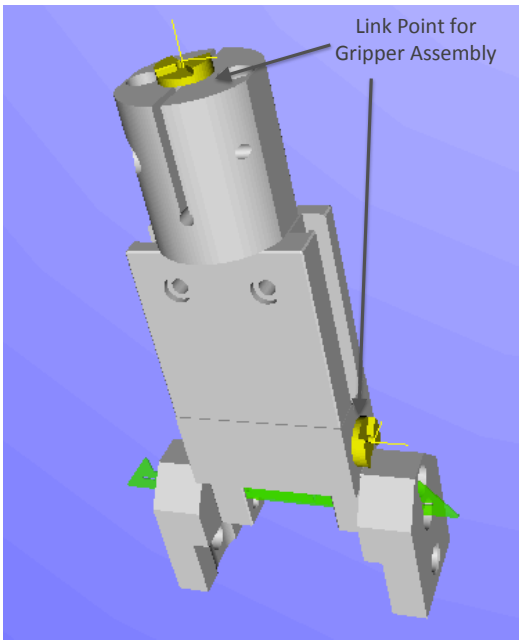


Figure 3-3 Link Points for assembling gripper

Finally, the control behaviour of a component can be described via State-Transition Diagram (STD). STD outlines the high-level functional states in which a component can exist. The transition from one state to another is controlled by transition conditions between the states. The transition conditions are configured during system modelling. An example STD to define the control behaviour of a 2-position actuator component that moves between home and work positions is shown in Figure 3-4.

An STD within CCE consists of three types of states: Home Initial, Static State and Dynamic State. A brief description of these states is given below.

Home Initial is the first state of a component and represents the home position of a component. This state is essentially a static state.

Static State is a known position of a component. In a static state, a component waits for a transition to become true to move to the next state.

Dynamic State is a state in which component moves between static states. In a dynamic state, the time required to complete the action is defined. This time is used to drive the simulation and calculate the cycle time, while a maximum allowable time is used to set the time-out period for run-time

diagnostic purposes.

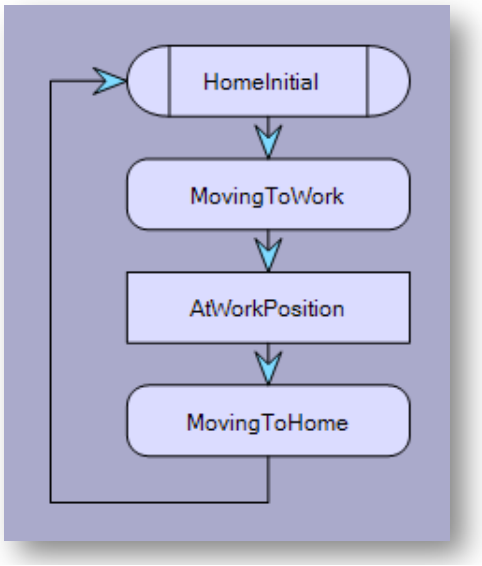


Figure 3-4 STD for a 2-position actuator

To facilitate simulation of a number of manufacturing resources, CCE supports modelling of the following types of components:

1. Actuator component

   Actuator component is defined as an electro-mechanical component, such as clamp. An Actuator has a physical geometry and control behaviour (for example, a proximity sensor turns ON and OFF depending on presence or absence of a part and its electrical configuration, i.e. PNP or NPN).

2. Sensor Component

   Sensor component detects external stimuli and responds in a distinctive manner. A Sensor component has a physical geometry and control behaviour.

3. Virtual Component

   Virtual component has no physical geometry but has behaviour that may affect the automation system such as a timer or a routing algorithm that based upon its inputs, controls the flow of workpiece through the automation system.

4. V-Man

   V-Man (virtual mannequin) represents a human operator in the simulation environment. An inverse kinematics engine for the limbs allows intuitive work sequence editing. V-Man has a physical geometry and control behaviour.

5. V-Rob

   V-Rob represents 6-axis industrial robots. Inverse kinematics functions of V-Rob allow

intuitive editing of a robot's axis position.

6. Geometry

   Geometry component has no inputs or outputs but is used to build automation systems providing a capability to represent a machine's standard framework such as guarding, frames and railings.

3.4.1.2 System Modelling

Figure 3-5 gives an overview of the system modelling. At system modelling level, components of the system are instantiated from the "component library". Components are instantiated one by one and assembled via link points. Once components are assembled, the sequence of operation of the system is then complemented by defining transition conditions to interlock components with states of other components (i.e., adding transition conditions to the STDs of the components).

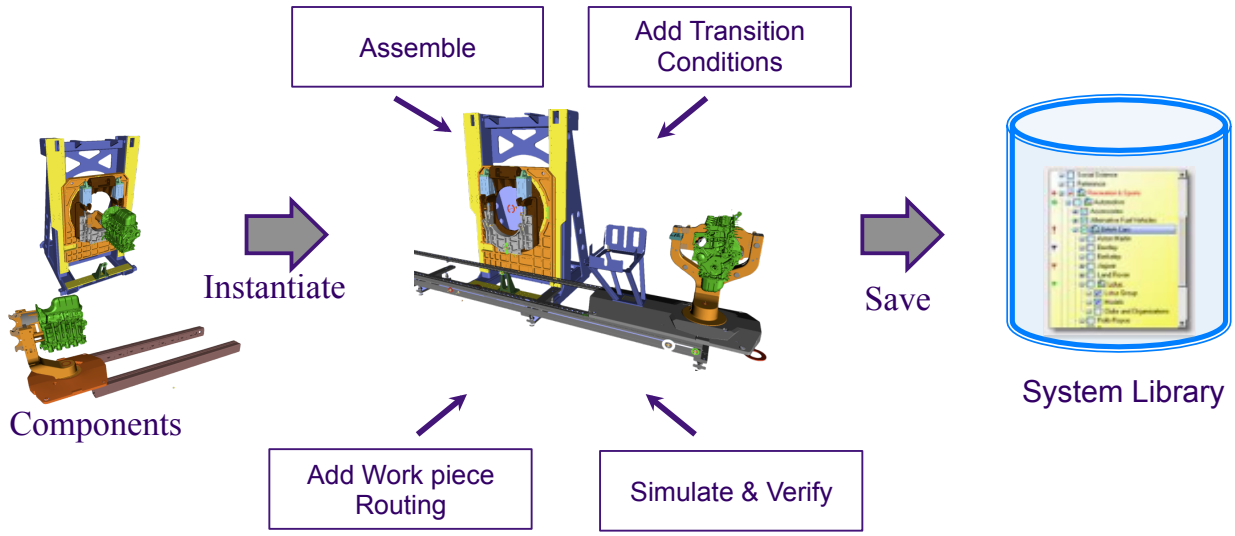


Figure 3-5 An overview of system modelling in CCE

The last step of system modelling is the workpiece routing logic. Workpiece routing logic is used to simulate triggering of sensor components and the routing of the workpiece through the system. An example workpiece routing screen is shown in Figure 3-6. Workpiece routing logic is essentially a sequential flowchart composed of steps, actions and transitions. Each state describes position of the workpiece in the system. Actions are associated with a state to turn sensors ON and OFF. The transition conditions are states of actuator components.

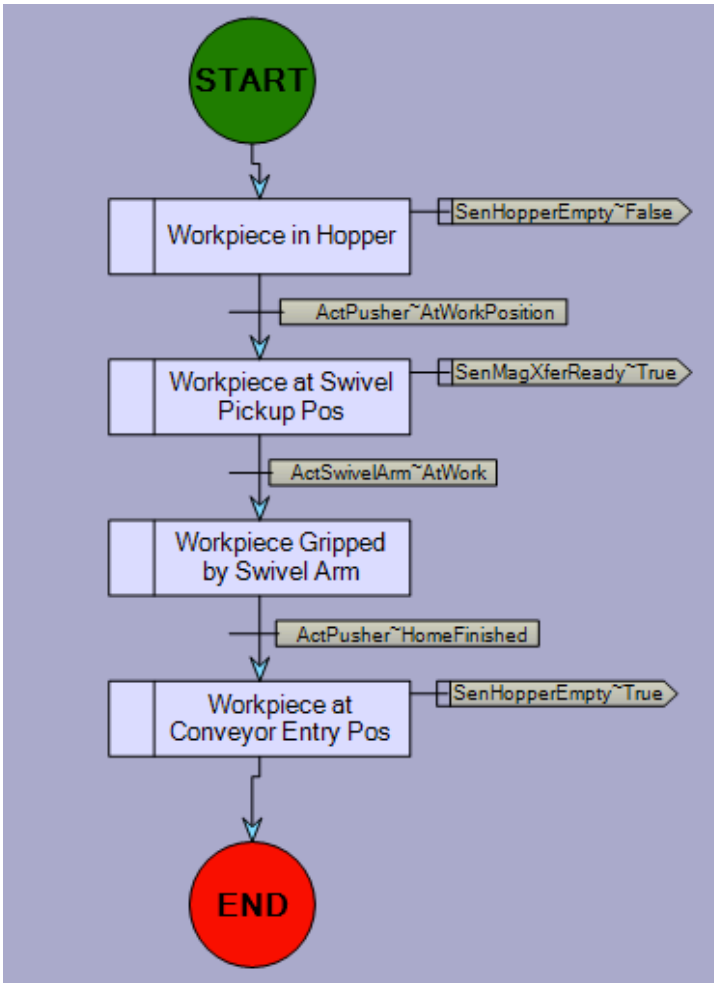


Figure 3-6 An example workpiece routing logic

Routing of workpiece is accomplished by transferring the link point of the workpiece from one component to another component. For example, when an engine is clamped from a pallet then to transfer engine from pallet to clamp, the link point of engine is transferred from pallet to clamp in the respective state of the workpiece routing logic.

In workpiece routing logic, triggering of sensors is done by associating states of sensor components (e.g. ON and OFF in case of proximity sensor) to the states of workpiece routing logic. In such an approach, simulation of the behaviour of sensor relies on the knowledge and experience of engineers to predict which sensors will be triggered as the workpiece travels through the system.

Finally, machine simulation is executed to verify the machine design and operation. Simulation enables the control logic to be executed and viewed in conjunction with a set of visualisation tools i.e. the 3D viewer and timing chart. Once the control behaviour has been verified, the machine configuration and control logic can be exported in XML format for further use at later phases of the engineering process, such as for control logic generation, discrete event simulation and energy consumption analysis.

3.5 Enhancement of CCE Tools for Control Code Deployment

As aforementioned, the runtime control code generated from the control behaviour (state-transition diagrams) defined within the CCE must fulfil the industrial control system requirements and fits well in the framework of current industrial practice. The existing method of logic definition within the CCE is well suited for simulation purposes but has the following major limitations when viewed from the control system requirements perspective:

1. The existing control logic definition method for actuator components does not provide a flag to distinguish between start and end of work cycle of actuators. This results in re-initialisation of the work cycle of actuators without any handshake signal typically used in runtime control systems to indicate end of operation. As CCE supports simulation of a single workpiece only, the abnormal behaviour of a system due to re-initialisation remains undetected during virtual commissioning. This can potentially leads to errors in the generated control code. It has been observed that in those scenarios where the workpiece remains at the same position at the end of the work cycle, the re-initialisation results in processing of a single workpiece again and again in a loop during runtime.

   For instance, the control behaviour of a two-position actuator shown in Figure 3-4 is defined with the help of four states. The last state (i.e. “moving to home pos”) in the work cycle is a dynamic state. As soon as the actuator reaches home position, the work cycle re-initialises and “Home Initial” state becomes active state.
2. The identity number of a component’s states (Component State ID) plays a critical role in mapping actuator components with Runtime Components (RCs). However, in CCE the assignment of State IDs does not follow a rigorous rule, which results in inconsistent numbering of State IDs. This makes the automatic code generation process prone to errors, which can potentially lead to an abnormal behaviour of a machine. Such errors in control code are often very difficult to debug.
3. The control logic used for simulating some mechanisms is not consistent with the runtime control logic required to run physical mechanisms. As a consequence, the control logic definition of such mechanism cannot be used for control code generation. For example, the behaviour of the conveyor system shown in Figure 3-7 can only be simulated with the help of five states STD. However, from control logic perspective conveyor is run by electric motor, which has only two logical states (i.e. ON and OFF). As a result, the control behaviour used for simulation of such components cannot be used to control the physical components in runtime.
4. The control logic definition of actuators does not support definition of interlock conditions during dynamic states, which is critical requirement for any control system to prevent mechanical clashes.

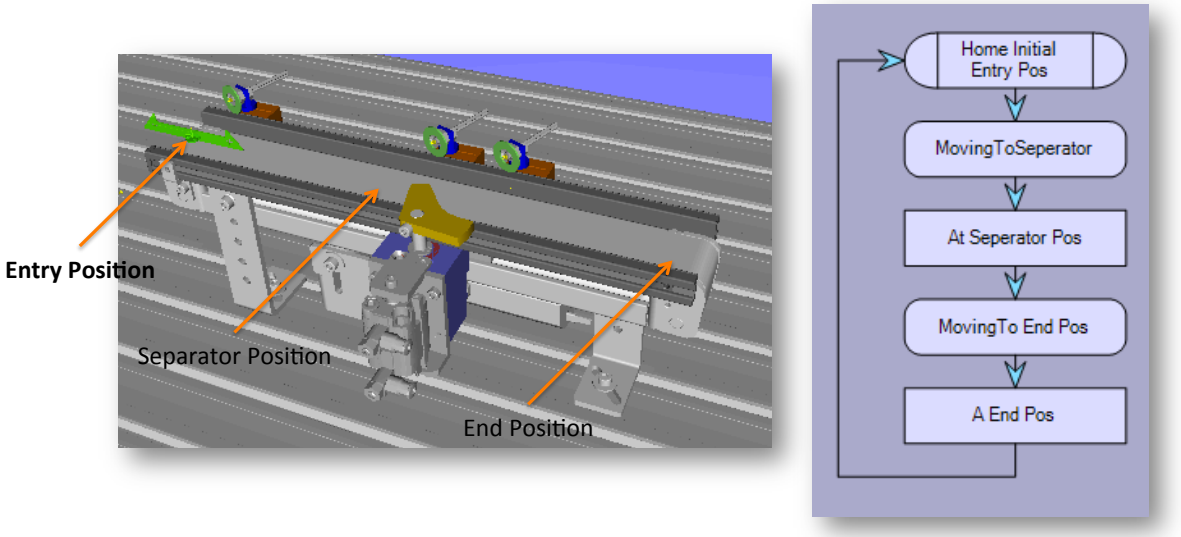


Figure 3-7 CCE model of a conveyor system transporting part from 'entry position' to 'separator position' and subsequently from 'separator position' to end position' after separator release

5. There is no top-level logic to drive the sequence of operations. Instead, the sequence of operations is controlled by the component logic itself.
6. The current data model underlying the CCE software functionalities does not distinguish between (PLC) control and non-control components. This results in difficulties while filtering information required for the program generation.
7. There are no functions to map the physical I/O addresses and the mapping of RCs with Sensor and Actuator components.
8. The simulation supports one workpiece only. However, for validating the control behaviour and cycle time calculation, simulation of machine operation involving more than one workpiece is critical.
9. To debug the control logic defined within VE tools, it is important to have a live view of state and transitions during simulation.
10. The workpiece routing logic often confuses users. It has been observed that users often confuse the workpiece routing with the control behaviour of a machine. In addition, it was found that firing sensors using workpiece routing logic fails to simulate the behaviour of sensors in some instances. For example, a proximity sensor used to detect the workpiece presence in an indexing table is triggered both by the workpiece as well as by the indexing table itself when the table rotates. Such scenarios are difficult to model using workpiece routing logic and can potentially leads to bugs in the control code that can cause a catastrophic failure. Therefore, a more realistic, intuitive and user-friendly method for simulating sensors and workpiece routing would be desirable.

Of these limitations, 1-7 are critical for control software generation and are addressed in this research, while 8-10 are critical for testing and debugging of the control logic, they are not required for software generation.

To overcome the critical limitations, a number of changes were implemented as a part of the research work achieved by the author in this thesis, summarised in Table 3-2. A brief description of these changes is given below.

Table 3-2 A summary of the changes in CCE

S. No	Limitations	Solution
1	Three types of states, i.e. Home Initial, Static and Dynamic	A new state Home Finished is introduced
2	Inconsistent State ID numbering	Consistent rule based State ID numbering and editable State IDs
3	No provision for modelling actuator components, which have a different runtime behaviour than simulation behaviour	Simulation-Only component is introduced
4	No provision for definition of interlocks	Provision for definition of interlock conditions in dynamic states
5	No provision for top-level logic to define sequence of operations	Process Logic is introduced to define sequence of operations
6	No categorisation of components to aid filtering of components required for control code generation	Categorisation of components into control and non-control components
7	No function for assigning PLC I/O addresses for runtime deployment	Mapper function developed for PLC I/O address allocation

3.5.1 Control Behaviour of Actuator Components

As stated in section 3.5, the existing practice of defining the control behaviour of actuators does not flag end of work cycle of actuator components, which can potentially results in bugs in generated control code. To address this issue, a static state 'Home Finished' is introduced as the last state of the internal sequence (i.e. STD) of actuator components to mark end of work cycle.

'Home Finished' is a virtual state which becomes active when an actuator moves back from work position to home position. The 'Home Finished' state remains active until it is acknowledged by a 'Handshake' signal from another component (i.e. Process Logic). Upon receipt of the acknowledgement signal, the internal sequence of component loops back to the 'Home Initial' state. Experience has shown that the use of 'Home Finished' state makes the definition of the control logic easy and safe.

For illustration, an STD of an actuator component is shown in Figure 3-8. This STD defines the

control behaviour of a two-position actuator that can move between 'Home Pos' and 'Work Pos'. The control behaviour of this actuator is represented by five discrete states. Each static state requires satisfaction of its respective transition condition(s) before moving to the subsequent dynamic state. The last state of the STD is "Home Finished" which represents the end of the working cycle of the component. The actuator remains in this state until the transition condition is fired by a 'Handshake' signal from a Process Logic.

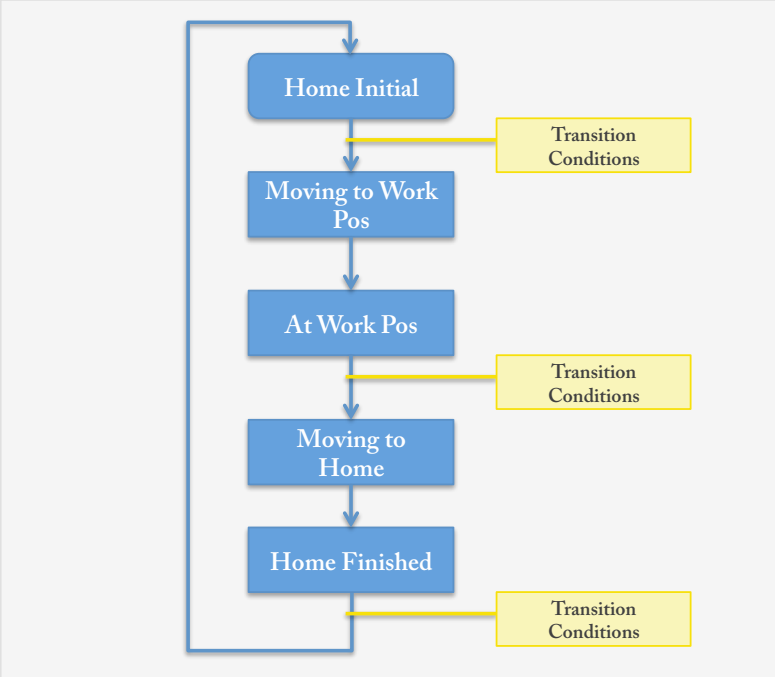


Figure 3-8 STD of a two-position actuator with a Home Finished state

3.5.2 Component State IDs

In the PLC runtime environment, the actuator components are mapped with the Runtime Components (RCs) to drive the physical devices. For mapping actuator components with RCs, the State IDs of actuator components must be identical to the State IDs of RCs. Any inconsistency in State IDs can result in a non-working machine and even catastrophic failure due to a mechanical clash.

In CCE, the State ID numbering is indistinct because it does not follow a rigorous rule. State IDs are assigned in ascending order, which does not take into account branching or later changes in the STDs (such as deleting a state). State ID is simply incremented each time when a new state instance is inserted in a STD. As a result, State ID depends on the state instantiating order and thus State IDs of two similar components can be significantly different if states are instantiated in a different order. The inconsistency in the State ID numbering is depicted in Figure 3-9.

To address this problem, State IDs are made visible and editable. To ensure the consistency of State IDs, a rule for the State IDs is defined. According to this rule, the State IDs are incremented from "top to bottom" and "left to right".

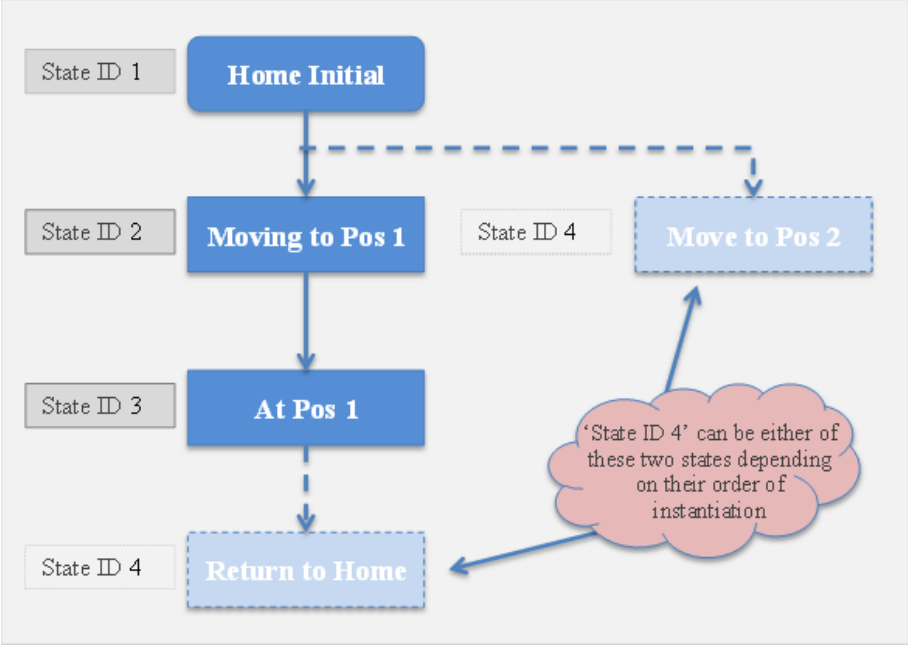


Figure 3-9 Ambiguous numbering of 'State ID'

3.5.3 Simulation-Only Actuators

As aforementioned, the logic behaviour required to simulate some actuators in CCE is entirely different than the runtime control logic. As a result, control code generated from simulation model composed of such actuators requires manual rework to make the generated code deployable. To address this issue, Simulation-Only component is introduced. Both Actuator component and Simulation-Only component are used to model such actuators. The Actuator component represent the runtime behaviour and the Simulation-Only component represent the control behaviour required for simulation.

To maintain the integrity of the model for a realistic virtual commissioning, Actuator component act as a driver of the Simulation-Only component. Simulation-Only component is categorised as a non-control component, which is filtered out during the control logic generation process. As a result, control code is generated for Actuator component only.

3.5.4 Component Interlocking

In machine control logic, interlocks are one of the prime features that prevent mechanical clashes between components. In CCE, interlocks are defined as transitional conditions. However, such a definition of interlocking cannot avoid a mechanical clash if the condition is void during movement (i.e. in a dynamic state). To address this issue, the component logic (STD) was modified to allow definition of interlocks in dynamic states, as shown in Figure 3-10. When a dynamic state is active, the respective interlock conditions are constantly monitored. The dynamic state goes into a halt state if any of the respective interlock conditions are violated.

In order to facilitate the definition of interlocks for both automatic mode and manual mode separately, each dynamic state accepts two sets of interlocks, i.e. one for manual interlocks and the other one for

automatic interlocks. Each set of interlocks can be viewed and specified separately.

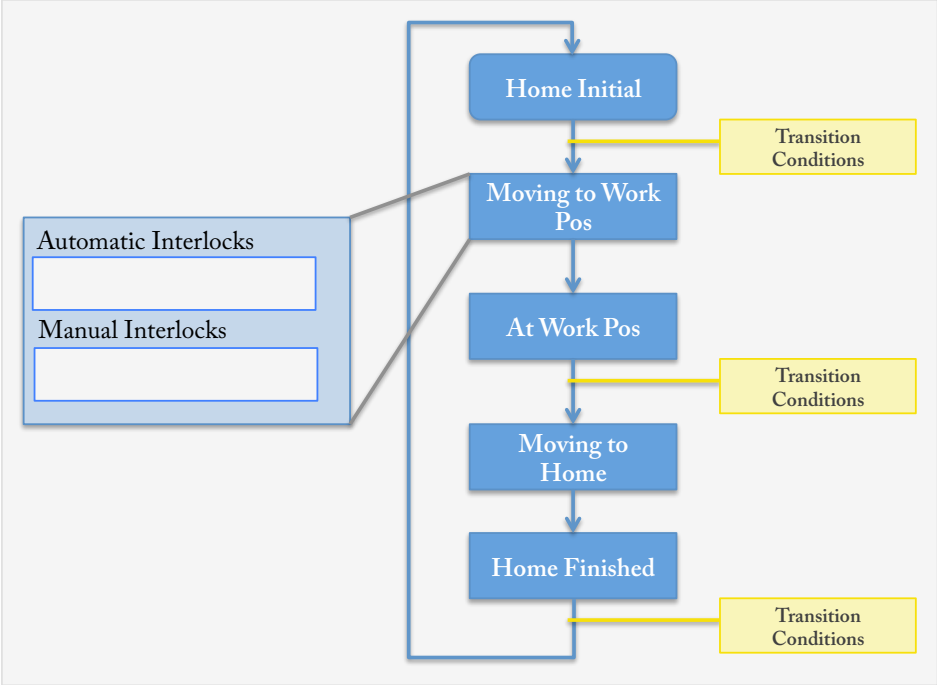


Figure 3-10 Definition of interlocks in STD of actuator components

3.5.5 Process Logic

To define the sequence of operations, Process Logic is introduced. Process Logic provides a set of services, which combine and orchestrate the service functionalities of a group of components to run a machine in a sequential manner. A machine can have several Process Logic sequences, communicating with each other to work in an integrated and controlled manner.

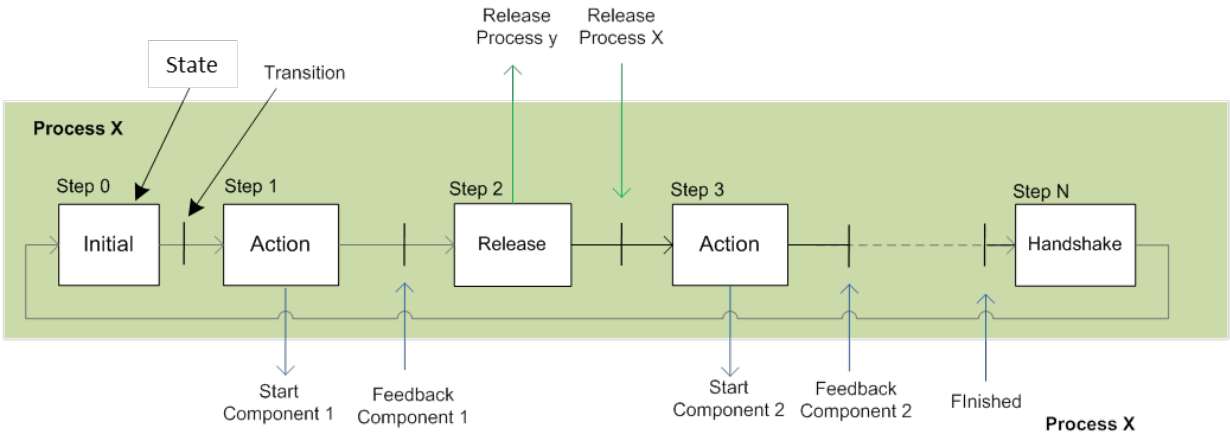


Figure 3-11 Example Process Logic sequence

Process Logic is modelled as a set of step and transition pairs, as shown in the Figure 3-11. The output of a step remains active until all conditions in the successive transition become true. Process Logic has following four types of step and transition pairs:

a. Initialisation

The first state of Process Logic is Initialisation. The purpose of this state is to flag the initial step of the Process Logic.

b. Action

An action state is used to drive components by firing their transitional conditions. Typically, an action step remains active until its transition condition receives feedback signal from the driven component to make sure that the action is completed.

c. Release

This step is used for communication between Process Logic sequences. The release step sends a release signal to another Process Logic sequence to resume its operation. The Release step remains active until its transition receives a Release signal back from the respective Process Logic.

d. Handshake

This step is used to acknowledge the "Finished" signal of actuator components. The process logic remains in this state until all accompanying components move back to their home state.

3.5.6 Component Categorisation

A manufacturing system consists of a range of resources to accomplish a manufacturing process. The virtual modelling environment integrates all these resources and thus contains diverse data. However, not all of these resources are controlled by PLCs. Consequently, filtering-out irrelevant data, such as virtual operators and robots, is an important step in control code generation.

To enable the filtering of control information, components are categorised into control and non-control components, as shown in Table 3-3. Control component can be defined as a sensor or actuator component that is controlled by a PLC and its underlying control logic behaviour is required for the control code generation (such as gripper, proximity sensor etc.). Whereas, non-control components are those components which are required for simulation or visualisation of a system but do not make part of a runtime control system. For example, modelling of a mannequin is essential for visualisation, cycle time calculation and operator trainings, however the behaviour of a mannequin is not required for control code generation.

Table 3-3 Categorisation of components

Control Components	Non-Control Components
Actuator	Geometry
Sensor	V-Man
Virtual	Simulation-Only Actuator
Process Logic	-
V-Rob	-

3.5.7 PLC I/O Address Allocation

For PLC I/O address allocation, a mapper function known as CCE Mapper has been designed and developed. I/O addresses can either by typed within the Mapping tool or imported from third party tools in XLS (eXceL Spread Sheet) file format. The I/O addressed can then be allocated to the actuator and sensor components using an intuitive graphical interface, shown in Figure 3-12.

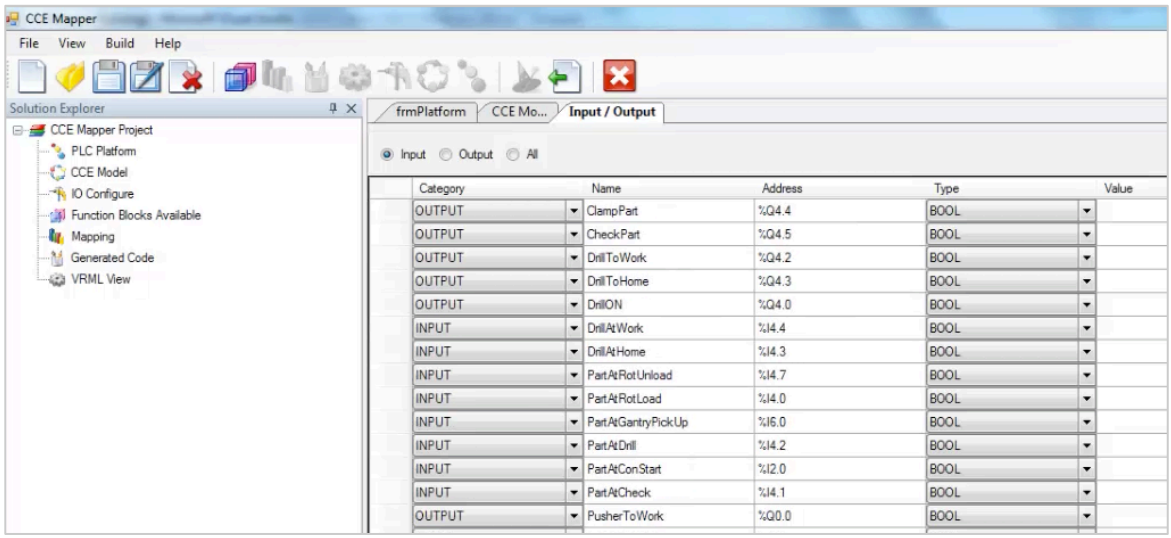


Figure 3-12 GUI of CCE Mapper for I/O address mapping

3.6 Alternative Approaches to Logic Generation

One of the very important aspects of the logic generation approach is the structure of the generated control code. The control logic can be generated either in a structure or unstructured manner. However, due to the requirement of the automotive industry FBs based structure code generation approach was only considered in this thesis.

Three different control logic structures were initially considered, namely SFC based structured logic generation, Logic Engine based logic generation and Structured Text (ST) based logic generation. The difference between the approaches is mainly in the structure of the code to control the sequence of operations of the machine. SFC based logic generation approach, as its name indicates, is based on the conversion of the Process Logic STDs into platform-specific equivalent SFC code. The Logic Engine based approach is based on conversion of the STDs into a data structure (i.e., Control System Data Model). While, the ST based approach is based on the conversion of the STDs into Structured Text code.

Due to a number of reasons, such as industrial acceptability and openness of the approach, the Logic Engine based approach was selected for the deployment of the control code in this research work. The strengths and limitation of the considered code generation approaches based on the author's experience form initial prototyping of each approach are presented in Table 3-4.

Table 3-4 Strengths and limitations of the alternative code generation approaches

Code Generation Approach	Attributes
ST Based Code	▪ Complexity level of developing the code generation tool is low ▪ The ST-based code generated for controlling sequence of operations can be used across a number of PLC platforms ▪ Difficult to understand the generated control code ▪ Fault diagnosability of the generated code is low ▪ Requires less memory than SFC based approach ▪ Fast PLC scan time compared to SFC based approach ▪ Does not support the dynamic HMI screens generation (presented in Section 3.9) ▪ Limited support for remote monitoring
SFC Based Code	▪ Complexity level of developing the code generating tool is high ▪ The generated SFCs for one PLC platform cannot be used for other platforms ▪ Easy to understand the generated control code ▪ Fault diagnosability of the generated code is high ▪ Requires more memory than the other two approaches ▪ Long PLC scan time compared to the other two approaches ▪ Does not support the dynamic HMI screens generation (presented in Section 3.9) ▪ Limited support for remote monitoring
Logic Engine Based code	▪ Complexity level of developing the code generating tool is low ▪ The same Logic Engine can be used across a number of PLC platforms ▪ Easy to understand the generated control code except the data structure ▪ Fault diagnosability of the generated code is high ▪ Requires less memory than the other two approaches ▪ Faster PLC scan time compared to the other two approaches ▪ Supports the dynamic HMI screens generation (presented in Section 3.9) ▪ Supports remote monitoring

3.7 Proposed Control Software Architecture

A typical PLC control software architecture is shown in Figure 3-13. The control software can be categorised into machine-specific and machine non-specific sections. The machine-specific part of the program refers to the control logic that is unique to each machine. This includes the application-specific sequence of operations, logic for driving machine mechanisms (such as clamp), interlocks and fault diagnostic. Whereas, the machine non-specific part of the program refers to the section of control logic that are generically required to run a machine.

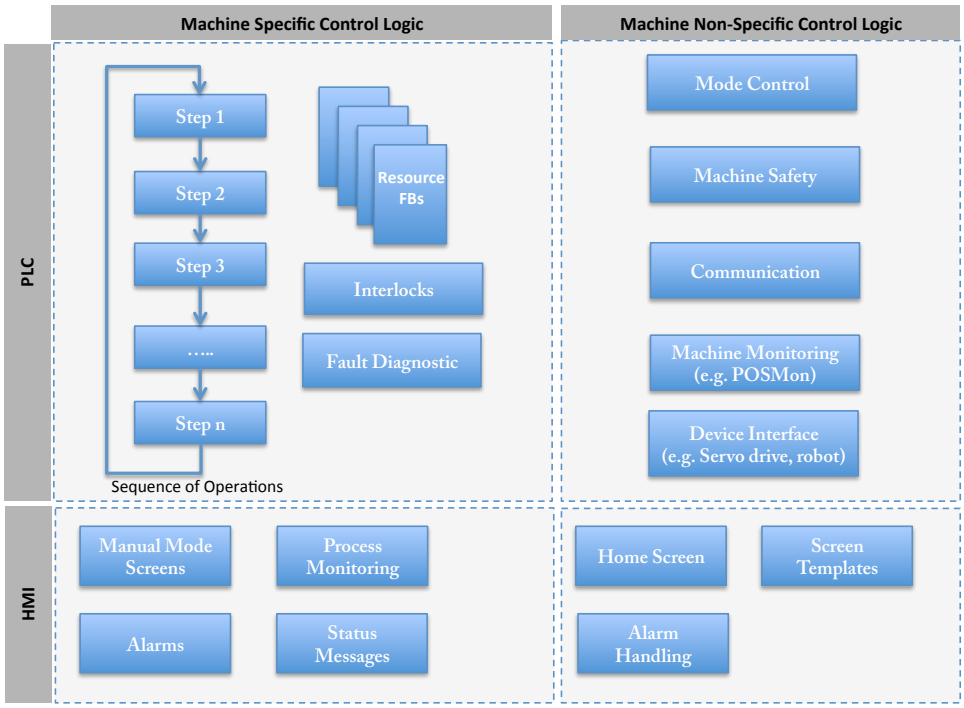


Figure 3-13 Basic parts of a typical PLC program, adopted and modified from [6]

The focus of this research is to deploy a complete program i.e. both machine specific and non-specific parts of the program. However, only the machine-specific part of the program is automatically generated from the control logic defined in the VE tools. Most of the machine non-specific part is predefined in a template.

In order to facilitate the implementation of the control software generation approach, a new control software architecture is designed in this research. The basic concept behind this software architecture is to generate the control programs using standard library components, which are driven by the control logic defined in the manufacturing process simulation tools. The overall architecture of control software is shown in Figure 3-14.

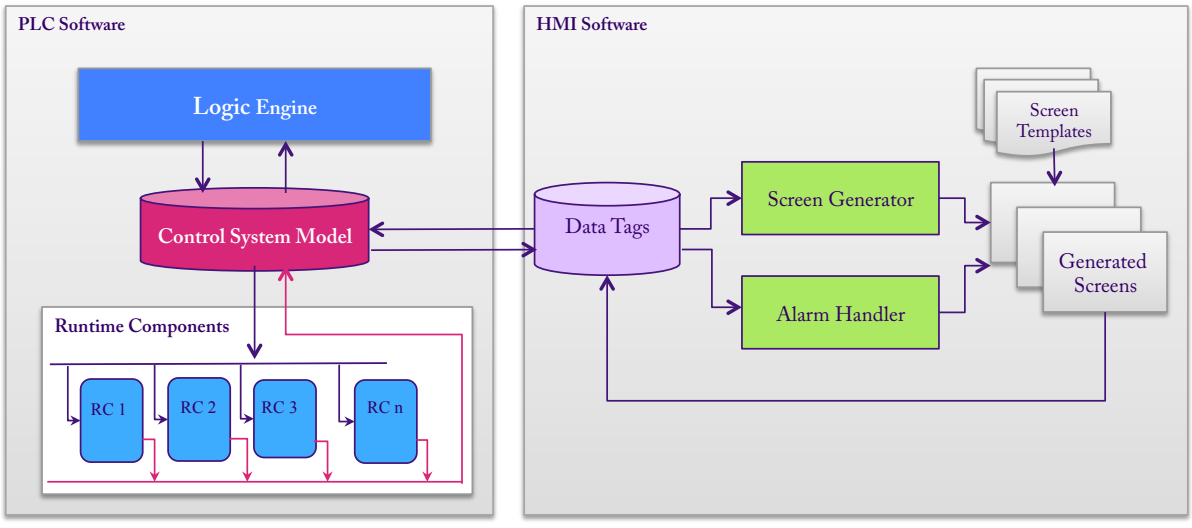


Figure 3-14 Overall control software architecture

It is worth noting that the designed software architecture is generic in nature and can be used for any control platform (e.g., from Siemens, Schneider or PLCopen). The architecture supports the automatic generation of both PLC software and the HMI screens, with consideration given to end-user software structure standards.

The working detail of the PLC software and the HMI software are presented in the following two subsections.

3.7.1 PLC Software Architecture

As shown in in Figure 3-14, the PLC software architecture is composed of three system components: Control System Data Model, Runtime Components, and Logic Engine. The description of these system components is given below:

3.7.1.1 Control System Data Model

It is a generic data structure defined to effectively store and organise the control information to run a system. It consists of different types of information, such as a system structure, operating modes, process control behaviour, component control behaviour, interlocks, fault messages. The Control System Data Model is generated on the basis of the control information defined within the VE simulation model of a machine cell.

The Control System Data Models can be further classified into the following three sub-models:

System Data Model

The system data model contains all the necessary control information to run a machine according to a specific sequence of operations. This control model is generated by automatically translating the control logic of the control components (i.e. actuator components, sensor components, virtual components and process logic) that compose a system. During the runtime, this data model is used as the interface between the Logic Engine and the Runtime Components.

The system data model defines the entire system's logical architecture (i.e. system, components, states, transitions, interlocks and the sequence checks in a system), as shown in Figure 3-15. A system consists of a number of components. A component can have two or more states. A state can have one or more transitions. A transition can have one or more sequence conditions. In addition, a state can have interlock conditions.

HMI Data Model

The HMI data model contains the data necessary to realise the automatic generation of the template-based HMI screens for manual mode operation and online control logic monitoring. This data model consist of two sub-models i.e. a data model for manual mode control and a data model for generation of process/control logic monitoring screens. The manual mode control model is used to generate the

manual control rows (pushbuttons) for manually driving movements of actuators. It is generated from the state behaviour of the actuator components only. While, the data model for process monitoring is used to generate the live view of the STDs for the Process Logic and Actuator Components on the HMI screen. It is generated from the state behaviour of the Actuator and Process Logic components. The architecture of the HMI Data Model is shown Figure 3-16.

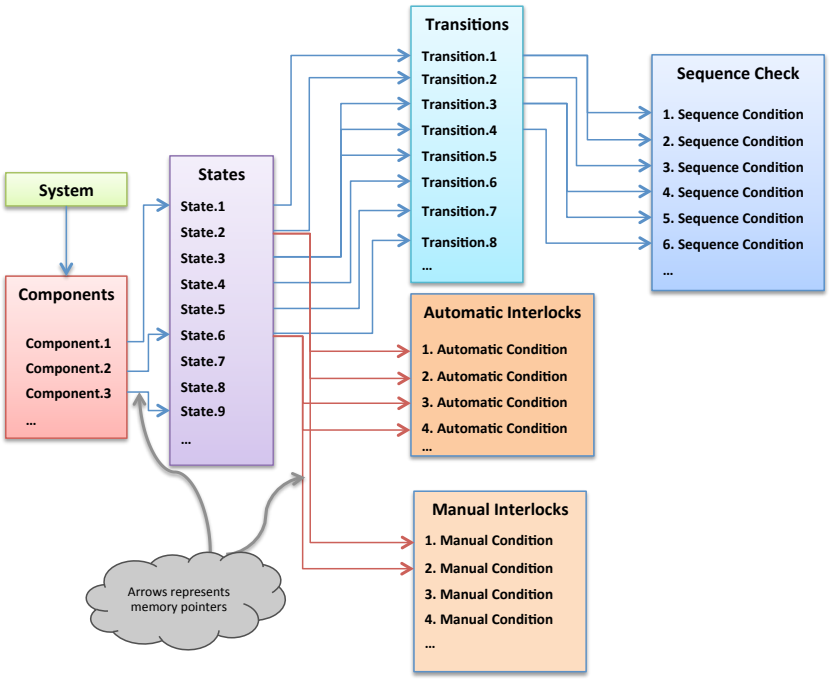


Figure 3-15 Architecture of System Data Model

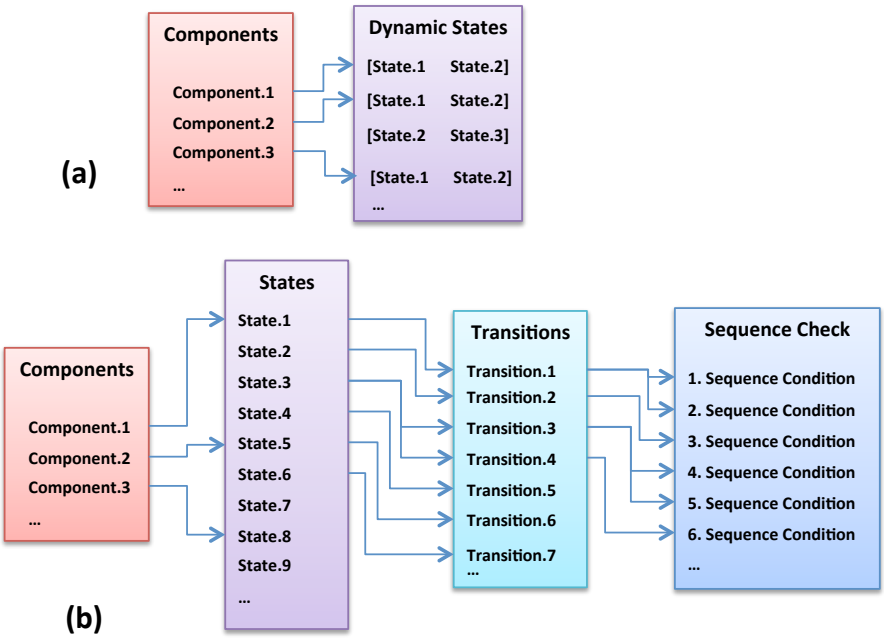


Figure 3-16 Architecture of HMI data model: (a) depicts the data model for manual mode control and (b) depicts the data model for process monitoring

Fault Management Model

Fault management models are used to store fault messages, which are displayed on the HMI screen. Component level fault messages are extracted from the CCE components while system level fault messages are defined manually. Faults are triggered by RCs, which are then communicated to the corresponding Control Models and then written into the Fault message models by the Logic Engine.

3.7.1.2 Logic Engine

Logic Engine is a principal component of the control software. It is a pre-validated and ready to use Function Block, which handles machine operations in a controlled manner. The entire source code of Logic Engine is generic and remains same for any system configuration. A simplified flow diagram of Logic Engine is shown in Figure 3-17.

Logic Engine is composed of a four functional modules, namely Operating Mode Handler, Component Orchestrator and Fault Manager. A brief description of each module is given below.

Operating Mode Handler

Operating Mode Handler is responsible for enabling and disabling the buttons on the HMI according to the selected mode of operation and state of the machine. For instance, it disables the automatic cycle start if all of the machine components are not at their initial position.

Component Orchestrator

Component Orchestrator continually scans Control System Data Model to evaluate the current working state and respective transition conditions of all components of the machine. If the logic conditions of any component are satisfied, a command for next working state is then generated according to the selected operating mode and cycle type.

In automatic mode, the next working state of a component is activated when the respective sequence and interlock conditions are satisfied. Whereas in manual mode, the next working state is defined by the command received from the HMI.

Fault Manager

Fault Manager periodically scans the fault status of all components. In case of any fault, all machine operations are inhibited and respective fault messages are triggered and saved in the Fault Management Model, which are then communicated to the HMI.

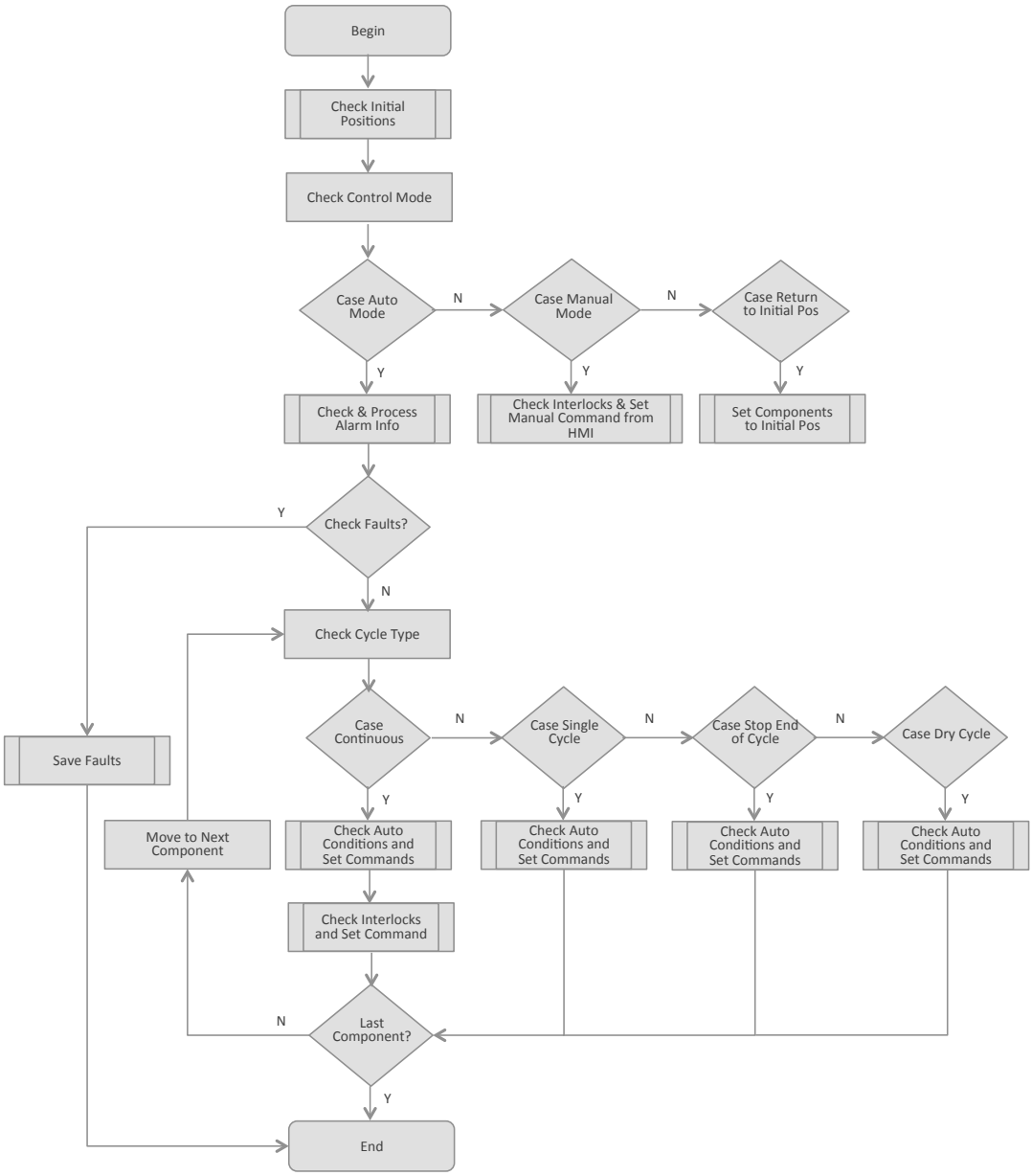


Figure 3-17 Flow diagram of Logic Engine

3.7.1.3 Runtime Components

Runtime Components (RCs) are pre-validated and ready to use resource-specific function blocks. An RC represents a machine actuator or sensor component in a PLC runtime environment. It is embedded with the control behaviour of a family of actuators and sensors with integrated diagnostics. As the RCs are generic and pre-validated, they are developed once and stored in a library for future reuse.

All events and faults of RCs are communicated to the calling instance i.e. the Logic Engine. RCs are directly deployable in a PLC program and are interfaced via direct parameterisation of the Control System Data Model as shown in Figure 3-18. The direct parameterisation refers to the interfacing through the input/output connection lines on the interface of the function block, which increase visi-

bility of the input/output variables. CCE Mapper carries out this automatic interfacing during program generation process.

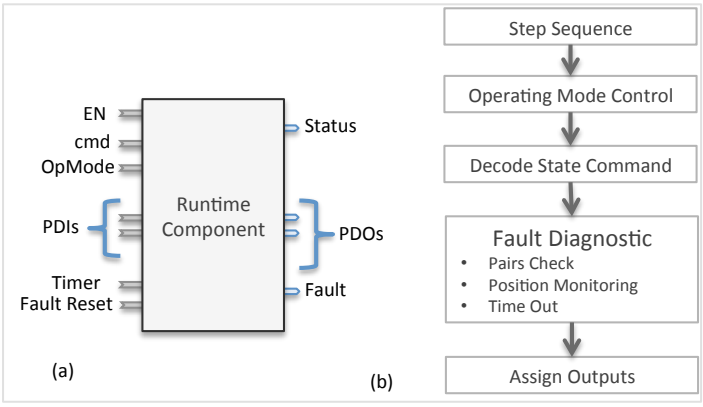


Figure 3-18 Structure of Runtime Component: (a) Interface (b) Internal structure

3.7.1.4 Inputs/Outputs

A number of inputs/outputs are provided on the RC interface. Some of the most common inputs/outputs are described below:

- **State Command (cmd)** is an integer input from the Logic Engine. It dictates the component to move to a specific state.
- **Operation Mode (OpMode)** is an integer input that controls the machine operating mode such as automatic mode and manual mode.
- **State Message (Status)** is an integer output that works as a feedback signal to the Control System Data Model. It updates the current working state of the Component in the Control System Data Model.
- **Fault** is the output of the integrated fault diagnostic part of the RC. This output is communicated to the Fault Management Model for a further action as required.
- **Timer** is an input (supervisory time) and is used to monitor the time required for an action to complete.
- **Process Digital Input (PDI)** is an input directly connected to the hardware, e.g. limit switch.
- **Process Digital Output (PDO)** is an output directly connected to the hardware e.g. electric motor.
- **Fault Reset** is an input to reset the active RC fault

3.7.1.5 Step Sequence

Step sequence resembles the STD of a corresponding component. The purpose of the step sequence is to a) compute the current working state of a component and b) to drive the component to next working state as required. The step sequence is driven by the PDIs, PDOs and 'State Command'. The output of the step sequence is communicated to the 'State Message' and PDO.

3.7.1.6 Fault Diagnostics

Several types of faults can be detected by RCs. A brief description of some of the most common fault

diagnostics is given below:

Pair Checking is used to make sure that the limit sensors of an actuator are working correctly. Example code for pair checking is shown in Figure 3-19. In this example, if the PDIHomePosition and PDIWorkPosition are true at the same time for more than 500ms, a fault will be triggered. The time delay is required to distinguish between a real fault and a switch bounce.

```
A #PDIHomePosition
A #PDIWorkPosition
= #tmpStartLimitSwitchFault
JCN LT03
L T#500MS
T #tmpTIme
LT03: NOP 0
```

Figure 3-19 Example code for pair checking

Position Monitoring monitors the end positions (static position) of the actuators. A fault is triggered if an actuator leaves its position unexpectedly. Example code for position monitoring is shown in Figure 3-20. The code shown is for two-positions actuator, and monitors the home and work position. A fault is triggered when the actuator leaves its position in the absence of the movement command.

```
A #HomePositionReached
AN #PDOWorkPosition
AN #PDIHomePosition
= #tmpHomePositionLost

A #WorkPositionReached
AN #PDOHomePosition
AN #PDIWorkPosition
= #tmpWorkPositionLost
```

Figure 3-20 Example code for position monitoring

Time-Out Fault is a supervisory timer. It is a classic method of detecting machine faults. The timer starts as soon as an actuator initiates a movement. A fault is raised if the actuator does not reach the destination position within the specified supervisory time. Example code for time-out fault supervision is shown in

Figure 3-21.

```
A(
A  #tmpToHomePosition
O  #tmpToWorkPosition
)
=  #tmpStartMonitoringTime
JCN XYZ
L  #MonitoringTime
T  #tmpTime
XYZ: NOP 0
```

Figure 3-21 Example code for time-out fault supervision

3.7.2 HMI Software Architecture

As shown in Figure 3-14, the HMI system architecture is composed of three system components: Screen Generator, Alarm Handler and HMI Data Model. A brief description of these system components is given below:

- **Data Tags**

  In HMI application, data tags represent internal and external variables. Data tags work as a communication link for the exchange of data between HMI and with other devices, such as PLC, required for the HMI operations. For automatic HMI generation, the external data tags are derived from the Control System Data Model to enable exchange of data between PLC and HMI during runtime.

- **Screen Templates**

  Screen templates are pre-developed generic HMI screens. These templates are automatically configured from the Control System Data Model during runtime.

- **Screen Generator**

  Screen Generator generates screens to support control operations, alarms screens and machine monitoring. It is composed of a number of functions to analyse HMI Data Model in order to analyse the structure of a machine configuration and the current status of machine components. When a screen is requested from HMI device, Screen Generator populates the corresponding predesigned template of screen and displays it on the HMI device.

- **Alarm Handler**

  Alarm Handler is responsible for reporting faults and warning messages to the operator. It displays the active fault messages on the screen according to their priority order. It also provides a function to access the fault history from the PLC software and displays it on the HMI.

3.8 Generation of PLC Control Code

This section presents a novel approach to generate complete and directly deployable control software from the control information derived from the CCE tools. To generate the control code, the software architecture was decomposed into the following three types of program elements:

- **Reusable static elements:** refers to the RCs, Logic Engine and the data structure of the Control System Data Model and a template of the structure of the control software, which are used. These elements are developed during pre-engineering phase and stored in the CCE Mapper library. During the code generation process, these elements are instantiated from the library and used directly without any change.
- **Dynamic elements:** refers to the logic repository and programs for actuator and sensor components. The logic repository is dynamically populated with runtime control models created

by translating the control behaviour of CCE components. For certain PLC platforms, some platform-specific elements might also need to be generated dynamically. For instance, in case of Siemens STEP 7, a data block for each RC has to be generated dynamically.

- **PLC platform-specific elements:** refers to PLC platform-specific information required to implement the source code of executable control software. These typically include header information for the overall control software and other blocks. For a specific platform, this information is identical for any system configuration. This information is defined once and stored in the reusable templates library of CCE Mapper.

The workflow for code generation is depicted in Figure 3-22. The code generation process was divided into pre-engineering phase and system engineering phases. The reusable elements and the platform specific elements are developed during the pre-engineering phase. These program elements are developed once and stored in a library. These library elements are reused for any number of systems. During the system engineering phase, dynamic elements are generated automatically from the simulation models of manufacturing cells and are combined with reusable and common system data to generate the complete control software. It can be seen from the Figure 3-22 that the only manual process required during the system engineering phase is the component mapping.

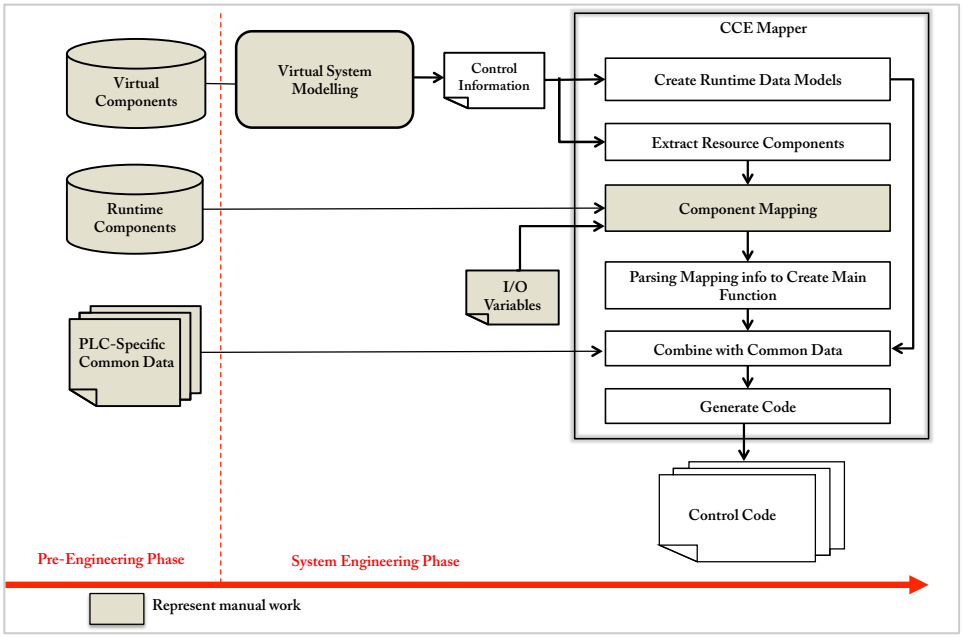


Figure 3-22 Workflow of the PLC code generation

The pre-engineering and system engineering tasks are explained in the following subsections.

3.8.1 Pre-Engineering Phase

The pre-engineering phase involves development of component library and PLC platform-specific elements. A brief description of these is given blow.

3.8.1.1 Component Library Development

The component library development task involves development of virtual machine components in CCE and the corresponding resource specific RCs in vendor-specific PLC programming tools. These components are validated and stored in their respective libraries.

3.8.1.2 Data Types Definition for Control System Data Model

The structure of the Control System Data Model cannot be described using the IEC61131-3 standard's elementary data types. This led to the definition of user Defined Data Types (UDTs). Six data types were defined to describe the structure of the elements of the Control System Data Model. A brief description of these data types is given in Table 3-5.

Table 3-5 Derived data types for component-based control logic

Type Name	Description	Constituent Elements
Component	STRUCT for describing control-related info of a component.	Name, Type, ID, Working State ID, index, etc.
State	STRUCT for the state of STD	Name, Type, ID, index, etc.
Transition	STRUCT for the transition of a state	ID, destination state ID, index, etc.
Condition	STRUCT for the condition of a transition	ID, operator, related state ID, etc.
Error	STRUCT for describing error messages	ID, error description
ActComponent	STRUCT for data models used to generate a row on HMI panel for controlling an actuator component	ComponentID, component name, position1, position2

3.8.1.3 Development of PLC Platform-Specific Elements

For the development of the platform-specific information, a reverse engineering process was used, as illustrated in Figure 3-23. For this purpose, a template of the PLC project is created in the PLC programming tool and the source code is exported. The information contained in the source code can be classified into reusable common information and project-specific information. The reusable common information is then decomposed into different elements and saved in the database as structured information. This reusable common information is used during the control software deployment phase to generate the control systems software.

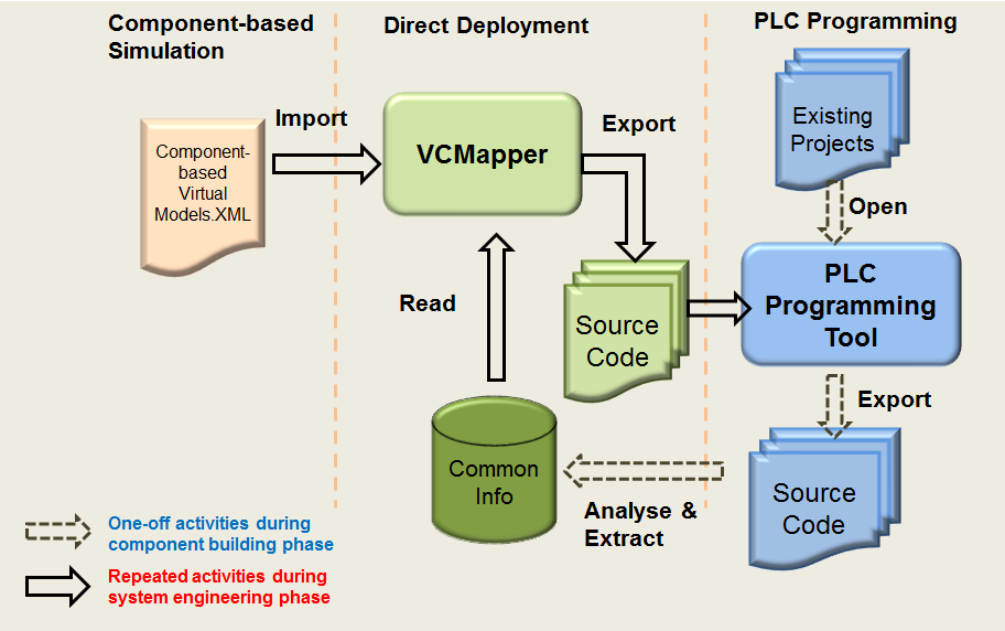


Figure 3-23 Reverse engineering process of direct deployment

The reusable common information typically includes:

Project header information: Most PLC programs require header information. However, some platforms, such as STEP 7, have no uniform project header information. This is because STEP 7 project is exported into multiple independent text files, instead of one single project file.

Software blocks header information: Software blocks refer to the objects that compose a control software project. According to the structure specification in IEC61131-3, software blocks include data types, program organisation units (POU), variable tables, instances and configurations. However, IEC61131-3 does not specify the data structure of these blocks and as a consequence their implementation is specific to each PLC platform.

3.8.2 System Engineering Phase

3.8.2.1 Generation of System Data Model

The workflow of the System Data Model generation is outlined in Figure 3-24. The System Data Model was generated by translating and describing the control behaviour of control components as instances of the predefined derived data types, as discussed in section 3.8.1.2.

In the translation process, each control component was analysed and translated in order to populate the corresponding control model described using the predefined derived “Component” data type. The states of each component were then translated to populate the corresponding control models described by the derived data type “State”. All the transitions and constituent condition groups were translated into the corresponding control models. Finally, the interlock conditions of dynamic states were translated into control models (similarly to transitional conditions).

The data model for a component and its interfacing is discussed below.

A control model component is represented as:

RCCOMP = {ID, name, type, sindex, stcount, wsid, scid}

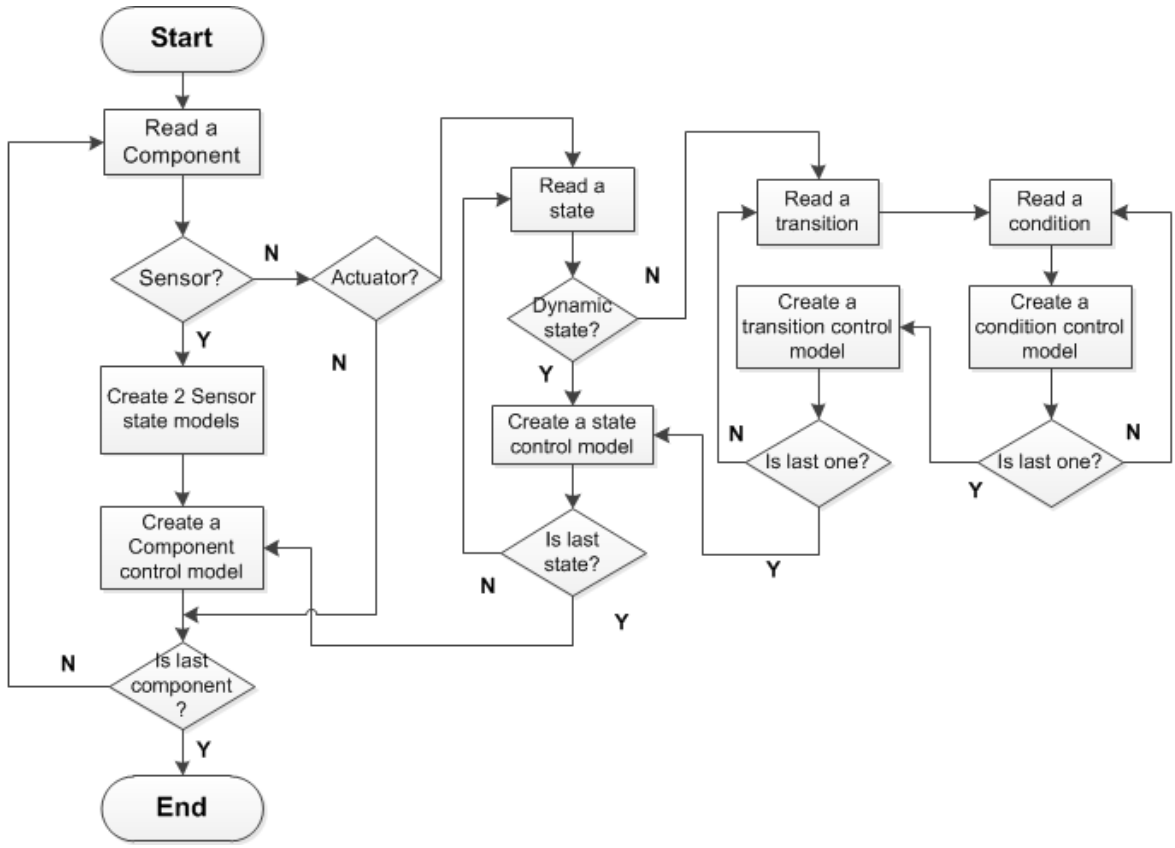


Figure 3-24 Workflow of system data model generation

Of the attributes of the component, the following are static attributes:

- ID: the unique ID of the component.
- name: the name of a component.
- type: the type of a component, which can be “actuator”, “sensor” or “process”.
- sindex: the index number of the first component’s state. The states of all the components in a system are stored in a structured collection of data models. Each state model has an index number for data access and all the states of the same component are saved in order. Therefore, the index number of the first state can be used as data access reference.
- stcount: the number of states a component has.

The attributes “wsid” and “scid” are defined as variables. The attribute “wsid” refers to the current working state ID. It is used to collect the current working state information of the corresponding

runtime actuator component. The "scid" refers to the ID of the state command and is used to send the command to the runtime component. The data models of states, transitions and conditions are monitored by the Logic Engine during the runtime. According to the current working state of each component, the relevant Data Model is updated by the Logic Engine. Real-time state commands are sent to the corresponding Runtime Components in order to drive the physical components.

The attributes "wsid" and "scid" are updated with real-time data at runtime. Therefore, they are set to their initial values during the control model generation process, whose main purpose is to set the values of static attributes for all control models by encoding the virtual control models' corresponding items.

3.8.2.2 Generation of HMI Data Model

As discussed in section 3.7.1.1, HMI Data model is composed of the data models required for manual control for the generation of process monitoring screens. The process of generating data model for process monitoring is same as of System Data Model Generation. The workflow of generating manual mode control models for an actuator component is illustrated in Figure 3-25. As the manual control is only required for actuator components, only the state behaviour of actuator components are analysed. Based on the state behaviour of each component, described as State Transition Diagram (STD), the static and dynamic states are identified and the position pairs are created.

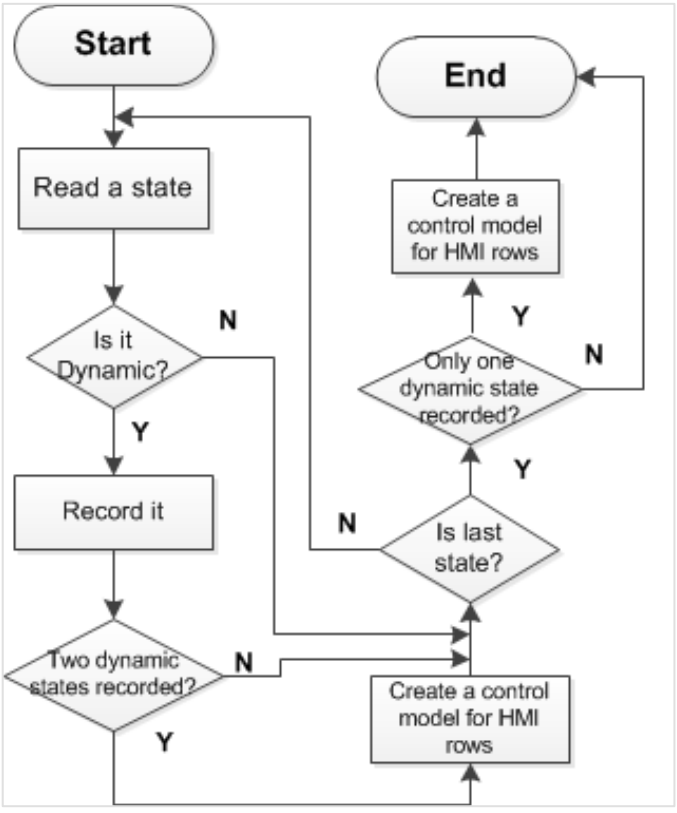


Figure 3-25 Workflow of manual mode control model generation

The position pairs created were used for the generation of manual rows for the HMI (see section 3.9) that controls the movement between two positions. The number of position pairs depends on the number of the dynamic states. The relation between the number of position pairs and the number of dynamic states is given by:

$N_{pp} = N_{ds} - 1$ (if $N_{ds} > 1$)

$N_{pp} = N_{ds}$ (if $N_{ds} = 1$)

Where,

N_{pp} is the number of position pairs

N_{ds} is the number of dynamic states

If the number of dynamic states is lower or equal to two, only one row (two pushbuttons) will be generated. Otherwise, N_{ds}-1 rows are generated.

3.8.2.3 Generation of Logic Repository

The Logic Repository is the hierarchical organisation of control models. The control models are automatically converted into structured data sets using arrays. The size of each array is determined by the number of the runtime control models contained in a system. For example, the array of runtime control model "Component" for a system composed of ten components can be declared in STEP 7 as:

lsaComponent: ARRAY[0..9] of "Component"

Where, 'lsaComponent' is the name of the array and "Component" is the name of the derived data type for describing actuator components.

In order to support the functionalities for both the automatic and manual control modes, the logic repository consists of six arrays of control models described in the derived data types (section 3.8.1.2). The declared arrays are then populated with their respective runtime control data models, which are generated according to the process described in the section 3.5.3.

3.8.2.1 Component Mapping

Component mapping refers to a) the mapping of RCs with respective actuator and sensor components and b) the interfacing of each actuator and sensor component of a system to their corresponding physical I/O addresses. Component mapping is performed manually during the program generation process. Automation of the component mapping is possible. However, due to the limited access to the CCE database during this research the automation of component mapping has not been investigated in this research. This work is the subject of future R&D project such as the EPSRC Knowledge Driven Configurable Manufacturing (KDCM) programme.

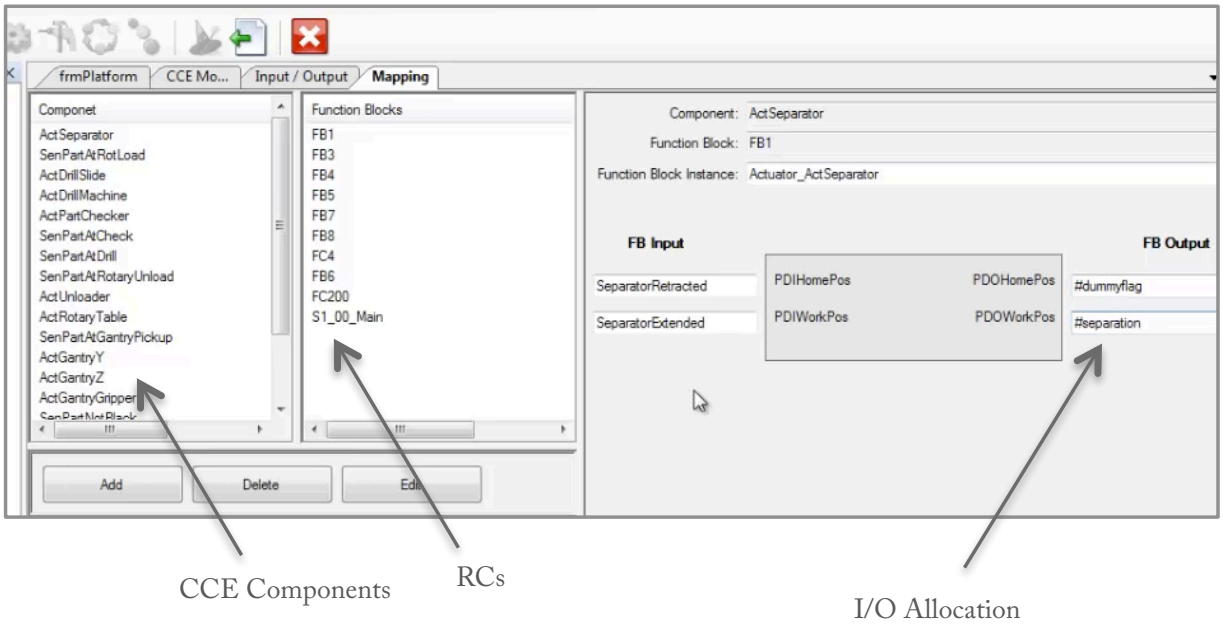


Figure 3-26 GUI for component mapping and I/O allocation

To carry out component mapping in this research, a graphical user interface (GUI) software module was implemented to enable intuitive and effective component mapping. The GUI screen is shown in Figure 3-26. For each CCE component a corresponding RC is selected from the list of available RCs. Once a CCE component and RC is paired, the graphical representation of the RC appears on the screen with I/O pins. I/O address is then allocated to each I/O pin.

The author was heavily involved in the requirement specification and design phase of this software module. Details of the software implementation carried out by other member of the research group can be found in [128].

3.8.2.2 Generation of POUs

In IEC 61131-3, blocks from which programs and projects are built are known as POUs (Program Organisation Units). There are three types of POUs in a PLC project: Function, Function Block and Program. The number of the POUs to be generated in a project depends on the PLC platform-specific software structure.

RCs and Logic Engine appear as FBs in the control software. Therefore, one POU is created for each instance of the RC and the Logic Engine. The number of instances of the RCs depends on the number of Sensors and Actuators components.

In this thesis the POUs generation refers to the generation of the control logic for Logic Engine and RCs (for actuator and sensor components). As mentioned in section 3.7.1, all these components are pre-defined and stored in a library. The Logic Engine only appears once in a program while the number of instances of RCs of sensor and actuators depends on the number of sensor and actuator components in a system. For each instance of these components in a program, a POU is required.

POUs for Logic Engine and RCs are created by calling the instances of these pre-defined components from the RC library.

As mentioned before, the logic engine appears in a program as a function block without any I/O mapping parameters. This is because the communication of the Logic Engine with other program elements is complemented via internal indirect parameterisation. Thus the POU for the Logic Engine does not require the I/O interfacing. On the other hand, RCs for sensor and actuators communicates to other system components via direct parameterisation and thus require I/O mapping for POU generation. The I/O mapping of physical addresses of these sensors and actuators is carried out manually during the aforementioned component mapping phase. Other communication parameters are mapped automatically to corresponding runtime control models by a mapping function.

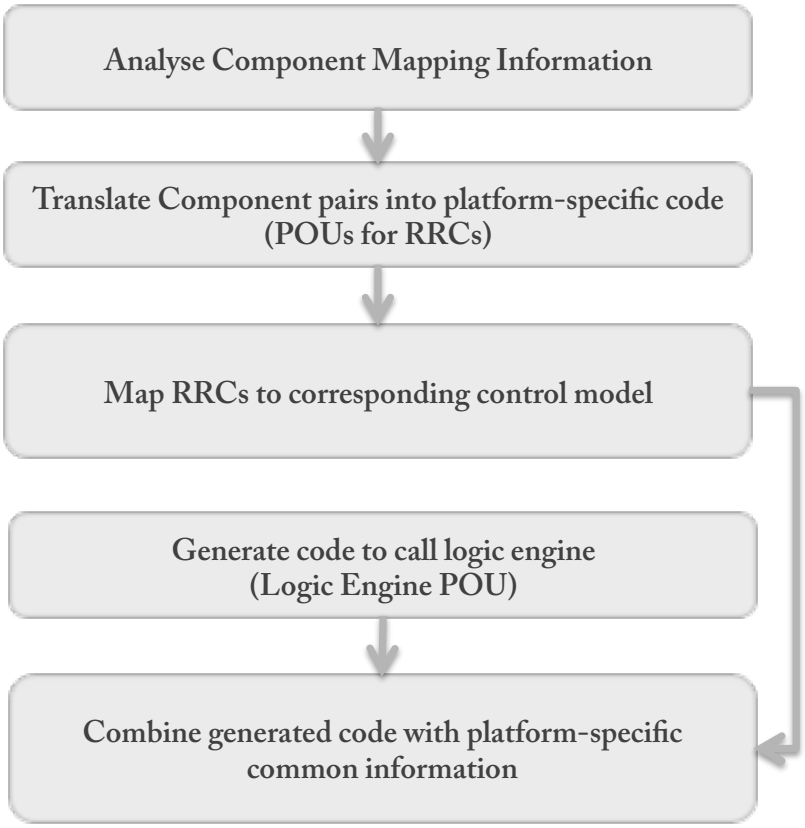


Figure 3-27 Process of generating POUs

The workflow of POUs generation is outlined in Figure 3-27. Firstly, the I/O mapping data and the RC mapping data for all the actuator components and sensor components are analysed and processed. According the identification number (CID) of each component included in the mapping data, the corresponding runtime control model is identified and then the I/O parameters, which should be connected to corresponding related attributes of its control runtime model, are mapped automatically. For actuator components, the respective I/O parameters for reporting working state (swid), sending state command (scid), resetting (reset) and reporting error (alarm) are mapped. For sensor components, only one parameter for reporting status needs to be mapped. Lastly, the Logic Engine function

is called automatically.

In addition to the declaration code specific to control platforms, some additional blocks may need to be created for calling RCs. For example, in Step 7, an instance data block needs to be created when a RC is called. To address this, during the software generation, functions are provided to automatically generate these platform-specific blocks.

3.8.2.3 Generation of the Complete Source Code

The last step in the PLC program generation is the dynamic generation of the complete source code of the control software. This process is shown in Figure 3-28. The complete source code generation is the combination of the source code of the I/O variable table, User defined Data Types (UDTs), Logic Depository and POUs. This process involves the population of the platform-specific template of the PLC template project with source code for these elements. The format and the number of source code files depend on the PLC platform. For example, for STEP 7 a number of text files are generated, while for Unity Pro only one XML file of source code is generated.

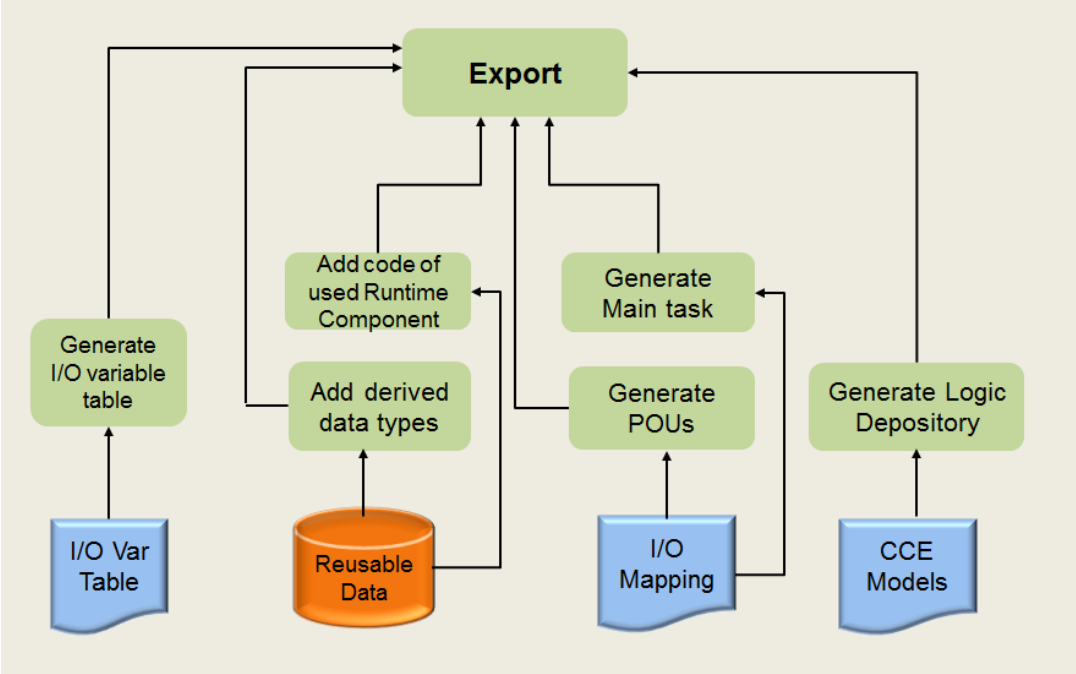


Figure 3-28 Complete source code generation

3.9 Generation of HMI Screens

In this research, two methods were initially evaluated for screen generation, namely static and dynamic approaches. The static approach refers to the automatic generation of source code for the HMI application, whereas the dynamic approach refers to the automatic population of generic screen templates from the PLC code at runtime. Prototype screens for both static and dynamic approach were developed using SIMATIC WInCC to evaluate the strengths and weaknesses of both approaches. Due to numerous benefits of dynamic approach over static approach, the dynamic template-based

approach was favoured. A comparison of the two methods is given in Table 3-6.

Table 3-6 Comparison of static and dynamic approach for HMI screen generation

Characteristics	Static Approach	Dynamic Approach
Openness	✗	✔
Data Transfer	✗	✔
Program Generation Effort	✗	✔
Complexity	✗	✔
PLC Scan Time	✔	✗
PLC Program Memory	✔	✗

✔ represents advantage over the alternative method

The main objective of HMI screen generation is to automate the process of generating machine-specific screens. The standard screens are pre-designed and embedded in the HMI software while the machine specific screens are generated with the help of system components within the HMI software from the information derived from the PLC software during runtime. Thus, the HMI software remains consistent for every machine cell.

3.9.1 Manual Mode Screen Generation

The HMI screens for the manual mode control are system specific, and thus unique for each manufacturing cell. Typically, rows of two pushbuttons are provided on the manual mode screens for each actuator. Using these pushbuttons the operator can control a machine by driving the actuators independently.

For the manual screen generation a template screen, consisting of screen objects and tags, was developed as shown in Figure 3-29. Based on the HMI screens specifications of Ford Motor Company, the template was composed of five pushbutton rows. The number of instances of the template depends on the number of position pairs in the HMI Data Model. As the template consists of five rows of pushbuttons, the unutilised rows are hidden from the user. To map the tags of the template with the HMI Data Model a ‘mapper function’ has been developed, which automatically maps the tags in the template with the corresponding position pairs in the HMI Data Model.

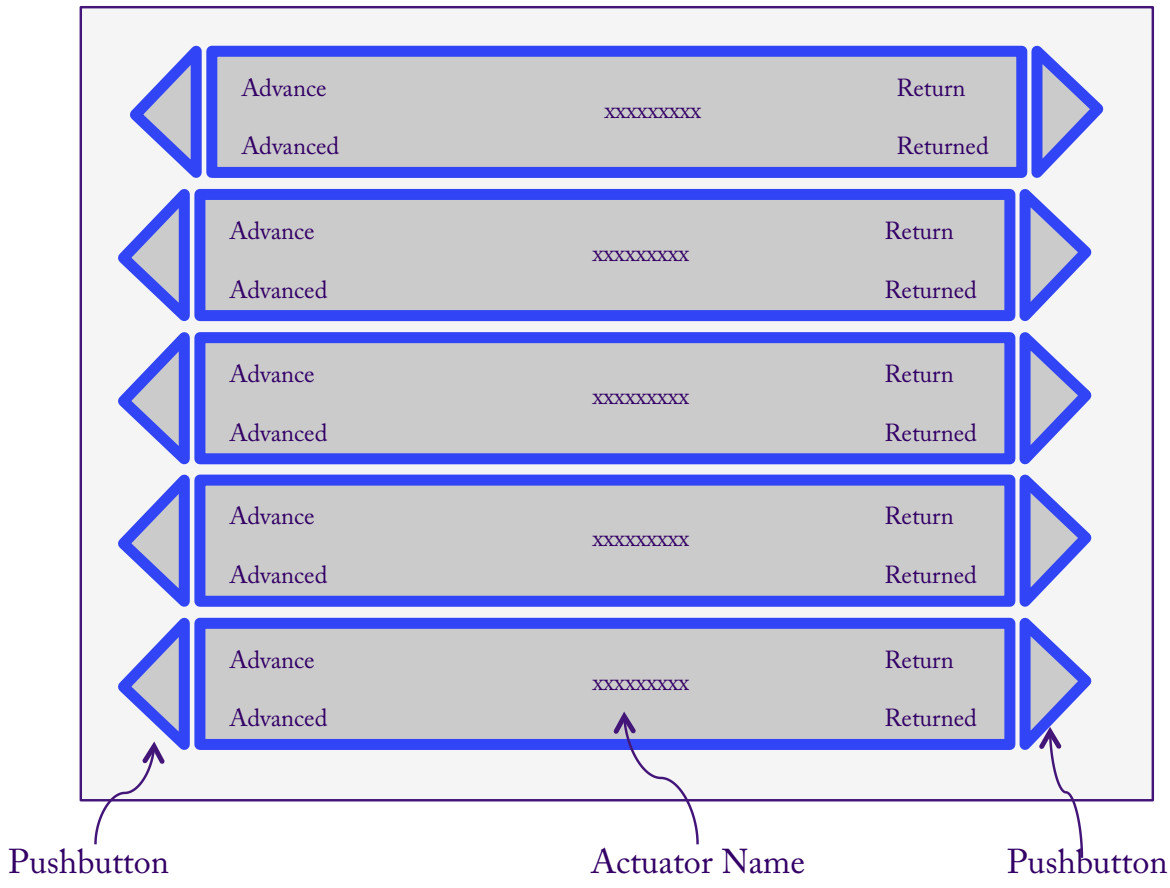


Figure 3-29 Manual mode screen template

The workflow of the mapper function is shown in Figure 3-30. The offset calculation for indexing variables during mapping is critical during the screen generation. The mapper function is divided into three sub-functions. One function is to set the index variables to default values. The default values are the addresses of the data array elements of first five actuators in the HMI Data Model. The second function adds a offset to each of the indexing tags to show the next five actuators. The offset is defined by the length of the UDT used for the Data Model and the number of manual rows that are used on one screen. The third function decrements the tag values by a particular offset. The first function is called whenever the manual screens are requested from the HMI. The other two functions are used for navigation of manual screens if more than one screen are required for a system.

3.9.1 Control Logic Monitoring Screen Generation

The control logic monitoring screens provide functionality to show the live view of the control logic (STDs), which is defined using the CCE tools. The control logic monitoring screens are categorised into Process Logic monitoring and Actuator monitoring screens.

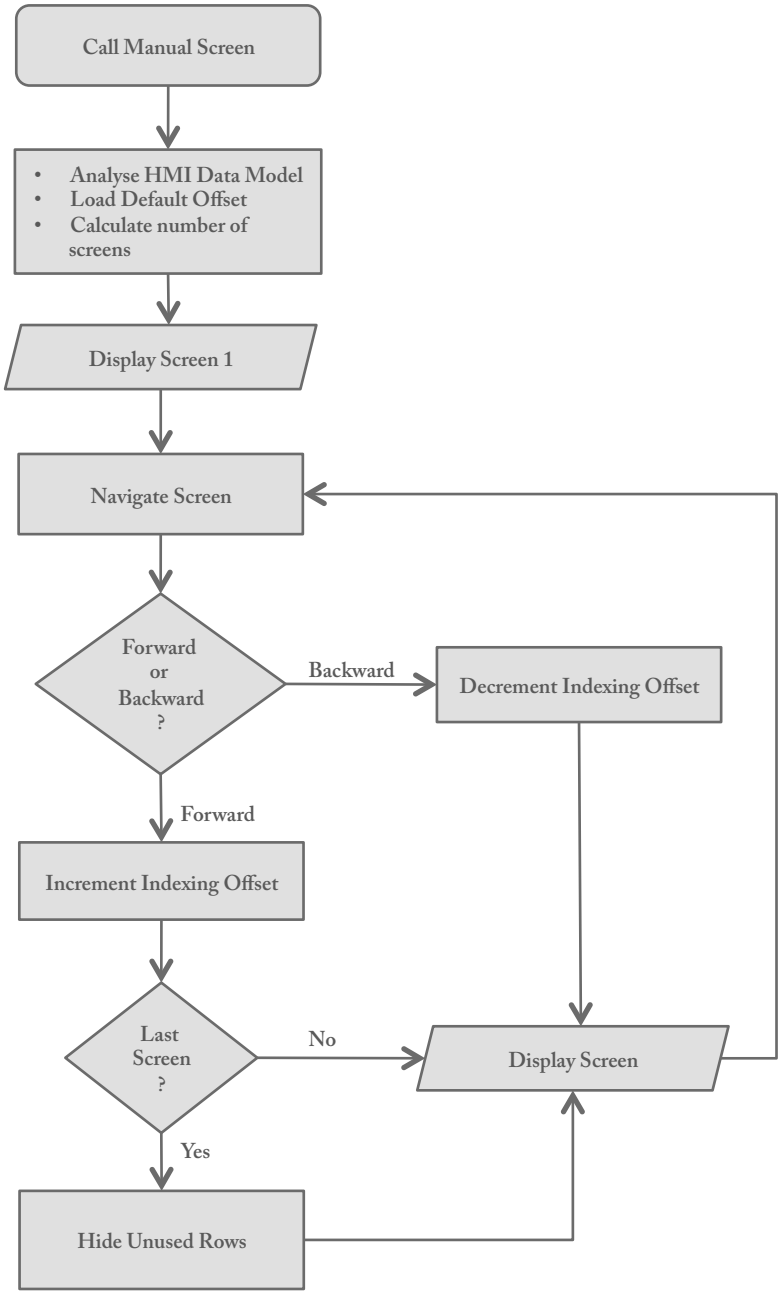


Figure 3-30 Workflow of mapper function for manual mode screens navigation

3.9.1.1 Process Logic Monitoring Screens Generation

As aforementioned in section 3.5.5, Process Logic is used to define the sequence of operations of a machine. The live view of the Process Logic STDs gives technicians an overview of the machine sequence. The Process Logic STDs are translated into a data model and are stored in the HMI Data Model in the PLC program. To generate Process Logic monitoring screens, a dynamic template-based approach has been used. To simulate the template, a mapping function has been developed to map the corresponding runtime data model on the PLC with the objects in the template.

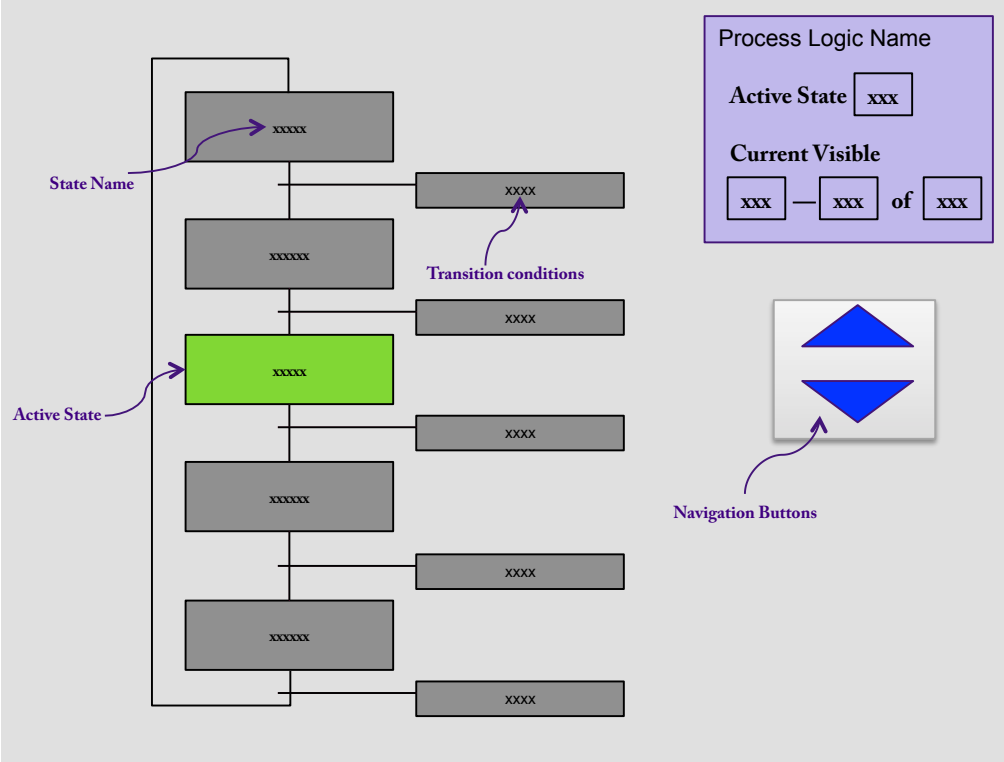


Figure 3-31 Template of Process Logic monitoring screen

Figure 3-31 shows an example template created for the online moitoring of the STDs. The colour of the active state changes from grey to green. The template consists of five states and five transitions. Since, the number of states in process logic varies, the visibility of the objects is modified dynamically and the tags for the text fields are automatically indexed. On the right-hand side of the template the two navigation buttons are provided to navigate screen up and down in case if process logic has more than five states.

3.9.1.2 Actuator Monitoring Screens Generation

Actuator monitoring screens shows the runtime visualisation of the STDs of actuator components and their corresponding RCs. The actuator monitoring screen is generated with the help of visualisation objects, object tags and a mapper function. The mapper function dynamically maps the object tags of the template screen with the corresponding runtime data model within the PLC program.

The screen template used for actuator monitoring is shown in Figure 3-32. The STD in the template consists of five states and three transitions. Since, the number of states of an actuator can vary, the visibility of objects is dynamically modified and the tags are multiplexed. At the bottom of the template navigation buttons are provided to navigate through the STDs if an STD consist of more than 5 states. In the STD, the active state is highlighted in a green colour.

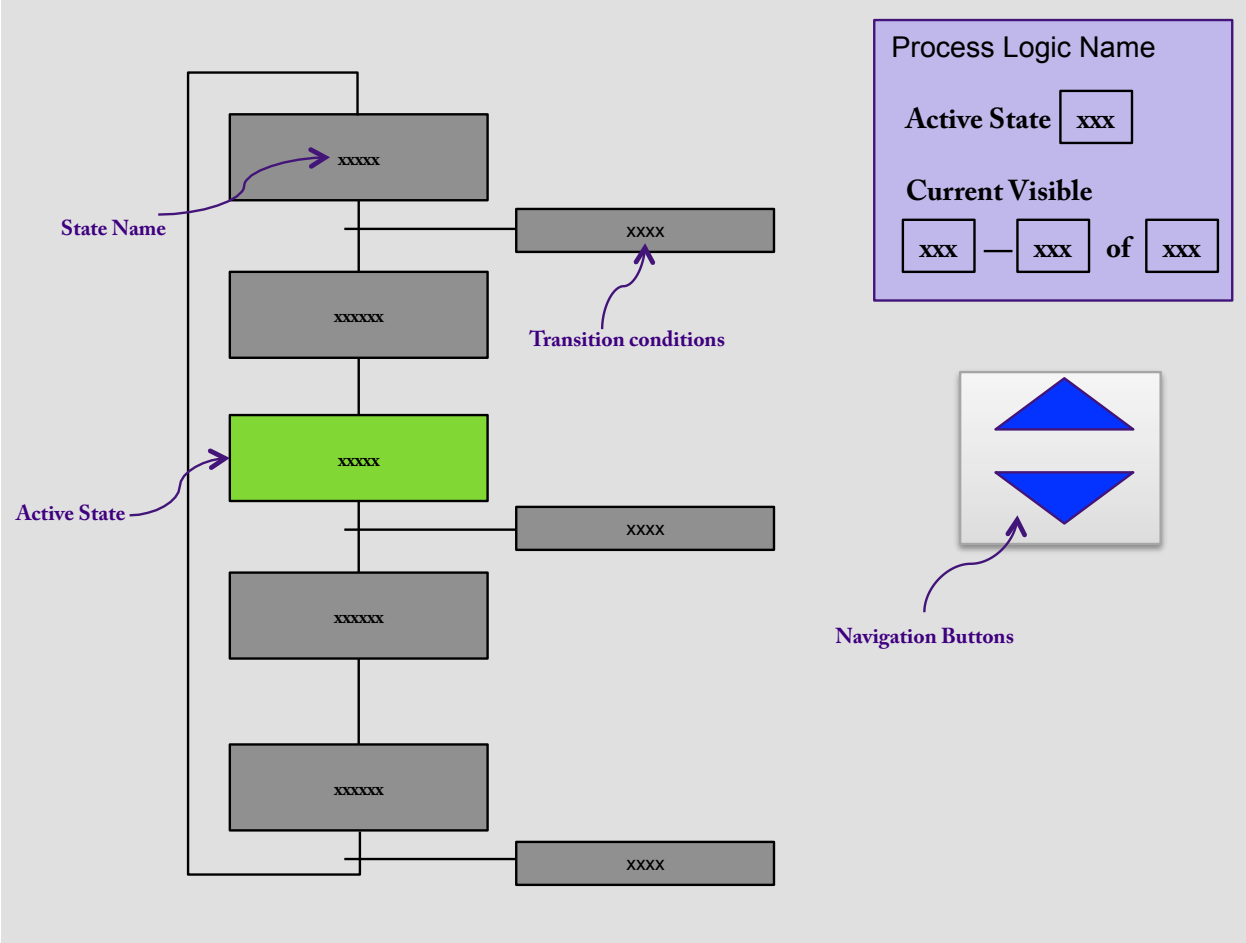


Figure 3-32 Example of actuator monitoring screen template

To show the online status of the RCs, the instance data of the RC (which consists of the current status of inputs and outputs) is mapped with the graphical representation objects of the RC in the screen template.

3.9.2 Alarm Handling

As stated in section 3.7, the faults and warning messages are stored in a Fault Management Data Model. When a fault or warning is triggered, Logic Engine locates the corresponding message in the Data Model and copies it into the fault accumulator. If there is more than one fault message only the high priority fault is processed. The fault accumulator stores the active fault as well as the fault history.

A schematic diagram of the fault handling is shown in Figure 3-33. To display fault messages, a text banner is provided on the 'home screen' template. The Alarm Handler continually scans the 'fault accumulator' within the PLC program and displays the active fault on the Home screen. The Alarm Handler also provides a function to acknowledge active fault to resume machine operation. Alarm Handler also provides function to show the fault history. The alarm history screen displays the ten most recent faults.

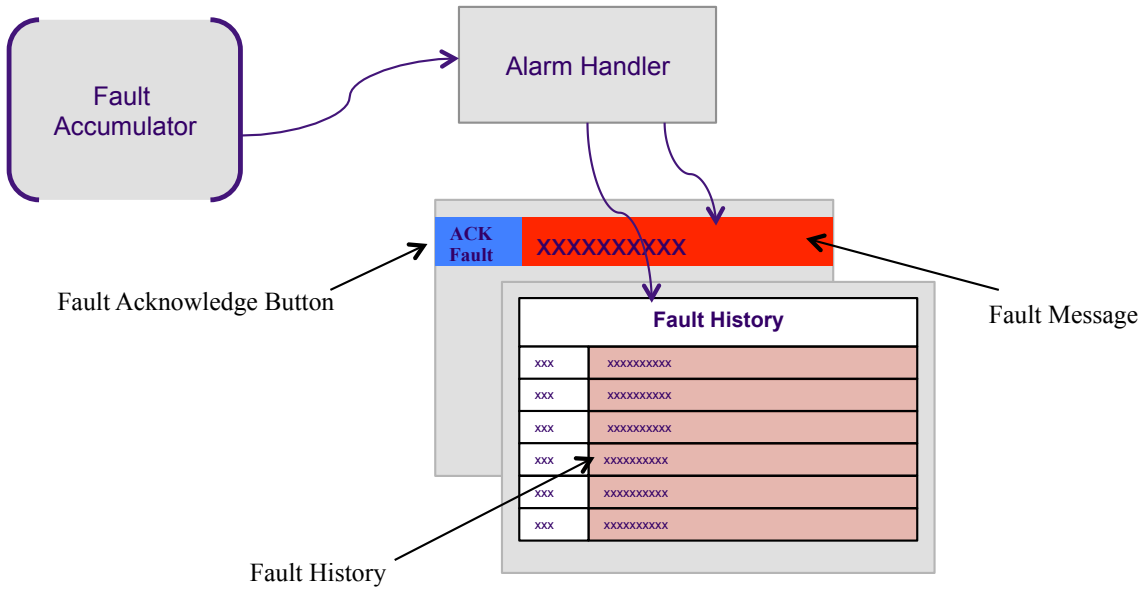


Figure 3-33 Schematic diagram of fault handling

3.10 Summary

This chapter has presented a framework and control software architecture that enabled the automatic generation of control software from the control behaviour defined in 3D-based manufacturing process planning tools. The functional and structural industrial requirements of the control software were identified in detail in order to propose a realistic approach. The method of defining control behaviour within the Virtual Engineering Environment (i.e. CCE tool) has been redesigned to enhance the control information and hence enable the generation of a complete control code. A novel control software architecture has been designed that enables the direct deployment of both the PLC control code and HMI screens. This new architecture is vendor-independent and can be adopted for any control platform.

To design the methodology for the implementation, the proposed PLC software architecture was broken down into reusable, dynamic and platform-specific elements. The reusable elements consist of RCs and the data structure of the Control System Data Model. The reusable elements are manually developed and stored in a library. To develop the platform-specific elements, a reverse engineering approach has been used. This involved the writing of template software in a PLC programming tool and then analysing the data structure of the source code. The dynamic elements were developed from the control logic defined within the CCE tools. The process of generating complete and executable source code from these software elements was described.

The methodology of generating machine-specific HMI screens was also presented in this chapter. Two methods (i.e. static and dynamic approaches) for the implementation of the HMI screens were initially considered. However, dynamic approach was selected for the implementation due to its inherent benefits. The dynamic approach is essentially a template-based approach that uses a mapper

function to dynamically populate the screens from the HMI Data Model at runtime. The generated interface consists of manual mode control screens, process monitoring screens and alarm screens.

4 Case Study and Evaluation

4.1 Introduction

The purpose of this chapter is to demonstrate the feasibility, features and performance of the proposed VE based automatic code generation approach based on the use-cases to evaluate the proposed approach from various perspectives, such as application development time, reconfiguration time and runtime performance. The control code was generated for three PLC platforms (i.e., Siemens SIMATIC STEP7, Unity Pro and PLCopen) using the code generation approach presented in Chapter 3. To test the generated control code three different test beds were used, namely Festo test rig, Automation System Demonstrator (ASD) machine and MTC test loop (based at Manufacturing Technology Centre, UK). Each of the application use-case required a physical automation system, virtual modelling of the system in the CCE tool, and development of RCs. A brief description of the test beds is given below.

The Festo test rig, shown in Figure 4-1, used for the use case is a modular automation system designed for educational training. This small-scale rig was used for the initial development as a proof-of-concept to ensure the validity of the proposed concept with a minimum of safety issues as compared to a full-scale machine.

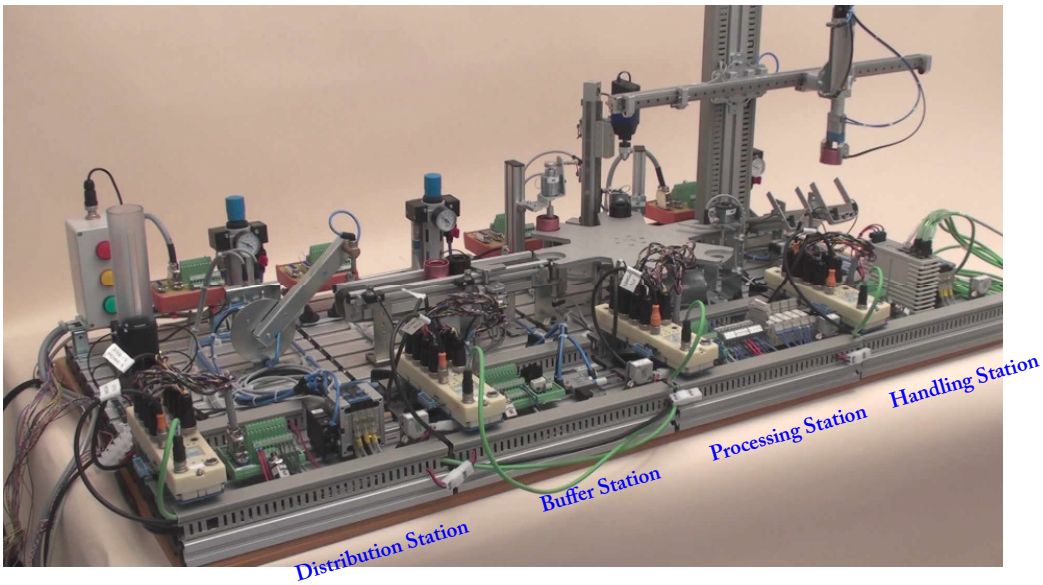


Figure 4-1 Festo test rig

The ASD machine is a multi-station machining and assembly automation system, designed and build by Ford Motor Company in collaboration with its OEMs and control vendors. The machine uses automation equipment from various automation suppliers and is based on traditional master-slave control system architecture. The machine represents a small-scale model of engine assembly line consisting of 12 automatic workstations. The engine blocks are placed on a pallet by a robot and are

transported on a conveyor to workstations to perform various manufacturing processes such as profiling, assembly of engine block and head, nut running, air wash and inspection. Each workstation has its own mobile plug-and-play type control unit.

The MTC test loop is built by ThyssenKrupp Krause System Engineering GmbH, which features a robotic pick and place station, an IMS+ lift-rotate station and a manual assembly station. For code generation only IMS+ station was targeted. This type of station has been used in a number of automotive engine assembly lines by various automotive industries, including Ford Motor Company.

The test beds used cover a wide range of automation equipment and control system architecture used in industry and thus gave an understanding of the strengths and limitations of the proposed control system engineering approach in various scenarios. The features of each test bed from control perspective are highlighted in Table 4-1.

Table 4-1 Features of test beds used for use-cases

Test Bed	Control Hardware Vendors	PLCs and HMIs	Other Control Hardware
Festo Test Rig	▪ Siemens ▪ Schneider Electric	▪ SIMATIC ET 200s PLC ▪ SIMATIC MP277 8" Touch HMI ▪ Modicon TSX P57 PLC ▪ Magelis Advanced XBT HMI	▪ Pneumatic actuators ▪ Electrical actuators ▪ Proximity sensors ▪ Safety devices
ASD Machine	▪ Siemens ▪ Schneider Electric ▪ ABB ▪ Bosch Rexroth ▪ Allen-Bradley	▪ SIMATIC S300 PLC ▪ SIMATIC MP277 8" Touch HMI ▪ Modicon TSX P57 PLCs ▪ Magelis iPC ▪ IPC System 200 ▪ BTV20 HMI	▪ Pneumatic actuators ▪ Hydraulic actuators ▪ Electrical actuators ▪ Servo drives ▪ Robot ▪ Vision system ▪ Barcode scanner ▪ Proximity sensors ▪ Proximity, temperature, pressure, current and vibration sensors ▪ Safety devices
IMS+ Test Loop	▪ Siemens ▪ ABB	▪ SIMATIC ET 200s Pro PLC ▪ SIMATIC MP277 10" Touch HMI ▪ SIMATIC MP277 8" Touch HMI	▪ Pneumatic actuators ▪ Electrical actuators ▪ Servo drives ▪ Robot ▪ Proximity sensors, pressure and current sensors ▪ Safety devices

In order to enable a one to one comparison of the automatic control code generation method with the traditional manual programming practice, as required for this thesis, the Festo test rig was selected.

For comparison of the generated programs with the state of the art programming practice, the test rig utilised was programmed to the Global Sigma programming structure specified and used by the Ford Motor Company. Further to this, the reconfigurability of both programming approaches was assessed by changing the physical configuration of the Festo test rig stations.

4.2 Case Study

The Festo test rig used for the case study is shown in Figure 4-1. This type of test rig is used by Ford Motor Company to test new control systems and to train their staff. The rig provides a realistic automation problem and thus effective testing and evaluation of the runtime system may be achieved.

The test rig is composed of four stations, i.e. Distribution Station, Buffer Station, Processing Station and Handling Station. Each station is composed of various kinds of components. A taxonomy of the components of the test rig is given in Figure 4-2. A complete list of components of each station is given in Table 4-2.

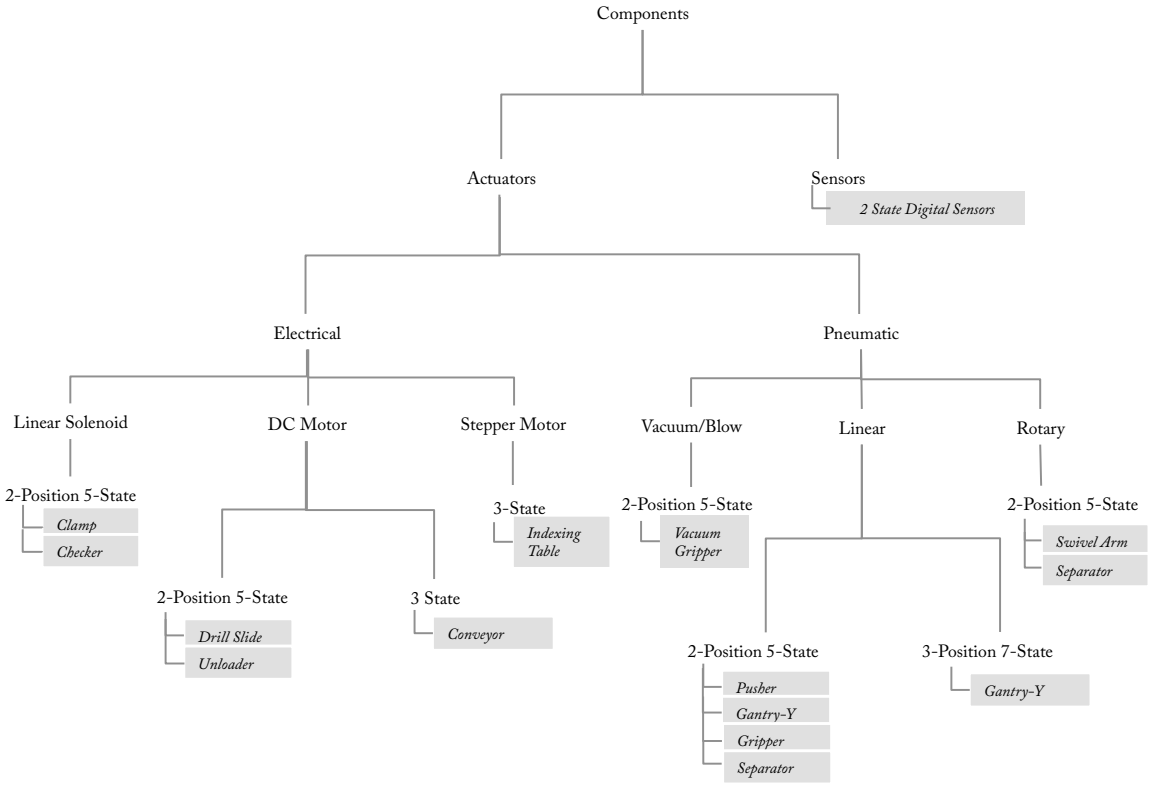


Figure 4-2 A taxonomy of components of Festo test rig

The basic operation of the test rig is to convey a workpiece from one end of the machine to the other while performing a number of operations such as gripping, transferring, indexing, clamping, drilling and gauging. An important characteristic of the system is that several operations have to occur simultaneously on various workpieces passing through the system.

Table 4-2 List of sensors and actuators of the Festo test rig

Stations	Components				
	Sensors/Actuators	Type	Digital Inputs	Digital Outputs	Positions/States
Distribution Station	Pusher	Pneumatic actuator	2	1	2
	Part in Magazine	Digital Sensor	-	-	2
	MagXfer	Mechanical switch	-	-	2
	Swivel Cylinder	Pneumatic actuator	2	2	2
	Vacuum Gripper	Pneumatic actuator	1	2	2
Buffer Station	Separator	Pneumatic actuator	2	2	2
	Conveyor	Electrical actuator	0	1	2
	Part at Start	Proximity sensor	-	-	2
	Part at separator	Proximity sensor	-	-	2
	Part at end	Proximity sensor	-	-	2
Processing Station	Checker	Electrical	2	1	2
	Drill slide	Electrical	2	2	2
	Drill Machine	Electrical	0	1	2
	Clamp	Electrical	2	1	2
	Unloader	Electrical	2	1	2
	Indexing Table	Electrical	0	1	2
	Part at Entry	Proximity	-	-	2
	Part at Check	Proximity	-	-	2
	Part at drill	Proximity	-	-	2
	Part at unload	Proximity sensor	-	-	2
Handling Station	Gantry Z-Axis	Pneumatic actuator	2	2	2
	Gantry Y-Axis	Pneumatic actuator	3	2	3
	Gripper	Pneumatic actuator	2	2	2
	Part Not Black	Reflective sensor	-	-	2
	Part Available	Proximity sensor	-	-	2

In this case study a Siemens SIMATIC S300 PLC with distributed I/O modules and a SIMATIC MP277 8" Touch Panel was chosen as the control hardware. For runtime communication, PROFINET was used to connect the devices. The software tools used were SIMATIC STEP7 V5.4 for PLC programming and SIMATIC WinCC Flexible 2008 for HMI development.

The test rig was programmed with both automatic control code generation method and traditional manual programming. Only the details of automatic control code generation method are presented here.

As explained in detail in section 3.8, the automatic code generation process can be broken down into the pre-engineering phase and the system engineering phase. The pre-engineering phase is composed of RC development and control code template development. This phase is performed once and the da-

ta generated and stored in the CCE Mapper library is application independent. This data is reused during system engineering phase. The tasks conducted during the system engineering phase are specific to the targeted system. A brief description of the tasks performed during pre-engineering phase and system engineering phase to program the Festo test rig is given in the following sub-sections.

4.2.1 Pre-Engineering Phase

4.2.1.1 RC Library Development

In the automatic code generation approach, each actuator and sensor component requires RC, which is a pre-validated resource specific generic FB. For sensors, the design of an RC depends on the number of states. While for actuators, it depends on the number of states and driving power (such as pneumatic or electrical) of actuator component. If the numbers of states and driving power are similar then the same RC can be used for mechanically dissimilar actuators. For instance, in the Festo test rig, the component Pusher and the component Swivel Arm, which are both actuated by pneumatic power and have five states and two positions, use the same RC (id: LFB_HP_2P5S_2I2O).

Similarly, because the test rig comprises only 2-state digital sensors, only one type of RC (LFB_SEN) was needed. For the actuator components, eight RCs were developed in total. Each RC was developed and tested individually in STEP 7 using SCL programming language. The source code of these RCs was exported and saved in the RC Library of the CCE Mapper tool. A list of the developed RCs and their corresponding sensor and actuator components is given in Table 4-3.

Table 4-3 Runtime components for the Festo test rig

RCs	Comments	Related Components
LFB_SEN	For all the digital 2-state sensors	All the 15 sensor components
LFB_HP_2P5S_2I2O	For 2-position 5-state pneumatic actuators	Eject Cylinder, Swivel Arm, Unloader, Gantry-Z, Gantry Clamp, Gantry Gripper and Separator
LFB_SwiGripper_2P5S_1I2O	For pneumatic grippers	Pneumatic Gripper
LFB_ED_2P5S_2I2O	For 2-position 5-state electrical actuators	Drill Slide
LFB_PartChecker_2P3S_1I1O	For 2-position 3-state electrical actuators	Part Checker
LFB_RotTable_3S_1I1O	For indexing table	Indexing Table
LFB_ED_2P2S_0I1O	For two state electrical actuators	Drill Machine
LFB_HP_3P7S_3I2O	For 3-position 7-state pneumatic actuators	Gantry-Y

4.2.1.2 PLC Program Template Development

The PLC program template serves two purposes: 1) it defines the structure of the software and 2) it is used for extraction of the platform-specific common information for the generation of the PLC software. The program templates are developed once and then imported into the database of CCE Mapper

to be reused for the code generation of any machine application on that PLC.

SIMATIC STEP 7 programs are based on a proprietary data structure. Therefore, the common information templates were created using a reverse engineering approach. An example project was first created in STEP 7 and the source code of the parts of the project was then exported as text files. The source code within the text files was then analysed and the required S7-specific templates were created and stored in the CCE Mapper library. The created templates are listed in Table 4-4.

Table 4-4 Components of STEP 7 templates

Name	Dynamic data	Description
UDT	None	User-derived Data Types for describing control models
Instance Data Block	Definition of variables	A data block for an instance of a FB
Shared Data Block	Size of arrays, Main Body of arrays	A shared data block for storing runtime control models
Organisation Block	Name, Main Body	The organisation block of the project

4.2.2 System Engineering Phase

4.2.2.1 Virtual Modelling of the Festo Test Rig

According to the proposed workflow, the test rig was first virtually prototyped and commissioned using the CCE tools. Conforming to the architecture of the component-based approach, each station of the rig was decomposed into separate components. The detail of the decomposition of the test rig is given in Table 4-2. The virtual model of the test rig was built by modelling and assembling these components in the CCE tool. The virtual prototype of the test rig is shown in Figure 4-3.

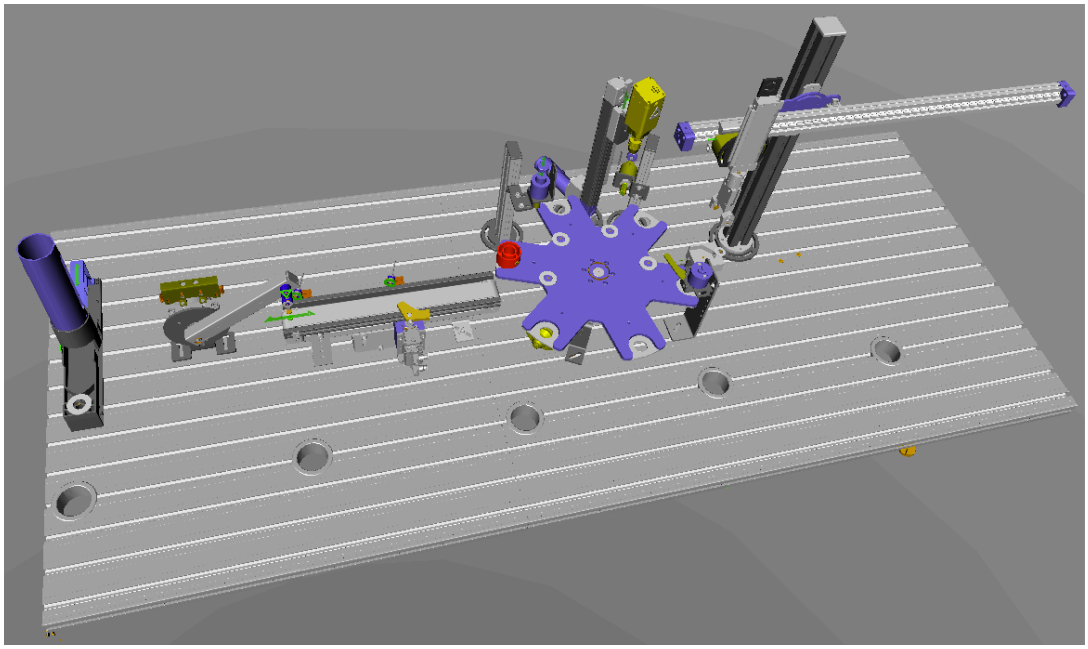


Figure 4-3 Virtual prototype of the Festo test rig

For illustration, the virtual modelling of Distribution Station is described in the remainder of this section. As the focus of this thesis is the control logic generation, the virtual modelling is therefore mainly explained here from the control behaviour definition perspective.

A. Process Description of Distribution Station

In order to define the control logic for the Distribution Station, it is important to have a description of the station's process. The Distribution Station consists of Magazine, Pusher, Swivel Arm, Vacuum Gripper and two proximity sensors. The Magazine stores parts and presents them to the Pusher. The Pusher delivers parts from the Magazine to the Swivel Arm pickup position. The Swivel Arm picks up a part and transfers it to the Conveyor of Buffer Station. One of the proximity sensors (Part in Magazine) indicates the presence of a part in the magazine, while the other proximity sensor (MagXfer) indicates the presence of a part for swivel arm pickup. The flowchart of process sequence is shown in Figure 4-4.

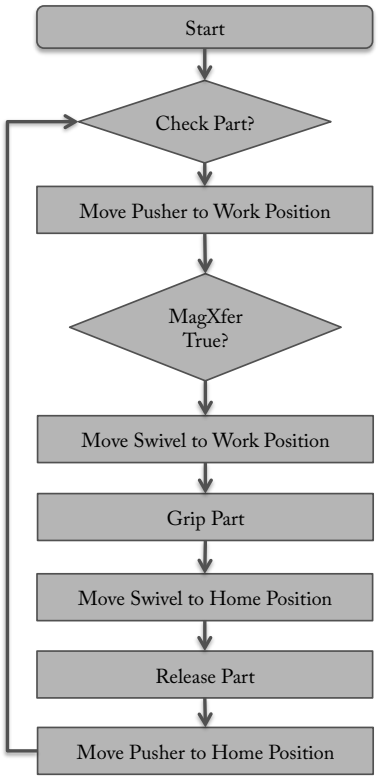


Figure 4-4 Sequence of operations of Distribution Station

B. Component Logic Definition

The Distribution Station consists of three control components, i.e. three Actuator components and two Sensor components. The control behaviour of the Sensor components is described using a two-state STD. The actuators (Pusher, Swivel Arm and Vacuum Gripper) have quite different mechanisms, but the control behaviour of these actuators is similar, i.e. all three actuators have two static states and two dynamic states. The control behaviour of these actuators was defined with a five-state STD. As an ex-

ample, the virtual model and STD of the Pusher component is shown in Figure 4-5.

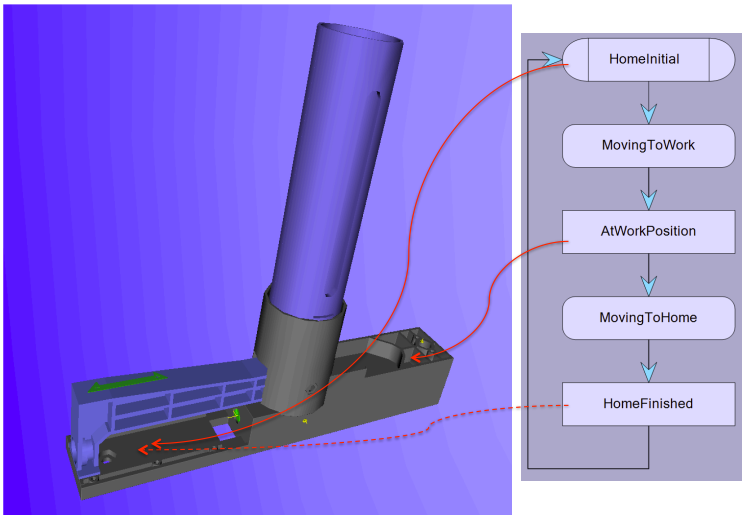


Figure 4-5 Example of an actuator component – Pusher

C. Process Logic Definition

The operation of the Distribution Station was defined using Process Logic sequences, i.e. Process Pusher and Process Swivel. Process Pusher controlled the sequence of operations of the Pusher and Process Swivel controlled the movements of the Swivel Arm and the Vacuum Gripper. The Process Logic sequences also communicate with each other using 'Release' signals for to ensure safe operation.

The STDs of the Process Logics, actuators and sensors for the Distribution Station are shown in Figure 4-6. The sequence checks are represented with the help of arrows.

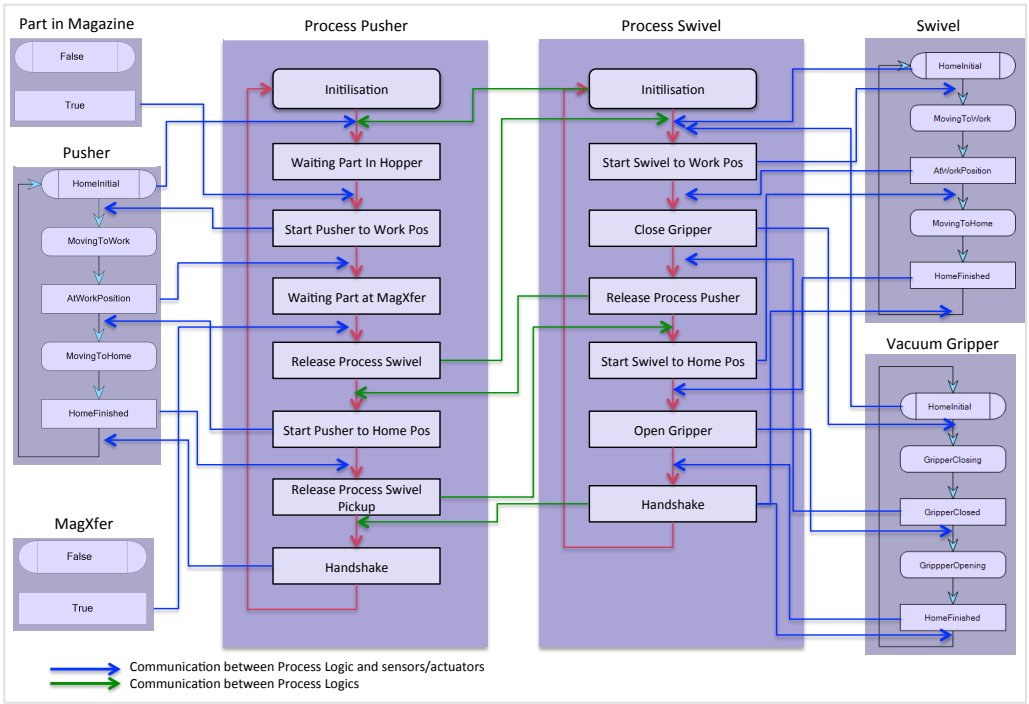


Figure 4-6 Definition of sequence of operations of the Distribution Station using Process Logic

D. Interlocking of Actuator Components

The final step of the control behaviour definition was the interlocking of components states to other components to avoid mechanical clashes during operation. The interlocking of the components of the Distribution Station is illustrated in Figure 4-7.

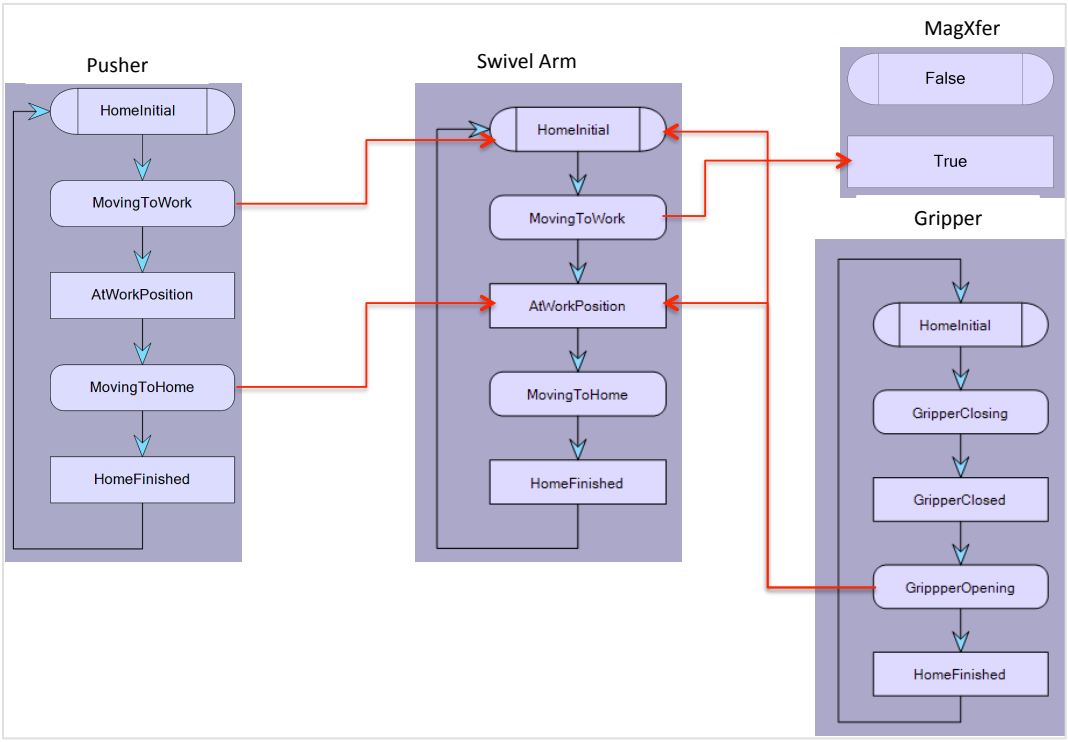


Figure 4-7 Interlock conditions for Distribution Station

4.2.2.2 Control Code Generation

The workflow and the dataflow of the control logic generation are shown in Figure 4-8. The only manual work required during the control code generation phase was mapping of components with RRs and mapping of the physical I/O addresses.

To generate the control code for the test rig, the virtual model of the test rig (XML file) and I/O variables, RCs and platform-specific templates were imported into the CCE Mapper. For component and I/O mapping, the CCE Mapper displays all resource components (i.e. sensors and actuators) and RCs on the screen, as shown in Figure 4-9. For mapping sensor components only the I/O variables were allocated to the RC interface. For actuator components mapping, each actuator component and its corresponding RC was selected and added to the system one by one. Once pairing of actuator components and corresponding RCs has been completed then the I/O variable mapping is carried out then the physical I/O address mapping is carried out.

After the mapping was completed, the code generation functions of the CCE Mapper were executed to generate the source code with the help of 'Generate Code' user interface button. The generated code

was exported as a source code file. For SIMATIC STEP 7, the source code was exported into multiple plain-text files. The information of exported files is outlined in Table 4-5. As shown in the table, different items are represented in different programming languages and saved in different data formats. This is one of the reasons why the source codes for Step 7 must be exported into multiple files. Another important reason is that these items must be compiled in a specific order as listed in the table. This is because of the dependency relationship between these items. For example, the instance DBs must be compiled after the compilation of FBs since all the variables in an instance DB are dependent on the variables of the related FB.

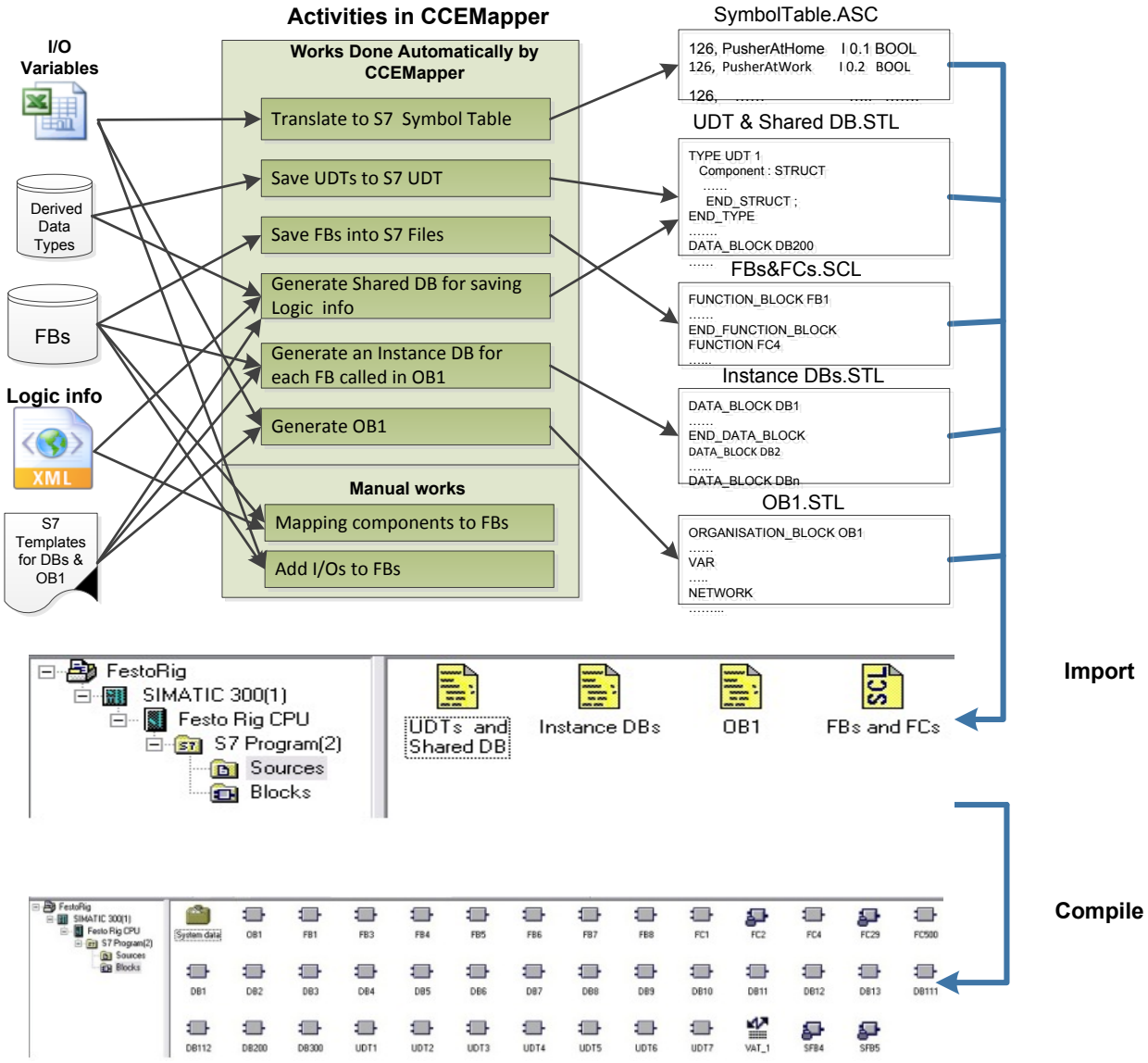


Figure 4-8 Workflow and dataflow of the control code generation process

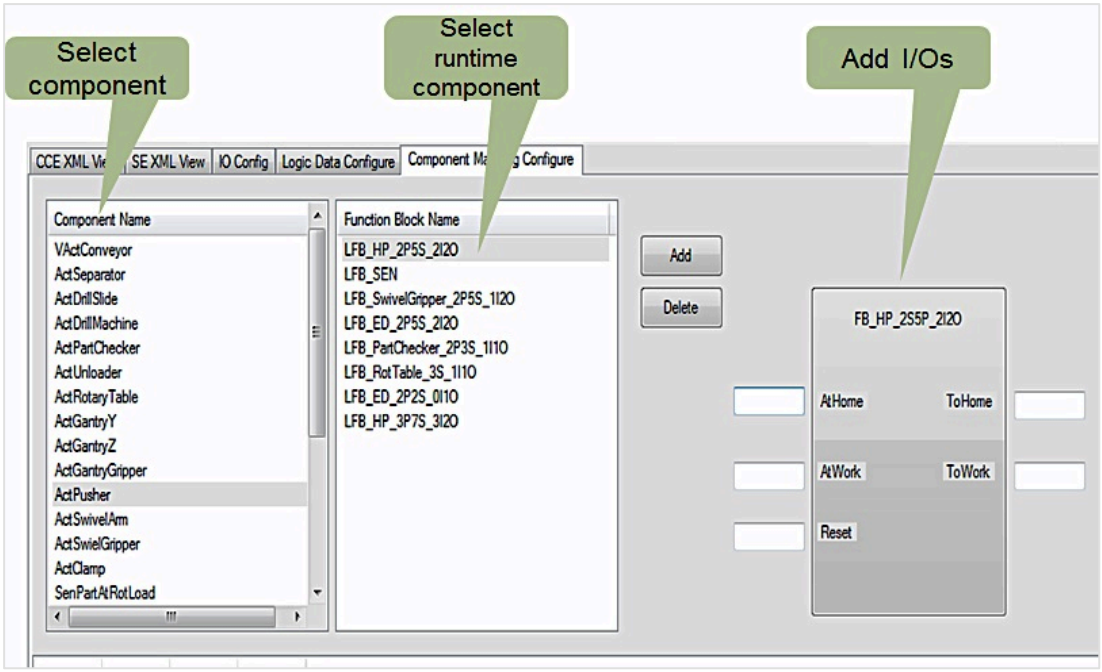


Figure 4-9 GUI for component and I/O mapping of the CCE Mapper

Table 4-5 Source codes export for Step 7

File Name	Contained items	Languages	File Format	Order
Symbol.asc	I/O Variables	N/A	ASC	1
UDT&SharedDB.awl	UDTs, Shared DBs	Statement List	AWL	2
FBs.scl	All the FBs and FCs	Structured Control Language	SCL	3
InstanceDB.awl	All the instance DBs	Statement List	AWL	4
OB1.awl	Organisation Block	Statement List	AWL	5

4.2.2.3 PLC Code Installation

The first step of the PLC installation was to create a new project in STEP 7 and perform the necessary hardware configuration. The generated source code files were imported into the STEP 7 and compiled in the order shown in Table 4-5. By compiling in this order, the UDTs (User Data Types) were generated first, followed by the shared DB that contains instances of the UDTs. Since the Logic Engine uses the data of the Shared DB, the FBs and FCs were generated after the generation of the shared DB. The Organisation Block (OB1) must be generated lastly as it contains the instances of the Function Bocks and the data of the shared DB. The project was compiled and downloaded to the PLC.

4.2.2.4 HMI Screens Generation

As described in section 3.9, template based approach was favoured for the HMI screen generation. A WinCC project was created with predefined templates of screen objects (such as home screen, manual mode screen, logic monitoring screen), screen generation and mapping functions. The template

screens are shown in Figure 4-10 in a hierarchical manner. These template screens consist of both machine specific and non-specific screen objects. The non-specific objects are standard functions, such as Home Screen. Such objects are static and are developed during template preparation stage. Machine specific objects are dynamically populated from the HMI Data Model by the corresponding mapper function in realtime when a screen is requested. The HMI Data Model for describing the machine specific HMI screens are automatically generated by the CCE Mapper and saved in the shared DB of the PLC program. The HMI communicates with this shared DB during runtime to send/receive the required information. A brief description of the dynamically generated screens is given in the rest of this section.

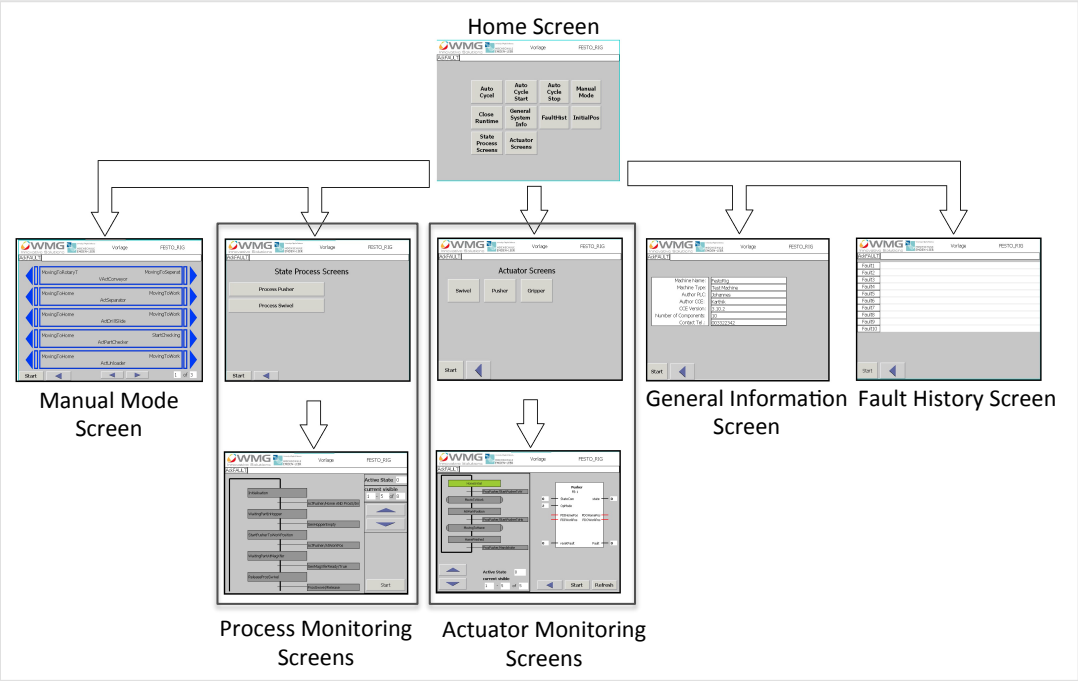


Figure 4-10 Overview of the HMI screens

A. Manual Mode Screen

Manual mode screen consists of pushbuttons for each actuator of a machine to move it between its home and work positions independent of the sequence of operations. In the traditional manual programming approach, these pushbuttons are manually created and mapped to the manual control function of the actuators, which is a time consuming and error prone process. In addition, any modification of a system requires respective changes in the control function as well as manual mode screens, which make the manual method of HMI screen development error prone.

In the automatic code generation approach, a novel methodology is used to automate the process of manual mode screens development. The manual mode screen is composed of generic template of five rows of pushbuttons, which are dynamically populated from the HMI Data Model according to the dynamic position pairs. A pushbutton row is composed of five objects: component name, the names for two reachable positions and two pushbuttons. Each of these objects is associated with an indexed

variable. The mapper function dynamically performs indexing of the indexed variables to associate them with the respective memory location within the HMI Data Model.

The manual mode screen generation method was presented in section 3.9.1. The approach was tested via the Festo test rig system. For illustration, an example of the pushbuttons row for component 'VActConveyor' is shown in Figure 4-11, which depicts the correlation and mapping of the graphical objects of the manual row template, indexed variables, the mapper script and the HMI Data Model. Each graphical object of the template, marked in red, is associated with the respective indexed variables in the template. The mapper script dynamically maps the indexed variables with the corresponding memory locations in the HMI Data Model. As a result of the mapping, the string variables of the memory locations are copied and displayed on the graphical objects.

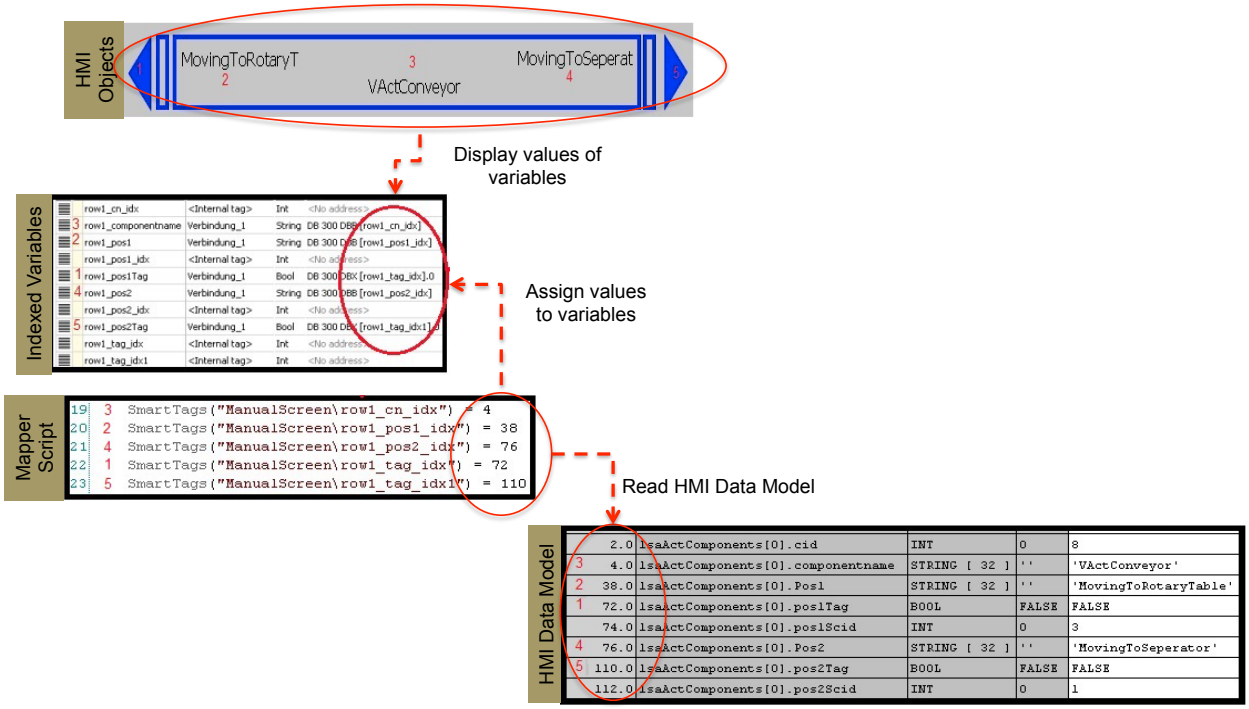


Figure 4-11 Manual pushbutton row generation

To explain the runtime control of the Festo test rig in manual mode, the manual mode control process of component Pusher is depicted in Figure 4-12. When the button "Workposition" on the screen for moving Pusher to its working position is pressed, the command ID is sent to the related data location in the HMI Data Model (Shared DB). The Logic Engine detects the data change and updates the State Command within the Control Model of the component Pusher, which connects with the RC Pusher. The RC Pusher decodes the State Command and sends output command to the I/O variable "Pusher_ToWork" in order to drive the Pusher to move to its working position.

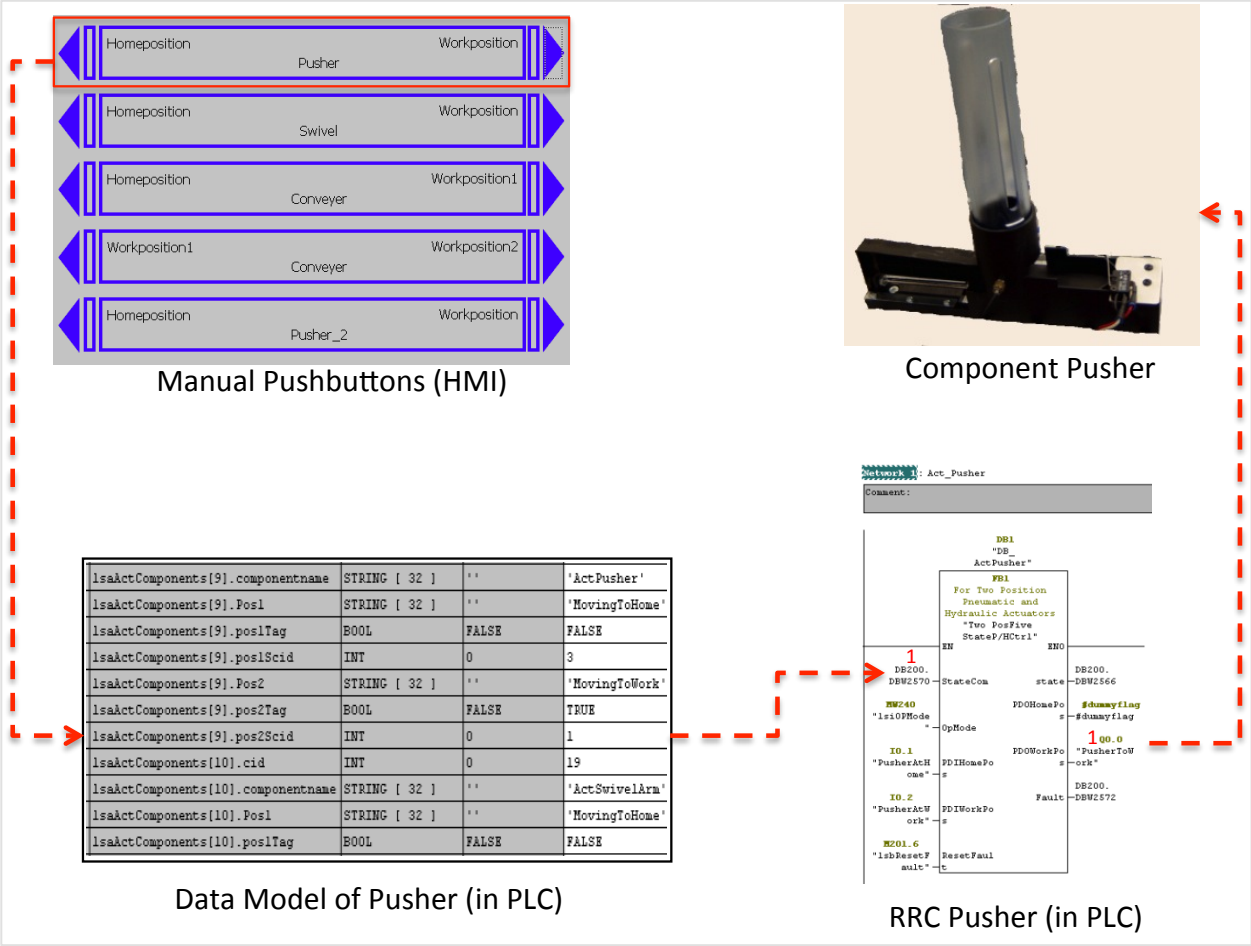


Figure 4-12 Dataflow for manual control of component Pusher

4.2.2.4.1 Process Logic Monitoring Screen

The Process Logic monitoring screen is used to monitor the sequence of operations of the test rig at runtime. This eliminates the need to connect a programming terminal to the PLC required by the traditional programming approach. A screen capture of the Process Logic monitoring screen is shown in Figure 4-13. The generic template of the screen is shown in Figure 4-13 (a) and the runtime view of the generated screen is shown in Figure 4-13 (b). When a Process Logic screen is requested, the mapper function dynamically populates the screen template with the relevant Data Model. The mapper function also continually scans the HMI Data Model and refreshes the active state ID. With the help of the active state ID, the background colour of that state changes from grey to green.

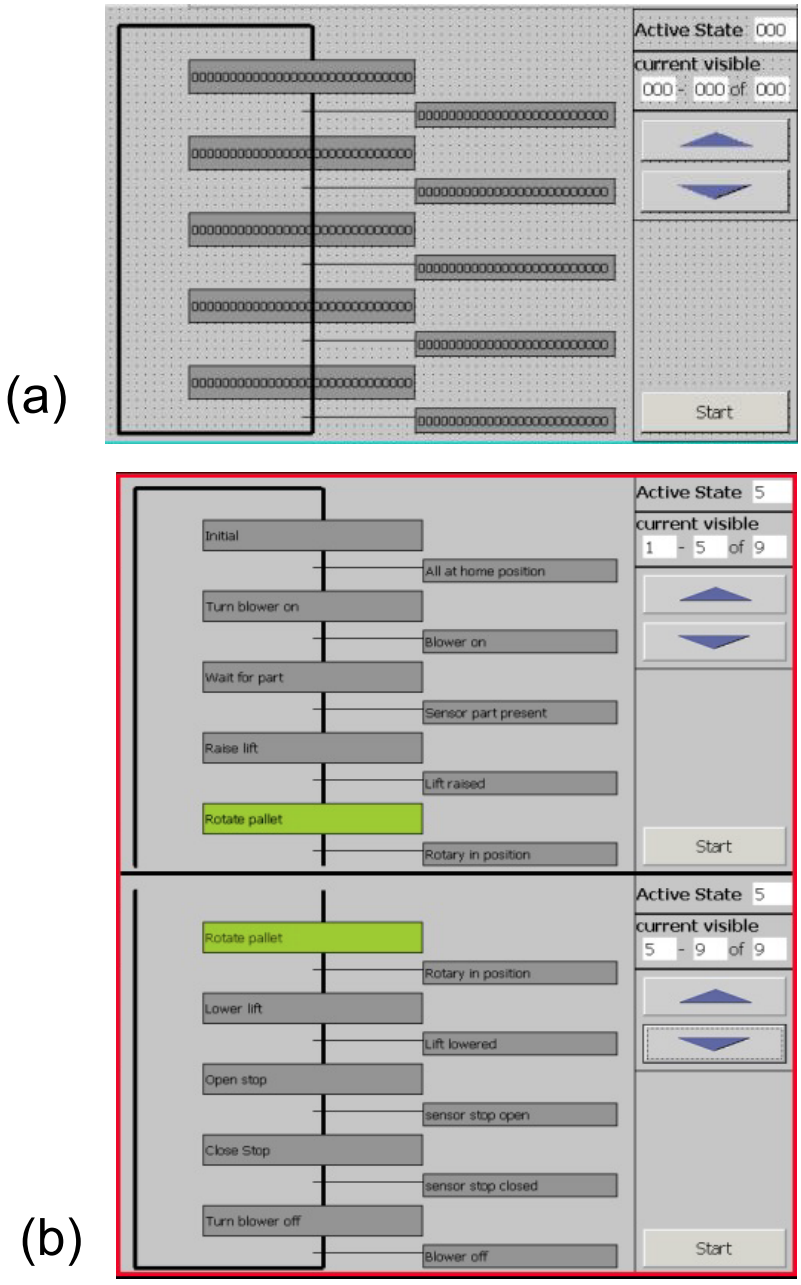
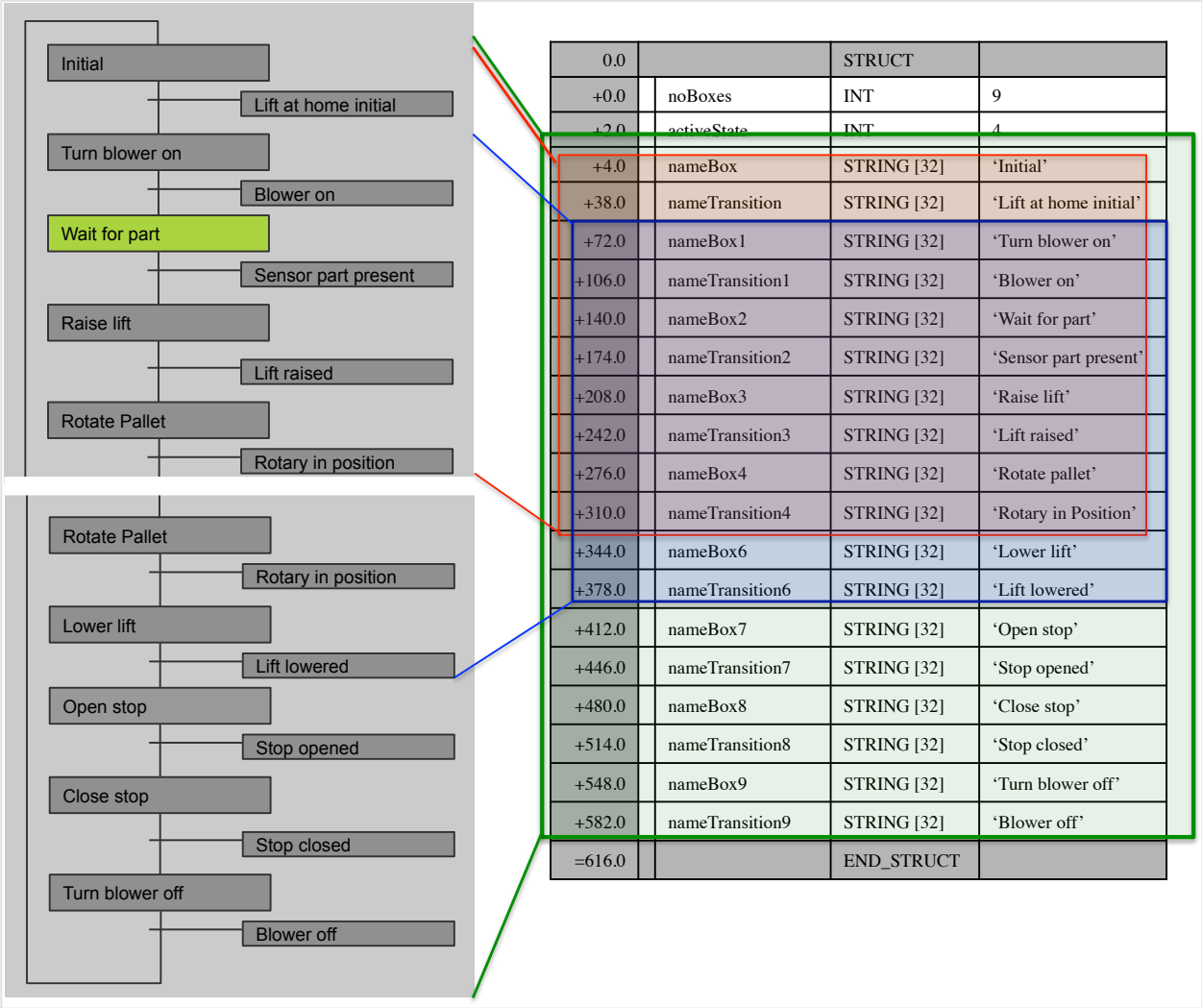


Figure 4-13 Generated Process Logic monitoring screen

Figure 4-14 shows the navigation of the Process Logic monitoring screen by switching the memory area within the HMI Data Model by changing the offset of indexed variables. The green markers are pointing the entire memory area of the Process Logic. The red markers are pointing the default offset and the blue markers are pointing the next offset to navigate the Process Logic screen. As the position of the active state also changes with screen navigation, the value of the active state is re-calculated each time with the navigation by using case-structure in the mapper function.



0.0		STRUCT	
+0.0	noBoxes	INT	9
+2.0	activeState	INT	4
+4.0	nameBox	STRING [32]	'Initial'
+38.0	nameTransition	STRING [32]	'Lift at home initial'
+72.0	nameBox1	STRING [32]	'Turn blower on'
+106.0	nameTransition1	STRING [32]	'Blower on'
+140.0	nameBox2	STRING [32]	'Wait for part'
+174.0	nameTransition2	STRING [32]	'Sensor part present'
+208.0	nameBox3	STRING [32]	'Raise lift'
+242.0	nameTransition3	STRING [32]	'Lift raised'
+276.0	nameBox4	STRING [32]	'Rotate pallet'
+310.0	nameTransition4	STRING [32]	'Rotary in Position'
+344.0	nameBox6	STRING [32]	'Lower lift'
+378.0	nameTransition6	STRING [32]	'Lift lowered'
+412.0	nameBox7	STRING [32]	'Open stop'
+446.0	nameTransition7	STRING [32]	'Stop opened'
+480.0	nameBox8	STRING [32]	'Close stop'
+514.0	nameTransition8	STRING [32]	'Stop closed'
+548.0	nameBox9	STRING [32]	'Turn blower off'
+582.0	nameTransition9	STRING [32]	'Blower off'
=616.0		END_STRUCT	

Figure 4-14 Navigation of Process Logic monitoring screen via indexing offset

4.2.2.4.2 Actuator Monitoring Screen

Similar to the Process Logic monitoring screens, the actuator monitoring screen is used to monitor the STD and the RC of actuators of the test rig at runtime. This eliminates the need to connect programming terminal to the PLC required by the traditional programming approach for runtime monitoring of FBs of actuators. The actuator monitor screen generated for the component Pusher is shown in Figure 4-15. The animated graphical view of the STD of the actuator Pusher is displayed on the left side of the screen. The active state of the STD is highlighted with the help of background colour change from grey to green. On the right-hand side of the screen, the graphical representation of the RC is displayed. The current values of both binary and integer inputs, as discussed in section 3.7.1.4, are displayed. The binary inputs and outputs of the RC are animated by changing their color from red (false) to green (true), while the integer inputs displays their current values in a text field.

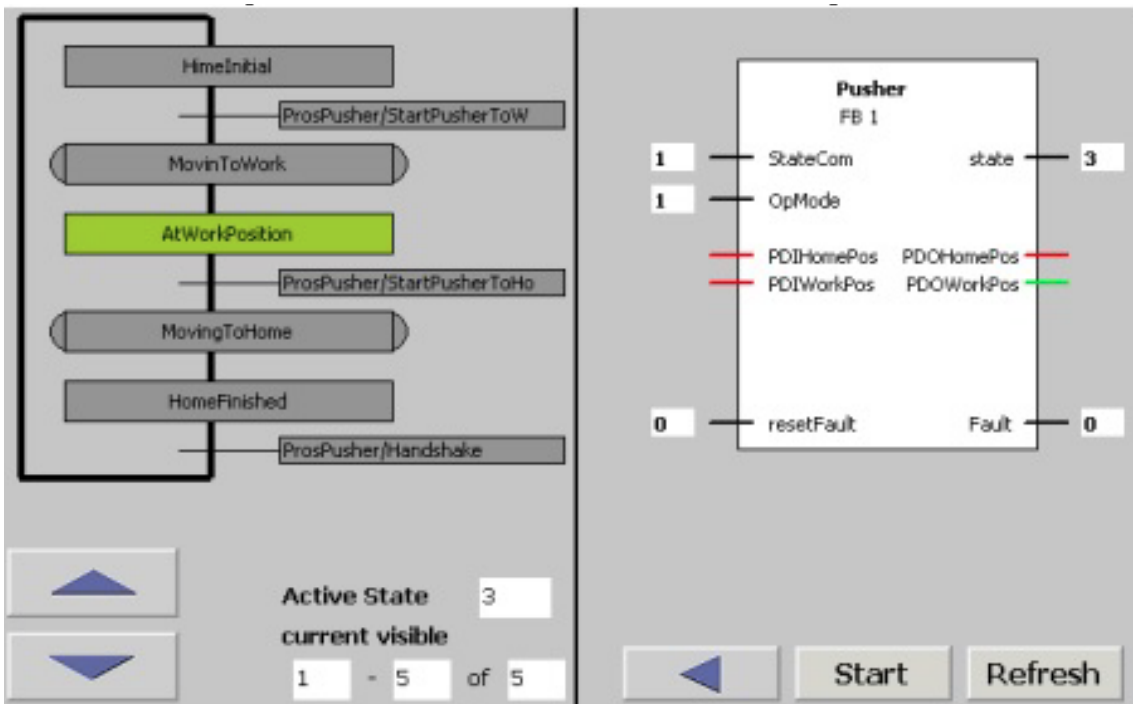


Figure 4-15 Generated actuator monitoring screen for component Pusher

4.2.2.4.3 Fault Message Screen

The generated fault message screen is shown in Figure 4-16. The active fault is displayed on a banner on the home screen as well as on the fault history screen. When a fault occurs then the background colour of the banner changes from grey to red and the fault message is displayed on the banner. The ACK Fault buttons is provided on the home screen to reset the fault. The fault history screens display the previous ten faults. This fault history screen can be accessed via a pushbutton provided on the home screen.

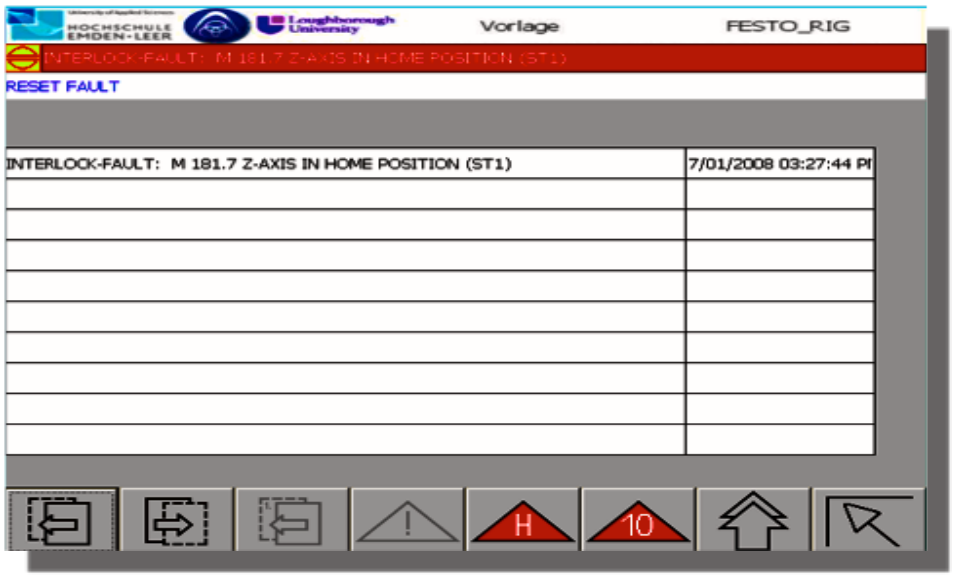


Figure 4-16 Fault history screen

4.3 Evaluation

This section evaluates the proposed automatic code generation approach against the limitations of the current control engineering practice and the research gap outlined in chapter 2. The evaluation method was based on the approaches presented in the relevant literature by Lucas and Tilbury [129] and Hajarnavis and Young [130] and the information drawn from discussions with engineers at Ford Motor Company and TKSE. In addition to the case studies presented, a set of use-cases was designed in order to evaluate the proposed approach from various perspectives and to assess its strengths and weaknesses over the entire automation system lifecycle. Each scenario was carried out using the proposed automatic code generation approach as well as a manual programming approach.

To minimise the effect of the programmers' experience and other human factors, the same person, having good knowledge of both approaches, created the control software using both the manual programming and automatic generation approaches.

The time estimates in study are based on the results recorded by the author together with extensive discussions of the results with control programming experts at Ford and TKSE. However, it is appreciated that an independent evaluation of the time estimates would be needed in the future to confirm these results. This work is likely to be the subject of a future research and development project such as the new TSB funded 3Deployment project but was beyond the scope of the author's research.

The evaluation spans control software development, reconfigurability, portability, diagnostics and maintenance, and runtime performance.

4.3.1 Control Software Development

To comparison of the control software development, the Festo test rig was programmed using both manual programming approach and automatic generation approach. For manual programming, Ford's Global Sigma PLC programming standard was used, which is currently considered as one of the best programming standard within Ford's Powertrain Operations (PTO) Manufacturing Engineering. The same control functions were manually programmed and automatically generated wherever practical. The control hardware and software used in this study included an S300 PLC, MP 277 Touch HMI, STEP 7 and WinCC.

Although a pre-requisite for automatic control code generation, the resources (i.e. time and skills) required to develop the Virtual Prototype using the CCE tools was not included in this comparison. This decision was based on the hypothesis that in practice, the Virtual System Modelling and Virtual Process Planning tasks were carried out prior to, and regardless of, whether automatic or manual is conducted.

4.3.1.1 Results and Observations

A. Development Time

For the comparison of time required to produce both manual and auto-generated control code, the software development process was decomposed into a number of tasks. The time for each task was then recorded independently. The approximate times (in hours) taken for both programming approaches are listed in Table 4-6 and Table 4-7 from same control functionality. By comparing the total time taken via each approach, it can be seen that the manual programming approach account for approximately twice the time taken by the automatic generation approach which represents a total saving of 4 days of work for the test system.

Table 4-6 Manual programming tasks and their respective time

Task		Time (hr)
Pre-Engineering	Template development	3.0
	FBs development and testing	16
System Engineering	Instantiating FBs and creating DBs	1.0
	I/O mapping	1.0
	Writing SFCs	16
	Writing diagnostic code	4.0
	Developing HMI screens	8.0
	Installation and Commissioning	16
Total Time		**65**

Table 4-7 Automatic code generation tasks and time

Task		Time (hr)
Pre-Engineering	Template development	4.0
	RCs development and testing	16
System Engineering	Defining control behaviour in CCE	8.0
	Virtual commissioning	2.0
	RCs and I/O Mapping	1.5
	Installation and Commissioning	2.0
Total Time		**33.5**

B. Impact of Virtual Commissioning

As seen from Table 4-6 and Table 4-7, the installation and commissioning time taken by automatic generation approach was much lesser than the installation and commissioning time taken by manual programming. One of the reasons is that the control logic validation was carried out during the virtual commissioning phase, using the CCE tools. As the virtual commissioning was performed offline, a significant potential improvement in the project critical path was envisaged. In this study, the test rig was a laboratory-based machine and was available for programming, which is usually not the case when a new machine is developed. It was anticipated that the project lead-time difference for the two approaches would have further increased if the test rig was not readily available for programming.

In addition, it can be noted from the time comparison that even the combined time required to achieve

virtual commissioning and subsequent physical commissioning using automatic code generation, is still lower than the time required to conduct manual control code programming. This is due to the fact that the debugging of the control logic of manual programming approach was cumbersome and time consuming compared to the control behaviour debugging within CCE tools. Moreover, during commissioning of manually written program a number of mechanical clashes were witnessed (such as rotation of the indexing table during checking process) due to missing interlocks, which in turn required some amount of code re-writing.

C. Required Knowledge and Programming Skills

The required knowledge and programing skills for both approaches are shown in Table 4-8 and Table 4-9. The Manual programming approach requires extensive experience of PLC and HMI programming for both the pre-engineering and system engineering tasks. In addition, the programmers must possess a good knowledge of the programming standard used (in this case Global Sigma). In comparison, for the automatic generation approach only the template and RCs development requires PLC programming skills. As these are pre-engineering tasks and are performed only once, it can be concluded that system engineering using the proposed approach only requires basic PLC programming skills.

As the HMI screen is automatically generated, no HMI programming experience is required. In addition, the software is structured automatically, so the engineers involved in the system engineering do not require extensive knowledge of the PLC structured programming standard.

Table 4-8 Required knowledge and programming skills for the manual programming approach

Task		Knowledge/ Skills	Skill / knowledge Level
Pre-Engineering	Template development	▪ STEP 7 ▪ WinCC ▪ Software architecture	Advanced
	FBs development and testing	▪ STEP 7 ▪ Software architecture	Advanced
System Engineering	Instantiating FBs and creating DBs	▪ STEP 7 ▪ Software architecture	Basic
	I/O mapping	▪ STEP 7 ▪ Software architecture	Basic
	Writing SFCs	▪ STEP 7 ▪ Software architecture ▪ Process planning	Advanced
	Writing diagnostic code	▪ STEP 7 ▪ Software architecture	Advanced
	Developing HMI screens	▪ STEP 7 ▪ WinCC ▪ Software architecture	Advanced
	Installation and Commissioning	▪ STEP 7 ▪ Process Planning	Advanced

Table 4-9 Required knowledge and programming skills for the automatic code generation approach

Task		Knowledge	Skill / Knowledge Level
Pre-Engineering	Template development	▪ STEP 7 ▪ Software architecture	Expert
	RCs development and testing	▪ STEP 7 ▪ Software architecture	Advanced
System Engineering	Defining control behaviour in CCE	▪ Process planning	Basic
	Virtual commissioning	▪ Process planning	Basic
	RCs and I/O Mapping	▪ No specialised Skills	Basic
	Installation and Commissioning	▪ STEP 7 ▪ Process planning ▪ Software architecture	Basic

D. Integration of Control Code

The integration of the control functions, diagnostic code and the HMI screen generation is completely automated using the automatic code generation approach. Only the control behaviour needs to be defined within the CCE, while the diagnostic code and HMI screens are generated automatically. On the other hand, the manual programming approach requires the control functions, fault diagnostic code and the HMI screens to be handled individually which makes the programming process time consuming and prone to mistakes. The Ford's Global Sigma programming standard used for manual programming approach integrates the control functions with the diagnostic code and HMI code. However, this integration of code is only from the software structure perspective, which still requires manual modification of code.

4.3.2 Reconfigurability

As discussed earlier, reconfigurability of a system is a key requirement. Reconfiguration is required to modify an existing system in order to accommodate with changing production requirements. The reconfiguration capability provided by a programming approach can be measured by analysing the ease with which a system can be altered to meet new requirements. This can be quantified by measuring the time required to conduct a change in a system.

In order to evaluate and compare reconfigurability of the automatic code generation approach with the manual programming approach, the Buffer Station (which consists of Conveyor, Separator and three proximity sensors) was removed from the test rig. After removing Buffer Station, work piece was transported directly to Processing Station. This reconfiguration experiment required physical reconfiguration as well as control software reconfiguration. Since this study was only aimed at evaluating the reconfigurability of the control code, the time required for physical reconfiguration was not taken in account but was the same in both cases. The physically reconfigured test rig is shown in Figure 4-17.

Figure 4-17 Reconfigured Festo test rig

For the manual programming approach, the reconfiguration was performed by modifying the corresponding parts of the program. The following tasks were performed to reconfigure the control code:

- Delete the related I/Os from the variables table
- Delete the FBs and SFC for Conveyor and Separator
- Delete the relevant diagnostic code
- Modify the HMI program to remove the relevant pushbuttons rows from the manual mode screen and do the related changes in the PLC program
- Modifying the SFCs to change the sequence of operations of Swivel Arm, Vacuum Gripper and Rotary Table.
- Download the modified programs to PLC and HMI
- Re-commission the test rig

The step-by-step tasks performed to reconfigure the control code using automatic code generation approach are given below:

- Reconfigure the system in CCE tools by deleting the related components and modifying the transition conditions of STDs of Swivel Arm, Vacuum Gripper and Rotary Table for the new process flow.
- Perform virtual commissioning to validate the control behaviour
- Load the reconfigured project file and generate the control code
- Import the control code to STEP 7 and download to the PLC

- Re-commission the test rig

4.3.2.1 Assessment and Observations

A. Reconfiguration Time

The time taken for each approach is shown in Table 4-10. By comparing the total time, it can be seen that the automatic generation approach requires less reconfiguration time compared to the manual programming approach for this potential test case. A number of factors contribute to the time saving achieved during automatic code generation. Some of these factors include:

- automation of the program generation
- dynamic reconfiguration approach used in the HMI software design, and
- ease of editing STDs as compared to the editing of low-level PLC control code.

Table 4-10 Time to reconfigure the test rig

Manual Programming Approach		Automatic Generation Approach	
Tasks	Time (hr)	Tasks	Time (hr)
PLC code modifications	1.2	Virtual model reconfiguration	1.0
HMI modifications	0.5	Virtual Commissioning	0.5
Installation and re-commissioning	1.0	Installation and re-commissioning	0.2
Total Time	2.7		1.7

B. Impact of Virtual Commissioning

The most important factor during the reconfiguration process is the production downtime. During the reconfiguration process, the production downtime is the time required to re-commission the machine. As the virtual commissioning was performed offline, the time spent during virtual commissioning did not contribute to the production downtime. As a consequence, the production downtime during the re-configuration process for automatic code generation approach was five times smaller than the manual programming approach.

C. Required Programming Skills and Knowledge

To achieve reconfiguration, the manual programming approach requires alteration of the PLC control code and the HMI screens. These alterations required knowledge of process planning, extensive PLC programming experience (both STEP 7 and WinCC) and knowledge of the PLC program structure. While the reconfiguration process for the automatic code generation approach required knowledge of process planning and basic knowledge of the PLC programming. As the program generation and the HMI screen reconfiguration are automated, no knowledge of the PLC software structure and HMI programming is required. The knowledge and programming skills required for reconfiguration using the manual programming approach and the automatic code generation approach are given in Table 4-11 and Table 4-12.

Table 4-11 Knowledge and programming skills required for reconfiguration using the manual programming approach

Task	Knowledge/ Skills	Skill / knowledge Level
PLC code modifications	- STEP 7 - Process Planning - Software architecture	Advanced
HMI modifications	- STEP 7 - WinCC - Software architecture	Advanced
Installation and re-commissioning	- STEP 7 - Software architecture - Process planning	Advanced

Table 4-12 Knowledge and programming skills required for reconfiguration using the automatic generation approach

Task	Knowledge/ Skills	Skill / knowledge Level
Control logic editing in CCE	- Process Planning	Basic
Virtual Commissioning	- Process Planning	Basic
Installation and re-commissioning	- STEP 7 - Software architecture - Process planning	Basic

4.3.3 Portability of Control Logic across PLC Platforms

As discussed in Chapter 2, the portability of PLC code is a well-known issue. To enable portability, PLCopen introduced an XML based portable neutral format for import/export of source code, but the standard has not been widely adopted by control vendors. The automatic generation approach addresses the portability of the control logic to a great extent due to the vendor neutral control behaviour definition within CCE, mapping of the I/O variables and mapping of RCs.

To assess the extent of the portability of the control logic of both approaches, the Siemens S300 PLC and STEP 7 were changed to Schneider Electric's Modicon TSX P57 PLC and Unity Pro. The activities of both programming approaches and the time taken to re-program the Festo test rig using Modicon TSX P57 PLC are shown in Table 4-13 and Table 4-14.

As shown in Table 4-13, the portability of code in the manual programming approach was very limited and was based on ad-hoc procedures. The true portability of code was attained to some extent by porting the code of the library Function Blocks from STEP 7 to Unity Pro via cut-paste method. However, the code needed slight modifications due to differences in the STEP 7 and Unity Pro syntaxes. The rest of the code written in STEP 7 in the pre-engineering phase and system engineering phase was not portable. As a consequence, paper prints of the STEP 7 project were taken and the code was re-typed in the Unity Pro. A total time of 29.8 hours was required to achieve manual duplication of

the code.

The automatic code generation approach provides a limited amount of portability for the code written in the pre-engineering phase. However 100% of the control logic (defined in CCE) and mapping information were successfully ported, without manual code re-writing. As a result, the system engineering phase only required 15 minutes (0.25 hours) for completion. This portability was only possible because the control logic defined in CCE and the consequent I/O mapping and RCs mapping information are stored in a vendor neutral format.

Table 4-13 Control logic portability of the manual programming approach

Activity		Time (hr)	Comments
Pre-Engineering	Template development	2 .0	Not portable
	FBs library development and testing	7.0	Portable to some extent
System Engineering	Instantiating FBs	0.3	Not portable
	I/O mapping	0.5	Not portable
	SFCs	10	Not portable
	Diagnostic code	2	Not portable
	HMI development	6	Not portable
	Installation and commissioning	4	A number of bugs were found
Total Time		29.8	

Table 4-14 Control logic portability of the automatic code generation approach

Activity		Time (hr)	Comments
Pre-Engineering	Template development	1.5	Not portable
	RCs development and testing	8.0	Portable to some extent
System Engineering	Defining control behaviour in CCE	0.0	Portable
	Virtual commissioning	0.0	Portable
	RCs and I/O mapping	0.0	Portable
	Installation and commissioning	0.25	No debugging was required
Total Time		9.75	

4.3.4 Fault Diagnosis and Maintenance

This section evaluates the effort required for fault detection and maintenance of the auto-generated code during the commissioning and machine operation phase. Diagnosis is needed when an unexpected event happens in the controlled process. The cause of unexpected event can be the result of a hardware fault or a bug in the control code. The corrective action varies in each scenario and depends on the root cause of the error. For diagnosis and maintenance of the control system, the two most im-

portant aspects are the visibility of control logic and the ease of debugging.

A. Visibility of Control Logic

In this thesis the term visibility refers to the ease of accessibility and interpretability of the control code, which facilitates the identification of the causes and effects of an event or fault that requires diagnosis. Diagnosis is carried out by monitoring the status of sensors, actuators and sequential logic in realtime, which makes the runtime visibility of control code a significant requirement. This is one of the reasons why the LD and SFC are commonly used to program PLCs rather than languages such as IL and ST.

Unlike the traditional manual programming approach that needs a programming terminal to monitor the control logic in realtime, the automatic code generation approach allows the Process Logic and RCs to be monitored by connecting PLC programming terminal to the PLC as well as from the HMI screens.

In the auto-generated code, actuators are controlled by RCs and Process Logic. Therefore, in order to find actuators malfunctions, the respective Process Logic and RC need to be examined. Within the PLC program, Process Logic appears as a runtime control model, which is difficult to access and directly interpret. However, the HMI program converts the runtime model into STDs, which depict the sequence of operations via a simple graphical user interface. The STDs of Process the Logic are displayed on the HMI screen (see Figure 4-13) with active states highlighted in green which makes tracing of the current state in a sequence of operation very intuitive compared to the manual program code interpretation.

RCs appear as FBs in the generated program. RCs have well-defined interfaces that can readily depict the status of an actuator, its related sensors and the intended control operation. RCs can be monitored from the HMI as well as by using the online monitoring function of the PLC programming tool. An example RC for the Pusher component is shown in Figure 4-18. Checking whether a fault has occurred in the hardware (such as faulty I/O connection), or spotting a bug in the sequential logic or in the RC is a simple task when looking at the interface variables of the RC.

On the other hand, in order to view the code for the control logic the user would need to connect the PLC to the programming terminal to locate and understand the intended control behaviour. In order to do so, the user must have knowledge of the complete programming structure. This often implies a great degree of complexity and requires higher level of expertise in the PLC languages to understand the system behaviour.

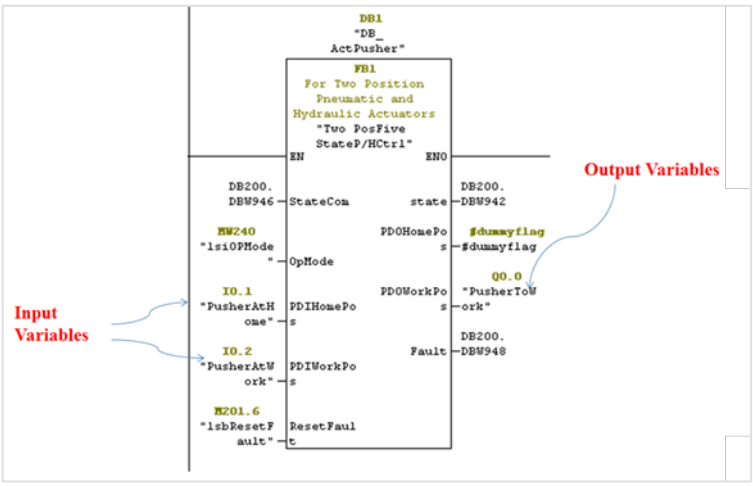


Figure 4-18 RC for actuator Pusher

B. Debugging of Control Code

The auto-generated code is based on the RCs and the defined sequential logic in the CCE. The control code of RCs is pre-validated, the chances of bugs occurrences in the RCs are very rare. During physical commissioning of the Festo test rig, no faults were observed in the RCs. However, debugging and diagnostic of the control code of RC, is the same as that of traditional control software. The RC can be directly diagnosed in the PLC programming tool. Once the bug is found, it can be directly fixed by modifying the code of the runtime component.

On the other hand, the sequential logic defined within the CCE is debugged during virtual commissioning phase. However, due to some limitations of the CCE tools' simulation capabilities (such as lack of multiple part simulation) the virtual commissioning phase could not validate the sequential control logic completely. During commissioning of the test rig, a number of bugs were noticed. The debugging of the sequential control logic could only be achieved within the CCE tools. This is because the sequential control logic defined within the CCE tools is converted into Runtime Data Models and stored in a database. The runtime control models are described in a machine-understandable format, but are not readable by control engineers. The reduced readability of the runtime data models makes the code very hard to debug within the PLC programming tool. However, this should be considered as the limitation of the simulation capability of the CCE tool rather than the limitation of the auto-generated code.

4.3.5 Runtime Performance

The runtime performance of the control code generated by the automatic generation approach is analysed by measuring the program memory size and scan time. The control software developed by the manual programming approach (Ford's Global Sigma format) is widely acceptable to industry and thus used in this thesis as a benchmark for evaluation of the runtime performance of the automatically generated control code.

A. Program Memory

The memory occupied by the control software can be classified into load memory, system memory and main memory. The load memory is used for storing all the control software information when the project is downloaded to the PLC. The system memory is random access memory (RAM) that contains elements such as markers, timers, counters, block stack and interrupt stacks. The main memory, also a RAM, is optimised for high-speed access. At start-up the PLC copies the parts of the load memory necessary for program execution to the main memory. The main memory is further divided into two parts. One part is used for storing the runtime-relevant code, process input image (PII), process output image (POI) and the diagnostics buffer for the code. The other part is used for the runtime-relevant data and also contains the data from the local data stack [131].

The memory occupied by the control code of both approaches is given in Table 4-15. It can be seen that the auto-generated code requires less program memory, less load memory and less work memory, than the manually written program, the main reason being the difference in the software architecture of the two programming approaches. The sequence of operations in the automatic code generation approach is based on Logic Engine and Control System Data Model, which require much less memory compared to the SFCs used in the manual programming approach.

Table 4-15 Comparison of PLC program memory

Programming Approach	Load Memory	Main Memory	System Memory
Automatic Generation Approach	24378 bytes	21610 bytes	12170 bytes
Manual Programming Approach	48462 bytes	27058 bytes	12170 bytes

B. Scan Time

Unlike event-based systems, PLCs execute programs in a cyclic manner as shown in Figure 4-19. One complete cycle is known as a scan. The scan begins by reading the inputs and updating the process input image (PII). The control code is then executed based on the PII. The outputs are written to the process output image (POI). Once the application program execution is completed, the PLC performs diagnostic and communication tasks. The time required to execute one cycle is known as the scan time. Because a manufacturing control system is a hard-realtime system, large scan times directly affect the accuracy of the PII and can result in unintended operation.

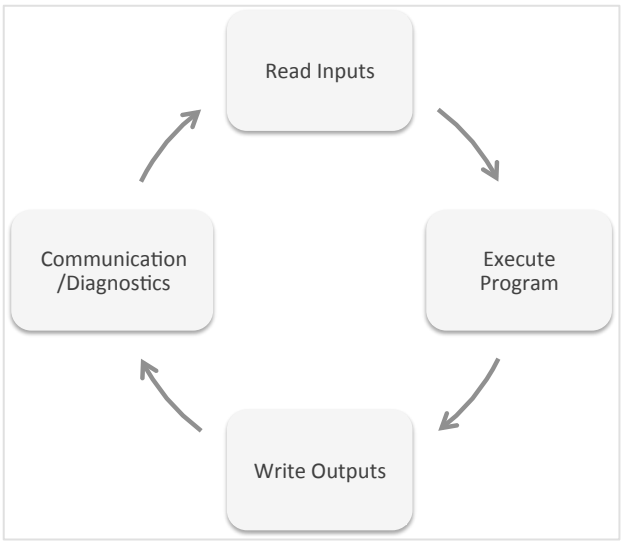


Figure 4-19 PLC scan cycle

To determine the scan time, a function block that records and calculates the scan time was used for both programming approaches. During the automatic test rig operation, the maximum scan time and the minimum scan time were recorded and the average scan time was calculated (see Table 4-16). It can be seen that the scan time for the auto-generated code is lower than the scan time required to process the manually written code.

Two factors possibly contribute to the better scan time performance of the auto-generated code. The first reason lays in the smaller memory requirement of the auto-generated code. The second reason is the way control commands are realised in the auto-generated code, which is simple and faster than that in a manually written program. This is because the machine control logic is modelled as runtime control data models and saved in a Data Block. During each scan cycle, the Logic Engine scans these runtime control models and generates commands for the RCs. For the manually written program the command for controlling resource FBs is generated by the respective SFC, which might still need to communicate with other SFCs to generate the command. This potentially leads to increased scan time.

Table 4-16 Comparison of scan time

Programming Approach	Maximum Scan Time	Minimum Scan Time	Average Scan Time
Auto-generated	8ms	<0.5ms	3ms
Manually written	19ms	<0.5ms	14ms

4.4 Summary

This chapter has presented a proof-of-concept case study to show the applicability, comparative advantages and performances of the proposed automatic code generation approach relative to a state-of-the-art manually coded control software. A representative assembly automation test rig was chosen for the case study. Control code was developed for Siemens and Schneider Electric PLCs using both

automatic code generation and manual programming. A number of experimental studies were conducted to test and compare the approaches against a set of key qualitative and quantitative performance criteria.

Numerous advantages were observed through the evaluation of the prototype automatic code generation. The generated control code could be complied to comply with the program structural template requirements of the automotive industry, which is a fundamental requirement for industry for acceptability. The automatic generation approach allowed the control software development time, commissioning time and the need for specialised programming skills to be significantly reduced, and thus could shrinks the cost and lead-time of a project compared to the manual programming approach. The generation of structured control code and logic monitoring functionality now integrated to the HMI screens resulted in the enhanced visibility of the control code to readily diagnose faults without the need to connect a programming terminal to the PLC. The runtime performance of the auto-generated code was far better than the manual programming approach with 400% improvement in overall scan time and a 30% reduction in memory requirements for the test application. This better runtime performance could not only increase the reliability of a system by preventing excess scan-time related issues but could also allow the use of smaller lower cost PLCs for complex applications.

Due to some limitations in the simulation capabilities of the CCE tool, the auto-generated control code needed some minor debugging of the sequential control logic (originally defined within the CCE tool) during physical commissioning. However, this should be considered a current limitation of the simulation capability of the CCE tool rather than the limitation of the new programming approach.

5 Conclusion

This chapter concludes the research work documented in this thesis. A comprehensive summary of the achievements, contributions and benefits of this research are outlined. At the end recommendations for the future work are given.

5.1 Achievement of Research Objectives

The aim of this research was to investigate the automatic generation of control code by reusing control information defined within manufacturing process simulation tools, referred to as virtual engineering (VE) tools. To realise the aim, a number of research objectives were outlined in Section 1.2.3. This section summarises the achievement of the research objectives in this work.

- ***Objective 1:*** *To review existing PLC programming practice within the automotive manufacturing sector to recognise the control software structural and functional requirements, identify the current challenges and future requirements.*

An extensive review of the development of the automation systems and current PLC programming practices within automotive sector is presented in Chapter 2. The limitations of the existing PLC programming practice in the context of future requirements are highlighted. In addition to this, relevant emerging approaches in controls engineering are reviewed. From the literature review, it was established that the use of IT tools for integrated development of automation systems, virtual commissioning and automatic code generation can potentially address the existing limitation of the control engineering practices in PLC programming. It was further concluded that for industrial acceptability of any new approach for PLC programming, it is important that it should fit within the existing engineering workflow and must address the functional and structural requirements of the industry.

- ***Objective 2:*** *To enhance control information and specify a method for the definition of control logic within a virtual process simulation tool, the CCE, to enable direct deployment of the complete control code.*

The existing method of control behaviour definition within the CCE tools was critically reviewed in Chapter 3. A number of limitations of CCE from control code deployment and control behaviour validation perspective were documented. To address these limitations, a number of changes were implemented in CCE tools. These changes enabled control code generation from the control information defined within virtual models.

- ***Objective 3:*** *To design a software architecture that complies with the current PLC software structures used in automotive industry production machines and to design an approach to au-*

tomatically generate PLC control code according to this software architecture by utilising control information defined in virtual models of manufacturing cells developed in the manufacturing process planning tool, the CCE.

A novel control software architecture is designed and implemented, which is presented in Section 3.7. The software structure is derived from the FAST (Ford and Siemens Transline) specification for PLC based automation systems at Ford Motor Company. An approach for code generation according to the software architecture is presented in Section 3.8. The approach enabled generation of complete executable PLC control code in an automated manner from virtual models of automation systems developed in CCE tools. The generated code provides functions for automatic mode control, manual mode control and fault diagnostics.

- ***Objective 4:*** *To design and develop an approach for integrated and automatic generation of HMI Screens.*

A novel approach for automatic HMI screen generation is designed and implemented. The details of the approach are given in Sections 3.7.2 and 3.9. The method of HMI screens generation automatically integrates the objects of the HMI screens with the runtime data models within the PLC code, thus enabling the dynamic generation and reconfiguration of HMI screens. The approach also enabled monitoring of the control logic using on-machine HMI screens instead of an external programming terminal.

- ***Objective 5:*** *Implementation of a prototype system to validate the research hypothesis and evaluate the approach.*

A case study of a prototype application is presented in Chapter 4. Using the developed approach for PLC code generation and automatic HMI screens generation, STEP 7 control code and WInCC HMI screens for a Festo test rig were automatically generated. The control code was compiled and downloaded without any manual application coding to an S300 PLC installed on the test rig. The test rig was then commissioned in both manual and automatic modes, thus validating the research hypothesis presented in Section 1.2.2.

Based on the case study and some additional experiments, evaluation of the approach was carried out from various perspectives. The evaluation revealed several benefits of the approach over the traditional manual programming approach, such as reduced application development time, reduced reconfiguration time, ease of debugging control logic and improved PLC runtime performance.

5.2 Research Contributions

This thesis makes the following original contributions to the field of the virtual engineering and control engineering of automation systems:

- A detailed understanding of the requirements that must be met and current limitations that

must be resolved in order to design an acceptable approach for VE-based automatic PLC code generation applicable to the automotive manufacturing sector.

- A methodology for control logic definition within component-based VE tools to enable the direct deployment of validated PLC control code.
- An approach for structured PLC control software generation (conformant to current industry standards) based on the control information from component-based virtual models of manufacturing cells and utilising pre-validated runtime components.
- An approach for the automatic generation of HMI screens from the state behaviour of the component-based virtual models.
- An approach for control logic visualisation on HMI

5.3 Research Benefits

The case studies and evaluation (described in Chapter 4) have demonstrated a number of benefits of adopting the proposed approach for PLC code deployment. A summary of these benefits is given below.

5.3.1 Integrated Engineering of Automation Systems

Instead of writing control code independently in PLC programming tools at the system engineering phase, the use of VE tools enabled control engineers to define the control logic in close collaboration with mechanical and process engineers. In contrast to the traditional programming approach, the 3D visualisation of the machine during the control logic definition stage was considered very helpful. In addition, the control logic was defined at a high-level of abstraction by using STDs, which allowed mechanical engineers and process engineers to understand and edit the control logic directly.

5.3.2 Deployment of Virtually Verified Control Logic

The integration of control logic with the 3D CAD model of a machine in the VE tools facilitated commissioning of machine control behaviour in a virtual environment. The case studies showed that the control logic could be optimised and substantially debugged by simulating machine model in virtual environment. The benefit of the early debugging and optimisation of the control logic could be seen in the form of reduced time for physical commissioning during the machine development phase and negligible downtime during the machine reconfiguration phase (see Sections 4.3.1 and 4.3.2) for the use-cases demonstrated on the test rig.

5.3.3 Integrated Control Software Development

The control logic, fault diagnostic and HMI screen generation realised here are all based on a common machine configuration data i.e. the Control System Data Model. Thus, any change in the Control System Data Model is automatically reflected in all three areas of the control software. This approach

enables a seamless and dynamic integration of the control behaviour, fault diagnostic and HMI screens.

5.3.4 Reduced System Development Time

The evaluation presented in Section 4.3.1 showed that for the investigated use-case the proposed automatic code generation approach significantly reduced the system development time as compared to the traditional manual programming approach. The reduction in the system engineering time is mainly due to the automation of the PLC code generation and HMI screen generation.

5.3.5 Reduced Level of Required PLC Programming Skills

In contrast to the traditional manual programming approach, the proposed approach reduced the need for PLC programming skills at the application programming stage to a large extent. It can be seen from Table 4-12 that only a basic understanding of PLC programming is required at application specific system engineering phase.

5.3.6 Enhanced Reconfigurability

The reconfigurability evaluation presented in Section 4.3.2 showed that for the investigated scenario the proposed automatic code generation requires less effort, time, programming expertise and knowledge of the existing control code to reconfigure a system as compared to the traditional manual programming approach. Reconfiguration is carried out offline by editing the STDs within the VE tools. The control information of the reconfigured system is then automatically converted into control code using the CCE Mapper. Once the generated code is downloaded into the PLC, the HMI screens are automatically updated using machine configuration data shared through the Control System Data Model at runtime.

5.3.7 Enhanced Reuse of Control Code

In the proposed approach, low-level PLC programming is only required during the pre-engineering phase to develop the program template and RCs. The program template and RCs are then stored in the CCE Mapper library. This proven pre-developed control code is retrieved from the library and directly used during system engineering phase without the need to understand its underlying low-level details.

5.3.8 Platform Independent Logic Definition

The evaluation related to control code portability presented in Section 4.3.3 showed that the automatic code generation approach enhances the portability of control logic. This is because the control logic defined during system engineering phase (i.e. control behaviour definition within CCE, mapping of the I/O variables and mapping of RCs) is in a vendor neutral format, which via the approach presented in this thesis is automatically converted into a vendor specific format by selecting the type of PLC.

5.3.9 Monitoring of Control Logic from HMI

Traditionally, PLC control code can be monitored only by connecting a programming terminal to the PLC. However, the proposed software architecture enables monitoring of the sequence of operations (i.e. Process Logic) and actuators (i.e. RCs and STD of actuator components) in real time directly from the machine HMI screens. This novel functionality is provided by the proposed automatic code generation approach.

5.4 Future Research Directions

The research work presented in the thesis provided the fundamental groundwork to achieve the aim of this research outlined in Section 1.2.3. However, it is envisioned that further developments and appraisal of the approach could be made by researching the following additional research objectives.

5.4.1 HIL Commissioning

In the proposed framework for automatic code generation, the virtual commissioning can only verify the control logic defined within the VE tools. Thus, the RCs developed during the pre-engineering phase, the I/O mapping of the physical addresses and RC mapping with components cannot be validated until the physical commissioning phase. As a result any error in the RCs code, I/O mapping or RC mapping will remain undetected in the generated code until physical commissioning occurs. To validate the control logic 100% in a virtual environment, there is a need for HIL virtual commissioning of the generated control code, i.e., to enable the control code to be executed on the physical PLC whilst in control of a virtual machine model. The HIL virtual commissioning can be performed by creating connections between the generated control code and the virtual model of the machine during runtime. A suitable communication link between the PLC and VE tools can be realised by using the OPC protocol over TCP/IP.

5.4.1 Web-based Remote Monitoring and Support

The data model in the VE tools consists of 3D CAD geometry, kinematics and control information. This common data model can be used for the development of web-based applications with realtime 3D-based visualisation and control logic monitoring (described in Section 3.9.1) capabilities. This will enable to monitor and support machines remotely regardless of machine's geographical location. Such a web-based application can be connected with the PLC using an OPC link for realtime data communication. The performance of data transfer via TCP/IP Ethernet using OPC needs to be analysed to investigate the runtime performance of such applications.

5.4.2 Automatic Generation of OPC UA Configuration

For HIL virtual commissioning and web-based remote monitoring applications, the connections between the control systems and virtual prototypes need to be created and configured based on OPC. Manual creation and configuration of such a connection is a complicated process and can decrease the

feasibility of additional functions due to the associated time and cost. This task can however be automated by automatically generating the OPC Unified Architecture (OPC UA) client objects based on the virtual common data model. This will significantly enhance further reuse of the virtual data models for HIL virtual commissioning and remote monitoring applications during the machine operation phase.

5.4.3 Deployment of SFC Based Sequence of Operations

In the software architecture proposed and implemented via this thesis, the sequence of operations is controlled by a Logic Engine. The logic engine works as a system orchestrator, which scans the control information related to the sequence of operations, interlocks, actuators and HMI stored within the Control System Data Model and send control commands to the actuators and HMI. During runtime it is impossible for maintenance engineers to directly visualise the intended machine sequence from the control code. This limitation of the control code is complemented by the control logic monitoring screens of the HMI. However, it could be considered preferable to represent the sequence of operations in SFC form. Therefore, in order to enhance the visibility and understandability of the automatic code generation approach in industry, generation of SFC-based code instead of using a Logic Engine can be potentially investigated.

5.4.4 Development of SFC Based Runtime CCE Tool

During the evaluation of the presented approach for logic generation, it was envisioned to develop a runtime version of the CCE tools, which allows the editing and runtime monitoring of the SFC compliant STDs. This can potentially reduce the time required to modify the control logic and can eliminate the need for deployment of SFC based sequence of operations (described in section 5.4.4).

5.4.5 Optimisation of the Control System Data Model

Currently, the Control System Data Model is composed of the System Data Model, HMI Data Model and Fault Management Data Model. Each of these data models contains same basic information about the system configuration. This repeated information significantly increases the size of the Control System Data Model. As a consequence, the memory and the scan time of the control code also increases. To address this, the generation of a unified control system data model should be considered to reduce the memory size and scan time of the PLC control code.

5.4.6 Visibility of Interlocks

In the current software architecture, the interlock check is performed by the Logic Engine whereas the interlock data is stored in the Control System Data Model. As a result, the interlock information is difficult to access and modify at runtime. To address this issue, the interlock checking control code should be moved to the RCs and the interlock conditions should be automatically provided on the interface of the RCs. This will not only make the status of interlock conditions visible, but will also

enable the temporary bypassing of interlocks during maintenance or in the event of hardware failure.

5.4.7 RC-Based Manual Mode Screens Generation

In the current approach implemented by the author the manual mode screens are generated from the HMI Data Model. As the Data Block of RCs contains all the required information to generate the manual pushbuttons configuration for manual mode control, the potential to generate the manual mode screen by utilising the RC data should be investigated. This will potentially reduce the complexity of Control System Data Model and the memory size and scan time of control code.

5.4.8 Automatic RC Mapping within VE tools

Currently, the underlying data structure of the CCE Mapper is different from the data structure of the CCE tools. As RCs are stored in the CCE Mapper library therefore mapping of the RCs with CCE components is currently performed manually. To make the generation of code more efficient, the CCE Mapper should be integrated with the common data model of the CCE tools. This will enable the storage and mapping of RCs within the component library of the CCE tools during the pre-engineering phase. This wills potentially reduce the manual work and consequently the chances of human errors in the RC mapping during system engineering phase.

5.4.9 Formal Verification of Control Logic

The 3D CAD based virtual commissioning involves visual inspection of a machines operation to validate and debug its control logic. Such a method of validating the control logic is not been able to detect hidden errors such as deadlock and race conditions, which may only be triggered in the long run and can result in machine breakdown. To identify such errors in the control logic, a formal method (discussed in Section 2.6.4) of verification is desirable to perform rigorous mathematical analysis and prove that the control logic has the required properties and does not have any hidden errors. The application STDs i.e., of the components and Process Logic sequences could be used directly to automatically generate equivalent formal models (such as Petri Nets) for analysis.

5.4.10 Deployment of Distributed Control Systems

The proposed approach supports the runtime system engineering of centralised control systems. However, due to the inherent limitations of centralised control systems in terms of flexibility, reconfigurability and interaction with business systems, industry is striving to engineer distributed control system architectures that can effectively integrate with other systems. In a distributed control architecture, such as Service Oriented Architecture (SOA), the concept of horizontal and vertical communication between system components especially for monitoring and process control applications is advantageous. A number of research projects such as COMPAG [4, 132], SODA [133] SOCRADES [94], SIRENA [134] and AESOP [135] have implemented and demonstrated the use of distributed control and SOA in several application domains in manufacturing automation

systems. Based on the research outcomes of these projects, the proposed automatic code generation approach should be extended to investigate the direct deployment of SOA based intelligent control components from VE tools.

5.4.11 Evaluation

In this thesis, preliminary evaluation has been conducted to compare the proposed code generation approach with the equivalent traditional manual programing method. However, a more elaborate qualitative and quantitative evaluation, from the industrial acceptability perspective, should be carried out in the future in a realistic automation system supply chain environment. This could involve evaluating the structure of the code, time to make a given design change and the level of knowledge and programming experience required to be able to make a given change. The scope of the future analysis might cover the system development, reconfiguration, reusability, fault diagnostics and debugging. In each case, the automatic code generation approach should be compared with current best practices. Such a qualitative and quantitative evaluation would provide a more thorough overview of the benefits and limitations of the proposed code generation approach than could be provided here.

References

1. James A. Jordon, J. and F.J. Michel, *Next Generation Manufacturing: methods and techniques*. 2000, New York: John Wiley and sons, inc. 464.
2. Rogers, G.G. and L. Bottaci, *Modular production systems: a new manufacturing paradigm.* Journal of Intelligent Manufacturing, 1997. **8**(2): p. 147-156.
3. National Research Council, *VISIONARY MANUFACTURING CHALLENGES FOR 2020, Committee on Visionary Manufacturing Challenges*. 1998, National Research Council: Washington DC.
4. Lee, S.M., *A component-based distributed control paradigm for manufacturing automation system*, in *Wolfson School of Mechanical and Manufacturing Engineering*. 2004, Loughborough University: Leicestershire.
5. Zoitll, A., *Real-time execution for IEC 61499*. 2009: ISA.
6. Richardsson, J. and M. Fabian. *Automatic generation of PLC programs for control of flexible manufacturing cells*. in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*. 2003.
7. Richardsson, J. and M. Fabian, *Modeling the control of a flexible manufacturing cell for automatic verification and control program generation.* International Journal of Flexible Manufacturing Systems, 2006. 18(3): p. 191-208.
8. Henerick, O., G. Licht, and W. Sofka, *Challenges and oppurtunities for the European automotive industry*, in *Europe's automotive industry on the move: Competitiveness in the changing world*, O. Henerick, G. Licht, and W. Sofka, Editors. 2005, Physica-Verlog: Germany.
9. Harrison, R., A.A. West, and L.J. Lee. *Lifecycle Engineering of Future Automation Systems in the Automotive Powertrain Sector*. in *Industrial Informatics, 2006 IEEE International Conference on*. 2006.
10. Anon, *Annual report 2012*. 2012, Toyota Motor Corporation. p. 127.
11. Anon, *Challenges facing the global automotive industry.* inshights-Booz.Allen & Hamilton, 1999. **Vol 1**(1).
12. Lee, S., R. Harrison, A. West, and M. Ong, *A component-based approach to the design and implementation of assembly automation system.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2007. 221(5): p. 763-773.
13. Kidd, P.T., *Revolutioninsing new product development: a blueprint for success in the global automotive industry*. 1997, Management reprot, FT Automotive Publishing.
14. Westkämper, E., *New trends in production*, in *Reconfigurable Manufacturing Systems and Transformable Factories*, A.I. Dashchenko, Editor. 2006, Springer-Verlag: Netherlands. p. 759.
15. Dowlatshahi, S. and Q. Cao, *The relationships among virtual enterprise, information technology, and business performance in agile manufacturing: An industry perspective.* European Journal of Operational Research, 2006. 174(2): p. 835-860.
16. Harrison, R., A. Colombo, A. West, and S. Lee, *Reconfigurable modular automation systems for automotive power-train manufacture.* International Journal of Flexible Manufacturing Systems, 2006. **18**(3): p. 175-190.
17. Guttel, K., P. Weber, and A. Fay. *Automatic generation of PLC code beyond the nominal sequence*. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008.
18. Zhiqiang, G. and R.R. Rhinehart. *Theory vs. practice: the challenges from industry*. in *American Control Conference, 2004. Proceedings of the 2004*. 2004.
19. Thapa, D., C. Park, S. Park, and G.-N. Wang, *Auto-generation of IEC standard PLC code using t -MPSG.* International Journal of Control, Automation and Systems, 2009. **7**(2): p. 165-174.
20. Andersson, K., J. Richardsson, B. Lennartson, and M. Fabian, *Coordination of Operations*

by Relation Extraction for Manufacturing Cell Controllers. Control Systems Technology, IEEE Transactions on, 2010. **18**(2): p. 414-429.
21. Kong, X., B. Ahmad, R. Harrison, A. Jain, Y. Park, and L.J. Lee. *Realising the open virtual commissioning of modular automation systems*. in *7th CIRP International Conference on Digital Enterprise Technology*. 2011. Athens, Greece.
22. Makris, S., G. Michalos, and G. Chryssolouris, *Virtual Commissioning of an Assembly Cell with Cooperating Robots.* Advances in Decision Sciences, 2012. 2012: p. 11.
23. Reinhart, G. and G. Wünsch, *Economic application of virtual commissioning to mechatronic production systems.* Production engineering, 2007. 1(4): p. 371-379.
24. Ong, M.H., *Evaluating the impact of adopting the component-based system within the automotive domain*, in *Wolfson School of Mechanical and Manufacturing Engineering*. 2004, Loughborough University: Leicestershire.
25. Harrison, R., D. Vera, S. McLeod, and A. Jain, *Virtual commissioning methods and tools*. 2012, Loughborough University and Airbus.
26. Moore, P.R., J. Pu, H.C. Ng, C.B. Wong, S.K. Chong, X. Chen, J. Adolfsson, P. Olofsgård, and J.O. Lundgren, *Virtual engineering: an integrated approach to agile manufacturing machinery design and control.* Mechatronics, 2003. **13**(10): p. 1105-1121.
27. Anon, *DELMIA V5 automation platform: merging digital manufacturing with automation*. 2006, ARC Advisory Group.
28. Molina, A., C.A. Rodriguez, H. Ahuett, J.A. CortÃ©s, M. RamÃrez, G. JimÃ©nez, and S. Martinez, *Next-generation manufacturing systems: key research issues in developing and integrating reconfigurable and intelligent machines.* 2005. 18(7): p. 525 - 536.
29. Drath, R., P. Weber, and N. Mauser. *An evolutionary approach for the industrial introduction of virtual commissioning*. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008.
30. David, K., *Virtual commissioning of factory floor automation: the new paradigm in vehicle manufacturing*, in *SAE 2010 world congress & exhibition*. 2010: Detroit, USA.
31. Falkman, P., E. Helander, and M. Andersson. *Automatic generation: A way of ensuring PLC and HMI standards*. in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. 2011.
32. Bergert, M., C. Diedrich, J. Kiefer, and T. Bar. *Automated PLC software generation based on standardized digital process information*. in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. 2007.
33. Kumar, R., *Research methodology: a step-by-step guide for beginners* Second ed. 2005: SAGE Publications.
34. Jackson, M., *An Analysis of Flexible and Reconfigurable Production Systems*, in *Mechanical Engineering*. 2000, Linkopings Universitet: Linkoping.
35. Humphrey, J., Y. Lecler, and M. Salerno, *Introduction*, in *Global Strategies and Local Realities: The Auto Industry in Emerging Markets*, J. Humphrey, Y. Lecler, and M. Salerno, Editors. 2000, Macmillan Press Ltd.: London.
36. Goldman, S., R. Nagel, and K. Preiss, *Agile Competitors and Virtual Organizations - Strategies for Enriching the Customers*. 1995, New York: Van Nostrand Reinhold.
37. Anon, *Global Manufacturing Competitiveness Index*. 2010, Deloitte Development LLC.
38. Lee, W., T. Baines, B. Tjahjono, and R. Greenough, *Towards a conceptual framework of manufacturing paradigms.* SIMTech Technical Reports, 2006. **7**(3): p. 169-177.
39. Ford Motor Company. *The revolution of mass production*. 2013 [cited 2013 22 April]; Available from: http://www.ford.co.uk/experience-ford/Heritage/EvolutionOfMass Production.
40. Womack, J.P., D.T. Jones, and D. Roos, *The machine that changed the world*. 1990, New York: Simon & Schuster.
41. Dauguay, C.R., S. Landry, and F. Pasin, *From mass production to flexible/agile production.* International Journal of Operations & Production Management, 1997. 17(12): p. 1183-1195.
42. Lipietz, A., *Towards a new economic order: postfordism, ecology, and democracy*. 1992:

Oxford University Press.
43. Pine, B.J., *Paradigm shift: from mass production to mass customization*, in *Applied Mathematics*. 1980, University of Wisconsin.
44. Koren, Y., *The global manufacturing revolution: product-process-business Integration and reconfigurable systems*. Systems Engineering and Management. 2010, New Jersey: John Wiley & Sons. 399.
45. Ohno, T., *Toyota Production System: Beyond Large-Scale Production*. 1988, New York: Productivity Press.
46. The Manufacturer, *Fifth Lean Manufacturing Report*. 2006, http://www.themanufacturer.com.
47. Wiendahl, H.P., H.A. ElMaraghy, P. Nyhuis, M.F. Zäh, H.H. Wiendahl, N. Duffie, and M. Brieke, *Changeable Manufacturing - Classification, Design and Operation.* CIRP Annals - Manufacturing Technology, 2007. **56**(2): p. 783-809.
48. Haq, I., *Innovative Configurable and Collaborative Approach to Automation Ssytems Engineering for Automotive Powertrain Assembly*, in *Mechanical and Manufacturing Engineering*. 2009, Loughborough University: Loughborough.
49. Yusuf, Y.Y., M. Sarhadi, and A. Gunasekaran, *Agile manufacturing:: The drivers, concepts and attributes.* International Journal of Production Economics, 1999. 62(1-2): p. 33-43.
50. Masood, T., *Enhanced Integrated Modelling Appraoch to Reconfiguring Manufacturing Enterprises*, in *Wolfson School of Mechanical and Manufacturing Engineering*. 2009, Loughborough University. p. 314.
51. Gunasekaran, A. and Y.Y. Yusuf, *Agile manufacturing: A taxonomy of strategic and technological imperatives.* International Journal of Production Research, 2002. 40(6): p. 1357-1385.
52. Gunasekaran, A., *Agile manufacturing: enablers and an implementation framework.* International Journal of Production Research, 1998. 36(5): p. 1223 - 1247.
53. Kidd, P.T. *Agile Manufacturing: a strategy for the 21st century*. in *Agile Manufacturing, IEE Colloquium on*. 1995.
54. Gupta, U.G. and R.O. Mittal. *Quality, time, and innovation based performance measurement system for agile anufacturing*. in *Annual Meeting of the Decision Sciences Institute*. 1996.
55. Backhouse, C.J. and N.D. Burns, *Agile value chains for manufacturing-implications for performance measures.* International Journal of Agile Management Systems, 1999. **1**(2): p. 76-82.
56. Koren, Y., U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H.V. brussel, *Reconfigurable manufacturing systems*, in *manufacturing technologies for machines of the future*, A. dashchenko, Editor. 2003, Springer-Verlag: germany. p. 820.
57. ElMaraghy, H., *Flexible and reconfigurable manufacturing systems paradigms.* International Journal of Flexible Manufacturing Systems, 2005. 17(4): p. 261-276.
58. Koren, Y., U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel, *Reconfigurable Manufacturing Systems.* CIRP Annals - Manufacturing Technology, 1999. **48**(2): p. 527-540.
59. ElMaraghy, H. and H.P. Wiendahl, *Changeability-An Introduction*, in *Changeable and reconfigurable manufacturing systems*, H. ElMaraghy, Editor. 2009, Springer-Verlag.
60. Bi, Z.M., S.Y.T. Lang, W. Shen, and L. Wang, *Reconfigurable manufacturing systems: the state of the art*. 2008, Taylor & Francis. p. 967 - 992.
61. Scholten, B., *A road to Integration: A guide to applying the ISA-95 standard in manufacturing* 2007, USA: ISA-Instrumentation, Systems, and Automation Society.
62. Colombo, A.W., F. Jammes, H. Smit, R. Harrison, J.L.M. Lastra, and I.M. Delamer. *Service-oriented architectures for collaborative automation*. in *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*. 2005.
63. McLeod, C.S., *Development of a toolkit for component-based automation systems*, in *Wolfson School*. 2012, Loughborough University: Loughborough.
64. van der Wal, E. *IEC 1131 or 61131 : status of the Standard,*. 2009 [cited 2009; Available

from: http://www.plcopen.org/pages/tc1_standards/iec_1131_or_61131/.
65. Hoske, M.T., *Choose the right programming language.* Control Engineering, 2003. **50**(7): p. 52.
66. John, K.-H. and M. Tiegelkamp, *IEC 61131-3: Programming industrial automation systems*. 2001, Berlin Heidelberg: Springer-Verlag.
67. Estevez, E., M. Marcos, E. Irisarri, F. Lopez, I. Sarachaga, and A. Burgos. *A novel approach to attain the true reusability of the code between different PLC programming tools*. in *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop on*. 2008.
68. Lee, S., M.A. Ang, J. Lee, L. Lee, and D.M. Tilbury, *Automatic generation of logic control*. 2006, Loughborough University, University of Michigan and Ford Motor Company.
69. Hajarnavis, V. and K. Young, *An investigation into programmable logic controller software design techniques in the automotive industry.* Assembly Automation, 2008. 28(1): p. 45-54.
70. Hajarnavis, V. and K. Young. *A comparison of sequential function chart and object-modelling PLC programming*. in *American Control Conference, 2005. Proceedings of the 2005*. 2005.
71. Öhman, M., S. Johansson, and K.-E. Årzén, *Implementation aspects of the PLC standard IEC 1131-3.* Control Engineering Practice, 1998. 6(4): p. 547-555.
72. Ford Motor Company, *Structured transfer-machine EDDI programming system specification*. 1996.
73. Doughty, M.J. *The need for standardisation of diagnostic techniques [for PLC programs]*. in *Advances in Software Engineering for PLC*. 1993.
74. Lee, L.J., *A next generation manufacturing control system for a lean production environment*, in *Wolfson school of mechanical and manufacturing engineering*. 2004, Loughborough University: Leicestershire. p. 265.
75. Ford Motor Company, *DVM4 auto station engine assembly specification*. 2007.
76. Ford Motor Company, *FAST PLC structure manual.* 2013.
77. Roberts, R., *Zone Logic: A unique method of practical artificial intelligence*. 1989, Pennsylvania: Computer Books.
78. ThyssenKrupp System Engineering, *Function oriented modularity: A programming course*. 2008: Bremen.
79. Lucas, M.R., *Understanding and assessing logic control design methodologies*. 2003, University of Michigan: Michigan. p. 113.
80. Jain, A., D. Vera, and R. Harrison, *Virtual commissioning of modular automation systems*, in *10th IFAC workshop on intelligent manufacturing systems*. 2010, Elsevier: Lisbon, Portugal.
81. Estevez, E. and M. Marcos. *An approach to use model driven design in industrial automation*. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008.
82. Lucas, M.R. and D.M. Tilbury, *A study of current logic design practices in the automotive manufacturing industry.* International Journal of Human-Computer Studies, 2003. 59(5): p. 725-753.
83. Harrison, R. and A.W. Colombo. *Collaborative automation from rigid coupling towards dynamic reconfigurable production systems*. in *16th IFAC World Congress*. 2005. Czech Republic: elsevier.
84. Spath, D. and R. Landwehr, *Three-dimensional simulation and programming of PLC controlled manufacturing systems.* International Journal for Manufacturing Science and Production, 2001. Vol.4(4).
85. Spath, D. and U. Osmers. *Virtual reality-an approach to improve the generation of fault free software for programmable logic controllers (PLC)*. in *Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on*. 1996.
86. Harrison, R., S.M. Lee, and A.A. West. *Lifecycle engineering of modular automated machines*. in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International*

Conference on. 2004.
87. Magar, C.R., N. Jazdi, and P. Gohner. *Requirements on engineering tools for increasing reuse in industrial automation*. in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. 2011.
88. Brennan, R.W., P. Vrba, P. Tichy, A. Zoitl, C. Sünder, T. Strasser, and V. Marik, *Developments in dynamic and intelligent reconfiguration of industrial automation.* Computers in Industry, 2008. 59(6): p. 533-547.
89. Thapa, D., C.M. Park, K.H. Han, S.C. Park, and G.-N. Wang, *Architecture for modeling, simulation, and execution of PLC based manufacturing system*, in *Proceedings of the 40th Conference on Winter Simulation*. 2008, Winter Simulation Conference: Miami, Florida.
90. Tu, Q., M.A. Vonderembse, T.S. Ragu-Nathan, and B. Ragu-Nathan, *Measuring Modularity-Based Manufacturing Practices and Their Impact on Mass Customization Capability: A Customer-Driven Perspective.* Decision Sciences, 2004. **35**(2): p. 147-168.
91. Baldwin, C.Y. and K.B. Clark, *Design Rules: The power of modularity*. 2000, Boston: MIT Press.
92. Schilling, M.A., *Toward a general modular systems theory and its appliation to interfirm product modularity*, in *Managing in the modular age: Architectures, Networks, and organizations*, R. Garud, A. Kumaraswamy, and R.N. Langlois, Editors. 2003, Blackwell Publishing: Oxford.
93. Schafer, C., *On the modularity of manufacturing systems.* Industrial Electronics Magazine, IEEE, 2007. 1(3): p. 20-27.
94. Phaithoonbuathong, P., *Web service control of component-based agile manufacturing systems*, in *Mechanical and Manufacturing Engineering*. 2009, Loughborough University: Loughborough.
95. McFarlane, D. *Modular distributed manufacturing systems and the implications for integrated control*. in *Choosing the Right Control Structure for Your Process (Digest No. 1998/280), IEE Colloquium on*. 1998.
96. Vrba, P., M. Radakovič, M. Obitko, and V. Mařík, *Semantic technologies: latest advances in agent-based manufacturing control systems.* International Journal of Production Research, 2010. 49(5): p. 1483-1496.
97. Leitão, P., *Agent-based distributed manufacturing control: A state-of-the-art survey.* Engineering Applications of Artificial Intelligence, 2009. 22(7): p. 979-991.
98. Leitao, P., V. Marik, and P. Vrba, *Past, Present, and Future of Industrial Agent Applications.* Industrial Informatics, IEEE Transactions on, 2012. PP(99): p. 1-1.
99. Marik, V., P. Vrba, K.H. Hall, and F.P. Maturana, *Rockwell automation agents for manufacturing*, in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 2005, ACM: The Netherlands. p. 107-113.
100. Hegny, I., O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan. *Integrating software agents and IEC 61499 realtime control for reconfigurable distributed manufacturing systems*. in *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*. 2008.
101. Colombo, A.W., R. Schoop, P. Leitao, and F. Restivo. *A collaborative automation approach to distributed production systems*. in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*. 2004.
102. Szer-Ming, L., R. Harrison, and A.A. West. *A component-based distributed control system for assembly automation*. in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*. 2004.
103. Lee, S., R. Harrison, and A. West, *A component-based control system for agile manufacturing.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2005. 219(1): p. 123-135.
104. Jain, A., D.A. Vera, and R. Harrison, *Virtual comissioning of modular automation systems*, in *10th IFAC Workshop on Intelligent Manufacturing Systems*. 2010: Portugal.
105. Bergert, M. and J. Kiefer, *Mechatronic data models in production engineering*, in *10th IFAC Workshop on Intelligent Manufacturing System*. 2010, Elsevier: Lisbon, Portugal. p. 67-72.

106. Schmidgall, G., J. Kiefer, and B. Thomas. *Objectives of integrated digital production engineering in the automotive industry*. in *Proceedings of 16th IFAC World Congress*. 2005. Czech Republic: Elsevier.
107. Pinto, G., *Simulation in a virtual environment to operate with an automatic production line used in the automotive industry*, in *SAE technical paper series*. 2010.
108. Danielsson, K., J. Richardssorn, B. Lennartson, and M. Fabian. *Automatic scheduling and verification of the control function of flexible assembly cells in an information reuse environment*. in *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005). The 6th IEEE International Symposium on*. 2005.
109. Ljungkrantz, O., K. Akesson, J. Richardsson, and K. Andersson. *Implementing a Control System Framework for Automatic Generation of Manufacturing Cell Controllers*. in *Robotics and Automation, 2007 IEEE International Conference on*. 2007.
110. Steinegger, M. and A. Zolti, *Automated Code Generation for Programmable Logic Controllers based on Knowledge Acquisition from Engineering Artifacts: Concept and Case Study*, in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*. 2012 Kraków, Poland.
111. Hundt, L., A. Luder, A. Kohlein, and N. Gewald. *Methodology for the evaluation of tools with respect to its applicability within mechatronical engineering*. in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. 2011.
112. Ang, M., R. Harrison, J. Lee, L. Lee, S. Lee, and D.M. Tilbury, *A comparison study of automatic logic control generation tools for industrial manufacturing control systems*, in *2nd international conference on changeable, agile, reconfigurable, and virtual production*. 2007: Toronto, Ontario, Canada.
113. Ljungkrantz, O. and K. Akesson. *A Study of Industrial Logic Control Programming using Library Components*. in *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*. 2007.
114. Frey, G. and L. Litz. *Formal methods in PLC programming*. in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*. 2000.
115. Uzam, M., H. Jones, and I. Yücel, *Using a Petri-Net-Based Approach for the Real-Time Supervisory Control of an Experimental Manufacturing System.* The International Journal of Advanced Manufacturing Technology, 2000. 16(7): p. 498-515.
116. Feldmann, K., A.W. Colombo, C. Schnur, and T. Stockel, *Specification, design, and implementation of logic controllers based on colored Petri net models and the standard IEC 1131. II. Design and implementation.* Control Systems Technology, IEEE Transactions on, 1999. **7**(6): p. 666-674.
117. Lee, J.S. and P.L. Hsu, *An improved evaluation of ladder logic diagrams and Petri nets for the sequence controller design in manufacturing systems.* The International Journal of Advanced Manufacturing Technology, 2004. 24(3): p. 279-287.
118. Hajarnavis, V. and K. Young, *An Assessment of PLC Software Structure Suitability for the Support of Flexible Manufacturing Processes.* Automation Science and Engineering, IEEE Transactions on, 2008. 5(4): p. 641-650.
119. Endsley, E.W., E.E. Almeida, and D.M. Tilbury, *Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation.* Control Engineering Practice, 2006. 14(10): p. 1127-1142.
120. Thapa, D., S.C. Park, C.M. Park, and G.-N. Wang, *Modeling, verification, and implementation of PLC program using timed-MPSG*, in *Proceedings of the 2007 summer computer simulation conference*. 2007, Society for Computer Simulation International: San Diego, California. p. 533-540.
121. Park, C.M., S. Park, and G.-N. Wang, *Control logic verification for an automotive body assembly line using simulation.* International Journal of Production Research, 2009. 47(24): p. 6835-6853.
122. Ljungkrantz, O., K. Akesson, M. Fabian, and Y. Chengyin, *Formal Specification and Verification of Industrial Control Logic Components.* Automation Science and Engineering, IEEE Transactions on, 2010. **7**(3): p. 538-548.

123. Lucas, M.R. and D.M. Tilbury. *Comparing industrial logic design methods used in the automotive industry*. in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*. 2003.
124. Pellicciari, M., A. Andrisano, F. Leali, and A. Vergnano, *Engineering method for adaptive manufacturing systems design.* International Journal on Interactive Design and Manufacturing (IJIDeM), 2009. **3**(2): p. 81-91.
125. Flordal, H., M. Fabian, K. Åkesson, and D. Spensieri, *Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells.* Control Engineering Practice, 2007. 15(11): p. 1416-1426.
126. Harrison, R., *A component-based control system for agile manufacturing.* Engineering Manufacture, 2005.
127. Vera, D.A., A. West, and R. Harrison, *Innovative virtual prototyping environment for reconfigurable manufacturing system engineering.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2009. 223(6): p. 609-621.
128. Kong, X., *An approach to open virtual commissioning for component-based automation, Doctoral Thesis*, in *Wolfson School of Mechanical and Manufacturing Engineering*. 2013, Loughborough University: Loughborough, Leicestershire.
129. Lucas, M. and D. Tilbury. *Quantitative and qualitative comparisons of PLC programs for a small testbed with a focus on human issues*. in *American Control Conference, 2002. Proceedings of the 2002*. 2002. IEEE.
130. Hajarnavis, V. and K. Young. *A comparison of sequential function chart and object-modelling PLC programming*. in *American Control Conference, 2005. Proceedings of the 2005*. 2005. IEEE.
131. Siemens. *Siemens industry online support*. n.d. [cited 2013 15 Nov]; Available from: http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objid=7302549&caller=view.
132. Lee, S.-M., R. Harrison, and A.A. West. *A component-based distributed control system for assembly automation*. in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*. 2004.
133. Delsing, J., J. Eliasson, R. Kyusakov, A.W. Colombo, F. Jammes, J. Nessaether, S. Karnouskos, and C. Diedrich. *A migration approach towards a SOA-based next generation process control and monitoring*. in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*. 2011.
134. Bohn, H., A. Bobek, and F. Golatowski. *SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains*. in *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*. 2006.
135. AESOP. *Architecture for service-oriented process*. n.d. [cited 2013 27 November]; Available from: http://www.imc-aesop.eu.

Publications

- **Ahmad, B.**, Kong, X., Harrison, R., Watermann, J., Colombo, A. W., (2013) "Automatic generation of Human Machine Interface screens from component-based reconfigurable virtual manufacturing cell" In Proceedings of Annual Conference of the IEEE Industrial Electronics Society, IECON 2013-39th, Vienna, Austria

- Kaur, N., McLeod, C. S., Jain, A. , Harrison, R. , **Ahmad, B.** , Colombo, A. W. & Delsing (2013) "Design and simulation of a SOA-based system of Systems for automation in the residential sector" In Proceedings of IEEE International Conference on Industrial Technology (ICIT), South Africa

- X. Kong, **B. Ahmad**, R. Harrison, Y. Park, Leslie J Lee (2012), "Direct deployment of component-based automation systems" In Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Krakow, Poland

- Kong, X., **Ahmad, B**., Harrison, R., Jain, A., Park, Y., Lee, L., (2011), "Realising the open virtual commissioning of modular automation systems", Proceedings of the 7th CIRP International Conference on Digital Enterprise Technology (DET), Athens, Greece

- Haq, I., Masood, I., **Ahmad, B.**, Raza, B., Monfared, R., Harrison, R., (2011), "Product to process lifecycle management in assembly automation systems", Proceedings of the 7th CIRP International Conference on Digital Enterprise Technology (DET), Athens, Greece