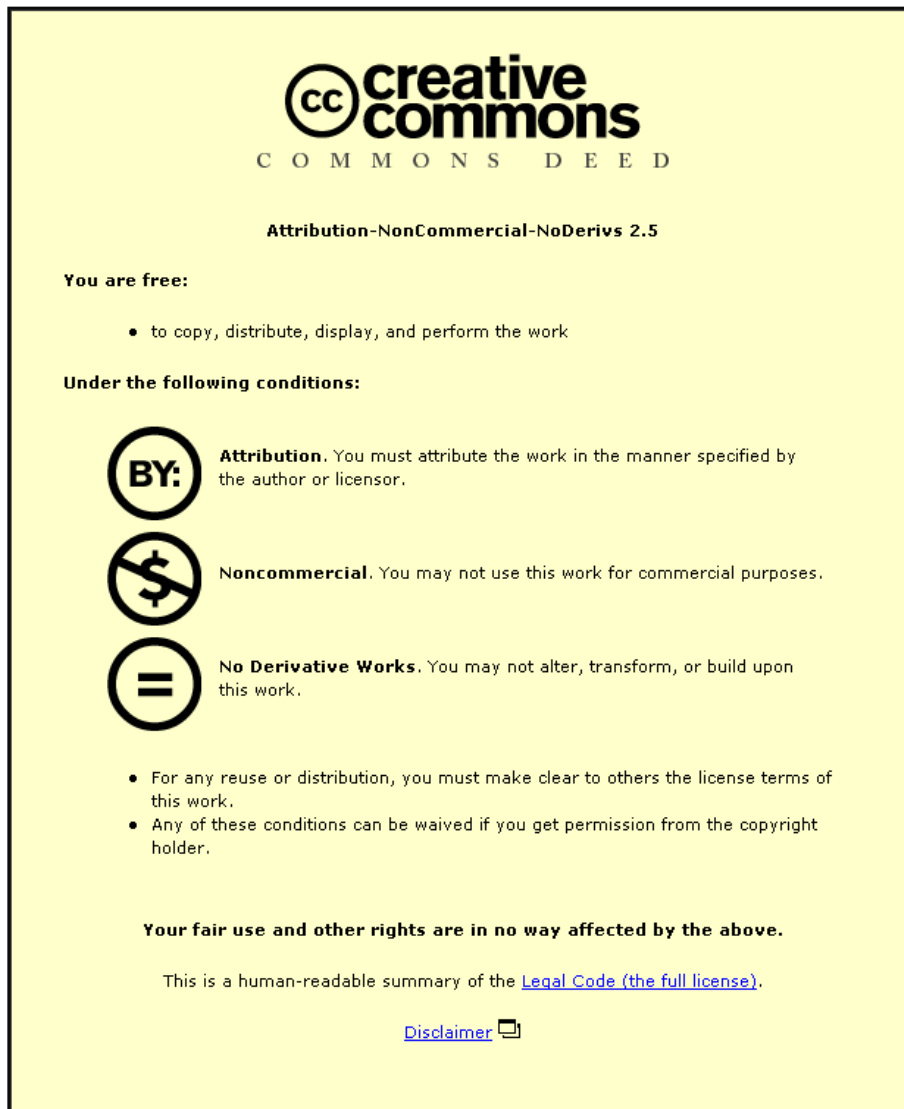


This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.




CC creative commons
COMMONS DEED


Attribution-NonCommercial-NoDerivs 2.5


You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

 **Attribution.** You must attribute the work in the manner specified by the author or licensor.


 **Noncommercial.** You may not use this work for commercial purposes.

 **No Derivative Works.** You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#) 

For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

An Approach to Open Virtual Commissioning for Component-Based Automation

By

Xiangjun Kong

**Doctoral thesis submitted in partial fulfilment of the requirements
of the award of Doctor of Philosophy of Loughborough University**

Loughborough University

November 2013

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Professor Robert Harrison for his invaluable encouragement, patience, support and guidance during the course of this research. My gratitude goes beyond the words I could write on this page.

I would also like to give my gratitude to my colleagues of the ASG research group who have helped and taught me so much, especially Bilal Ahmad, Youngsaeng Park and Charles Stuart McLeod for many constructive discussions that contributed to my knowledge in the area of automation as well as software engineering.

My gratitude also goes to Professor Richard Weston for his encouragement and help in the process of my PhD application.

I am also grateful to my special friends – Cunjia Liu, Yunnan Gao, Jian Ma, Zhaoting Xiong, Yan Zhang, Yang Liu, Zihua Cui and Ting Wang for their friendship and support.

Most importantly, I must recognise the sacrifice made by my family and their support throughout my years of study. I wholeheartedly thank my parents and parents-in-law for their support and love. I am profoundly thankful to my wife and best friend, Peinan Li, for her support, encouragement and love. I have been blessed with a remarkable wife.

Abstract

Increasing market demands for highly customised products with shorter time-to-market and at lower prices are forcing manufacturing systems to be built and operated in a more efficient ways. In order to overcome some of the limitations in traditional methods of automation system engineering, this thesis focuses on the creation of a new approach to Virtual Commissioning (VC).

In current VC approaches, virtual models are driven by pre-programmed PLC control software. These approaches are still time-consuming and heavily control expertise-reliant as the required programming and debugging activities are mainly performed by control engineers. Another current limitation is that virtual models validated during VC are difficult to reuse due to a lack of tool-independent data models. Therefore, in order to maximise the potential of VC, there is a need for new VC approaches and tools to address these limitations.

The main contributions of this research are (1) to develop a new approach and the related engineering tool functionality for directly deploying PLC control software based on component-based VC models and reusable components, and (2) to build tool-independent common data models for describing component-based virtual automation systems in order to enable data reusability. This research is part of the BDA (Business Driven Automation) project conducted by the Automation Systems Group. The aim of BDA is to provide a new component-based VC framework which minimises the time and expertise required to commission automation systems. This thesis details the development of the approaches as well as related engineering tool features required by the new VC framework for achieving control software deployment and efficient data reuse. In order to simplify PLC control software development, a novel approach and related engineering tool functions for directly deploying the PLC control software of automation systems based on the control behaviour of the component-based virtual models have been designed and implemented. To achieve efficient reuse of validated virtual models, a new common data model for describing the component-based virtual systems has been developed utilising the domain-specific open standard-AutomationML.

Keywords: Component-Based Automation, Virtual Commissioning, Direct Deployment, Common Data Model, AutomationML.

Table of Contents

ABSTRACT	I
LIST OF FIGURES	VI
LIST OF TABLES	IX
ABBREVIATIONS	X
CHAPTER 1. INTRODUCTION	1
1.1. BACKGROUND	1
1.2. RESEARCH MOTIVATION.....	1
1.2.1. <i>Justification for Research</i>	3
1.2.2. <i>Problem Statement</i>	3
1.3. RESEARCH DESCRIPTION.....	4
1.3.1. <i>Hypothesis</i>	4
1.3.2. <i>Objectives</i>	4
1.3.3. <i>Focus and Related Research</i>	5
1.3.4. <i>Methodology</i>	7
1.3.5. <i>Contributions</i>	9
1.3.6. <i>Research Scope</i>	10
1.4. THESIS STRUCTURE.....	10
CHAPTER 2. LITERATURE REVIEW: STATE-OF-THE-ART AUTOMATION SYSTEM ENGINEERING	12
2.1. RECONFIGURABLE MANUFACTURING SYSTEMS	12
2.1.1. <i>The Need for RMS</i>	12
2.1.2. <i>Re-configurability</i>	13
2.1.3. <i>Modular Approaches to Automation System Engineering</i>	14
2.2. PLC CONTROL SYSTEM ENGINEERING	21
2.2.1. <i>Current Practices in PLC Control Software Development</i>	22
2.2.2. <i>Emerging Approaches to Efficient Development of Control Logic</i>	26
2.3. VIRTUAL COMMISSIONING.....	34
2.3.1. <i>Overview</i>	34

2.3.2.	<i>Hardware/Software-In-the-Loop (HIL/SIL)</i>	37
2.3.3.	<i>CCE - A VC Engineering Tool for VCOM</i>	41
2.4.	OPENNESS OF VC	43
2.4.1.	<i>Why openness needed - Potentials of reusing virtual models</i>	45
2.4.2.	<i>How to achieve openness – Approaches to data exchange</i>	48
2.4.3.	<i>Data Representation of VC Models</i>	49
2.4.4.	<i>Tool-independent Data Description</i>	51
2.5.	ASSESSMENT AND SUMMARY	59
2.5.1.	<i>Assessment of state-of-the-art</i>	59
2.5.2.	<i>Identification of research gaps</i>	60
CHAPTER 3.	APPROACH AND METHODOLOGY	62
3.1.	VCOM - A NEW OPEN VC FRAMEWORK.....	62
3.1.1.	<i>The Need for the Open VC Framework</i>	62
3.1.2.	<i>Overview of the Open VC Framework</i>	63
3.2.	VIRTUAL MODULAR COMMON DATA MODEL (VMCDM)	65
3.2.1.	<i>Basis of VMCDM</i>	65
3.2.2.	<i>VMCDM for Component-based Automation Systems</i>	70
3.2.3.	<i>VMCDM-specific Role Classes and Interface Classes</i>	72
3.2.4.	<i>Element representation</i>	74
3.2.5.	<i>Virtual Component</i>	78
3.3.	MAPPING COMPONENT-BASED VIRTUAL MODELS TO VMCDM	80
3.3.1.	<i>Overall process</i>	80
3.3.2.	<i>Semantic Bridging</i>	83
3.4.	DEPLOYABLE CONTROL SOFTWARE ARCHITECTURE	88
3.4.1.	<i>PLC control system</i>	88
3.4.2.	<i>HMI</i>	91
3.5.	DIRECT DEPLOYMENT OF CONTROL SOFTWARE.....	92
3.5.1.	<i>Overview</i>	92

3.5.2.	<i>Development of Reusable Static Data</i>	94
3.5.3.	<i>Dynamic Generation of Runtime Control Models</i>	98
3.5.4.	<i>Dynamic Generation of Logic Depository</i>	101
3.5.5.	<i>I/O Mapping for Actuator/Sensor Components</i>	102
3.5.6.	<i>Dynamic Generation of Programs for Actuators and Sensors</i>	102
3.5.7.	<i>Generation and Output of Complete Control Code</i>	104
3.6.	SOFTWARE FOR DATA MAPPING AND DIRECT DEPLOYMENT.....	105
3.6.1.	<i>Software Architecture Design</i>	105
3.6.2.	<i>System Design</i>	106
3.7.	CHAPTER OVERVIEW	110
CHAPTER 4.	IMPLEMENTATION AND EXPERIMENTAL STUDY	112
4.1.	PROTOTYPE IMPLEMENTATION	112
4.1.1.	<i>VMCDM Prototype</i>	112
4.1.2.	<i>VCMapper Prototype</i>	114
4.2.	OVERVIEW OF EXPERIMENTS.....	123
4.2.1.	<i>Introduction to experiment resources</i>	123
4.2.2.	<i>Case Studies</i>	128
4.3.	VIRTUAL MODEL MAPPING EXPERIMENT.....	130
4.3.1.	<i>Overview</i>	130
4.3.2.	<i>CCE Virtual Model Export</i>	131
4.3.3.	<i>Mapping CCE to VMCDM</i>	132
4.3.4.	<i>Data Reuse of VMCDM</i>	134
4.3.5.	<i>Evaluation</i>	135
4.4.	DIRECT DEPLOYMENT EXPERIMENT	136
4.4.1.	<i>Overview</i>	136
4.4.2.	<i>Platform-specific common information development</i>	137
4.4.3.	<i>Control software deployment</i>	141
4.4.4.	<i>Commissioning</i>	143

4.4.5. Evaluation	149
4.5. SUMMARY OF CHAPTER	161
4.5.1. About the VMCDM and virtual model mapping.....	162
4.5.2. About the direct deployment solution.....	162
CHAPTER 5. CONCLUSIONS	164
5.1. CONCLUSIONS	164
5.2. RESEARCH CONTRIBUTIONS	165
5.3. FUTURE WORK.....	166
5.3.1. Potential enhancements	166
5.3.2. Future research directions.....	168
PUBLICATIONS.....	170
REFERENCES	171

List of Figures

List of Figures

FIGURE 1-1: RESEARCH FOCUS AND OTHER RELATED RESEARCHES.....	5
FIGURE 1-2: DEVELOPMENT WORK CONDUCTED BY THE AUTHOR AND OTHER ASG RESEARCHERS.....	6
FIGURE 2-1: CURRENT PROCESS OF AUTOMATION SYSTEM ENGINEERING [1].....	15
FIGURE 2-2 INFORMATION SETS OF MECHATRONICAL OBJECT [20]	16
FIGURE 2-3 AGENT-BASED CONTROL IN MANUFACTURING [23].....	18
FIGURE 2-4 CONTROL SYSTEM WITH AGENT AND HOLON [27].....	19
FIGURE 2-5 SYSTEM LEVEL CONTROL DEVELOPMENT BY INTERLOCKING COMPONENTS [8]	21
FIGURE 2-6 OVERVIEW OF LOGIC DEVELOPMENT PROCESS [34].....	22
FIGURE 2-7 FOM STRUCTURE [12]	26
FIGURE 2-8 REQUIRED DETAILS AND RESOURCES OF DIFFERENT VIRTUAL ENGINEERING TECHNOLOGIES	35
FIGURE 2-9 TIME BENEFIT OF VC [7]	36
FIGURE 2-10 SOFTWARE IN THE LOOP (SIL) AND HARDWARE IN THE LOOP (HIL)	37
FIGURE 2-11 SYSTEM ARCHITECTURE FOR HIL/SIL	38
FIGURE 2-12 THEORETICAL BASIS OF CCE - THE COMPONENT-BASED APPROACH	42
FIGURE 2-13 COMPONENT BEHAVIOUR MODELLING IN CCE	43
FIGURE 2-14 BUILDING SYSTEM CONTROL LOGIC BY INTERLOCKING BEHAVIOURS OF COMPONENTS.....	43
FIGURE 2-15 COMMISSIONING REQUIRES DATA FROM DIFFERENT PHASES	44
FIGURE 2-16 DATA FLOW OF VIRTUAL COMMISSIONING.....	44
FIGURE 2-17 VIRTUAL MODELS USED AS REFERENCE MODELS DURING OPERATIVE CONTROL [66]	46
FIGURE 2-18 SCENARIOS OF REUSING HIL MODELS DURING SYSTEM OPERATION PHASE	47
FIGURE 2-19 NEED FOR DATA REUSE DRIVEN BY THE NEED OF OEMS.....	48
FIGURE 2-20 EXISTING APPROACHES OF DATA EXCHANGE.....	49
FIGURE 2-21 VIRTUAL MODELS INTEGRATES INFORMATION FROM MULTI-DISCIPLINES.....	49
FIGURE 2-22 XML-BASED COMMUNICATION AND INTEGRATION IN DIGITAL FACTORY [69]	52
FIGURE 2-23 AUTOMATIONML REDUCES COMPLEXITY AND CLOSES GAPS [106].....	54
FIGURE 2-24 ARCHITECTURE OF AUTOMATIONML[106].....	55
FIGURE 2-25 INFORMATION COVERED BY AUTOMATIONML[106]	56
FIGURE 3-1 THE OBJECTIVE OPEN VC FRAMEWORK - VCOM	64
FIGURE 3-2 CAEX ITEMS AND THEIR RELATIONS [116]	68
FIGURE 3-3 CAEX, AUTOMATIONML AND VMCDM AND RESPECTIVE CONTRIBUTIONS	70
FIGURE 3-4 CONSTITUENTS OF A VIRTUAL COMPONENT.....	71

List of Figures

FIGURE 3-5 STRUCTURE RELATIONSHIP BETWEEN OBJECTS OF VMCDM	72
FIGURE 3-6 USER-DEFINED INTERFACE - “VCINTERFACE” FOR CONNECTING STATES WITH PLC CONTROL LOGIC	73
FIGURE 3-7 DATA STRUCTURE OF AN ACTUATOR ELEMENT.....	77
FIGURE 3-8 DATA STRUCTURE OF STATIC ELEMENT.....	78
FIGURE 3-9 ACTUATOR COMPONENT DESCRIBED IN UML.....	79
FIGURE 3-10 SENSOR COMPONENT REPRESENTED IN VMCDM	80
FIGURE 3-11 SYSTEMUNITCLASS FOR NON-CONTROL COMPONENT.....	80
FIGURE 3-12 PROCESS OF DATA EXCHANGE BASED ON NEUTRAL DATA MODEL.....	81
FIGURE 3-13 ONTOLOGY MAPPING PROCESS [119]	82
FIGURE 3-14 PROCESS OF MAPPING FROM CCE VIRTUAL MODELS TO VMCDM.....	83
FIGURE 3-15 SEMANTIC BRIDGING FOR MAPPING CCE TO VMCDM.....	84
FIGURE 3-16 ARCHITECTURE MAPPING DESCRIBED IN UML	85
FIGURE 3-17 DATA MODEL MAPPING BY CALLING FUNCTION IN XSLT	86
FIGURE 3-18 OVERALL CONTROL SYSTEM ARCHITECTURE	88
FIGURE 3-19 COMPONENT-BASED PLC CONTROL SYSTEM ARCHITECTURE	89
FIGURE 3-20 COMPONENT-BASED PLC CONTROL SOFTWARE ENGINEERING.....	93
FIGURE 3-21 REVERSE ENGINEERING PROCESS OF DIRECT DEPLOYMENT	96
FIGURE 3-22 TEMPLATE OF THE SHARED DB HEADER IN SIMATIC STEP7	97
FIGURE 3-23 WORKFLOW OF AUTOMATIC-MODE CONTROL MODEL GENERATION.....	98
FIGURE 3-24 WORKFLOW OF MANUAL MODE CONTROL MODEL GENERATION	100
FIGURE 3-25 STATE BEHAVIOURS AND THE CORRESPONDING HMI CONTROL PANEL	101
FIGURE 3-26 PROCESS OF GENERATING PROGRAM(S).....	103
FIGURE 3-27 SOURCE CODE INTEGRATION AND EXPORT	104
FIGURE 3-28 ARCHITECTURE OF THE VCMAPPER.....	105
FIGURE 3-29 DATA FLOW DIAGRAM OF VCMAPPER.....	107
FIGURE 3-30 WORKFLOW OF VCMAPPER	108
FIGURE 3-31 UNIFIED INTERFACE FOR DIFFERENT CODE GENERATION MODULES.....	109
FIGURE 4-1 DEVELOPMENT OF VMCDM	113
FIGURE 4-2 RELATIONSHIP BETWEEN THE USER INTERFACE AND UNDERLYING MODULES	115
FIGURE 4-3 USER INTERFACE FOR I/O MAPPING	116
FIGURE 4-4 XML FILE FOR I/O MAPPING	118
FIGURE 4-5 DATABASE TABLES FOR STORING REUSABLE DATA.....	123
FIGURE 4-6 THE EXPERIMENTAL TEST BED.....	124

List of Figures

FIGURE 4-7 DECOMPOSITION OF THE TEST RIG.....	125
FIGURE 4-8 EXAMPLE OF ACTUATOR COMPONENTS – SWIVEL ARM (PHYSICAL, VIRTUAL AND STATE BEHAVIOUR).....	126
FIGURE 4-9 VIRTUAL PROTOTYPE OF THE TEST RIG.....	127
FIGURE 4-10 CONTROL LOGIC OF STATION 1 (PUSHER, SWIVEL, VACUUM)	127
FIGURE 4-11 CONTROL LOGIC INFORMATION OF EXPORTED CCE MODELS IN XML FILES.....	132
FIGURE 4-12 HIERARCHY INFORMATION OF EXPORTED CCE MODELS IN XML FILES	132
FIGURE 4-13 TRANSFORM CCE VIRTUAL MODEL OF THE TEST BED TO VMCDM.....	133
FIGURE 4-14 THE VMCDM OF THE TEST BED OPENED IN THE AUTOMATIONML EDITOR.....	135
FIGURE 4-15 PROCESS OF TESTING BY THE DIRECT DEPLOYMENT SOLUTION.....	137
FIGURE 4-16 EXAMPLES OF THE PLCOPENXML TEMPLATES	139
FIGURE 4-17 EXAMPLE OF S7 SPECIFIC OBJECTS – HEADER OF OB1	140
FIGURE 4-18 EXAMPLE OF S7-SPECIFIC OBJECT – INSTANCE DATA BLOCK	141
FIGURE 4-19 I/O MAPPING FOR ACTUATORS AND SENSORS.....	142
FIGURE 4-20 COMMISSIONING ENVIRONMENT FOR PLCOPENXML.....	144
FIGURE 4-21 COMPONENT SWIVEL ARM AND ITS RELATED CONTROL CODE IN CODESYS V3.....	145
FIGURE 4-22 PROCESS OF S7 CONTROL SOFTWARE DEPLOYMENT	146
FIGURE 4-23 COMPONENT PUSHER AND ITS RELATED CONTROL CODE IN STEP 7.....	147
FIGURE 4-24 SCREENS OF GENERATED HMI	148
FIGURE 4-25 WORK FLOW OF MANUAL CONTROL VIA HMI	149
FIGURE 4-26 SCENARIOS OF RECONFIGURING THE TEST RIG	152
FIGURE 4-27 RUNTIME COMPONENT (AN ACTUATOR AND A SENSOR).....	158
FIGURE 4-28 CONTROL MODELS REPRESENTED AS ARRAYS OF UDTs (DIFFICULT TO READ)	160

List of Tables

List of Tables

TABLE 2-1 KEY CHARACTERISTICS OF A RECONFIGURABLE MANUFACTURING SYSTEM	14
TABLE 2-2 ACTIVITIES FOR CONTROL SOFTWARE DEVELOPMENT.....	23
TABLE 2-3 IEC61131-3 PROGRAMMING LANGUAGES [35].....	24
TABLE 2-4 SUMMARY OF EXISTING APPROACHES TO AUTOMATIC CODE GENERATION	33
TABLE 2-5 DATA REPRESENTATION IN REPRESENTATIVE VC TOOLS	51
TABLE 2-6 COMPARISON OF EXISTING DATA REPRESENTATION METHODS AND FORMATS.....	58
TABLE 2-7 A SUMMARY OF THE STATE OF THE ART	60
TABLE 3-1 SUMMARY OF RESEARCH WORK COVERED	64
TABLE 3-2 ROLE CLASSES DEFINED IN THIS RESEARCH FOR DESCRIPTION OF VC MODELS.....	73
TABLE 3-3 REQUIRED FUNCTIONS FOR ONTOLOGY MAPPING.....	87
TABLE 3-4 DERIVED DATA TYPES FOR COMPONENT-BASED CONTROL LOGIC.....	95
TABLE 4-1 DECOMPOSITION OF THE TEST BED	125
TABLE 4-2 RUNTIME COMPONENTS FOR THE FESTO RIG.....	128
TABLE 4-3 DATA FORMATS OF THE SOURCE TOOL AND THE TARGET TOOL	130
TABLE 4-4 ROLE OF EACH COMPONENT OF THE TEST BED.....	133
TABLE 4-5 HARDWARE AND SOFTWARE USED FOR THE EXPERIMENTS.....	136
TABLE 4-6 CONSTITUENTS OF THE PLCOPENXML COMMON INFORMATION (ALSO CALLED TEMPLATES).....	138
TABLE 4-7 COMPONENTS OF S7 TEMPLATES.....	139
TABLE 4-8 SOURCE CODES EXPORT FOR STEP 7.....	143
TABLE 4-9 TIME TO DEVELOP CONTROL SOFTWARE OF TEST RIG USING TWO SELECT APPROACHES.....	151
TABLE 4-10 TIME TO RECONFIGURE THE TEST RIG	154
TABLE 4-11 COMPARISON OF PLC PROGRAMS.....	156
TABLE 4-12 COMPARISON OF SCAN TIME	157
TABLE 4-13 : COMPARISON OF EFFORTS TO DIAGNOSTIC AND DEBUG	161

Abbreviations

Abbreviations

ASG	Automation Systems Group
CAEX	Computer Aided Engineering Exchange
CBA	Component-Based Approach
CCE	Core Component Editor
DOM	Document Objective Model
HIL	Hardware In the Loop
HMI	Human Machine Interface
OLE	Object Link and Embedding
OPC	OLE for Process Control
OPC UA	OPC Unified Architecture
PLC	Programmable Logic Controller
RMS	Reconfigurable Manufacturing System
SIL	Software In the Loop
STD	State Transition Diagram
UI	User Interface
VCOM	Virtual Commissioning using Components
VMCDM	Virtual Modular Common Data Model
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
XSLT	Extensible Stylesheet Language Transformations

Chapter 1. Introduction

This chapter introduces the area of research and targeted problem, the objectives, and the adopted methodologies.

1.1. Background

Nowadays manufacturing enterprises are under unprecedented pressure resulting from the turbulent market environments with aggressive competition on a global scale [1-3]. Due to the competition for key market share, automotive enterprises are forced to shorten production time when introducing new products. It is recognised that the delay in the launch of a new product can directly cause a significant reduction in profit margin [4].

Traditional manufacturing automation systems are normally implemented in rigid hierarchical structures. In the current approach, the design, build and validation of automation systems take place sequentially and system validation cannot be carried out until the last stage of the system's development, when all electrical, mechanical units and the control software have been integrated. It is obvious that any unforeseen delays that occur during these activities will result in the delay of succeeding activities and hence delay the system delivery date. This adversely affects the lead time of a production machine and thus results in a failure to gain a competitive edge and market share [1]. Moreover, such an engineering approach heavily relies on the knowledge and experience of the engineering team. The control code developed for such systems is often monolithic and unstructured, making it difficult to understand, modify and reuse. Due to this, any alteration in the automation system is time consuming, complex, error prone and expensive. This results in an adverse impact on the commissioning and ramp-up time and might lead to performance degradation.

1.2. Research Motivation

To gain a competitive edge in the market by providing more product variants more rapidly, innovative approaches to automation system engineering are required to achieve agility in the manufacturing systems. An important consideration is that new production systems must be scalable in capacity and functionality thereby making them able to convert quickly to produce new products [5]. In this context, modular production systems, which are one type of Reconfigurable Manufacturing System (RMS), are designed at the onset to be re-configurable

and created from basic hardware and software modules which can be re-arranged quickly and reliably [6]. Using a modular approach, machine builders are able to build a new system by combining the needed components, potentially from different component vendors, without the need to understand their potentially complex implementation details. The modular approaches radically change the way of automation system engineering and can significantly reduce the complexity of control systems engineering.

However, few of existing modular approaches have been applied to large scale industrial applications. PLC-based control systems are widely used by industry and the time to build and validate such systems increases as the system complexity grows [4]. However, the competition for key market shares makes shorter time in production ramp-ups of key importance [7]. The correction of defective control software consumes up to 60% of commissioning time and accounts for 15% of time-to-delivery [7]. This challenge can be relieved by Virtual Commissioning (VC), in which a virtual model of the to-be system is used to validate the control software on an actual Programmable Logic Controller (PLC) and Human-Machine Interface (HMI) before the physical integration of all the devices occurs on the shop-floor, thereby a saving of ramp-up time can be achieved.

Current VC approaches can be classified into Hardware-in-the-Loop (HIL) and Software-in-the-Loop (SIL) [7]. The SIL approach includes a simulation of the production equipment as well as the control hardware itself. Therefore, it can be carried out without the control system hardware. In a HIL simulation, on the other hand, the control software are tested under more realistic conditions by connecting the virtual model of a machine to real control hardware, thereby avoiding making changes to the software runtime environment afterwards. In general, the HIL and SIL approaches both have their respective advantages and shortcomings summarised as follows:

- HIL has been widely accepted to perform virtual commissioning as it realises the validation against real PLC control code. However, HIL relies on the expertise of control engineers since the PLC codes need to be manually developed before VC and subsequently the debugging of the PLC control software during VC is also the responsibility of control engineers. Obviously, control hardware is required and the connections with virtual models need to be created first.

- SIL does not require physical PLCs and therefore the connections between real PLC and virtual models are not required. However, manual programming and debugging is still required. Another penalty is that low availability of up-to-date control simulation packages for a particular PLC normally leads to less realistic commissioning.
- Additionally, the virtual models validated by either HIL or SIL can rarely be shared between different related engineering tools as no common data models are available for achieving efficient data exchange of these virtual models. As a result, the virtual models are mainly restricted to be (re)used by the engineering tool in which they are created.

1.2.1. Justification for Research

Automation systems have a key role to play in the process of building and operating manufacturing systems in the following aspects:

- Machine ramp-up time
- Control software validation during commissioning
- Remote support and diagnostic during machine operative phase

The inherent attributes of VC make it an appropriate way to relieve the challenges raised from the above factors. However, it is observed that limitations still exist in the current VC approaches. Time-consuming and expertise-reliant manual programming is still required and validated virtual models cannot be efficiently reused during the lifecycle of automation systems. Research is therefore needed to further explore the potentials of VC to meet the increasing demand for more a rapid and economical way of automation system engineering and operation.

1.2.2. Problem Statement

In order to develop and manage automation systems rapidly and cost-efficiently through maximising the contributions of VC, VC approaches should become more efficient and validated virtual models need to be efficiently reused. Considering the plurality of PLC platforms and the diversity of current VC engineering tools, the problem statement can be therefore formulated as follows:

How to (1) enable a VC approach which better facilitates manufacturing systems building and the resultant control software development, and (2) impact systems development and management throughout its lifecycle through efficient reuse of validated virtual models?

1.3. Research Description

1.3.1. Hypothesis

The principle hypotheses for this work include (1) if an automation system has been virtually prototyped and commissioned using the component-based approach in a virtual engineering tool, the required PLC control software of this system can be directly deployed based on the component-based virtual model and pre-developed reusable runtime components, and (2) the virtually built components and systems can be further reused during the system's lifecycle and this can be achieved via a tool-independent common data model described in an open standard.

1.3.2. Objectives

The research work documented in this dissertation is part of the group research work carried out by the Automation Systems Group (ASG) which was formerly at Loughborough University UK and moved to the University of Warwick UK in April 2013. This research work mainly took place at Loughborough University and now continues at the University of Warwick. The overall objective of the group's research is to develop an innovative open VC framework, which is named **VCOM** (**V**irtual **C**ommissioning using **C**omponents) and presented in the section 3.1 of this thesis. The research required to achieve the desired framework is summarised in Figure 1-1. Of this research, the component-based approach as the theoretical basis of the whole research work has been proposed by Harrison et al. in [4, 8]. The component-based simulation engineering toolset, introduced in the section 2.3.3.2, and the new control software architecture for facilitating the direct deployment of PLC control software based on the component-based virtual models, presented in the section 3.4, were respectively developed by other researchers of this group.

The research work conducted by the author, illustrated as the green part of Figure, is focused on the technologies to enable the openness and the direct deployment function of this framework. The principal objective is to develop the required approaches and engineering tools for achieving control software deployment and efficient virtual model reuse. To be more

specific, the objectives are to (1) propose and develop a solution that supports the direct deployment of PLC control software based on the control logic of the component-based virtual models, (2) develop open standard-based common data models to describe virtual models of component-based automation systems, and (3) develop related tool functionality to map component-based virtual models to the common data models. In order to support lossless and efficient data reuse, the proposed common data models must cover the multi-disciplinary data which comprise the virtual models. To facilitate the efficient development of error-free PLC control software, the proposed approach to direct deployment must maximise the automated reuse of validated component-based control logic and minimise required manual work in the process of deployment.

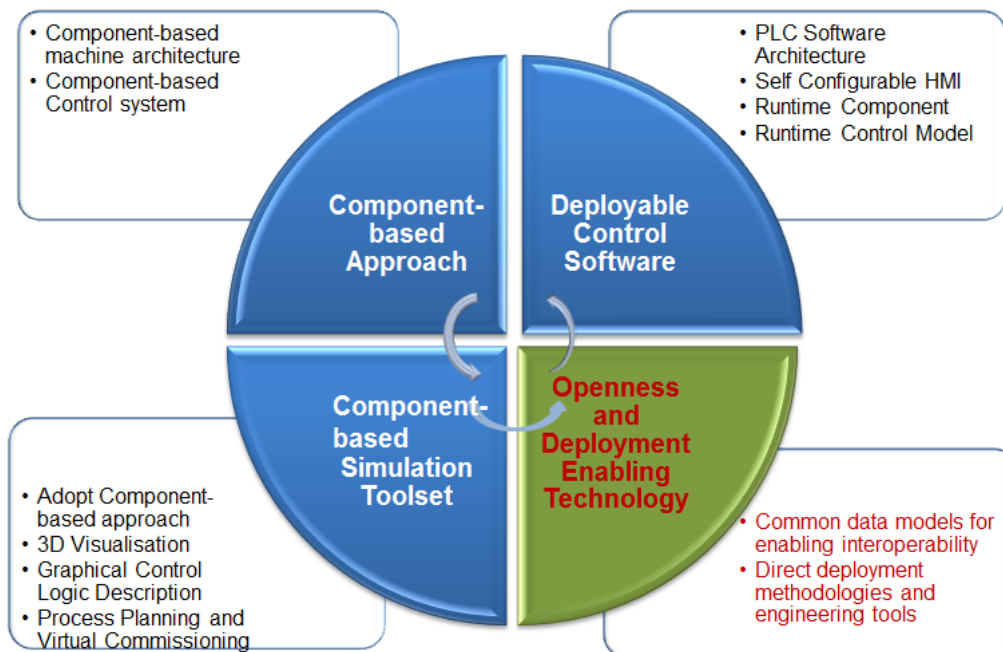


Figure 1-1: research focus and other related researches

1.3.3. Focus and Related Research

The focus of the author's research has been placed on proposing new approaches to achieve the objectives specified above. In order to demonstrate the achievement of these objectives, an engineering tool to implement the proposed approaches to direct deployment and virtual model mapping is needed. In this research, the required engineering tool has been developed jointly by the author and another researcher of the ASG. Also, some of the functions developed by the author are according to the new PLC control software architecture and the new HMI software architecture developed by other researchers of ASG. In order to

clarify the development work conducted by the author and its relationship with the work of other ASG researchers, Figure 1-2 illustrates the division of the related work.

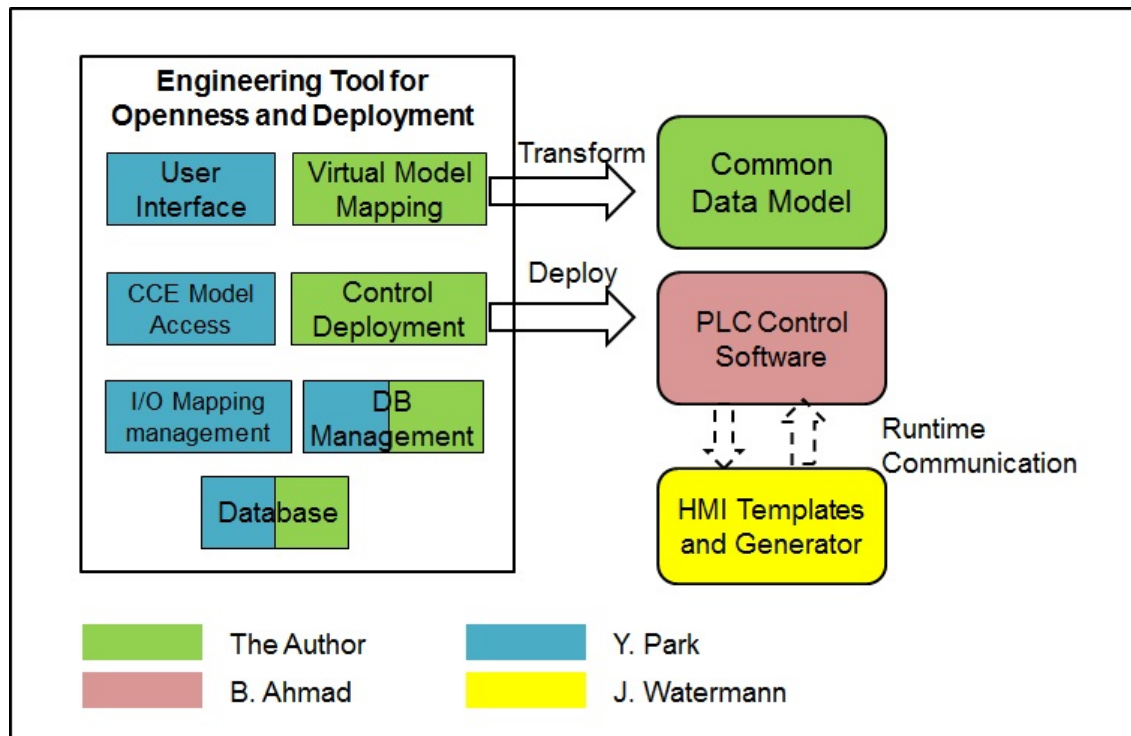


Figure 1-2: Development work conducted by the author and other ASG researchers

In the process of developing the engineering tool (detailed description can be found in section 4.1.2), the author was mainly focused on designing and implementing:

- The common data models for describing component-based virtual models
- The virtual model mapping module of the engineering tool for mapping component-based virtual models to the common data models
- The control software deployment module

The author was also involved in the joint design and implementation of the following modules of the engineering tool:

- Database
- Database management module

The new deployable control software architecture, according to which the PLC control software can be generated by the control deployment module, is described in section 3.4. The new HMI software is able to generate the HMI screens based on the predefined HMI screen

templates and the runtime control models in PLC program. The HMI software is used for the experimental studies (see section 4.4.4.2) of this research and more details about it can be found in [9, 10].

The engineering tool for virtually prototyping component-based automation systems is described in Section 2.3.3 and more details can be found in the dissertations of another researcher[11]. Also, it is not the purpose of this work to develop the reusable runtime components which are developed and validated by control engineers during component building phase and are used as black-boxes in direct deployment of control software for systems. Details of developing the runtime components can be found in [12].

1.3.4. Methodology

The following steps are followed in the documented research.

Survey of related methods and engineering tools

Extensive review of literature and industrial practices precedes the development of any original contribution. In order to propose domain-specific common data model, available VC engineering tools and data exchange formats related to automation system engineering have been reviewed. Existing methods of PLC software development from both academic researchers and commercial vendors have been assessed, in order to leverage an applicable solution for component-based direct deployment of control software during subsequent research activities.

Design of a direct deployment approach

A direct deployment approach is proposed and developed in order to reuse the control logic lying in the validated virtual models and to subsequently achieve the direct deployment of complete PLC control software. This is achieved by:

1. Specifying the workflow for component-based automatic generation of PLC control software.
2. Defining data structures for describing the simulated control logic as PLC-interpretable runtime control models.

3. Providing a means to automatically translate the control behaviours of virtual actuator components into runtime control models for populating the Human Machine Interface (HMI).
4. Providing a means to automatically populate the aforementioned runtime control models by extracting the control logic from virtual models.
5. Providing a means to build the communication between the runtime control models and corresponding runtime components.

Design and development of common data models

Assuming that the hypothesis holds, common models that enable efficient data reuse should be designed and developed and represented using an appropriate open standard data formats selected from the surveyed ones.

Development of a virtual model mapping approach

An approach to mapping component-based virtual models into the proposed HIL common data models is designed and also implemented using XSLT. This is needed due to the fact that HIL is currently the most widely accepted VC approach and the HIL models can be potentially reused throughout the lifecycle of automation systems.

Development of the related engineering tool

An engineering tool is developed in order to implement the proposed approaches. The engineering tool supports automatic generation of executable control software for the selected PLCs and the mapping of virtual models. These functionalities are realised by:

1. Implementing the proposed direct deployment approach.
2. Managing the reusable runtime components, PLC-specific common information and other reusable data.
3. Providing a user interface for facilitating I/O mapping.
4. Automatically generating source code of executable PLC control software by combining related reusable data with runtime control models that are dynamically generated.
5. Calling the developed XSLT files to implement virtual model mapping function.

Empirical study

A set of experiments has been carried out in order to validate the hypothesis of this research and to evaluate the performance of the proposed approaches and the developed engineering tool. A test rig and the component-based simulation engineering toolset named Core Component Editor (CCE) are used to illustrate the approach in different cases of interest:

1. Data exchange through the proposed common data model: requires virtual model of the test rig developed in CCE to be mapped into the proposed common data models which are then reused by another engineering tool.
2. Automatic generation of PLC control software with automatic operation mode: demonstrates the applicability of the approach to generating executable PLC control software for two PLCs from different vendors.
3. Automatic generation of PLC control software including both automatic and manual operation modes: demonstrates the applicability of the approach to generating both executable PLC control software and the related control data models for HMI software for the Siemens S7 platform.

1.3.5. Contributions

The main contributions presented by this dissertation can be classified into new methodologies and new technologies introduced herein.

1.3.5.1. New methodology

Component-based approach to automated control software engineering

A new approach in which PLC control software can be directly deployed based on the component-based control logic which is validated by CCE. In this approach, PLC control software of a desired machine is deployed by automatically combining pre-defined runtime components with runtime control models, generated by translating the component-based control logic, so that required manual work during the system engineering phase has been significantly reduced. This approach reduces the time and complexity of control software development. Hence, control software engineering does not rely so heavily on the expertise of control engineers and can be performed at an earlier phase of automation system engineering. This approach significantly overcomes the limitations in existing VC approaches.

Common data models for Modular Automation Systems

A set of new data models for describing virtual models of Component-based Automation Systems is proposed. These new models represent the components of a virtual modular automation system as a whole by integrating the constituent information from different disciplines. The model is represented by combining well-recognised XML-based domain specific open standards which provide semantics to the data models; therefore it significantly facilitates the data reuse of virtual models.

1.3.5.2. New Technology

Engineering tools and solutions

A consolidated set of functions and user interfaces that serve as an infrastructure for interpreting virtual models and generating executable PLC control software are developed. The engineering tool also provides functions for transforming the component-based virtual models into the corresponding proposed HIL common data models. This eliminates the data exchange gaps between VC applications and other related applications. The engineering tool, which simplifies the process of control system engineering by automating the majority of the required work, provides the required functions for achieving enhanced agility, reusability and adaptability in reconfigurable manufacturing systems.

1.3.6. Research Scope

Automation can be applied to many domains such as packaging, warehousing and building automation and each of them has its domain specific requirements. However, the author has been involved in the project for discrete part manufacturing with in the automotive industry. Therefore, the work presented in this dissertation has been carried out focusing on the domain of automation of automotive assembly.

1.4. Thesis Structure

The rest of the dissertation is structured as follows. Chapter 2 provides a review of relevant methods, engineering tools, data formats and research practices, and concludes with an assessment of the state of the art in the field of the dissertation. Chapter 3 presents the proposed data models, approaches and the design of the engineering tool to implement the proposed approaches. Chapter 4 describes the implementation of the engineering tool, the conducted experimental work to demonstrate that the research hypothesis is held and to

Chapter 1 Introduction

evaluate the performance of the proposed approaches. Chapter 5 summarises the main contributions of the work and suggests potential future research work.

Chapter 2. Literature Review: State-Of-The-Art Automation System Engineering

This chapter provides a review and assessment of existing relevant work. The chapter begins with a review of terms related to reconfigurable manufacturing systems and system architectures proposed by the state-of-the-art modular approaches to automation system engineering.

Two groups of methodologies of PLC control software engineering are then separately reviewed. Traditional approaches, which are being widely used by the current manufacturing industry, to developing and verifying control software are covered first. Subsequently, emerging approaches from both academia and industries for facilitating the control software engineering are then reviewed.

Consequently, Virtual Commissioning (VC), which is considered as a promising way of reducing the time and cost of control software validation, is then separately reviewed. The HIL and SIL approaches as the current mainstream approaches to VC, are reviewed. The CCE toolset, a component-based simulation engineering tool used for the new VCOM approach is then described.

Lastly, openness of VC tools is assessed from the perspectives of why openness is needed, how to achieve the needed openness, the openness of existing engineering tools and existing relevant open standards which can be potentially adopted to achieve the needed openness.

The chapter concludes with an assessment the state of the art and an identification of the research gaps.

2.1. Reconfigurable Manufacturing Systems

2.1.1. The Need for RMS

In order to cope with the new challenges characterised by growing demand for more customer-oriented product variants with reducing development time, an important consideration is that new production systems must be scalable in capacity and functionality thereby making them able to convert quickly to produce new products [5]. In this context, the Reconfigurable Manufacturing System (RMS) paradigm is widely considered as promising

key technology to enable responsiveness in the mass-customisation production era [13]. RMS is a new manufacturing paradigm aiming at meeting the objectives of cost-effective and rapid system changes. Compared with Flexible Manufacturing Systems (FMS) providing generalised flexibility, RMS provides customised flexibility [14]. Moreover, with RMS, more economic objectives can be achieved by permitting: a) reduced lead time to launch new systems and reconfiguration of existing systems, and b) rapid upgrading and quick integration of new functionalities into existing systems [5].

2.1.2. Re-configurability

According to Koren and Ulsoy, an RMS has the ability to reconfigure hardware and control resources at all of the functional and organisational levels, in order to quickly adjust the production capacities and functionalities in response to sudden changes in market or in regulatory requirements[15]. The characteristic feature that defines an RMS is the possibility of being changed easily in order to adapt to changing production requirements.

2.1.2.1. Qualitative Attributes

Reconfigurability refers to the possibility of making changes to a system in order to implement a different set of processes. Theoretically speaking, any system can be reconfigured if enough effort is invested [16]. Practically, for a system to be considered as an RMS it must be possible to make changes with minimum effort. Since to date there is no method for calculating how much effort is required to reconfigure a system before knowing the type of reconfigurations that will be needed, it is only possible to consider reconfigurability as a qualitative attribute.

Several attributes of system architecture can define the reconfigurability of a system in the different scenarios and stages. Table 2-1 summarises a compilation of qualitative attributes [5] that can be used to assess reconfigurability.

It must be noted that different approaches to building RMSs might have different strategies to achieve specific attributes of system reconfigurability in one way or another. A typical example is that Delamer considered modularity as the most influential facilitator for achieving integrability, convertibility and reusability [16]; while Harrison et al. emphasised the importance of reusability in building reconfigurable modular automation systems [6].

Table 2-1 Key characteristics of a reconfigurable manufacturing system

Attribute	Description
Modularity	Design all system components, both software and hardware, to be modular.
Integrability	Design systems and components for both ready integration and future introduction of new technology
Reusability	The capability to reuse elements in different systems without making changes to those elements
Convertibility	Allow quick changeover between existing products and quick system adaptability for future products.
Adaptability	The capability of the system to adapt to different known situations, i.e. to make changes within the intrinsic flexibility of the system.
Interoperability	The capability to utilise different types, models of elements those implement the same functionality in different systems.

2.1.2.2. Critical Design Issues

The reconfigurability of RMS can be achieved by reconfiguring hardware and/or software resources. System reconfigurability can be classified in terms of the levels where the reconfigurable actions are taken. Reconfigurability, at lower levels, is mainly achieved by changing hardware resources while it is achieved, at higher levels, mainly by changing software resources. The critical issues in designing a RMS can be categorised as architecture design, configuration design and control design, all of which were reviewed in details in [17].

- **Architecture design** is to design system components as encapsulated modules and define their interactions for the options when the modules are assembled.
- **Configuration design** determines system configuration under given system architecture for a specific task. A configuration is an assembly of the selected modules in order to fulfil the given task optimally.
- **Control design** is to design the appropriate control software for a reconfigurable system so that a configuration can be operated to fulfil the task satisfactorily.

2.1.3. Modular Approaches to Automation System Engineering

The traditional approach to automation system engineering is supported by well-established and well proven methods [18]. This approach is relatively effective but the process of designing and building the automation system is almost entirely sequential and

heavily segmented into different engineering disciplines. As shown in Figure 2-1, in the traditional approach, the design, implementation and validation of automation systems take place sequentially. This leads to time-consuming tasks which are difficult to accommodate unexpected changes occurring during the task duration and heavily rely on the knowledge and experience of the engineer team. The control codes developed for such systems are often monolithic and unstructured, making them difficult to understand, modify and reuse. Due to this, any alteration in the automation system is time consuming, complex, error prone and expensive. This results in an adverse impact on the commissioning and ramp-up time and also leads to performance degradation[8].

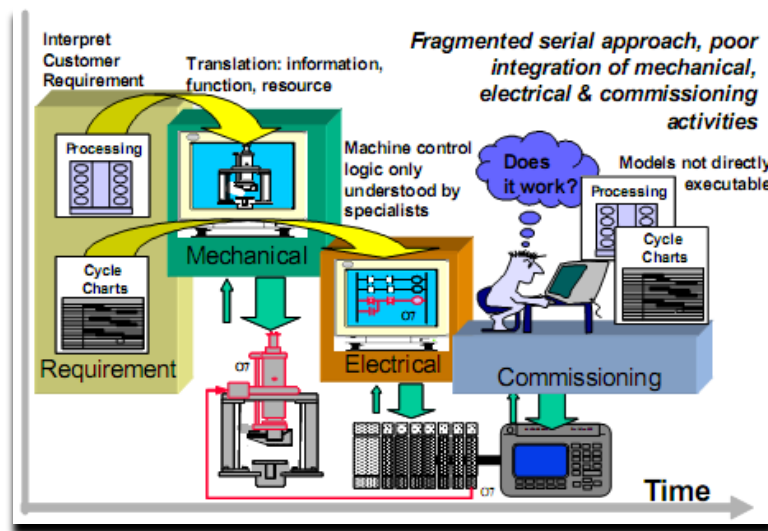


Figure 2-1: Current process of automation system engineering [1]

Modular approaches have been regarded as an answer for organizations to manage complexities and adapt to changes rapidly. Instead of building a system from scratch in a sequential manner, modular approaches facilitate system development using previously developed system elements. It encourages and enables system development by building on and reusing past experiences and knowledge.

The advantages of modular approaches can be summarised as low degree of coupling, concurrent and independent component build and validation, and reduced system ramp-up time. These will bring significant benefits to industry. According to Harrison and Colombo, a potential saving of 20 million Euros can be gained by saving 50% in the ramp-up time on a typical European automotive engine production line installation project [19]. To gain these advantages, reconfigurable modular production systems are designed at the onset to be re-

configurable and created from basic hardware and software modules that can be re-arranged quickly and reliably.

2.1.3.1. Mechatronic-based Modular Architecture

The basic hardware elements of assembly automation systems are field-level sensors and actuators, which are combined with mechanical structures to create composite production units that implement assembly or material handling processes. In the current modular approaches found in the relevant literatures, a recent trend is to encapsulate sensors and actuators into modular subsystems called mechatronic devices, which typically encapsulate additional control software elements. Once defined, these mechatronic units can be used as a whole along the process of whole production system engineering.

For different modular approaches, terms used for naming the basic constituent blocks can be various. Also, the approaches of defining the control software encapsulated in constituent blocks normally vary from one modular approach to another. However, these approaches are all typically mechatronic-based, in which mechatronic units are considered as manufacturing components with embedded control intelligence. A mechatronic unit can be generally described as a combination of physical, electrical, and control elements.

The structure of a complete mechatronic unit was summarised and illustrated in Figure 2-2. For different kinds of application areas, mechatronic units might contain only part of the attributes illustrated in Figure 2-2[20].

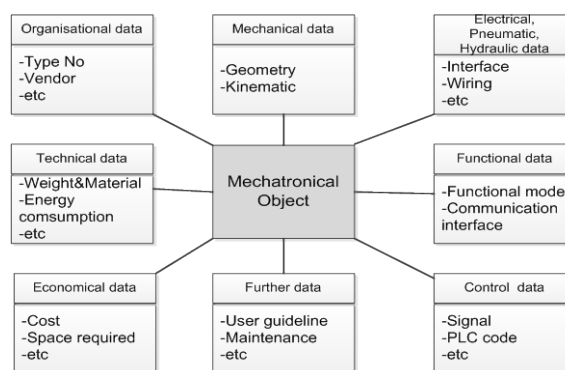


Figure 2-2 Information sets of mechatronic object [20]

2.1.3.2. Modular Control Software

At the hardware level, with mechatronic devices, rapid integration and reconfiguration can be enabled. However, a huge reprogramming effort is still required at the software level [21].

In addition, compared to hardware reconfiguration, the integration and reconfiguration of software elements is more complex and currently requires highly qualified and trained labour[16]. In this context, most of the efforts had been invested on the modularity of the control software. The control software elements can be functionally classified into the logic control and the coordination control according to their levels of control. Different modular approaches respectively focus on different levels to achieve reconfigurability.

The coordination control sometimes also called supervisory control organises the execution sequence of logic control applications. The elements of coordination/supervisory control provide the set-point control signals to logic-controlled operations. The events used by the coordination control algorithms notify of conditions such as reaching the commanded set point. By configuring the logic-controlled operations, the control signal set points and the sequence of invocations, different types of processes can be achieved. In many cases this functionality could be implemented directly within the logic control, for instance to coordinate the devices which compose a mechatronic machine that won't change its structure. But for many cases in which modules are elements of a reconfigurable system the coordination control was separated from the logic control [21].

A. Agent-based Modular Control

Many modular approaches aim at revising the coordination/supervisory control to facilitate the reconfiguration. Of these modular approaches, agent-based paradigms and multi-agent modular paradigms seem promising and have been heavily studied. This can be observed from the large amount of existing relevant literature. Leitao reviewed existing agent-based approaches and summarised an agent as “*An autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it does not possess knowledge and skills to reach alone its objectives*”[22] .

The most important properties of an agent were identified as the autonomy, intelligence, adaptation and co-operation. Of these properties, autonomy and intelligence refer to the ability to act autonomously to deal with unpredictable circumstances, while adaption and cooperation refer to the ability to communicate and collaborate with other agents or effective components to achieve the best solution for the control task [22]. An example of a multi-agent system was given by Colombo as shown in Figure 2-3. In this system, a set of agents

represent the objects of a system. In such a system, the agents need to be able to communicate in order to achieve a pre-defined goal or solve a problem. Agents interact with others, when some of them do not have enough knowledge and/or skills to achieve individually their objectives. These features allow a high performance against disturbances. In addition, the expansibility of the system is easier by only modifying the functioning of some agents or adding new agents to the control system[23].

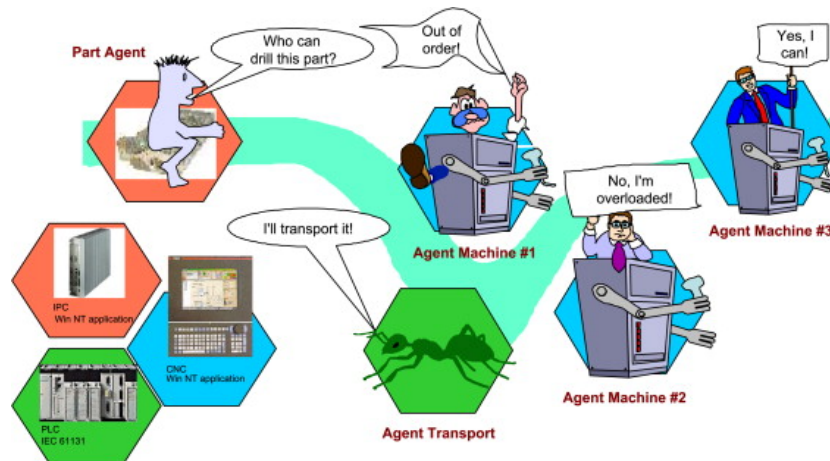


Figure 2-3 Agent-based control in manufacturing [23]

B. Holonic Modular Control

Holonic control is another modular approach which is similar to agent-based approach. A holon can represent a physical or logical activity, such as a robot, a machine, an order, a flexible manufacturing system or even an human operator[24]. Agent-based and holonic manufacturing paradigms have been developed under the same fundamental principles of autonomy and co-operation. The implementation of the holonic manufacturing concepts can be done using agent technology. The use of agent technology addresses mainly the high-level of abstraction [25], as illustrated in Figure 2-4. In the low level control, logic control functions interconnect with the physical sensors and actuators. Currently, the lowest level real-time control is usually carried out by industrial PLCs running in a classical scan-based manner. In the area of holon and multi-agent automation systems, the Rockwell Automation has invested heavily and has presented a set of methodologies and tools to support the development of the agent-based applications. The details of the research activities of Rockwell Automation can be found in [26].

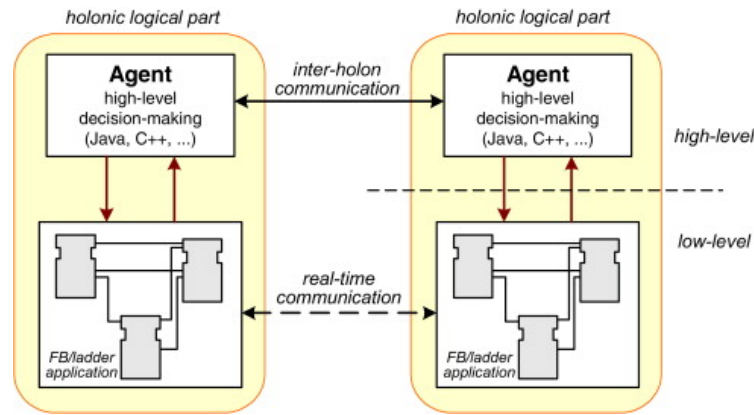


Figure 2-4 Control system with agent and holon [27]

C. IEC61499 Standard for Distributed Control

The IEC61499 standard is developed by the International Electrotechnical Commission (IEC) as an extension of the IEC61131 standard [28], which will be introduced in Section 2.2.1.3. The IEC 61499 standard is mainly used for the development of distributed control systems. It defines a distributed model for splitting different parts of an industrial automation process and complex machinery control into functional modules called function blocks. Apart from the normal function blocks introduced by the IEC61131, IEC61499 also defines communication function blocks which can coordinates the communication between normal function blocks. The communication function blocks can be programmed in different programming languages and support different forms of network access. Therefore, the function blocks of control software can be distributed to different network devices and can communicate with each other through communication function blocks [29].

IEC61499 has been regarded by many researchers as a basis to resolve the requirements for portability, configurability and interoperability of control systems [29, 30]. However, the literature review also suggests that it has been mainly promoted by the academic community and some researchers express doubt about whether the aforementioned advantages can be brought to control systems [31].

D. Component-based Approach to Logic/Loop Control

Logic control is achieved using control algorithms that process the data provided by sensors and command signals to the actuators at the device level. The plant models used at the loop control level can be either continuous time dynamic models or discrete-event dynamic models. In the scope of assembly automation, the logic control is based on discrete-

event dynamic models, which normally use logic-based evaluation of sensor and control signals to create different actuation signals when certain discrete conditions are met.

The literature suggests that many of the approaches to modular control were focused on revising the coordination control paradigm to achieve reconfigurability. On the other hand, modular approaches which focus on logic control are rare. A typical example of such an approach found in the existing literature is the Component-Based Approach (CBA) proposed by Harrison et al [4, 8]. The CBA proposes a new approach of implementing a fully distributed network-based control system without any master controller. The automation system is composed of autonomous mechatronic units known as components.

In the CBA, a component physically is a mechatronic-like unit which is composed of a microprocessor, interface electronics and the automation device. The component can be integrated into the desired automation system as a common reusable building block without the need to know its low-level implementation details. The interfacing electronics are used to condition and translate the output control signals from the controller to the automation device and the input signals from the device to the controller. Therefore, a component is a self-contained unit that is ready to be deployed immediately to an automation system.

Apart from the physical elements, a component also contains a control application which describes the generic control behaviour of the automation device. Sensors can be potentially contained in a component if it is necessary to provide local close-loop control. The application provides local control to the automation hardware and communicates with other components in the network in a peer-to-peer fashion. The control behaviour of a component is represented by a Finite State Machine (FSM). The FSM provides an abstract description of the component's embedded control behaviour. A FSM is an abstract machine that has only a finite, constant number of *states*. Each state has *transitions* to one or more states. The transition from one state to another is governed by *conditions* or rules. Some examples of the control behaviours represented in FSM were shown in Figure 2-5.

Regarding the coordination/supervisory control between the components, in the CBA it is realised by interlocking components together directly without using agent-like supervisory components. Via the interface, components can be interlocked - associating the conditions for the transition as a logical combination of the states of other components. The states of the component (FSM) are available through the component's logical interface. System

application is defined through the process of component interlocking. An example of the interlocking between components can refer to Figure 2-5 and more pertinent details can be found in [8]. This paradigm allows a component to be developed separately and then reconfigured (interlocked) later to particular control system requirements.

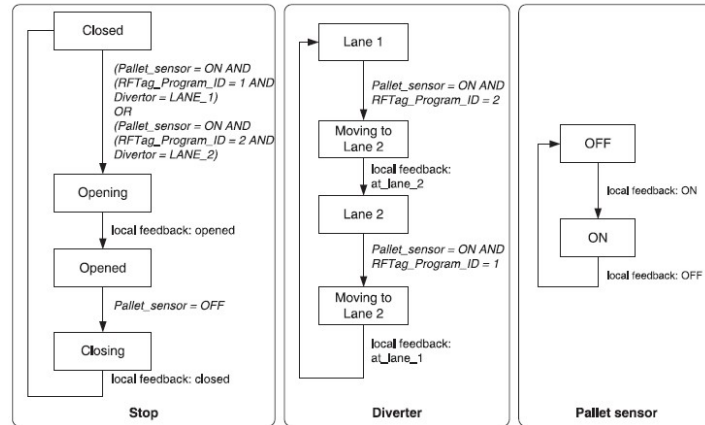


Figure 2-5 System level control development by interlocking components [8]

Apart from the CBA, there were also a few similar approaches found from existing research; however, few details about their design and implementations are available. For example, the VIR-ENG project designed and implemented a virtual environment for validating modular automation systems [32]. The virtual system was built based on mechatronic models and using traditional PLC-based logic control system. However, few details about its control software have been presented.

2.2. PLC Control System Engineering

The aforementioned modular approaches brought radical changes to traditional processes of automation control system engineering. However, these new paradigms have only led to some laboratorial prototypes or industrial test prototypes. Few of them have actually resulted in a large scale of industrial applications. This is mainly due to several issues related to conceptual efficiency and some development-related issues, both of which were summarised by Leitao in [22]. In the current practice of many industries, PLCs are still established as the device of choice for implementing the control functionality [33].

PLC control software development is one of the most time-consuming and important portions of control system engineering. This section first reviewed the industrial practices in

PLC control software development and then reviewed the related emerging methodologies from both academia and industry.

2.2.1. Current Practices in PLC Control Software Development

2.2.1.1. General Process of Control Logic Development

The process of control-related development in current industries was overviewed by Lucas and Tilbury [34] and was illustrated in Figure 2-6. The control engineers get project-specific specifications from engineers of other disciplines both before the commencement of the logic design. Project specific requirements include details about the actions the machine must perform to create parts, diagrams of physical and electrical components, and a description of the diagnostics desired. Control engineers combine the project-specific specifications with an additional set of standard specifications, which are usually from previous projects, to create the logic needed to control the machine. The standard specifications include the details of implementing the system and also include the needed safety and reliability requirements. On the other hand, some potentially unspecified requirements could be given to the control engineer during the logic design phase. The unspecified requirements normally are late changes or unexpected constraints in the machine or electronics.

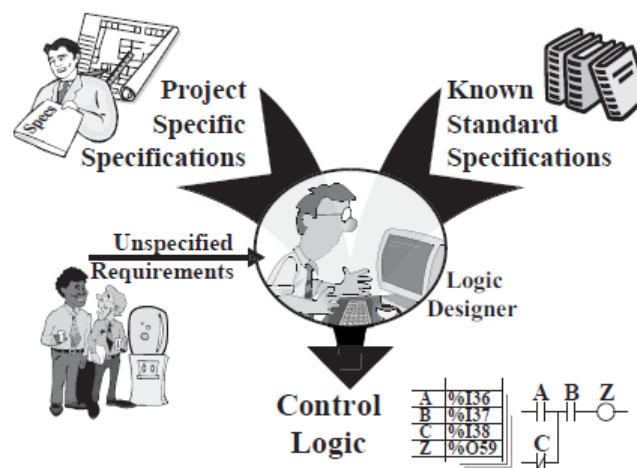


Figure 2-6 Overview of logic development process [34]

2.2.1.2. Activities in Control Software Development

In the current practices of industrial control development, most of the development work is mainly completed by control engineers despite the input of requirements from engineers of other disciplines. Activities that are performed by the control engineers to successfully

generate industrial logic can be further categorised into project coordination and documentation, memory management, logic programming, diagnostic and HMI development and the final debugging step. Based on the study conducted by Lucas and Tilbury in [34], these activities and their respective key features are briefed in Table 2-2.

Table 2-2 Activities for control software development

Activity	Descriptions and Features
Project coordination and documentation	<ul style="list-style-type: none"> Coordinating between control engineers working on a project to ensure consistent communications between various processors in the system.
Memory management	<ul style="list-style-type: none"> Creating or modifying the manually allocated PLC memory space of the project.
Logic programming	<ul style="list-style-type: none"> Developing the desired control program by creating new logic or potentially reusing existing logic from different sources. Code reuse Mainly through copying and pasting
HMI & Diagnostic development	<ul style="list-style-type: none"> Creating an HMI with the functions for manual control and diagnostics and connecting the HMI with the corresponding PLC control software. Mainly based on vendor-specific templates. PLC control, HMI and diagnostics are developed separately and connected manually.
Debugging	<ul style="list-style-type: none"> Testing developed logic and making any necessary changes. Mainly performed by control engineers on real machine

2.2.1.3. Control Software Programming Standard

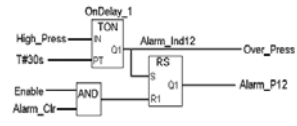
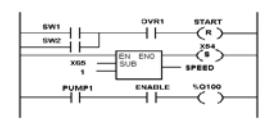
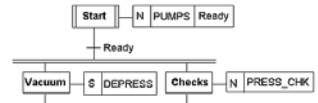
The size and complexity of some software application programs, which typically lack modularity and are difficult to reuse, has led to considerable research into structured methods of programming. Additionally, the increasing number of control hardware options and related programming languages results in inefficient reusability and maintainability and has aroused the need for standard programming languages.

In this context, many initiatives and research projects have been conducted aiming at providing either standard programming languages or uniform program structures. Resultant achievements include the standard programming languages defined in IEC61131-3, related standards defined by PLCOpen organisation [35] for facilitating the application of IEC61131-3, EDDI from Ford, FOM from TKSE and Zone Logic. Some of these have been widely

adopted by industry, e.g., IEC61131-3, EDDI and POLARIS, while some had been abandoned, such as SIMPLE and Zoner Logic. This section will provide a review of IEC61131-3 which is the most widely used programming standard in various industries including the automotive industry, EDDI which has been widely used in the European automotive industry, and FOM which is used by the representative machine builder ThyssenKrupp System Engineering (TKSE).

PLC Programming Standard IEC61131-3

Table 2-3 IEC61131-3 programming languages [35]

Language	Description	Example
Statement List	A low level textual language consisting of simple operation codes and analogous to ladder logic.	LD %IX20 AND Valve2 JMPNC Lab6 ST Tank_Level
Structured Text	A high-level programming language with syntax similar to Pascal and designed to make PLCs more accessible to programmers familiar with traditional programming languages.	IF TANK2 > 50 THEN Valve1 := ON; ELSE Valve1 := OFF; END_IF;
Function Block Diagram	A graphical language for depicting signal and data flow through Function Blocks. The blocks are reusable software elements.	
Ladder Diagram	The most traditional and commonly used graphic programming language.	
Sequential Function Charts (SFC)	A language made up of graphical elements called steps and transitions and derived from Petri-nets.	

IEC61131 is the International Electrotechnical Commissions (IEC) standard for PLCs. The details of IEC61131 can be found in the official website of IEC [28]. The IEC61131-3 is part 3 of this standard. It describes standard programming languages for PLCs in an attempt to provide an open, vendor-independent, consistent and structured approach to the development of control software. The standard enables algorithms to be written in any of the five languages defined within the scope of the standard as illustrated in Table 2-3 and packaged as

reusable software Function Blocks. Function blocks have well-defined interfaces with input and output parameters so that they can be readily interconnected.

The IEC61131-3 programming languages are currently the most widely used languages in many different industries. Most of the PLC programming tools on the market support, either fully or partially, the programming languages of IEC61131-3. In addition, the organisation PLCOpen, found in 1997, is dedicated to developing technical specifications around IEC61131-3. In order to eliminate the barriers arising from various data formats of different PLC vendors, PLCOpen has developed an independent standard data format PLCOpenXML based on IEC61131-3 and XML to achieve efficient data exchange of programs developed in languages of IEC61131-3.

EDDI

EDDI (Error Diagnostic Dynamic Indicator) has been the most widely used supplier independent programming structure in the Automotive Industry. EDDI was a European initiative led by Ford. Currently, in their European operations, Ford, Jaguar and General Motors apply various forms of EDDI or its successor STEPS however all contain the same basic principles. [36]

The EDDI is an application software structure in its purest form which does not require special hardware or software and can be applied on a variety of PLC and PC based software platforms. It can also be applied within the constraints of the IEC61131-3 specifications.

The EDDI contributes a number of innovative concepts and principles to control software development. It is a no-proprietary software structure for use with PLC systems. It provides a document system that can be specified by end users and taught to operators, maintenance staff and if necessary machine tool builders. Diagnostics can be fully integrated with the EDDI sequence control program.

Function Oriented Modularity (FOM)

Function Oriented Modularity (FOM) is a control software structure introduced by TKSE for programming assembly automation systems. The purpose of this new 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 selected set of function blocks can be instantiated and configured as required.

The overall software structure is shown in Figure 2-7. Unit and Process Step are the basic building blocks of the software structure. Unit is the smallest and lowest level enclosing working functionality and might represent mechanics, electrics, and control or their combination. Units appear as black boxes and can only be parameterised. A unit not only controls the behaviour of a mechanism but it has integrated fault diagnostic and generates the required HMI screens for manual mode control. Whereas, Process Step is a function for the sequencing 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. It reads the RFID data-tag at start of operation and writes the status back to the tag when the operation completes. A Process Step consists of a number of sub tasks (such as open clamp, close clamp) known as ‘Process Single Steps’. All subtasks are coordinated via a process coordinator and designed in a combination of FBs and LD.

FOM has been used in a number of automotive companies, such as at Ford and Volvo. From the program development point of view, its implementation is fairly easy for ThyssenKrupp engineers. However, FOM has not received much recognition from end-user engineers. Due to the black box nature of the code, FOM programs are regarded as very complicated and difficult to understand as compared to alternative programming structures based on LDs and SFCs.[12]

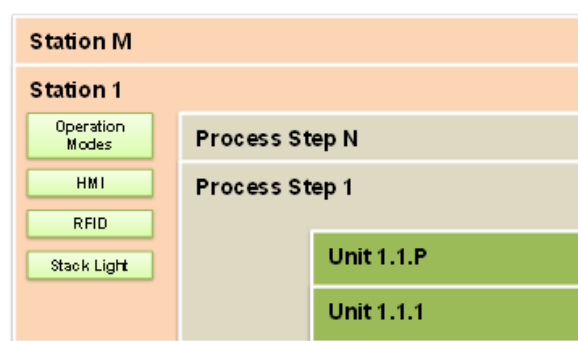


Figure 2-7 FOM Structure [12]

2.2.2. Emerging Approaches to Efficient Development of Control Logic

Existing emerging approaches to facilitating the development of control logic or control software include formal methods, automatic code generation and virtual commissioning. The

following sub-sections provide a review of the former two types of approaches while the virtual commissioning is reviewed separately in section 2.3.

2.2.2.1. Formal Methods

Formal methods refers to mathematical reasoning about system model properties [37]. The complexity of programming and verifying large systems has resulted in interest in the possibility of using formal modelling and analysis techniques [38, 39]. The key benefit of formal methods in controls engineering is to authenticate the control code by performing mathematical analysis to check stability, reachability and deadlock. Typically, these methods comprise the formalisation of informal specifications followed by automatic synthesis and implementation of the PLC code [40]. The most common languages used for formal modelling are Petri Net (PN) and finite state automata [40].

The use of Petri Nets (PN) has gained many interests as a potential tool for design and verification of PLC programs. A Petri Net is essentially a graphical method of defining discrete event systems, consisting of places, tokens, transition, and arcs. This section briefly names a few examples of existing researches. Uzam et al. proposed the use of Petri Net to synthesise a supervisor [41]. This supervisor can be converted to Ladder Diagram via a token passing logic controller. Feldmann and Colombo developed an approach to validating control logic and generating PLC code according to the standard IEC1131 in [42]. They also utilised Petri Net to develop high level model-based monitoring systems for monitoring the operations and behaviours of flexible production systems[43]. Frey et al. presented researches using PNs to model controller using graphical description[44-46].

The views of research community on the use of PNs seem to be divided. Some researchers, such as Lee [47], referred to PN as a flexible method which is easier to use than ladder logic. This statement is based on comparing the number of logical elements or conditions in LD and PN programs. However, Ljungkrantz [48] stated that the number of logical conditions and elements does not represent the work required to configure a control system. Hajarnavis et al. stated that such a comparison of methods is “questionable and not fair” [49]. Practitioners in industry have shown very little enthusiasm for the direct use of PN [49].

Finite state automata have also been considered by many researchers to model and analyse manufacturing systems. However, finite state automata (as well as Petri Nets) suffer from state explosion when reachability analysis is conducted for a complex system with too many

reachable states. To avoid state explosion, Endsley et al. used an extension to finite automata called modular finite state machines to generate a verifiable controller [50]. 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 et al. presented a number of research on control logic modelling using formalism using timed-MPSG (Message-based Part State Graph) [38, 51, 52], an extended version of finite state automata. The model is then converted into a textual specification for formal verification using a model checker tool. The formal model of the system can also be interfaced with a 3D model via 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 with the I/O addresses and the executable PLC code is then generated for Siemens Step7.

The use of formal methods has received great attention from academia but has received very little attention from industry. The formal methods are normally performed using languages that are very mathematical. Therefore the modelling complexity and non-familiarity of the modelling languages to control engineers make them unattractive to industry. As a consequence these methods are still confined in research laboratories [53]. According to Lucas the benefit of these new methods over the current practice have not been well demonstrated [54]. According to Thapa et al. the PN approach does not fit within the current engineering practices and is not well known to control engineers and technicians [38]. Logic design using PN is quite different and complex compared to existing approaches used in industry. For example, enabling/firing of transitions can be a quite cumbersome task. According to Danielsson et al. [55] formal methods require users to learn new skills (such as new modelling languages and computer programming), which are complex compared to the conventional PLC programming methods. Some researchers have developed tools for formalisation of existing IEC 61131 PLC code. But this still requires the user to learn new languages and tools for the specification development [56].

2.2.2.2. Automatic Code Generation

The current practice of logic design relies excessively on copying, pasting and adaptation of functionality from one project to another project, which requires highly experienced programmers. Manually coded programs are highly time consuming, vulnerable to errors and inconsistencies. Any modification in the program is quite cumbersome and often results in

discrepancies. Another problem is the redundancy of the previous work at the time of logic design and verification. For example, control logic is specified within the virtual engineering tools to simulate the machine behaviour. However, due to the lack of integration between virtual engineering and PLC programming tools, the same control logic is then re-implemented manually in the PLC programming tools. In order to cope with these limitations, new planning methodologies are necessary to enable collaborative and integrated engineering of automation systems [38]. Automating the process of generating PLC control software based on existing process data or simulated control logic is another way of facilitating control software development.

In the past decade, the concept of automatic generation of control logic emerged and received attention of both academia and industry. As virtual resource models encompass almost all information about the control aspects of a manufacturing cell; therefore, the machine configuration data required to generate control code can be extracted from these virtual models [57]. This will not only avoid manual programming but will also ensure consistency in the structure and quality of the control programs [58]. As the control code for the HMI and PLC are generated from the same model, thus discrepancies between HMI and PLC can be avoided. Automatic generation of control code is potentially the most efficient and effective way to significantly compress the development and commissioning time of control programs [51].

In this context, a number of automatic code generation methodologies have been proposed by academic researchers as well as some vendors of commercial engineering tools.

A. Academic Methodologies

Estevez [59] described an approach to generating platform-specific PLC software by transforming the existing PLC program developed for another platform. Steinegger [60] presented a general paradigm of generating a PLC program by integrating related data existing in related engineering tools. However, the proposed methods are conceptual and no practical solution has so far been presented yet. Approaches to generate PLC code from control logic described in different graphical forms have been proposed and implemented. Bevan [61] presented an approach to generating PLC code for components of transitive systems from control logic described in Petri Net form. Thapa [62] also proposed and implemented the automatic code generation method based on virtual models described using

t-MPSG. Bergert [63] presented a framework for the automatic generation of PLC programs from digital process information extracted from work cells modelled in DELMIA Process Engineer. In this approach, the cell specific process plan developed using a pert chart was converted into SFC. The SFC is then connected to manually coded resource specific PLC function blocks which describe the behaviour of the manufacturing resources and contain all I/O signals from the resource. Some other automatic code generation approaches proposed before 2007 were reviewed by Bergert in [63].

These approaches from academic researchers are mainly focused on generating the source code of a specific function block or a specific program rather than the whole PLC control software. In reality, this only represents a small percentage of the machine control software. Furthermore, most of these tools do not support integration with high-level engineering tools.

B. Siemens - SIMATIC Automation Designer

SIMATIC Automation Designer is built to enable the reuse of information from planning phase to develop control software. It allows integrated engineering of mechanical, electrical, and control aspects of a component and enables modular configuration of a system [64]. Automation Designer includes tools for the automatic generation of PLC code for Siemens Step7 and HMI screens for 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 and contains information about the object including hardware information, PLC code, and HMI screens. The templates for the generation of the PLC program can either be written inside Automation Designer or be imported from the S7 library

A tree structure of the required templates is then created in Automation Designer to describe the hierarchical structure of a to-be system. The simulation model of the system output from Process Simulate is then imported into the Automation Designer. Based on the tree structure and the simulation model, the configuration of the system can be achieved in Automation Designer and the PLC and HMI code can be generated. The generated PLC code mainly consists of the required function blocks, hardware configurations, HMIs and the connections between HMIs and PLCs. However, it does not include control logic for coordinating sequence and therefore manual programming is still needed.

A number of research activities which adopt Automation Designer to achieve automatic code generation have been found in the literature. Falkman [58] conducted a thorough review

of the potential of Automation Designer. The case study presented involves the use of a simulation study and templates to automatically generate the PLC code and HMI screens to the Volvo Car Corporation's standard. Andersson and Helander report research activities conducted in Chalmers University of Technology of Sweden to automatically generate the PLC code and HMI screens for three stations in the Volvo Car Corporation using Automation Designer [65].

C. Dassault Systems - Delmia Automation

Delmia Automation extends the suite of Dassault System's end-to-end Process Lifecycle Management (PLM) solution by providing a tool dedicated to a) the implementation of production system 3D models, and b) the editing, testing and debugging of system control logic against the 3D model. It provides a software module for the manual programming of the control logic in the standard IEC 61131 languages, which can be validated against a 3D simulation model. This validated code can be automatically translated into platform-specific code for PLCs from different vendors such as Schneider, Siemens and Omron.

D. Allen Bradley - Enterprise Controls

Enterprise Controls and RS TestStand, from Rockwell Automation, were designed to improve the efficiency of the logic development process in 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 is written twice, i.e. once for virtual model and then for code generation.

TestStand allows simulating the behaviour on 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. The device template includes integrated HMI generation, and error handling and diagnostic capabilities. Once tested, these generic templates are then stored in a library for use. These templates are then automatically translated into ladder logic code.

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 real I/Os or virtual model of RS TestStand. Once the application definition completes then the control code is automatically generated for Allen-

Bradley HMI and PLC and can be connected to virtual model in the RS TestStand via OLE for Process Control (OPC) for virtual verification.

E. Siemens – eM-PLC

eM-Engineering and eM-PLC are applications from Siemens Tecnomatix that enable the generation of PLC logic code and provide a simulation environment for control verification. Of these two applications, eM-Engineer is used to develop the 3D based visualisation virtual model of a machine and eM-PLC enables users to edit control information to the virtual model and generate PLC control code for Siemens Step 7. The generated PLC code can be further validated against the virtual models through connecting the Step 7 program on the PLC with eM-Engineer via an OPC link.

3D visualisation virtual models can be created in eM-Engineer by importing existing CAD data and then adding a kinematic specification. The input information required for automatic code generation needs to be created in eM-Engineer and eM-PLC. The control sequence is added to the virtual models in the form of Gantt Chart and validated first in eM-Engineer. In eM-PLC, the control actions, namely the switching of a signal bit, and supervision conditions and interlocks, namely boolean variables in logic equations, are then added. Based on the control actions, conditions, interlocks and the sequences imported from eM-Engineer, eM-PLC converts each sequence to a S7 Graph (similar to SFC) function block. Each operation corresponds to a step in an SFC function block.

It can be seen that the automatic code generation of eM-PLC is actually the conversion of the entered specifications into Step7 graph. The preparation of the specifications requires users to consider detailed control behaviours and logic. This is as difficult and tedious as usual programming. Moreover, the control code generated by eM-PLC might not be able to control the real machine without manual modification in Step 7 environment. In some cases, unnecessary control states and signals, which are not used in physical machine but required in controlling virtual models, have to be removed.

F. Summary

A summary of existing approaches to automatic code generation is presented in Table 2-4. These approaches are mainly assessed from the aspects of required input, namely how the logic is specified as the input of the generation approach, and the output, namely what logic

code is generated. It can be seen that these existing approaches both have their respective merits and limitations.

The academic approaches normally model the logic use mathematical modelling languages therefore normally no complex engineering tools are required. However, the mathematical languages are difficult to be understood by control engineers. Moreover, these approaches can only generate pieces of control code instead of the complete desired control software therefore manual modifications are still required.

The commercial vendors, on the other hand, provide more powerful solutions which can generate applicable code in the ways that are more acceptable for control engineers. However, the limitations of these solutions lie in the way of modelling input logic. The required input logic is normally created in the way which is similar to that of manual programming therefore still complex and time-consuming. Also, for most of the vendors, the automatic code generation solutions are vendor-specific, which means only the control code for their own PLCs can be generated.

Table 2-4 Summary of existing approaches to automatic code generation

Approach/Tool	Input Logic	Output	Advantages	Disadvantages
Academic Approaches	<ul style="list-style-type: none"> Mainly mathematical models 	<ul style="list-style-type: none"> Function blocks or programs for specific control 	<ul style="list-style-type: none"> Requires only simple modeling tools 	<ul style="list-style-type: none"> Very mathematical Only partial generation of code
Siemens Automation Designer	<ul style="list-style-type: none"> PLC control templates HMI templates Simulation models I/Os 	<ul style="list-style-type: none"> PLC code without sequence control HMI with connections to PLC Configurations 	<ul style="list-style-type: none"> Generate structure of complete Step 7 projects even including HMI and hardware configuration 	<ul style="list-style-type: none"> Manual programming for sequence control still required Only for Siemens PLCs
Dassault – Delmia Automation	<ul style="list-style-type: none"> Complete PLC code in IEC61131-3 languages 	<ul style="list-style-type: none"> PLC code for different PLCs 	<ul style="list-style-type: none"> Generate complete PLC code PLC-independent. 	<ul style="list-style-type: none"> Requires manual programming to complete PLC code.
Allan Bradley – Enterprise Controls	<ul style="list-style-type: none"> Sequence I/Os Machine-specific code Templates with diagnostic 	<ul style="list-style-type: none"> Ladder logic code for Allen-Bradley PLC 	<ul style="list-style-type: none"> Generate both PLC and HMI 	<ul style="list-style-type: none"> Can only generate SFC code Only for AB PLCs.
Siemens – eM-PLC	<ul style="list-style-type: none"> Sequence Conditions Interlocks I/Os 	<ul style="list-style-type: none"> SFC function blocks for Step 7 	<ul style="list-style-type: none"> Generated code can be validated virtually first. 	<ul style="list-style-type: none"> Only generate SFC FBs May need manual change Only for Siemens PLCs.

2.3. Virtual Commissioning

After a manufacturing system is physically built, the ramp-up phase starts and it ends when full target quality at a specified cost and output rate are achieved. The ramp-up can be divided into the commissioning phase and the following run-up phase. The commissioning phase ends with up the production of the first product that meets the specification and the acceptance of the customer while the run-up phase transfers the operational production system into stable production conditions in compliance with target cost, demanded quality and output [7]. Issues arise during the actual commissioning phase when unseen design errors translate into non-working machine, unmatched functionalities and sometimes catastrophic failures (e.g. collisions between actuators/products). Such unforeseen events are due to the fact that prior to commissioning engineers do not have any means to check the consistency of mechanical and process with control engineers together. To relieve this situation, virtual engineering technologies, especially virtual commissioning, can be highly effective for reducing ramp-up cost and time by testing different ‘what-if’ scenarios in a virtual environment prior to the physical system building.

2.3.1. Overview

2.3.1.1. Concept and Methods

Virtual Commissioning technology has been researched for more than a decade however no explicit and formal definitions have been found in the literature. Some literature [66-68] identifies that the essential requirements of virtual commissioning to include ‘virtual plant’, ‘real PLC’ and ‘real control code’ while some other literature [7] asserts that virtual commissioning can also be performed using a simulated PLC and simulated control code. The author defines the Virtual Commissioning in the context of this research as:

“To detect and correct the errors generated during planning, design and control engineering prior to the physical assembly of the real machine through running a 3D virtual model of a desired machine which can be driven by either real PLC code running on PLC controllers (either physical or simulated) or simulated control logic.”

Obviously, virtual commissioning is a type of virtual engineering technology. Currently, there are also other different types of virtual engineering techniques employed which are summarised by Kuhn in [69]. For these different types of approaches with different purposes,

the level of details and the required resources are different. Figure 2-8 summarises and demonstrates the differences between some other techniques employed in the current industrial practices.

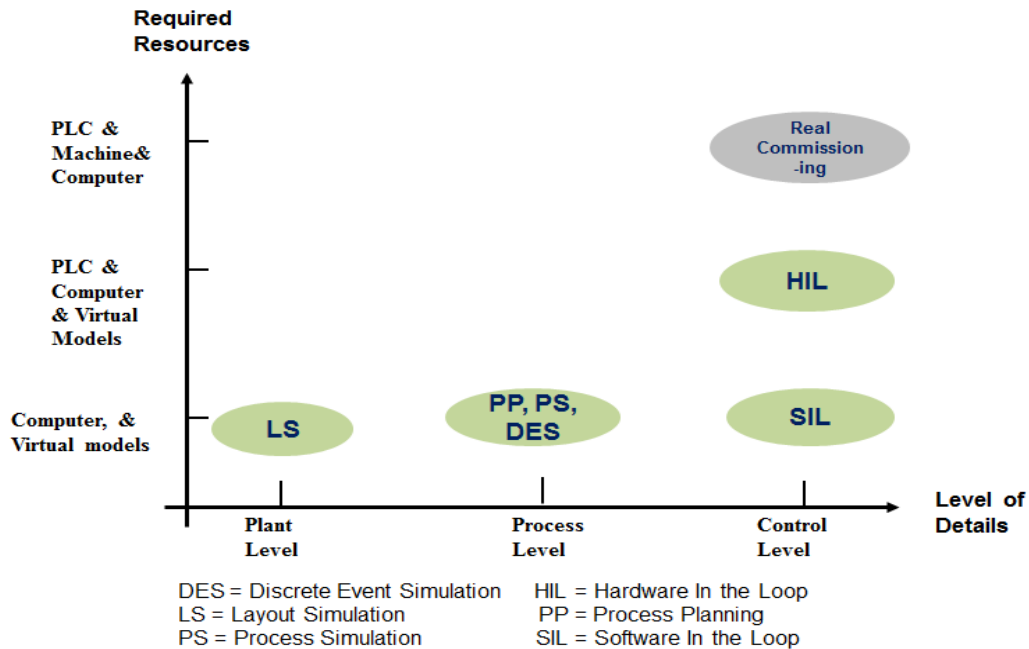


Figure 2-8 Required details and resources of different virtual engineering technologies

Some similarities and differences between the involved techniques can be seen from Figure 2-8:

- For each type of simulation, virtual models are required.
- Virtual commissioning requires specification of control level functionality while other simulations only need higher level details. For example, the layout simulation only needs static virtual plants and details of plant level while process simulate and process planning requires a dynamic virtual models and details at process level.
- For different types of VC approaches, which will be introduced in the following two sections, the required resources are different. HIL requires real PLCs while SIL does not.

The main benefit of VC is the achievement of a shorter production ramp-up time which is an important factor for a product’s economic success. According to an analysis of critical points of the ramp-up process of a final assembly, Eversheim et al determine control system malfunctions to be a major source of time delay[70]. The particular reasons mostly lie within untested and newly developed control systems, new communication technology and the lack

of adequate monitoring and diagnosis system. However, according to Glas, control software engineering is responsible for over 50% of the functionality of highly automated production equipment [71]. An investigation for the German Association of Machine Tool Builders (VDW) showed that the commissioning phase of a production system accounts for up to a 25% of the total project cycle time [72]. A remarkable 90% of the commissioning time is used for delays and activities related to electric and control devices. Again, 70% of this time delay was associated with errors in control software. In other words, the correction of defective control software consumes up to 60% of commissioning time or 15% of time-to-delivery.

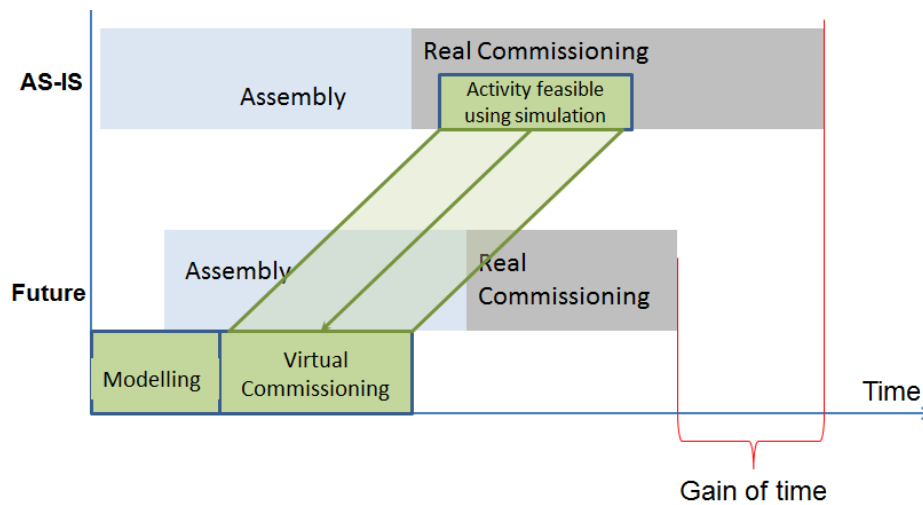


Figure 2-9 Time benefit of VC [7]

2.3.1.2. Existing VC Approaches

The method or workflow typically implemented through the use of VC solution is to build a virtual equivalent of the physical system that comprises 3D geometrical (shape) and kinematic (possible motion) data. The real time control data is then implemented and mapped to the possible 3D model motion in order to merge 3D and control model behaviours. At this stage, most of the system can be tested and both mechanical and control engineers use an engineering tool that allows them to coordinate design process. Additional digital design can be conducted to model product flow, safety features, human operation ergonomics etc.

Current approaches to building a prototype for virtual commissioning can be classified as either Software-In-the-Loop (SIL) or Hardware-In-the-Loop (HIL) simulation, as shown in Figure 2-10. Under the Software in the Loop (SIL) method, the control programs for the resource controllers (PLC or other) are downloaded to virtual controllers and TCP/IP connection is established between the mechatronic object and the software-emulating

controllers. It is obvious that the main advantage of the SIL approach is that no hardware is required during the designing and validation of control software, while standard desktop PCs can be used for its implementation. On the other hand, often simulated packages do not support all the versions of a specific PLC family therefore the exact replication of the physical PLC might not be achieved [7]. The second method, known as hardware in the loop (HIL), involves the simulation of production peripheral equipment in real time, connected to the real control hardware via fieldbus protocol. Under this setup, commissioning and testing of complex control and automation scenarios, under laboratory conditions, can be carried out for different plant levels (field, line, or plant) [73].

Relevant literature [66-68] suggests that most of the researchers agreed that using real control code is one of the key requirements of VC. In spite of using different control systems, both SIL and HIL validate the control logic based on real control code. SIL or HIL approach has been adopted by most researchers to perform VC.

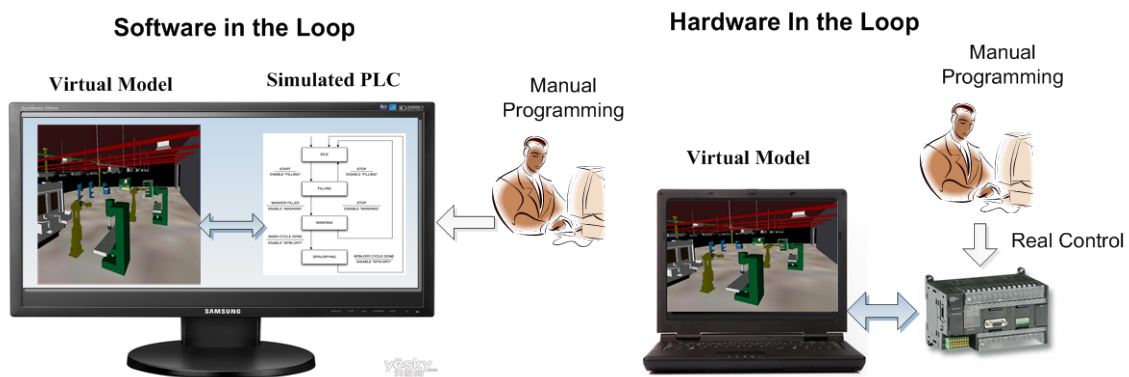


Figure 2-10 Software in the Loop (SIL) and Hardware in the Loop (HIL)

2.3.2. Hardware/Software-In-the-Loop (HIL/SIL)

The involvement of real control code in SIL or HIL approaches ensures the control software can be tested under more realistic conditions without the necessity to make changes to the software afterwards. Moreover, this enables a seamless transition of control logic from virtual commissioning to physical commissioning, which eliminates the potential of bringing in new errors through later manual work. In both approaches, production peripheral equipment is simulated in real time and connected to the control systems. The only difference between these two approaches is that HIL involves real control hardware while in SIL simulated controllers are used. Therefore, the required engineering tools for SIL and HIL

might be different. Apart from this, from the users' point of view, there is no significant difference between SIL and HIL in terms of the related data modelling process.

2.3.2.1. Data Modelling

For separate verification of mechanical design, a 3D simulation of the expected and specified mechanical behaviour is sufficient. For separate validation of the control programs, a simulation reflecting the specified behaviour of the manufacturing system mechanics at I/O level is required. However, the principle of VC is to detect errors in the control software through observing the simulated mechanical behaviour of the virtual systems, If the impact of control programs on the 3D mechanical behaviour of the manufacturing system is to be tested in detail in an integrated manner, the modelling and simulation of the complete functional chain from control programs through sensors, actuators and drives onto the mechanical movements, is necessary. This includes both, simulation of mechanical behaviour and of the control programs. [67]

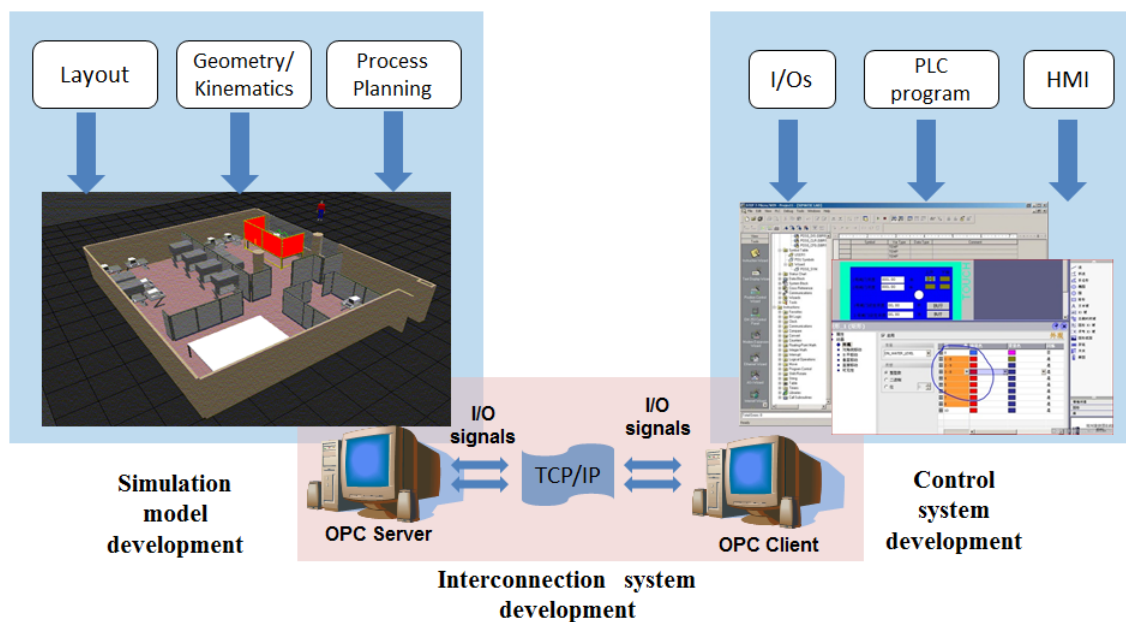


Figure 2-11 System architecture for HIL/SIL

As it has been identified, a HIL/SIL VC system is normally built by integrating three subsystems, as shown in Figure 2-11, which include (a) a simulation model composed of mechatronic units including actuators, sensors, and behavioural description of a system related functional model, (b) a machine control system, including its input and output signals, and (c) connections between the simulated sensors/actuators and the control [74]. This

section is dedicated to describing the activities involved in building such a HIL/SIL VC system.

a. Simulation Model Development

This step refers to the development of the required 3D visualisation model upon which the control software will be validated. Existing data from related engineering tools can be used for the detailing of the resources, followed by their conversion into the needed data formats. The simulation model is developed in a 3D environment through the following steps:

(1) Create 3D simulation models of mechatronic units of the machine to be commissioned. The simulation models mainly contain the respective properties of geometry, kinematic behaviour, and electronic connection interfaces.

(2) Assemble the virtual mechatronic units into the desired machine and specify the layout of the production cell, involving exact placing of resources and all relevant equipment.

(3) Define the material flow in the virtual machine, involving sequence of operations and interdependencies between the production processes. The behaviour model of the production system is completed at this step by defining all simulation activities (such as operator paths) and realising any software interlocks in the control simulation software.

(4) Assign the I/O signals to corresponding mechatronic units. I/O signals list used by the PLC or soft PLC, namely emulated PLC, can be imported from existing files. According the material flow defined, the I/O signals are then assigned to the corresponding sensors, actuators and any other resource entities.

b. Control System Development

In the HIL/SIL approach, the control software needs to be developed separately using the approaches which were described in section 2.2.1. The developed control software can be executed on either the actual control systems (for HIL) or the emulation software (for SIL) to control the simulation model of the machine. In the case of SIL, where no actual hardware is available for the validation procedure, the human machine interfaces (HMI) also needs to be simulated in a virtual environment. Additional capabilities, like safety systems, can be programmed in the control system in order for the operation of the virtual cell to be validated sufficiently.

c. Interconnection between Virtual Models and Real Control Systems

The interconnection is mainly implemented based on an IT infrastructure, such as software drivers and communication protocols for the networking between the control system and the simulation model. The communication protocols are normally TCP/IP. Currently the required IT infrastructure is normally a standard third-party application such as the Object Linking and Embedding (OLE) for Process Control (OPC). Before these standard third-party applications released, the required IT infrastructure was also part of the VC environment development work [75].

2.3.2.2. Related Researches and Engineering Tools

As identified in the last section, the most important factors of HIL/SIL are visualisation tools, simulation tools, control technology as well as I/O connections. In early applications of VC approach, many efforts had been made in connecting the controllers to the discrete event simulation tools since there was no standard application available for data exchange between different control devices. Schludermann et al. presented a typical example of performing HIL VC based on user-defined I/O connection middleware [75]. In order to establish a connection between the visualisation model and the PLC, in this research, a communication protocol and the related I/O device driver were developed first. This situation has been greatly relieved since the OPC were developed as a standard specification. OPC was developed in 1996 by an industrial automation industry task force, and now are widely accepted by the industry as a standard for device communication. OPC is a data integration middleware in the industrial control field, which defines an industry standard for exchanging data between field devices, control systems and other applications [76]. It now has been widely supported by both device vendors and engineering tool vendors.

Over the last years, a plethora of commercial packages that can be used for the implementation of a VC project, are available on the market. Delmia from Dassault Systems allows the virtual prototyping of PLC control systems for cells, machines and production lines which uses OPC communication for the coupling of the real control system with the simulated resource. Similarly, the Process Simulate commissioning package from Tecnomatix, enables users to simulate real PLC code with the actual hardware by using OPC and the actual robot programs, thus enabling the most realistic virtual commissioning environment. Generally speaking, the current Virtual Engineering tool market is mainly

driven by Dassault Systems and Unigraphics/Siemens. Apart from these two providers, many other VC tools are also widely used by the researchers from both industrial and academia. A typical example of HIL is the HIL environment realised in VIR-ENG project [32], which adopted IGRIP and Quest for virtual model prototyping and ISaGRAF PRO IEC1131-3 for PLC code programming. Researchers from Daimler are working on building virtual commissioning environment for modular automation systems using INVISION for 3D modelling and WinMOD for modelling control behaviours [77]. WindMOD and INVISION are also adopted by Markris et al for the VC of an assembly cell with cooperating robots [68].

Apart from the commercial VC tools available in the market, a few VC environments have been proposed and implemented by academic researchers in order to implement innovative approaches to data modelling. For example, Qin et al. described a 3D simulation environment with an embedded programmable logic controllers (PLC) for the development and test of a cell control [78].

2.3.3. CCE - A VC Engineering Tool for VCOM

The Core Component Editor (CCE) toolset is currently being developed by the ASG researchers based on the component-based approach. In the Virtual Commissioning using Component (VCOM) framework, it is used to perform virtual prototyping, simulation and validation of both process and control logic. This section briefly describes its functions and its component-based architecture (more details can be found in [11]).

A. Function description

The CCE toolset provides a lightweight, non-proprietary package to support production system lifecycle management enabling a) the design, visualisation, testing and debugging of control logic and b) a 3D modelling environment for building machine models against which the control logic can be tested. The CCE tool set uses standard VRML formats for 3D modelling and generic State-Transition Diagrams to support control logic editing/visualisation.

The CCE Toolset provides User Interface (UI) and functions dedicated to the design of automation systems' control layout as well as lightweight 3D virtual environment that can be linked to the real-time control simulation engine in order to visualise, test, debug and validate the system behaviour in a virtual form. Logic editing and virtual modelling environment are

integrated around a common data structure referred to as common model architecture, which describes a hierarchical system as a composition of “component”. A set of system representations (state transition, sequence interlock, timing/Gantt chart diagrams) provides 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.

B. Component-based architecture

The whole CCE tools development is driven by the concept of “component” and “component-based” system architecture [4, 8] which seek to enable re-usability and re-configurability of basic modelling constructs. The concept of a “Component”, which is defined as a re-usable, reconfigurable data block providing the data integration mechanisms for control, 3D modelling, kinematics and other data types describing a particular resource, is central to the CCE tool development.

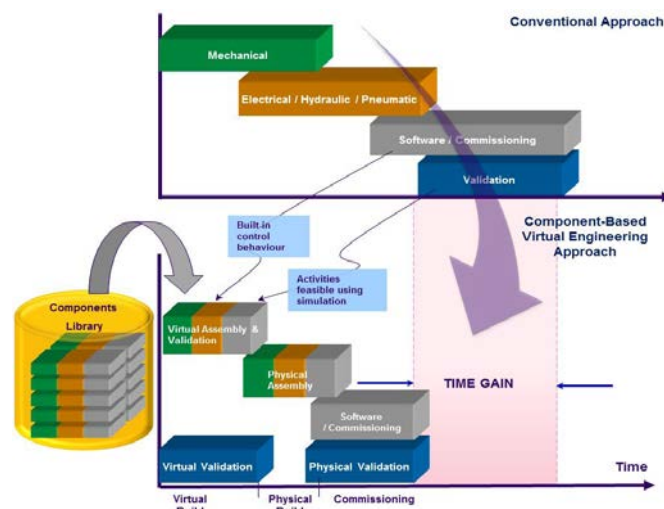


Figure 2-12 theoretical basis of CCE - the Component-Based Approach

Using the CCE tool, the overall control design and simulation model editing workflow is broken down into two main tasks which are the “component (library) editing” and “system editing”, as outlined in Figure 2-12. The component editing task consists of building up a library of reusable system components which encapsulate both modelling (3D geometry, kinematics data) and control behaviour (as a set of state, possible state transitions and associated conditions), as shown in Figure 2-13.

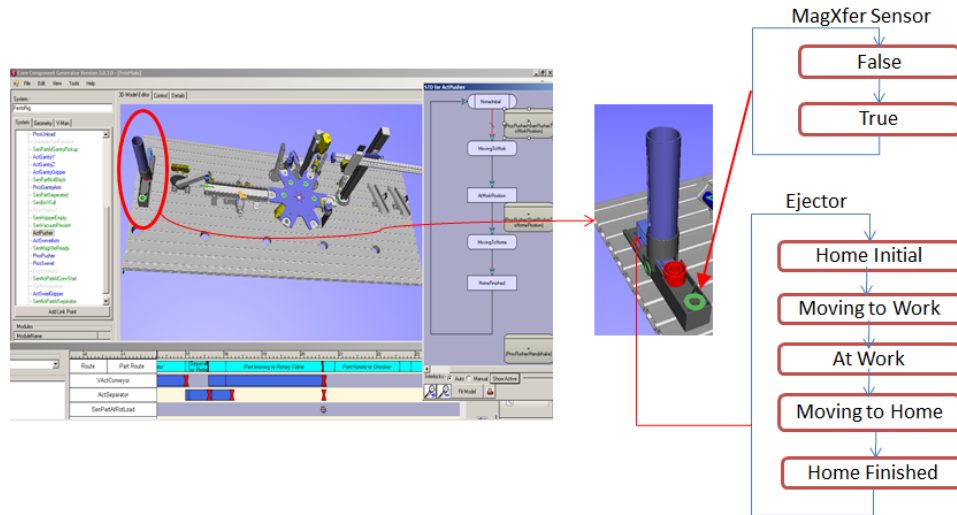


Figure 2-13 Component behaviour modelling in CCE

The complete system model editing task consists in assembling components together by 1) defining a spatial layout using a unique assembly point and 2) a control layout by defining sequence logic, as illustrated in Figure 2-14.

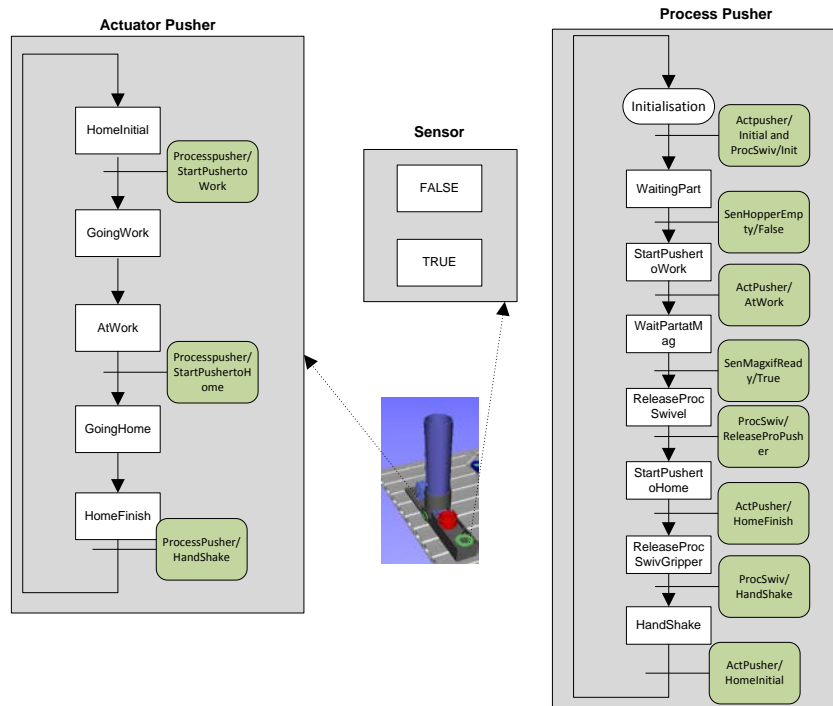


Figure 2-14 Building system control logic by interlocking behaviours of components

2.4. Openness of VC

In the context of this research, openness refers to the ability of engineering tools to interoperate or exchange data with other related tools or systems.

The current approach to the development of increasingly complex automation systems is to break down the overall complexity a system to different sub tasks. Accordingly different structures for each sub task need to be developed [79]. Different technologies and engineering tools are developed to solve subtask specific problems. Typically, during the process of complex automation system engineering, a range of engineering disciplines need to cooperate [80], which means the interoperability of the different engineering tools in the process is unavoidable. Before commissioning, different categories of data are integrated to build the machine, as shown in Figure 2-15.

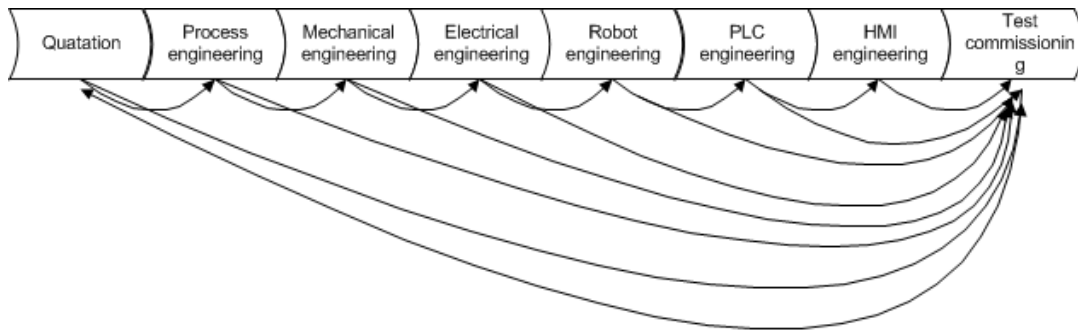


Figure 2-15 Commissioning requires data from different phases

As the virtual equivalency of the to-be physical system, the virtual model used in virtual commissioning also needs to be built by integrating data of different disciplines. Currently, the virtual model can be seen as a simplified equivalency of the physical system. Normally only the data required to create the 3D visualisation, kinematic movement, process logic and control logic are involved. The data flow of virtual commissioning is shown in Figure 2-16.

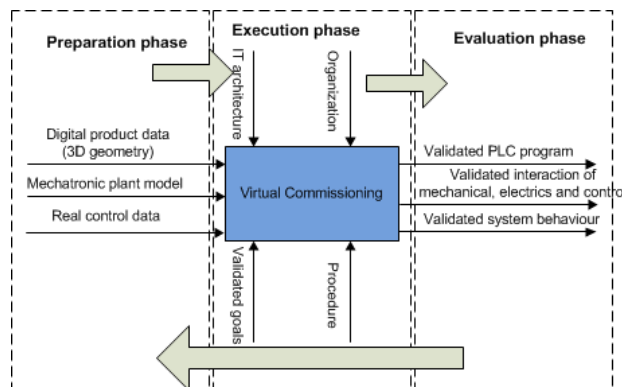


Figure 2-16 Data flow of Virtual Commissioning

It can be observed that virtual commissioning requires transfer of data between different engineering tools, due to the necessary hybrid data inputs in the form of digital product and

resource as well as control data [77]. Currently, data exchange between virtual commissioning toolsets and discipline-specific engineering tools has been achieved based on available neutral data formats, such as STEP, JT for the CAD exchange, VRML for 3D data exchange, PLCOpenXML for the control logic exchange. However, the data exchange of virtual models between engineering tools that need to reuse the validated virtual models has drawn little attention. The following section will now discuss the need for openness between engineering tools and how this might best be achieved.

2.4.1. Why openness needed - Potentials of reusing virtual models

In order to perform Virtual commissioning, it is essential to first get the relevant data such as CAD and control logic to build the virtual models. In the current practice, these data can be efficiently exchanged via corresponding available open standards. However, the openness required to exchange the validated virtual models as integrated objects between engineering tools has been neglected.

VC is part of a larger concept which is referred as Virtual Engineering, which is also part of another larger concept – Digital Factory [69]. Openness and tool integration are key requirements of implementing the concept of the Digital Factory. The Digital Factory is defined as “*The generic term for a comprehensive network of digital models, methods, and tools – including simulation and 3D/Virtual Reality visualization – which are integrated by a continuous data management system*”[69]. The virtual model validated by VC can be seen as the virtual equivalency of the to-be realistic system. Although it does not include all the data of the realistic system currently, the information it contains can still be used to make further contributions so that the cost-value ratio of the virtual model can be further reduced. Unfortunately, research in this field is rather rare. Even though, in the later phases of the production system lifecycle, there are many scenarios in which the HIL virtual models can be reused. Also, virtual models can also be reused by different VC engineering tools during the subsequent system redesign or reconfiguration phase.

2.4.1.1. Data reuse during system operation phase

Related literature suggests that, after VC, the virtual models can be potentially reused for different operative purposes.

To build the digital factory, a virtual model can be involved in a feedback loop in order to update general data, model structure and model parameters with the actual situation from the factory. In this feedback loop, the virtual model used as a reference model plays the same role as it does in the virtual commissioning phase, as shown in Figure 2-17. Changes to the control logic can be pre-tested against the virtual models. On the other hand, it is also used for the purpose of real-time monitoring. The runtime status of the real machine can be collected and duplicated by the virtual models and then feedback to the control design for making further potential improvement.

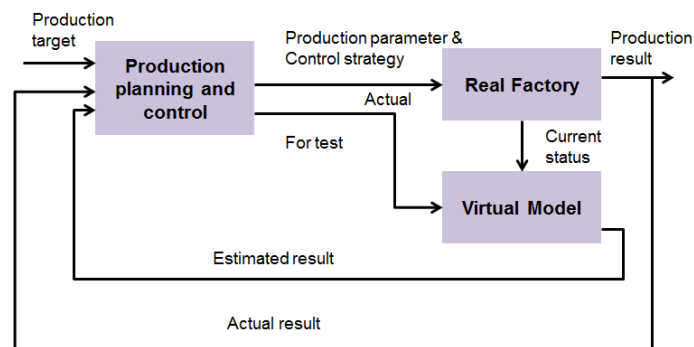


Figure 2-17 Virtual models used as reference models during operative control [66]

During the system operation phase, the validated HIL models can also be used for 3D based diagnosis or monitoring in a number of scenarios [81-86], as summarised and illustrated in Figure 2-18. Using HIL simulation models for diagnosis is based on online comparison between the values of the measured signals of the plant and those of the simulation. Concerning diagnosis two approaches can be applied [87]. In the first approach, the required behaviours of the production plant were emulated by the HIL simulation. In case of deviations between simulation and real plant, non-intended behaviours of the plant can be detected easily and an appropriate reaction can be executed. In the second approach, different incorrect behaviours of the plant were emulated and compared to the real plant. If the measured behaviour of the plant matches with one of these models, a specific incorrect behaviour of the real plant can be detected [87].

The HIL models are particularly useful for remote diagnosis or remote monitoring. Kain et al. present a general discussion of reusing HIL for further diagnosis or control system optimisation [88], however, no further details and methodologies are proposed. The typical scenarios found in the relevant literature which are capable of reusing HIL models include the press line and diagnosis system presented by Ng et al. in [85], the 3D visualisation-based

manufacturing and facility control system presented by Alabdulkarim et al. in [86], and the real-time event-based 3D-monitoring of material flow systems described by Feldhorst et al. in [89].

In addition to the scenarios mentioned above, there are other potential scenarios to the adoption of digital factory concept. As summarised by Leitao et al. in [24], the VC models can be reused for operator training and maintenance, and the simulations can be repeated as many times as necessary to aid the correct understanding and tuning of the system control.

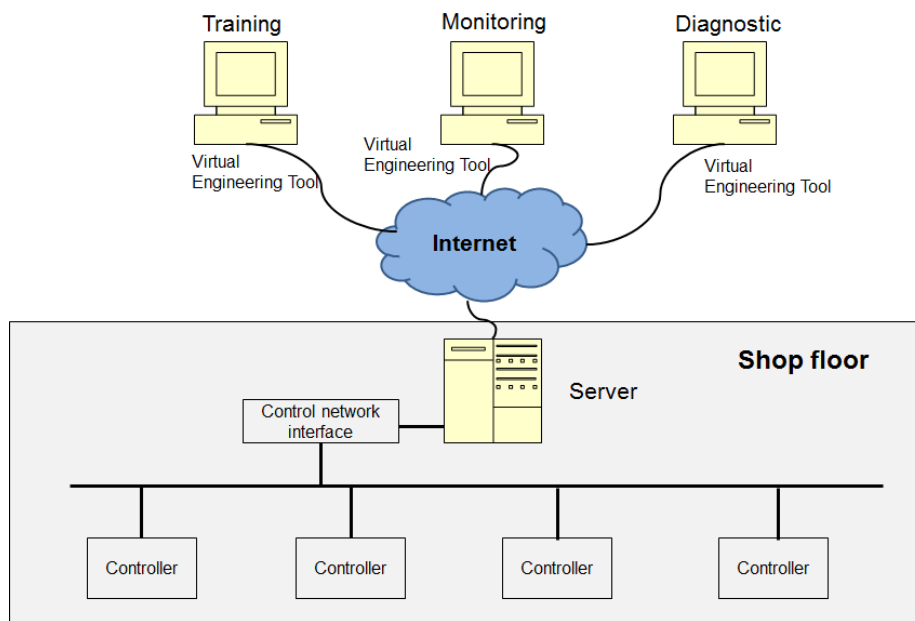


Figure 2-18 Scenarios of reusing HIL models during system operation phase

2.4.1.2. Data reuse during system design phase

Another scenario in which the reuse of VC models is necessary is to duplicate the same simulation models in different engineering tools. The reusability is required even between the engineering tools from the same discipline in some situations for the reusability of existing data. One example for this situation exists between the Original Equipment Manufacturing (OEM) and its suppliers[90], as illustrated in Figure 2-19. In a specific supply chain, each OEM usually requires their suppliers to use the appointed engineering tools to avoid system heterogeneity. However, the suppliers usually work for more than one OEM, which means they have to use more than one engineering tools. In this situation, to avoid redundant works which is a waste of time, finance and human cost, the seamless interoperability for data exchange between the systems one supplier adopt is of great importance. Furthermore, to date,

the design of VC models has certainly required a high level of expertise and considerable effort, which makes virtual commissioning unattractive, especially for small and medium-sized enterprises [67]. This situation can be relieved if the models can be efficiently reused between different enterprises.

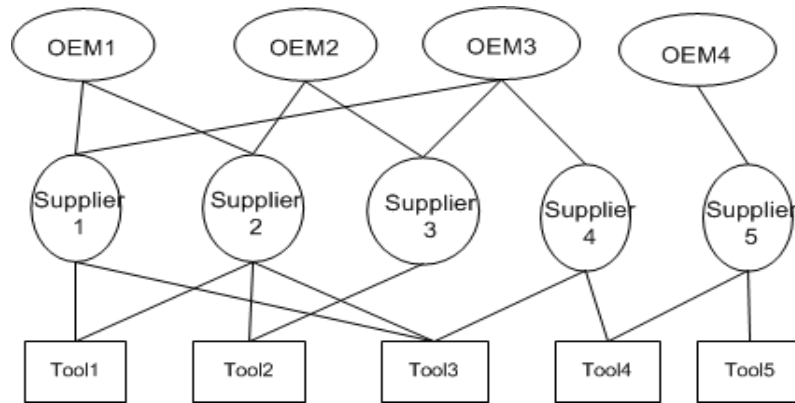


Figure 2-19 need for data reuse driven by the need of OEMs

2.4.2. How to achieve openness – Approaches to data exchange

Achieving openness, in the context of this research, refers to enabling an engineering tool to interact with other related engineering tools. The ability of engineering tools to interact is normally called interoperability [91]. Interoperability can be achieved by either tool integration towards a “tool suite” or a file-based data exchange. Of these two approaches, integration involves some degree of function dependence while tools interacting based on data exchange can function independently [92]. Therefore, tool integration normally works with tools of the same vendors while file-based data exchange is applicable for tools from any vendor [93]. This research is focused on the approaches to file-based exchange.

Basically, there are three types of approaches to data exchange, as shown in Figure 2-20. Obviously, compared to data exchange via point-to-point interfaces or standard interfaces, the approach to exchanging data based on a common data format significantly reduces the number of interfaces and associated problems. Apart from the function as a data exchange format, the common data format can also be used as a platform for automated mechanisms of information technology [94, 95].

For the data of specific disciplines involved in VC, there are widely accepted standard formats available, which enable different engineering tools to be able to exchange data efficiently. For geometry and kinematic data, the Standard for the Exchange of Product

model data (STEP) has been widely supported by many engineering tools. There are also some other open standards available for mechanical data exchange, such as JT, IGES etc. Therefore, efficient data exchange from CAD tools to the VC tools can be achieved. The issues lie in the exchange of VC models which are integration of data of multi-disciplines.

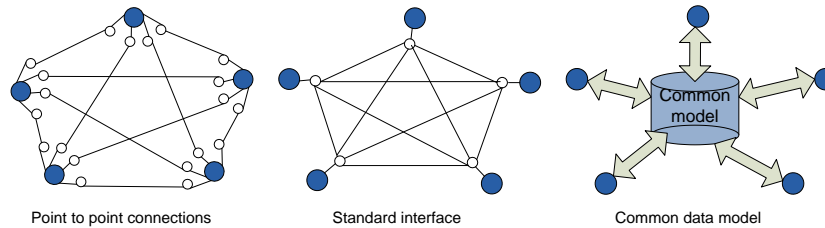


Figure 2-20 Existing approaches of data exchange

2.4.3. Data Representation of VC Models

VC tools make use of 3D modelling technologies to provide an intuitive model the characteristics (spatial and behavioural) of which result from the integration of various types of engineering data. The number, type and format of the data that can be integrated depend on the particular tool being used. The review of relevant engineering tools suggests that the virtual models of components or systems are normally composed of the following information, as shown in Figure 2-21:

- 3D geometry and modelling data
- Mechanical behaviour data
- Real time behaviour modelling data
- Modelling data integration

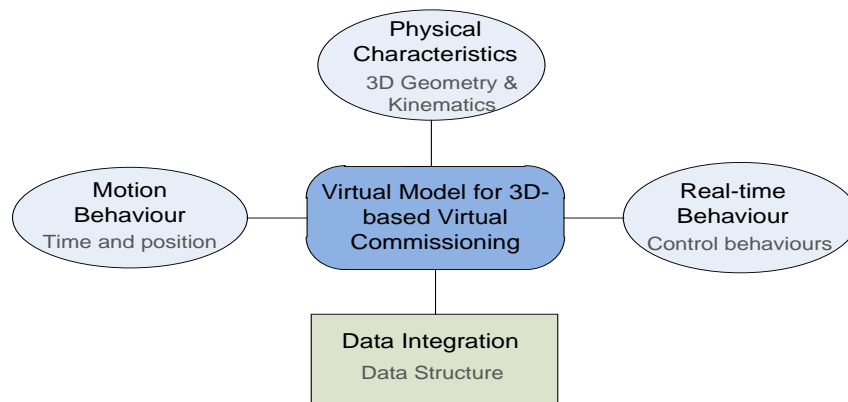


Figure 2-21 Virtual models integrates information from multi-disciplines

A. Geometry and kinematic data representation

In the current available VC tools, the data architecture and data format used for representing the validated VC models vary from tool to tool. The geometry data is normally modelled as virtual reality models. According to the review, VRML data format is widely supported by many VC engineering tools. The typical VRML-based VC tools are INVISION and Winmod. Another two widely used engineering tools - Delmia Automaiton from Dassault and Process Simulate from Siemens have their respective proprietary data format and they also support the importing and exporting of VRML data. Apart from VRML, another open standard – COLLADA is also a widely used data format. The kinematic and the motion behaviour i.e. reachable positions, on the other hand, are normally represented in tool-specific data format since there is no well-accepted standard for describing these kinds of data.

B. Control logic representation

In HIL and SIL, the control logic used is typically real PLC control code which is mainly represented as graphical languages, such as ladder logic, function block or SFC diagram. However, although IEC61131-3 has been recognised as an industrial standard for years, vendor-specific PLC programming languages still exist and are being used. A typical example is the SFC+ of Siemens Step 7. In terms of the data format for describing the PLC control code, the diversity is even more than that for programming languages. Although the PLCOpen organisation has made many efforts in defining PLCOpenXML for describing control code in IEC61131-3 programming language and striving to make it an industrial standard, vendor-specific data formats are still supported by respective PLC platforms. For example, the Siemens Step 7 still uses its own plain-text-based data format for importing and exporting PLC control source code and Schneider PLC use a vendor specific XML-based data format.

C. Hierarchical topology data representation

Regarding the representation and storage of hierarchical data, which represents the integration of mechanical data and control logic, currently no description framework or language for describing hierarchical information is widely accepted by the industry. The ways of storing this sort of information vary from one tool to another. Although most of the engineering tools can export these data as XML-based files, the semantic is still an issue to be

resolved since the terminologies are mainly tool-specific and are thus difficult to be interpreted by other tools.

A comparison of the data formats of different related engineering tools is presented in Table 2-5.

Table 2-5 Data representation in representative VC tools

	Delmia Automation	Process Simulate	Winmod	INVISION	CCE Tool
Data Format of Geometry Import	CATIA/DELMA V5 native data, VRML and STEP	Siemens PLM native data, VRML and STEP	VRML	VRML and Siemens PLM native data	VRML
Integrated Behavior-simulation	AS,KOP,FBS(Similar to IEC 61131-3)	Single Function block	Function block diagram	-	State Transition Diagram
Hierarchy info	Tool-specific	Tool-specific	Tool-specific	Tool-specific	Tool-specific XML

2.4.4. Tool-independent Data Description

Facing the increasing number of heterogeneous engineering tools with individual and often proprietary data formats, the data exchange has been a significant bottleneck for the interoperability of these tools [96]. Therefore, as pointed out in section 2.4.2, a cross-function data exchange based on a tool-independent data format is of key importance.

Current approaches to achieving efficient data exchange during automation system engineering are mainly based on three different types of data description approaches, namely XML-based, formal description-based and open standard based.

2.4.4.1. XML-based

It is evident that following the data exchange approach based on a common data format can significantly reduce the number of interfaces as well as the expenditure on maintenance. To realise the interoperability of engineering tools, the eXtensible Markup Language (XML) [97] was regarded as a promising technology for exchanging data or metadata over different platforms and systems. Obviously, XML has its own merits to be used for inter-tool data exchange due to its characteristics of platform-independence and flexibilities. However, a big

limitation of XML is that it cannot provide semantic interoperability, which refers to providing systems with a consistent way to interpret the meaning of data, information and knowledge, for data exchange [98].

Existing XML-based data exchange approaches described by other researches normally adopt XML as the basis of data exchange format but they only describe the approaches conceptually. To achieve an open architecture for the Digital Factory, Kuhen[69] proposed an XML-based scalable digital enterprise backbone as a common data model to transform the process of digital manufacturing within the digital factory, as shown in Figure 2-22. An XML-based information model which facilitates data exchange among the machine shop's manufacturing execution system, scheduling system, and simulation system, has been developed at the National Institute of Standards and Technology (NIST) and was described by Yan et al. in [99]. Using this model, the data transformation and exchange between a database system and an XML can be realised.

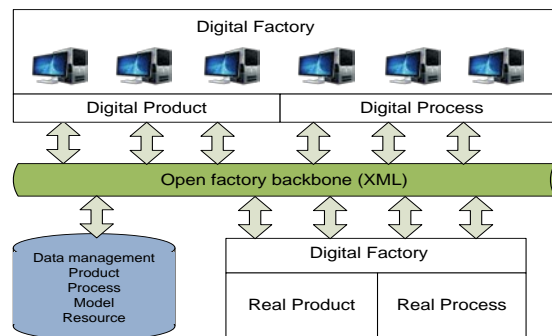


Figure 2-22 XML-based communication and integration in Digital Factory [69]

2.4.4.2. Formal description methods

XML can be used for data exchange however extra efforts are still required to achieve semantic interoperability. In order to achieve semantics, the simplest solution is to build shared meta data repositories that describe the shared data in the same ways. However, this is not efficient and also impossible to achieve due to both technical and business related issues. Another solution is to develop an ontology to support interoperability.

The most commonly quoted definition of the ontology is “*a formal, explicit specification of a shared conceptualization*” [100]. Compared with XML, ontology provides formal and explicit description of shared concepts and the relations between the concepts. Ontology has been widely accepted as the de facto standard way of achieving semantics, especially in the

area of the semantic web. Ontology is a comprehensive concept and enormous amount of literatures related to ontology can be found. This section only covers the ontology-related research in the domain of manufacturing and automation.

In the area of automation, many researchers have adopted ontology to achieve semantic data interoperability. Moser et al [80] proposed an ontology-based data modelling approach focused on providing links between data structure of engineering tools to support the semantic integration in manufacturing automation system engineering. This approach, named Engineering Knowledge Base (EKB) aims to support explicitly modelling of existing knowledge in machine-understandable syntax. The main objective of EKB is to facilitate the efficient data exchange between tools and data sources by providing an explicit and machine-understandable representation of the so-called Virtual Common Data Model (VCDM). Based on this data exchange, more complex engineering process tasks like performing analyses across tools can be supported. Another ontology-based information integration framework for mechatronics system is presented by Bi et al. in [101]. The framework adopts component-based architecture which facilitates the design and simulation applications plug-and-play in the framework to meet the requirement of mechatronics system multi-disciplinary design and to guarantee extensibility. Two main aspects of this framework are Mechatronics System Ontology (MSO) to capture key concepts and relationships in mechatronics system multi-disciplinary design process, and standard component interfaces which specifies the interfaces in an abstract form which are used to exchange information among components in a standard way.

Considerable researches have been done to identify the relationships and differences between ontologies and other data formats or programming languages [102-104]. Some features of ontology-based approaches can be observed from the existing research and comparisons. Firstly, ontology languages are more difficult to learn and require mathematical training as they normally provide mathematical rigor for analysis and prototyping of designs. Current ontology languages are mainly designed for and used by web-based applications which requires not only semantic sharing of knowledge but also additional functions for reasoning and querying. Reasoning is necessary to derive the information that is not expressed explicitly in the shared information, for example to retrieve all products that produced by a specific manufacturer. Enabling reasoning and querying capacity is the reason why shared information should be described in formal language. They are essential for data

integration and web-based searching. Data models described using formal approaches are normally impossible to read directly by human beings. Although OWL is accepted as a de facto standard language for web-based ontology, there are simply too many ontologies in other domains and no commonly agreed standard representation of these ontologies [98]. In this context, additional transformations between ontologies are required.

2.4.4.3. AutomationML - A neutral data format for automation engineering

Data exchange based on domain-specific open standards is another effective way of achieving interoperability. In the domain of manufacturing automation, there are a number of existing domain-specific standards. However, these standards or data formats are mainly specific to data description within respective disciplines. Typical examples include STEP and IGES for CAD data exchange, JT, VRML and COLLADA as 3D data formats, PLCOpenXML [105] as a control logic data format. Considering that the virtual models required in VC are the integration of multi-disciplinary data, the following sections will not elaborate on these discipline-specific data formats. They will provide an overview of AutomationML, a new data format which is specific to the domain of automation and provides capabilities of describing multi-disciplinary data.

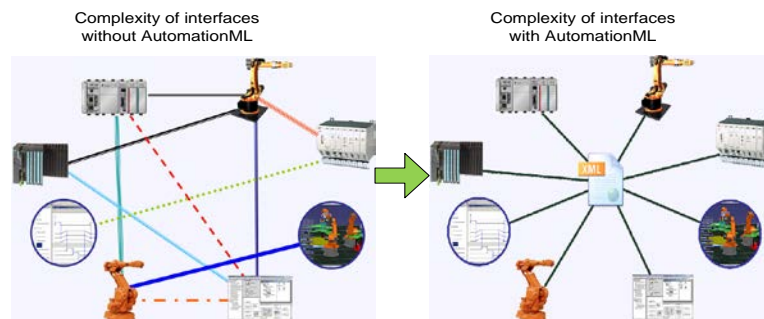


Figure 2-23 AutomationML reduces complexity and closes gaps [106]

AutomationML (Automation Markup Language) [106], a neutral data format usable for data exchange among the various discipline-specific engineering tools involved in the whole process of manufacturing systems engineering, was developed and released in 2009 by the AutomationML organization the members of which include Daimler AG, ABB, Siemens, Rockwell, Kuka, netAllied, Zühlke and the University of Karlsruhe and so on. The goal of AutomationML is to interconnect engineering tools from the existing heterogeneous tool landscape in their different disciplines, e.g. mechanical plant engineering, electrical design, process engineering, process control engineering, HMI development, PLC

programming, robot programming etc, as illustrated in Figure 2-23. In the following subsections, the general architecture, capability of information exchange and base library of AutomationML will be described.

A. Object-oriented architecture

AutomationML is an XML schema-based data format designed for the vendor independent exchange of plant engineering information. AutomationML stores engineering information following the object-oriented paradigm, which is introduced by CAEX, and allows modelling of real plant components as data objects encapsulating different aspects typically consisting of its geometry, kinematic, behaviour, position within the hierarchical plant topology and the relations to other objects. An object can consist of other sub-objects, and can itself be part of a larger composition or aggregation.

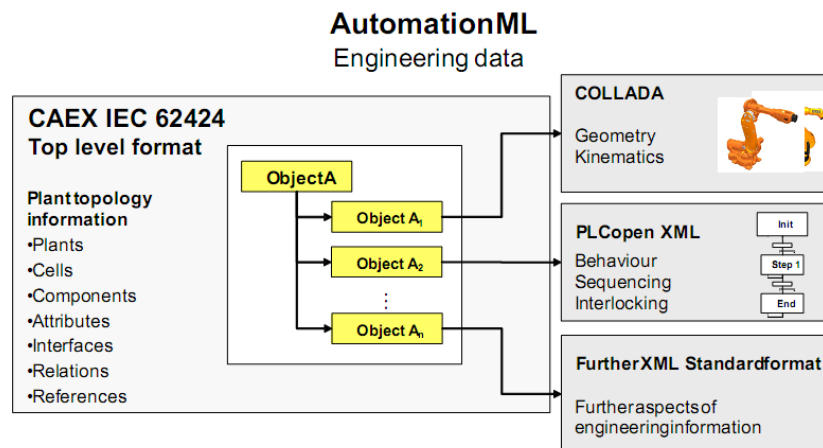


Figure 2-24 Architecture of AutomationML[106]

As shown in Figure 2-24, AutomationML combine existing industry data formats for the storage of different aspects of engineering information: COLLADA is used for storage of geometric and kinematic information, PLCopen XML serves for the storage of sequences and behaviour and CAEX is used as the top level format that connects the different data formats to comprise the plant topology. Therefore, AutomationML has inherent distributed document architecture. Moreover, there are many advantages of this architecture: (1) usage of proven and established file formats which reduces the specification effort for AutomationML, (2) the distribution of data to different files which eases the handling of bulk information, (3) the simplified usage of library files which can be stored, exchanged and accessed separately, and (4) the ability to store different geometry variants separately.

According to the description in object-oriented programming, before an object is created, a class need to be defined as first the blueprint (template) of this object. Therefore, AutomationML provides the definition of following basic classes and libraries consisting of corresponding classes:

Interface Class: Interfaces classes describe relations between objects. AutomationML provides a predefined interface library consisting of a number of abstract interface classes for general automation systems. These classes are syntactically and semantically well-defined and allow the modelling of user defined interface instances.

Role Class: A role class describes the general functionality of a CAEX object within its context, for example, a robot, a roller bed, a PLC etc. A role class facilitates semantic interpretability of a user defined object.

SystemUnitClass: SystemUnitClass refers user-defined classes for specific objects. The class must be defined based on standard role classes.

B. Capability of information representation and storage

The information that can be stored and exchanged with AutomationML includes (illustrated in Figure 2-25):

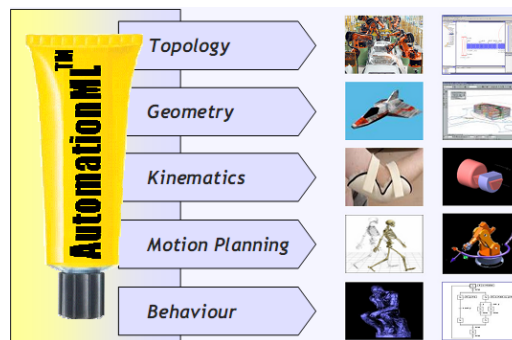


Figure 2-25 information covered by AutomationML[106]

Plant topology information: The plant topology describes a plant as a hierarchical structure composed of plant objects which describe respective items comprising the plant. Different components of an item are described as corresponding AutomationML objects which are stored in a certain level of detail, e.g. robot, gripper, but not axles or joints; an object has its individual properties and interfaces to other objects in their hierarchical

structure. The plant topology as the top level data structure is stored by means of the CAEX data format according to IEC 62424.

Geometry information and Kinematics information: Both geometry and kinematics information are stored in separate files using the file format “COLLADA” of the Khronos Group [107]. These files can be referenced out of CAEX and can be linked within CAEX using CAEX link mechanisms. The interfaces of geometry and kinematics information with CAEX can be “published” as CAEX External Interfaces within the top level format for later interlinking.

Logic information: The logic information refers to the PLC control software. This information is stored in external files by means of the data format PLCopenXML. Variables or signals in these files can be “published” as CAEX ExternalInterfaces which enables these files to be referenced out of CAEX and linked within CAEX.

Reference and relation information: Embodied in AutomationML references and relations are different concepts. References describe links from a CAEX file to other distributed files while relations depict associations between CAEX objects;

C. Existing researches on AutomationML

Since it was released in 2009, AutomationML has been adopted in many research projects from both academia and industry.

Many academic researchers have discussed the potential of adopting AutomationML to improve the data exchange efficiencies [80, 108-112]. However, these discussions are just generally conceptual and at a very basic level. Some researchers partially investigated specific formats of AutomationML. For example, Persson et al. proposes an approach which focuses on converting CAEX files into RDF triples to expose them via a SPARQL endpoint via which other applications can access and query the information [113]. Researchers from Dortmund University have developed an importer of AutomationML for a Robot programming and simulation environment, however, only the import of COLLADA is supported [109].

AutomationML has also drawn the attentions of industry. Members of the AutomationML organisation, such as Siemens, ABB and Kuka, have announced the support of AutomationML in their engineering tools. However, no engineering tools with such

functionality have been released yet. The members of the AutomationML organisation have also been striving to promote AutomationML and some resultant achievements can be seen. Drath from the ABB Corporation has contributed several key papers and made presentations in recent domain-specific international conference [91, 93]. Part one of the AutomationML specifications has been accepted as an IEC standard (IEC 62714) and promises to be an open standard in the domain of automation.

2.4.4.4. Summary

Based on the review, the features of the reviewed methods or formats for tool-independent data representation can be summarised as follows (also see Table 2-6):

- XML is widely used for data description and data exchange; however, it does not provide any semantics and this could potentially leads to misunderstanding.
- Formal methods provide explicit and formal description and therefore can guarantee semantics; however, formal methods are complex and normally not used for data exchange between engineering tools. They are mainly used for web-based applications and knowledge management applications.
- Existing domain-specific open standards are mainly focused on data description of specific disciplines and therefore not suitable for the description of component-based virtual models which integrate multi-disciplinary data.
- The newly released standard AutomationML has a suitable architecture and data description capability required for describing virtual component-based automation systems. However, currently few tools support AutomationML.

Table 2-6 Comparison of existing data representation methods and formats

Options	Languages/ Formats	Advantages/Features	Limitations
XML	<ul style="list-style-type: none"> • XML 	<ul style="list-style-type: none"> • Simple • Widely used 	<ul style="list-style-type: none"> • No semantics
Formal Description	<ul style="list-style-type: none"> • OWL, • RDF • etc. 	<ul style="list-style-type: none"> • Guarantee semantics of data models • Object-oriented architecture • Mainly XML-based • Widely used in web-based applications and knowledge description 	<ul style="list-style-type: none"> • Complex • Poor readability • Not suitable for inter-tool data exchange
AutomationML	<ul style="list-style-type: none"> • CAEX, • COLLADA • PLCOpenX ML 	<ul style="list-style-type: none"> • IEC Standard • XML-based • Object-Oriented • Provides domain-specific semantics 	<ul style="list-style-type: none"> • New standard • Not widely supported

2.5. Assessment and Summary

2.5.1. Assessment of state-of-the-art

Modularity, of both hardware and software elements, is a facilitator for rapid system (re)configuration by providing units that can be (re)assembled in different combinations to achieve different functionalities. Existing modular approaches use mechatronic devices encapsulating control functions as the basic building blocks of RMS. Modularity of either high level (coordination) control or low level (logic) control has been covered by the reviewed modular approaches. However, the survey of current industrial practices indicates that few of these modular control approaches has been adopted in a large scale by industry.

PLC-based control systems are still widely used by industry. The traditional process of PLC control software development and validation is mature and stable; however, it is also time-consuming and expertise-reliant since all the work is manually carried out by control engineers at the last stage of manufacturing system engineering. In order to relieve this situation, efforts have been made and a number of new approaches have been proposed. The approaches, which aim at improving rather than radically revising the existing process of PLC control software development, are helpful but limited in their impact. Of the emerging approaches from academia, formal verification enables some aspects of control engineering to be better analysed thereby compressing the overall time of automation system engineering. However, its high modelling complexity and required languages, which are normally very mathematical and totally different from the traditional ones used for programming, hinder it from being adopted by industry. On the other hand, the approach to automatically generating PLC code based on high level models seems promising since it automates the control software development. Unfortunately, the current approaches from both academia and commercial tools can only generate pieces of PLC control software and manual programming of the remaining pieces is still required.

VC mainly aims to facilitate the validation of control logic. The HIL and SIL approaches reduce the time of control software engineering by bringing control software validation activities forward to be performed in an virtual environment thereby also decreases the cost of system building. The problems of HIL/SIL observed are that PLC control software still needs to be programmed manually before performing VC and extra time and efforts are needed to connect virtual models to real or simulated PLC controllers.

From the review of current research it can be seen that the virtual models validated by VC can be further reused during the machine operation phase as well as during the system re-design phase; however, the current diversity of data representation approaches and data formats of different engineering tools also present significant barriers to reuse these virtual models. There are many neutral data formats for describing specific aspects of VC models. However, in order to support a modular approach, a VC mechatronic model, which is an integration of multi-disciplinary information, needs to be reused as a whole. To date, no well-recognised data format for describing the hierarchical topology information has been widely adopted. A neutral data format – AutomationML, which is specific for the automation domain and can potentially become an open standard, is suitable for the data representation and data exchange of modular VC models. Nevertheless, no modular virtual models completely described in AutomationML have been found.

The state of the art is summarised in Table 2-7.

Table 2-7 A summary of the state of the art

State of the art	
Modular approaches to Automation System Engineering	<ul style="list-style-type: none"> • Few existing modular control paradigms have been adopted by industry on a large scale
Control system engineering	<ul style="list-style-type: none"> • Mainly PLC-based control systems • Traditional manual process of control software engineering is being used • Emerging approaches of automatic code generation still have limitations
Virtual Commissioning	<ul style="list-style-type: none"> • Mainly Hardware/Software-in-the-Loop which requires PLC code • Manual programming is required
Openness of VC Tools	<ul style="list-style-type: none"> • Reuse of virtual models is needed • Lack tool-independent model to represent VC virtual models

2.5.2. Identification of research gaps

Based on the assessment of the state-of-the-art researches reviewed, the following research gaps are identified:

- In order to build the new component-based VC framework, a direct deployment approach, which automates the way of deploying executable PLC control software based on the virtual models, is needed. This approach is proposed by the author and is described in Section 3.5 of this thesis.
- Related engineering tool features for implementing the proposed direct deployment approach are needed and currently missing from the functionality of the engineering tool involved in the new VCOM virtual commissioning framework. The needed tool functionality is designed by the author and is presented in Section 3.6. The implementation is also described in Section 4.1.2.
- Common data models for representing the component-based virtual models, which play a key role in efficient inter-tool data exchange, are needed. This gap is addressed in this research by creating the required data models which are described in Section 3.2.
- Engineering tool features for mapping the tool-specific component-based virtual models into the proposed common data models are needed. The needed functionality is developed in this research and the design and implementation are respectively presented in Section 3.3 and Section 4.1.2.

Chapter 3. Approach and Methodology

This chapter describes the enabling technologies developed in this research to realise the control deployment function and the data reusability of the new VCOM framework. The chapter begins with an overview of the framework and the required key enablers.

After the overview is given in section 3.1, the discussion is organised in six sections. Section 3.2 covers the common data models used to provide semantic descriptions of virtual models for facilitating data exchange and data reuse. The functions required for transforming component-based virtual models to the proposed common data models are described in section 3.3. Section 3.4 and section 3.5 describe the two steps of work for deploying the control software based on the control logic validated using the component-based approach. Section 3.4 presents a deployable PLC control software architecture proposed to facilitate the deployment while section 3.5 describes the approaches to the direct deployment based on the deployable architecture. Section 3.6 describes and illustrates the general design of an engineering tool that facilitates control software deployment as well as virtual model mapping. Section 3.7 summarises the work presented in this chapter.

3.1. VCOM - A New Open VC Framework

This section provides an overview of the desired open VCOM framework. First, based on the summarisation of the state-of-the-art VC approaches, the need for an open VC framework is justified. Subsequently, the VCOM, a new open VC framework, which enhances virtual model reusability and automates control software deployment, is described and illustrated.

3.1.1. The Need for the Open VC Framework

The literature review suggests that VC, mainly HIL or SIL, has been regarded as an effective approach to facilitating control software development and validation. In the current HIL approaches, Virtual Reality (VR)-based 3D models are normally driven by pre-programmed control software which run on either real PLCs or simulated soft-PLCs. This method is very mature and adopted by most of the mainstream VC engineering tools. It can be also observed, from existing relevant research, that there are also increasing demands for the VR-based applications during the operational phase of automation systems. These applications actually involve the similar virtual models to those used for VC. However, the efforts to date have been mainly invested in developing VC engineering tools and applying

these tools in actually performing VC. This is often because the virtual models built using a particular VC tool are constrained inside this tool or other tools from a specific vendor. Due to data format diversity, the need for reusing VC models is thus poorly supported.

The current approach of validating real PLC codes using HIL approach has proved to be effective and reliable. However, a number of drawbacks exist in the HIL approach. First of all, the way in which related engineers other than control engineers cannot be involved in control engineering has not been changed. The development and debugging of PLC control programs still heavily rely on the experience of control engineers. Secondly, building the connections between virtual models and PLC control system is time-consuming. Although the CCE tool can only validate component-based control logic instead of real PLC control software, the above shortcomings of HIL do not exist in CCE.

Given the above facts, this research develops the technologies required to realise a new VCOM VC framework which resolves the mentioned limitations of current VC approaches.

3.1.2. Overview of the Open VC Framework

The new VCOM open VC framework is shown in Figure 3-1. Automation systems are first virtually prototyped and commissioned using component-based approach in the engineering tool CCE. After VC, the validated component-based virtual models are then mapped to a common data model for further reuse. The PLC control software is then directly deployed based on the control logic of component-based virtual systems.

The work carried out by the author in this research to make this VCOM framework feasible, open, and efficient is marked in green in Figure 3-1. First of all, common data models for describing component-based virtual models are proposed and designed. This is mainly to eliminate the data exchange barriers between virtual engineering tools which hinder the reuse of HIL virtual models by other relevant applications. A common data model for describing modular virtual models is defined and represented in a well-accepted domain-specific standard data format. Subsequently, in order to map the component-based virtual models to the proposed common data model, the approach to the mapping is then designed and implemented. It is worth mentioning that the author's approach mainly covers mapping of hierarchical topology data. 3D geometric and kinematic data is not the focus of this research due to the fact that standard engineering tools for the transformations of such data are already available.

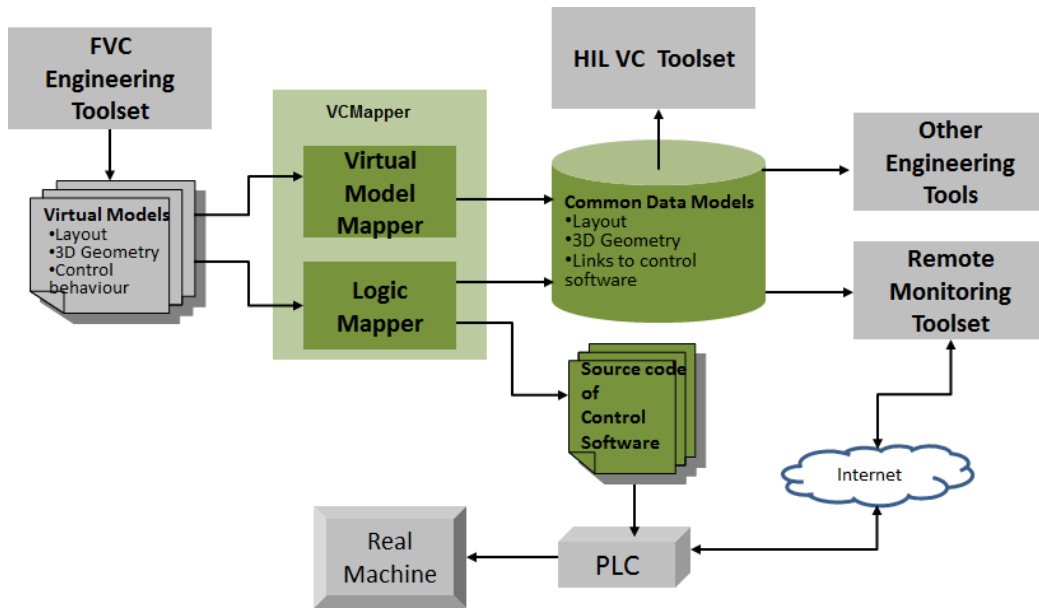


Figure 3-1 the objective open VC framework - VCOM

Table 3-1 Summary of research work covered

Work	Description	Section
Common data models	Data models created using AutomationML’s data description approach for describing component-based virtual models.	3.2
Virtual model mapping	Maps component-based virtual models into the corresponding HIL common data models.	3.3
Deployable PLC control software architecture	Introduces a new PLC control software architecture which is the basis of the proposed direct deployment approaches.	3.4
Direct deployment approach	Generates PLC control software automatically, with the proposed deployable architecture, through reusing the component-based virtual models and pre-programmed runtime components.	3.5
Engineering tool VCMapper	- An engineering tool which implements the functions of virtual model mapping and control software deployment.	3.6

The author’s approach to deploying complete PLC control source code based on the control logic of component-based virtual models are then proposed and designed. The objective of the direct deployment approach proposed in this research is to enable the automatic generation of source code for complete PLC control software based on the component-based control logic validated by CCE. To achieve this objective, a new control

software architecture, which was created by the Automation Systems Group for facilitating the direct deployment, is described first. Design of the direct deployment approach based on the deployable architecture is then presented.

Lastly, design of an engineering tool to implement the virtual model mapping and the direct deployment is described.

3.2. Virtual Modular Common Data Model (VMCDM)

This section describes the common data models that are proposed to provide semantics for virtual models of component-based automation systems.

First, the requirements of the common data models are considered. The common data models are built based on AutomationML which provides the basic structure, terminology and data formats for describing automation systems. Then, the classes required for describing the classification of objects and relationships between objects are presented. These classes were designed by inheriting the standard classes of AutomationML.

Consequently, the classes for describing the different concrete objects of the virtual models are presented.

3.2.1. Basis of VMCDM

The common data model, which is named Virtual Modular Common Data Model (VMCDM), involved in this research was designed with the aim of facilitating data reuse between different VC engineering tools. Before finally building the desired common data model, it is necessary to first clarify the requirements of the common data model for this purpose and then justify for the adoption of AutomationML as the description framework.

3.2.1.1. Requirements of the Common Data Models

The key characteristics of ontology, which is widely adopted to provide semantics, are formal, explicit and shared. The generic definitions of these characteristics were provided by Ushold et al. [114]. Ushold also categorised different ontology application scenarios. For different scenarios, the specification of the ontology and the languages required can be different. For instance, the work presented by Runde and Fay specified the data format required for data exchange in building automation system engineering [94]. The common data model proposed in this research is to be used for the purpose of “common access to

information”. To achieve this, the requirements of the common data model proposed in this research are identified as following:

Object-oriented architecture

The virtual models used to perform VC in this research are built according to the Component-Based approach [4]. A component encapsulates information of multiple disciplines and is used in a black-box manner when a system is built. To describe the components and the systems based on such an approach, the required data model should support the object-oriented architecture.

Semantic Requirements

Semantic interoperability implies that terms used in a given engineering tool can be interpreted by other engineering tools. To achieve semantic interoperability in this research, the common data models were created with reference to the method of creating ontology. The key elements required to create an ontology have been summarised in many articles [102, 114, 115]. In the case of this research, the desired data model, which is used for describing the hierarchical topology information, should contain the following concepts:

- Properties including data types and relations which are used to describe hierarchical information.
- Classes for defining the data models of constituent objects of component-based automation systems
- Classes for defining the integration between the hierarchical objects and the related disciplinary objects such as 3D geometry model and control logic.
- Instances of the defined classes.

XML-based data format

XML schema cannot provide the required semantics. The differences between XML schema and ontology has been identified by Klein [102]. However, most of the ontology description languages still adopt XML as the data exchange format. XML has been regarded as an open and effective way of resolving syntax issues and syntax interoperability is also a basic requirement of tool interoperability. Therefore, the proposed Virtual Modular Common Data Model (VMCDM) should be described in XML-based data formats. The platform-

independent feature of XML also guarantees that the VMCDM is usable by different applications running in different operating platforms.

3.2.1.2. Common Data Model based on AutomationML

Based on the requirements summarised above, the author adopted the AutomationML as the basis of the proposed common data model. AutomationML, instead of being built from scratch, is built based on the existing XML-based data format - CAEX. This section provides a discussion of the respective contributions of CAEX, AutomationML and VMCDM to generally show that why AutomationML is adopted and how the desired VMCDM is built.

CAEX – A generic description framework

CAEX is an XML-based neutral data format which is generic for describing hierarchy information of various domains such as buildings, machinery, plant, etc. The general goal of CAEX is the vendor independent storage of hierarchical object information. Although CAEX is a semi-formal description framework rather than a formal language, it still provides some essential concepts required for describing semantic. First of all, it supports library concepts and object-oriented architecture. Secondly, it defines the vocabulary of properties for describing hierarchical information. Additionally, the terms for describing relationships between classes and instances are also defined in CAEX.

As shown in Figure 3-2, at the top level, the CAEX data model consists of: three different types of libraries – RoleLibrary, InterfaceLibrary and SystemUnitLibrary – and the specific plant structure- InstanceHierarchy. The information of a specific system, named InstanceHierarchy in CAEX, is built by combining the instances of predefined classes in the SystemUnitClass Library. Before defining the SystemUnitLibrary, the RoleLibrary and InterfaceLibrary need to be defined. RoleClasses are defined in order to assign a role to an instance object and to describe its general requirements. The main goal of the CAEX role concept is the separation of abstract role information and the definition of concrete implementation information which is included in SystemUnitClass. InterfaceClasses, on the other hand, are used to describe the relations between different items. The relations between those four concepts are illustrated in Figure 3-2.

In the CAEX approach, prior to the definition of system unit classes for describing components, required roles and interfaces need to be defined. The roles implement an “I am

a...” relationship while interfaces determine the type of relation and the semantic meaning of the connection between two objects that are connected via their respective interfaces.

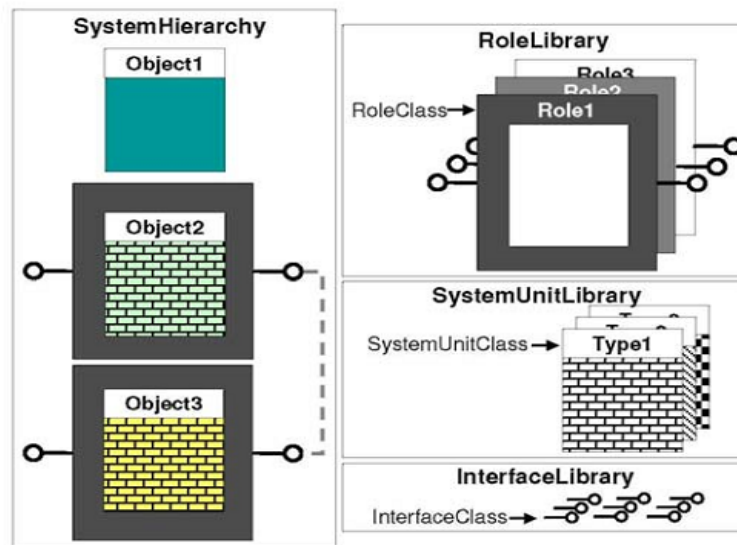


Figure 3-2 CAEX items and their relations [116]

AutomationML – Defining domain-specific terms for automation systems

While the concepts and architecture for describing object-oriented approach have been defined in CAEX, CAEX does not provide the definitions of domain-specific RoleClass and InterfaceClass. Also, SystemUnitClass and InstanceHierarchy obviously are not defined since they are user-specific. In this context, AutomationML defines role classes and interface classes which are specific for the automation domain. This is of significant importance to enable the semantic data exchange in the domain of automation systems. Nevertheless, AutomationML only defines the names and descriptions of these roles and interfaces. The concrete implementations still need to be defined according to the users' needs. Extended user specific roles or interfaces can also be defined by inheriting the normative roles of AutomationML. Despite the user specific items still existing, the semantics of those items can be obtained considering they are derived from predefined standard concepts.

Domain specific roles and interfaces related to automation system engineering have been defined in AutomationML's normative library, however, these definitions only include names and descriptions and are basic and high level ones, for instance, top level roles - resource, product, process and top level interfaces – COLLADAInterface, PLCOpenInterface. Depending on the granularity and aspect of the classification, user specific sub-roles and sub-interfaces still need to be defined by inheriting the normative ones. It is worth noting that

increasing number of user defined roles and interfaces potentially lead to weaker semantics. Therefore, user specific roles and interfaces should be defined only if they are really required and are missing from the normative library.

Common Data Model based on AutomationML

The essential characteristics of a formal language for describing ontology are object-oriented architecture, vocabulary of properties and relations, and logic descriptions for reasoning and querying. AutomationML supports object-oriented architecture and provides domain-specific vocabulary but no capability for reasoning and querying. However, the function of reasoning and querying is mainly used for web-based data integration and not required for data reuse between engineering tools.

The object-oriented architecture of AutomationML is suitable for describing component-based modular automation systems. Also, the role classes and interface classes defined by AutomationML are suitable for describing automation systems. However, roles and interfaces specific for describing virtual models are missing from the corresponding library of AutomationML. Therefore, role classes and interface classes required for representing VMCDM need to be defined.

Moreover, the SystemUnitClasses of AutomationML which corresponds to the components of VMCDM are completely user-specific. Therefore, the classes for describing different types of components of VMCDM are then defined. Obviously, the HierarchicalInstance which is used to describe a specific system is mainly system-specific and can be created based on the SystemUnitClasses created in this research.

The comparison of the respective contributions of CAEX, AutomationML and VMCDM are summarised and demonstrated in Figure 3-3. CAEX provides vocabularies of generic concepts and relations required for describing hierarchical data models, and then AutomationML adds the role classes and interface classes to CAEX as domain-specific standard classes for describing automation systems. Finally, based on the standard classes of AutomationML, VMCDM defines the specific role classes, interface classes and system unit classes for describing the virtual models of component-based automation systems.

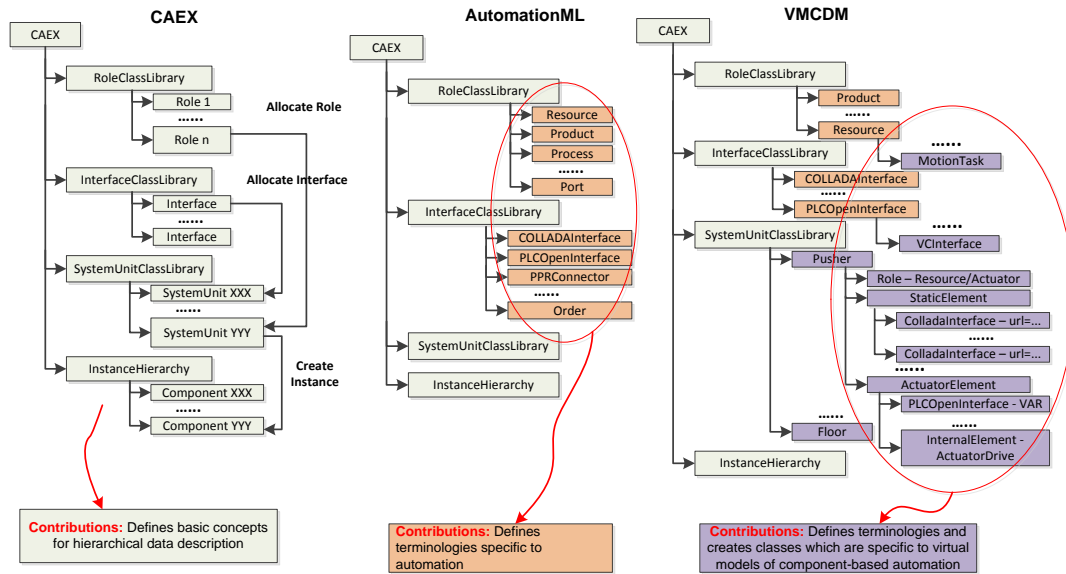


Figure 3-3 CAEX, AutomaitonML and VMCDM and respective contributions

3.2.2. VMCDM for Component-based Automation Systems

The constituent objects of a component in the component-based approach have been described by Harrison et al. [8]. In current virtual commissioning approaches, the virtual prototype of a physical system normally contains part of the constituents which are required for the validation of control logic. The main objective of designing the VMCDM is to bring semantics to the data representation of virtual models. In order to achieve semantic representation, as analysed in section 3.2.1, it is of key importance to define all the required properties and objects based on AutomationML. The required properties and objects are determined by the constituents of virtual components to be described. The constituents of a virtual component are shown in Figure 3-4.

As it can be observed from Figure 3-4, in order to integrate the disciplinary constituent information, the hierarchical topology of a virtual component has the following data:

- Interfaces to geometric elements: each element has an interface to its 3D geometry information which is saved separately.
- Object for describing state behaviour: for an actuator, the state behaviour is represented using position information and time. For a sensor, it is described using different colours.
- Interfaces to the control software: one for reporting the status and another for receiving commands.

- Object for connecting a specific behaviour to the related control interfaces, which control the state behaviour, and the related geometry interface, which specifies which 3D geometry model performs the mechanical behaviour.

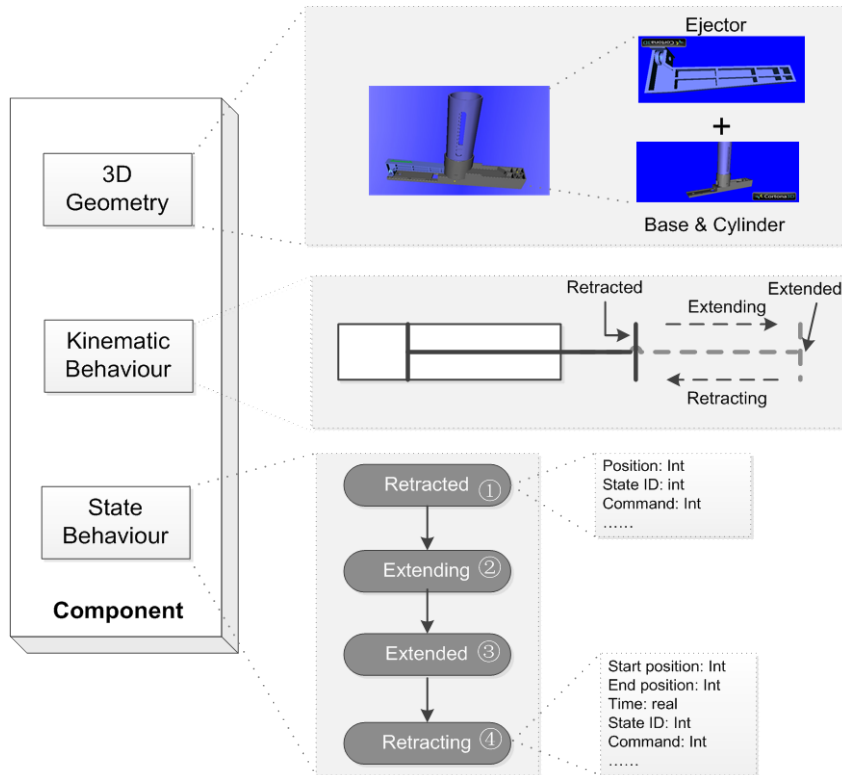


Figure 3-4 Constituents of a virtual component

It is worth mentioning that, some components, a floor for instance, have no mechanical behaviour and control behaviour. For those components, only interfaces to the geometry data are needed.

According to the architecture of the components, the architecture of VMCDM is built based on AutomationML's approach and is illustrated using Unified Modelling Language (UML) in Figure 3-5.

Based on the architecture in Figure 3-5, the required classes and elements that are not included in the standard library of AutomationML will be defined and described in the following sections. These objects include role classes, interface classes, system unit classes and some constituent objects of system unit classes.

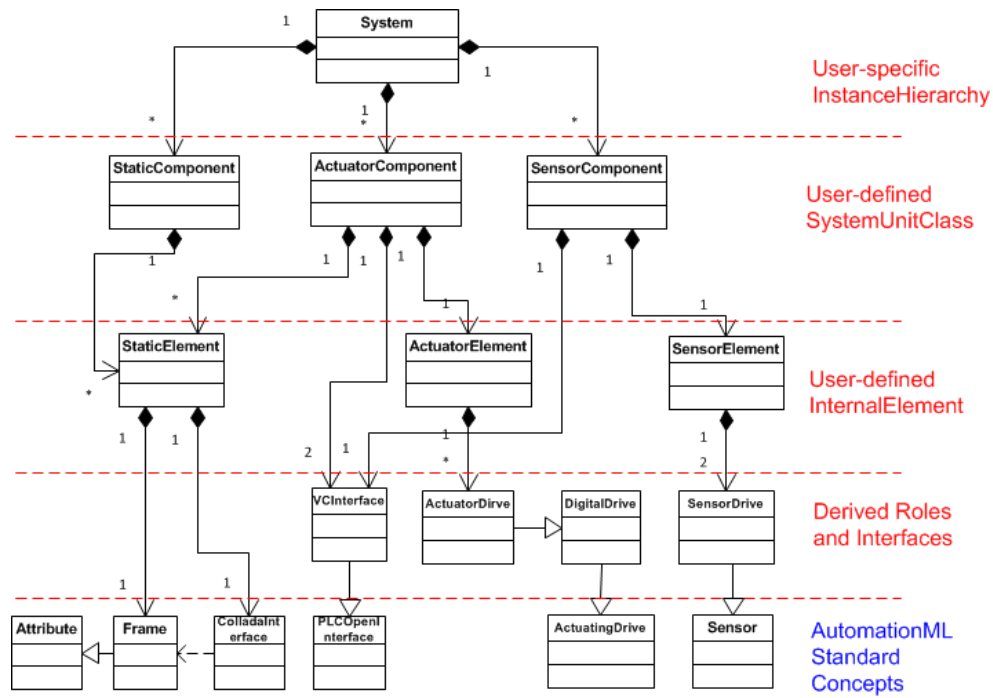


Figure 3-5 Structure relationship between objects of VMCDM

3.2.3. VMCDM-specific Role Classes and Interface Classes

The normative library provided by AutomationML defines classes of roles and interfaces required for describing automation systems. A virtual model, which can be seen as the virtual equivalence of the corresponding physical system, typically integrates only geometry, kinematic and control logic information. However, in order to integrate relevant data together into final data sets that describe virtual models, additional classes still need to be defined.

3.2.3.1. VC-specific Interface Classes

In order to connect the mechanical behaviours of a virtual model to its corresponding control behaviours, an interface class called VCInterface is defined. The VCInterface is defined by inheriting PLCOpenInterface which is a normative AutomationML interface class for connecting hierarchical data with control software. As shown in Figure 3-6, the VCInterface contains the following properties:

- RefBaseClassPath: a standard attribute defined by CAEX to specify the parent class of current class.
- Direction: an attribute which indicates direction of the signal exchange between control systems and the related machine component. Its value can be either “In” or “Out”.

Other attributes can be potentially added when it is instantiated and included in a mechanical behaviour, as shown in Figure 3-6. Although AutomationML supports the object-oriented concept, there are still several differences between AutomationML and object-oriented programming languages. One of important differences is that the structure of a class can be changed when it is instantiated.

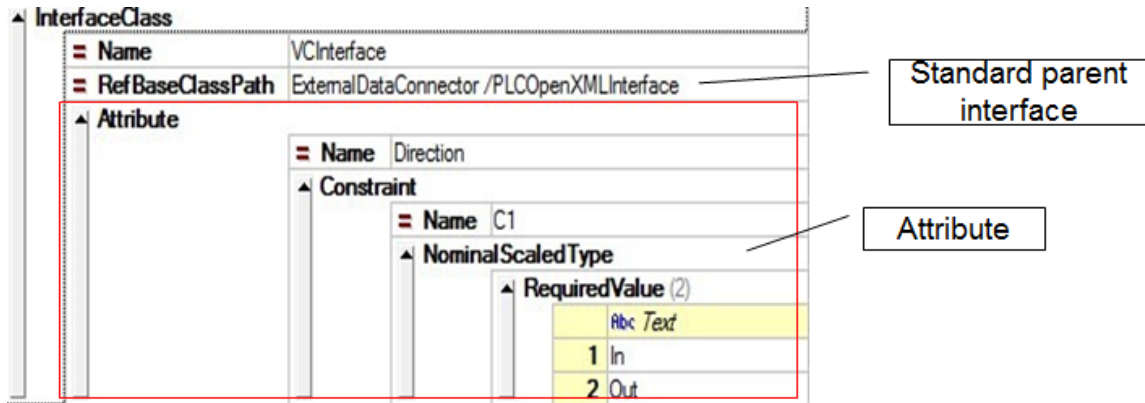


Figure 3-6 User-defined interface - “VCInterface” for connecting states with PLC control logic

3.2.3.2. VC-specific Role Classes

An important mechanism of AutomationML to achieve semantic definition is that every object must be assigned a role which should be predefined in the role library. The domain-specific standard role classes defined by AutomationML can be used as the roles for components of component-based automation systems. However, role classes for the internal elements contained in components are not defined in AutomationML therefore need to be defined. In order to achieve semantics, these roles must be defined by inheriting corresponding normative role classes.

Table 3-2 Role classes defined in this research for description of VC models

Role Name	Description	Parent Normative Role
DigitalDrive	Used for objects those contain the info of states which drive corresponding mechanical behaviors of actuators.	AutomationMLCSRoleClassLib /ControlEquipment/Actuator/ActuatingDrive/
SensorState	Used for objects those contain the info of sensor states.	AutomationMLCSRoleClassLib /ControlEquipment/Sensor/

In this research, as presented in Table 3-2, two roles classes required specifically for describing virtual models of component-based automation systems are defined and presented as follows. According to the specification of CAEX, a role class only indicates the abstract requirements of the related element and does not contain the concrete implementation information of the defined role classes. These roles are only defined by specifying its parent normative roles.

DigitalDrive for actuator states

In physical automation systems, actuators can be driven by various types of power source e.g., electric, pneumatic and hydraulic. In the normative role library of AutomationML, there is a role class named “Actuator/ActuatingDrive” used for representing the role of a physical unit for driving mechanically actuated controlling element.

In contrast to the physical systems, in the virtual models for virtual commissioning, behaviours of an actuator are simply defined and driven by “states” each of which is composed of a set of parameters including position information and time or speed information. This is because the focus of virtual commissioning is to validate control logic rather than to simulate how the movements of actuators are driven. In the context of virtual commissioning, a role class named “DigitalDrive” is defined by inheriting the role “Actuator/ActuatingDrive” in AutomationML’s informative library for control systems.

SenorState for sensor states

In the component-based approach, a binary sensor component has two states indicating the “ON” and “OFF” states. In virtual models, each state of a sensor component has an interface for publishing its state to related control behaviour and attributes for describing corresponding colour which represents the corresponding working state of the sensor. A role named “SensorState” is defined to represent the state behaviour of a sensor. In the normative role library of AutomationML, there is a role class “Sensor” and no other role classes specific for representing the states of sensors. Therefore, the role inherits informative role “Sensor” of AutomationML.

3.2.4. Element representation

According to the data structure of a component illustrated in Figure 3-4 of section 3.2.2, the constituent items of the hierarchical information of a virtual component can be classified as follows:

- **Attributes:** for describing the general properties of a component.
- **Interfaces:** reference to its 3D geometry file.
- **Attributes:** for describing the position information of the 3D geometric data.
- **Internal objects:** for describing the state behaviour of actuator or sensor components. The internal object “state” of actuator components has different attributes from the “state” of sensor components. Therefore, the object “state” can be further classified.

The concept “component” of component-based automation systems corresponds to the “SystemUnitClass” of AutomationML. According to the definition of the “SystemUnitClass” of AutomationML, a component consists of the following properties or sub-classes:

- **Attribute:** allows the specification of object attributes
- **ExternalInterface:** allows the specification of object interfaces
- **InternalElement:** allows the specification of nested internal objects
- **SupportedRoleClass:** allows specification of supported RoleClasses
- **InternalLink:** allows specification of relations between interfaces

Of the properties, “Attribute” and “InternalLink” are standard concepts of CAEX while interface classes and role classes have been defined in AutomationML. However, the classes for internal elements still need to be defined. Therefore, prior to defining the classes for representing components, the classes for elements are defined first. An element can be either a simple one or a composite one which further contains nested elements.

Based on the respective data structures of “Component” and its corresponding term “SystemUnitClass”, three types of element are defined and presented in the following sub-sections.

3.2.4.1. Actuator element

A virtual actuator component is further decomposed into an actuator element and other one or multiple static elements. A static element mainly contains an interface to its geometric data.

The actuator element, on the other hand, contains all state behaviour of an actuator component. An actuator element is further decomposed into the following properties and internal objects:

- “Rolerequirement”: property specifying its role class which should be the standard AutomationML role- “Actuator”.
- “InternalElements”: internal objects which represent the states each of which contains all the properties related to the state behaviours.

The state behaviour integrates the information of mechanical behaviour and the related interfaces to control software. The standard role “DigitalActDrive” of AutomationML is assigned to the “Rolerequirement” of each state behaviour element.

A state element is composed of three attributes which are used by engineering tools’ functions to control the distance and speed of a kinematic movement driven by this actuator.

- “StartPosition”: attribute that indicates the beginning position of a mechanical behaviour.
- “EndPosition”: attributes that indicates the end position of a mechanical behaviour.
- “Time”: the time the mechanical behaviour takes to move from the StartPosition to the EndPosition.

The specifications of the attributes are also defined. For a dynamic state, for example “Moving to Work Postion”, the value of attribute “Time” should be bigger than 0 and the “EndPostion” is not equal to “StartPosition”. For a static state, like “At Work Position”, “Time” is 0 and the “EndPostion” should be equal to “StartPosition”. These specifications will be implemented in the XML schema of the VMCDM.

Also, a state element contains the following two VCInterfaces which are used to connect a state with its related control variable.

- “StateCmd”: the command variable received from the control software to trigger the mechanical behaviour. Its direction is “IN”.
- “Status”: the variable for reporting the current status of the mechanical behaviour to the control software. Its direction is “OUT”.

Additionally, in order to provide a classified view on different types of attributes of an object, a role class “Facet” is defined in AutomationML. Internal elements with the role “Facet” can be added to an object in addition to its normal constituent items. The attributes of the “Facet” elements must be the existing attributes of the object. In the case of the actuator element, two facet elements are defined as following.

- “BehaviourFacet”: The BehaviourFacet is an internal element used for providing a view on the attributes related to the mechanical behaviours of actuators or sensors, namely attributes representing the information of time and position. Apart from the attributes, it should also be assigned a role “Facet”.
- “ControlFacet”: The “ControlFacet” is an element used for providing a view on the interfaces connecting to the PLC control software. It also has the normative role- “PLCFacet”.

Based on the above definitions, the data structure of an actuator element is illustrated in Figure 3-7.

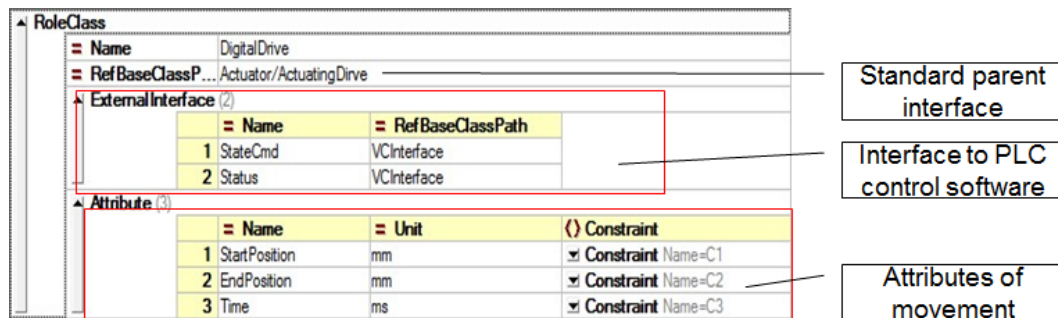


Figure 3-7 Data structure of an actuator element

3.2.4.2. Sensor Element

As defined in the component-based approach, a sensor component also has state behaviour and corresponding interfaces to PLC control software. The only differences between the state behaviours of sensor components and those of actuator components lie in the attributes for representing the states. Therefore, a sensor element for representing the state behaviours of a virtual sensor component is also defined in the same manner as that for defining actuator element. This section only describes the differences between a sensor element and an actuator element.

First of all, in VC, state behaviours of a sensor are represented by changing colour instead of position and time. A composite attribute “Colour” is contained in the sensor element;

- “Colour”: attribute that indicates the colour of the sensor state behaviour.

Secondly, given the fact that no command from the control software is needed, a virtual sensor only has one PLCInterface “Status” to report its real-time state to the PLC.

The sensor element also has the generic attribute “RoleRequirement” and two facet elements “BehaviourFacet” and “ControlFacet”. However, the “BehaviourFacet” is a view on the attribute “Colour”.

3.2.4.3. Static element

A static element contains all the attributes and interfaces that relate to the 3D geometry data of a component. Obviously, a static element does not have any state behaviour or interface to respond to control signal. Therefore, as shown in Figure 3-8, a static element is described using the following items:

- “RoleRequirement”: for specifying the role this element plays, normally set to the standard role “StaticObject”.
- “Frame”: attribute for saving the relative location of a 3D geometry model. It is a standard AutomatinML attribute and described as {x, y, z, Rx, Ry, Rz}, where x, y, and z represent the relative position and Rx, Ry and Rz represents the rotation.
- “ColladaInterface”: external Interface reference to its geometry data.

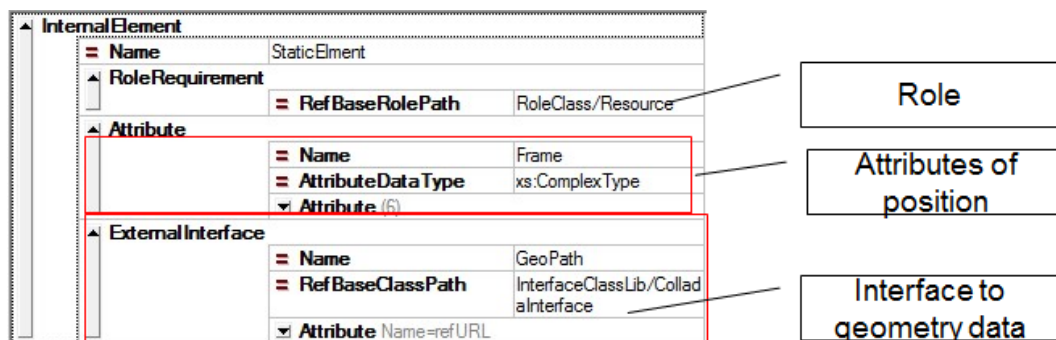


Figure 3-8 Data structure of static element

3.2.5. Virtual Component

In the component-based approach, resource components can be categorised into actuator components, sensor components and non-control components. As aforementioned, the concept ‘component’ of component-based automation corresponds to the ‘SystemUnitClass’ of AutomationML. Based on the standard items defined by AutomationML and the elements defined in the above section, the corresponding ‘SystemUnitClass’ for these three types of components are defined and presented in the following sections.

3.2.5.1. Actuator Component

An actuator component is typically composed of static elements, an actuator element and other attributes, as shown in Figure 3-9. The static elements encapsulate the attributes and interfaces related to the geometry data while the actuator element encapsulates the integration of state behaviours and interfaces to the control logic. The attribute “SupportedRoleClass” of an actuator component should be set to a specific normative role, such as “Conveyor”, “Pusher”, et al.

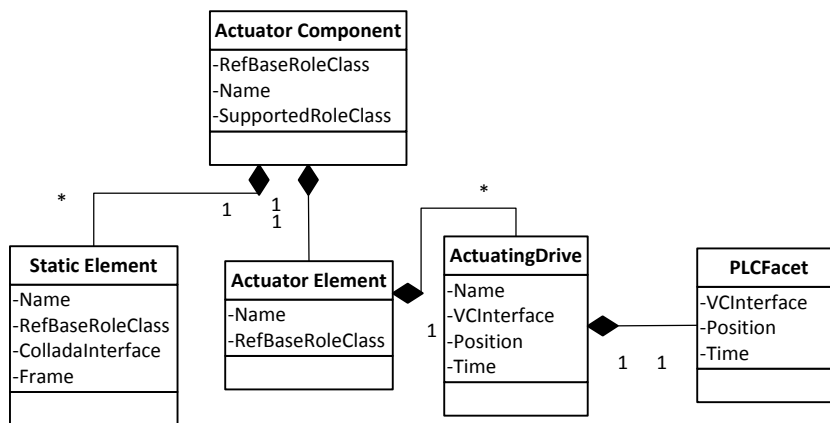


Figure 3-9 Actuator Component described in UML

3.2.5.2. Sensor Component

In the virtual models, a sensor component is defined as the integration of a 3D geometry model and two states. In VMCDM, correspondingly, the “SystemUnitClass” for describing a sensor component consists of a static element and a sensor element. It also contains an attribute of “SupportedRole” the value of which is set to the normative role “Sensor”, as illustrated in Figure 3-10.

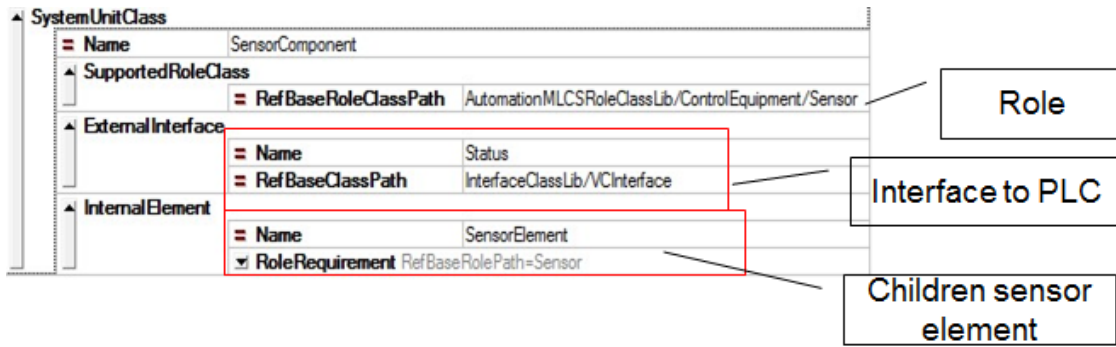


Figure 3-10 Sensor Component represented in VMCDM

3.2.5.3. Non-control Component

Non-control components refer to static objects, a fence or a floor for instance, which are only composed of geometry data. A non-control component has no mechanical behaviours and obviously it has no control behaviour either. In VMCDM ontology, a non-control component is composed of at least one static element. Its attribute “SupportedRole” for specifying the role should also be set to as either a normative role or a user-defined role. An example is given in Figure 3-11.

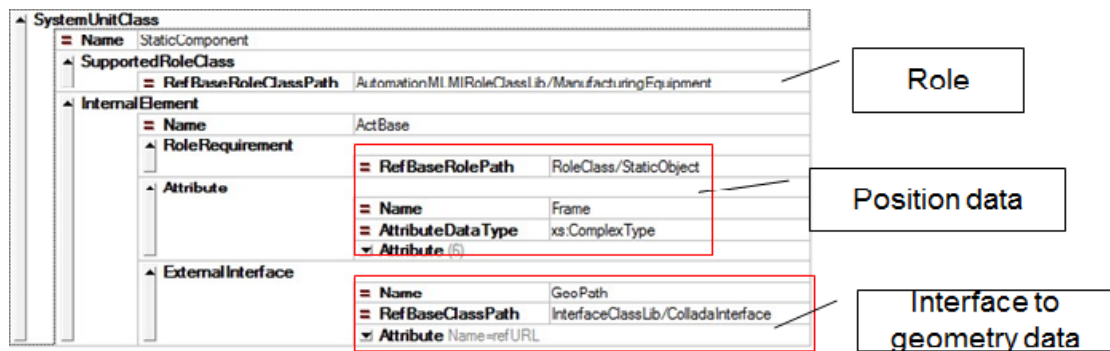


Figure 3-11 SystemUnitClass for Non-control Component

3.3. Mapping Component-based Virtual Models to VMCDM

This section describes the approach to mapping the component-based virtual models described in XML into the proposed VMCDM described in AutomationML. The data model mapping is required to achieve efficient data exchange based on common data models.

3.3.1. Overall process

The process of exchanging data based on a neutral data model between different engineering tools can be divided into four steps, as illustrated in Figure 3-12. Firstly, the

source engineering data is exported by a data exporter from a source engineering tool. Subsequently, the engineering data is mapped from the proprietary tool-specific data model into the common data model. The following two steps mirror the previous two steps. In step 3, the engineering data will be mapped from common data model into the proprietary data model of the target engineering tool, and then the target engineering tool imports the data using a data importer in step 4. In some particular circumstances, the step 2 and/or step 3 might not be necessary if the involved engineering tools directly support the common data model.

In the case of this research, data model mapping is required since few engineering tools has been found supporting AutomationML. This research is mainly focused on the step 2, namely mapping from component-based virtual models to the VMCMD.

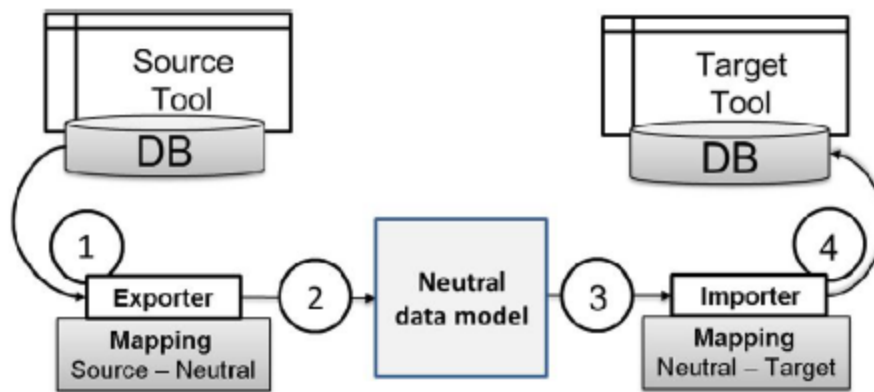


Figure 3-12 Process of Data Exchange based on Neutral data model

As analysed in the previous section, AutomationML can be seen as a semi-formal description framework which is between XML schema and ontology. Given the fact that the tool-specific data models are stored as XML files and AutomationML is also XML-based, the approach to mapping the data models in this research learns from the approaches to mapping between XML files and approaches to mapping between ontologies.

3.3.1.1. Existing approaches to data mapping

For data mapping between different XML files, Extensible Stylesheet Language Transformations (XSLT) has been widely adopted. XSLT is a language for transforming XML documents into other XML documents or other objects such as HTML for web pages, plain text, and so on [117]. Data mapping from XML to ontologies

has been widely researched and many effective approaches have been proposed [103, 118]. Also, the ontology mapping process has been defined and researched by many literatures [119, 120].

Through the study of the existing ontology-related mapping approaches from the literature, the process of ontology mapping is summarised in Figure 3-13. The ontology mapping process normally begins with normalisation which focuses on raising all data to be mapped onto the same representation level. The objective of normalisation is to cope with syntactical, structural and language heterogeneity between source data and target data. In normalisation, both ontologies must be normalised into the uniform representation thus eliminating syntax differences and make semantic difference more apparent [121]. The second step – semantic bridging, is responsible for establishing correspondence between entities from the source and target ontology. The entities involved in the bridging include concepts, relations and attributes. In the following step – execution, the instances of source model are actually transformed into corresponding target instances based on the semantic mapping rules. An additional post-processing step can be potentially needed to check and improve the transformation result. The result of post-processing step can be used as feedback to further improve the whole mapping process.

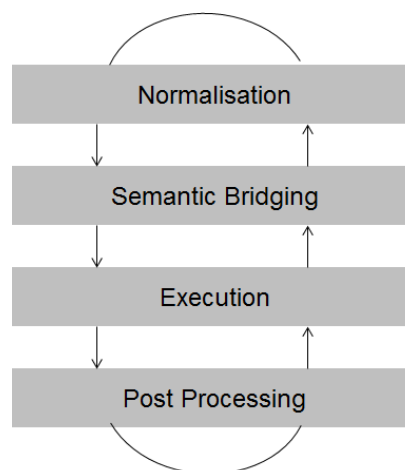


Figure 3-13 Ontology mapping process [119]

3.3.1.2. Approaches to mapping Component-based models to VMCDM

Based on the existing relevant approaches [120, 122, 123], the approach to mapping component-based virtual models described in XML to the VMCDM described in AutomationML is designed by the author and is illustrated in Figure 3-14.

In the case of this research, XML is regarded as the data format for normalisation, considering that XML is widely supported by most engineering tools and also the target data format – AutomationML is also XML-based. Therefore, before establishing the semantic bridging, the XML schema of the source data model needs to be generated first if it is not available. The schema of the target data model can be generated according to the VMCDM proposed in the section 3.2. This step will not be elaborated since a XML schema can be easily created or generated from the existing XML file.

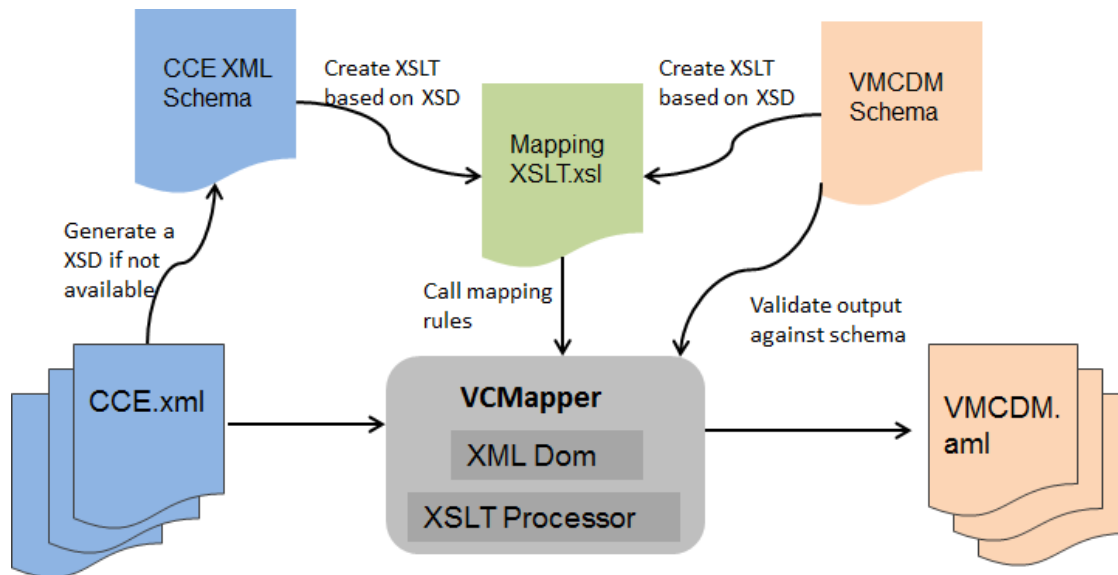


Figure 3-14 Process of mapping from CCE virtual models to VMCDM

The following section is focused on the required semantic bridging to transform component-based virtual models to the VMCDM. The semantic bridging process is implemented using XSLT. The execution of the XSLT is performed by the engineering tool which is designed by the author and described in section 3.6. It should be also notified that semantic bridging only covers mapping of the hierarchical data of the virtual models while the control logic mapping and geometry mapping are not involved. This is due to the fact that standard transformation tools for geometry data mapping are normally available while the control logic mapping will be realised by the direct deployment approach which is elaborated separately in section 3.5.

3.3.2. Semantic Bridging

Process of semantic bridging can further divided into a number of sub-steps which were presented by Maedeche [119]. Through analysing the data models involved in this research

and referring to the semantic mapping process proposed by Maedeche, the approach to semantic mapping is built from three dimensions those are concept mapping, property mapping and mapping functions, as illustrated in Figure 3-15.

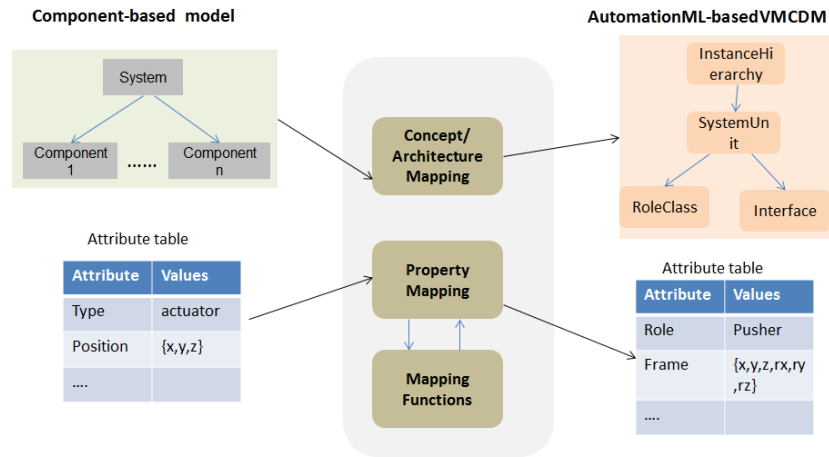


Figure 3-15 Semantic bridging for mapping CCE to VMCDM

3.3.2.1. Concept mapping

Concept mapping is to identify the pairs of concepts to be bridged. According to the mapping relation between a source concept and its corresponding target concept, three distinct cases are identified in this research.

The first case refers to that a source concept corresponds to a target concept. This implies that a source instance will give rise to just one instance of the target concepts. In this case, the main job is to map the data architecture of source concept to that of the corresponding target concept. In our case, the mapping is realised in a top-down manner. Due to the adoption of object-oriented architecture of both the component-based virtual models and the VMCDM, the concept system and component can be directly mapped to the “InstanceHierarchy” and “SystemUnitClass” of VMCDM. By further decomposing a component, the information for integrating its different aspects, are respectively mapped to corresponding concepts of VMCDM. The information of integrating geometry data is mapped to “Static element” and the mechanical behaviours for integrating kinematic data and control logic are mapped to “Actuator elements” or “Sensor elements”. The state behaviour of an actuator or a sensor in the component-based approach is mapped to the “InternalElement” with the role of “DigitalDrive”.

In the second case, some target concepts are completely new concepts for the source concept. In this research, these concepts refer to the “Role class” and “Interface class”. In the component-based approach, no concepts corresponding to “role” and “interface” have been defined. In this case, these required classes that are predefined in VMCDM will be directly included in target data models.

The last case refers to that a source concept does not have a specific counterpart target concept. In the case of this research, the control logic of component-based virtual models is normally modelled as State Transition Diagrams. Obviously, it is not applicable to the VMCDM in which the control logic is modelled as real control code. In the component based approach, the control behaviours of an actuator component are composed of state, transitions and conditions of transitions. However, in the VMCDM, only the “state” is used as an internal element with role “DigitalDrive” . The transitions and conditions will be translated into control software and are not included in the VMCDM. Therefore, in the concept mapping, the concepts of “transition” and “condition” are discarded.

An example of the architecture mapping from the source model- component-based virtual models to the target common model – VMCDM is demonstrated in Figure 3-16.

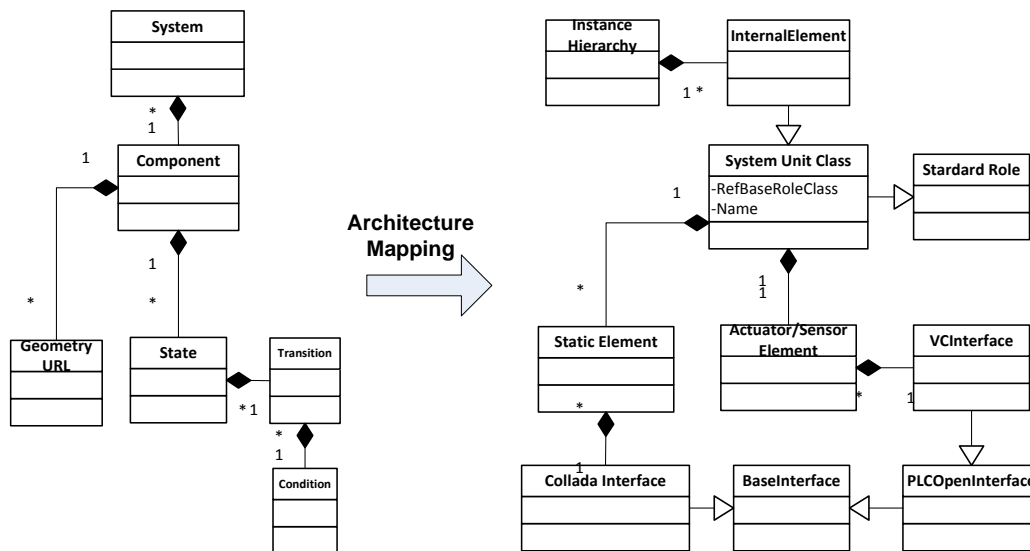


Figure 3-16 Architecture mapping described in UML

3.3.2.2. Property mapping

Property mapping is a complementary step to the concept mapping. Its main objective is to map the terminologies and values of attributes. In this research, for a static source attribute

for which a corresponding fixed target attribute can be found, the mapping is realised in a XSLT file. In the XSLT file, such kind of attributes will be directly translated into the corresponding target attributes.

On the other hand, for some attributes, their values can vary depending on the parent objects. For instance, the attribute “RoleRequirement” of actuator components can be various, such as pusher, conveyor, lift, etc. In the component-based virtual models, they are just generally labelled as “actuator”. However, in the VMCDM, in order to enhance the semantic, each of such actuator components should have its specific role. For this kind of attributes, a mapping table which maps the GUID of an actuator component to its corresponding role needs to be created. This mapping table will be scanned by the XSLT during the execution phase of ontology mapping. Apart from attribute “RoleRequirement”, the mapping between a component and its related PLCOpenInterfaces is also required. The mapping table for this purpose is created by the direct deployment engineering tools which will be elaborated in section 3.5.

3.3.2.3. Functions for transformation

The aim of creating functions for transformation is to associate the data mapping, in a way that specific properties of source instances to be translated into counterpart properties of target instances. For some attributes of source instances, their values, instead of being directly mapped into the target counterparts, need to be processed by specific algorithms to translate to the target values. For this purpose, pre-defined functions, which implement the respective transformation algorithms, needs be defined. These functions will be called in XSLT during the execution phase, as shown in Figure 3-17.

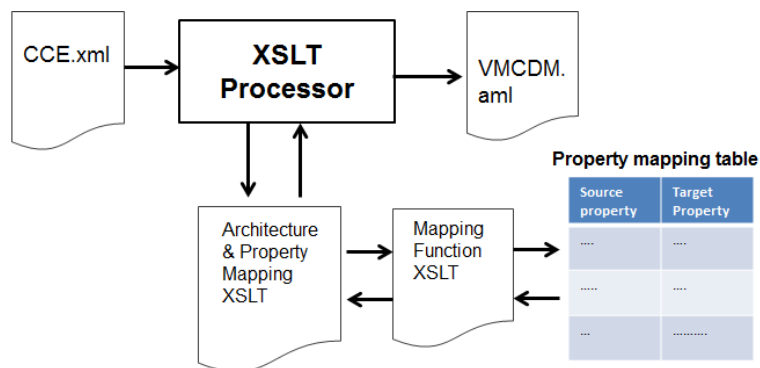


Figure 3-17 Data model mapping by calling function in XSLT

In this research, functions for the following purposes are required:

- Transformation of the positions of 3D geometry: The position of a geometry part relative to its parent component or the position of a component relative to its parent system can be described using various approaches. AutomationML adopts the concept “Frame” as the uniform description of position information. In the component-based engineering tool, the relative position of an object is represented by the concept “Link point”. A convert algorithm is required to transform a “Link point” into a “Frame”. Hence, a function “ToFrame” which implements this transformation is defined.
- Mapping table scanning: mapping tables, which have been defined in the property mapping for dynamically mapping different attributes, are scanned by the XSLT in order to retrieve the corresponding target attribute of a source attribute. Functions for retrieving the following information are defined in this research: attributes “RoleRequirement” of components and attributes “VCinterface” for actuator and sensor components. The mapping between each component and its target attribute “RoleRequirement” is created in the property mapping phase. On the other hand, the mapping of “VCinterface” for each component instance of a system is automatically generated in the process of directly deploying the control software. The direct deployment approach is presented in section 3.5.

The required functions for transformation are summarised in Table 3-3.

Table 3-3 required functions for ontology mapping

Name	Description	Parameters	Return value
ToFrame	Transforms position from “LinkPoint” to “Frame”	Position data of a source instance	Frame: STRUCT
GetRole	Get the role of a component	ComponentID: String	Role: String
GetVCInterface	Get the VCinterface of a component	ComponentID: String	VCInterface: String

3.4. Deployable Control Software Architecture

In order to facilitate the implementation of the proposed direct deployment approach, a new control software architecture, which is developed by the ASG, is adopted in this research. The overall architecture of control software including the HMI is depicted in Figure 3-18. The PLC software on the left side and the HMI on the right side are actually two independent systems and communicate with each other through Ethernet during runtime.

The following two subsections will respectively describe the PLC control software and the HMI, regarding the functions of their components and how the pertinent components cooperate with each other.

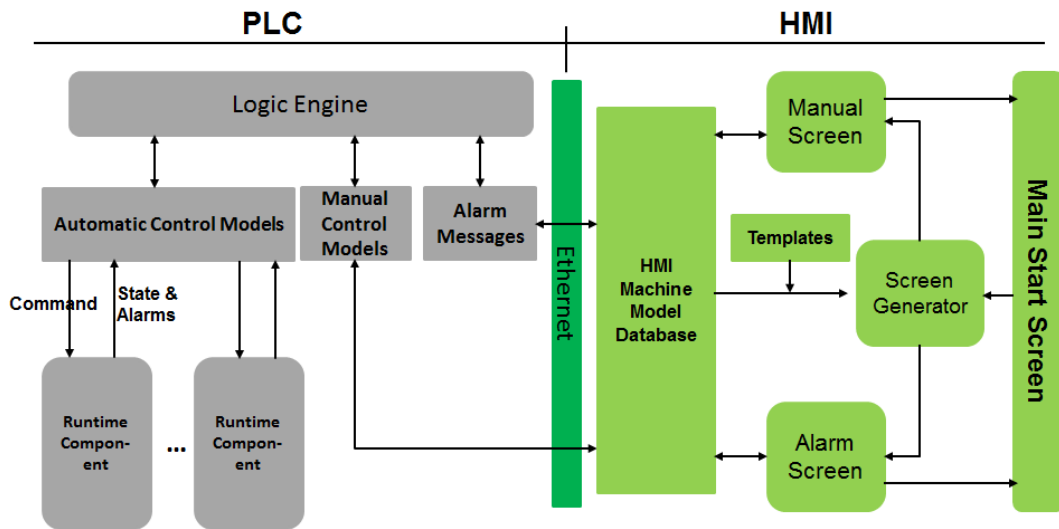


Figure 3-18 Overall control system architecture

3.4.1. PLC control system

The PLC control system architecture is depicted in Figure 3-19 with more details. The system is composed of Control Data Models, Runtime Components, and Logic Engine, all of which are described respectively in the following subsections.

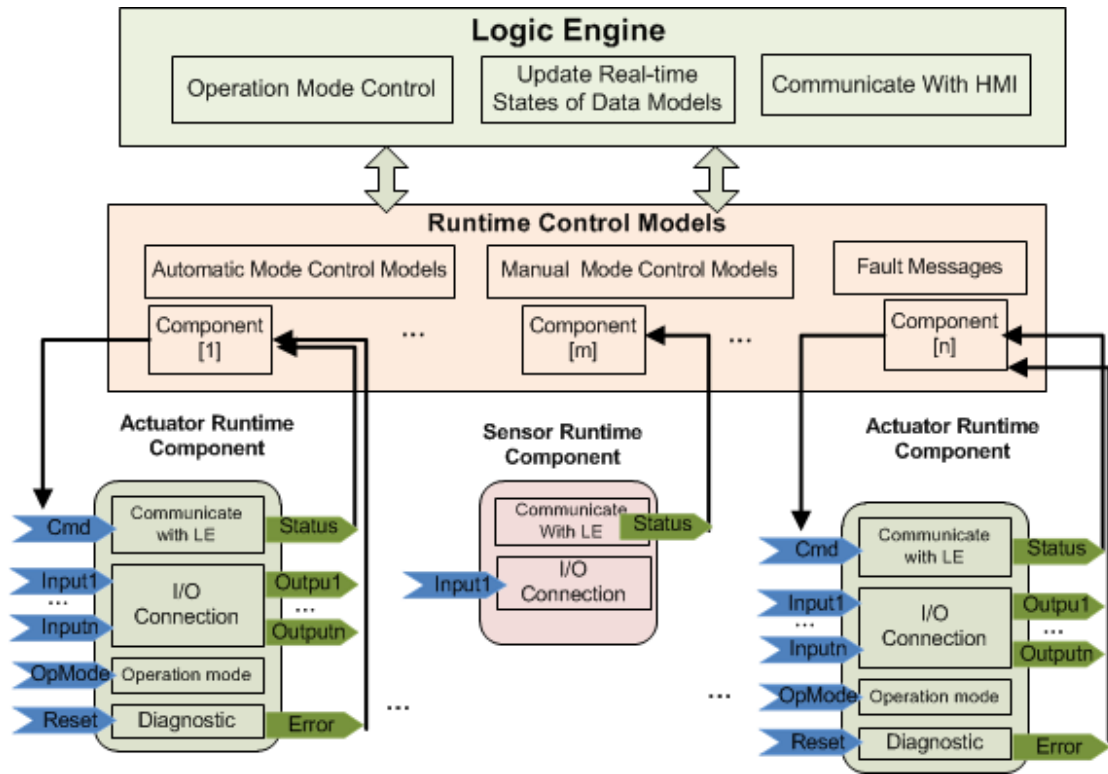


Figure 3-19 Component-based PLC control system architecture

3.4.1.1. Runtime Component

Runtime Component is a pre-validated and ready to use standard resource specific function block. Runtime Component represents an actuator component or a sensor component of a machine cell in a PLC runtime environment. It is embedded with the control behaviour of a family of actuators and sensors with integrated diagnostic, and thus it is developed once and stored in a Runtime Component Library for future reuse.

All the events and faults of a Runtime Component are communicated to the calling instance, i.e. the Logic Engine. Runtime Component is directly deployable in a PLC program and is interfaced via direct parameterisation, as shown in Figure 3-19. The following parameters are provided for the interfacing of the Runtime Component:

- **State Command** (cmd) is an input from the Logic Engine. It dictates component to move to a specific state.
- **Operation Mode** (OpMode) controls the machine operating mode; such as automatic, manual or dry-run.
- **State Message** (status) is to provide a feedback signal to the Logic Engine to update the working state of the corresponding Actuator Component.

- **Fault** is the output of the integrated fault diagnostic part of the Runtime Component. This output is communicated to the fault management block of the PLC program for a further necessary action as required.
- **Reset** is an input from the fault management block to acknowledge and reset the current fault after the operator has taken maintenance action.
- **Process Digital Inputs/Outputs** (Input/Output 1...n) are input/output signals directly connected to the hardware to interface sensors and actuators respectively.

3.4.1.2. Runtime Control Models

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

As shown in Figure 3-19, the Runtime Control Models can be further classified into the following types:

Auto-mode control models: contains all the logic information for the auto-mode control of the desired machine. These control models are generated by automatically translating the control logic of the component-based virtual models. During the runtime, auto-mode control models are used as the interfaces of the control software to communicate with the Runtime Components.

Manual-mode control models: contains the data for generating the HMI control screens for all the actuator components. These control models are also generated by analysing and transforming the state behaviours of the virtual components. During the runtime, these models are accessed by the HMI screen generator to generate the HMI and to send commands to corresponding components to control the machine.

Fault-message models: are used to store error messages which are read by the HMI to display on the screen. Error messages from runtime components are sent to the corresponding Control Models and then written into the Fault message models by the Logic Engine which is described below.

3.4.1.3. Logic Engine

Logic Engine is a principal component and works as a system orchestrator to execute the manufacturing operations. It has a number of functional components to perform various system operations such as operating mode handling, device management, fault management, and communicating with HMI.

During runtime, the Logic Engine interacts with all the runtime components and HMI through the control data models. On one hand, the Logic Engine keeps scanning state behaviour of each component saved in the control data model to decide the next working state of each component and updates the state command of the respective component. Under the automatic mode, the next working state of a component is decided when the respective transition conditions are satisfied, while under the manual mode, it is decided by the command received from the HMI. On the other hand, after receiving the state command through its connected control data model, the corresponding actuator runtime component moves to the respective state and updates its current working state into the connected control data model.

3.4.2. HMI

The HMI system architecture is composed of three system components: Screen Generator, Alarm Handler & Visualiser, and Machine Monitoring. The HMI application dynamically generates screens for manual mode control and communicates with the PLC control models during runtime. The operator screens for manual mode control are system specific, and thus unique for each manufacturing cell. Typically, rows of two pushbuttons are provided on the manual screens for each actuator. Via these pushbuttons machine operator can control the machine by driving the actuators between their home and work positions. The screen generator communicates with the runtime Control Models of the PLC control software, analyses the system components and generates the manual rows for each actuator according to the number of positions of actuator. Also, the Alarm Handler is responsible to report error and warning messages to the operator.

It can be observed that the bases on which the HMI application dynamically generates the operator screens, communicates with the machine and displays the error messages, are the runtime control models of the PLC control software. This research is mainly focused on

directly deploying the PLC control software of the depicted system. Hence, the development of the components of the HMI will not be elaborated in this dissertation.

3.5. Direct Deployment of Control Software

This section presents the approach proposed in this research to directly deploying the control software for a physical assembly system based on its validated component-based virtual model.

First, an overview of the process of realising the proposed approach is provided. Consequently, the key steps of this process are then elaborated one by one.

3.5.1. Overview

As stated before, one of the main objectives of this research is to automatically generate the complete control software based on the component-based virtual models of the desired systems. In order to achieve this objective, the first important step is to clarify that which kind of data are reusable or static and which kind of data need to be dynamically generated. Based on the classifications, the process of automating the control software development process then can be specified.

3.5.1.1. Decomposition of the deployable control software architecture

Prior to designing the approach to directly deploying the desired control software, the proposed control software architecture is decomposed and all the constituent elements are classified as following:

- **Reusable elements:** refer to the runtime components, the logic engine and derived data types for describing relevant virtual control models. These elements are developed during component build phase and are directly reused during the control system development phase. When the control software for a specific system is developed, the related runtime components are selected from the runtime component library and used in a black-box manner.
- **Platform-specific common information:** refers to PLC platform-specific information which is required to combine with the above mentioned data together to compose the source code of completely executable control software. These typically include header information of the overall control software and other different blocks.

The most significant characteristic of this kind of information is that, for a specific PLC platform, they are identical for control software of any automation systems. Hence, this information is refined and stored as templates.

- **Dynamic elements:** refers to the logic depository and programs for resource components. The logic depository is dynamically populated with runtime control models which are created by translating the control behaviour of component-based models. Additionally, for certain PLC platforms, some platform specific elements might need to be generated dynamically. For Siemens Step 7, for instance, an instance data block for each runtime component called by programs are required to be created dynamically.

3.5.1.2. Overall process of direct deployment

Based on the classification of the elements of the deployable control software, the process of the direct deployment approach is designed and outlined in Figure 3-20.

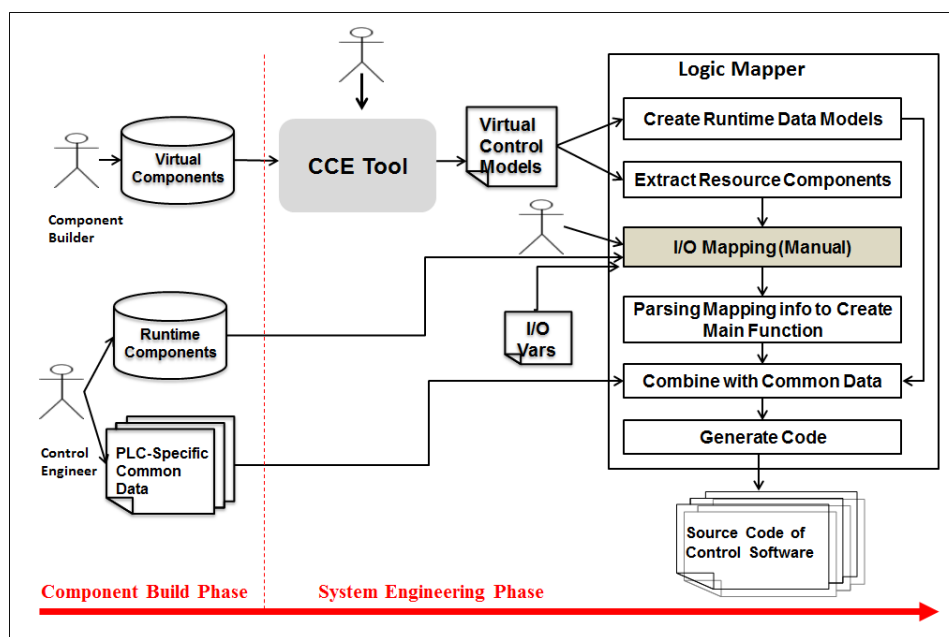


Figure 3-20 Component-based PLC control software engineering

Corresponding to the component-based approach, the process of control software development can be divided into component building phase and system building phase. As shown in Figure 3-20, all the reusable components and PLC-specific common information are developed during the component build phase. The main task of system engineering phase

is to automatically generate the dynamic data and combine them with reusable and common data to generate the complete control software.

In the following subsections, the process of developing reusable static elements during component building phase and the process of automatically generating dynamic elements and the final complete control software during system engineering phase are presented in details.

3.5.2. Development of Reusable Static Data

As stated in the previous section, static elements contained in the deployable control software include runtime components, derived data types and PLC-specific common information. This section describes these elements in details and presents the process of their respective development. By reusing these pre-developed static data, the control software can be deployed in a more flexible and reconfigurable manner.

3.5.2.1. Reusable Runtime Components

Runtime components are pre-developed and pre-validated by control engineers separately during component building phase. The development of Runtime Components relies on the expertise of control engineers and is not the focus of this research. Instead of development, how to reuse the reusable runtime components to generate complete control software during system engineering phase is one of the key aspects of the proposed direct deployment approach. Therefore, the process of developing Runtime component is not elaborated in this dissertation.

3.5.2.2. Data Types for Component-Based Control

In the component-based virtual models, the state behaviour of an actuator component is represented in STD which is not a PLC understandable language. The STDs can be easily transformed into SFC PLC function blocks in reference to existing approaches. However, programs generated in this way only contain basic control functionalities while other functionalities required for industrial application, such as diagnostics, are missing. In this research, reusable runtime components with all required functionalities have been developed during the component building phase. The actual control logic of each component will be translated into Runtime Control Models which communicate with corresponding Runtime Components and the Logic Engine.

The control logic of component-based virtual models cannot be described using the standard elementary data types of IEC61131-3; therefore, derived data types are required to be defined to describe component-based control data models in a PLC interpretable manner. Four derived data types for describing the Runtime Control Models are defined: component, state, transition and condition. In order to facilitate the automatic generation of control program for manual control mode, two more derived data types for describing fault messages and HMI rows are also defined.

All the derived data types defined are outlined in Table 3-4.

Table 3-4 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

3.5.2.3. PLC Platform-Specific Common Information

The development of the common information of a specific platform is based on a reverse engineering process, as illustrated in Figure 3-21. Firstly, the source code of existing control software project is exported from a specific PLC programming tool. Secondly, through analysing the source code file(s), data of the source code can be classified into platform-specific common information and project-specific information. The platform-specific common information will then be decomposed into different element and saved in the database as structured information. These common information will be reused during control software deployment phase to generate the control software for new systems.

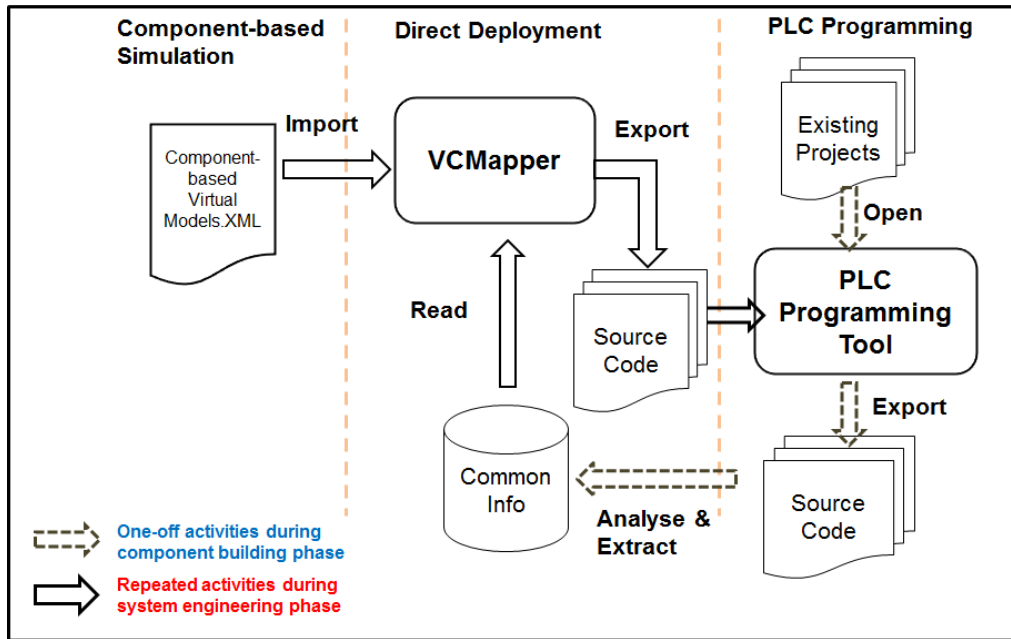


Figure 3-21 Reverse engineering process of direct deployment

The current PLC programming engineering tools all support programming in graphic languages. However, for importing and exporting of programs, the source codes are all described in textual languages. By analysing the source code of program for a specific platform, the common information can be extracted. The most significant characteristic of this platform-specific common information is that it is included by the source code of any project and is identical for any project. To be more specific, these types of information mainly include:

Header information of projects: header information of a control software project. It is also worth noticing that for some particular platform, in S7 for instance, there is no uniform project header information as a S7 project is exported into multiple independent text files instead of one single project file.

Header information of blocks: The blocks here refer to different objects which compose a control software project. According to the structure of IEC61131-3, this kind of blocks can be data types, program organisation units (POU), variable table, instances, configurations, etc. PLCOpenXML describes these blocks as structured XML elements, which makes the common information easy to manage and reuse. Some PLC programming tools may use platform-specific constituent blocks. Some platforms might export the source code as unstructured data. In these cases, the common information of each block need to be managed separately using specific ways according to the data structures of the blocks.

The header data of a shared data block which contains the Runtime Control Models in S7 platform is taken as an example and illustrated in Figure 3-22. The source code of S7 data blocks are saved as unstructured plain-text files and the header also contains some variables the value of which are changeable for different automation systems. In this case, these changeable variables are represented using reserved key words during the template development phase and they will be replaced with their actual values during the system engineering phase.

In this data block, the changeable upper bounds of the arrays are represented using reserved words. As shown in Figure 3-22, the words in red are common information while the words in blue are changeable and will be replaced by specific values during the direct deployment process.

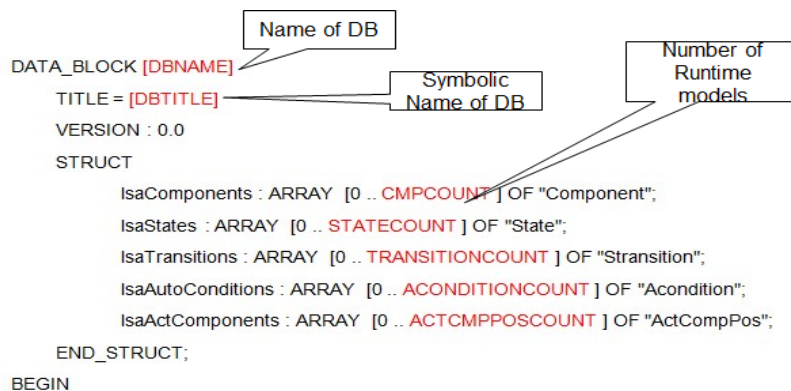


Figure 3-22 Template of the shared DB header in SIMATIC Step7

In addition to the PLC-specific common information, some end-users also provide, vendor specific templates which define the templates for both hardware configuration and software development. In our case, only the software-related templates are involved. Unlike the domain-specific templates and the platform-specific templates which mainly provide reusable static information, the software templates actually define regulations for software architecture, memory mapping, naming conventions, etc. For example, some templates define architecture of the main program, memory address scopes of specific function blocks, and prefixes and suffixes of the symbolic names of variables involved.

The development of the vendor-specific templates is not the focus of this research. However, achieving direct deployment of control software according to vendor specific templates is a key requirement for industrial application, and therefore it is also one objective of this research. Hence, functions for implementing the coding conventions defined by related

vendor specific templates are developed and contained in the proposed direct deployment engineering tool.

3.5.3. Dynamic Generation of Runtime Control Models

Differing from the Runtime Components and the static common information, which are pre-developed in the component engineering phase, the Runtime Control Modes are generated dynamically during the system engineering phase by translating control logic of component-based virtual models into PLC interpretable data models. In order to generate the control software with functions which meet the requirements of industrial application, Runtime Control Models for both automatic-mode control and manual-mode control are automatically generated.

3.5.3.1. Automatic-Mode Control Model Generation

Control models for automatic-mode are generated by translating the control behaviours of components and then describing them as instances of the predefined derived data types. The process of auto-mode control model generation is outlined in Figure 3-23.

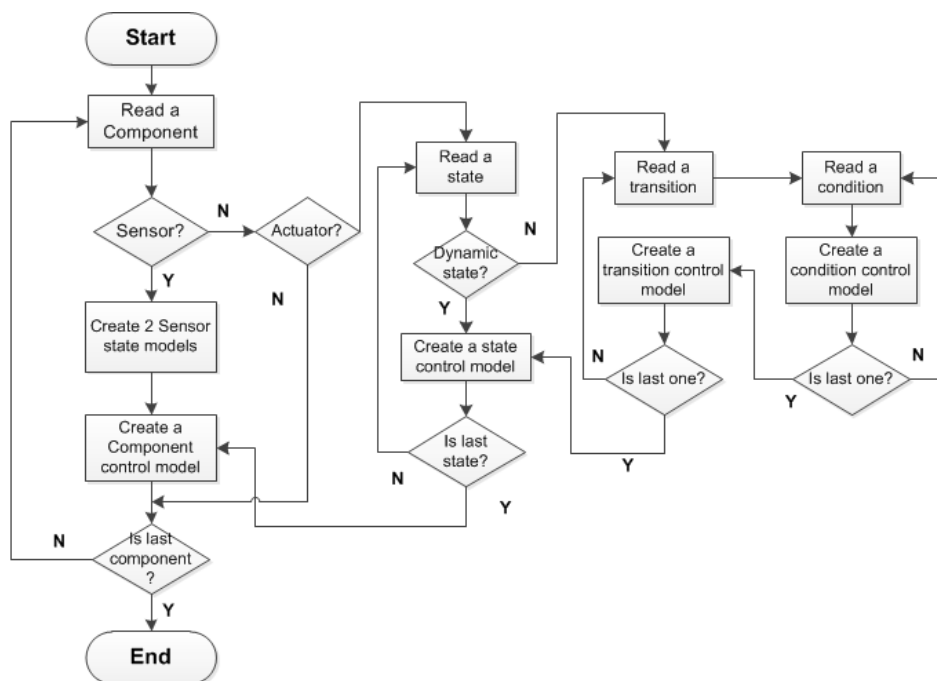


Figure 3-23 Workflow of automatic-mode control model generation

During the translation, the components of a system are scanned and translated to populate the corresponding control model described using the predefined derived data type – “Component”. For each component, its states are then translated to populate the

corresponding control models described in the derived data type “State”. Further, all the transitions and the constituent condition groups are then translated into the corresponding control models. Additionally, for a dynamic state which has interlock conditions, the conditions are also translated into control models in the same way of translating transition conditions.

According to the deployable control software architecture illustrated in Figure 3-18, the control model components are used as the interface for the communication between corresponding Runtime Components and the Logic Engine. Therefore, this section mainly takes “component” as an example to explain how the virtual models are described as PLC-interpretable data models. The translation of “state”, “transition” and “conditions” will not be elaborated. A control model component is represented as:

$$\text{RCCOMP} = \{\text{ID, name, type, index, stcount, swid, scid, fault}\}$$

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

- ID: the unique ID of the component.
- Name: the name of the component.
- Type: the type of the component which can be “actuator”, “sensor” or “process”.
- Index: the index number of its first state. The states of all the components of a system will be 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 continuously. Therefore, the index number of the first state can be used as the entrance point for data accessing.
- Stcount: the number of the states a component has.
- Fault: the index ID of the fault message.

The attributes “swid” and “scid” have changeable values. The attribute “swid” refers to the ID of the current working state. It is used to collect the current working state 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 runtime control models of states, transitions and conditions are monitored by the Logic Engine during the run time. According to the current working state of each component, the relevant runtime control models will be updated by the Logic Engine. Real-time state commands will be sent to corresponding Runtime Components to drive the physical components.

As mentioned above, the attributes “swid” and “scid” will be updated with real-time data at the run time. Therefore, they are set to respective initial values, during the control model generation process. During the control model generation process, the main task is to set the values of the static attributes of all the control models by encoding the corresponding items of the virtual control models.

3.5.3.2. Manual-Mode Control Model Generation

Manual-mode control models are used to generate the corresponding rows of the HMI control panel for manually controlling actuator components. Manual mode control model(s) of a component can be generated by translating its state behaviour. Unlike auto-mode control models which cover states, transitions and conditions, the manual mode control models are generated only according to the dynamic states of the related component. The workflow of generating manual mode control models of an actuator component is illustrated in Figure 3-24.

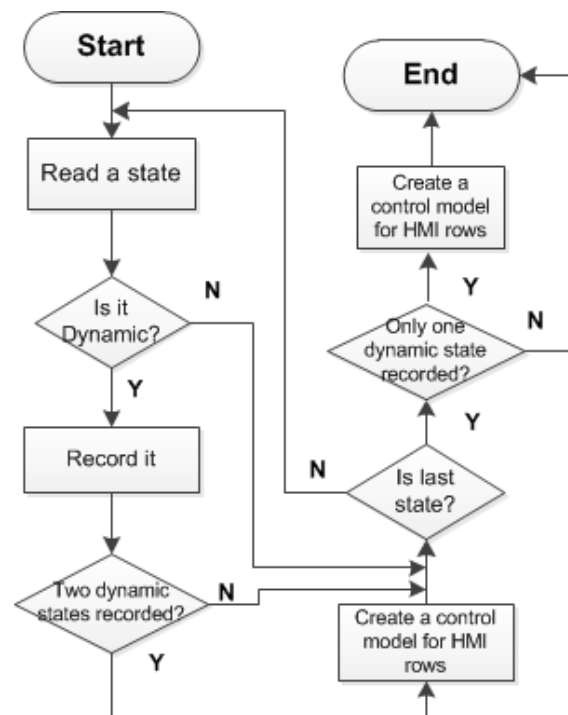


Figure 3-24 Workflow of manual mode control model generation

A manual mode control model will be represented as a row of the HMI, which controls the movement between two positions. For an actuator component, the relation between the number of its manual mode control models (represented as N_{mm}) and the number of its dynamic states (represented as N_{ds}) is:

$$N_{mm} = N_{ds} - 1 \text{ (if } N_{ds} > 1)$$

$$N_{mm} = N_{ds} \text{ (if } N_{ds} = 1)$$

An example of two actuator components – a pusher and a gantry, is shown in Figure 3-25. The control behaviour of the actuator “Pusher” with four states, two of which are dynamic states, are translated into one manual mode control model and will further be represented as one row on the HMI. The component “Gantry” with three dynamic states corresponds to two HMI rows.

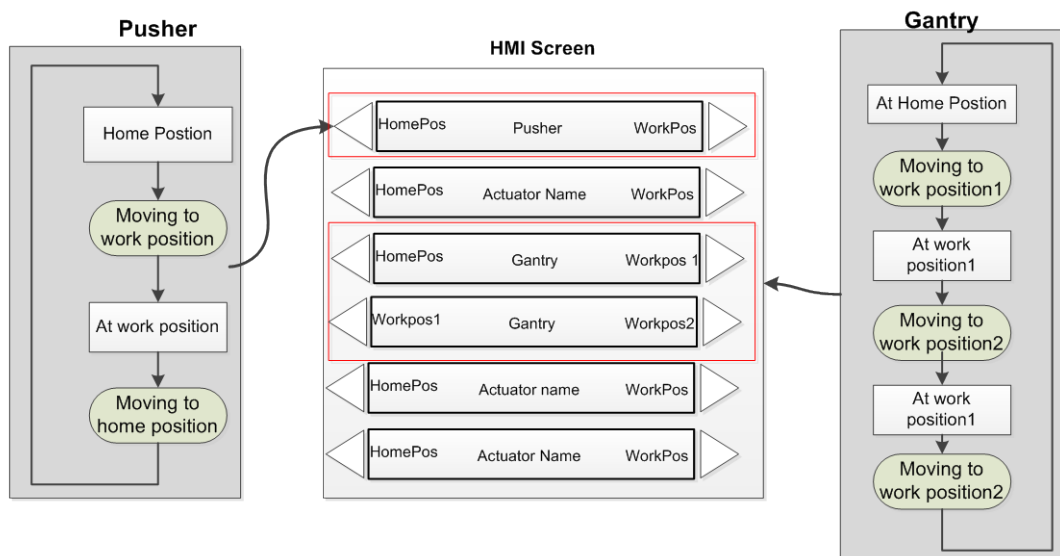


Figure 3-25 State behaviours and the corresponding HMI control panel

3.5.4. Dynamic Generation of Logic Depository

The Logic Depository is generated by automatically organising all the runtime control models and describing them as structured data sets. In this research, according to the IEC61131-3 architecture based on which the control software are generated, the Logic Depository is represented as different arrays of the runtime control models. The size of each array is determined by the number of the respective runtime control models contained in a system. For example, the array of runtime control model “Component” for a system which composed of ten actuator/sensor components can be declared as:

IsaComponent: ARRAY[0..9] of “Component”

Where “IsaComponent ” is the name of the array and “Component” is the name of the derived data type for describing actuator components.

In this research, in order to support the functionalities for automatic mode control and manual mode control, the logic depository consists of six arrays of control models described in the derived data types defined in section 3.5.2.2. The declared arrays are then populated with the respective runtime control data models which are generated according to the process described in the section 3.5.3.

3.5.5. I/O Mapping for Actuator/Sensor Components

I/O mapping in this research refers to mapping each actuator or sensor component of a system to its corresponding runtime component and pertinent I/O variables. This is the only step which has to be performed manually during the system engineering phase. It is essential for generating the real PLC code based on the virtual models as I/O connections are not involved in the CCE simulation.

A user interface is required to facilitate the I/O mapping. The reusable Runtime Components and the related I/O variables are imported into the engineering tool for I/O mapping and listed out on the user interface. On the other hand, all the actuator components and sensor components which need to be connected with I/Os are automatically selected and listed out on the user interface. The task of consequent I/O mapping is to select every actuator/sensor component one by one and its corresponding runtime component and then add pertinent I/Os to the runtime component. The mapping information between a component, the related runtime component and I/O variables will be used to generate the program for controlling this component. The user interface implemented for I/O mapping is described in section 4.2.2.4.

3.5.6. Dynamic Generation of Programs for Actuators and Sensors

Program generation refers to (1) parsing the I/O mapping data from the I/O mapping step to generate program organisation units, and (2) generating other required platform-specific source code. The number of the program POU's to be generated depends on the requirements of the adopted vendor-specific template.

Each program POU is created by calling one of the pre-defined runtime components which can be further classified as runtime actuator components, runtime sensor components and the Logic Engine. Of these runtime components, the Logic Engine is only called once within a

project and the times of calling runtime actuator or sensor components are corresponding to the number of actuator and sensor components contained in the virtual models.

As mentioned before, the logic engine is actually a function without any I/O parameters. For a runtime actuator/sensor component, I/O mapping for its parameters which are required to connect to I/O variables has been done during the I/O mapping phase. However, the communication parameters that should be connected to the corresponding runtime control models need to be mapped automatically.

The workflow of program generation is outlined in Figure 3-26. Firstly, the I/O mapping data for all the actuator components and sensor components are processed. According the identification (ID) of each component included in the I/O mapping data, its 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. If it is an actuator component, the respective I/O parameters for reporting working state (swid), sending state command (scid), resetting (reset) and reporting error (alarm) are mapped. For a sensor component, only one parameter for reporting status needs to be mapped. Lastly, the Logic Engine function is called automatically. The program generated in this way is well-structured since every runtime component, except the logic engine, contained in the control software corresponds to either an actuator component or a sensor component of the to-be system.

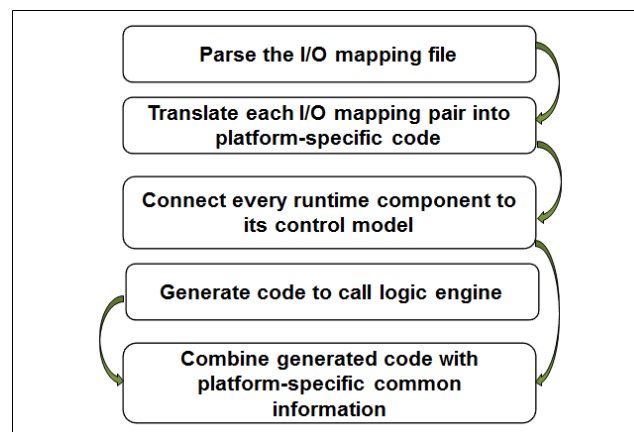


Figure 3-26 Process of generating program(s)

In traditional manual programming approaches, the declaration of a FB/FC instance is done automatically by the used programming tool. Correspondingly, in the direct deployment approach, the code for declaring program instances is generated automatically.

Apart from the declaration code, for some platform, some additional blocks might need to be created for calling a runtime component. For example, in Step 7, an instance data block needs to be created when a function block is called and this instance data block is created automatically by parsing all the parameters of the called function block. Correspondingly, the direct deployment solution also provides the functions of automatically generating platform-specific blocks.

3.5.7. Generation and Output of Complete Control Code

As the last step of the direct deployment solution, source code generation and output refers to the process of combining all the dynamic-generated items together with the reusable and common information to generate the desired source code and finally exporting the code into file(s) of platform-specific data format(s).

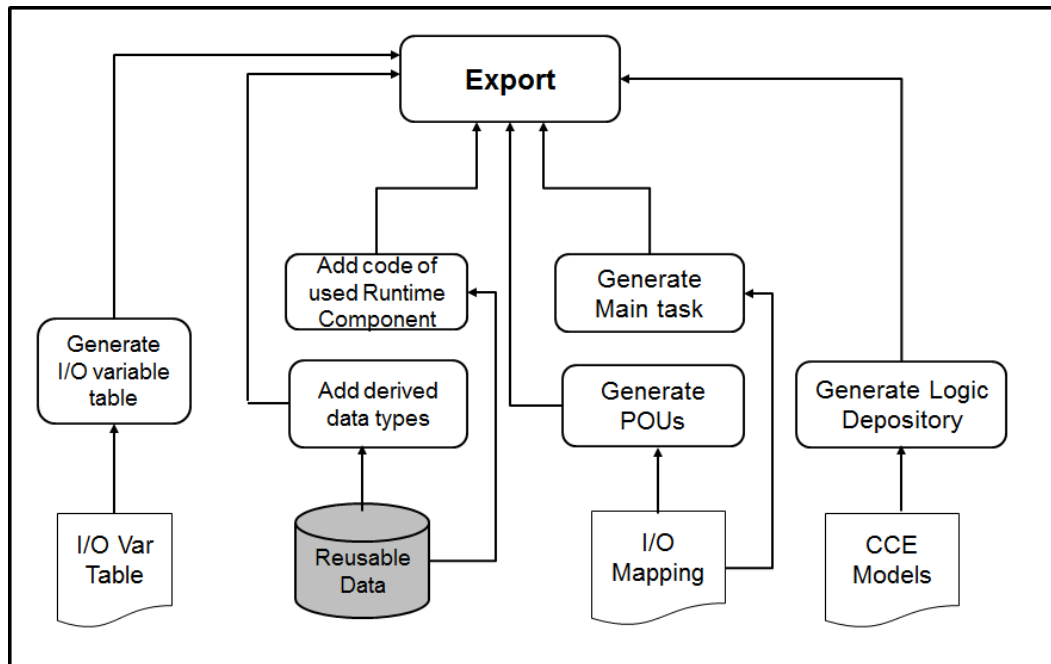


Figure 3-27 Source code integration and export

The process of generating and exporting relevant code are performed in a set of sub-steps. Firstly, the variable table is created by fitting the I/O variables imported from external files into the platform-specific template. Secondly, the source code of logic depository, program POU's, the main task and other potentially required platform-specific blocks are then combined with its respective template. Thirdly, source codes of all the reusable models are combined with the related PLC-specific common information. The involved static information includes the derived data types, and the reusable runtime components which are

used in the I/O mapping. Consequently, all the source code will be fitted into the template of the project which contains the project-specific common information. Finally, source code of the desired project is exported into either one single file or multiple files of the required data formats.

3.6. Software for Data Mapping and Direct Deployment

This section presents the process of designing an engineering tool which implements the VC model mapping approach and the direct deployment approach proposed in this research. After an overview of the overall software architecture, the work flow and data flow for implementing the process of data model mapping and direct deployment are then presented.

3.6.1. Software Architecture Design

The main objective of developing the engineering tool, named VCMapper in this dissertation, is to facilitate the application of the proposed methodologies by automating required virtual model transformation process and control software deployment process. To achieve the goal, the engineering tool contains a user interface for performing the required mapping activities and processes all the data automatically by the background modules.

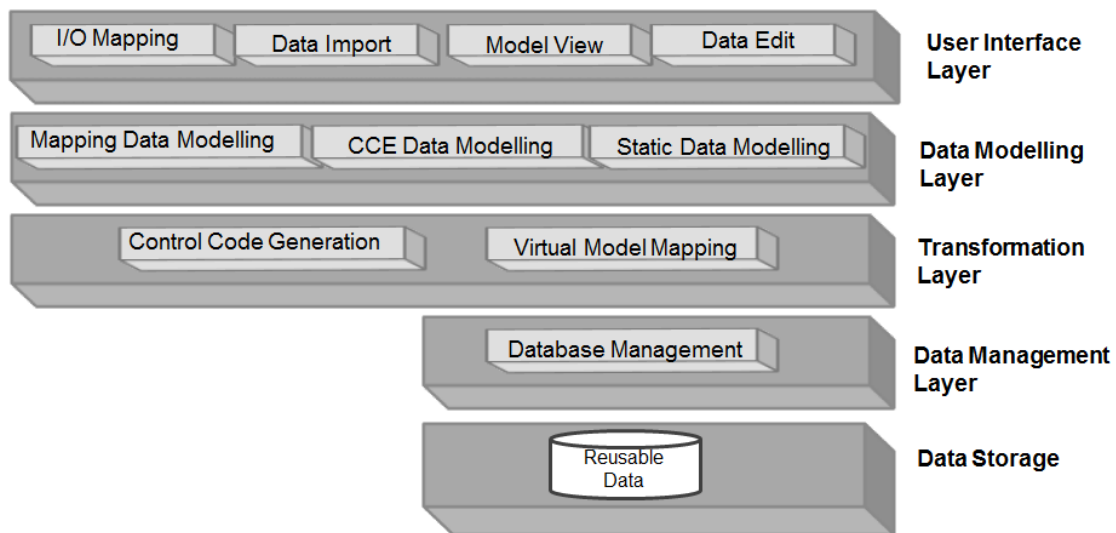


Figure 3-28 Architecture of the VCMapper

As outlined in Figure 3-28, the architecture of the engineering tool can be decomposed into four layers which are respectively responsible for different required functions. The modules of different layers exchange data with each other to enable the overall function of the tool.

User interface layer: the main function of the user interface layer is to enable I/O mapping. Additionally, it has the functions of importing required data from external files and displaying reusable data which are saved in the database.

Data modelling layer: includes three modules which are respectively used for (1) modelling the information of reusable runtime components saved in the database and passing the modelled data to the user interface layer, (2) parsing the component-based virtual models and (3) modelling the I/O mapping data.

Transformation layer: includes the core modules of the tool. One is to implement the data model mapping approach while the other module is to implement the direct deployment solution.

Data management layer: the function of the database management module is to provide generic functions for accessing the reusable data in the database. On the other hand, the output data export module aims at exporting the output data of code generation layer into files of required data formats.

3.6.2. System Design

Based on the architecture described above, the software is designed following the waterfall model of software engineering[124]. The software is implemented based on object-oriented programming paradigm. According to software design process, the data flow diagram should be first designed. Other key steps which include workflow design and interface design then take place.

3.6.2.1. Data Flow

The Data Flow Diagram (DFD) of this engineering tool is shown in Figure 3-29. The data flows can be further classified into reusable data flow and system-specific data flow.

As shown in Figure 3-29, the data flow related to the reusable data is illustrated in the right side of the DFD. The function of this data flow is to transfer the reusable and static data from storage files into the database and read the data from the database during runtime. Three modules are needed for realising the functions. The “UI-based Importing” module mainly provides a user interface for specifying the path of data storage files. The “Reusable Data Modelling” module reads data from storage file(s) and transfers the data into structured data.

The structured data is then passed to the “DBMS” module which writes the structured data into the database. The “DBMS” module also provides the function of retrieving data from the database.

Another six modules are involved in processing the system-specific data, as shown on the left side of the figure. The “CCE Modelling” module reads the data from storage files and creates the data models. The “I/O Mapping” module gets related data from the “CCE Modelling” module and provides a user interface for performing I/O mapping. The I/O mapping data is then processed by the “Mapping Modelling” module and passed to the “Control Deployment” module. Also, the I/O mapping data can be saved to an external file if the I/O mapping process of a project is suspended when it has not been finished. The “Control Deployment” module combines the reusable data from the database and generates the source codes. Also, the connections between each component and its PLCOpenInterface are generated by the “Control Deployment” module and are used by the “Virtual Model Mapping” which uses the predefined XSLT file to perform the mapping. Finally, both the “Virtual Model Mapping” module and the “Control Deploy” module pass its respective output data to the “Data Export” module to export the data into files of required data formats.

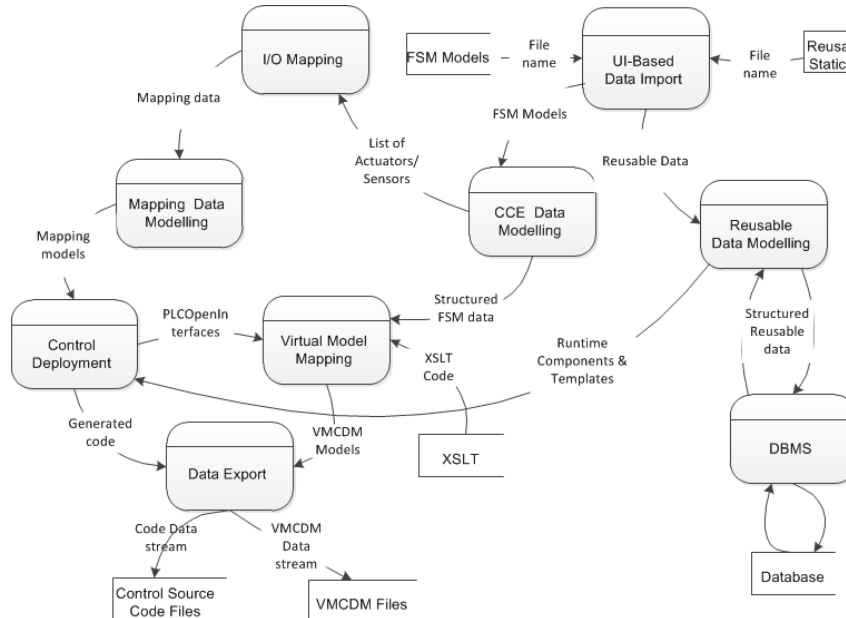


Figure 3-29 Data flow diagram of VCMapper

3.6.2.2. Workflow

Based on the dataflow designed above, the work flow of the system is designed and outlined in Figure 3-30.

The work flow starts from either creating a new project or opening an existing project. If a new project is created, in the next step, the PLC platform for which the control codes are generated must be specified. Obviously, only the platform which is supported by the software can be selected. Then files storing the component-based virtual models of the desired system need to be specified by the user and then loaded by the software. After the manual I/O mapping finishes, the mapping data will be automatically translated into PLC-interpretable control code which is then combined with reusable data to generate the complete source code. Also, mapping from component-based virtual model to the VMCDM is performed automatically to generate the desired VMCDM. It is worth noticing that the virtual model mapping must take place after the control deployment since the information of PLCOpenInterfaces required by the ontology mapping phase is generated during the control deployment.

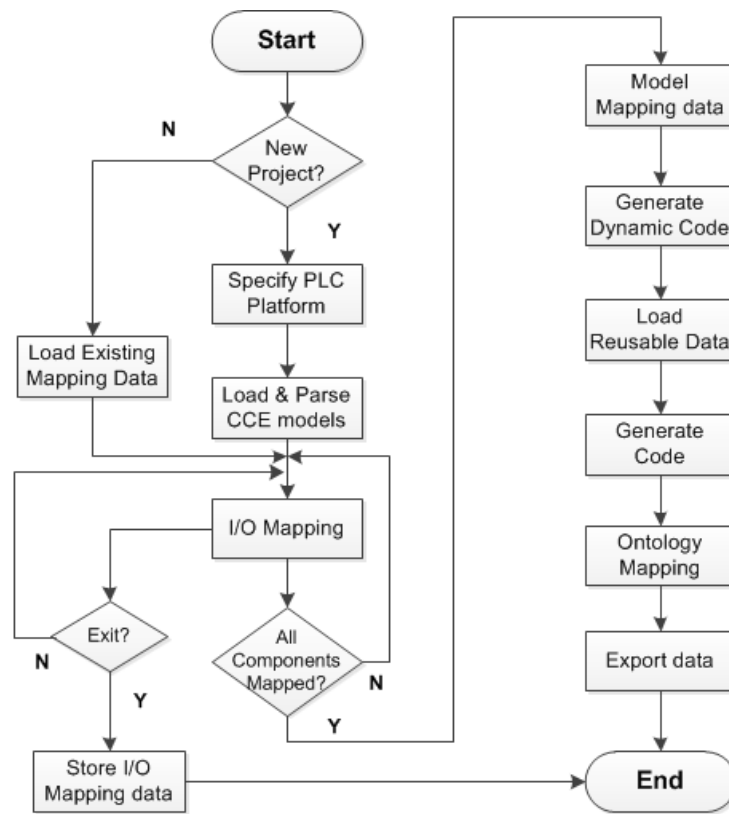


Figure 3-30 Workflow of VCMapper

3.6.2.3. Interface Design

Due to the diversities of PLCs available on the market, the code generation module should contain different sub-modules for deploying PLC code for different PLC platforms. Different platform-specific sub-modules can be potentially developed by different software engineers

and are finally integrated with other related modules. However, there is only one user interface to communicate with all the control deployment sub-modules. In order to enable the extensibility and integrability of the software, uniformed interfaces need to be defined.

An “Interface” is an abstract type specifying a set of methods, fields and properties [125]. Interfaces can only be declared and cannot be instantiated. A class that implements an interface must implement all of the methods described in the interface.

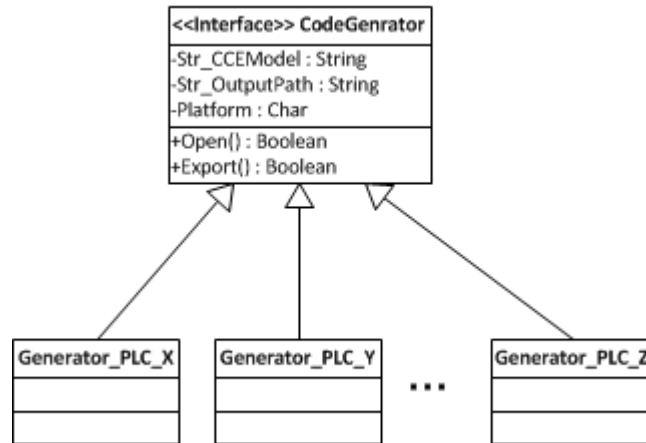


Figure 3-31 Unified interface for different code generation modules

By using ‘Interface’, the way of calling the methods, in the UI module, of classes defined in different control deployment modules for different PLC platforms, can be unified. In the case of this research, an interface named “CodeGenerator” is defined. As shown in Figure 3-31, the attributes and methods it must have are as following:

- Str_CCEModels: attribute for specifying the directory where files of the CCE models are saved in.
- Str_OutputPath: attribute for specifying the directory where the source code should be exported to.
- Platform: attribute which specifies the PLC platform for which the code will be generated.
- Open(): method used to open the CCE models the storage path has been specified by attribute Str_CCEModels.
- Export(): method used to export the generated source code to the directory specified by the attribute Str_OutputPath.

All the platform-specific sub-modules must inherit this interface and implement the required functionalities. According to the software structure of specific platform, different sub-modules might have other different attributes and methods. However, the user interface module would always call the method “Open” of any module to trigger the direct deployment process and call the method “Export” to end the process. On the other hand, the path of the CCE models will always be passed to the parameter “Str_CCEModels” and the path for saving the exported source code will be passed through “Str_OutputPath”.

3.7. Chapter Overview

Of the five key elements of the desired open virtual commissioning framework illustrated in Figure 3-1, the simulation engineering tool CCE and the new control software architecture have been designed and implemented by other ASG researchers. Therefore the focus of this chapter has been placed on the author’s work in proposing and designing the approaches and methodologies for achieving tool interoperability and the direct deployment of PLC control software.

The common data models for enabling virtual engineering tools to efficiently exchange data were first developed based on the domain-specific open standard- AutomationML. The common data models should be built based on object-oriented architecture, vocabulary of shared terminologies and XML-based data format. AutomationML is an XML-based data format which employs object-oriented architecture and provides shared vocabulary of domain-specific terms. In order to build the common data models, vocabulary of terms required to describe virtual models was first defined by inheriting the terms in standard vocabulary. Consequently, classes for describing different types of components and elements in the component-based virtual models were created and described using the standard terminologies of AutomationML. Then the approach to mapping the component-based virtual models to the common data models created in this research was designed. It should be noted that the common data models only cover the hierarchical information of the virtual models as many standard common data models for describing 3D geometric data and PLC control logic data have been available.

The approach to the direct deployment of control software was also designed based on the component-based approach. The approach development begins with developing reusable data at the component building phase. The reusable data includes runtime components, derived

data types and PLC-specific common information. Of these reusable data, the runtime components, which are used to communicate with the physical I/O of the related physical component, need to be programmed by control engineers. This aspect is beyond the scope of this research. The derived data types used to describe the control logic of virtual models in a PLC-interpretable manner were first developed. The PLC-specific common information was developed via a reverse engineering approach, namely through extracting common information from the source code of existing control software. In addition to the development of reusable static data, the approach to generating dynamic data through translating the control logic of virtual models in the system engineering phase was also designed. The dynamic data includes runtime control models for automatic mode control and manual mode control. The direct deployment process ends with combining related reusable data with dynamic data to create the source code of the complete control software.

The direct deployment process also involves a manual I/O mapping step to map each actuator/sensor component with its corresponding runtime component and I/O variables. In order to provide a user interface for the I/O mapping, the engineering tool VCMapper was designed in this research. This engineering tool was also designed to implement the virtual model mapping approach and the direct deployment approach. The data flow, work flow and required interface of the engineering tools were presented.

Chapter 4. Implementation and Experimental Study

This chapter reports the work carried out to implement and test the proposed approach.

The implementation of the proposed common data model and the engineering tool for mapping virtual models and deploying PLC control software are presented. The chapter provides an overall introduction to the test bed used to carry out the experiments. The experiment of applying the implemented common data model and virtual model mapping tool to realise inter-tool data exchange is described. Experiments for testing and evaluating the proposed direct deployment approach are presented. The chapter ends with a discussion on the results of the experiments.

4.1. Prototype Implementation

According to the design presented in chapter 3, in order to achieve the open VC framework, the enablers to be developed in this research are the VMCDM for describing component-based virtual models, and the engineering tool – VCMapper for implementing virtual model mapping and control software deployment.

This section describes the creation of the VMCDM model and the implementation of the VCMapper.

4.1.1. VMCDM Prototype

The VMCDM, which is described in AutomationML, is composed of a RoleClass library, an InterfaceClass library, a SystemUnitClass library and the InstanceHierarchy. They were respectively created, as illustrated in Figure 4-1, in the following ways:

RoleClass library and InterfaceClass library: were created through defining the classes required for describing virtual models and adding these classes into the standard libraries of AutomationML. The required role classes and interface classes and the approaches to defining them have been described in section 3.2. Considering Role Classes and Interface Classes are non-system-specific and AutomationML supports distributed data storage, these two libraries are stored as one separate AutomationML file and system-specific files can use them via external referencing.

SystemUnitClass library for Components: As stated in chapter 3, the ‘SystemUnitClass’ of AutomationML correspond to the ‘Component’ of component-based automation systems. Therefore, SystemUnitClass library were created through mapping virtual models of components into corresponding SystemUnitClasses. The mapping is performed by the VCMapper which will be presented in section 4.2.2. The SystemUnitClass library is also stored as a separate AutomationML file. As each SystemUnitClass needs to be assigned a role and it might contain interface classes, the SystemUnitClass library includes the RoleClass library and InterfaceClass library via an external reference. The InstanceHierarchy for describing system-specific data also includes SystemUnitClass library via an external reference.

InstanceHierarchy for System: describes virtual models of a specific system and can only be created through transforming the virtual models of a specific component-based system. It is also stored as a separate AutomationML file and connected with the SystemUnitClass library via an external reference.

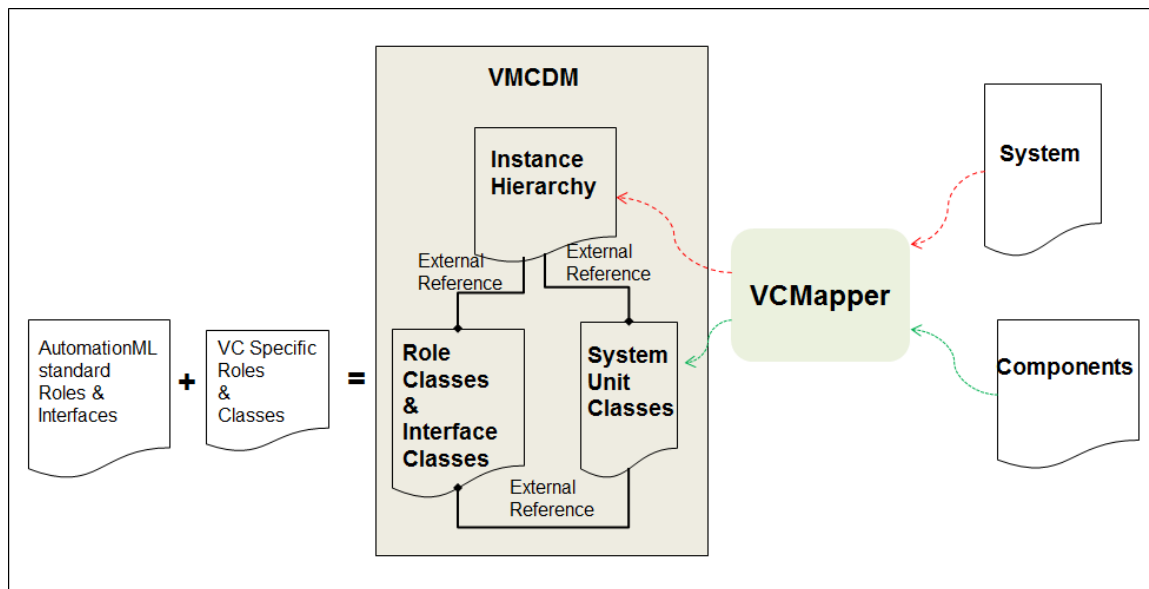


Figure 4-1 Development of VMCDM

Based on the VMCDM proposed in section 3.2, the XML Schema file was also created. XML Schema is used to define the legal building blocks of an XML documents. To be more specific, it defines constraints on structure and content of XML documents. These constraints include data types and default values of elements and attributes, attributes and elements that can appear in a document, the order and number of child elements, and so on.

The generated VMCDM needs to be validated against the constraints defined in the XML Schema.

An XML Schema is an XML-based file represented in XML Schema language which is also referred to as XML Schema Definition (XSD). AutomationML provides a standard XML Schema file based on its data structure. Therefore, the required schema file for the VMCDM model was developed by modifying the generated schema.

4.1.2. VCMapper Prototype

Based on the design presented in section 3.6, the VCMapper was developed using the Visual Basic.net programming language in the Microsoft Visual Studio 2010 development environment.

The objective of developing the VCMapper is to automate virtual model mapping and control software deployment. Apart from a simple user interface provided for performing I/O mapping, other modules underlying the interface were built as Dynamic Link Libraries (DLLs). Apart from DLLs, virtual model mapping was implemented using XSLT as it is more flexible. A database was created using Microsoft Access for storing the reusable data. The VCMapper comprises the following modules:

1. User Interface
2. CCE Model Management Module
3. Virtual Model Mapping Module
4. I/O Mapping Management Module
5. PLCOpenXML Deployment Module
6. Siemens S7 Deployment Module
7. Database Module

The relationship between the functionality of the user interface and the other modules are illustrated in Figure 4-2. The data exchange and relationship between these modules can refer to the data flow diagram illustrated in Figure 3-29. The following sections describe the implementation of the above modules.

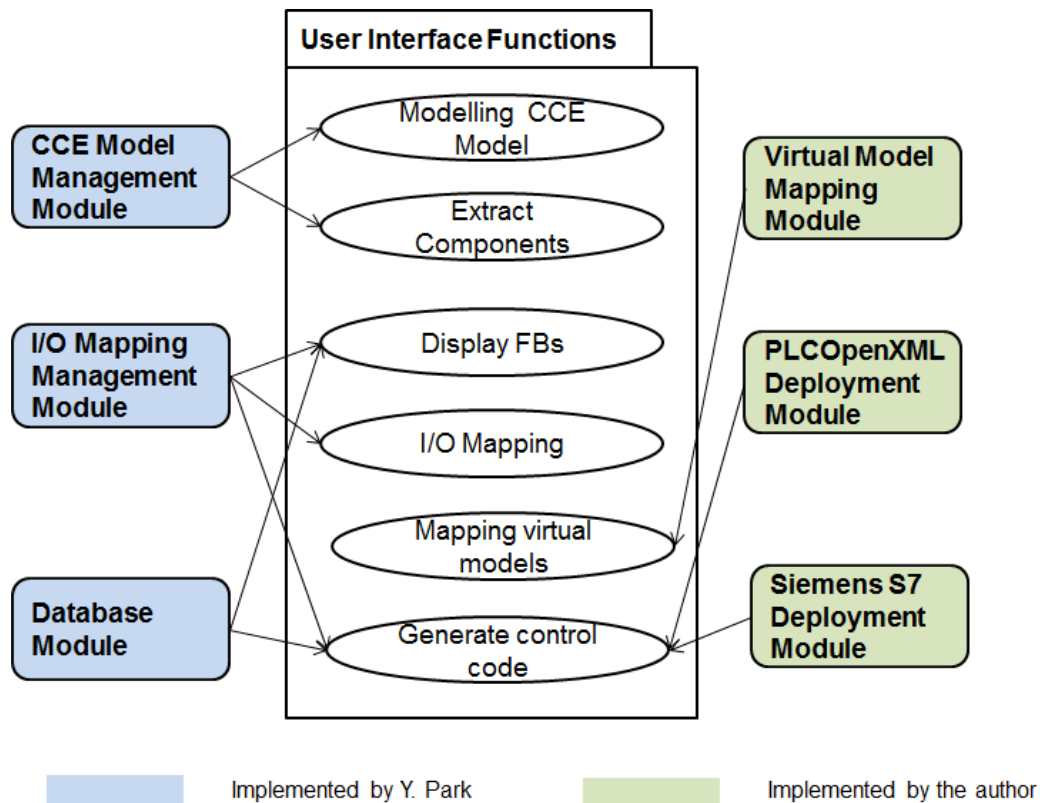


Figure 4-2 Relationship between the user interface and underlying modules

4.1.2.1. User Interface

The user interface was implemented mainly to provide the required functions for performing I/O mapping. A snapshot of the developed UI is illustrated in Figure 4-3. It can be seen that the functions implemented in this module include those for:

- Reading the runtime components from the database and listing them out on the UI were implemented.
- Extracting the actuator and sensor components from the CCE data module and listing them out.
- Adding I/O variables to a selected runtime component. The I/O mapping data will be processed to the underlying I/O mapping management module.

Additionally, other functions which can assist the I/O mapping activities were also implemented. These functions were developed for importing the following required data:

- CCE virtual models in XML files
- I/O variables provided by control engineers in the form of Microsoft Excel sheet

- Reusable data including derived data types, runtime components and PLC-specific common information in text files.

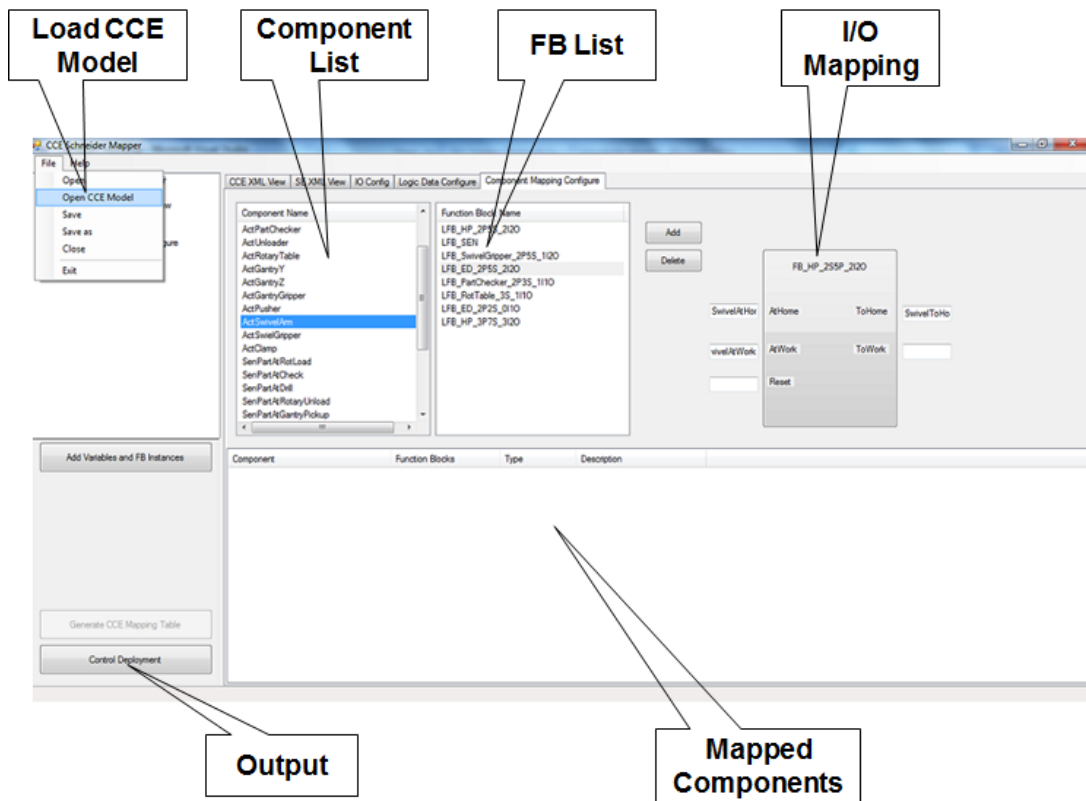


Figure 4-3 User interface for I/O Mapping

4.1.2.2. CCE Model Management Module

This module was implemented for accessing the CCE virtual models imported via the User Interface. Information about sensor and actuator components was extracted out and passed to the User Interface. The control logic of the systems was modelled in order to be used by the control deployment modules.

According to the structures of component-based automation systems, classes for describing system, component, state, transition and condition were respectively implemented. These five classes have inclusion relations in the order listed.

The class “clsSystem” was implemented as the access point of the whole library. A function called “CreateCCEModel()” was implemented as the interface to create and access the XML DOM object of a CCE model.

4.1.2.3. Virtual Model Mapping Module

According to the design in section 3.3.2, items required for virtual model mapping include:

- XSLT Style sheets: for implementing the transformation from source XML files of component-based virtual models to target XML files of VMCDM.
- XML Document Object Model (DOM): a standard interface for accessing and manipulating XML documents. The DOM presents an XML document as an in-memory tree-structure.
- XSLT Processor: an application executing the functions of XSLT style sheet to perform the transformation.

This section is focused on presenting how to develop the required XSLT style sheet and use the developed XSLT sheet in VCMapper to implement virtual model mapping. The required XML DOM and XSLT Processor can be implemented in the VCMapper by adopting the DOM parser and XSLT processor provided by the Microsoft Visual Studio.

The objects and attributes of CCE models which can be directly mapped to its VMDCM equivalences were selected using XPath, which is a query language for selecting nodes from an XML document, and the transformation rules were implemented in the XSLT. In order to implement the transformation of components and systems, three types of XSLT templates were developed respectively for transforming static components, sensor components and actuator components.

Given the fact that XSLT is only a simple language for the transformation of XML-based file style, attribute transformation functions were implemented using VB.net programming language in a XSLT file named "VBFunction.xslt". For the target attributes the values of which can be gained through searching the mapping tables, a function was implemented and is called by all the related templates. For the source attributes the values of which need to undergo a computation to get the values of the equivalent target attributes, a function was defined for each of these attributes. The transformation from the position information of CCE to "Frame" was implemented as a VB function. Other similar functions are those for transforming interfaces to geometry files and interfaces to control software.

4.1.2.4. I/O Mapping Management Module

The objective of implementing this module is to model the I/O mapping data received from the user interface. This module can also save the data as an XML file or read I/O mapping data from an existing XML file.

Apart from the library, the structure of the XML file for saving I/O mapping data was also created. The I/O mapping data of a project is stored as an XML file so that it can be reused to create another project by reconfiguration. XML-based data storage is also required for the cases in which the information of the whole project needs to be stored if users exit the tool before completing the I/O mapping for all the components. As outlined in Figure 4-4, apart from the mapping information, the XML file also contains the following information:

- Project information: mainly includes the information of the selected PLC platform.
- CCE model information: includes the name and the file name of the involved CCE model.
- Information of I/O variables: information of all the I/O variables involved in a project.

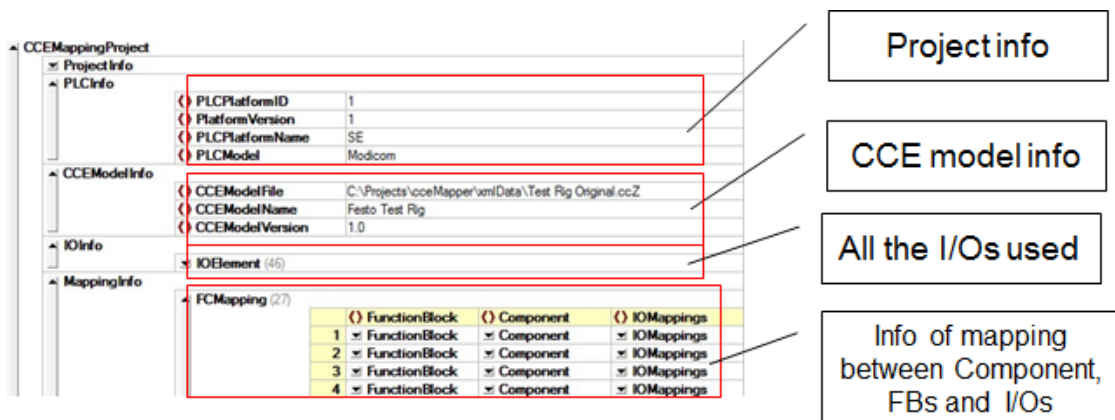


Figure 4-4 XML file for I/O Mapping

4.1.2.5. PLCOpenXML Deployment Module

Considering the diversity of available PLCs as well as the data formats of their related programming software, the control deployment module was divided into different sub modules each of which implements the deployment for a specific PLC platform. In order to be compliant with the AutomationML open standard, a library for deploying control software in the standard format of PLCOpenXML was developed. Due to the fact that Siemens PLCs

are widely used by the automotive industry, another library for deploying control software for the Siemens Step 7 platform was also implemented and is presented in Section 4.1.2.6.

In this library, a class “clsPLCOpenXML” was implemented as the access point and for generating the finalised source code. The functions included in this class were implemented for the following different purposes:

- (1) Exchanging data with other modules and this type of functions were developed for:
 - Receiving CCE model: the function receives a CCE model DOM object and then calls functions of the CCE model library to create and populate a CCE model object with the DOM object.
 - Outputting generated source code: the function outputs the generated source code as stream data which will be exported by the UI module.
- (2) Populating the pertinent instance objects. All the constituent items of the PLCOpenXML are populated respectively by the corresponding functions. According to the data used to populate an item, these items can be further classified as follows:
 - Populated using reusable data: the DataTypes and POUs are populated respectively by reading the derived data types and the runtime components from the database.
 - Populated using imported static data: refers to the elementary variables of the global variable table. The function gets the variable data from the UI either by importing from an external Excel file or input direct through the UI and then translates them to the format of the PLCOpenXML.
 - Populated using dynamically generated data: first of all, arrays of the corresponding derived data types of the Global variable table GlobalVar for storing the information of control models were created by analysing the CCE models. Secondly, a function for initialising the arrays was then implemented through analysing the CCE models to populate the arrays. Thirdly, the POUInstance list was populated by analysing the I/O mapping data from the UI. Lastly, a function was implemented to generate the final source code by combining the body strings of all the populated items.

4.1.2.6. Siemens Step 7 Deployment Module

Siemens Step 7 platform partially supports the IEC61131-3 standard. There are still S7-specific languages and objects. Through the analysis of existing control software programs of Step 7, it was identified that Step 7 differs from the IEC61131-3 standard in the following respects:

- **S7-specific languages:** the textual languages used in Step 7 are similar to the Instruction List and Structured Text of IEC61131-3. However, differences still exist in the data format.
- **S7-specific data format:** all the data formats of exported source code files of S7 are plain-text-based; however, the suffixes of the files indicate that they are specific to the S7.
- **S7-specific naming conventions:** the control software structure of S7 generally follows that of IEC61131-3 standard; however, the names of some objects are different. For example, the ‘Task’ of PLCOpen is named as Organisation Block (OB) in Step 7.
- **S7-specific objects:** an instance Data Block (DB) is required for each Function Block (FB) instance to provide a static memory area for its related variables. All the variables of derived data types, which are part of the global variable list in PLCOpenXML, must be stored in one or more share Data Blocks (DB) in Step 7. In addition, the element unit of the OB is “Network”. A “Network” comprises the names of (1) the FB or Function called back by the OB, (2) the required instance DB if a FB is called back, and (3) the I/O variables.

It can be observed that of the above outlined items, the first three ones actually have their corresponding counterparts in the IEC61131 standard. The differences lie in the function for generating the final source code according to the S7-specific requirements. Also, for the function “Export()” which was implemented to export the generated source code, the file names are assigned with the suffixes which are required by the Step 7 (S7).

Classes for representing the S7-specific Data Blocks (DBs) were implemented specifically for the Step 7. The class for shared DB was implemented in the same way as that used in the PLCOpenXML library. The difference lies in its function which formats the final source code according to the data structure of the S7 shared DB. The class for instance DB was implemented specifically for the S7 platform. A function of the instance DB class was

implemented to generate the source code by 1) parsing the source code of the related FB and getting the information of all the variables of this FB, and 2) parsing the information of every variable and generating the source code of the desired DB.

Functions for automatic generation of the HMI for the manual mode control were also implemented in the S7 library. Based on the software architecture presented in Section 3.6.1, the class “clsDB4HMI” was implemented to generate the required shared DB for HMI generation. The approach to generating the data for HMI, which has been described in the Section 3.5.3.2, was also implemented to generate the source code of a shared DB which includes the data to be used to create the runtime HMI.

The class “clsS7Program” was implemented as the access point to generate the complete source code for S7. This class plays the same role as the class “clsPLCOpenXML” does in the PLCOpenXML library. Due to the S7-specific features outlined above, the process of its function “PopBodyString” is different from the counterpart function of PLCOpenXML. The functionalities implemented in this function to generate the final source code, by calling the related functions of the pertinent instances, were outlined as follows:

- 1) Populate the variable table according to the data format of S7 variable table.
- 2) Load the derived data types from the data base to generate the source code.
- 3) Generate the shared DBs for auto-mode control and HMI generation.
- 4) Parse the I/O mapping data to: a) generate the list of “Network” instances, and b) generate the mapping table from component to the VCInterface to be used by the ontology mapping module.
- 5) Parse the list of “Network” to : a) get the source code of all the FB/FCs involved from the database and generate the source code of the FB/FCs, b) generate the source code of all the required DBs, c) generate the source code of the OB1 by combining the source code of all the “Network” instances.
- 6) Combine all the related objects to generate the final source code of the desired project and export them according the data formats of different source codes.

4.1.2.7. Database Module

This module is implemented for manipulating the reusable data which is saved in the database. In this module, the database and the library for accessing data in the database were respectively devised.

The reusable data are accessed during the code generation process. In order to gain efficient data accessibility, the static/reusable data are stored in a relational database. The Microsoft Access database was adopted for the storage of the static/reusable data.

Various tables need to be created in order to store various types of reusable data and other required data. As outlined in Figure 4-5, the following tables were created:

- **PLCDataType**: for storing the derived data types defined for describing the components and its control behaviours.
- **PLCFunctionBlock**: for storing the data of runtime components.
- **PLCFunctionBlockIO**: for storing the input and output parameters of Runtime Components (function blocks).
- **PLCTemplates**: for storing the static templates data of all the platforms that the mapper can generate code for.
- **PLCPlatform**: information of the PLC platforms for which the VCMapper can generate the control code.

The structures of the above mentioned database tables and their relationships were shown in Figure 4-5. Due to the data format differences between the platforms, a reusable object might have different versions for different platforms. Therefore, it can be observed that the table **PLCDataType** and table **PLCFunctionBlock** respectively contain a field named **PFID** for specifying the platform.

According to the data flow of the mapper which was depicted in Figure 3-29 of section 3.6.2.1, the reusable data is first read from external files and then written to the database. During the deployment process, the reusable data is then read by both the user interface module and the control deployment module. In this context, functions for reading and writing database were implemented.

For the database management library, VB.net classes were implemented to describe the corresponding reusable data. Each class defined here only has properties for describing the attributes of the corresponding data. A class “**clsPLCDatabase**” was implemented as the access point to all the other classes. This class contains the methods of interacting with database and populating respective instance objects.

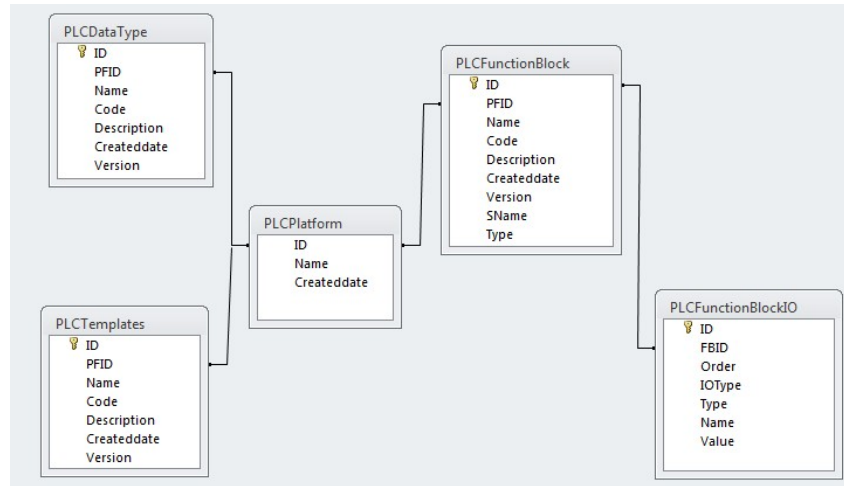


Figure 4-5 Database tables for storing reusable data

4.2. Overview of Experiments

The overall purpose of the experimental study reported in this chapter is to illustrate the feasibility and features of the proposed VCOM virtual commissioning framework. To be more specific, the purpose can be detailed as:

- To illustrate the features of data exchange and data representation of virtual models using the proposed VMCDM domain-specific models.
- To demonstrate the feasibility and performance of the approach to directly deploying the control software of an automation system based on its component-based virtual models.

The rest of this section provides an introduction to the physical test bed as well as its virtual model and related runtime components, which were used for performing the case studies. An overview of the particular case studies to be performed on the adopted test bed is also presented.

4.2.1. Introduction to experiment resources

In order to demonstrate the features of the VCOM open virtual commissioning framework which adopts the proposed VMCDM models and direct deployment approach, required resources include: 1) a physical automation system, 2) the virtual model of the physical system which has been virtually commissioned, and 3) the relevant reusable runtime components which have been validated.

4.2.1.1. Physical test bed



Figure 4-6 the experimental test bed

The test bed adopted for the experimental purpose is depicted in Figure 4-6. This test rig provides movements similar to a Ford powertrain assembly line and the functionality of this automation test rig is deemed to be applicable to real industrial machinery and control applications experienced by the Ford Motor Company. The test rig comprises various sensors and actuators, which are electrically or pneumatically controlled and accessible through the distributed I/O interface modules of the rig.

The test bed is controlled by a PLC-based control system with distributed I/Os. In order to demonstrate the features of the proposed direct deployment approach, two PLCs from different vendors were selected as the control systems for two independent experiments:

- CodeSys SoftPLC and Modicon I/Os: the CodeSys PLC programming software supports the PLCOpenXML standard data format and the Modicon I/Os support the standard Modbus TCP protocol.
- Siemens S300 and Siemens ET200S Remote I/Os: the Siemens PLC programming software partially supports the IEC61131-3 standard programming languages and it also utilises its own proprietary programming languages and data formats. The ET200S Remote I/Os support ProfiNet.

4.2.1.2. Virtual commissioning of the test bed

According to the workflow of the proposed VC framework, the test rig was first virtually prototyped and commissioned using the component-based engineering tool – CCE.

Corresponding to the architecture (Subsystem – Component - Element) of the Component-based approach, the test rig was considered as a system and decomposed into four subsystems which are also called stations in this dissertation. Each of the stations was further decomposed into multiple components. The decomposition of the test rig is illustrated in Figure 4-7 and the details of each station are outlined in Table 4-1.

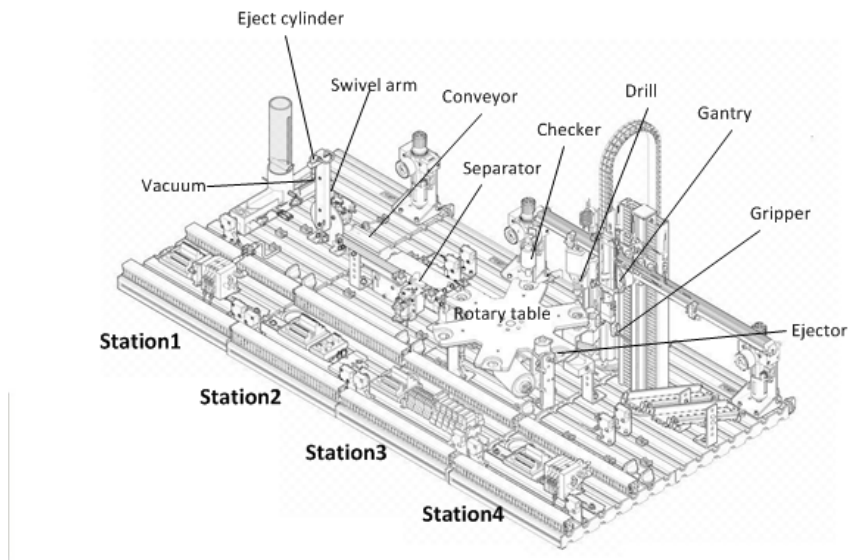


Figure 4-7 Decomposition of the Test Rig

Table 4-1 Decomposition of the test bed

Subsystem	Components	
	Actuator	Sensor
Station 1	<ul style="list-style-type: none"> • Eject_cylinder • Swivel_Drive • Vacuum 	<ul style="list-style-type: none"> • Magazine • Magixer_Ready • Gripper
Station 2	<ul style="list-style-type: none"> • Conveyor • Separator 	<ul style="list-style-type: none"> • WP_at_Start • WP_at_Separator • WP_at_End
Station 3	<ul style="list-style-type: none"> • Indexing_Rotary_Table • Checking_Actuator • Drill • Drill_Spindle • WP_Clamp • Eject_Actuator 	<ul style="list-style-type: none"> • WP_Available • WP_at_Checker • WP_at_Drilling • WP_at_Eject
Station 4	<ul style="list-style-type: none"> • Arm • Gripper • Gripper_Extend_Cylinder 	<ul style="list-style-type: none"> • WP_IsNot_Black • WP_Receptable

According to the component-based approach, the components of the test rig were built and saved in reusable component library. The control behaviours of sensor components are described using a two-state State Transition Diagram and non-control components obviously have no control behaviours. However, control behaviours of actuator components are diverse and depend on the mechanical behaviours of specific components. For example, as shown in Figure 4-8, the control behaviour of the actuator “Swivel Arm” is represented as a STD with five states, two of which are dynamic states.

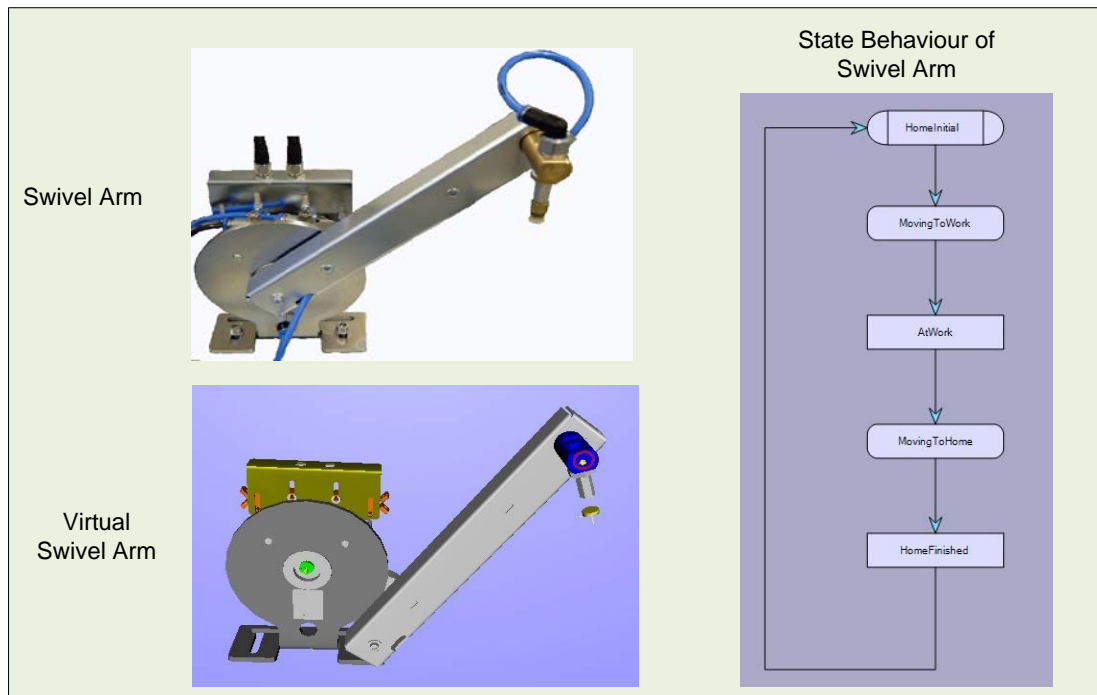


Figure 4-8 Example of actuator components – Swivel Arm (Physical, virtual and state behaviour)

The virtual model of the whole test rig was built by combining the pertinent components. The complete virtual prototype is illustrated in Figure 4-9. The control logic of the virtual rig was developed by interlocking the state behaviours of related components. As an example, the control logic of the Station 1 is shown in Figure 4-10.

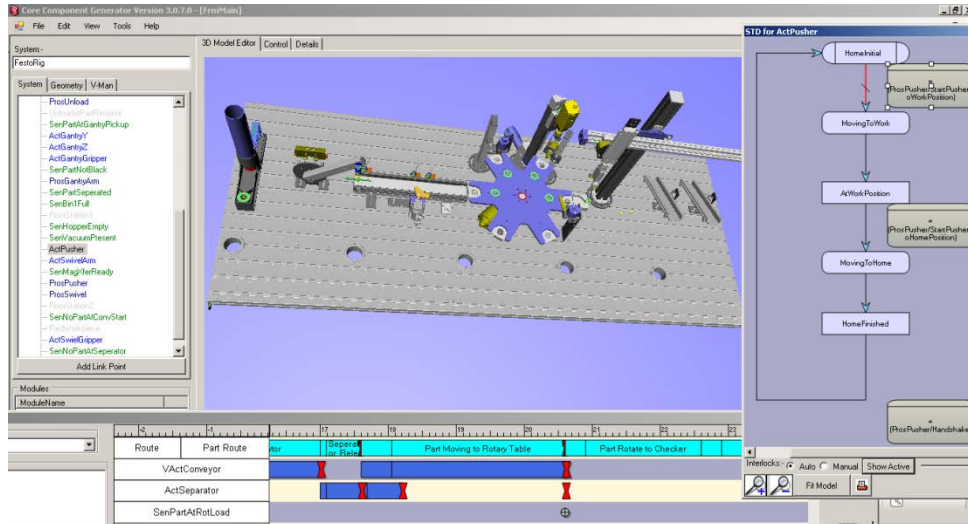


Figure 4-9 Virtual prototype of the test rig

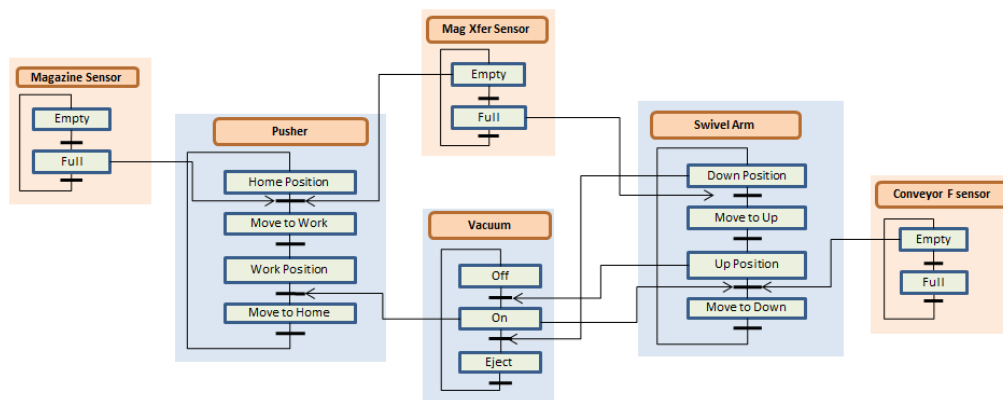


Figure 4-10 Control logic of Station 1 (Pusher, Swivel, Vacuum)

4.2.1.3. Runtime components for the test bed

The relationship between runtime components and resource components is a 1-to-N relationship. Different actuator components might be represented as the same runtime component in the control software and all the sensor components correspond to the same runtime component. The factors for judging whether two different actuator components correspond to the same runtime component are the number of states and kinematic positions, and the type of driving power [12].

For the thirteen actuator components of the virtual model of the test bed, eight runtime components are required for implementing the PLC control system. Some actuator components correspond to the same runtime component. For instance, 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 function block named 'LFB_HP_2P5S_2I2O'. All sensor components use the same Runtime Component, i.e. LFB_SEN. The relations between the runtime components and resource components are given in Table 4-2.

Table 4-2 Runtime components for the Festo Rig

Function/FB	Related Components	Comments
LFB_SEN	All the 15 sensor components	Function for all the sensor components
LFB_HP_2P5S_2I2O	Pusher, Swivel Arm, Ejector, GantryZ, Gantry Clamp, Gantry Gripper, Separator	FB for pneumatic actuator components with 2 positions and 5 states
LFB_SwiGripper_2P5S_1I2O	Swivel arm suction	FB for pneumatic actuator components with 2 positions and 5 states
LFB_ED_2P5S_2I2O	Drill	FB for electronic actuator components with 2 positions and 5 states
LFB_PartChecker_2P3S_1I1O	Part Checker	FB for pneumatic actuator components with 2 positions and 3 states
LFB_RotTable_3S_1I1O	Rotate Table	FB for rotate table with 3 states
LFB_ED_2P2S_0I1O	Drill Spin	FB for electronic actuator components with 2 positions and 2 states
LFB_HP_3P7S_3I2O	GantryY	FB for pneumatic actuator components with 3 positions and 7 states

4.2.2. Case Studies

The test bed, its virtual model and the runtime components offer a number of cases of interest for illustrating the use of the proposed concepts and approaches. With the offered cases, the feasibility and features of the proposed framework can be proven.

4.2.2.1. Data reuse of virtual models

The virtual models of the test bed created using the CCE editor were viewed using another engineering tool. The virtual models were first transformed from tool-specific data models into the tool-independent VMCDM data models and then imported into an independent engineering tool – the AutomationML Editor, which supports the AutomationML data format.

This is mainly to illustrate the features of the VMCDM developed in this research. In addition, this is also to demonstrate:

- The compatibility of the VMCDM for describing the virtual models of component-based automation systems.
- The feasibility of the data model mapping from component-based models to HIL models.
- The semantics the VMCDM provides. The virtual models can be viewed in the AutomationML Editor which provides a classified view on the attributes and objects of the VMCDM models.

4.2.2.2. Direct deployment of control software for PLCOpenXML

The control software for CodeSys softPLC was directly deployed by the CCEMapper which translates the control behaviours of the virtual models and reuses the runtime components. The generated control software was exported as a PLCOpenXML file and the PLCOpeninterfaces for connection the VMCDM models with corresponding variables in the PLCOpenXML file were also automatically created. This is to show:

- The feasibility of the proposed direct deployment approach. The control software for PLC-based control system can be automatically generated based on the component-based modular virtual counterpart of the desired assembly system.
- The process of automatically building the connection between VMCDM models with its control software via PLCOpeninterface. This is actually required to complete the virtual model mapping.

4.2.2.3. Direct deployment of S7 control software

The Siemens S300 PLCs are widely used by the automotive industry and provide the functions required for industrial application. Apart from aiming at illustrating the feasibility of the direct deployment approach, the direct deployment of control software for the Siemens S300 PLC is also to illustrate:

- The feasibility of the direct deployment of the control models for HMI. The HMI for manual mode control and diagnostic is required for industrial application. For a solution which aims at achieving industrial applicability, the direct deployment of HMI has to be achieved.

- The features of the direct deployment approach. In addition to the feasibility, the efficiency of the proposed approach and the performance of the PLC control software, which is directly deployed, are also important factors in evaluating the applicability of the approach.

4.3. Virtual Model Mapping Experiment

This section illustrates the application of the proposed common data models and virtual model mapping function to enable data exchange between engineering tools. The component-based virtual model of the test rig was transformed into VMCDM which were then reused by another engineering tool – AutomationML Editor.

4.3.1. Overview

In this experiment, the engineering tool – CCE tool was selected as the source engineering tool while the AutomationML Editor was adopted as the target engineering tool.

Comparison between the respective data formats of the CCE and the AutomationML Editor is outlined in Table 4-3. The CCE Tool exports the component-based virtual models into XML-based files. However, except from geometry information described in VRML, the hierarchy information and control logic of CCE virtual models are represented in tool-specific XML files. The AutomationML Editor supports AutomationML standard as it is a toolset provided by the AutomationML organisation.

Table 4-3 Data formats of the source tool and the target tool

	Source tool –CCE	Target Tool- AutomationML Editor
Hierarchy Data	XML	AutomationML
3D Geometry	VRML	COLLADA
Control Logic	STD in XML	PLCOpenXML

According to the procedure of common model-based data exchange described in Section 3.3.1, the experiment was performed via the following steps:

- Component-based data model export: to export the virtual models of the test bed from the CCE tool to XML files.

- Data model mapping: to transform the CCE-specific virtual models of the test bed into tool-independent VMCDM common data models.
- Common data reuse: to open and view the transformed VMCDM common data models using the neutral engineering tool – AutomationML Editor.

4.3.2. CCE Virtual Model Export

In the first step of data exchange experiment, virtual models were exported from the CCE engineering tool to XML files. Information from different disciplines which composes a component or a system was exported into different XML files. The hierarchical structure information was exported to an XML file while the control behaviours were exported to another separate XML file. For the same component of a system, it has identical global unified identity (GUID) in the two XML files. The constituent geometrical elements were exported to VRML files.

Examples of the exported XML files were shown in Figure 4-11 and Figure 4-12. It can be observed that the reasons why the CCE models are difficult to be reused include:

- Tool-specific vocabulary: All the tags used in the XML files exported from CCE are tool-specific. Although some of them provide semantic information which makes it readable and understandable, direct reuse of these XML files by other engineering tools is still difficult.
- Non object-oriented architecture: although the component-based approach, on which the CCE tool were based, adopts the object-oriented architecture. However, the exported XML files only contained the information of the specific system and did not represent an object-oriented architecture.
- Simulated control logic: the control logic which is represented as STD and cannot be reused by most of the virtual commissioning engineering tools which validate virtual models against real control code.
- Lack of definition and classification: terms for integrating different disciplinary information in the CCE models are not predefined and classified. This makes the virtual models difficult to be understood by either human beings or engineering tools.

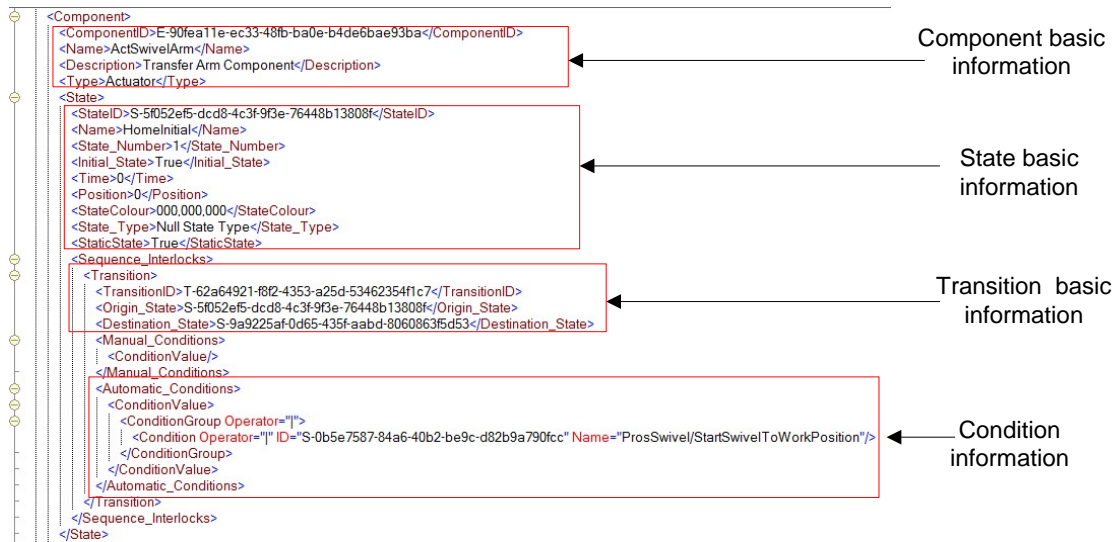


Figure 4-11 Control logic information of exported CCE models in XML files



Figure 4-12 Hierarchy information of exported CCE models in XML files

4.3.3. Mapping CCE to VMCDM

The virtual model mapping in this thesis is focused on the hierarchical data transformation and the control logic transformation is implemented by the direct deployment solution. As illustrated in Figure 4-13, the logic mapping module generates the interfaces to the control software for the virtual model mapping module. In order to demonstrate a visible result of the ontology mapping, the geometry data of the test bed, which were exported as VRML files, were manually transformed into COLLADA files using a commercially available engineering tool – Blender [126].

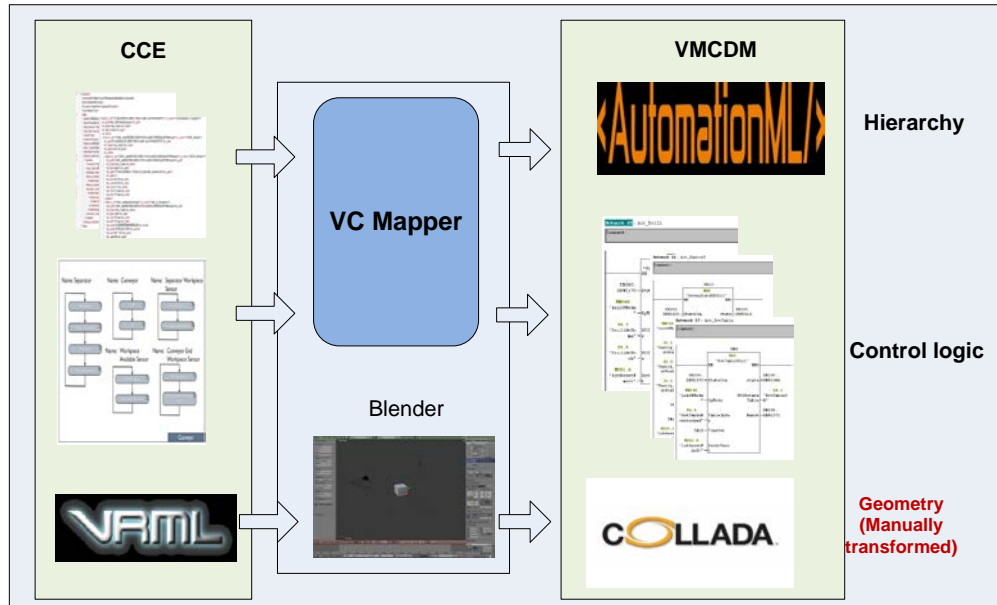


Figure 4-13 Transform CCE virtual model of the test bed to VMCDM

Hierarchical Data transformation

The hierarchical data was transformed by the VCMapper. As stated in section 4.2.2.3, mapping of terminologies which are identical to any component or system had been implemented in the XSLT. For other terms which are system-specific, a mapping table was created and saved as a XML file to be used by the XSLT during mapping. The mapping table, which specifies the role of each component, for the test bed is shown in Table 4-4.

Table 4-4 Role of each component of the test bed

Components	Role
Pusher, Swivel, Gantry	AutomationMLMIRoleClassLib/ManufacturingEquipment/Transport
Vacuum, Clamp	AutomationMLMIRoleClassLib/ManufacturingEquipment/Fixture
Conveyor	AutomationMLExtendedRoleClassLib/Conveyor/BeltConveyor
Separator, Ejector	AutomationMLMIRoleClassLib/ManufacturingEquipment/Movable Tool
Rotary Table	AutomationMLExtendedRoleClassLib/TurnTable
Drill, PartChecker	AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Resource/Measurement Equipment
Floor1, Floor2, Floor3, Floor4,	AutomationMLMIRoleClassLib/StaticObject
Part receiver, Bin1, Bin2	AutomationMLMIRoleClassLib/ManufacturingEquipment/Storage

It can be observed that all the roles used for the test bed are normative roles of AutomationML. Actually, to provide more specific semantics, more informative roles can be defined according to the need of users. For instance, an informative role “Rotate device” can be defined by inheriting the normative role “Transport” for the component “Rotate table”. This can obviously enhance the human-readability of the XML files. However, user-specific classes can also reduce the semantic interoperability between engineering tools.

Interfaces to geometry data

Since each VRML file which represents geometry data were transformed to a COLLADA file, its corresponding URL contained in the hierarchical data needs to be changed as well. Therefore, another table which specified the mappings between the URL of each VRML file and its target COLLADA file was also created and stored as a XML file.

4.3.4. Data Reuse of VMCDM

Although AutomationML has been accepted as an IEC standard and promises to be an industrial open standard, so far no commercial engineering tool is available on the market which supports AutomationML. This is mainly because that AutomationML has only been released for around three years. According to the documentations released by the AutomationML organisation, a few academic researches which used AutomationML as data exchange format were conducted. However, no further information has been available by September 2013 when this thesis was completed. In this context, an engineering tool named AutomationML Editor provided by the AutomationML organisation for viewing and editing AutomationML files was adopted to view the VMCDM model of the test bed in a tool-independent way.

The VMCDM model of the test bed can be directly imported into the AutomationML Editor, as illustrated in Figure 4-14. The role classes, interface classes, components and systems were displayed in different sections. The 3D model of Station 1 was also displayed. Information about the attributes of each component or internal element can be viewed and edited in the right side of the tool window.

However, no further functions based on the virtual models can be tested or demonstrated in the AutomationML Editor because this is only a tool for viewing and editing AutomationML files.

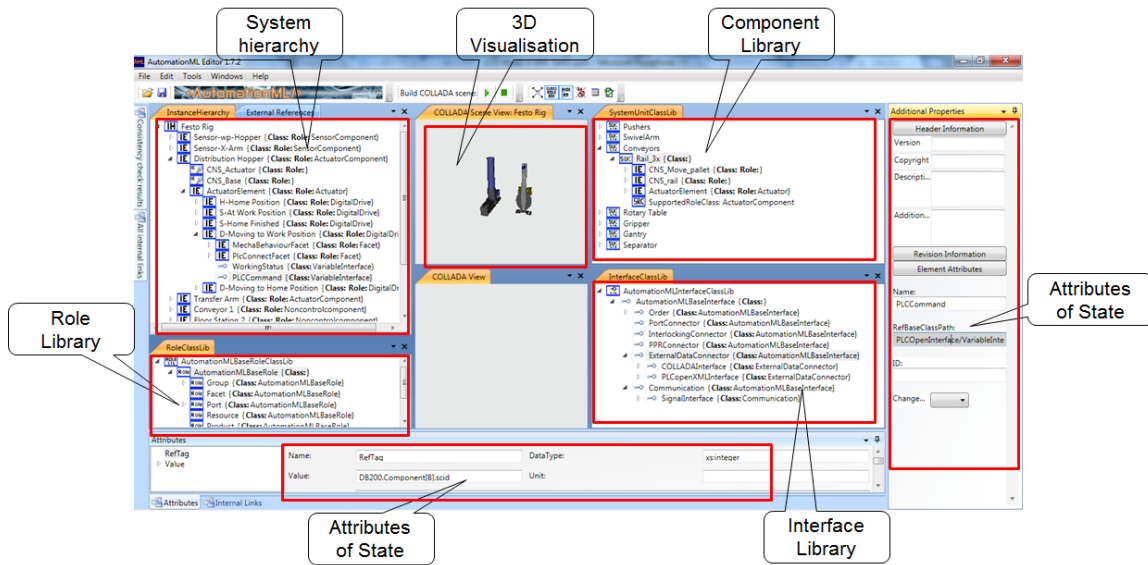


Figure 4-14 the VMCDM of the test bed opened in the AutomationML Editor

4.3.5. Evaluation

The experiment of exchanging virtual models from the CCE tool to the AutomationML Editor via the VMCDM shows the feasibility of describing the component-based virtual models in AutomationML. However, features of the VMCDM cannot be sufficiently tested and demonstrated due to the limited functions of the AutomationML Editor. In this context, the evaluation of the VMCDM was performed through analysis on some key features of the VMCDM.

Some advantages of the VMCDM can be observed from the experiment. Obviously, the VMCDM enables the virtual components to be efficiently reused. This is consistent with the philosophy of the component-based approach. Two factors contribute to the achievement of this advantage. Firstly, the semantic, provided by AutomationML through defining the vocabulary and domain specific terms, is the basis of the efficient data reuse. Compared with pure XML-based description, AutomationML defines standard terminologies and provides classifications of the defined terms. This is important to the description of virtual commissioning models as it is an integration of multi-disciplinary information. Secondly, embedding the component-based automation into the framework of AutomationML leads to the achievement of efficient data reuse. The VMCDM keeps object-oriented architecture of the component-based automation. Moreover, the control logic of a virtual component, which is specific to the component-based automation, was translated to the widely used PLC-based control logic and the connection between virtual models and PLC control were also built in

VMCDM. This is of significant importance for the virtual models of component-based automation to be reused.

On the other hand, a few limitations lying in the VMCDM can also be observed. While VMCDM provides standard representation of virtual models, some advantageous features of the CCE model cannot be kept in the VMCDM. In the CCE virtual models, the assembly of 3D geometry elements is realised using “Link Points” which is an innovative approach in 3D modelling proposed by Vera [127]. However, in the VMCDM, the relative position can only be represented using “Frames”, which cannot represent all the features of “Link Points”.

It is can also be observed that the conducted experiment did not show that the seamless data exchange via VMCDM has been achieved as all the functionalities of the CCE virtual models had not been replicated in AutomationML Editor. This is mainly because that AutomationML Editor is not a engineering tool with all the functionalities required for virtual engineering. However, once AutomationML-compliant VC engineering tools are available, VC can be repeated in these tools through reusing the VMCDM model as it contains all the data required for performing VC.

4.4. Direct deployment experiment

This section tested and evaluated the direct deployment function of the VC mapper through examples of directly deploying the control software for the test bed.

4.4.1. Overview

The required PLCs and the related programming software were selected first. To test the deployment of PLCOpenXML and Siemens PLCs, which the VCmapper supports at the time of writing, the respective hardware and engineering tools employed are outlined in Table 4-5.

Table 4-5 Hardware and software used for the experiments

Platform	PLC hardware	I/Os	Programming tool	Software for HMI
PLCOpenXML	CodeSys V3 SoftPLC	Modicon I/O using Modbus TCP/IP protocol	CodeSys V3.5	–
Siemens	Siemens S300 series	Siemens	Step 7 V5.4	WinCC

The process of conducting the experiments was then designed and shown in Figure 4-15. Some features, which have been mentioned in chapter 3, were depicted in this feature. First, the platform-specific common information was created using the reverse engineering process described in section 3.5.2. Another significant feature is that only I/O mapping needs to be performed manually during the deployment process and all other aspects are automated. For any platform, the I/O mapping process is completed using the same user interfaces and in the same way. During the phase of real commissioning, the hardware configuration needs to be performed manually.

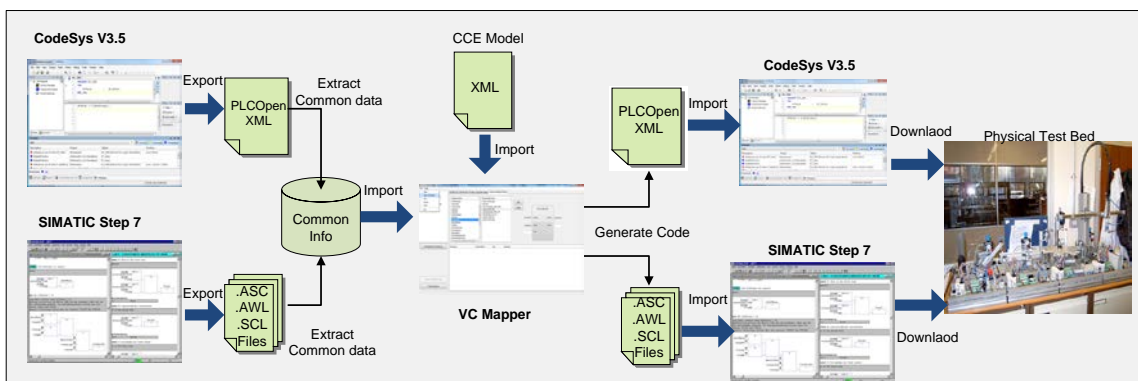


Figure 4-15 Process of testing by the direct deployment solution

After the real commissioning, the direct deployment approach was evaluated. Considering that the qualitative features of the approach can be observed during the commissioning, the evaluation was mainly focused on some quantitative aspects of both the direct deployment approach and the control software it deployed.

4.4.2. Platform-specific common information development

This section describes the platform-specific common information developed respectively for the two PLC platforms which were adopted for the experimental study. The templates for PLCOpenXML and Step 7 V5.4 were created according to the reverse engineering approach. The created templates were then finally imported into the database of VCMapper to be reused for direct deployment.

4.4.2.1. Templates for PLCOpenXML

For PLCOpenXML, an open standard, its structures are available on the official website of the PLCOpen organisation. To facilitate the accessibility of the templates data, a

PLCOpenXML file was divided into different constituent elements which are outlined in Table 4-6.

Table 4-6 Constituents of the PLCOpenXML common information (also called templates)

Name	Dynamic data	Description
GVL	Information of all the variables	Global variable list
Elementary Vars	The name, type and initial value of a variable	Elementary variable
MainProgram	None	Template for PRG_MAIN
Function InitArrays	The main body of the function	Function for initial control models
POU	Name, Type, Input variables, Output variables and Main Body of a POU.	Information of a program organization unit
POUInstance	Name, Type and Type name	Instance of a POU.
Task	Name, Interval and Priority	The main task of the control software
Configuration, Resources	Name	Standard items of IEC61131-3
Content header	Name and modification time	Header information of file content.
File header	Name, Version and Create Time	Header info of the file

All the elements of the PLCOpenXML templates are actually XML nodes as PLCOpenXML is completely XML-based. Therefore, they can be manipulated using the XML DOM when the dynamic data is populated. However, in this research, they were regarded as text string in the VC mapper and the dynamic data was populated by replacing the particular reserved symbols with project-specific data. This is mainly to unify the way of manipulating the different platform-specific templates considering that for some platforms, the common information is unstructured plain-text-based and can only be manipulated through the replacement of text.

Some example elements of the PLCOpenXML templates are given in Figure 4-16. As depicted in the figure, the project-specific dynamic data were represented using predefined keywords and will be populated during the deployment process.

Overall structure of PLCOpenXML

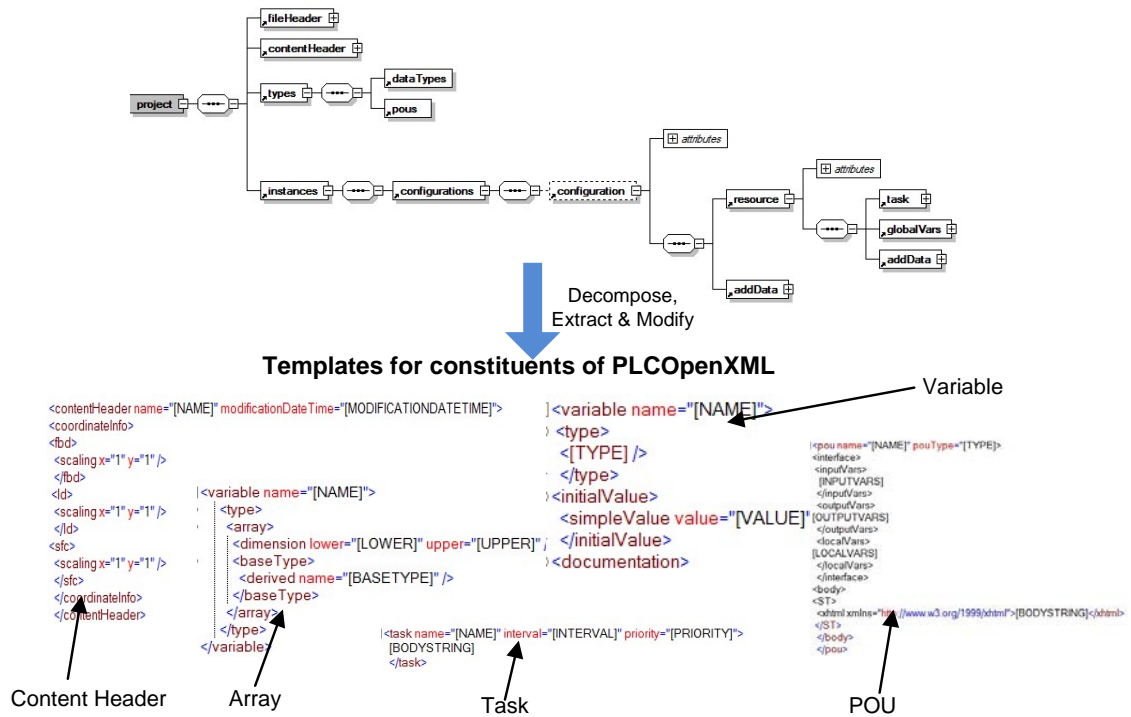


Figure 4-16 Examples of the PLCOpenXML templates

4.4.2.2. Step7-specific templates

Given the fact that the data format of Step 7 is completely platform-specific, the common information templates were created in the strict reverse way. Through analysing the source codes of existing projects, the constituents of the S7-specific templates were created and outlined in Table 4-7.

Table 4-7 Components of S7 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 Function Block
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 organization block of the project

It is worth mentioning that the source codes of the Step 7 are exported as completely unstructured plain-text based files. Therefore, each template can be only manipulated as a text string in the VCMapper. Taking the shared Data Block (DB) which contains the runtime control models for example, the header information of the shared DB for the test rig was generated by replacing the reserved words in the template illustrated in Figure 3-22.

```

DATA_BLOCK DB200

    TITLE = RuntimeControlModels

    VERSION : 0.0

    STRUCT

        IsaComponents : ARRAY [0 .. 32 ] OF "Component";

        IsaStates : ARRAY [0 .. 149 ] OF "State";

        IsaTransitions : ARRAY [0 .. 133 ] OF "Stransition";

        IsaAutoConditions : ARRAY [0 .. 162 ] OF "Acondition";

        IsaActComponents : ARRAY [0 .. 12 ] OF "ActCompPos";

    END_STRUCT;

BEGIN
    
```

Common information for the S7-specific objects including the instance data block (DB) for a FB instance, Organisation Block and symbol table were also created. For different objects, different key words were used to represent the dynamic data in the corresponding templates. Some of the objects are shown in Figure 4-17 and Figure 4-18 as examples.

```

ORGANIZATION_BLOCK "OB1"
TITLE = "Main Program Sweep (Cycle)"
VERSION : 0.1
VAR_TEMP
    OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
    OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
    OB1_PRIORITY : BYTE ; //Priority of OB Execution
    OB1_OB_NUMBR : BYTE ; //1 (Organization block 1, OB1)
    OB1_RESERVED_1 : BYTE ; //Reserved for system
    OB1_RESERVED_2 : BYTE ; //Reserved for system
    OB1_PREV_CYCLE : INT ; //Cycle time of previous OB1 scan (milliseconds)
    OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
    OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
    OB1_DATE_TIME : DATE_AND_TIME ; //Date and time OB1 started
    dummyflag : BOOL ;
    Separation : BOOL ;
    AutoMode : BOOL ;
END_VAR
BEGIN
    
```

Figure 4-17 Example of S7 specific objects – header of OB1

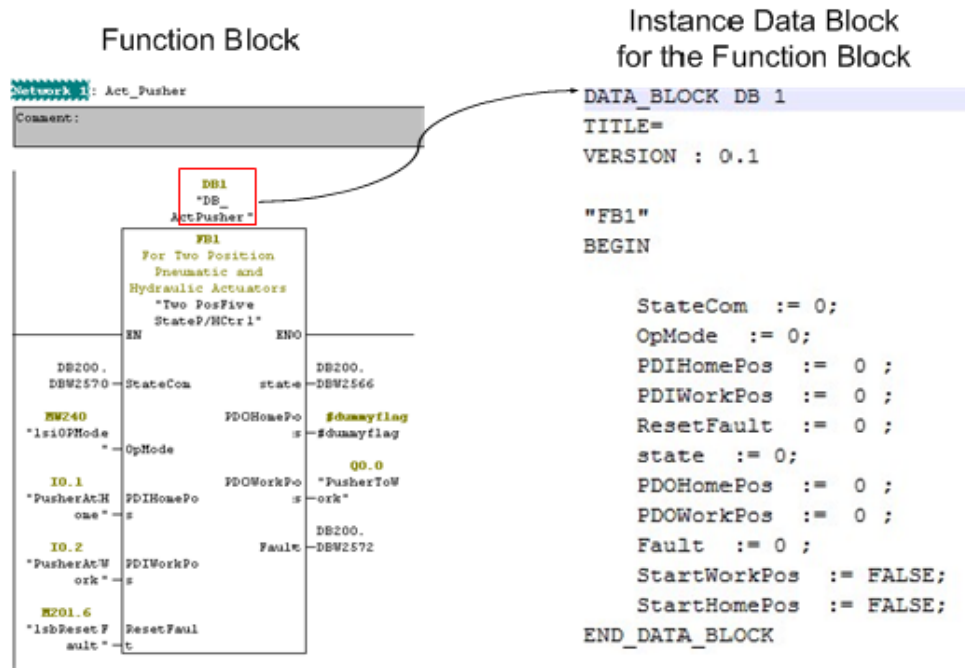


Figure 4-18 Example of S7-specific Object – Instance Data Block

4.4.3. Control software deployment

One of the intentions of proposing and implementing the direct deployment approach is to simplify the control software development process during the system engineering phase. With the VCMapper implemented in the Section 4.1, control codes for the two platforms were respectively developed following the same steps described below:

- (1) Create a new project: the platform, for which the control codes are generated, was selected from the drop down list of the logic mapper.
- (2) Import the CCE model: the storage path of the XML file containing the control logic of the virtual model of the test bed was specified.
- (3) Import I/O variables: variables were imported from external Excel files. For the PLCOpen and the Step 7, different variable tables were used due to the differences between the naming conventions of their I/O addresses.
- (4) I/O mapping for sensor and actuator components: as shown in Figure 4-19 - the snapshot of the I/O mapping, all the sensor components and actuator components were extracted from the XML file and listed out. Also, all the reusable runtime components in the database were listed out. For deployment of any PLC platform involved, the I/O mapping was done by selecting a component and its corresponding runtime component from the list and adding the I/O variables to the selected runtime components. For

sensor components, only an output variable was added. For actuator components, the number of input variables and output variables of a component varies.

- (5) Generate the code by clicking a button and then automatically export the code into files of data formats of the selected PLC platform.

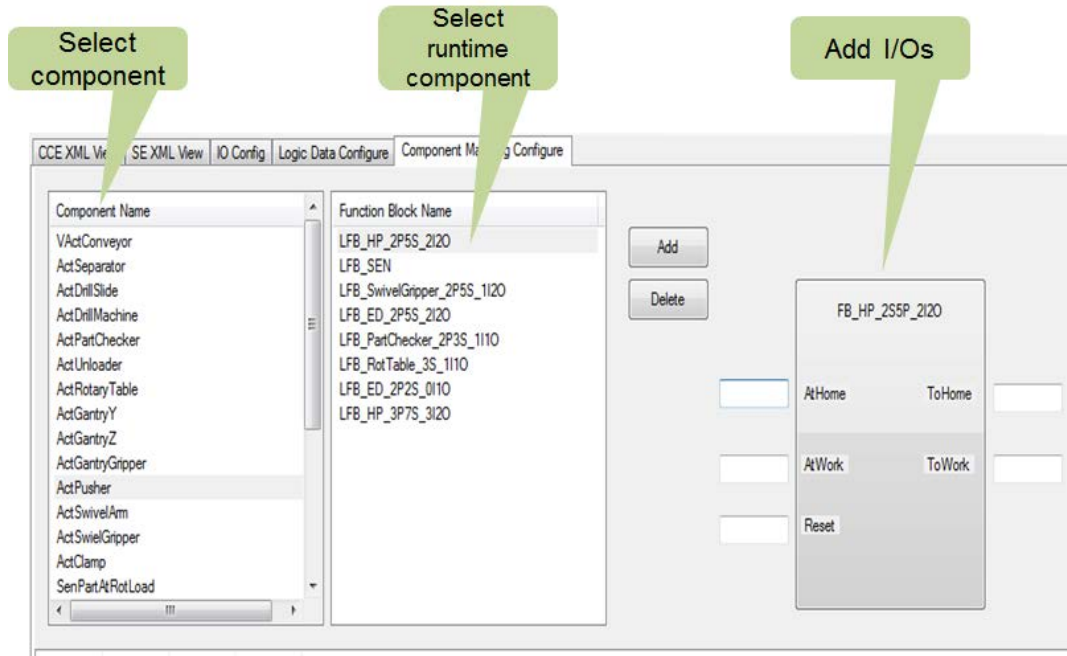


Figure 4-19 I/O Mapping for actuators and sensors

For the CodeSys platform, all the source codes were exported into a single XML file according to the data structure of PLCOpenXML.

For the Siemens platform, on the other hand, the source codes of different items were exported into multiple plain-text files. The information of exported files was outlined in Table 4-8. As shown in Table 4-8, different items were 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 be compiled in a specific order as listed in the table due to the dependency between these items. For example, the instance DBs must be compiled after the compilation of FBs as all the variables in an instance DB are dependent on the variables of a specific FB.

Table 4-8 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

4.4.4. Commissioning

The generated control software was directly used to commission the physical test bed. In this step, the PLC programming engineering tools were mainly used to configure the hardware, compile the generated source code to build executable program and download it onto the PLCs.

4.4.4.1. Commissioning with CodeSys SoftPLC

The environment set up for commissioning the test bed using the PLCOpenXML control code is depicted in Figure 4-20. The CodeSys V3, which is developed by 3S-Smart Software Solutions GmbH and supports PLCOpenXML, was adopted as the programming software for compiling and downloading the generated control code. In terms of the control devices, the CodeSys Control RTE V3, which is a SoftPLC provided by the 3S-Smart Software Solutions GmbH, was adopted as the controller and the Advantys OTB (Optimised Terminal Block) I/O modules were used. The SoftPLC device can run on a PC and communicates with the OTB I/Os via Ethernet network based on the ModBus TCP protocol [128].

Configuration of the SoftPLC was performed following the importing the generated source code. The connections to connect the master device (the SoftPLC controller) and slave devices (the OTB I/Os) were built manually according to the requirements of the ModBus TCP. The IP addresses of the slave devices were configured first. Additionally, the virtual addresses of all the I/O variables were allocated and mapped to corresponding ModBus TCP Slaves. Two ModBus Slave channels were created first by setting their respective modes of accessing and offset addresses. Then the I/O variables were allocated to the corresponding channels.

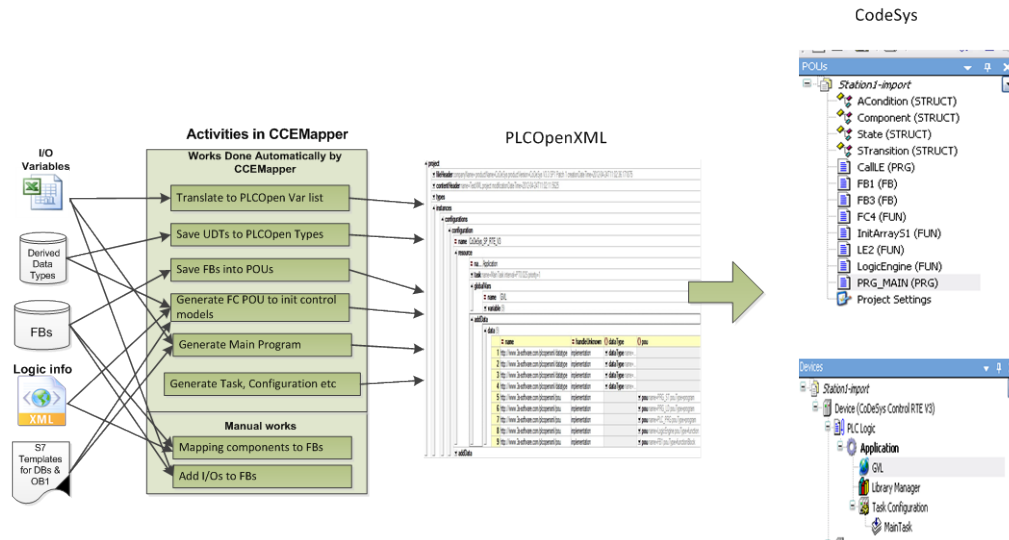


Figure 4-20 Commissioning environment for PLCOpenXML

The CodeSys V3 does not support importing of control code programmed in graphical languages. Hence, all the objects were generated based on textual languages – Structured Text. As shown in the above figure, all the POU and the Device Objects were automatically created after the generated code was imported into the CodeSys engineering tool.

Some key POU which were automatically generated can be seen in the programming tool. Of all the POU, the function POU “InitialControlModels” and the program POU “PRG_MAIN” were generated dynamically based on the virtual control models while all the other POU were generated by combining reusable data. The function “InitialControlModels” was created for initialising the Arrays of control models. The program “PRG_MAIN” was generated as the main program which contains runtime control codes for each resource components. As an example, the runtime control program for the actuator Swivel Arm is illustrated in Figure 4-21. Also, part of the source code of the function “InitialControlModels” was shown in this figure.

Of all the Device Objects, the GVL which refers to the Global Variable List was generated by transforming the I/O variable forms. The object “Main Task” was generated dynamically by calling the PRG_MAIN.

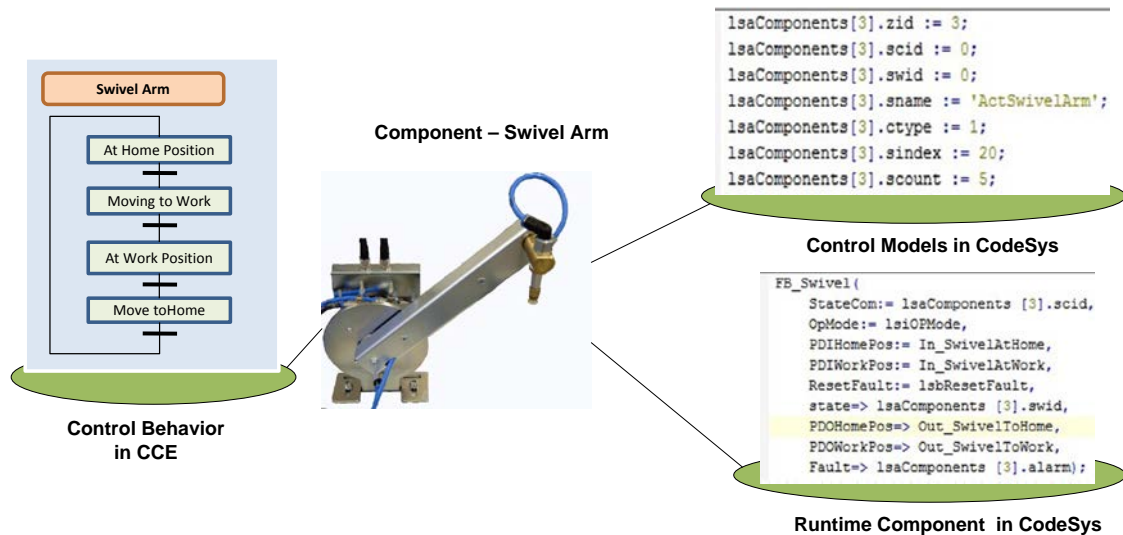


Figure 4-21 Component Swivel Arm and its related control code in CodeSys V3

With the CodeSys platform, the test bed was commissioned under the automatic-mode. The result of the experiment indicated that the test bed was running correctly under the control of the control software generated by the VCMapper.

4.4.4.2. Commissioning with Siemens S300

The test bed was commissioned using Siemens control devices under both automatic mode and manual mode. The Siemens S300 PLC with distributed I/O modules was chosen as the hardware controller. The programming tool used is the Siemens SIMATIC STEP7 V5.4. For commissioning of the manual control, we adopted the Siemens HMI devices and the WinCC as the HMI engineering software. The adopted Siemens devices communicate through Profinet.

As aforementioned, Siemens PLC programming tools normally adopt proprietary data formats and include some S7-specific objects. After the generated source codes with the specific data formats were imported into the S7, these files were compiled in the specific order as pointed out in section 4.4.3. By compiling in this order, the UDTs (User Data Types) were generated first, followed by the shared DB which contains instances of UDTs. Since the Logic Engine uses the data of the Shared DB, the FBs and FCs were then generated after the shared DB was available. The Organisation Block was generated at last as it contains the instances of the function blocks and the data of the shared DB. The overall process of deploying control software for Siemens Step 7 is illustrated in Figure 4-22.

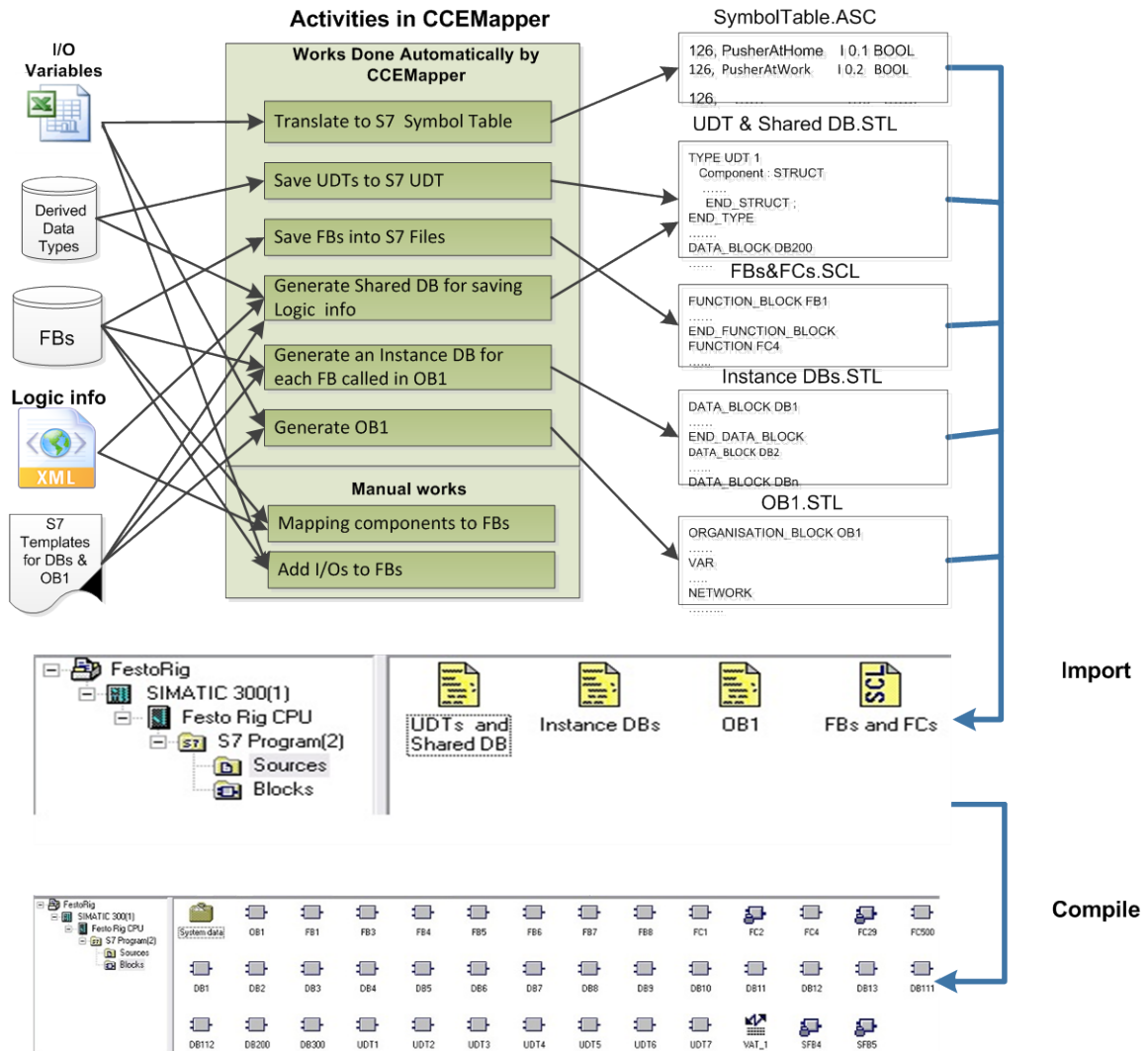


Figure 4-22 Process of S7 control software deployment

In Step 7 the main program created from compiling the generated source code can be viewed in different languages. As an example, the source code for the runtime control of the actuator component “Pusher” can be viewed as a FBD instance in the Step 7. A runtime control program, which is called Network in Step 7, was generated for each actuator or sensor component and finally the Logic Engine was also called by the OB. Twenty-nine FB/FC instances in total were generated in the OB for the test bed. An example was shown in Figure 4-23.

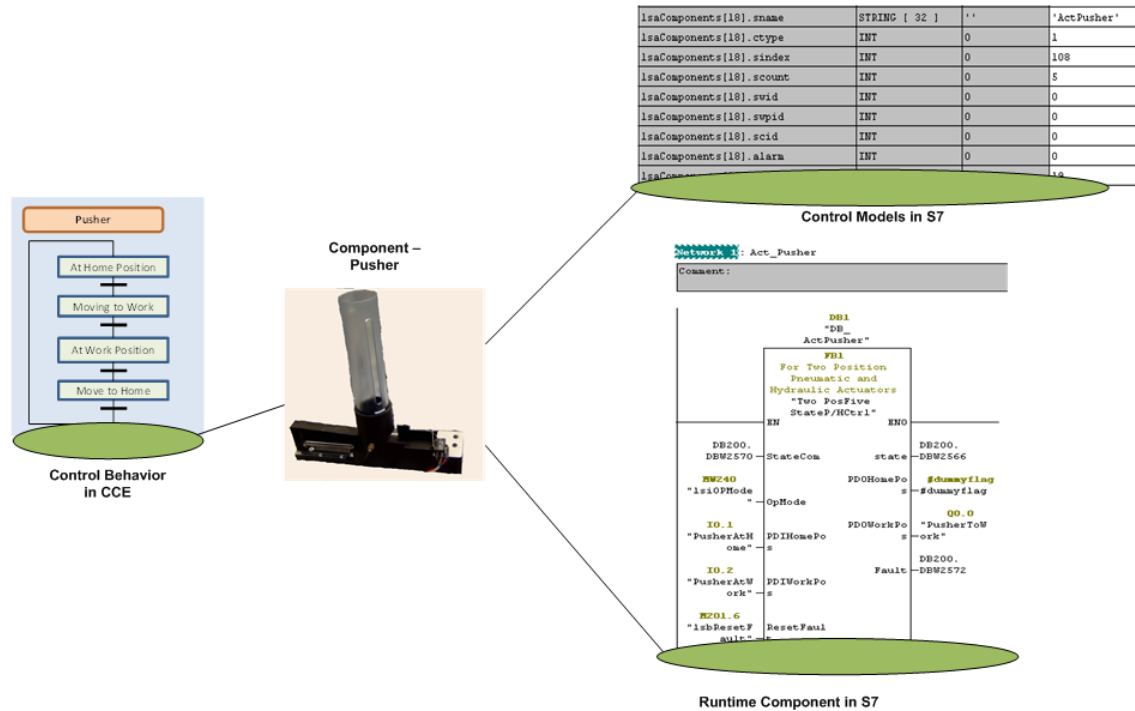


Figure 4-23 Component Pusher and its related control code in Step 7

Commissioning under Manual mode

The test bed was also commissioned under manual mode. The generated control software contains the functions for both automatic mode control and manual mode control. For manual mode control, a HMI generator application was developed by other ASG researchers [9, 10].

The main functions of the HMI generator are to 1) generate the HMI screens during runtime based on the control models contained in the PLC control software, 2) generating commands according to the operations on HMI and sending the commands to PLC via the control models, and 3) report and record fault messages.

A script program was first developed to be run on the WinCC as an HMI generator. The HMI control models for describing HMI rows and the fault message descriptors were automatically generated by the VCMapper and included in the shared DB of S7 PLC. During the runtime, when the HMI is opened, the HMI generator gets the HMI control models from the share DB and displays them as HMI rows. As illustrated in Figure 4-24, the HMI rows which were generated by transforming the control behaviours of corresponding components were displayed on the HMI. Other snapshots of the HMI are also shown in Figure 4-24.

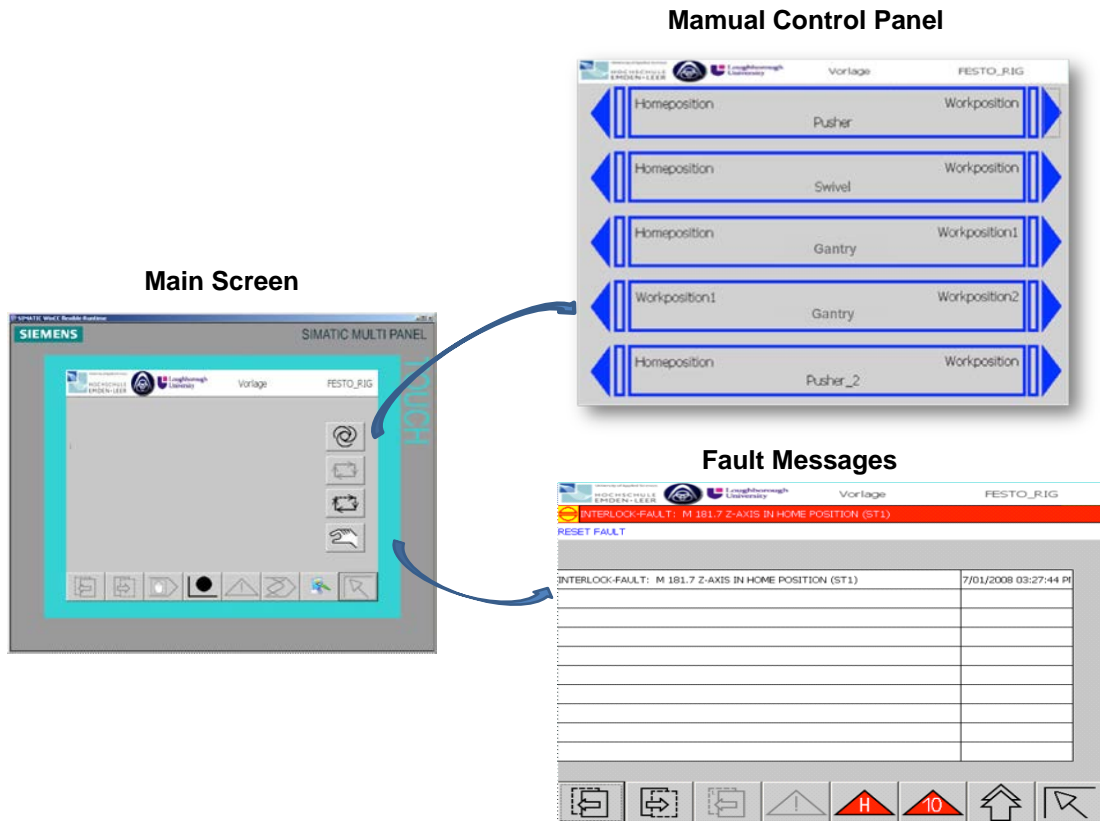


Figure 4-24 Screens of generated HMI

During the commissioning under the manual control mode, the PLC exchanges data with the HMI using the share DB. Here we took the component Pusher as an example to demonstrate the working process of manual control. When the button “Working Position” on the HMI for controlling the Pusher to move to its working position was pressed, the command ID was sent to the corresponding manual-mode control model in the share DB. The Logic Engine detected the data update and updated the corresponding control model which connects with the instance Runtime Component of “Pusher”. The instance Runtime Component of “Pusher” then received the command “1” from the Shared DB and output the command to the I/O variable “Pusehr_ToWork” to drive the “Pusher” to move to its working position.

The data for saving fault messages were also generated and saved in the share DB. The fault messages generated by the Runtime Components will be displayed by the HMI. When a fault happens in a component, the corresponding runtime component generates an error message and output the ID of corresponding fault message to the shared DB. The HMI engine

then gets the fault message from the DB and displays it on the HMI. Also, the historical fault messages stored in the shared DB can be retrieved to display when needed.

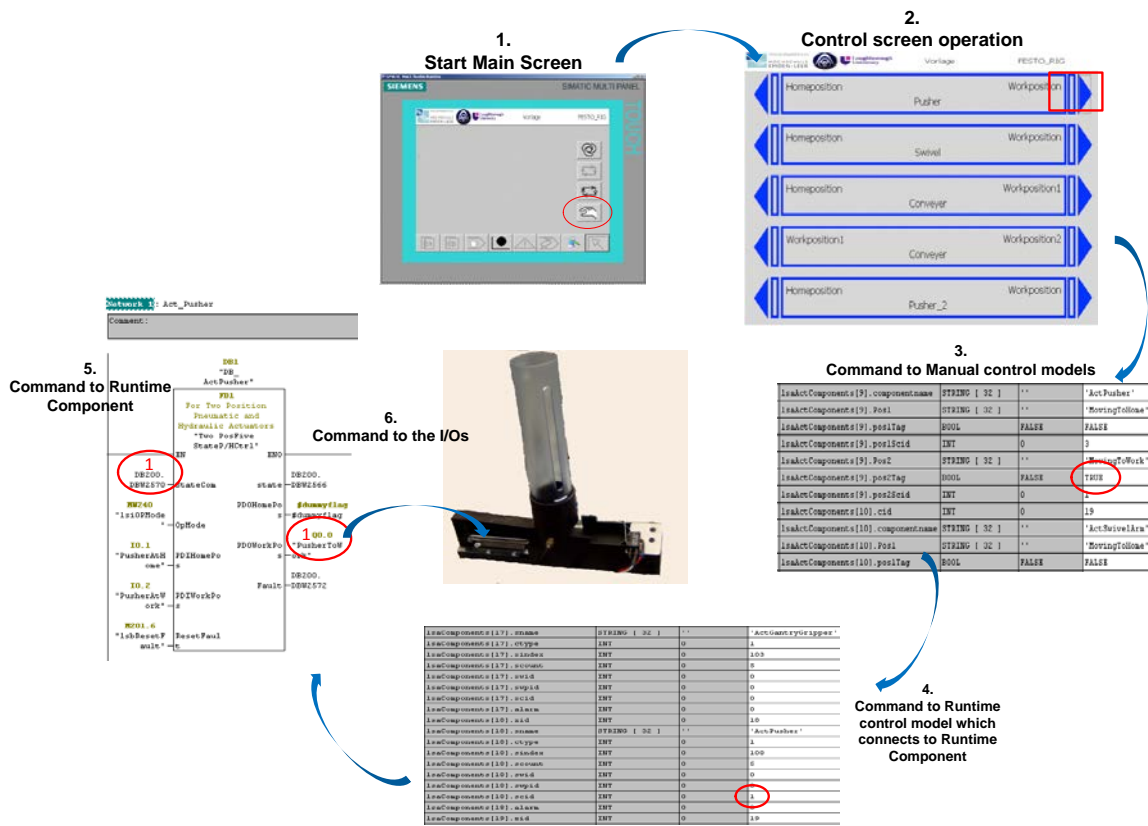


Figure 4-25 Work flow of manual control via HMI

4.4.5. Evaluation

The experiments of commissioning the test bed with the directly deployed control software have offered several cases of interest to illustrate the functions of the proposed direct deployment approach. However, it does not sufficiently reveal the qualitative and quantitative advantages and limitations of the approach. Likewise, the features of the control software generated by the proposed approach have not been sufficiently illustrated. In view of these limitations, a set of complimentary experiments was carried out to evaluate the proposed approach through comparing it with a traditional PLC programming approach from several key perspectives.

For the emerging approaches to control system engineering, to compress the required time and effort needed during the machine engineering phase has been regarded as a key objective. It is also essential, for any control engineering approaches, to guarantee that the developed control software meets the requirements of machine operation. Hence, experiments were

conducted to evaluate the direct deployment approach as well as the control programs deployed by this approach.

The evaluation experiments were also carried out on the same test rig. In order to make a comparison, another set of control software of the test rig was programmed manually. The comparisons were carried out both during the control software development process and during the machine operation phase.

4.4.5.1. Evaluation on control software development

This section reports a series of additional experiments as well as analyses on the experimental results, both of which were conducted to make a comparison between the direct deployment approach and one of traditional methods in terms of the following aspects:

- Time required to develop the control software.
- Time required to reconfigure existing control software.

In order to make a comparison, a control engineer, who is able to perform both VC with CCE tool and manual PLC programming, was involved in the experiments. This is mainly to minimise the effects of human factors lying in the differences of personal skills. In each selected scenario, the control engineer developed the required control software respectively using direct deployment approach and the manual programming approach. To be more specific, on one hand, the control engineer carried out component-based simulation of the test rig with the CCE tool and then directly deployed the control software based on the virtual models and the Runtime Components which have been described in section 4.4. On the other hand, the engineer manually programmed the sequence control Function Blocks of the control software using a programming language similar to SFC.

A. Time to Develop

Time to develop the control software using the proposed direct deployment approach was first measured. The Siemens S7-300 PLC and STEP 7 programming software were adopted for this experiment. The process of the direct deployment approach was decomposed into more detailed sub-processes in order to record and analyse the time of each sub-process. The process began with virtually prototyping the test rig. However, given the fact that the virtual prototype is also usable for other virtual engineering activities, virtual prototyping is not

counted as part of the control software development. In this experiment, time required to perform the following sub-processes were counted:

- Runtime component development
- Control logic editing in the CCE tool
- Virtual commissioning of the test rig
- I/O mapping for control software deployment
- Real commissioning

The process of develop the control software using the selected traditional approach can be decomposed into:

- Developing function blocks for resource components
- Developing SFC function blocks for process control
- Developing Organisation Blocks
- Real commissioning.

The function blocks for resource components are the same as the runtime components used in the direct deployment approach. The differences of development time are mainly decided by the time required to manually develop other part of the control software and correct the errors through commissioning. The time to program the mentioned items and carry out the following real commissioning were recorded in Table 4-9.

Table 4-9 Time to develop control software of test rig using two select approaches

	Direct deployment	Manual programming
VC logic editing	1 day	-
Virtual commissioning	1 day	-
Sequential FBs programming	-	1 day
Program development	2 hours	2 days
Real commissioning	2 hours	2 days
Overall time	2.5 days	5 days

Time saving can be observed from the comparison. The direct deployment approach compressed the time to develop the control software as the development of system-specific program was automated. Commissioning time was also significantly reduced by the virtual

commissioning. In the direct deployment process, time to real commissioning was significantly reduced as control logic had been validated by virtual commissioning. It took much less time, compared to that of real commissioning, to develop simulated control logic and perform virtual commissioning as it was carried out in a fully computer-based environment, which makes commissioning simple to perform. It worth mentioning that for large scale machine, the advantages of virtual commissioning will be more significant as the manual programming and real commissioning need much more time, efforts and investment.

B. Re-configurability

Based on the control software developed in the above section, the respective re-configurability of the direct deployment approach and the traditional approach was also compared. Two scenarios, in which it needs to reconfigure the existing control software to develop the desired control software, were considered. The two scenarios are illustrated in Figure 4-26.

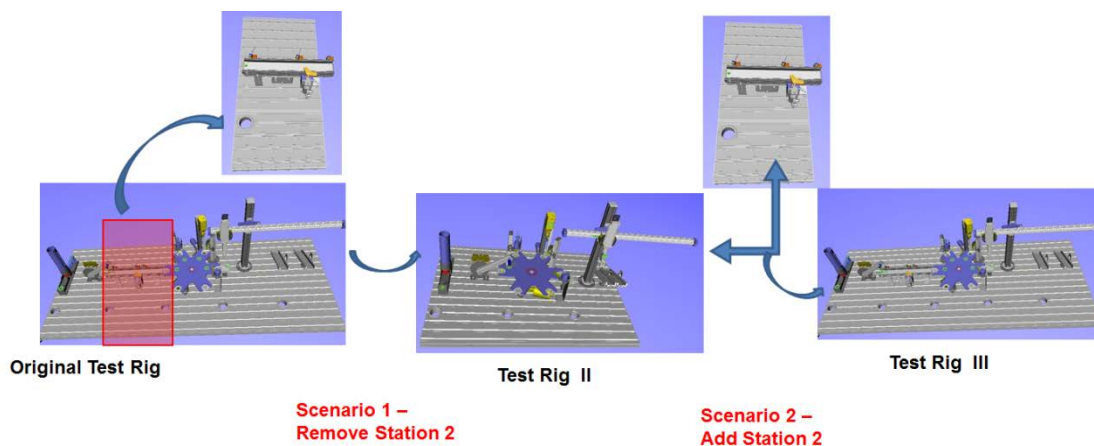


Figure 4-26 Scenarios of reconfiguring the test rig

Scenario 1- Remove components

The scenario of reducing the number of components of the test rig was first considered. The station 2, which consists of two actuators and two sensors, was removed from the test rig. As shown in Figure 4-26, after removing the station 2, the work piece is transported from the Swivel Arm directly to the Rotary Table.

For the manual programming approach, the reconfiguration was performed in the Step 7 through modifying related program. The following activities were performed to complete the reconfiguration:

- Modifying the SFC FBs related to the Swivel Arm, the Vacuum Sucker, and the Rotary Table.
- Deleting both SFC FBs and Runtime Components related to the components of Station 2.

The whole process, as well the additional consequent commissioning, took around 3 hours.

For the direct deployment approach, on the other hand, the reconfiguration was carried out in the CCE toolset. The new system was built through removing the related components and modifying the transitional conditions of state behaviours of relevant components which include the Swivel Arm, Vacuum Sucker and Rotary Table.

After virtual commissioning, the consequent reconfiguration to the PLC program was then completely automated by the VCMapper within the following step:

- Load the existing project files of the original test rig.
- Compare it with the control models of Test Rig (II).
- Delete the I/O mapping information of the removed components
- Generate new runtime control models according the new virtual models.
- Output the new control code.

It can be seen in this scenario the modification of the PLC control software was completely automated by direct deployment approach. The time taken (around 1 hour) lies in reconfiguring the virtual models and virtual commissioning.

Scenario 2- Add components

In the other scenario, the Station 2 was added into the Test Rig (II). The relevant components which need to be modified in this case are mainly the same as in the first scenarios. However, there are still some differences from the first scenario.

In manual programming approach, the reconfiguration was to add new programs for the components of Station 2 and modify the SFC FBs that relate to these components. It took around 4 hours to complete this.

For the direct deployment approach, the process of reconfiguring virtual models and virtual commissioning took around 1.5 hours. During the control deployment process, although the

process of updating the information of control models was automated, the I/O mapping for the new components of station 2 were required to be performed manually and took 0.5 hour.

Table 4-10 Time to reconfigure the test rig

Scenario	Manual Programming	Direct Deployment
Remove components	3 hours	1 hour
Add components	4 hours	2 hours

It can be observed, from the results of the above experiments, that the direct deployment approach reduced the time required for reconfiguration. This was partly achieved through the automation of the program generation. Especially in the scenario in which no new components were added, the reconfiguration of the PLC program can be completely automated thereby required time can be compressed. Additionally, the simplification of the logic editing and commissioning by the virtual commissioning tool also contributed to the time savings. Obviously, re-configurability can also be evaluated in some other scenarios which need the reconfiguration, such as changing the position of certain components. The direct deployment approach can also be evaluated in more complicated scenarios.

4.4.5.2. Evaluation on control software developed

This section reports the evaluations on the control software which was generated by the direct deployment approach in order to reveal the advantages and limitations of the auto-generated control software. Considering that the control software developed by traditional approaches have been proven through usage to be mature and acceptable by industry, this evaluation was performed through making comparisons between the auto-generated control software and the traditional control software.

Existing relevant literature has proposed different approaches to evaluating PLC control software from various perspectives. These approaches are proposed for the evaluation of certain specific languages. The software quality model of ISO 9126, which is mainly used for computer software evaluation, was adopted to evaluate the complexity of PLC software [129]. Some methodologies of measuring the complexities and accessibility of PLC programs written in specific languages were also proposed by Lucas et al. in [130]. However, the methodologies are proposed to measure the programs written in ladder diagrams, Petri nets

and modular finite state machines. Comparisons of specific programming languages were also conducted. A comparison of SFC and Object-Modelling PLC programming was reported by Hajarnavis and Young [131]. Apart from the time and efforts required for the two selected programming methods, the performances of the executing programs, including processor scan time and occupied memory, were also compared. Unfortunately, no concrete methods of measuring the programs quantitatively were given in this literature. Hajarnavis and Young also reported the main benefits of interest for the industries concluded from a survey of control engineers.

Based on the evaluation approaches from existing relevant literature as well as the information which other ASG researchers have drawn from interviews with industry, the control programs were evaluated from several selected perspectives described in detail in the following sub-sections. The evaluation was carried out based on the programs developed in section 4.4.5.1 and via additional experiments when necessary. The auto-generated control software is represented in the languages of ST and FBD, while the manually programmed software is programmed in ST and SFC. In the following sub-sections, the memory performance and time performance were first measured with additional experiments, through which quantitative results were given. Then other performance related to the purpose of operations and maintenances was evaluated qualitatively. The evaluation was also performed based on the Siemens Step 7 and S-300 PLC.

A. Memory required

The respective structure and required memory for loading and running the control software were compared, as outlined in Table 4-11.

The memory occupied by the control software can be classified into load memory and work memory. The load memory is for storing all the information of the control software when the project is downloaded to the PLC. The work memory is permanently located on the CPU. It is optimised for high-speed access, and at start-up the PLC copies the parts of the load memory necessary for program execution from load memory to work memory. It is common that the amount of work memory occupied will be less than the amount of load memory.

The differences in the code structure can be seen from Table 4-11. Obviously, the load memory that the auto-generated program takes is only half of that taken by the software

manually developed. This is mainly because that the former was programmed in the textual language ST while some FBs of the latter were in graphical language SFC. The differences lying in the adopted language were also reflected by the work memory which the code occupies. The data memory of the auto-generated program is bigger than that of the manual-developed program as the former contains a data block for saving all the runtime control models. The manual-developed program uses SFC function blocks for the sequential control. Hence, it has more FBs and instance DBs than the auto-generated program.

Table 4-11 Comparison of PLC programs

	Direct deployment (ST+FBD)	Manual Programming (SFC+FBD)
Load memory	24378 bytes	48462 bytes
Work memory/Code	8460 bytes	20094 bytes
Work memory/Data	13150 bytes	6964 bytes
Work memory/Total	21610 bytes	27058 bytes
System data memory	12170 bytes	12170 bytes
OB	1	1
DB	17	24
FB	8	15
FC	2	1
SFC	0	5
UDT	4	0

In summary, from this example, it can be seen that the component-based control software requires less memory, either load memory or work memory, than the traditional program due to the differences lying in the code structure and programming languages adopted. It is still hard to say that the former is definitely better than the latter in terms of the memory required. However, at least it indicates that the memory that the component-based control software requires is in a reasonable range which is around 50% smaller in load memory and 20% smaller in work memory than the manual equivalent for this test case.

B. Time Performance

The performance of the control software was evaluated by measuring the process scan time. For each program, a function block for recording and calculating the scan cycles was added

into OB1. During the runtime of the test rig, the maximum scan cycle and the minimum scan cycle were respectively recorded and the average scan cycle was also calculated.

The results of the experiment were shown in Table 4-12. It can be seen that the component-based control software has obviously better performance than the traditional software. It can be analysed that two potential factors contribute to the better performance of the component-based control software.

Table 4-12 Comparison of scan time

	Directly Deployed	Manual Programmed
Max scan time	8ms	19ms
Min scan time	<0.5ms	<0.5ms
Average scan time	3ms	14ms

First of all, the way of generating control commands in the component-based control software is faster than that in the traditional program. In the component-based control software, all of the control logic information is modelled as runtime control models and saved in the same data block. In each scan cycle, the Logic Engine scans these runtime control models in order and then generates commands to corresponding runtime components. In the traditional program, on the other hand, the command for controlling a specific component is generated by the corresponding SFC function block which might still need to communicate with other sequential control function blocks to finally generate the command. This can potentially increase the scan time.

Another possible reason potentially lies in the differences of the programming languages used. As aforementioned, the program for sequential control in component-based control software was automatically in textual language ST while the FBs for sequential control in the traditional program were programmed in SFC. Although no research work has been done to test the performance differences between existing PLC programming languages, the language difference can still be seen as a potential factor to the performance difference.

C. Diagnostics and Debug

Diagnostics refers to finding the source of a failure in a system, while debug is aimed at correcting the program errors. This section is mainly to evaluate the effort required to

diagnose and debug the directly deployed control software during the machine operation phase.

The process of diagnostics can vary depending on the cause of the fault. Diagnosis is needed after a failure is detected when an expected event in the controlled process is missing, e.g. an actuator should do something, but is not doing it. Manual diagnosis normally starts by checking the hardware actuator responsible for the missing action. If the actuator is found to be working correctly, the reason is then tracked back to the PLC program. Normally, the dependency of the output signal corresponding to the failing actuator is tracked back, possibly over several internal variables of PLC, to the corresponding input signals, is first tracked. If the reason still has not been found out finally, the algorithm of related control program, which can be a function or a function block, needs to be checked.

Several scenarios, in which the causes of faults are different, were considered in this section.

Scenario 1 - Fault in I/O Modules

For the first step of PLC program diagnosis, the relevant I/O variables connected to the failure actuator were checked.

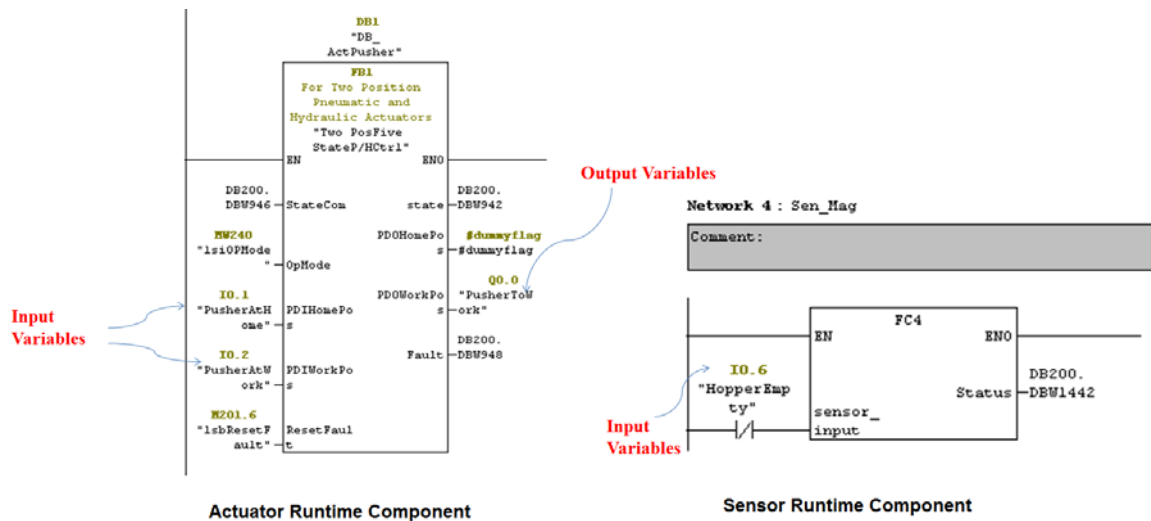


Figure 4-27 Runtime Component (an actuator and a sensor)

In the component-based control system, there is a one-to-one correspondence between physical components and runtime components in the PLC program. As shown in Figure 4-27, all the variables connected to an actuator or sensor component correspond to the variables

connected to the runtime component. It is intuitive and simple to check whether there is fault in the related I/O module.

For the control program which was manually programmed using SFC and FBD, the process of tracking the I/O variables is the same.

Scenario 2 - Error in Runtime Component

If the related input variables of the runtime components are found correct while its output variables are not set to the correct value, the diagnosis needs to track into the Runtime Component.

In this case, the process of diagnosing and debugging the component-based control software is the same as that of traditional control software. The runtime component 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.

For the direct deployment approach, extra work is required after debug. As the component-based control software is deployed automatically through reusing of runtime components, modifications to the runtime component must be updated to the runtime component library as well. Therefore, the new runtime component still needs to be re-downloaded into the database of the VCMapper. However, this greatly aids the reuse and the maintainability of the PLC code.

Scenario 3 - Error in sequential control

If it is found that the failure is caused by the wrong command to the runtime component, it is likely that the cause of the fault lies in the runtime control models. For example, if the “Magazine Sensor” is ON and the “Magxifer Sensor” is OFF, the state command to the actuator “Pusher”, which should be “1”, is till “0”. This means the control logic of the actuator “Pusher” is not correct. In this case, the control logic for the sequential control needs to be diagnosed and debugged.

For the traditional PLC program in this experiment, the FBs for sequential control, which is programmed in SFC, can be directly checked, and modified if necessary, in Step 7. This process is also applicable to other manually developed PLC programs as well.

On the other hand, diagnosis and debug of the component-based control program, in this case, is totally different as it has to be performed in the VC tool rather than in the PLC programming tool. This is mainly due to two reasons which are analysed as follows.

First of all, the principle of the proposed open VC framework is to achieve the advanced development and validation of control logic in a virtual environment. In other words, it enables control engineers to start developing and validating control logic earlier using a 3D simulation toolset rather than PLC programming tools. Hence, control logic information should be modified in the VC toolset. This also ensures complete data consistence between the virtual models and the control program.

Secondly, the runtime control models are described in a machine-understandable manner, but not readable for control engineers. Pieces of the runtime control data models are illustrated in Figure 4-28 as an example. The control logic information, in the direct deployment approach, is contained in the runtime control models which are represented as arrays of user-derived data types (UDTs). This improves the real-time performance of the program and reduces the memory required. However, it also reduces the readability of the code and makes the code impossible to be directly debugged by control engineers.

2522.0	lsaComponents[17].alarm	INT	0	0
2524.0	lsaComponents[18].zid	INT	0	18
2526.0	lsaComponents[18].sname	STRING [32]	''	'ActPusher'
2560.0	lsaComponents[18].ctype	INT	0	1
2562.0	lsaComponents[18].sindex	INT	0	108
2564.0	lsaComponents[18].scount	INT	0	5
2566.0	lsaComponents[18].swid	INT	0	0
2568.0	lsaComponents[18].swpid	INT	0	0
2570.0	lsaComponents[18].scid	INT	0	0
2572.0	lsaComponents[18].alarm	INT	0	0
2574.0	lsaComponents[19].zid	INT	0	19
2576.0	lsaComponents[19].sname	STRING [32]	''	'ActSwivelArm'
2610.0	lsaComponents[19].ctype	INT	0	1
2612.0	lsaComponents[19].sindex	INT	0	113
2614.0	lsaComponents[19].scount	INT	0	5
2616.0	lsaComponents[19].swid	INT	0	0

Figure 4-28 Control models represented as arrays of UDTs (difficult to read)

A summary of the respective efforts to diagnose and debug the component-based control software and the traditional control software was given in Table 4-13. Generally speaking, when failures are caused by hardware faults, the component-based control software provides a good diagnosability; on the other hand, if failures are caused by errors lying in control logic, the diagnosis and debug of the component-based control software can only be performed in

the relevant VC engineering tool while the traditional control software can be directly diagnosed and debugged in the PLC programming tool. However, directly debugging in the PLC programming tool may cause problems due to ad-hoc and uncontrolled editing of the logic. If errors exist in the runtime component, the debug work in either approach requires almost the same amount of efforts.

Table 4-13 : Comparison of efforts to diagnostic and debug

	Directly Deployed	Traditional Program
Error in I/Os	Check one RC only	Check one FB or multi FBs
Error in Function Blocks	1.Debug in PLC programming tool 2.Update RC library	Debug in PLC programming tool
Error in sequential controls	1.Debug only in VC tool 2.Update the control software in VC	Debug in PLC programming tool

The control software can be further evaluated from various other perspectives depending on the requirements of given industries. The aspects evaluated and discussed in this section can also be further evaluated in a quantified way in terms of the exact time and effort required. To achieve a comprehensive and in-depth evaluation, reasonable and applicable evaluation methods must be available. However, to propose and design such evaluation methods is beyond the scope of this research.

4.5. Summary of Chapter

This section first presented the development of the VCMapper, an engineering tool which implements the proposed approach to virtual model mapping and direct deployment of PLC control software. At the time of writing, the following functions have been implemented in the VCMapper:

- Mapping component-based virtual models of component-based automation to the virtual modular common data models (VMCDM) described in AutomationML.
- Directly deploying control software for PLC platforms which support PLCOpenXML.
- Directly deploying the control software for Siemens Step 7.

Experimental studies to test and evaluate the above functions were then presented. A test rig was used to conduct the experiments. Through the experiments, the feasibilities of the proposed approaches were proven and demonstrated. The approaches were also evaluated from some selected aspects. Several advantages and limitations can be observed through the evaluation. Some features of the proposed approaches have not been demonstrated and more experiments for further evaluation are needed in the future.

4.5.1. About the VMCDM and virtual model mapping

The experiments have partially explored the capabilities of the proposed common data model and the efficiencies of the data exchange approach based on the common data models.

The experiment of transforming the virtual models of the test bed from the CCE tool to the VMCDM, and viewing the transformed virtual models in the AutomationML Editor highlighted and demonstrated:

- The semantics that the VMCDM presents. The object-oriented architecture and the vocabulary of concepts provided by AutomationML are the basis of achieving semantics and efficient data reuse.
- The feasibility of describing virtual models of component-based automation using AutomationML. Building the VMCDM by embedding the component-based automation into the framework of AutomationML leads to the achievement of efficient data reuse.

However, the experiment did not shown the capability of the common data models in supporting lossless data exchange of data models used in virtual commissioning. This is mainly because that the AutomationML, as a newly released open standard, has not been supported by other commercially available engineering tools. Further experiments are needed to show that other engineering tools can directly reuse the data and efficiently rebuild relevant functionalities based on the common data models.

4.5.2. About the direct deployment solution

Through experiments conducted on the selected test rig, the following features of the proposed direct deployment solution were demonstrated:

- Directly deployment of 100% executable PLC control software. This means that, for the generated source code, no error-prone manual programming is required. The generated code contains functions for both automatic mode control and manual mode control.
- Reducing the required time, effort and expertise to control software development and reconfiguration. The direct deployment approach automates most of the work in control software engineering. It also unifies and simplifies the method of I/O mapping for different PLC platforms.
- Generating well-structured control software with better memory performance and time performance. The generated control software occupies less PLC memory and executes with a shorter scan time. It also has reasonable diagnosability.
- VCOM engineering tools are required for diagnosis under some circumstances. The control models in the generated control software are difficult to understand directly by control engineers and thus the diagnosis of the control logic can only be performed via the VCMapper if errors exist in the runtime control models.

In summary, the conducted experiments have proven the feasibility of the direct deployment approach and also shown some of the advantages and underlying limitations of the approach. Further experimental studies need to be performed to test and evaluate the approach from more perspectives and with more realistic machines.

Chapter 5. Conclusions

This chapter concludes the research work documented in this dissertation through summarising the achievements to date, identifying the contributions and pointing out potential future research work.

5.1. Conclusions

The overall objective of this research is to develop the approaches and related engineering tool functionality required to enable the new component-based VC framework - VCOM. The research objectives identified in Section 1.3.3 were listed as:

- To design a new approach to directly deploying PLC control software based component-based virtual models and reusable components.
- To develop related engineering tool features to implement the new direct deployment approach to generate PLC control software for PLCs from different vendors.
- To create open standard-based common data models for describing component-based virtual models in a tool-independent method.
- To develop engineering tool features for mapping the tool-specific component-based virtual models into the created common data models.

Section 3.5 presents the design of the direct deployment approach. The proposed approach generates complete PLC control software in an automated manner through combining reusable data with dynamically generated runtime control models. The PLC control software generated by this approach contains the functions required for automatic-mode control, manual-mode control and diagnostics.

Section 3.2 outlines the design approach taken to create the needed common data models - VMCDM. The VMCDM was built based on AutomationML - an open standard specific to the domain of automation.

The design of the needed engineering tool - VCMapper which implements both the direct deployment approach and the approach to mapping virtual model into the common data models is presented in Section 3.6. According to the design, Section 4.1 presents the implementation of the VCMapper. In the current VCMapper, the direct deployment of control software for two PLC platforms – PLCOpenXML and Siemens Step 7, has been implemented.

The conducted experiments validate the hypotheses of this research outlined in Section 1.3.1. The experiments presented in Section 4.4 proves that the hypothesis “*if an automation system has been virtually prototyped and commissioned using the component-based approach, its PLC control software can be directly deployed based on the virtual model and reuse components*” is correct. The PLC control software generated by the VCMapper was directly used, without any manual changes, to run the physical test rig under both automatic mode and manual mode. The experiment described in Section 4.3 proves the other research hypothesis “*component-based based virtual models validated by VC can be directly reused by other engineering tools if these models are described using an open standard*”. The virtual models of the test rig described using the VMCDM created in this research were directly used by the AutomationML Editor – a standard engineering tool provided by the AutomationML Organisation for viewing and editing AutomationML models.

The strength and potential weakness of the methods developed in this research were also revealed via the conducted evaluation experiments. The experiment presented in Section 4.3.5 shows that the component-based virtual models described in VMCDM can be directly and correctly reused by a standard engineering tool. However, the characteristic method of describing “position” information in the CCE Toolset cannot be utilised in the VMCDM as AutomationML adopts a different method of describing “position”. The experiments presented in Section 4.4.5 shows that the direct deployment approach significantly compresses the time required to develop, validate and reconfigure PLC control software. Also, the generated control software provides better execution speed and occupies less memory. However, the control software generated by the direct deployment approach is potentially difficult to diagnose and debug under a few specific circumstances. Additional engineering tools are potentially needed to aid the diagnosis and debug in these circumstances. The development of such engineering tools is considered as future work and presented in Section 5.3.

5.2. Research Contributions

As also described in more details in section 1.3.4, this dissertation makes the following original contributions to the field of the virtual engineering of automation systems:

1. A tool-independent data model in AutomationML for describing the virtual models of component-based automation system, which potentially enables these models to be

reused efficiently thereby expanding the impact of VC to the whole lifecycles of automation systems.

2. A consolidated set of methodologies and engineering tool functionality that enables a component-based open virtual commissioning framework, including the following original technologies:
 - a. An approach and engineering tool functionality for mapping the virtual models of component-based automation systems to the HIL common data models described in an open standard.
 - b. An approach and engineering tool functionality for directly deploying well-structured PLC control software based on the simulated control logic of the component-based virtual models and the pre-developed reusable runtime components.
 - c. An approach and engineering tool functionality for directly deploying the HMI manual control models based on the state behaviour of the component-based virtual models.

5.3. Future Work

5.3.1. Potential enhancements

The proposed data models and deployment solutions provide the fundamental groundwork for achieving the set objectives. However, it is envisioned that further or more ambitious objectives could be reached by introducing additional functionalities.

5.3.1.1. VCMapper and CCE tool integration

The current Logic Mapper for directly deploying control software is independent of the CCE tool. They exchange data by exporting and importing XML files. This can lead to inefficiency of information synchronisation or even data inconsistency when changes are made to the control logic of virtual data models in the CCE tool. This issue can be resolved by integrating the logic mapper with the CCE tool to automate the data update of control logic.

5.3.1.2. HIL prior to the physical commissioning

In the current VC framework, control software generated by the direct deployment engineering tool is applied to perform real commissioning directly. However, potential errors

might be brought in during the manual I/O mapping step despite its simplicity. If HIL can be performed through connecting the automatically generated control software with the virtual models, potential errors in I/O mapping can be eliminated prior to the real commissioning.

5.3.1.3. Direct deployment of OPCUA configuration

The current common data models contain the information for connecting the control software with the virtual models. However, the connections between control systems and virtual prototypes, still need to be created, normally based on OPC, in a manual manner. The OPC client objects still need to be manually created and configured. This work can be largely automated by automatically generating the OPC Unified Architecture (OPC UA) [132] client objects based on the virtual common data models.

The OPC foundation and the PLCOpen foundation have combined OPC UA with the IEC61131-3 languages in order to achieve efficient communication. OPC UA object types can be created from the declaration of IEC61131-3 function blocks in the PLC and corresponding OPC UA objects from instances of the function blocks. This results in the advantage that a control program, regardless of the controller on which it is executed and the OPC UA server via which the data is accessed, is always implemented in the same structure of objects in the address area. For UA clients, this results in identical UA access at the semantic level.

Therefore, a new function, which can be added into the current direct deployment solution, is to automatically generate OPC UA client objects when the corresponding control software is automatically generated. This can significantly facilitate the further reuse of the virtual data models for HIL virtual commissioning or monitor-related applications during machine operational phase.

5.3.1.4. Direct deployment of control software in graphical languages

The current direct deployment solution is mainly focused on the automatic generation of the control software in textual languages. This is mainly due to the complexity of the source code of the graphical-based programs. However, control programs are preferred even required, to be represented in graphical languages in specific industrial applications. Therefore, in order to enhance the applicability in industry, the function of automatically generating source code of graphical-based programs can be potentially implemented. To

achieve this, a key success factor is to find an innovative approach to analysing the complex source code of graphical programs.

It is worth mentioning that another constraint is that many PLC programming tools do not support import and export of graphical-based programs. However, addressing this need is beyond the scope of this research.

5.3.2. Future research directions

As a result of the work presented in this dissertation, a set of new research directions have arisen. The most immediate directions that directly complement this research work are described in the following sub sections.

5.3.2.1. Web-based 3D remote monitoring system

The proposed data model mapping solution has transformed the virtual models into domain-specific common data models described in a XML-based open standard. These common data models can be regarded as the basis of 3D-based remote monitoring systems. In order to make the best use of the virtual data models to build desirable remote monitoring systems, new applications are needed and a few key issues still need to be researched. Web-based applications for running the 3D models according to the real-time states of the real machine are required. In order to be reused by web-based applications, virtual models should be described in formal ontology languages. Additionally, the performance of data transfer via TCP/IP Ethernet need to be investigated and researched.

5.3.2.2. Reverse engineering of direct deployment

The proposed direct deployment approach enables the control software development to be performed in a graphical virtual environment. The approach also introduces risks when changes to the generated control software are made on the shop floor, in which case there will be inconsistency between the control software and the corresponding virtual models. A potential solution would be to introduce reverse engineering functions into the current direct deployment solution by transforming the control software back to the original state behaviour. In this case, the control engineers in the shop floor are able to modify the control logic through modifying the virtual models in the logic mapper and afterwards the logic mapper can automatically update the control software as well as the virtual models, thereby data consistency between virtual models and the real control software can be achieved.

5.3.2.3. Direct deployment of other control software

The proposed direct deployment solution has been developed in the context of the specific domain of PLC control software deployment and HMI deployment with the automation sector. However, it is believed that the principle of simulation-based direct deployment approach could be applicable to other sectors of PLC usage and the other control domains such as robotics control software deployment. Further research into the wider applicability of the approach and possible required features is necessary.

Publications

- 1) Kong, X., Ahmad, B., Harrison, R., Jain, A., Park, Y. and Lee, L. J., “Realising the open virtual commissioning of modular automation systems”, Proceedings of DET2011, 7th International Conference on Digital Enterprise Technology, Athens, Greece, 28-30 September 2011, pp. 402 - 410
- 2) Kong, X., Ahmad, B., Harrison, R., Park, Y. and Lee, L.J., "Direct deployment of component-based automation systems", IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), 2012, pp.1,4, 17-21 Sept. 2012.
- 3) Ahmad, B., Watermann, J., Kong, X., Harrison, R. and Colombo, A.W., “Automatic Generation of Human Machine Interface Screens from Component-Based Reconfigurable Virtual Manufacturing Cell”, IECON 2013, 39th Annual Conference of the IEEE Industrial Electronics, Society, Vienna, Austria, 10-13 November, 2013

References

- [1] Harrison, R., A.A. West, R.H. Weston, and R.P. Monfared, *Distributed engineering of manufacturing machines*. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2001. **215**(2): p. 217-231.
- [2] Camarinha-Matos, L.M. *Virtual organizations in manufacturing trends and challenges*. in *International Conference on Flexible Automation and Intelligent Manufacturing*. 2002. Dresden, Germany.
- [3] ElMaraghy, H., *Flexible and reconfigurable manufacturing systems paradigms*. International Journal of Flexible Manufacturing Systems, 2005. **17**(4): p. 261-276.
- [4] 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.
- [5] Mehrabi, M.G., A.G. Ulsoy, and Y. Koren, *Reconfigurable manufacturing systems: Key to future manufacturing*. Journal of Intelligent Manufacturing, 2000. **11**(4): p. 403-419.
- [6] 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.
- [7] Reinhart, G. and G. Wünsch, *Economic application of virtual commissioning to mechatronic production systems*. Production Engineering, 2007. **1**(4): p. 371-379.
- [8] Harrison, R., *A component-based control system for agile manufacturing*. Engineering Manufacture, 2005.
- [9] Ahmad, B., J. Watermann, X. Kong, R. Harrison, and A. Colombo, *Automatic Generation of human machine interface screens from component based reconfigurable virtual manufacturing cell*, in *39th Annual Conference of the IEEE Industrial Electronics Society (IECON 2013)*2013: Vienna, Austria.
- [10] Watermann, J., *Self Configurable Human Machine Interface Screens for Component-Based Automation Systems*. University of Applied Sciences Emden, 2013. Thesis of **Master**
- [11] McLeod, S., *Development of a Toolkit for Component-Based Automation Systems*. Mechanical and Manufacturing Engineering, Loughborough University 2012. Thesis of **Doctor of Philosophy**
- [12] Ahmad, B., *A component-based virtual engineering approach to PLC code generation for automation systems*. Mechanical and Manufacturing Engineering Loughborough University, 2013. Thesis of **Doctor of Philosophy**
- [13] Xing, B.e.a., *Reconfigurable manufacturing system for agile mass customization manufacturing*, in *22nd International Conference on CAD/CAM, Robotics and Factories of the Future*2006: India.

References

- [14] ElMaraghy, H.A., M.A. Ismail, and H.A. ElMaraghy, *Component Oriented Design of Change-Ready MPC Systems*, in *Changeable and Reconfigurable Manufacturing Systems2009*, Springer London. p. 213-226.
- [15] 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.
- [16] Delamer, I.M., *Event-based middleware for reconfigurable manufacturing systems: a semantic web services approach*. Department of Automation, Tampere University of Technology, Finland, 2006. Thesis of **Doctor of Philosophy**
- [17] Bi, M. Z, Lang, T. S. Y, Shen, W, Wang, and L, *Reconfigurable manufacturing systems : the state of the art*. Vol. 46. 2008, Abingdon, ROYAUME-UNI: Taylor & Francis. 26.
- [18] 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.
- [19] Harrison, R. and A.W. Colombo. *Collaborative automation from rigid coupling towards dynamic reconfigurable production systems in 16th IFAC World Conference*. 2005.
- [20] Lüder, A., E. Estévez, L. Hundt, and M. Marcos, *Automatic transformation of logic models within engineering of embedded mechatronical units*. The International Journal of Advanced Manufacturing Technology, 2011. **54**(9-12): p. 1077-1089.
- [21] Orozco, O.J.L. and J.L.M. Lastra. *Agent-based control model for reconfigurable manufacturing systems*. in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. 2007.
- [22] Leitão, P., *Agent-based distributed manufacturing control: A state-of-the-art survey*. Engineering Applications of Artificial Intelligence, 2009. **22**(7): p. 979-991.
- [23] Colombo, A.W. *Industrial agents: towards collaborative production-automation,-management and-organization*. IEEE INDUSTRIAL ELECTRONICS SOCIETY NEWSLETTER, 2005. 17.
- [24] Leitao, P., J.M. Mendes, and A.W. Colombo. *Smooth migration from the Virtual design to the real manufacturing control*. in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009.
- [25] Mařík, V., M. Fletcher, and M. Pěchouček, *Holons & agents: Recent developments and mutual impacts*, in *Multi-Agent Systems and Applications II2002*, Springer. p. 233-267.
- [26] Vrba, P., P. Tichy, V. Mařík, K.H. Hall, R.J. Staron, F.P. Maturana, and P. Kadera, *Rockwell Automation's Holonic and Multiagent Control Systems Compendium*. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2011. **41**(1): p. 14-30.

References

- [27] Marik, V. and D. McFarlane, *Industrial adoption of agent-based technologies*. Intelligent Systems, IEEE, 2005. **20**(1): p. 27-35.
- [28] IEC. <http://www.iec.ch/>.
- [29] Sunder, C., A. Zoitll, J.H. Christensen, H. Steininger, and J. Ritsche. *Considering IEC 61131-3 and IEC 61499 in the context of Component Frameworks*. in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. 2008. IEEE.
- [30] Sunder, C., A. Zoitl, J.H. Christensen, V. Vyatkin, R. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J.L. Martinez-Lastra, and F. Auinger. *Usability and Interoperability of IEC 61499 based distributed automation systems*. in *Industrial Informatics, 2006 IEEE International Conference on*. 2006. IEEE.
- [31] Thramboulidis, K., *IEC 61499 vs. 61131: A Comparison Based on Misperceptions*. arXiv preprint arXiv:1303.4761, 2013.
- [32] 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.
- [33] Hajarnavis, V. and K. Young, *An investigation into programmable logic controller software design techniques in the automotive industry*. Assembly Automation, 2008. **28**(1): p. 43-54.
- [34] Lucas, M. and D. 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.
- [35] PLCOpen. www.plcopen.org/pages/tc1_standards/iec_61131_3/.
- [36] Lee, L.J., *A Next Generation Manufacturing Control System For A Lean Production Environment*. Mechanical and Manufacturing Engineering, Loughborough University, 2004. Thesis of **Doctor of Philosophy**
- [37] 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.
- [38] 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.
- [39] 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.
- [40] Frey, G. and L. Litz. *Formal methods in PLC programming*. in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*. 2000.

References

- [41] 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.
- [42] 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. I. Specification and design*. Control Systems Technology, IEEE Transactions on, 1999. **7**(6): p. 657-665.
- [43] Feldmann, K. and A.W. Colombo, *Monitoring of flexible production systems using high-level Petri net specifications*. Control Engineering Practice, 1999. **7**(12): p. 1449-1466.
- [44] Klein, S., G. Frey, and M. Minas, *PLC programming with signal interpreted Petri nets*, in *Applications and theory of Petri nets 2003*, Springer. p. 440-449.
- [45] Frey, G. and M. Minas. *Editing, visualizing, and implementing signal interpreted petri nets*. in *Proceedings of the AWPN 2000*. 2000.
- [46] Frey, G. *Automatic implementation of Petri net based control algorithms on PLC*. in *American Control Conference, 2000. Proceedings of the 2000*. 2000. IEEE.
- [47] 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.
- [48] 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.
- [49] 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.
- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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.

References

- [54] 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.
- [55] Danielsson, K., J. Richardsson, 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.
- [56] 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.
- [57] 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.
- [58] 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.
- [59] Estévez, E., M. Marcos, and D. Orive, *Automatic generation of PLC automation projects from component-based models*. *The International Journal of Advanced Manufacturing Technology*, 2007. **35**(5): p. 527-540.
- [60] 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.
- [61] B´evan, R., P. Berruet, and F. Lamotte, *Generation of multiplatform control for transitic systems using a component-based approach*, in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*2012: Kraków, Poland.
- [62] 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.
- [63] 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.
- [64] Hundt, L., A. Luder, A. Kohlein, and N. Gewalt. *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.
- [65] Andersson, M. and E. Helander, *Automatic generation of PLC programs using Automation Designer*. Department of Signals and Systems, CHALMERS UNIVERSITY OF TECHNOLOGY, Sweden, 2010. Thesis of

References

- [66] 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.
- [67] Hoffmann, P., R. Schumann, T. Maksoud, and G.C. Premier. *Virtual Commissioning of Manufacturing Systems—A Review and new Approaches for Simplification*. in *Proceedings 24th European Conference on Modelling and Simulation, Edited by A. Bargiela, SA Ali, D. Crowley, and E.J.H. Kerckhoffs*. 2010.
- [68] Makris, S., G. Michalos, and G. Chryssolouris, *Virtual Commissioning of an Assembly Cell with Cooperating Robots*. *Advances in Decision Sciences*, 2012. **2012**: p. 11.
- [69] Kuehn, W. and hn, *Digital factory: integration of simulation enhancing the product and production process towards operative control and optimisation*, in *Proceedings of the 38th conference on Winter simulation2006*, Winter Simulation Conference: Monterey, California.
- [70] Eversheim, W., D. Koerth, and J. Gentschke, *Gesellschaft Produktionstechnik: Inbetriebnahme komplexer Maschinen und Anlagen: Strategien und Praxisbeispiele zur Rationalisierung in der Einzel-und Kleinserienproduktion*. Düsseldorf: VDI-Verl, 1990.
- [71] Glas, J., *Standardisierter aufbau anwendungsspezifischer zellenrechnersoftware*1993: Springer Berlin.
- [72] VDM-Bericht, *Abteilungsübergreifende Projektierung komplexer Maschinen und Anlagen*", 1997, Verein Deutscher Werkzeugmaschinenhersteller: Aachen.
- [73] Kain, S., F. Schiller, and T. Frank. *Monitoring and diagnostics of hybrid automation systems based on synchronous simulation*. in *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. 2010. IEEE.
- [74] Kiefer, J., *Mechatronikorientierte Planung automatisierter Fertigungszellen im Bereich Karosserierohbau*. 2008.
- [75] Schludermann, H., T. Kirchmair, and M. Vorderwinkler. *Soft-commissioning: hardware-in-the-loop-based verification of controller software*. in *Proceedings of the 32nd conference on Winter simulation*. 2000. Society for Computer Simulation International.
- [76] Jiaxin, Q. *Application research on enterprise integrated automation based on OPC & OPC-XML*. in *Automation and Logistics, 2009. ICAL'09. IEEE International Conference on*. 2009. IEEE.
- [77] M. Bergert, J.K., *Mechatronic Data Models in Production Engineering in 10th IFAC Workshop on Intelligent Manufacturing Systems*2010: Lisbon, Portugal.
- [78] Qin S F, H.R., West, A A, Wright D K, *Study of 3D simulation modelling for supporting a plug-and-play distributed control system*. *International Journal of Agile Manufacturing*, 2005. **8**(1): p. 101-109.

References

- [79] Rainer Drath, A.L., Jörn Peschke, Lorentz Hundt, *Seamless automation engineering with AutomationML*, in *14th International Conference on Concurrent Enterprising IEC2008*: Lissabon.
- [80] Moser, T. and S. Biffel. *Semantic tool interoperability for engineering manufacturing systems*. in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*.
- [81] Bellamine, M., N. Abe, K. Tanaka, and H. Taki. *Remote machinery maintenance system with the use of virtual reality*. in *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*. 2002. IEEE.
- [82] Wenzel, S., U. Jessen, and J. Bernhard, *Classifications and conventions structure the handling of models within the Digital Factory*. *Computers in Industry*, 2005. **56**(4): p. 334-346.
- [83] Kim, G.Y., J.Y. Lee, H.S. Kang, and S.D. Noh, *Digital Factory Wizard: an integrated system for concurrent digital engineering in product lifecycle management*. *International Journal of Computer Integrated Manufacturing*, 2010. **23**(11): p. 1028-1045.
- [84] Li, S., Y. Wang, T. Yang, B. Chen, and H. Yang. *A Novel Digital Factory Technology in Complex Production Application*. in *Digital Manufacturing and Automation (ICDMA), 2010 International Conference on*. 2010. IEEE.
- [85] Ng, A.H., J. Adolfsson, M. Sundberg, and L.J. De Vin, *Virtual manufacturing for press line monitoring and diagnostics*. *International Journal of Machine Tools and Manufacture*, 2008. **48**(5): p. 565-575.
- [86] Alabdulkarim, A.A., P.D. Ball, and A. Tiwari. *Rapid modeling of field maintenance using discrete event simulation*. in *Simulation Conference (WSC), Proceedings of the 2011 Winter*. 2011. IEEE.
- [87] Kain, S., S. Dominka, M. Merz, and F. Schiller. *Reuse of HiL simulation models in the operation phase of production plants*. in *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on*. 2009. IEEE.
- [88] Kain, S., F. Schiller, and S. Dominka. *Reuse of models in the lifecycle of production plants using HiL simulation models for diagnosis*. in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*. 2008. IEEE.
- [89] Feldhorst, S., M. Fiedler, M. Heinemann, M. ten Hompel, and H. Krumm. *Event-based 3D-monitoring of material flow systems in real-time*. in *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. 2010. IEEE.
- [90] Yan, X.-T., C. Jiang, B. Eynard, J. Zhao, Z. Zhang, X. Tian, and X. Jia, *Toward the Manufacturing Software Interoperability*, in *Advanced Design and Manufacture to Gain a Competitive Edge2008*, Springer London. p. 387-396.
- [91] Barth, M., R. Drath, A. Fay, F. Zimmer, and K. Eckert. *Evaluation of the openness of automation tools for interoperability in engineering tool chains*. in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. 2012. IEEE.

References

- [92] Biffel, S., A. Schatten, and A. Zoitl. *Integration of heterogeneous engineering environments for the automation systems lifecycle*. in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009.
- [93] Drath, R. and M. Barth. *Concept for managing multiple semantics with AutomationML—Maturity level concept of semantic standardization*. in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. 2012. IEEE.
- [94] Runde, S. and A. Fay. *A data exchange format for the engineering of building automation systems*. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008.
- [95] Schleipen, M., R. Drath, and O. Sauer. *The system-independent data exchange format CAEX for supporting an automatic configuration of a production monitoring and control system*. in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*. 2008.
- [96] Drath, R., A. Luder, J. Peschke, and L. Hundt. *AutomationML - the glue for seamless automation engineering*. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008.
- [97] XML, W.C. <http://www.w3schools.com/xml/>.
- [98] Vernadat, F.B., *Technical, semantic and organizational issues of enterprise interoperability and networking*. *Annual Reviews in Control*, 2010. **34**(1): p. 139-144.
- [99] Luo Yan, L.Y.T., *Data exchange strategy for manufacturing simulation of shop floor information systems*. *International Journal of Radio Frequency Identification Technology and Applicat*, 2009. **2**: p. 216-227.
- [100] Gruber, T.R., *A translation approach to portable ontology specifications*. *Knowledge acquisition*, 1993. **5**(2): p. 199-220.
- [101] Luyan, B., J. Zongxia, and F. Shengtao. *Ontology-based information integration framework for mechatronics system multi-disciplinary design*. in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. 2008.
- [102] Klein, M., D. Fensel, F. Van Harmelen, and I. Horrocks, *The relation between ontologies and XML schemas*. *Electronic Trans. on Artificial Intelligence*, 2001.
- [103] Yang, K., R. Steele, and A. Lo. *An Ontology for XML Schema to Ontology Mapping Representation*. in *iiWAS2007*. 2007.
- [104] Antoniou, G., *A semantic web primer*2004: the MIT Press.
- [105] PLCOpen. www.plcopen.org/pages/tc6_xml/xml_intro/.
- [106] AutomationML-organization. *AutomationML overview*. 2008; Available from: www.automationml.org/.
- [107] *The Knronos Group*. Available from: <http://khronos.org>.

References

- [108] Estevez, E., M. Marcos, A. Lüder, and L. Hundt. *PLCopen for achieving interoperability between development phases*. in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. 2010. IEEE.
- [109] Lüder, A., L. Hundt, and A. Keibel. *Description of manufacturing processes using AutomationML*. in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. 2010. IEEE.
- [110] Kuhlenkötter, B., A. Schyja, A. Hypki, and V. Miegel. *Robot Workcell Simulation with AutomationML Support-An Element of the CAx-Tool Chain in Industrial Automation*. in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. 2010. VDE.
- [111] Botaschanjan, J., B. Hummel, T. Hensel, and A. Lindworsky. *Integrated behavior models for factory automation systems*. in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. 2009. IEEE.
- [112] Schleipen, M., D. Gutting, and F. Sauerwein. *Domain dependant matching of MES knowledge and domain independent mapping of AutomationML models*. in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. 2012. IEEE.
- [113] Persson, J., A. Gallois, A. Björkelund, L. Hafdell, M. Haage, J. Malec, K. Nilsson, and P. Nugues. *A knowledge integration framework for robotics*. in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. 2010. VDE.
- [114] Uschold, M. and M. Gruninger, *Ontologies and semantics for seamless connectivity*. ACM SIGMod Record, 2004. **33**(4): p. 58-64.
- [115] Runde, S., H. Dibowski, A. Fay, and K. Kabitzsch. *A semantic requirement ontology for the engineering of building automation systems by means of OWL*. in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. 2009. IEEE.
- [116] Schleipen, M., *Automated production monitoring and control system engineering by combining a standardized data format (CAEX) with standardized communication (OPC UA)*. Factory Automation, 2010: p. 978-953.
- [117] Clark, J., *Xsl transformations (xslt)*. World Wide Web Consortium (W3C). URL <http://www.w3.org/TR/xslt>, 1999.
- [118] Reif, G., M. Jazayeri, and H. Gall. *Towards Semantic Web Engineering: WEESA-Mapping XML Schema to Ontologies*. in *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*. 2004.
- [119] Maedche, A., B. Motik, N. Silva, and R. Volz, *MaFra—a mapping framework for distributed ontologies*, in *Knowledge engineering and knowledge management: ontologies and the semantic web2002*, Springer. p. 235-250.
- [120] Kalfoglou, Y. and M. Schorlemmer, *Ontology mapping: the state of the art*. The knowledge engineering review, 2003. **18**(1): p. 1-31.

References

- [121] Visser, P.R., D.M. Jones, T. Bench-Capon, and M. Shave. *An analysis of ontology mismatches; heterogeneity versus interoperability*. in *AAAI 1997 Spring Symposium on Ontological Engineering, Stanford CA., USA*. 1997.
- [122] Bohring, H. and S. Auer, *Mapping XML to OWL Ontologies*. Leipzig Informatik-Tage, 2005. **72**: p. 147-156.
- [123] Bourret, R., C. Bornhovd, and A. Buchmann. *A generic load/extract utility for data transfer between XML documents and relational databases*. in *Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop on*. 2000. IEEE.
- [124] Pressman, R.S. and D. Ince, *Software engineering: a practitioner's approach*. Vol. 5. 1992: McGraw-hill New York.
- [125] Microsoft. *Interface definition in Object-Oriented Programming* [http://msdn.microsoft.com/en-us/library/aa260635\(v=vs.60\).aspx](http://msdn.microsoft.com/en-us/library/aa260635(v=vs.60).aspx).
- [126] Blender, Available from: <http://www.blender.org/>.
- [127] Vera, D., *Innovative Approach to the Design and Realisation of a Virtual Prototyping Environment for Manufacturing System Engineering*. Mechanical and Manufacturing Engineering, Loughborough University, 2004. Thesis of **Doctor of Philosophy**
- [128] Swales, A., *Open Modbus/TCP Specification*. Schneider Electric, 1999. **29**.
- [129] Younis, M.B. and G. Frey. *Software quality measures to determine the diagnosability of PLC applications*. in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. 2007. IEEE.
- [130] 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.
- [131] 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.
- [132] *OPC UA - <https://www.opcfoundation.org/UA>*.