


This item is held in Loughborough University's Institutional Repository (<https://dspace.lboro.ac.uk/>) and was harvested from the British Library's EThOS service (<http://www.ethos.bl.uk/>). It is made available under the following Creative Commons Licence conditions.




creative
commons
C O M M O N S D E E D


Attribution-NonCommercial-NoDerivs 2.5

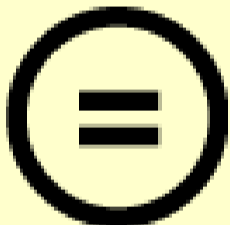
You are free:

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

Under the following conditions:

 **BY:** **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/>

WEB SERVICE CONTROL OF COMPONENT- BASED AGILE MANUFACTURING SYSTEMS

A Doctoral Thesis Submitted in Partial Fulfilment of the Requirements
for the Award of Doctor of Philosophy of
Loughborough University

By
Punnuluk Phaithoonbuathong

10 June, 2009

Wolfson School of Mechanical and Manufacturing Engineering
Loughborough University
United Kingdom

© Punnuluk Phaithoonbuathong 2009

To my wife and best friend – Aornsuda Wongwattanasat.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Dr. Robert Harrison and Dr. Andrew West for their great support. Their guidance, encouragement, cooperation and interest have helped me tremendously throughout the time of my study.

My sincere appreciation and thank also go to all my colleagues in the MSI Research Team of Mechanical and Manufacturing Engineering at Loughborough University who contributed to this study. I am gratefully inspired by their skills experiences and professionalism.

I would also like to especially acknowledge the support of the Ford Motor Company Limited, the EU FP7 SOCRADES and the EPSRC, IMCRC, GAIN and BDA projects and their collaborators in enabling various aspects of this research.

Most significantly, I wish to thank my family and friends for their sacrifice, love and support in which they never fail to grant me.

Finally, special thanks to my wife who is my eternal supporter. Nobody else could be more proud in my achievement than her.

ABSTRACT

Current global business competition has resulted in significant challenges for manufacturing and production sectors focused on shorter product lifecycles, more diverse and customized products as well as cost pressures from competitors and customers. To remain competitive, manufacturers, particularly in automotive industry, require the next generation of manufacturing paradigms supporting flexible and reconfigurable production systems that allow quick system changeovers for various types of products. In addition, closer integration of shop floor and business systems is required as indicated by the research efforts in investigating “Agile and Collaborative Manufacturing Systems” in supporting the production unit throughout the manufacturing lifecycles.

The integration of a business enterprise with its shop-floor and lifecycle supply partners is currently only achieved through complex proprietary solutions due to differences in technology, particularly between automation and business systems. The situation is further complicated by the diverse types of automation control devices employed. Recently, the emerging technology of Service Oriented Architecture’s (SOA’s) and Web Services (WS) has been demonstrated and proved successful in linking business applications. The adoption of this Web Services approach at the automation level, that would enable a seamless integration of business enterprise and a shop-floor system, is an active research topic within the automotive domain. If successful, reconfigurable automation systems formed by a network of collaborative autonomous and open control platform in distributed, loosely coupled manufacturing environment can be realized through a unifying platform of WS interfaces for devices communication.

The adoption of SOA-Web Services on embedded automation devices can be achieved employing Device Profile for Web Services (DPWS) protocols which encapsulate device control functionality as provided services (e.g. device I/O operation, device state notification, device discovery) and business application interfaces into physical control components of machining automation. This novel approach supports the possibility of integrating pervasive enterprise applications through unifying Web Services interfaces and neutral Simple Object Access Protocol (SOAP) message communication between control systems and business applications over standard Ethernet-Local Area Networks (LAN’s). In addition, the re-configurability of the automation system is enhanced via the utilisation of Web Services throughout an automated control, build, installation, test, maintenance and reuse system lifecycle via device self-discovery provided by the DPWS protocol.

The research presented in this thesis has investigated the research issues around the design, implementation, evaluation and reconfiguration of Web Services-based automation systems based upon a university automation test rig to show the feasibility and performance of WS for industrial machine usage. The precise evaluation and analysis of this proposed WS approach has been carried out as required by automotive supply chain and end-user industrialists and agile automation research paradigms. For example the assessment has been focused on quantification, qualification and comparison of parameters such as: (i) I/O interval processing time, (ii) Ethernet communication and reliability, (iii) business integration and (iv) process reconfiguration in contrast to centralized Programmable Logic Controller (PLC)-based systems and distributed LonWorks-based systems previously developed at MSI Research Institute, Loughborough University.

Keywords: Component-Based Design; Device Profile for Web Services (DPWS); Distributed Control System; Agile Manufacturing; Simple Object Access Protocol (SOAP); Mass Customization; Ethernet

TABLE OF CONTENTS

ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF FIGURES & TABLES	xii
ABBREVIATION	xvi
DEFINITIONS	xviii

CHAPTER 1 INTRODUCTION

1.1 PROBLEM DEFINITIONS.....	1
1.2 RESEARCH MOTIVATION.....	2
1.3 RESEARCH QUESTIONS.....	4
1.4 RESEARCH FOCUS.....	5
1.5 ORGANISATION OF THE THESIS.....	6

CHAPTER 2 MANUFACTURING SYSTEM REQUIREMENTS AND TRENDS

2.1 PROBLEM STATEMENT.....	8
2.2 DRIVERS OF CHANGE IN AUTOMOTIVE MANUFACTURING.....	8
2.3 CURRENT MANUFACTURING SYSTEMS.....	12
2.3.1 Manufacturing system architecture.....	13
2.3.2 Lifecycle of manufacturing system and machine design.....	14
2.3.3 Problem identification.....	16
2.3.4 Manufacturing and automation system requirements.....	19
2.4 THE NEXT GENERATION MANUFACTURING SYSTEM.....	20

CHAPTER 3 AGILE MANUFACTURING PARADIGM

3.1 PROBLEM STATEMENT.....	24
3.2 OVERVIEW OF AGILE MANUFACTURING.....	25
3.3 KEY ENABLERS OF AGILE MANUFACTURING.....	28
3.4 RELATED APPROACHES TOWARD AGILE MANUFACTURING SYSTEMS.....	34

3.4.1 OSACA	36
3.4.2 OPC Foundation.....	37
3.4.3 RIMACS.....	39
3.4.4 Key Characteristic Summary.....	40
3.5 EXISTING APPROACHES LIMITATION ANALYSIS	41
3.6 PROPOSED SOLUTION ON WEB SERVICES BASED AUTOMATION.....	43
3.7 ASSESSMENT IN MEETING AGILITY REQUIREMENTS	45
3.8 CONCLUSION.....	46

CHAPTER 4 OVERVIEW OF CURRENT TECHNOLOGY AND KEY ENABLERS FOR AGILE MANUFACTURING SYSTEM

4.1 PROBLEM STATEMENT.....	49
4.2 CONTROL SYSTEM ARCHITECTURES AND TRENDS.....	49
4.3 PROGRAMMABLE LOGIC CONTROLLERS.....	53
4.4 COMMUNICATION NETWORKS.....	56
4.4.1 Open industrial Fieldbus systems.....	56
4.4.2 Industrial Ethernet Networks.....	58
4.4.3 Ethernet Standard.....	63
4.5 COMMUNICATION ARCHITECTURES.....	65
4.5.1 Point-to-Point.....	65
4.5.2 Client-Server.....	65
4.5.3 Publish-Subscribe.....	66
4.6 A DISTRIBUTED AUTOMATION SYSTEM.....	69
4.6.1 LonWorks system with Fieldbus.....	70
4.6.2 Embedded Modules with Ethernet.....	72
4.6.3 Distributed Control Applications.....	73
4.7 MIDDLEWARE SERVER.....	77
4.7.1 CORBA.....	78
4.7.2 SOA Middleware.....	80
4.7.3 DEBATE: CORBA VS. SOA Middleware.....	83
4.8 AGENT-BASED MANUFACTURING SYSTEM.....	87
4.8.1 Object-Oriented Architecture.....	87

4.9 SERVICE-ORIENTED ARCHITECTURE (SOA) AND WEB SERVICES FOR MANUFACTURING SYSTEM.....	90
4.9.1 SOA Basic definitions.....	90
4.9.2 Web Services-the SOA connection.....	91
4.9.3 Web services consideration issues.....	95
4.10 CONCLUSION.....	96
 CHAPTER 5 RESEARCH FOCUS AND DESIGN	
5.1 PROBLEM STATEMENT.....	98
5.2 PROBLEM DEFINITION AND END-USER REQUIREMENTS.....	98
5.3 RELATED AUTOMATION RESEARCHES.....	100
5.4 RESEARCH OBJECTIVES.....	103
5.4.1 The area of development and novel contributions.....	105
5.4.2 Research Design.....	107
5.5 CONCLUSION.....	107
 CHAPTER 6 A WEB SERVICES COMPONENT-BASED AUTOMATION DESIGN	
6.1 PROBLEM STATEMENT.....	109
 PART I- DISTRIBUTED CB AUTOMATION SYSTEM	
6.2 COMPONENT-BASED (CB) SYSTEM DEVELOPMENT.....	111
6.2.1 Component-based construction principles.....	111
6.2.2 Component-based software development process and lifecycle.....	116
6.2.3 Component-based development design issues.....	119
6.3 A COMPONENT-BASED DESIGN FOR MANUFACTURING SYSTEMS.....	120
6.3.1 Encapsulated Industrial Component Based Systems.....	120
6.3.2 Specification of a component-based automation system.....	123
6.3.3 The CB control design model.....	126
6.4 DESIGN SPECIFICATION FOR POWERTRAIN ASSEMBLY MACHINES.....	128
6.5 THE CB SYSTEM MODEL.....	131
6.6 PART I CONCLUSION.....	133
 PART II- WEB SERVICES-BASEDAUTOMATION SYSTEM	
6.7 AN INTRODUCTION TO WEB SERVICES ENTERPRISE INTEGRATION.....	135

6.8 AUTOMATION SYSTEMS INTEGRATION USING WEB SERVICES.....	138
6.8.1 The Need for Web Services within the Automation Domain.....	138
6.8.2 Transformation of Web Services for CB automation systems.....	138
6.8.3 Building of control applications.....	140
6.8.4 Implementation and enabling technology.....	144
6.9 WEB SERVICES DESIGN APPROACH FOR DISTRIBUTED EMBEDDED CONTROL DEVICES.....	147
6.9.1 Localisation and Standard Discovery Lookup.....	148
6.9.2 Lightweight code development for embedded devices.....	150
6.9.2-1 The gSOAP Stub and Skeleton implementation.....	152
6.9.2-2 Building control functions in a Web Services environment.....	153
6.10 PART II CONCLUSION.....	156

CHAPTER 7 IMPLEMENTATION OF CB-WS

7.1 PROBLEM STATEMENT.....	157
7.2 INTRODUCTION.....	157
7.3 CASE STUDY 1: DISTRIBUTED AUTOMATION SYSTEM (FORD-FESTO RIG).....	159
7.3.1 FORD-FESTO test rig specifications.....	159
7.3.2 System development and integration.....	161
7.3.2-1 Work flow and Sequencing of the test rig system.....	163
7.3.2-2 State transition diagrams.....	164
7.3.2-3 Component software development.....	165
7.4 CASE STUDY 2- WEB SERVICES DEVICE CONTROL	167
7.4.1 Development platform.....	167
7.4.2 Web Services applications.....	170
7.5 CONCLUSION	173

CHAPTER 8 WEB SERVICES AUTOMATION RIG DESIGN AND IMPLEMENTATION

8.1 PROBLEM STATEMENT.....	175
8.2 TEST RIG DESIGNED SPECIFICATION.....	176
8.2.1 Overview.....	176
8.2.2 Control hardware specification.....	177
8.2.3 OS and Software architecture.....	178
8.3 DPWS COMPONENT DESIGN WITH WSDL DESCRIPTION.....	180

8.4 DESIGN TOOLS	183
8.4.1 SERVER- Multiple services on the FTB	183
8.5 INDUSTRIAL DEMONSTRATION	186
8.5.1 Web services- based control system integration	187
8.5.2 Services Orchestration Engine	190
8.5.3 Business Application Integration	194
8.5.3-1 SAP xMII and WS test rig Integration Platform	195
8.5.3-2 The SAP xMII process monitoring and error diagnostic application	197
8.6 CONCLUSION	199

CHAPTER 9 EVALUATION AND DISCUSSION

9.1 PROBLEM STATEMENT	200
9.2 DPWS PERFORMANCE ANALYSIS	201
9.2.1 DPWS- SOAP message structure	201
9.2.2 Ethernet TCP/IP network communication	203
9.2.2-1 TCP/IP communication approach	206
9.2.2-2 Ethernet packet delivery time and deterministic	208
9.2.3 DPWS processing time and Component I/O interval reaction time	209
9.3 EASE OF MACHINE RECONFIGURABILITY ASSESSMENT	211
9.3.1 Modifying a process work flow	216
9.3.2 Adding a sensor element	217
9.3.3 Adding a new component	219
9.3.4 Comparison of Distributed WS-CB and Conventional Centralized PLC Automation	222
9.4 COMPONENT MODULARLITY AND REUSABILITY ASSESSMENT	225
9.5 THE IMPLEMENTATION WITH PROCESS ENGINEERING TOOLS	228
9.6 SEAMLESS INTEGRATION ASSESSMENT	232
9.7 SUITABILITY OF THE WS BASED AUTOMATION IN INDUSTRY	235
9.8 FULFILLING THE END USER REQUIREMENTS WITHIN AGILE AUTOMATION	238
9.9 CONCLUSION	241

CHAPTER 10 CONCLUSION AND FUTURE WORKS

10.1 RESEARCH CONCLUSION	243
10.1.1 Research findings	244

10.1.2 Contribution to Knowledge.....	244
10.1.3 Implementation and Evaluation.....	245
10.2 RESEARCH ACHEIVEMENTS.....	246
10.3 FUTURE WORKS.....	249
REFERENCE.....	253
APPENDIX A THE TEST RIG STATE TRANSITION DIAGRAM.....	260
APPENDIX B THE TEST RIG FUNCTION BLOCK CONTROL PROGRAMME.....	267
APPENDIX C WEB SERVICES- ARM9 DEVELOPMENT PLATFORM.....	277
APPENDIX D WEB SERVICES APPLICATION PROGRAMMING.....	281
APPENDIX E PACKET ANALYSIS OF DPWS OPERATIONS.....	287
APPENDIX F PUBLICATION PAPER.....	288
APPENDIX G WEBSITE REFERENCE.....	289

LIST OF FIGURES

Figure 1-1	Research focus overview.....	5
Figure 2-1	Manufacturing System Structure and Production Process.....	12
Figure 2-2	Product Lifecycle and Machine Design& Build Process.....	14
Figure 2-3	Product Lifecycle and Reconfiguration Process.....	15
Figure 2-4	Current Machine Build Process.....	16
Figure 2-5	Collaborative and Agile Manufacturing Framework.....	22
Figure 3-1	Agility and Interaction in Manufacturing System Design.....	27
Figure 3-2	Conceptual Model of Key Enablers for Agile Manufacturing System.....	29
Figure 3-3	Collaborative Enterprise Model.....	35
Figure 3-4	OSACA Framework.....	37
Figure 3-5	OPC Framework.....	38
Figure 3-6	RIMACS Framework.....	40
Figure 3-7	Common Middleware Architecture.....	41
Figure 3-8	Web Services- based Automation Paradigm.....	44
Figure 4-1	Basic Control Architectures.....	50
Figure 4-2	A basic PLC illustrations.....	53
Figure 4-3	Language constructs within the IEC 61131-3 Programming Standard.....	54
Figure 4-4	Layer of the OSI model.....	59
Figure 4-5	OSI Messaging.....	60
Figure 4-6	The Structure of an IP Datagram.....	63
Figure 4-7	Request/Response and Publish/Subscribe Data-Flows for a Distributed Control System.....	69
Figure 4-8	Structure for a Distributed Control System.....	70
Figure 4-9	Event-driven IEC 61499 Execution Controls.....	74
Figure 4-10	Functionalities of a Component.....	75
Figure 4-11	Three Tier e-Business Architecture.....	78
Figure 4-12	The CORBA Client/Server Invocation Methods.....	79
Figure 4-13	SOA Client –Server Middleware Model.....	80
Figure 4-14	Providing SOAP with XML, WSDL and UDDI in Web Service.....	81
Figure 4-15	Holonic Framework via Mobile Agent.....	88
Figure 4-16	Holon Structure.....	89

Figure 4-17	Business Ecosystems.....	90
Figure 4-18	SOA Web Services Structure in Manufacturing Systems.....	92
Figure 4-19	Devices Profile for Web Services (DPWS) Protocol Stack.....	93
Figure 5-1	A Conceptual Model of the Test Rig used in this Thesis with Web Services.....	103
Figure 5-2	SOA-Web Services Integration Framework.....	105
Figure 5-3	Peer-to-Peer Web Services Enabled Control System.....	106
Figure 6-1	The Component-based Design Principle.....	112
Figure 6-2	Automation Software Component Construction.....	114
Figure 6-3	The Component-based Design Framework for Automation Systems.....	115
Figure 6-4	A Generic Component-based Lifecycle.....	116
Figure 6-5	The Parallel Process of Component-based Development.....	117
Figure 6-6	Various Developers and End User Roles in a Component-based Framework.....	118
Figure 6-7	Stakeholder Roles Constituted in the COMPAG Project Framework.....	122
Figure 6-8	Component-based Machine Control Hierarchy.....	123
Figure 6-9	Structure of the Distributed Component- based Automation System.....	124
Figure 6-10	The State Transition Diagram in the Activity Function Block.....	126
Figure 6-11	A Component-based Design Model.....	127
Figure 6-12	UML Class Diagram of the Component-based System Design.....	132
Figure 6-13	A Basic Principle of SOA Architecture.....	137
Figure 6-14	Service Orchestration Ontology.....	141
Figure 6-15	Component-based Web Services Implementation Model.....	144
Figure 6-16	The Implementation of the WS-CB Automation System Design.....	146
Figure 6-17	P2P Search Procedure.....	148
Figure 6-18	Central Directory Lookup.....	149
Figure 6-19	Remote Procedure Call in Device Binding.....	151
Figure 6-20	Design Time and Run Time of a Server and a Client.....	152
Figure 6-21	Web Services- a Client and Server Project Implementation.....	154
Figure 6-22	Web Services Compliant Control Device Architecture.....	155
Figure 7-1	The FORD-FESTO test rig platform.....	159
Figure 7-2	Ethernet Interface I/O Module.....	160
Figure 7-3	HMI Operator Screen.....	160

Figure 7-4	Common four-state Transition Behaviour of the swivel Transfer Arm Component.....	164
Figure 7-5	Generic FBD Structure.....	166
Figure 7-6	Visual Studio .NET Web Services Application Platform.....	168
Figure 7-7	Service Namespace.....	168
Figure 7-8	DPWS Client Initialisation.....	169
Figure 7-9	DPWS Client for Services Invocation.....	169
Figure 7-10	DPWS Components and Services Initialisation.....	170
Figure 7-11	Web Services Test Scenario.....	171
Figure 7-12	Client and Server Devices Interaction.....	172
Figure 8-1	Web Services- based Automation System and Integration Platform.....	176
Figure 8-2	Field Terminal Block (FTB) Hardware Module.....	178
Figure 8-3	OS and Software Architecture on the FTB.....	179
Figure 8-4	ARM Real View Development Suite (ARM- RVDS).....	184
Figure 8-5	Control codes of the DPWS enabled component.....	188
Figure 8-6	Distributed Web Services-based Control System	192
Figure 8-7	Business Integration Scenario with the SAP xMII application.....	196
Figure 8-8	SAP- Production Line Monitoring Utility using SAP xMII.....	197
Figure 8-9	Handling Arm Fault Diagnostic Test Scenario.....	198
Figure 9-1	Ethernet TCP/UDP Packet Structure.....	201
Figure 9-2	The DPWS Service Invocation and Notification Time Analysis.....	210
Figure 9-3	Process Reconfiguration Workflow.....	212
Figure 9-4	Workflow Alteration: Station 1/2/3/4 → 1/2/4.....	213
Figure 9-5	Adding a Sensor in the Hopper Component.....	214
Figure 9-6	Adding a New Drill Component.....	215
Figure 9-7	Distributed WS- based Control System.....	222
Figure 9-8	Conventional Centralized PLC- based Control System.....	222
Figure 9-9	Automation System Development and Reconfiguration Framework.....	229
Figure 9-10	MSI Developed Suite Engineering Tools.....	231
Figure 9-11	XLON Communication Protocol Interface.....	233
Figure 9-12	Monitor Pro PLC Communication Protocol Interface.....	233
Figure 9-13	The SOA Middleware Integration Bus.....	234
Figure 10-1	The Author Original Work and Novel Contributions.....	243

LIST OF TABLES

Table 3-1	Multi-Facet of Agility in Manufacturing.....	28
Table 3-2	OPC/OSACA/RIMACS Key Characteristic Summaries.....	40
Table 3-3	Assessing the Achievement in Key Agility Features of Existing Approaches.....	45
Table 3-4	Addressed Problems, Requirements and Key Agile Enablers Summary.....	47
Table 3-5	Limitations, Requirements and Proposed SOA-WS Solution Summary.....	48
Table 4-1	Centralised Control Architecture Description.....	51
Table 4-2	Hierarchical Control Architecture Description.....	51
Table 4-3	Modified Hierarchical Control Architecture Description.....	52
Table 4-4	Heterogeneous Control Architecture Description.....	52
Table 4-5	Summary of Common Fieldbus Type Networks.....	57
Table 4-6	Industrial Ethernet Network Features.....	60
Table 4-7	Summary of Communication Technologies.....	67
Table 4-8	Summary of Distributed Enabling Technologies Appraisal.....	97
Table 6-1	FORD Requirements of the Component- based Automation System	129
Table 6-2	The Adoption of Web Services within Automation Systems	139
Table 7-1	Test Rig HMI.....	161
Table 7-2	Decomposition of the Test Rig Assembly.....	161
Table 9-1	DPWS Sever Operation Invocation- Ethernet TCP/IP Network Communication.....	204
Table 9-2	DPWS Server State Notification- Ethernet TCP/IP Network Communication.....	205
Table 9-3	Network Performance Analysis and Comparison.....	207
Table 9-4	Conventional Centralized PLC Control System.....	223
Table 9-5	Decentralized WS-CB Control System.....	224
Table 9-6	Adding the New DPWS Drill Component Scenario.....	227
Table 9-7	FTB and PLC Cost Comparison.....	237
Table 10-1	The Research Implementation Summary.....	249

ABBREVIATIONS

AM	Agile Manufacturing
ANSI	American National Standards Institute
ASS	Application Services System
CBA	Component-Based Approach
CB-MAS	Component- Based Manufacturing System
CGI	Common Gateway Interface
COM	Common Object Model
COMPAG	Component Based Paradigm for Agile Automation
CORBA	Common Object Request Broker Architecture
CSMA/CD	Carrier Sense Multiple Action/Collision Detection
DCOM	Distributed Component Object Model
DCS	Distributed Control System
DPWS	Devices Profile for Web Services
EIA	Electronic Industries Alliance
ERP	Enterprise Resource Planning
FCS	Frame Check Sequence
GUI	Graphic User Interface
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IOP	Internet Inter-ORB Protocol
IP	Internet Protocol
MAC	Media Access Control
MES	Manufacturing Execution System
MIME	Multipurpose Internet Mail Extensions

MOM	Message-Oriented Middleware
MTS	Microsoft Transaction Server
NGMS	The Next Generation Manufacturing System
OLE	Object Linking and Embedding
OODBMS	Object-Oriented Database Management Systems
OOP	Object-Oriented Programming
ORB	Object Request Broker
OSI	Open System Interconnection
PDE	Process Definition Environment
PLC	Programmable Logic Controller
RDBMS	Relational Database Management System
RTOS	Real-Time Operating System
SMTP	Simple Mail Transfer Protocol
SNTP	Simple Network Time Protocol
SOA	Service-Orient Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modelling Language
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URN	Uniform Resource Name
UUID	Universally Unique Identifier
VRML	Virtual Reality Modelling Language
WS	Web Services
WSDL	Web Services Description Language
XML	Extensible Markup Language

DEFINITIONS

Agent	A representative agency that has the power or authority to act on behalf of another. It is capable of perception, acting on its environment, communicating and, eventually, of mobility in accomplishing specific tasks such as assisting co-operators with required process information for business planning systems.
Automation system	This term used in this research is referred to the definition described by wordreference.com that “the control of equipment with advance technology involving electronic hardware; automation replace human workers by machines”. In this research this term is used interchangeably with “control system”
Client	A computer or device as an event sink on the network that requests or consumes the services or functions provided by the server (see the server term below). In this research, the client has a function of invoking (calling) the interested services to perform the manufacturing functions in the control system.
Component	An artifact that is one of the individual parts of which a composite entity is made up; especially a part that can be separated from or attached to a system. This thesis refers the component term to a physical automation device which is constituted by mechanical, control software and electrical element in the automation system.
Enterprise	An entire business organization. In this thesis, it is implied subsidiaries of business planning and management, manufacturing execution control and production unit.
Flexible automation system	D. Vera [116] has characterised flexible automation systems as the system that has fixed machine control hardware but programmable software to handle changes in work orders, production schedules and tooling for several types of parts. In general, this automation system is consisted of general purpose computer numerically controlled machines (CNC) with sophisticated control configuration changing system to process multiple types of products.
High level application	In this thesis, this term is used to express the software application designed for a specific task and used in the manufacturing execution and business level to control and monitor the shop-floor production unit.

Lean manufacturing	Lean Manufacturing is an operational strategy oriented toward achieving the shortest possible cycle time, a high level of throughput with a minimum of inventory by reducing waste along production lines. The goal is to decrease the time between a customer order and shipment, and it is designed to radically improve profitability, customer satisfaction, and throughput time.
Manufacturing system	This term is referred from <i>bridgefieldgroup.com</i> that the system which has function used to identify and plan demand and materials, analyze resource availability and requirements and schedule, release and report production. In this research, several departments in associate with planning, controlling and producing company products constitute manufacturing system. Departments may be included production sections, manufacturing supervisory and control, ERP, and etc.
Mass customisation	Its concept has combined the low unit costs of mass production goods with differing individual product specifications. The product variety is achieved through the use of components that may be assembled in a number of different configurations in order to satisfy the heterogeneity of all customers' preferences. In the context of manufacturing for individual customization, the processing systems need to employ a good degree of flexibility and re-configurability in process alterations for various types of products.
Mass production	The manufacture of goods in large quantities in the short period of time with minimum cost, often using standardized designs and assembly-line techniques. Its purpose is to produce more per worker-hour, and to lower the labor cost of the end product. This in turn allows the product to be sold for a lower cost. Mass production technique is the idea of deploying skill workforces responsible only for a certain task, e.g., one is in charge of cutting, another is finishing, a third one is assembling, and so on, to achieve a high level of throughput.
Object-oriented	The term that used for programming language in an agent-based system where data carries with itself the "methods" or "functions" used to handle that data. Its programming takes the functional view of abstract manipulation rather than the logic required to manipulate them, so object-oriented code is more flexible and more organized and easier to write, read, and change than traditional procedural code.

Open system	In this research, the term of an open system is implied to the automation aspect such that a system in which the components and their composition are specified in a non-proprietary environment, enabling manufacturing system to use these standard components to build competitive systems. There are three perspectives on open systems: portability - the degree to which a system component can be used in various environments, interoperability - the ability of individual components to exchange information, and integration - the consistency of the various human-machine interfaces between an individual and all hardware and software in the system.
Ramp up time	The term which is associated with the time taken for the process to reach the normal production capability after process modifications or new installations.
Reconfigurable automation system	The automation system that is designed for rapid change in both hardware (mechanical structure) and control software. The system configuration can be rearranged (adding, removing and ordering) quickly by the modular design of control modules (hardware/software) that can be reused and reconfigured rather replaced for a new module.
Server	A computer or device as an event source on the network that provides the services or functions. In this research, the server means the producer unit that offers the function in relation to commanding the control system such as driving the machine outputs or reading the sensors as requested or controlled by the client (consumer). It is noted that the client and the server function could be allocated on the single device to co-operate in the peer-to-peer communication model in the distributed control environment.

CHAPTER 1

Introduction

The objective of this chapter is to provide an overview of the research and the scope of the work in the area of future manufacturing paradigms enabled by the shift of mass production systems toward mass customised systems. The preliminary research questions based upon a new approach to agile automation systems are highlighted. The scope of the research in contributing to process re-configurability, reusability and integration capability based upon the key enabling technologies of Web Services (WS) and Component-based (CB) design approaches within the automation domain is presented.

1.1 Problem Definitions

In today's competitive global market where customer demands can fluctuate dramatically and global competition has become intense, companies must respond quickly and cost-effectively to market changes in order to maintain their competitive advantage. As reported by A. Molina [1] the characteristics of the global markets for many types of product have progressively changed towards meeting customer demand for low-volume, high-quality, customised products. These factors have had an impact on product lifecycles, development times, and production lead times all of which need to be shorter. As a result of: (i) increasing number of product variants, (ii) increasing mix of product types, (iii) decreasing batch sizes and (iv) decreasing product lifecycles, a conventional mass production approach is no longer a viable manufacturing strategy because it cannot effectively produce low-price products in an environment where the demand fluctuates unpredictably [84].

Many enterprises are changing their manufacturing paradigms and systems toward more flexible and adaptive approaches to improve efficiency and to support this trend towards mass customization ⁽¹⁾. The evolution has been from mass production ⁽²⁾, beyond lean manufacturing ⁽³⁾, into agile manufacturing ⁽⁴⁾ [7].

Within the conceptual agile framework presented in [32], the agile manufacturing system has been focused on the consideration of a wider perspective in which not only the automation system, but also the business enterprise and the business architecture, need to be included in the development process related to the shop-floor systems. Manufacturers have to distribute intelligence and decision making authority as close to the points of action, delivery, sales and even after-sales service as possible. To improve their ability to respond, manufacturers need to share the design and production information with their business partners.

However, within this integrated manufacturing-business perspective, traditional manufacturing automation systems need to be evolved towards agile capabilities. These systems are often constructed in a rigid, centralised, hierarchical and, in many cases, proprietary manner [105]. It has been the case that the development of the automation system typically follows a sequential design-implement-test lifecycle. In addition, the reconfiguration of automation systems is currently a complex, time-consuming and error-prone process. As a result, these factors have a direct impact, increasing the process ramp-up time and leading to performance degradation when production needs change. Specifically, current automation systems usually require experts to (re)commission and maintain them due to the often complex and unstructured code and the use of vendor specific technology [33]. It is therefore unproductive for industrialists to implement highly customizable production capability within the current design of manufacturing automation systems.

1.2 Research Motivation

The work of this research is based within the automotive industries with the Ford Motor Company as the project collaborator in the UK. The automotive industry is a competitive, risky and high investment domain [103]. Many automotive suppliers have turned to mergers, acquisitions, and/or joint ventures in an effort to grow top-line revenue, maintain market share, more effectively utilize assets and cost reduction in order to remain competitive [32].

Over the past years, there have been a number of research studies on flexible and reconfigurable automation systems to support reducing product time to market as well

as quickly increasing production capacity. At Loughborough University, a project initiated by the Manufacturing Systems Integration (MSI) Research Institute has investigated a Component-Based Approach (CBA) for automation systems for improved flexibility and re-configurability within the engine production lines at Ford.

The work has established a new engineering visualisation environment implemented using the Virtual Reality Modelling Language (VRML), reusable control software embedded within automation components and a Process Definition Environment (PDE) in which automation components can be configured, simulated and deployed. The project has contributed to the development of a truly concurrent engineering design approach of manufacturing systems aiming to reduce developments cost in automation systems and time to market of the new products.

The extension of previous developments adopting the CBA with standard Web technologies would substantially benefit the end-users by supporting new “open” standard way of automation design. Within the requirements of agile manufacturing, it has been noted that automation systems are required to be more integrated with business, suppliers and external machine builders systems to generate reliable and cost effective solutions. Recently, an emerging Service-Orient Architecture (SOA) and Web Services (WS) technology solutions have been widely researched to support business-to-business integration as reported in [122]. These technologies have proved a success in the e-business domain and they are becoming a standard approach to integration supported by major software developers such as Microsoft and IBM. Research has now been targeted on whether the technology can be applied in industrial sectors for business to shop-floor application integration.

The adoption of a robust approach of implementing SOA's and WS is required for the manufacturing system to meet the required performance from the end-users point of view in term of response time, reliability, proof of agility and re-configurability. The verification of these key requirements is necessary for the justification of this approach as an open standard for the next generation of manufacturing automation. The research outlined in this thesis is focussed on the integration of SOA's and Web Services technology within the paradigm of CBA to determine whether enhanced capability of the manufacturing system composed of embedded “web enabled” devices is evident

when compared to current manufacturing automation paradigms (i.e. programmable logic controller focused systems).

1.3 Research Questions

The research questions of this thesis that are expanded in following chapters can be stated as:

1. What are the current manufacturing paradigms? Why do they no longer meet industry expectations in the presence of globalised production characteristics? *(Refer to Chapter 2)*
2. How can agile automation systems benefit manufacturers especially in terms of design and manufacture cost savings as well as reducing production lead times? *(Refer to Chapter 3)*
3. What are the key features of Agile Manufacturing Systems as well as the key enablers that allow an effective solution for mass customisation? *(Refer to Chapter 3)*
4. What are existing approaches towards agile manufacturing paradigms? *(Refer to Chapter 3)*
5. What are current technologies using in distributed automation systems? *(Refer to Chapter 4)*
6. What are the problems with current automation systems and specific requirements for agile automation from the perspective of automotive manufacturers (i.e. the Ford Motor Company)? *(Refer to Chapter 5)*
7. What is the “state of the art” in control systems for agile manufacturing? *(Refer to Chapter 3, and 5)*
8. How could SOA's / WS fit into an agile manufacturing context and be implemented in an automation system? *(Refer to Chapter 6, 7, and 8)*
9. What are the network performance, deterministic communication and actual input and output device response time in the WS approach relative to a PLC control-based automation solution? *(Refer to Chapter 9)*
10. What is the degree of re-configurability, reusability as well as seamless integration capability in relation to the agility of the manufacturing system? This question leads to the design, implementation and evaluation of a university

based measurement platform of WS- based automation. How can figures of merit for these parameters be explicitly evaluated? (Refer to Chapter 9)

11. What is a suitable platform for the end users? What are the current control and build practices that could be improved by the use of WS technologies? (Refer to Chapter 5, and 6)

1.4 Research Focus

The movement toward agile and collaborative manufacturing has gained considerable attention from many research institutions and organizations keen to explore and develop a new manufacturing environment. Areas of research include the business enterprise level, shop floor automation, warehousing and outsourced manufacturing.

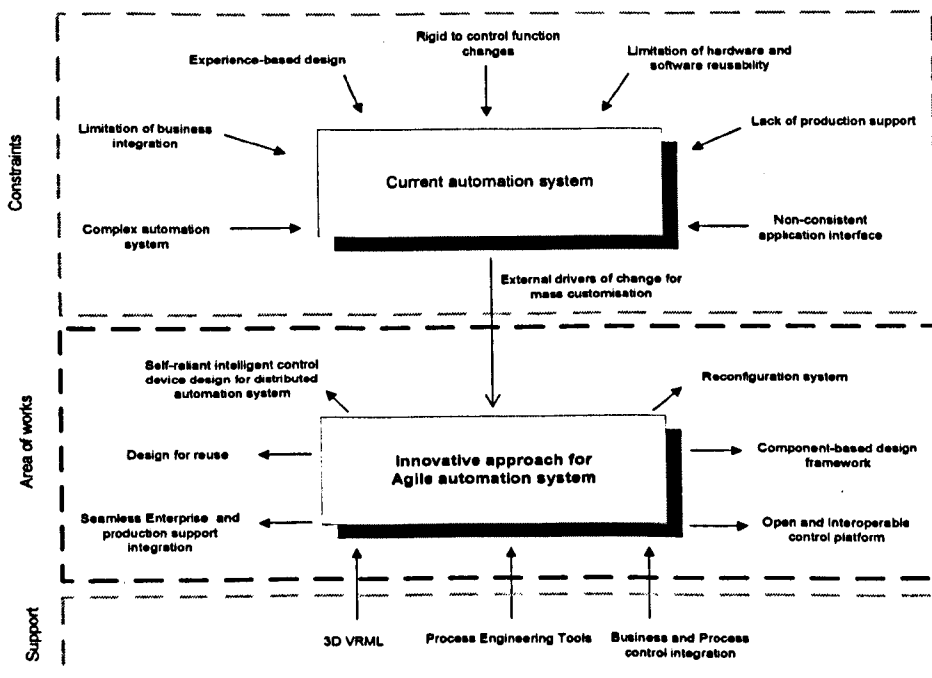


Figure 1-1: Research focus overview

As depicted in Figure 1-1, the primary aim of this thesis is to establish an innovative approach through *the component-based design of distributed intelligent automation systems* that is capable of supporting end-user requirements on *system re-configurability* and to create an automation platform that supports *business and production integration* to enable *agile manufacturing systems* throughout machine lifecycles. The primary application focus of the work is in the automotive domain, and in particular with the

Ford Motor Company Ltd. who is the primary industrial collaborator in this research. Control and communication technologies have become more mature and sophisticated and can currently be purchased as low cost, miniature, embedded devices. These technologies open up a new opportunity for the design of next generation manufacturing systems. Advanced embedded devices supporting TCP/IP networking communication could enable the design and implementation of control devices with greater interoperability between control devices, business and production support systems.

A methodology based upon the exploitation of Web technologies (SOA-WS) will be developed and evaluated with the aim of enhancing the lifecycle of manufacturing systems focusing on the aspect of autonomous distributed control and the adoption of a new paradigm for reconfigurable and reusable automation systems.

1.5 Organisation of the Thesis

The thesis is structured as follows:

Current Manufacturing Systems: Problems and Requirements

In Chapter 2, the literature relating to the trends, drivers of change and difficulties in current global markets is reviewed. Traditional manufacturing systems are reviewed to highlight the problem areas and the requirements of business and automation systems to enable the next generation of agile integrated manufacturing systems to be developed.

Agile Manufacturing

In Chapter 3 agile manufacturing is presented as the key concept for the future manufacturing approaches. The state of the art for enterprise integration and automation design and build are reviewed in this Chapter.

Manufacturing Automation Technologies and Key Enablers for Agile Automation

In Chapter 4 the current automation technologies for manufacturing systems are reviewed. The key enabling technologies both in software and hardware corresponding to agile automation requirements in open, distributed, reconfigurable and reusable control systems are detailed.

Research Focus, Design and Objective

In Chapter 5 the related research in the area of distributed automation systems for agile manufacturing is presented, including the novel framework of implementing Web service technologies with component-based design framework on automation system. The area of study and implementation is scoped on the research objective.

Design Methodology

In Chapter 6, a review and analysis of component-based approach is provided along with the design methodology for SOA-WS control systems. In addition, the SOA and WS concept are detailed and evaluated for the adoption of these technologies within the CB framework.

Industrial Case Study

The industrial case study implementing a component-based design on a PLC-based distributed control system and the implementation of Web Services on an embedded system (PC-based) are discussed in Chapter 7.

Test Rig Implementation and Industrial Demonstration

The development of WS- based automation via integrated embedded control devices is presented in Chapter 8, detailing the hardware and software architecture on the microcontroller device type supporting TCP/IP communication and a real time operating system RTOS. The industrial demonstration to illustrate the feasibility of Web Services automation systems within real control applications, supporting engineering tools and business integration as well as evaluation of system operation within a live production system environment are detailed.

Evaluation and Discussion

In Chapter 9, the key evaluation of performance, design, re-configurability and enterprise integration is assessed corresponding to the requirements of: (i) an agile automation system and (ii) the end-user requirements.

Future Work

The conclusions and future work based upon the exploitation of Web Services in industrial control systems are highlighted in Chapter 10.

CHAPTER 2

Manufacturing System Requirements and Trends

In this chapter, the drivers of change for agility-focused manufacturing paradigms are reviewed. Current manufacturing approaches are detailed and associated problems identified in order to highlight the requirements for next generation manufacturing automation systems.

2.1 Problem Statement

The objective of this chapter is to identify the features and characteristics of mass customisation in the global marketplace that have impacted on current manufacturing systems. The following research questions are addressed to support the specification of a new framework for the next generation manufacturing systems based upon the lifecycle requirements of the manufacturing supply chain.

1. How does globalisation impact the business and manufacturing strategy in automotive industry?
2. What are drivers of change that cause the manufacturing shift from mass production system to mass customisation?
3. What are the lifecycle requirements of the manufacturing system to cope with the highly competitive markets in the automotive sector?
4. What is the architecture and framework for the next generation manufacturing system?

2.2 Drivers of Change in Automotive Manufacturing

Global market competition and price pressure is leading companies to leverage new technologies to enhance their competitive advantage. They seek to differentiate their overall offering in order to compete [2]. The characteristics of global markets have been changed by the driver of customer satisfaction where greater choice of products from a larger number of companies is demanded. These factors have forced product variety to increase and batch size to decrease. In the automotive industry, reported trends that are having a dramatic impact on traditional mass production are [4]:

Global Competition

Many vehicle manufacturers face slow growing and saturated markets in their home countries [103]. Beside this, vehicle markets have become much more competitive by a growing number of firms selling cars in the mature markets such as the US, Germany, and Japan [106].

There has been a wave of new assembly and supplier plant construction in low wage economies and emerging markets such as China, India, Thailand, Vietnam, Brazil, Mexico and Eastern Europe. The trend is further enforced by host country requirements for local production and an effort by automotive manufacturers to cut costs within the context of regional trade arrangements such as the North American Free Trade Agreement (NAFTA) and the European Union [g28].

This shift in worldwide production of automobiles towards low-income countries only partly reflects increased competitive pressure. New suppliers such as China have expanded the production of automobiles for serving protected local markets, while lacking international competitiveness. However, several new suppliers, including Mexico, South Korea and Spain were quite successful in penetrating world automobile markets [108].

Customised Products with Low Volume Production

The consumption level of people is increasing along with the high-speed development of domestic economy. The number of *medium income* population is growing increasingly and the competition in the mid-grade automotive market has been intensified. Research has shown that the competition in the small car domain will continue to become more and more competitive, because such cars like the Masda C2, Chevrolet Aveo, Kia Rio have been targeted by a specific consumer group, that is young, fashionable and individualized. Since the economic power of the group is expanding quickly, its consuming capability and preference are affecting the automotive market strongly [g29].

The current and the future of global market trends in car industries as reported by Stephen Metzger [g30] are moving toward highly individualistic and diverse

markets in term of production types and functions. As a result, car manufacturers are focused on the ability, which quickly becomes a competitive requirement, to satisfy individual consumer tastes and preferences. Thus, the generic automotive market is comprised of a myriad of adaptations fitting the personal needs of the consumer, rather than so-called standard models.

Vehicle manufacturers, when introducing new products or concepts, are frequently uncertain of the precise market demand. The vehicle manufacturer normally assumes an initial low volume production. When the product grows in maturity the whole operation could be migrated seamlessly to the large volume manufacture as a going concern. Alternatively, if volumes do not reach expectation then the automation could be used to produce quite different components [111].

Reduction of Product Time to Market

As reported by Graves [g31], the average age of each Japanese vehicle is approximately two years, half that of their western competitors, and typically 500,000 vehicles are produced annually, compared with almost four times that number in the US and Europe. This assumed flexibility in capacity of model types and production volumes enables the Japanese producers not only to exploit their production and sales strategies to compete with the increasingly market demands of the world, but also to generate their own target customers in the global market place. In the presence of automotive markets characterised by rapidly changing technologies and innovations, it is important for car producers not only to incorporate the latest vehicles and technologies but also to offer products to customers in the shortest possible time.

Due to rapid changes in market demands in recent years, shortened product cycle is inevitably becoming the prevailing trend. For example, in the automotive industry, product life cycles has been shortened to 2–3 years as compared to the average 4–9 years from 1965-2000 and 3-4 years in 2005 [g33]. Additionally, market requirements demand significantly shorter new product realization cycles. Currently, it takes 24 months for the world's top automotive manufacturers to develop a new car. The trend is for development time to be

reduced to 12–18 months within the next five years [113] from 2004. However, these figures do not include the time for the development of the new car production line which normally takes 53 weeks [103] to complete.

It has been reported by P. R. Dean [3] that the mass production approach is not effective for mass customization production due to the large variations of customized products. Mass production was introduced in the later part of the 19th century during the Industrial Revolution, to support high volumes of customer demand at minimum cost (and minimum variety). The paradigm allowed manufacturers to produce more per worker-hour and to lower the labour cost of the end product. Mass production was successful because the market was characterised by stable demand, little product variety and a few competitors who dominated the market. Manufacturers and companies simply made profit by producing large batches of a product in order to minimise production costs.

The cost of variety can be interpreted as the sum of all the costs of attempting to offer customers variety with inflexible products that are produced in inflexible factories. This cost includes the actual costs of customizing or configuring products, all the setup costs, the costs of excessive numbers of parts, procedures, and processes, and the excessive operations costs. Under the mass production paradigm this cost increases exponentially with increasing marketed variety [g17].

Mass customisation is an emerging paradigm addressing the requirements of global markets that combines low price with extensive variation and adaptation. Products have to be manufactured at costs comparable with those items obtained by mass production techniques. At the same time, these products have to be highly customized not only in variety but also in parameters such as quantity and attributes [1]. There are many researchers trying to enable mass customisation by defining manufacturing systems that can respond quickly to changes of product types and customer demand. The details of alternative manufacturing paradigms proposed to enable mass customisation will be discussed later in section 2.4, focused around the next generation manufacturing systems.

However, in the following section, the traditional manufacturing architecture will be reviewed to provide an understanding of why such a conventional manufacturing architecture can no longer cope with these drivers of change in the global market. The associated problems with current manufacturing and production systems will be also highlighted.

2.3 Current Manufacturing Systems

Current manufacturing systems reviewed in this section will be focused on the system architecture (see Figure 2-1: Intra-Enterprise Domain) and manufacturing product lifecycle (see Figure 2-1: Lifecycle Domain).

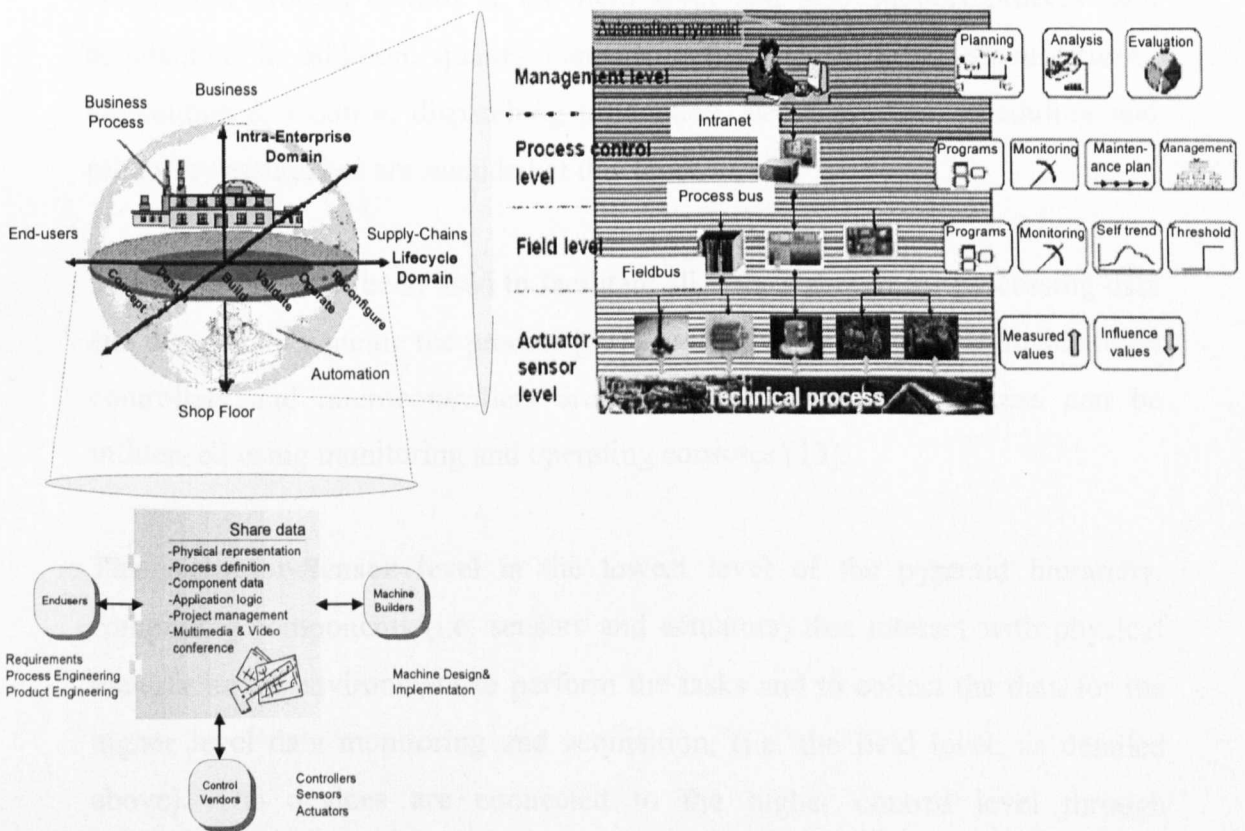


Figure 2-1: Manufacturing System Structure and Production Process

2.3.1 Manufacturing System Architecture

Current machine control systems are categorized according to their physical functionalities (e.g. programmable logic control, motion control, regulators) and are programmed separately to execute sequences of commands as function primitives.

Communication between the individual controllers is typically facilitated by a central system in a hierarchical network. A typical automation hierarchy based on the standard architecture ANSI/ISA-95, shown in Figure 2-1 (Intra-Enterprise Domain), is described as comprising a number of integrated levels as follows:

The Management level is concerned with the management of enterprise-level finance, resource planning and distribution, workflow planning and order management and fulfilment.

The Process control level has the primary task of the supervision of sites such as SCADA (Supervisory Control And Data Acquisition) to monitor the automation process control at the field level and also support process data acquisition. In addition, quality management, order tracking, Manufacturing Operations & Control, dispatching production, detail product scheduling and reliability assurances are included at this level.

The Field level has been used to facilitate all tasks required for processing data and directly influencing the process [12]. At the field level, programmable logic controllers and microcontrollers are used. In addition the process can be influenced using monitoring and operating consoles [13].

The Actuator-Sensor level is the lowest level of the pyramid hierarchy, comprising components (i.e. sensors and actuators) that interact with physical manufacturing environment to perform the tasks and to collect the data for the higher level data monitoring and acquisition, (i.e. the field level, as detailed above). The devices are connected to the higher control level through communication lines such as fieldbuses or industrial networks. Due to timing constraints that have to be strictly observed in an automation process, the applications at the field level require cyclic transport functions that transmit source information at regular intervals. The data representation must be as compact as possible in order to reduce message transfer time on the bus [6].

2.3.2 Lifecycle of Manufacturing System and Machine Design

Consumer products change frequently with lifecycles often measured in months rather than years. Shorter product life is forcing ever-shorter production machinery development cycles. Bringing products rapidly to market may only be possible through compressing time-scales by concurrently engineering the product, the production machinery and the manufacturing and distribution facilities [79].

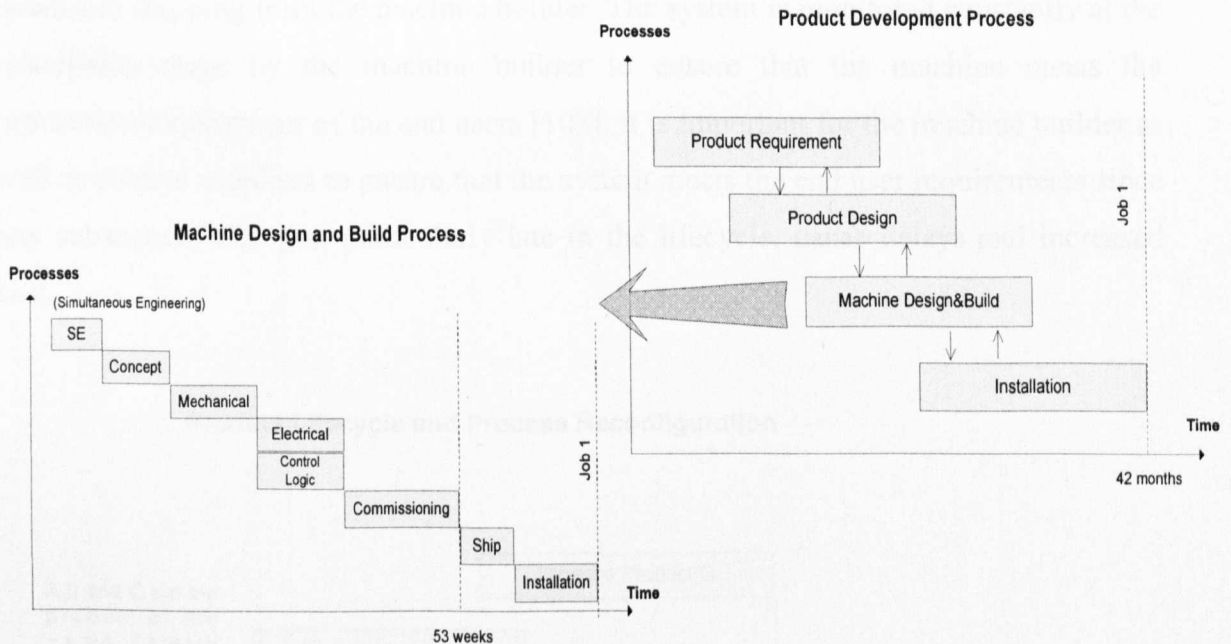


Figure 2-2: Product Lifecycle and Machine Design & Build Process [103]

The engine product design and implementation process in typical automotive industries takes approximately 42 months to complete [103]. The activities that support the concurrent engineering of product, machine and process facility are illustrated in Figure 2-2. The process involves three major collaborators: the end-user (the automotive production company), the machine builders and component suppliers (control components). The process begins with the conceptualisation of the new engine type and the determination of a set of product requirements and specifications, which are later translated into the design of the product. In the current case, before the product design is finalised, the end-user will contact the machine builder as well as the component suppliers to provide the design and build the manufacturing system. The process of machine design and build normally takes 53 weeks to complete with the machine validation carried out at the machine builder's site. At each stage of the machine

lifecycle the system is checked and closely monitored by engineers from the end-user to guarantee that the system meets the required specification.

It is noted that full system operation and functional performance of the machinery is carried out on completion at the machine builder's site. Additionally full system validation is undertaken on the end user's site during installation and tuning post strip down and shipping from the machine builder. The system is monitored constantly at the installation stage by the machine builder to ensure that the machine meets the production requirement of the end users [103]. It is important for the machine builder as well as control suppliers to ensure that the system meets the end user requirements since any subsequent changes, particularly late in the lifecycle, cause delays and increased cost.

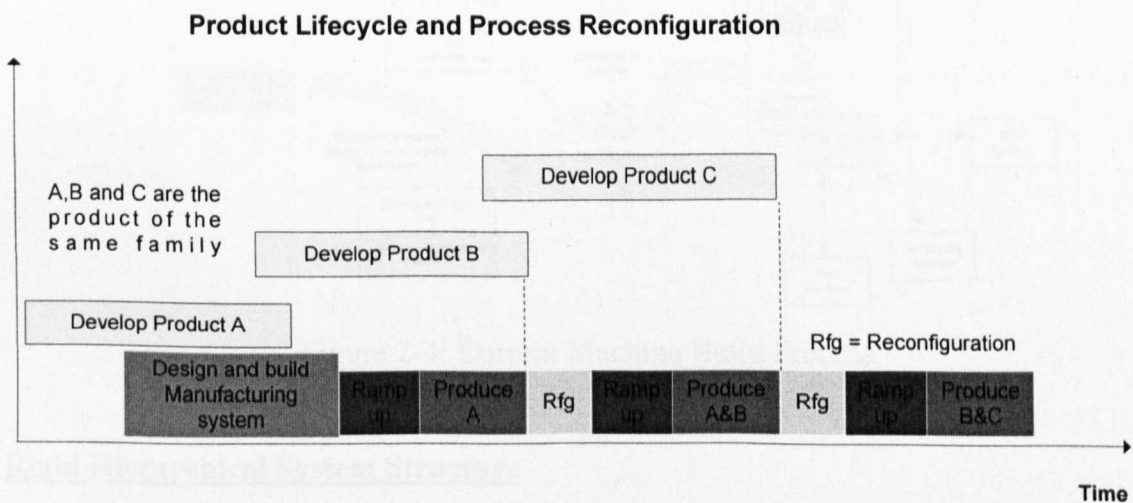


Figure 2-3: Product Lifecycle and Reconfiguration Process [79]

During the machine lifecycle as shown in Figure 2-3, production lines are subjected to system re-configurations for new types of products. Since every product type change results in a ramp up time to full production, there is a growing need for production machinery to support re-configuration and re-configurability more efficiently in order to maximise return on investment [79].

2.3.3 Problem Identification

The set of problems and issues surrounding the current manufacturing architectures and control systems have been derived from international projects and research collaborators (e.g. Schneider Electric, Comau, SOCRADES ⁽⁵⁾, RIMACS ⁽⁶⁾ and SAP) and the Ford Motor Company, UK. The common problems with traditional manufacturing systems associated with the automation systems, Figure 2-4, can be summarised as follows:

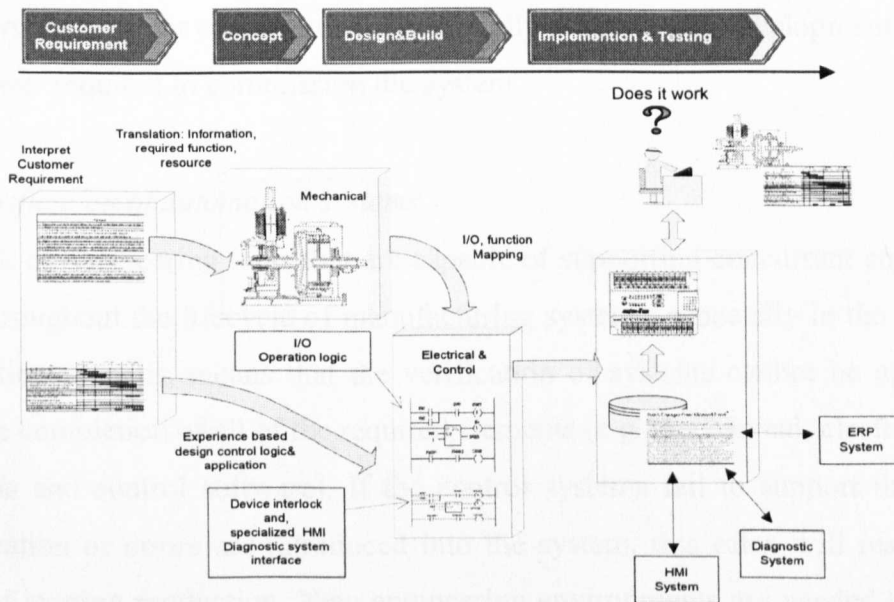


Figure 2-4: Current Machine Build Process

Rigid Hierarchical System Structure

- This traditional design approach has major deficiencies when used as a basis for an intelligent (reconfigurable) manufacturing control structure [9]. In a rigid, hierarchical approach, system development typically occurs as a series of vertically isolated activities [72]. It has been reported in [10] and [11] that the centralized and sequential manufacturing planning, scheduling and control mechanisms are increasingly being found insufficient for the current market environment due to a lack of flexibility (e.g. the ability to respond in a time and cost efficient manner to *planned* changes) and agility (e.g. the ability to respond in a time and cost efficient manner to *unplanned* changes) in order to respond to changing production styles and highly dynamic variations in production requirements.

5- Service-Oriented Cross-layer infrastructure for Distributed smart Embedded devices

6- Radically Innovative Mechatronics and Advanced Control Systems

Design of Automation Systems

- *Experience-based design and implementation*

Current automation systems are composed of diverse automation peripherals that make the system very complex. Therefore, the process of designing and developing automation systems is reliant upon the expertise of those who have participated in previous similar projects [33]. A lot of effort is still required to build consistent automation systems within the limited time available, even by the experts. The reuse of hardware and software from existing designs by employing high level development tools for example could potentially decrease the development time and man power required to commission the system.

- *Late verification of automation systems*

The lack of engineering tools that are capable of supporting concurrent engineering tasks throughout the lifecycle of manufacturing systems, especially in the design of automation systems, means that the verification of systems cannot be undertaken until the completion of all of the required elements (e.g. mechanical, electrical, fluid elements and control software). If the control systems fail to support the desired specification or errors are introduced into the system, this often will result in the delay of starting production. New engineering environments are needed to support concurrent automation systems design. The development of a component-based design methodology (*Chapter 6*), supports the concurrent design of automation systems by combining hardware and control software into an integrated automation unit (component) in which the control software can be verified before the completion of mechanical systems by simulation of embedded device states.

Control Systems

- *Rigid manufacturing development with centralized control*

On the shop floor, centralized manufacturing systems are not favored because they involve expensive investment costs at the implementation stage. In addition, if the product and manufacturing management systems are modified and the central control unit has to be altered, the whole system needs to be changed because devices or elements under the central control unit are reliant upon the central

controller. This type of the system is also at risk of a single point failure that can result in the whole system being inoperable.

- *Lack of reusable and reconfigurable automation peripherals*

Within current automation systems, the hardware is tightly coupled with the centralized control software systems such as traditional PLC's. Reusability and reconfigurability are difficult to support and the whole system is usually replaced when new designs are required. Changing types of hardware may also result in a requirement for new control software. Supporting these changes is a time consuming and error-prone process.

- *Diversity and complexity of automation devices*

Manufacturing systems are composed of various types of software applications such as Human Machine Interfaces (HMI's), data monitoring systems, control systems, gauges and data recording systems. Each unit has its own proprietary solution provided by different vendors working on different operational platforms such that Microsoft Windows, UNIX with serial communications, Ethernet, Fieldbus, or Modbus communication mechanisms. This makes the automation system very expensive and complex to maintain since it necessitates the fragmented use of heterogeneous tools and experts with knowledge of diverse technologies.

Operation

- *Lack of remote diagnostics and support*

Due to a lack of remote support, end-users cannot get immediate support from machine vendors when machines breakdown. The most cost effective option seems to be the difficult task of solving the problem over the telephone. The major issue with this is that engineers frequently do not have enough information about the cause of the breakdown to support efficient recovery to production. In many cases, a site visit is required which can be a problem when global support is required. In addition to cost penalties, machines can be out of production for a long time waiting for support. Down time can be significantly reduced if problems can be solved remotely via e.g. online expert assistance.

2.3.4 Manufacturing and Automation System Requirements

In section 2.3.3, the problems with current automation systems were identified as a set of requirements for the next generation manufacturing automation systems. The required attributes for advanced automotive manufacturing and automation systems are as follows:

Seamless Business- Manufacturing Process Integration – Rapid appreciation and preparation of functional and technological changes need to be appreciated throughout the business and by investment personnel. It is desired that the system can react to changes with less (or ideally without) effect on other processes within the complete manufacturing system.

Rapid Design of the Automation System – There is a need for an approach to allow the simultaneous system development of mechanical, electrical, fluid and control units. It is required that these units are designed independently and yet available in the common formats for the interest parties (mechanical, electrical, fluid, controls engineers and machine vendors) in building the complete system.

Quick Response to Change in the Production Capacity – Re-configurability of automation is the design requirement at the outset to enable support of rapid changes in the structure of software and hardware as described by D. A. Vera [116], to adjust production capacity and functionality in response to changing demand.

A High Degree of Reuse – It is desired to reuse as many as possible of the features (i.e. hardware, software design and engineering knowledge) of the existing system within the new manufacturing automation system in order to save time and cost of the development.

A More Consistent Integration Approach for Diagnostic and Maintenance Support – Consistent integration approaches are required to minimize the process downtime and the unnecessary waste. The integration effort and degree of complexity of such systems need to be easier to integrate without incurring added complexity.

Visualisation and Simulation – The functional and behavioral capability of the control system needs to be evaluated prior to installation in order to minimize time lost due to late validation.

Non-Vendor Specific Platforms – Open platforms [43, 44] allow the substitution of components with alternatives to improve capability, reliability or performance. This would also benefit the end-users in terms of reducing development costs and improving system flexibility and avoiding being “locked into” particular technological solutions.

Fault Tolerance and Recovery – Fault tolerance and error recovery capabilities enable systems to continue to operate efficiently in the event of the failure of some components. This is a significant requirement for distributed automation systems. A single failure should have a minimal impact on the rest of the system functionality back up procedures to recover safely (i.e. roll-back, roll-forward) need to be supported without interrupting the operation of other units. Checkpoints should be implemented in the system to monitor constantly the system operational state.

As highlighted in this section, these fundamental characteristics of automation systems need to be considered as key enablers of the Next Generation Manufacturing Systems (NGMS's) to enable mass customization. The key characteristic and the emerging trends of manufacturing systems are highlighted in the following section.

2.4 The Next Generation Manufacturing System

As outlined by François Jammes, Harm Smit [9], the next generation manufacturing systems of future manufacturing enterprises will be characterised by a need to adapt to frequently changing market demands, time-to-market pressures, continuously emerging new technologies and, above all, global competition. Therefore the future manufacturing system must support global competitiveness, innovation, introduction of new products and strong market responsiveness.

However, traditional rigid sequential engineering methods are inappropriate in this context as common problems detailed in sections 2.3. Furthermore the manufacturing architecture and automation structure within the complete engineering process for current production systems is inefficient and requires extensive human effort in terms of time, cost, and expertise in developing desired future manufacturing systems [72]. In addition, it has been reported [9] that the cost of maintenance and adaptation costs rises considerably for a non standard, obsolete and inflexible platform. This is due to the difficulty of reconfiguring existing applications (i.e. processes) to a new configuration, vendor specific hardware and software and tight integration between equipment and manufacturing control systems.

To overcome the common problem with traditional manufacturing systems (*section 2.3.3*) and to meet the manufacturing requirements (*section 2.3.4*) enabled by mass customisation, novel integrated business and manufacturing strategies are required. The need for that new manufacturing environments that are capable of supporting inherently multidisciplinary, parallel system engineering tasks has been discussed in [72]. The realisation of appropriate engineering tools requires not only a broad appreciation of mechatronics, manufacturing strategies, planning and operation, but also a deep understanding of the required integration of communications, information and advanced control functionality.

In relation to the above statement, agent-based agile manufacturing cells and Web-based agile manufacturing frameworks [22, 29, 77 and 92] have been proposed as possible solutions that could meet the challenges and requirements from various industrial sectors. In addition these agile manufacturing frameworks have been proposed with key characteristics (e.g. a neutral- platform system, reconfigurable and reusable manufacturing systems, concurrent manufacturing and automation design, and seamless business-shop floor integration) that can contribute to distributed collaborative manufacturing teamwork at both the enterprise and plant levels in order to support flexible and quickly adaptable manufacturing process capabilities.

The Emergence of Collaborative Automation in Agile Manufacturing

The empirical study on an appropriate agile automation framework has been carried out on research projects, e.g. RIMACS, SOCRADES, Business Driven Automation (BDA) and discussed in collaborative manufacturing and automation publications [2, 32, and 72]. This research has provided the fundamental framework and the core background to the research outlined in this thesis. A common framework for supporting the lifecycle of agile systems is illustrated in Figure 2- 5.

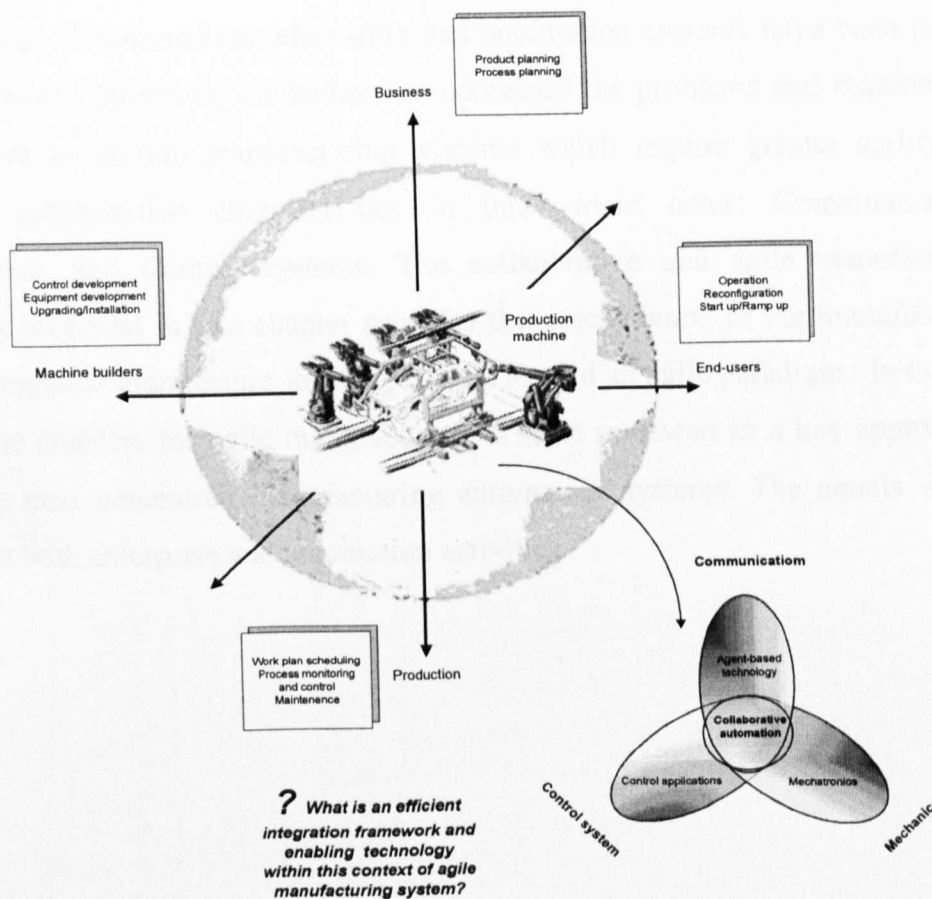


Figure 2-5: Collaborative and Agile Manufacturing Framework

The framework covers the business management, the build of the control system and the production operation during the manufacturing lifecycle. The collaborative automation system is embedded with the core functionality to support the business and manufacturing functions as shown in the diagram. The automation system is composed of Control System, Mechatronics and

Communication capability required to support agile operation throughout the lifecycle. However, as is the case with common frameworks, various instantiations of solutions have been employed by different researchers. These will be discussed in detail in Chapter 3.

In this chapter, the drivers of manufacturing paradigm shifts from mass production to mass customisation have been discussed. Accommodation of these drivers has impact on the requirements of future manufacturing automation systems that need to support greater agility and be more cost effective in the presence of rapidly changing environments. Traditional manufacturing and automation systems have been proven to be inefficient. However, the author has addressed the problems and requirements for the next generation manufacturing systems which require greater agility and enhanced collaborative characteristics in three main areas: Communications, Mechatronics, and Control systems. The collaborative and agile manufacturing framework proposed in this chapter provides the precise view of the manufacturing system integration architecture for this research toward an agile paradigm. In the next chapter, the enablers for agile manufacturing will be reviewed as a key approach to enable the next generation manufacturing automation systems. The details will be focused on both enterprise and automation activities.

CHAPTER 3

Agile Manufacturing Paradigm

In the previous chapter, the need for greater agility in manufacturing has been addressed with the definition of the specific requirements of automation to support the next generation manufacturing capability. In this chapter, the details of agile manufacturing approaches are discussed. The definition and the characteristics of the facets of agility in manufacturing systems are identified. In addition, existing solutions for agile manufacturing systems are assessed in terms of integration platforms, implemented technologies, problems and limitations in each approach. In comparison to these solutions, a platform for automation systems based upon a Service-Oriented Architecture is proposed to address the limitations inherent in existing approaches.

3.1 Problem Statement

There are various definitions of agility and key facets presented by different researchers. It is important to determine the commonality in core concepts and enabling technologies that can be applied in the research outlined in this thesis in the domain of the automotive industry. This chapter aims to address the following questions:

1. What are the key characteristics of the agile manufacturing paradigm?
2. What are previous and current projects aim at developing agile manufacturing platforms? How do these solutions address the needs and the requirements from the end-user?
3. What features constitute an effective agile manufacturing integration framework and enabling technologies that can meet end user requirements?
4. What are the features of the new approach that could potentially establish the agile paradigm in the automotive automation domain?

3.2 Overview of Agile Manufacturing

The concept of Agile Manufacturing Systems (AM) was first introduced in 1991 [119]. Since then, the concepts have been widely disseminated as embodying the essential requirements for next generation and intelligent manufacturing systems.

Although agile manufacturing can be regarded as a relatively new automation paradigm it is not well defined or understood. Agile manufacturing often is confused with lean production, flexible manufacturing or Computer Integrated Manufacturing (CIM), but it has a distinctly different meaning.

Common definitions of agile manufacturing are:

“Agile manufacturing as the ability to accomplish rapid changeover from the assembly of one product to the assembly of another product. Rapid hardware changeover is made possible through the use of robots, flexible part feeders, modular grippers and modular assembly hardware” [85]

“The concept of agile manufacturing is also built around the synthesis of a number of enterprises that each have some core skills or competencies which they bring to a joint venturing operation, which is based on using each partners facilities and resources” [64]

“A manufacturing system with extraordinary capabilities (Internal capabilities: hard and soft technologies, human resources, educated management, information) to meet the rapidly changing needs of the marketplace (speed, flexibility, customers, competitors, suppliers, infrastructure, responsiveness)” [119]

These definitions reflect the various approaches towards agility in business, product, production and entities of the manufacturing system to achieve a common goal of coping with the rapid change of global markets driven by customer demand. As reported by A. Gunasekaran [8], the concept of agility is at the heart of manufacturing systems integration for supply chain companies aiming to perform collaborative tasks to deal with unpredictable and rapid changes in both business and production systems. Ideally this group of companies would “partner” to form an integrated enterprise

supporting a range of the best available resources for the business opportunities of interest.

To become agile, manufacturers have to distribute intelligence and decision making authority as close to the points of delivery, sale and even after-sale service as possible. To improve agility, they have to integrate the design and production information with their business partners [77].

Agility is required, in many areas throughout the manufacturing lifecycle i.e. from the business requirements definition stages in product design to the manufacturing and reconfiguration production resources [4], [7] and [8].

The agility of these entities during the manufacturing system design lifecycle is facilitated by the agility in *management, people, organisation, production, system design and marketing* as shown in Figure 3-1. The multi-faceted nature of automation and production systems agility is also depicted in Table 3-1.

The research outlined in this thesis is focused on the production and the systems integration facets of agility. The aim is to develop flexible and reconfigurable production lines enabling effective support of the variation of product types and volumes in short lead times. Systems integration aspects have been focused on production and business integration in order to utilise and share information with the aim of reducing the cost and improving the throughput of the production system. This will also improve business collaboration and support through effective database, visualisation and monitoring systems integration.

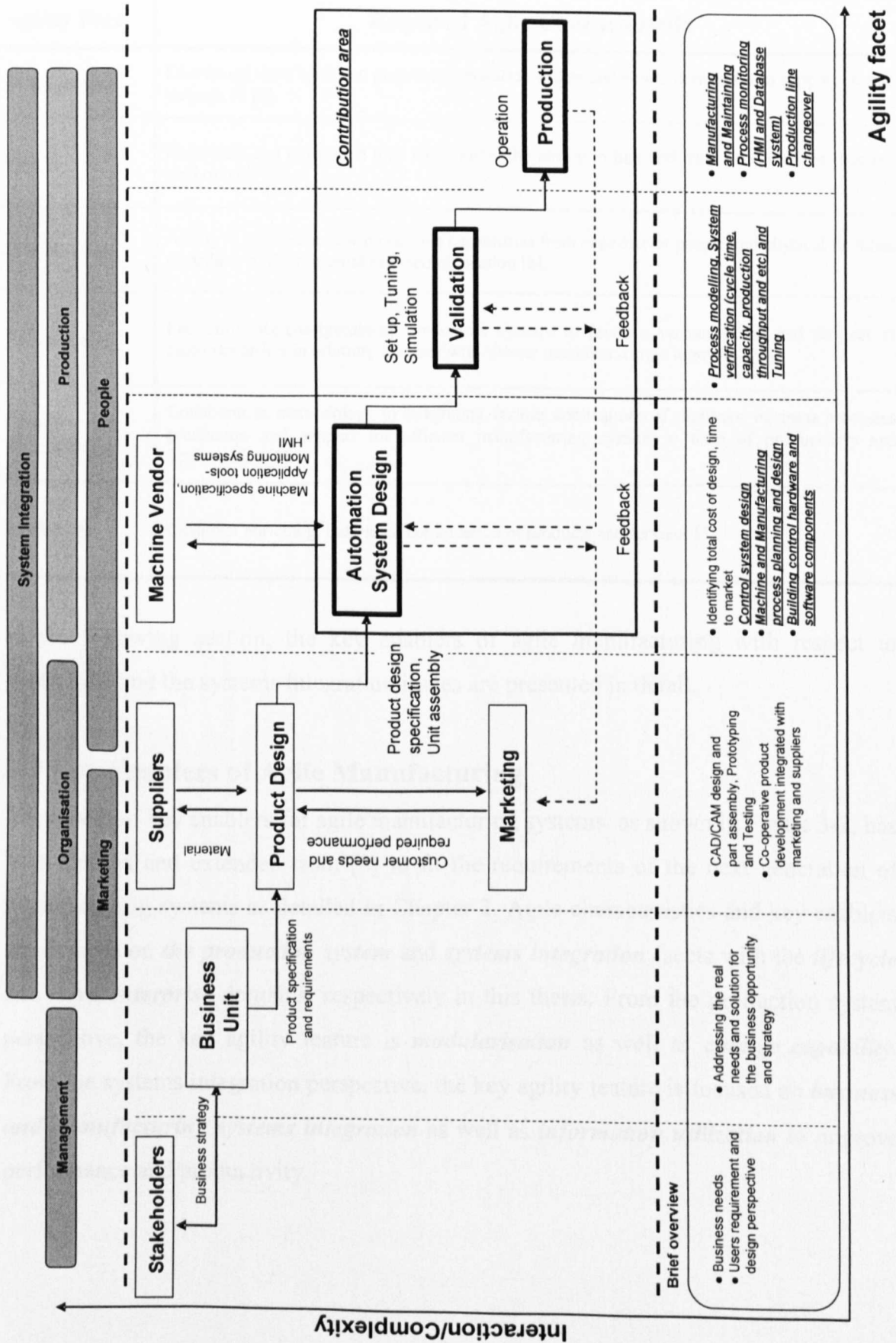


Figure 3-1: Agility and Interaction in Manufacturing System Design

Table 3-1: Multi-Facet of Agility in Manufacturing [7]

Agility Facet	Required Agile Characteristic
Management	Distributed team work and project collaboration within and across companies to support change through IT [8].
People	Multi-skill and innovative total work force, the ability to hire and train people to the required skill quickly [8].
Organisation	Ability to synthesize new productive capabilities from expertise of people and physical facilities regardless of their internal or external location [8].
Production	Flexibility/ Re-configurability production systems to produce various goods and services to customer orders in arbitrary lot sizes with shorter manufacturing's lifecycle [72].
System Integration	Collaborative methodology in integrating various applications of suppliers, business processes production and support for efficient manufacturing system in term of productivity and performance [2].
Marketing	Customer enriching, individual combination of products and services [7].

In the following section, the key enablers of agile manufacturing with respect to production and the systems integration issues are presented in detail.

3.3 Key Enablers of Agile Manufacturing

The model of key enablers for agile manufacturing systems, as shown in Figure 3-2, has been studied and extended from [8] to fit the requirements of the next generation of manufacturing systems as detailed in Chapter 2. Agile characteristics and key enablers are focused on *the production system* and *systems integration* facets with the *lifecycle* and *intra-enterprise* domains, respectively in this thesis. From the production system perspective, the key agility feature is *modularisation* as well as *change capability*. From the systems integration perspective, the key agility feature is focused on *business and manufacturing systems integration* as well as *information utilization* to improve performance and productivity.

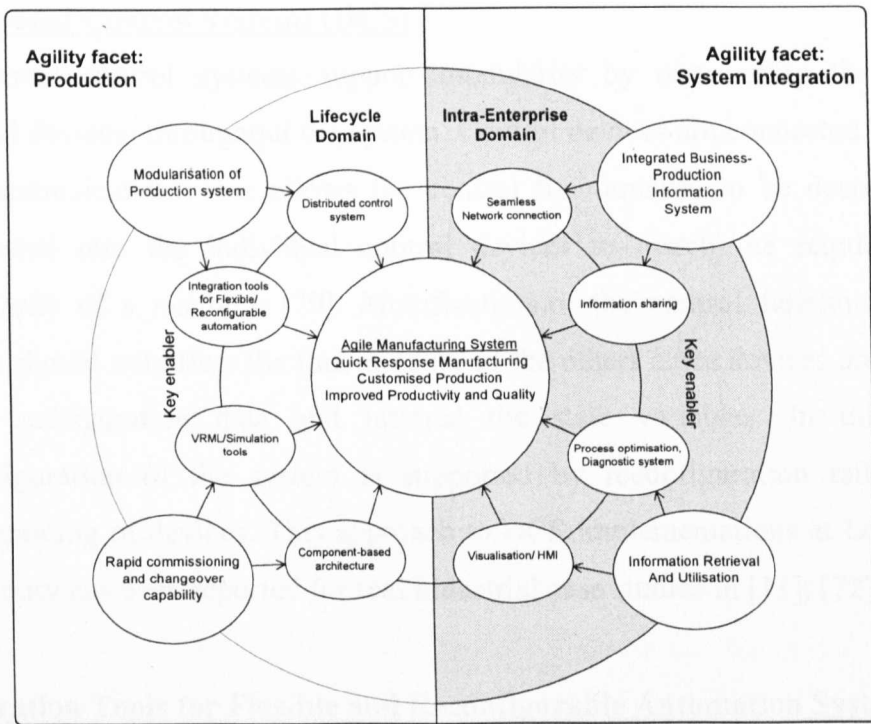


Figure 3-2: Conceptual Model of Key Enablers for Agile Manufacturing System

The details of these agility features are as follows:

1). Modularisation of Production Systems

Modularity is typically introduced into a manufacturing operation to increase the flexibility of the operation both in terms of its range of functions and also its ability to be easily reconfigured in the face of changing conditions [115]. Modularity almost always leads to a distribution of functionality and also physical distribution [91]. Modularisation of production allows changes to be made to a few isolated functional elements of the production without necessarily affecting the design of other elements [59].

Modular production system design research has proposed solutions for both reconfigurable mechanical structures and also control software applications enabling quick changeovers in decentralized automation systems to improve flexibility and adaptability of machine systems or production cells as presented in [72], [79], [115]. Key modularisation concepts are: i) distributed control of loosely coupled structures and their functionalities (control software) and ii) integration tools to support the re-configuration process and design for reuse of the control system.

Distributed Control Systems (DCS)

Distributed control systems support modularity by distributing the controllers (control devices) throughout the system. Control devices are connected by networks for communication. This allows the control functionality to be decomposed and distributed into the individual control devices to match the required physical modularity of a machine [79]. Modification of the control functionality of one device should not affect the functionality of the others since devices are interlocked using configuration data and internal the state variables. In this way, the reconfiguration of the system is supported by reconfiguration rather than re-programming of devices. This approach to DCS implementations at Loughborough University has been reported for real industrial case studies in [11], [72], [73], [79].

Integration Tools for Flexible and Reconfigurable Automation Systems

The concept of modularisation applies to the design of modular machines where ad hoc parts could be installed with minimal impact on the complete machine system. This concept supports re-configurability where physical mechanical parts and their control software can be changed without altering other system components. To support the adoption of this modular decomposition of control systems, integrated engineering tools are required for building, changing and managing machine applications, for example, synchronization, and internal and external interlocking between control devices.

2). Rapid Commissioning and Changeover Capability of Automation Systems

Manufacturers seeking more agility and flexibility of production systems have increased the demands for an integrated engineering environment to support the simultaneous design of the manufacturing system, especially from the control system perspective. New engineering automation design approaches will need to support concurrent design activities of those who are involved in various design activities addressing machine structures, electrical components, fluid components and control software. Central to the research in this thesis is that this requirement can be enabled by a component-based design approach (*detailed in Chapter 6*). In addition, it is important to be able to validate the design of the machine system (via for example simulation tools) in early lifecycle stages in order to detect failures and deviations from requirements and specifications as early (and hence economically) as possible.

Hence the strategy to support rapid change capability is via the integration of individual mechanical, electrical, and control components within concurrent development and validation processes of manufacturing systems. The enabling concepts are:

Component-Based Architectures

In typical component-based architectures, components are made up of mechanical units, internal state based control software, control interfaces and physical representations i.e. “virtual” CAD drawings. Examples of the hierarchical breakdown in terms of *system-subsystems-modules-components-elements* adopted in this thesis are detailed in Chapter 6. The control components: (i) are responsible for their own actions and (ii) monitor the necessary sequence and interlock conditions of other components in the system in order to fulfill the overall application requirements [79].

In the design of component-based systems, each component is considered as a reusable entity for independent control configurations throughout the manufacturing lifecycle. The component control software: (i) consists of the low-level logic (i.e. actuation, sensing, interfacing), (ii) defines the devices behavior, (iii) is encapsulated in the controller and (iv) is exposed to integrated engineering tools through specific interfaces for building manufacturing applications.

Virtual Reality Modelling and Simulation Tools

In current systems, control verification can only be carried out during commissioning phases when all the mechanical, electrical and control parts have been integrated [103]. Revision and re-design at this stage results in the costly delays for the project. In the current CB approach [116], the logic of the *real* control application (i.e. embedded within the integrated components) can be evaluated via simulation tools to provide the more reliable and accurate approach for the system validation. The system validation can be carried out prior to the completion of the design process; significantly earlier than within the current approaches. Any defects / deviations from specification can be determined early in the lifecycle and changed quickly and cost effectively.

3). Integrated Business-Production Information System

Various kinds of critical information can be found distributed at the various levels within manufacturing enterprises i.e. at the business level and / or the shop floor (i.e. barcodes, status of devices, production capability, quality metrics). This information needs to be available to people or relevant systems whenever or wherever they need it across the enterprise. There is a trend of increasing the intelligence of automation devices at the shop floor level to enhance flexibility and re-configurability by increasing information transparency and data mobility across heterogeneous platform systems [45].

Information Sharing

In the collaborative working environment where the information is distributed and shared across departments, the use of a shared ontology allows a better exchange of information among team members with similar interests and a better access to information. For example the industrial design engineer can use a *sharing module* to elaborate on design documentation and detail new knowledge by integrating existing documents and content descriptions [55], thus reducing the design life cycle of products and machines. This approach is also applicable to other manufacturing activities such the machine maintenance to support learning from the outcomes of previous projects to avoid previous errors and to solve new problems [g10].

Seamless Network Connection

To gain the maximum benefit from knowledge sharing, content needs to be characterised in an explicit way so that others can access and understand the data. Tools have been proposed that support information content descriptions indexed with metadata and embedded annotations within documents or geographical document zones for better content exploitation and sharing [55].

4). Information Retrieval and Utilisation

The current trends toward the application of increasingly sophisticated devices on the shop floor are driving an approach for collaborative manufacturing where the collection, dissemination and analysis of information about production operation is recognised to be strategically as important, if not more important, than the physical products produced [2]. It is important that information systems are organised in

standard formats allowing other entities to understand and use the information effectively [8]. With the high-speed data communication of Ethernet and Fieldbus technologies, distributed devices at shop-floor level can be monitored, analysed in real-time and configured or upgraded during run-time. This enables enterprises to become more open, flexible, distributed and extensible with less cost. Shop floor information can be utilised in many areas:

Diagnostic Systems

The data collection from low-level manufacturing devices, including a wide variety of process and production data (such as temperatures, pressures, flow rates, and ID tags), helps companies monitor and analyse the current plant performance in real-time. The utilisation of data from any individual piece of equipment can be collected via networked communication (e.g. Fieldbus and Ethernet TCP/IP). Using these techniques, process downtime, parts degradation, faults and throughput can be monitored, visualized and evaluated to determine plant asset performance and support rapid deployment of recovery activities.

Process Optimisation

Advanced technology has been increasingly used in the manufacturing process as an aid for operators in optimizing operations, the simulation of performance and advanced process control. In addition plant managers need information systems support for effective product planning and scheduling. Models, simulations, and information generated by processes are beginning to be integrated and leveraged at the enterprise level [17, 29, and 116]. In the current business environment, on time delivery to customers and optimised process scheduling have become significant drivers for companies. Equipment downtime resulting in late delivery adversely affects customer satisfaction. The concept of running machines and field devices has been changed from reactive run-to-fail maintenance to predictive and proactive maintenance, aiming at near zero downtime [2].

Condition monitoring, implemented with intelligent devices, is under development by many vendor companies aiming to provide solutions for data logging, process monitoring and real-time, on-line device controls in many aspects of industrial domains to improve efficiency and enhance production throughput by minimizing

loss associated with the machine breakdown (e.g. SOCRADES, RIMACS EU projects).

Visualization and HMI

Human Machine Interfaces (HMI's) and Supervisory Control And Data Acquisition (SCADA) systems are the traditional tools for providing control and performance visualisation at the plant level. Currently these systems are the main human interfaces typically transferring significant amounts of data to be converted into information that is utilised by production management systems. In this aspect, there is a trend toward more open process control systems, which allow data to be collected freely from a variety of vendor equipment and prevents manufacturers from being locked into proprietary solutions and ultimately increasing the plant flexibility significantly.

3.4 Related Approaches toward Agile Manufacturing Systems

In this section, the general framework, existing research and solutions to enable agile manufacturing systems in the Intra-Enterprise and Lifecycle domains (*see section 3.3*) are discussed in terms of enabling technology and its limitations.

The dimension of manufacturing system designs that leads to the collaborative enterprise perspective is illustrated in Figure 3-3. In the diagram internal manufacturing, business processes (axis-1) and Inter-Enterprise connectivity (axis-3) between local business and external business partners are represented. Manufacturing plant, machine control vendors, and machine builders are linked throughout the lifecycles of the manufacturing (axis-2) domain.

The collaborative enterprise model represents the general framework and integrated technologies that are commonly used in the design of integrated manufacturing systems within any business and engineering application. Note: the Inter-Enterprise domain (axis-3) is not included in the discussion since this domain is outside of the scope of author's research.

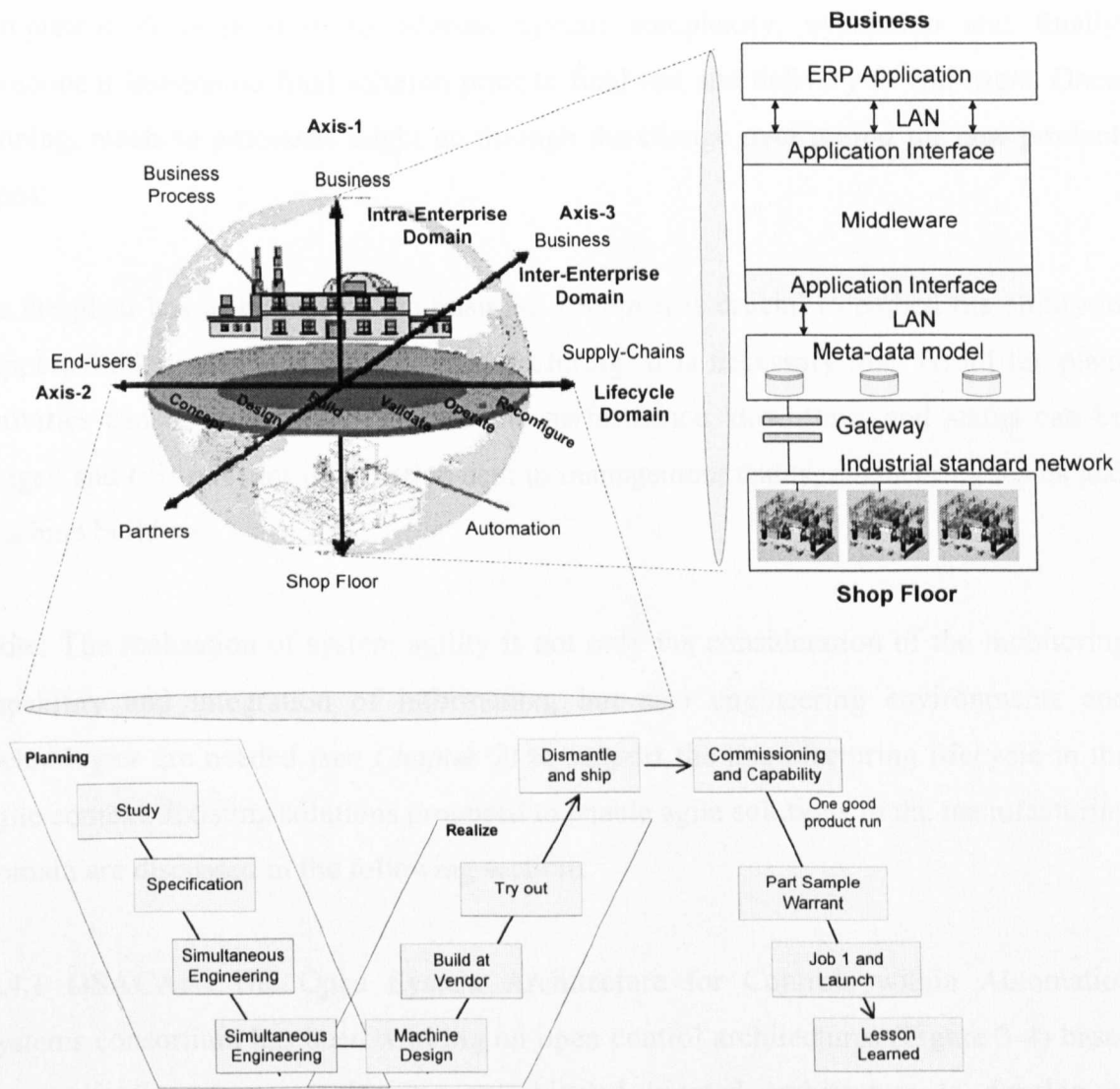


Figure 3-3: Collaborative Enterprise Model (extended from [78])

In the Intra-Enterprise Domain, business applications, manufacturing execution systems and other control applications are integrated into the manufacturing system through middleware and interface concepts to support information retrieval from database systems for use in higher level applications. Middleware is software specifically designed to integrate disparate software applications in heterogeneous environments [120, 45]. Networked communication is supported via standard office LANs at the business level and Gateway proxies for shop floors with different network types (e.g. Fieldbus, Industrial LAN).

Shop floor capability is the main focus of the research in this thesis i.e. automation issues within the manufacturing lifecycle. Typically the design of automation systems has followed the V-Model [78] starting from extracting requirements from end-users,

component decomposition to address system complexity, evaluation and finally component integration final solution prior to final test and delivery to end-users. Once running, machine processes might go through the change over period for new product types.

As the plant has to be part of the business system it is crucial to embed the ability to support the key elements of agile manufacturing. It is necessary that: (i) all the plant activities can be monitored, (ii) machine performance, downtime, and status can be logged and (iii) relevant information sent to management teams, engineering teams and machine builders.

Note: The realisation of system agility is not only via consideration of the monitoring capability and integration of information, but also engineering environments and technologies are needed (*see Chapter 2*) to support the manufacturing lifecycle in the agile context. Existing solutions proposed to enable agile solutions in the manufacturing domain are discussed in the following sections.

3.4.1 OSACA: The Open System Architecture for Controls within Automation systems consortium has been working on open control architectures (Figure 3-4) based on a client/server protocol using an objected-oriented architecture to develop an independent modular software structure for open control systems [g34]. The group has developed a communication system to support open data exchange between software modules within control systems. The reference architecture defines which tasks are performed and how tasks interact with each other. The OSACA platform supports interoperability between different vendor solutions. This vendor-neutral capability is implemented by the OSACA API (Application Programming Interface), which acts as an interface between the application object and the underlying infrastructure in the form of communication, operating system and electrical components. To enable the flexibility of manufacturing systems in distributed control environments and the interoperability among diverse vendor specific hardware types, the control application is defined as an object (i.e. a device), which encapsulates logical control, motion control and process control. Generic functionality is exposed and accessed through the object interface. This gives vendors of control software the freedom to implement the software

in their own fashion but solutions need to be complied with OSACA interface standard for interoperability.

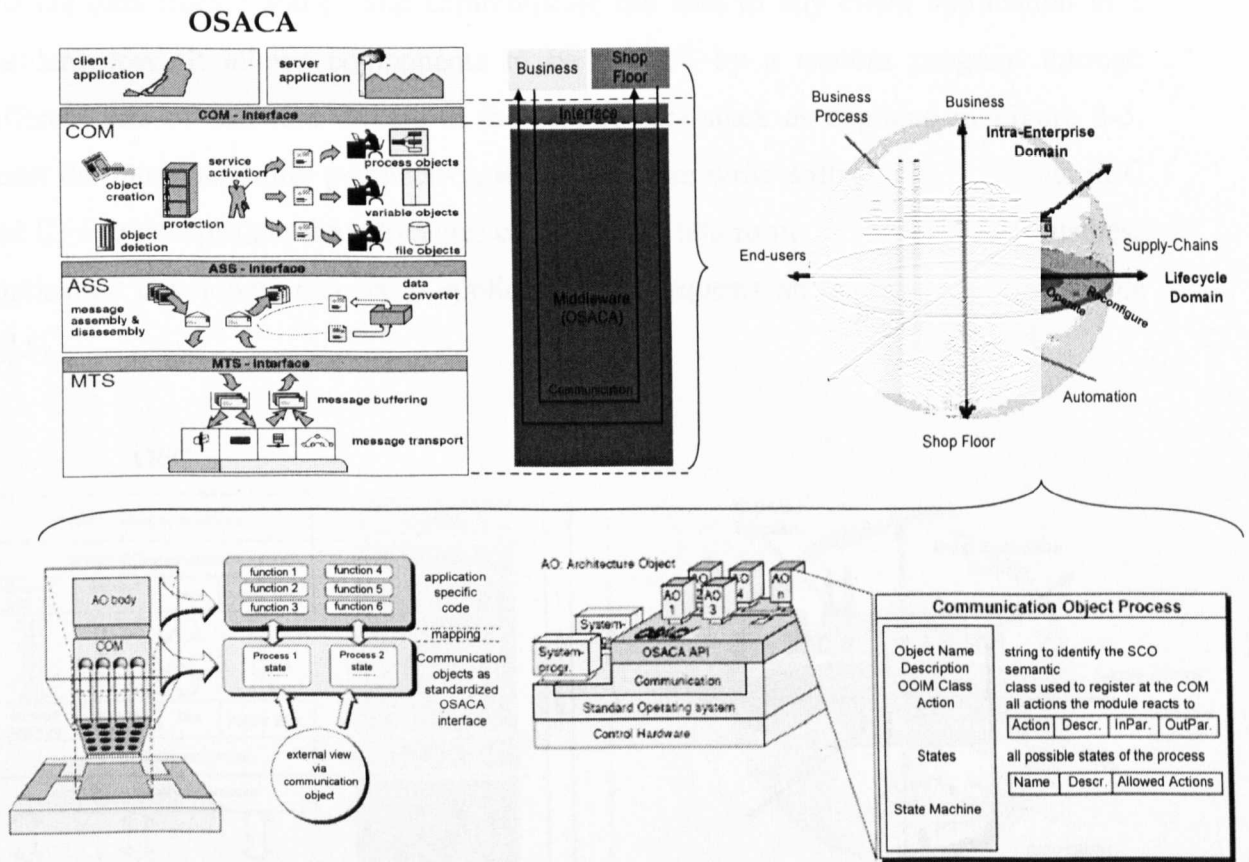


Figure 3-4: OSACA Framework

From the business and the shop floor integration perspectives, the OSACA equivalent middleware is implemented to integrate control level applications (server applications) to the business systems (client applications). The client finds and invokes the services through Common Object Model (COM), ASS (Application Services System) and Microsoft Transaction Server (MTS) middleware layers [70]. The physical communication can be made through LAN or gateway approaches if different networks and protocols are used on the shop floor.

3.4.2 OPC Foundation (Object Linking and Embedding (OLE) for Process Control): The OPC foundation is a non-profit corporation and has established a set of standard Microsoft OLE/COM interface protocols intended to foster greater interoperability between automation applications, field systems and devices and business applications in

the process control industry. OPC technology defines standard objects, methods and properties for servers of real-time information in distributed process environments, programmable logic controllers and smart field devices. OLE provides a mechanism to provide data from a source and communicate the data to any client application in a standard way. It allows components to be utilized by a custom program through different sets of software drivers in the middleware stack as depicted in Figure 3-5. From the interoperability perspective, developers can write software components in C and C++ to encapsulate the intricacies of accessing data from a device, so that business application developers can write applications to requests and utilise shop floor data [g15].

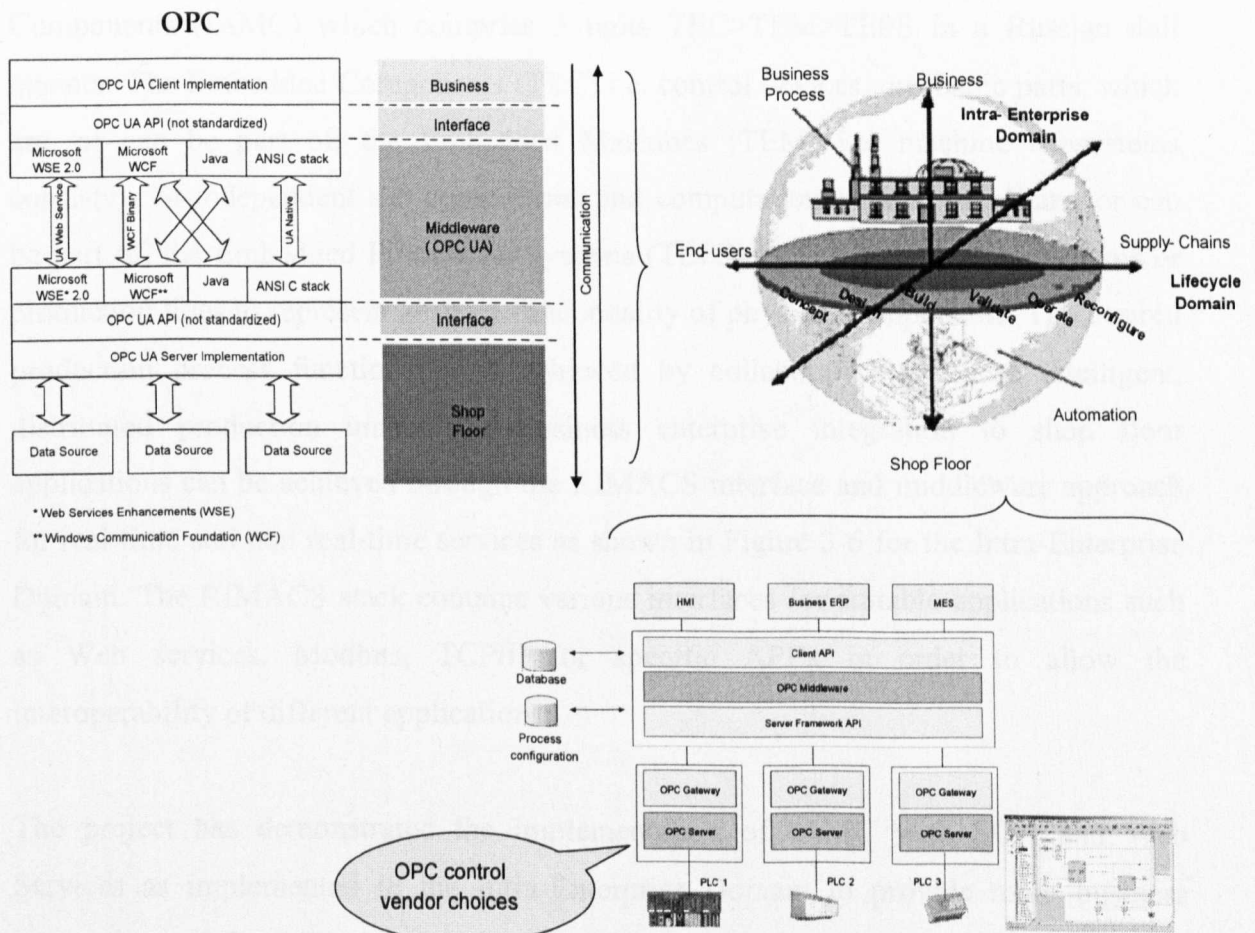


Figure 3-5: OPC Framework

However, in the design of automation systems from the lifecycle perspective, there is no well-structured approach from OPC to support flexibility and re-configurability. These capabilities depend on the capabilities of vendors to implement this functionality within the OPC framework.

3.4.3 RIMACS (Radically Innovative Mechatronics and Advanced Control Systems): RIMACS is a European FP6 project initiated in 2005. The consortium has proposed a collaborative automation paradigm based on an autonomous and modular component-based approach for flexible and agile manufacturing systems to enable mass customization with reduced lead-time. The RIMACS approach considers the set of production entities as a conglomerate of distributed, autonomous, intelligent and reusable units, which operate as a set of collaborating entities at production runtime. Each entity is typically constituted from hardware - mechatronics, control software and embedded intelligence and provides common communication capabilities [91].

The production entities are referred to as Intelligent Autonomous Mechatronic Components (IAMC) which comprise 3 units TEC>TEM>TEPS in a Russian doll manner. The Embedded Components (TEC) i.e. control devices, mechanic parts, which are, or can be part of, the Embedded Machines (TEM) i.e. machine subsystems consisting of independent sub-components and computational units, which are, or can be part of, the Embedded Production Systems (TEPS) i.e. overall machine systems or production lines to represent abstract functionality of physical components. The desired production process functionality is achieved by collaboration of these intelligent, distributed production units. The business enterprise integration to shop floor applications can be achieved through the RIMACS interface and middleware approach for real-time and non real-time services as shown in Figure 3-6 for the Intra-Enterprise Domain. The RIMACS stack contains various interfaces for suitable applications such as Web services, Modbus, TCP/IP or specific API's in order to allow the interoperability of different applications.

The project has demonstrated the implementation of IAMC with SOA and Web Services as implemented in the Intra-Enterprise Domain to provide more business integration and flexibility in control systems.

The modularisation of the production system requires the decomposition of the present “controller-oriented structure” into functional modules with a “manufacturing-task-oriented structure” These building blocks of the collaborative automation system, which are built upon the SOA will present their functionalities and production operations as Web Services in the building block network and form the desired production process by

collaboration using the communication methods provided by Web technology [91]. An additional module (Factory Cast HMI) is required to allow Web Services capability on PLC controllers as shown in Figure 3-6 Lifecycle Domain.

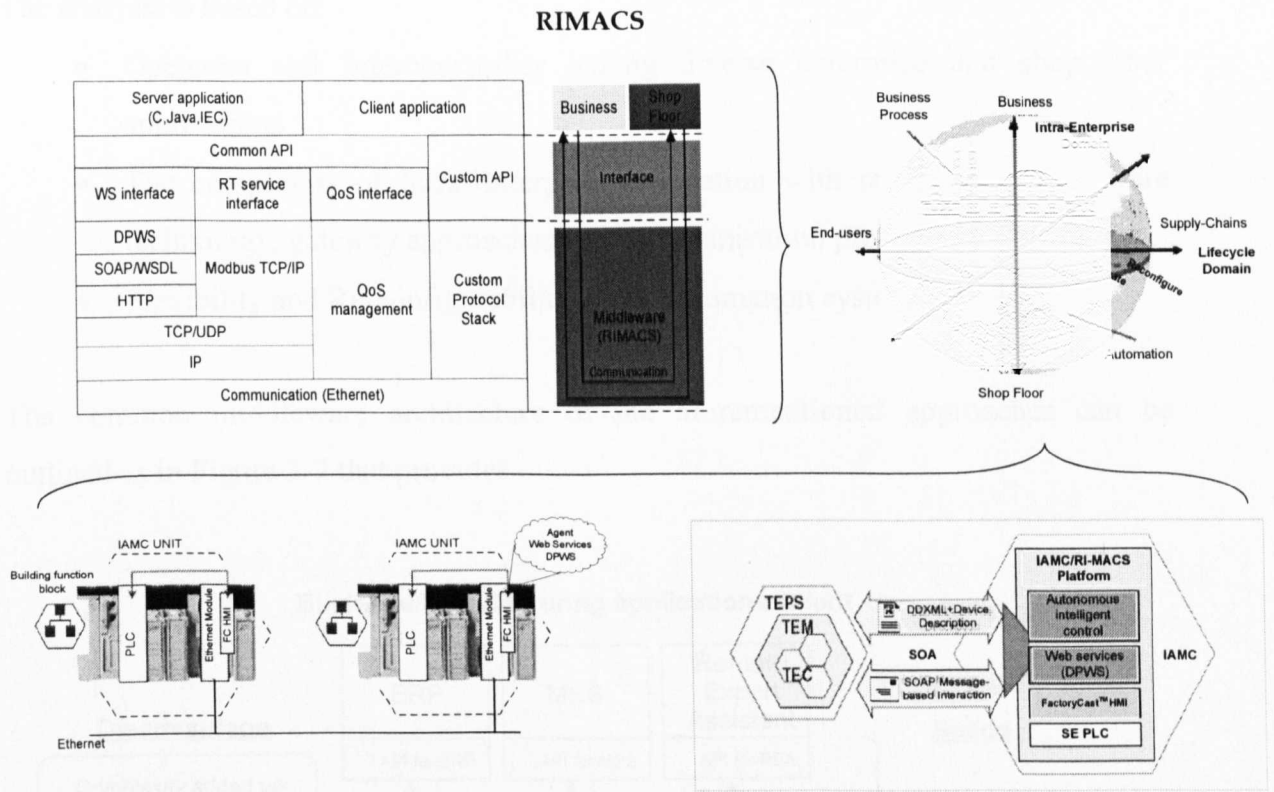


Figure 3-6: RIMACS Framework

3.4.4 Key Characteristics Summary

The summary of the system characteristic of OSACA, OPC and RIMACS frameworks in terms of system design technology and conceptual approaches is shown in Table 3-2 below.

Table 3-2: OPC/OSACA/RIMACS Key Characteristic Summaries

Characteristic	OPC	OSACA	RIMACS
Control system objective	Interoperability	Interoperability	Interoperability
Control system design	Vendor specific solution	Object-oriented design	Service-oriented design
Middleware	OPC	OSACA	RIMACS-SOA
Communication Approach	Request/Response	Request/Response Publish/Subscribe	Request/Response Publish/Subscribe
Support Network	LAN/Fieldbus	LAN/Fieldbus	LAN/Modbus TCP/IP
RTOS	√	√	√
Business- Shop Floor connection	OPC gateway	SERCOS gateway	PLC gateway

3.5 Existing Approaches Limitation Analysis

The objective of this section is to illustrate the limitations of these existing approaches on the requirements of agile manufacturing as determined previously (see Figure 3-2).

The analysis is based on:

- Openness and Interoperability among diverse enterprise and shop floor applications
- The complexity of Intra-Enterprise integration with regards to middleware technology, gateway approaches and communication protocols
- Flexibility and Re-configurability of the automation system

The common middleware architecture of the aforementioned approaches can be outlined as in Figure 3-7 that provides

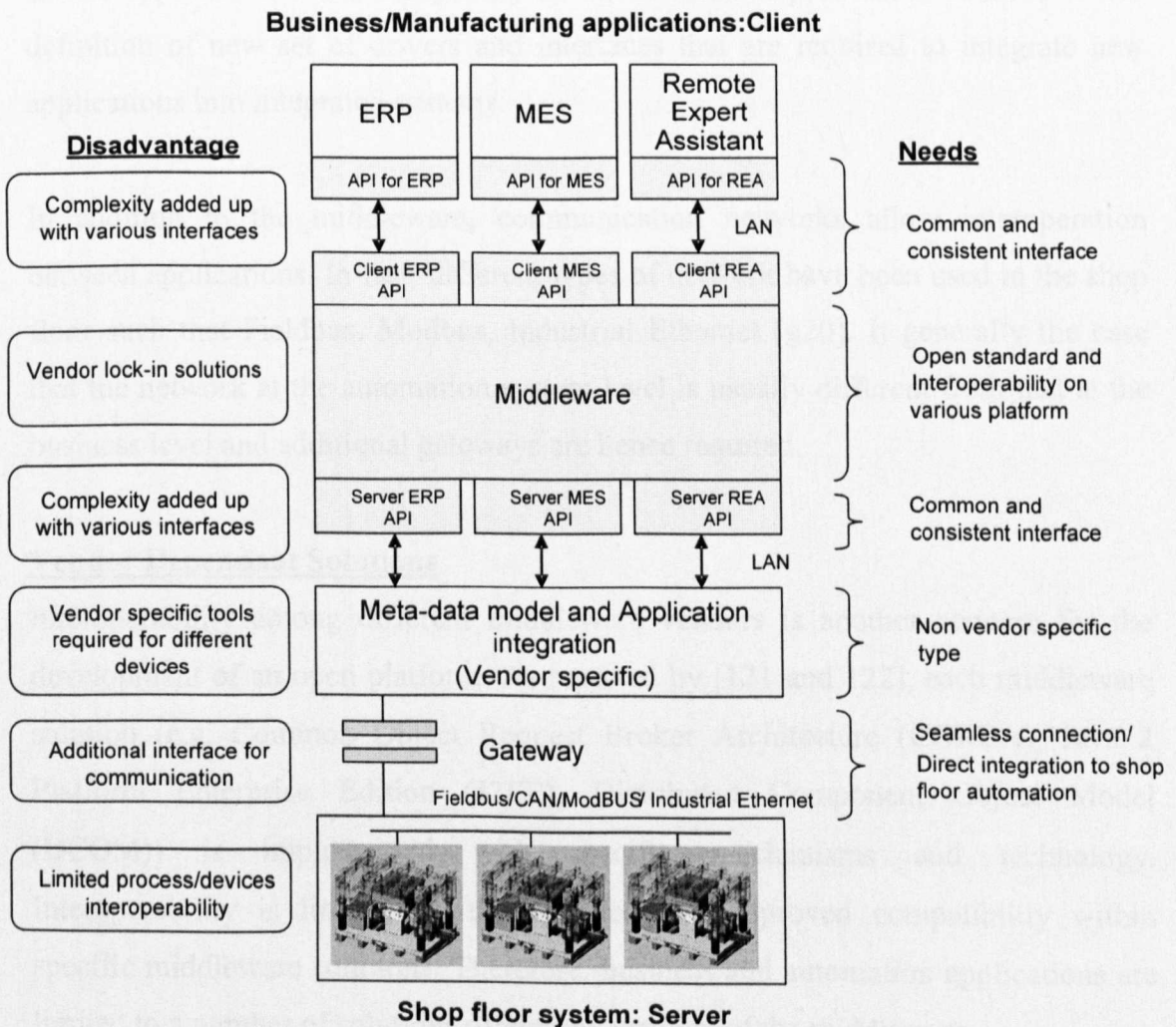


Figure 3-7: Common Middleware Architecture

a simplified version of the process of integrating distributed software applications using middleware. In this case, the middleware enables the connection between business applications (so called “client”) and shop floor applications (so called “server”) through interfaces and gateway proxies (if required) via communication mediums such that LAN and Fieldbus network.

Degree of Complexity

Based on current middleware solutions, diverse software applications on different operating systems are integrated via middleware through specific API interfaces that are compatible with source applications. The output from the middleware is another set of interfaces that translate the message into desired formats in the meta-data models where shop floor applications and business applications interoperate.

In this approach that the complexity of the integration problem is reduced to the definition of new set of drivers and interfaces that are required to integrate new applications into integrated systems.

In addition to the middleware, communication networks allow interoperation between applications. In fact, different types of network have been used in the shop floor such that Fieldbus, Modbus, Industrial Ethernet [g20]. It generally the case that the network at the automation system level is usually different from that at the business level and additional gateways are hence required.

Vendor Dependant Solutions

Interoperability among different middleware vendors is another concern for the development of an open platform. As reported by [121 and 122], each middleware solution (e.g. Common Object Request Broker Architecture (CORBA), Java 2 Platform Enterprise Edition (J2EE), Distributed Component Object Model (DCOM)) is implemented with specific mechanisms and technology. Interoperability is limited by the certified and approved compatibility within specific middleware solutions. Therefore, business and automation applications are limited to a number of solutions offered by vendors of the middleware.

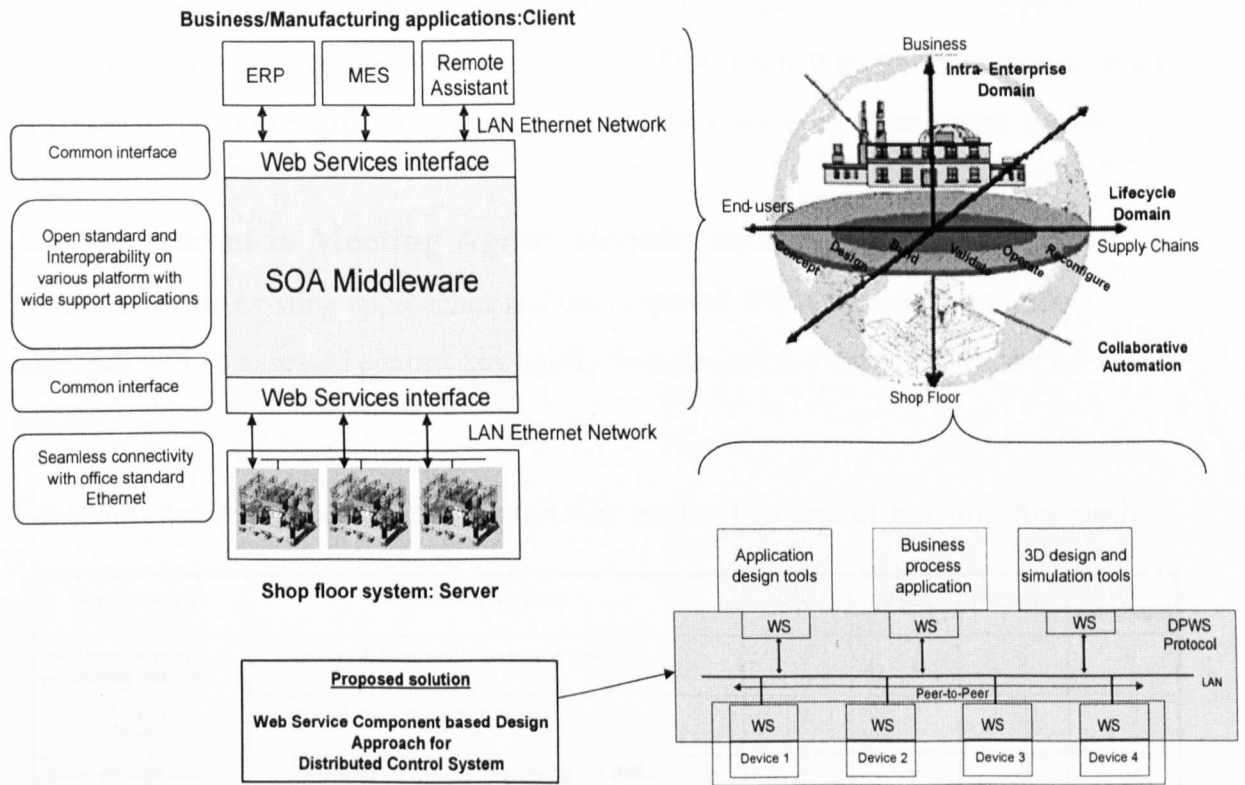


Figure 3-8: Web Services- based Automation Paradigm

As proposed on the SOCRADES project [86], to realise greater shop floor integration capability, the automation level needs to inherit SOA-Web Services characteristics to support higher level application integration. In addition, the flexibility and agility will be enhanced by the implementation of Web Services on distributed control devices to achieve loose coupling in the automation system. Other local run time and design time applications can be seamlessly integrated through a single set of standard Web Services interfaces. Furthermore these applications can be readily connected to the control system and devices via standard office Ethernet LAN's.

In terms of the reconfiguration of the control system to support the manufacturing system lifecycle, this research has proposed a novel framework of Web Services and Component-based design approaches for manufacturing systems integrated with process engineering tools supporting a “design for reuse” approach. Each mechanical part has a defined Web Services functionality as an abstract layer.

The low-level programming is encapsulated and exposed to control engineers and machine builders through the Web Services interface. Each Web Services component is managed by process engineering tools in the control system commissioning phases.

3.7 Assessment in Meeting Agility Requirements

In summary, the existing approaches and the proposed WS Automation solution approach will be assessed against key agility features and the expected impact on key enablers.

Table 3-3: Assessing the Achievement in Key Agility Features of Existing Approaches

Agility feature	Key enabler evaluation	OPC	OSACA	RIMACS	WS Automation
Modularisation of production system ①	Open control platform (Non vendor specific)	√	√	√	√
	Flexible/Reconfigurable automation	?	√	√	√
Quick change over automation system ②	Well supported process engineering tool (HMI, Simulation, Control Application Builder)	?	?	?	√
	Component -based design	?	√	√	√
Integrated business production IT ③	Seamless network connection	?	?	√	√
	Well Integrated information sharing	?	?	√	√
Information retrieval and utilisation ④	HMI system	√	√	√	√
	Common Database and Diagnostic system	?	?	?	√

The key features of agile manufacturing discussed in section 3.3, have been included in Table 3-3 in order to allow comparison each of the solutions. Evaluation of the approaches has been based on information and data reported in research publications. However, where there is uncertainty where features cannot be justified conclusively due to subjective opinions expressed in the articles they have been marked with a question mark.

As indicated in Table 3-3, there are areas where existing solutions do not deliver agility in the required areas. Most of the existing platforms are limited to open solutions from specific providers. The middleware is generally composed of different sets of drivers for each different application. Engineering tools, HMI's, and other integrated applications need to be designed by compilation of specific sets of additional

middleware interfaces that considerably increases the complexity and decreases the modularity of the system.

In comparison, the proposed WS automation system supports every aspect of agile system requirements as detailed in Table 3.3. The WS's are considered as a common technology in terms of integration and building of control systems to allow the creation of an overall homogenous platform. From the re-configurability perspective, the WS's can be complemented with: (i) a component-based design approach, (ii) the integration of process engineering and simulation tools and (iii) HMI systems to support increased agility in the manufacturing system.

3.8 Conclusion

In the first three chapters of this thesis the *need for a paradigm shift from mass production systems to mass customisation capabilities* due to characteristic changes from customers and increase in the global competition has been outlined. Following a review of traditional manufacturing systems, the reasons why this manufacturing approach is not suitable for today's mass customisation requirements where unpredictable changes in the customer demands and product specifications are the norm has been detailed. As a result, companies and manufacturers need to seek a new paradigm for their business strategies. Agile manufacturing, proposed as the next generation manufacturing paradigm and a possible solution to the traditional manufacturing limitations has been highlighted and its key enabling concepts presented. In addition, the *needs of industrialists in the automation domain* that are required to support agile manufacturing have been identified. The key details are summarized in Table 3-4.

The state of the art for agile manufacturing and enabling technologies has been reviewed from a study of selected projects that have reported a substantial impact on the research area. Each approach has achieved a measure of agility based upon its own pre-selected criteria. However each approach has limitations that have prevented full realisation of the agile manufacturing paradigm. The common problem is the level of interoperability between vendor specific technologies that contributes to the increased degree of complexity in integrating different applications in both business and manufacturing systems. In order to overcome this issue, the research in this thesis has

proposed the radically new design of the control and manufacturing platform using standard technology of a SOA and WS's as a neutral platform for business and manufacturing process integration. The summary details are presented in Table 3-5.

Previous research has indicated that a SOA and WS's can be readily utilized for business level application integration. However, the role of WS and a service oriented integration strategy in the manufacturing system, particularly in the real-time distributed control automation systems, has not been defined as yet. One major contribution of this research work is to outline the WS integration methodology for business and control automation that enables the achievement of agility in the complete manufacturing system and supply chain.

Table 3-4: Addressed Problems, Requirements and Key Agile Enablers Summary

Discussed in Chapter 2		Discussed in Chapter 3
Existing problems	End-user requirements	Key agile enablers
<u>Manufacturing system</u> <ul style="list-style-type: none"> • Rigid Hierarchical Structure 	<ul style="list-style-type: none"> • Seamless business- manufacturing process integration 	Refer to Table 3-3 ③
<u>Design of Automation Systems</u> <ul style="list-style-type: none"> • Experience-based design and implementation • Late verification of automation systems 	<ul style="list-style-type: none"> • Rapid design of the automation • Non-vendor specific platforms • Visualisation and Simulation 	Refer to Table 3-3 ①
<u>Control Systems</u> <ul style="list-style-type: none"> • Rigid manufacturing development with centralized control • Lack of reusable and reconfigurable automation peripherals • Diversity and complexity of automation devices 	<ul style="list-style-type: none"> • Quick response to change in the production capacity • A high degree of reuse • A more consistent integration approach for diagnostic and maintenance support 	Refer to Table 3-3 ②
<u>Operation</u> <ul style="list-style-type: none"> • Lack of remote diagnostics and support 	<ul style="list-style-type: none"> • Visualisation and Simulation • Fault tolerance and recovery 	Refer to Table 3-3 ④

Table 3-5: Limitations, Requirements and Proposed SOA-WS Solution Summary

Current Approach Limitation to Agility	The Needs	Proposed SOA-WS Solution
<u>Complexity:</u> <ul style="list-style-type: none"> • Required various sets of interface and gateway for business and shop-floor integration • Specific tools are required for various control systems 	<ul style="list-style-type: none"> • Common and consistent interface • Common network connection 	<ul style="list-style-type: none"> • Common WS interface for devices and business applications integration • Ethernet network connection
<u>Vendor Specific Solution:</u> <ul style="list-style-type: none"> • Vendor lock-in solution as well as non-interoperable middleware 	<ul style="list-style-type: none"> • Open standard, interoperability integration middleware 	<ul style="list-style-type: none"> • Standard SOA middleware
<u>Rigid Structure of Automation System:</u> <ul style="list-style-type: none"> • Tight coupling between devices to devices and devices to business applications 	<ul style="list-style-type: none"> • Describing component functionality as abstract interface for a more loose coupling 	<ul style="list-style-type: none"> • WS interface for device descriptions and control functions

CHAPTER 4

Overview of Current Technology and Key Enablers for Agile Manufacturing System

The features of agility and a novel approach for agile manufacturing based upon WS's were detailed in the last chapter. Using the collaborative automation system model as presented in Chapter 3, the implementation of distributed control system architecture (including PLC devices, the IEC 61131 standard and Network capabilities), distributed control applications (e.g. the IEC61499 standard and LonWorks system), and middleware (e.g. CORBA and SOA) are reviewed in this chapter to illustrate the capabilities of the current technology and future trends towards the support of agile automation.

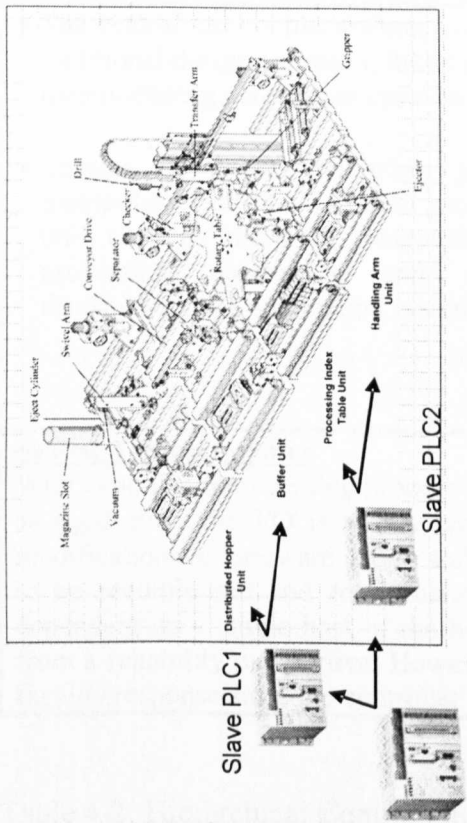
4.1 Problem Statement

There are various implementation technologies for production systems to support the development of "agile solutions". However, it is imperative that suitable and effective technologies are integrated to enable true agile manufacturing systems to be realised. The research questions to be addressed are:

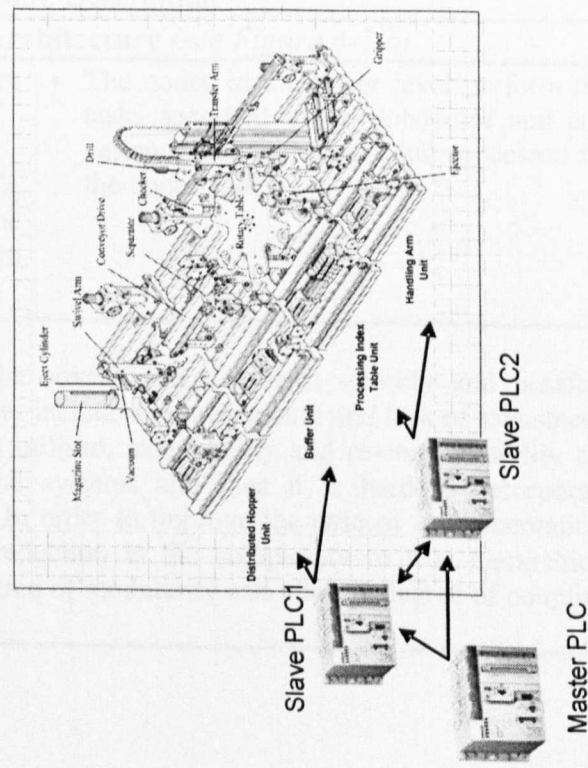
1. What are the architectures associated with the different types of distributed control systems? Which approach is most suitable for the next generation of manufacturing systems?
2. What types of networks are used in the industrial automation for control device communication? What interoperability issues are there with the different types of network?
3. What are architectures and technologies of the different types of middleware, such as CORBA, DCOM and SOA? What are the main differences and advantages/disadvantages between these?
4. Why has conventional middleware not been substantially used in manufacturing automation systems?

4.2 Control System Architectures and Trends

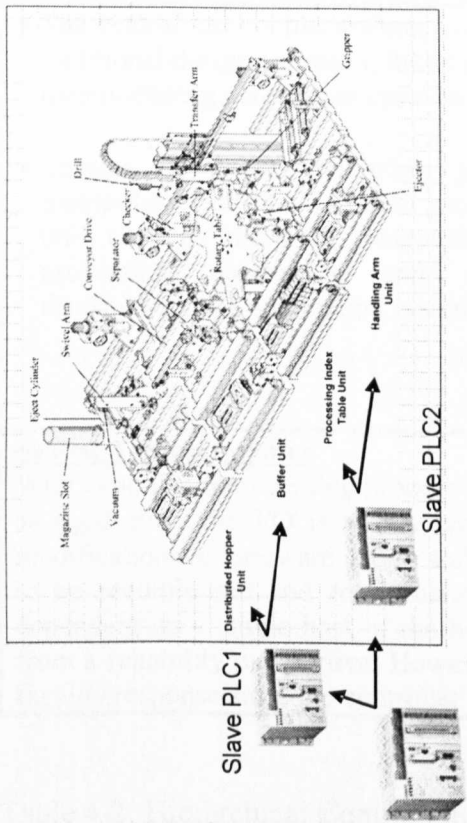
As reported in [31], basic control architectures can be classified into four categories: Centralized, True Hierarchical, Modified Hierarchical, and Heterarchical (see Figure 4-1).



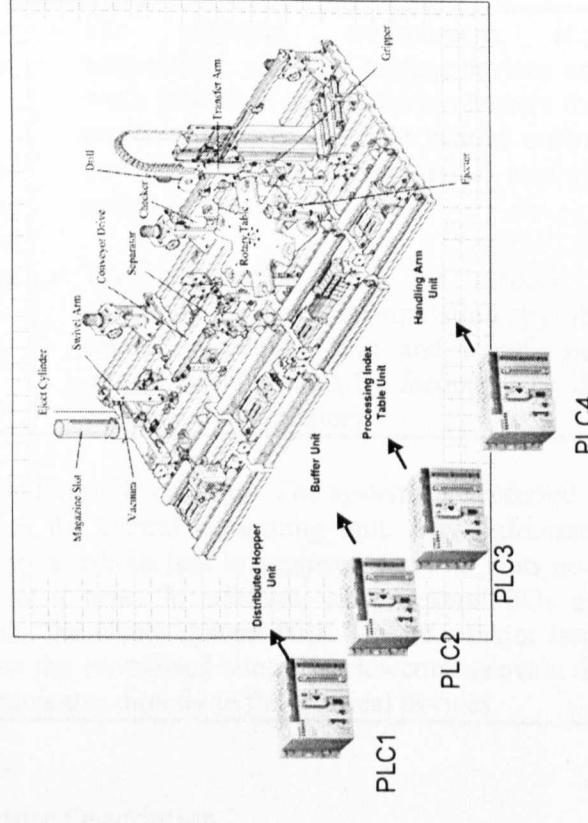
a) Centralised Control Architecture



c) Modified Hierarchical Control Architecture



b) Hierarchical Control Architecture



d) Heterogeneous Control Architecture

Figure 4-1: Basic Control Architectures

The details of these 4 control architectures are summarized in Table 4-1/2/3/4.

Table 4-1: Centralised Control Architecture Description

Centralised Control Architecture (see Figure 4-1/a)	
<ul style="list-style-type: none"> • The centralized control architecture is the traditional design approach, found in older manufacturing automation systems. • Sensors, actuators and other physical manipulators rely on a central processing unit such as the PLC, responsible for processing commands and making decisions for the automation systems. 	<ul style="list-style-type: none"> • The physical environment (e.g. temperature, status of tooling devices and work pieces) is monitored by sensors that are connected to a single central control unit for the entire machine / machine systems or plant. • The information from the sensors is gathered for data manipulation by the central processing unit and transformed into physical actions by feeding back the commands to actuators.
<p><u>System Characteristics</u></p> <p>With centralized processing process reconfiguration is complex. The systems are referred to as rigid since the I/O is tightly coupled to the central computing unit. Any unforeseen modifications to hardware and/or software are a tedious task to engineer, since all units need to be reconfigured and re-evaluated one at a time. In addition, since several I/Os are dependent on a single host, if the host fails, the whole system fails. This is a major issue from a reliability perspective. However, this the centralised control architecture provide the fast I/O response since the controller communicates directly to the physical devices.</p>	

Table 4-2: Hierarchical Control Architecture Description

Hierarchical Control Architecture (see Figure 4-1/b)	
<ul style="list-style-type: none"> • True hierarchical architectures have been widely used in manufacturing systems. • The node in the upper level is responsible for job management and provides the nodes in the lower level strict schedules to follow. 	<ul style="list-style-type: none"> • The nodes in the lower level perform the tasks specified in the schedules and any variance will be sensed and processed by the upper nodes.
<p><u>System Characteristics</u></p> <p>This true hierarchical architecture has the advantages of system stability and possible performance optimization, but disadvantages include a slow response and lack of robustness. Because pre-established fixed structure is utilised, extensibility and re-configurability are difficult to achieve with true hierarchical systems and thus it is hard to incorporate unforeseen changes into the system [32]. In order to improve the ease of implementation, debugging, testing, and maintenance, a reduction in the complexity of true hierarchical architecture is required where a higher degree of modularity and a lower degree of coupling between modules is provided [33].</p>	

Table 4-3: Modified Hierarchical Control Architecture Description

Modified Hierarchical Control Architecture (see Figure 4-1/c)	
<ul style="list-style-type: none"> • The modified hierarchical control architecture was proposed to overcome the problems identified with true hierarchical architectures. • The processing units, such as PLCs, can be distributed to machining stations to provide local control to automation devices. 	<ul style="list-style-type: none"> • To improve flexibility and overall performance, the control units at the same level of hierarchy can communicate with one another in a collaborative manner to react to specific interferences.
<p>System Characteristics</p> <p>However, this system is still restricted, with the main higher computing unit (i.e. the Master controller) providing overall control and sequencing of operations. Hence the modified hierarchical form inherits the disadvantages of proper hierarchical architecture, in terms of rigidity and master/slave relationships.</p>	

Table 4-4: Heterogeneous Control Architecture Description

Heterogeneous Control Architecture (see Figure 4-1/d)	
<ul style="list-style-type: none"> • Heterarchical architectures have gained much attention in the development of the next generation automation control systems. • There is no central controller in this approach that delegates the control autonomy to the local control unit / units. The control units in this structure are autonomous and intelligent enough to execute their own manipulations, since the processing units are encapsulated locally. 	<ul style="list-style-type: none"> • The relationships between control units are both cooperative and competitive: as more intelligent devices are incorporated into lower level devices, sensors and actuators perform local operations, such as data monitoring and local feedback control, without relying on a master controller. • This incorporates the advantages of robustness, flexibility and re-configurability. Failure of individual control units does not impact system performance, as both knowledge and control algorithms are distributed [32].
<p>System Characteristic:</p> <p>The design of automation systems should move towards heterarchical architecture, with highly distributed and loosely coupled devices with low-level peer-to-peer communications in real-time, in order to improve the agility and responsiveness of manufacturing automation systems, as reported in [29]. However, there are drawbacks in adopting heterarchical architectures [32]. The heterarchical control systems are unpredictable in terms of response time and that global system performance optimization is very difficult to achieve. Other authors have argued that heterarchical architectures could make systems more predictable by using publish-subscribe communication.</p>	

4.3 Programmable Logic Controllers

The Programmable Logic Controller (PLC) has been a part of manufacturing controls technology since the 1970's, and has become the most common choice for the control of automation systems. Traditionally, PLC's are mainly programmed by ladder logic that was developed to mimic relay logic i.e. a graphical language representing an electrical wiring diagram used for describing relay control schemes.

The operational principles of PLC's are based on simple 1/0 logic. The words "TRUE" and "FALSE" or "ON" and "OFF" are used to indicate the status of switches connected to relays within the control system. The ladder logic in the PLC is actually a computer program that the user can enter and change. Note that both of the input push buttons are normally open, but the ladder logic inside the PLC normally has one open contact and one closed. The ladder logic in the PLC does not need to match the inputs or outputs (e.g. SW1 with B, SW2 with A, as in Figure 4-2).

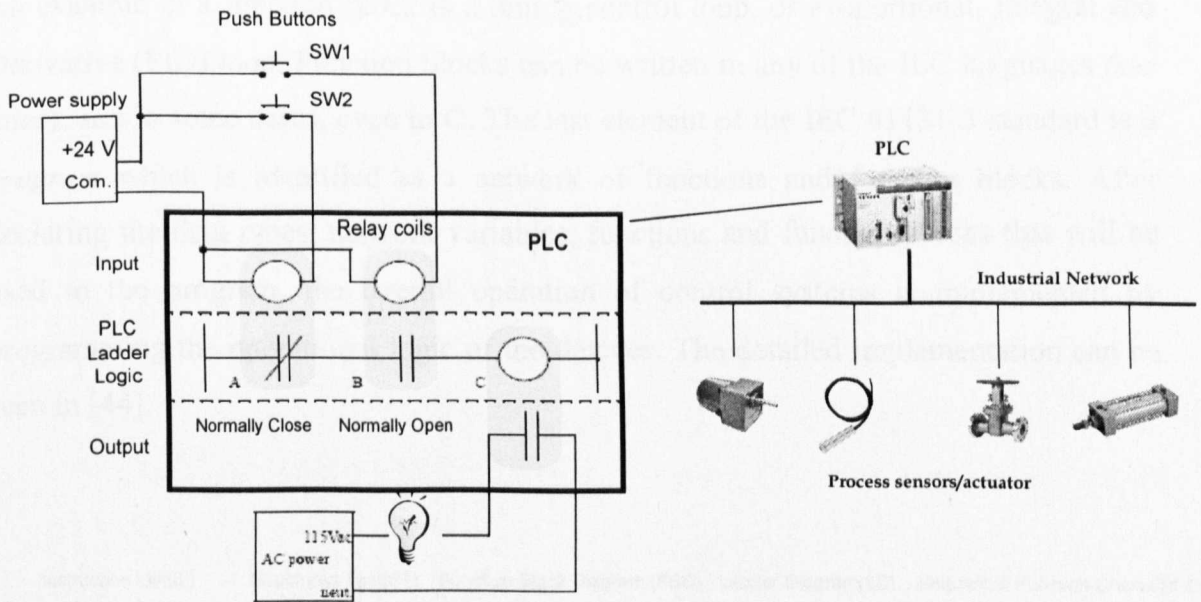


Figure 4-2: A basic PLC illustration

There are drawbacks with employing traditional PLC's [33]. Ladder logic has a lack of modularity that offers limited reusability of programming code and no support for complex programming structures. Implementations require highly experienced programmers for commissioning. To enhance traditional PLC programming techniques, the IEC 61131-3 standards were developed as a common and open framework for PLC software architectures. This was an attempt to encourage an open, interchangeable and structured approach to the development of control software. The standard is defined loosely enough so that each PLC manufacturer is able to maintain their own look-and-feel, but the core data representations are common.

The IEC 61131-3 standard encapsulates programs, function blocks and functions, otherwise referred to as Program Organization Units (POU's). *Standard functions* include instance ADD (addition), ABS (absolute), SIN and COS, and once users define functions, they can be re-used. *Function blocks* are the software equivalent of Integrated Circuits, (IC's), or black-boxes, representing a specialized control function. They can contain data as well as functionality, so they can keep track of past events and states. An example of a function block is a tuning control loop, or Proportional, Integral and Derivative (PID) loop. Function blocks can be written in any of the IEC languages (*see later*), and in some cases, even in C. The last element of the IEC 61131-3 standard is a *program* which is identified as a network of functions and function blocks. After declaring the data types, network variables, functions and function blocks that will be used in the program, the overall operation of control systems is implemented by programming the operational logic of the devices. The detailed implementation can be seen in [44].

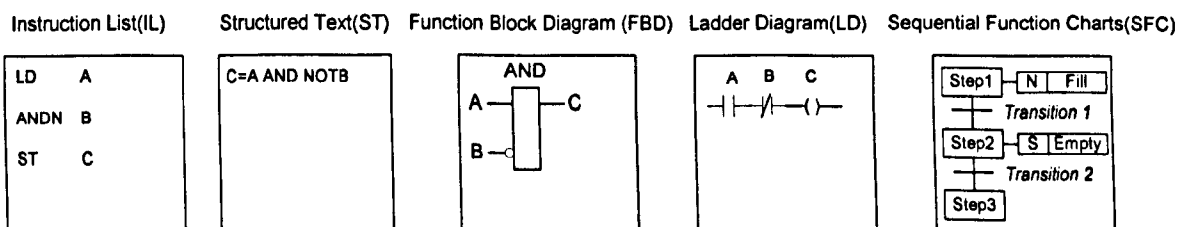


Figure 4-3: Language constructs within the IEC 61131-3 Programming Standard

The IEC 61131-3 can be constructed in any of the five language constructs defined within reusable function blocks [43], as depicted in Figure 4-3

IL (Instruction List) - is effectively mnemonic programming.

ST (Structured Text) - is a very powerful high-level language with its roots in Ada, Pascal and "C". It contains all the essential elements of a modern programming language, including selection branches (IF-THEN-ELSE and CASE OF) and iteration loops (FOR, WHILE and REPEAT), and these elements can also be nested. It is excellent for use in the definition of complex function blocks, which can be used within any of the other languages.

LD (Ladder Diagram) - is based on the graphical presentation of Relay Ladder Logic.

FBD (Function Block Diagram) - A graphical dataflow programming method very common in the process industry. Expresses the behavior of functions, function blocks and programs as a set of interconnected graphical blocks, like those in electronic circuit diagrams. Systems are represented in terms of the flow of signals between processing elements.

SFC (Sequential Function Charts) - A graphical method for structuring concurrent program. As illustrated in Figure 4-3, the flowline connects a step and a transition. The transition is used as a condition to allow control to move to the next step when a condition is met. At some point, there may be a desired action to be performed at each step.

4.4 Communication Networks

The control devices (controllers) communicate to other devices and the master controller through the industrial network, in order to pass and exchange the control data that will be executed on the controller in order to allow the physical I/O devices to function. The fieldbus network system has been used in automation manufacturing for many years, but Ethernet technology is fast becoming the standard for device communication. In this section, the capability, standards, implementation technologies and limitations of each fieldbus system and Ethernet is reviewed.

Although fieldbus technology has been around for many years, it is still not widely used because of a lack of international fieldbus protocol standards, which would ensure complete interchangeability and interoperability between different suppliers. Fieldbus is specified and implemented according to the Open Systems Interconnection (OSI) Reference Model, as discussed in section 4.4.2. However each fieldbus networking solution is not standard i.e. some fieldbus technology is based on a three-layer model: the physical layer, data-link layer and application layer, but other fieldbus technologies are based on a four layer model: the same as above plus a network layer. There are many types of fieldbus systems on the market and these are not directly interchangeable and interoperable.

Standard Ethernet is designed around two layers of the OSI model (i.e. the physical layer and a data-link layer) for compatibility and interoperability between computers. Ethernet is widely used in office and home to connect different types of computers and peripherals together, regardless of operating systems. It is foreseen that Ethernet may be developed for industrial sectors for the purpose of device communication and plant integration, through enterprises as demonstrated by [29], [92], [93], and many companies are keen to adopt Ethernet in industrial sectors. In the following section, fieldbuses will be compared with the emerging Ethernet technology for automation systems.

4.4.1 Open Industrial Fieldbus Systems

Fieldbus is a communication standard that enables communications between field devices and a master device such as a PLC. Currently, there are several types of fieldbus standards available for communication between control devices. Traditionally,

fieldbus provided users with proprietary solutions from specific vendors. However, since the introduction of open, non-proprietary protocol standards, many fieldbus providers have consistently developed technologies to offer manufacturing sectors, with the new solution for open industrial bus systems. Details of fieldbus can be found in [37, 38], and a summary of common fieldbuses is shown in Table 4-5.

Table 4-5: Summary of Common Fieldbus Type Networks

Name	Description	Characteristics	Company
Actuator sensor interface (AS-i)	<ul style="list-style-type: none"> • A master/slave open network. It is simply designed with a two-wire untwisted, unshielded cable, which is used for both communication and power supply. 	<ul style="list-style-type: none"> • A master can take up to 31 slaves • Node addresses are assigned either by the master or addressing units via bus connection • Capable of connecting to serial RS connections, and Profibus via proper interfaces. 	Allen-Brandley
Control Area Network (CAN)	<ul style="list-style-type: none"> • A type of serial bus data system for multiple devices, passing information between each other with high-speed data rates. 	<ul style="list-style-type: none"> • A “non- destructive bit-wise arbitration” to access the bus, by using the bus station identifier to allow higher priority to gain access first. 	Bosch
DeviceNet	<ul style="list-style-type: none"> • The design based on CANs to interconnect lower level devices with higher level controllers. • The cabling system consists of 4 conductor cables providing power and data communication. 	<ul style="list-style-type: none"> • Master/slave communication either by strobe or poll methods. It can support up to 64 nodes, and as many as 2048 devices. 	Allen-Brandley

Interbus-S	<ul style="list-style-type: none"> • An open device level network, allowing a twisted pair of fiber optic cables connected to each station. • Communication buses are local (TTL voltage) and remote (RS-485 voltage). 	<ul style="list-style-type: none"> • Master/slave register shifting procedure • Addressing I/O stations automatically during start up. 	Interbus-S Club
Profibus	<ul style="list-style-type: none"> • Is designed for communication between PLC and distributed low level devices on the bus system. • There are three types of Profibus: FMS (Field messaging specification), DP (Distributed processing), PA(Process Automation). • RS-485 voltage standard for FMS, DP and IEC 1158-2 for PA. 	<ul style="list-style-type: none"> • Master/slave with Logical Token Ring, to allow each master controller to communicate with slave devices. • Real devices (Communication objects) are defined at the local object dictionary (source OD) during programming phase. 	DIN 19245 standard
DeviceNet	<ul style="list-style-type: none"> • The design based on CANs to interconnect lower level devices with higher level controllers. • The cabling system consists of 4 conductor cables providing power and data communication. 	<ul style="list-style-type: none"> • Master/slave communication either by strobe or poll methods. It can support up to 64 nodes, and as many as 2048 devices. 	Allen-Brandley

4.4.2 Industrial Ethernet Networks

There is a strong interest in developing internet technology for integrating automation devices, due to the development of the distributed collaborative manufacturing systems. The concept of Ethernet-based process control has been introduced in recent years to propagate an open system communication standard and to enable the interchangeable use of equipment from various manufacturers. Some work has been done on developing

“Ethernet for Open Network Communication based on the IEEE 1451 standard” as proposed in [39, 81 and g3].

Industrial Ethernet applies the Ethernet standards, developed for data communication, to manufacturing control networks using IEEE standards-based equipment with the intention that organisations can migrate factory operations from fieldbus systems to an Ethernet environment [g20]. Industrial Ethernet technology tailored for control systems can be represented by the seven layer OSI model in Figure 4-4.

Layer 1	Physical layer	Electrical and mechanical definition of the system. It is concerned with transmitted raw data bits over a communication channel.	Ethernet Physical
Layer 2	Data link layer	This is used to ensure reliable communications through the physical layer. Framing and error correction format of data. This layer determines the structure of data and frame/packet size.	Ethernet MAC
Layer 3	Network layer	Optimum routing of message from one network to another; controls the operation of the subnet.	IP
Layer 4	Transport layer	Managing the flow of the message. It accepts data from the session layer and passes it to the network layer.	UDP, TCP
Layer 5	Session layer	Organization and synchronization of the data exchange. This layer is a user's interface to the network that user needs to negotiate in order to establish a process connection on another machine.	FTP, HTTP, SMTP, SNMP, Telnet
Layer 6	Presentation layer	The layer performs data conversion from one to another format for users, rather than leaving the user to find the solution.	
Layer 7	Application layer	This layer is for file transfer, message exchange and network management.	

Figure 4-4: Layers of the OSI Model for Industrial Ethernet

The OSI model may be interpreted as a collection of entities situated at each of the seven layers. A data (packet) starts at an upper layer, and passes down through each of the layers. As the packet moves down, it is enclosed in a “protocol envelope”, which carries addressing and control information that advises the next layer down what to do with the packet [37], as shown in Figure 4-5.

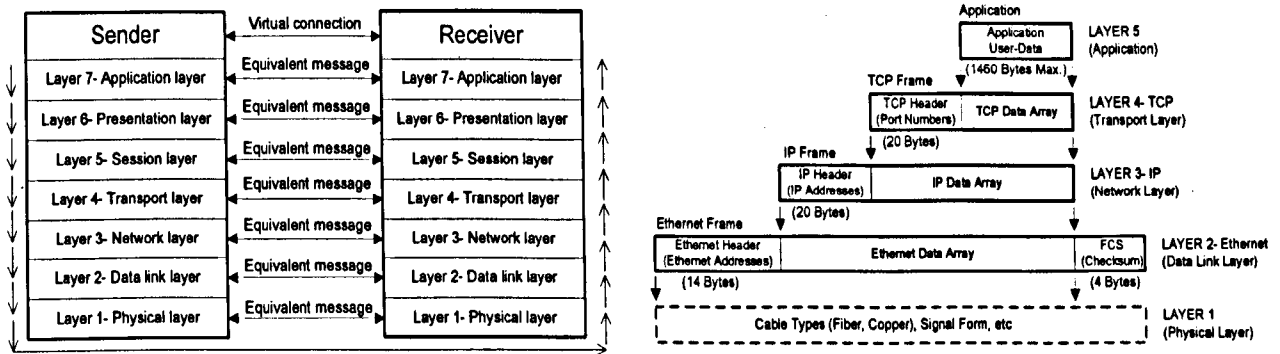


Figure 4-5: OSI Messaging

Currently, there are a number of automation companies offering Industrial Ethernet Network solutions and their control devices. The various adoptions of these Industrial Ethernet Networks are summarized in Table 4-6 below.

Table 4-6: Industrial Ethernet Network Features

Name	Protocol/Network	Characteristics	Standards
Modbus-TCP	Modbus RTU protocol, with a TCP Ethernet interface. Modbus protocol defines the rules for interpreting the data and message structure at and above session layer (Modbus specific).	Provided with Ethernet 10/100 Mbit/s, up to 1Gbit/s. Multi master-slave architecture for distributed automation environment.	Modbus-RTU with standard IEEE 802.3 Ethernet
Ethernet Powerlink	TCP/IP; UDP/IP Protocol with modification in the data link layer for the deterministic of network with collision avoidance.	Ethernet 100 Mbit/s, TCP/IP, a deterministic real-time protocol for standard Ethernet with a RTOS mixed Polling- and Time-slicing mechanism.	Ethernet POWERLINK Standardization Group based on standard IEEE 802.3

EtherCAT	Using full-duplex Ethernet physical layers, with the approach of one frame per node per cycle and processing on-the-fly, in order to optimise the network bandwidth and processing speed delay. The EtherCAT protocol follows IEEE 802.3 standards and can be inserted into UDP/IP datagram.	An open high-performance Ethernet-based fieldbus system (Ethernet 100 Mbit/s), with master/slave communication architecture between a controller and field devices.	IEC 61158, IEC/PAS 62407, IEC 61784-3, ISO 15745-4
EtherNet/IP (CIP)	TCP/IP; UDP/IP Protocol with modification of OSI model at session layer and above, in order to accommodate time-critical control data and message prioritization in multiple communication hierarchies.	Speed from 100 Mbit/s up to 1 Gbit/s. It may be configured to operate in both a master/slave and distributed control architecture, using peer-to-peer communication.	ODVA Ethernet/IP based on IEC 61158 standard
PROFINET I/O	TCP/UDP and IP for non-timing critical data exchange, and a prioritized real-time channel (IEEE 802.1Q) on top of standard Ethernet- data link layer for timing critical applications.	The protocol complies with the standard office Ethernet network.	PI International based on IEC 61158 and IEC 61784

As reported in [39], implementation of Industrial Ethernet can be subdivided into three concepts:

1. Encapsulation Technologies:

This is the extension of the application layer of existing fieldbus networks into TCP/IP networks, by adding the data portions of Ethernet TCP/IP. The original fieldbus networks are preserved.

(Developer: Ethernet/IP, Foundation Fieldbus HSE, and Modbus-TCP/IP)

2. Gateway and Proxy:

The standard fieldbus networks are integrated into the Industrial Ethernet networks through a hardware called gateway or proxy. This device is used as a translator, and interprets the control message between fieldbus and Ethernet networks.

(Developer: ProfiNet and Interbus)

3. Interface for Distributed Automation (IDA):

This concept uses real-time middleware services provided by the Real-Time Innovation Company, in order to accommodate communications between control applications over Ethernet using real-time publish/subscribe (RTPS) architectures that is built on top of User Datagram Protocols (UDP's), Internet Protocols (IP's) and Ethernet.

(Developer: IDA Group - not based upon an existing fieldbuses, unlike former two concepts)

The full details of these three protocol implementations could be found in [33, 39 - 42].

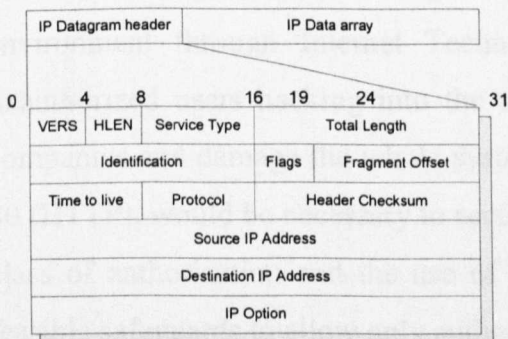
Although all these protocols use Ethernet, this does not mean they can automatically communicate and interoperability is not guaranteed. The application layer (layer 7) translates the incoming information into something the user can understand. Each company does not have the same application layer since they typically develop their own proprietary protocols at this level that does not interact with others [39]. In addition, other layers may be also be modified and adjusted to suit their own design specifications, such as real-time performance (*see Table 4-6*).

In recent years, there have been efforts in developing a new approach to Industrial Ethernet in order to overcome the problems associated with the various designs of protocols. The leading concept has been promoted by the IDA Group with support from Schneider Automation and Jetter. This concept omits layer 7 (the application layer), as this is where most vendors have developed their own designs and device compatibility. New automation concepts are employed, with embedded communication features compatible with the normal Ethernet. There is the strong potential for Ethernet to be connected to those low-level intelligent devices that are able to perform through Ethernet connectivity, in order to ease the commission of automation systems and to enable the plant agility. Ethernet-based process control should propagate the open system communication standard to enable equipment from various manufacturers to be used interchangeably.

4.4.3 Ethernet Standard

Ethernet is a family of frame-based computer networking technologies for local area networks (LANs) and the name comes from the physical concept of the ether. It defines a number of wiring and signaling standards for the physical layer, through the means of network access at the Media Access Control (MAC) /Data Link Layer (DLL), and a common addressing format (<http://en.wikipedia.org/wiki/Ethernet>).

Historically, the network is referred to as a Carrier Sense Multiple Access / Collision Detect (CSMA/CD) bus network type (see “Issues with Ethernet” section below), and it is generally implemented as a 10 Mbps baseband coaxial network or twisted pair cable (Cat 5). Recently, the speed of 100 Mbps has been introduced, and soon speeds of 1 Gbps will be commonly used for home and business purposes. TCP/IP is a protocol that fits into the data frame area of the Ethernet frame, and the protocol regarding the defining of the packet delivery system as “an unreliable (no guaranteed delivery), best effort, connectionless packet delivery” [75]. The basic packet called an IP datagram is shown in Figure 4-6.



VERS	A version of the protocol
HLEN	The datagram header length in 32 bit words
Service Type	This is merely a recommendation to the routing software on the service required
Total Length	Length of the datagram in bytes (including the header section)
Identification	Each datagram must have a unique number
Fragment Offset	This specifies the offset of the data in the original datagram
Time to Live (TTL)	As the datagram passes through the network, its time is decremented for each pass of each gateway or host
Protocol	This specifies the protocol format for the data payload area
Header Checksum	Complement the result of adding the IP header as a series of 16 bit integers using one's complement arithmetic
Source IP and Destination IP Addresses	The IP addresses of source and destination nodes
IP Option	Option used for control purposes

IP Header in bytes = $(32 \text{ bits} \times 5) / 8 = 160 \text{ bits}$ (not included IP Option) / 8 = 20 bytes

(Note: Packet size = Payload + TCP Header + IP Header + Ethernet Header)

Figure 4-6: The Structure of an IP Datagram (from [75])

However, there are a number of practicalities that need to be considered before migrating from the traditional fieldbus industrial network or Industrial Ethernet to Ethernet TCP/IP communication. There are some fundamental problems with applying Ethernet to industrial applications [39], as follows:

1). Non Deterministic and Uncertain Real-time Performance

The fundamental access algorithm, CSMA/CD, cannot provide sufficiently consistent latency for deterministic applications. Its mechanism is to see if the physical layer is idle, then to begin transmitting, or back off and wait a random period of time before retrying until the network is free to accept data packets [g7]. Delays are inevitable. However, as the Ethernet speed becomes faster, up from 10 Mbit/sec - 100 Mbit/sec - 1Gbit/sec, non-deterministic behaviour become less of an issue since the bandwidth is large enough to render delays irrelevant for all but the hardest real time applications [g7]. In some industrial domains, where real-time performance is not crucial and critical to safety, Ethernet could be utilized currently as a supporting technology for open and flexible automation systems.

2). Delivery Speed Degrades with Loading Increases

As the Ethernet speed is significantly increased, this fundamental speed degradation as loading increases becomes of less concern to system commissioners. In addition, there is an extension to the standard, IEEE 802.1p, addressing this problem. This allows a system designer to guarantee the fast delivery of critical data, by means of giving priority to messages as in the real-time system [g8].

3). Security

This is the major issue of integrating industrial automation devices to the outside environment through Internet Technology. The network could be targeted from unauthorized users hacking into the system, and viruses could spread out to linked companies and damage the whole system. Other protection, in addition to firewall port 80 (HTTP), would be necessary to secure links. End-to-End data encryption, a level and class of authorization and the use of private servers to route the traffic are some of feasible safeguards to allow only authorized users to log into the systems.

4). Durability

Using Ethernet creates the large overhead for I/O operations. Usually, most industrial applications use small periodic data transfer, whereas Ethernet deals with large aperiodic messages. Ethernet cables need to be able to work consistently under the harsh industrial environment with high temperature and noise, unlike using in the offices.

4.5 Communication Architectures

In this section, the semantics behind the communication amongst intelligent control units (nodes) are reviewed. Autonomous control devices could be distributed across entire production machines and would be required to interact with one another through the industrial network. The software applications need to exchange information. Complex distributed applications require a more powerful communication model and several types of software technologies (i.e. middleware - *see section 4.7*) have emerged to meet this need [34]. The middleware communication architecture may comprise three categories: Point-to-Point, Client-Server, and Publish-Subscribe.

4.5.1 Point-to-Point

A point-to-point connection is a dedicated *one-to-one* communication system that links two systems or processes. The connection between two nodes consists of two packets exclusively using the connection to communicate. On shared networks, all nodes listen to signals on the cable from broadcasting nodes. However, when one node addresses frames to another node and only that node receives the frames, essentially the two nodes are engaged in point-to-point communications across the shared medium. This is a simple and straightforward approach that gives high-bandwidth but does not scale very well with many nodes.

4.5.2 Client-Server

Client-Server networks include servers (i.e. machines that store data) and clients (i.e. machines that request data). The Client-server is fundamentally a *many-to-one* design (i.e. one central server node and many client nodes) and this type of server works well with centralized information systems such as databases, transaction processing systems, and central file servers. However, if multiple nodes are also generating information, client-server architectures require that all information be sent to the server for subsequent redistribution to the clients, and such indirect client-to-client communication is inefficient, particularly in a real-time environment. The central server also adds an unknown delay to the system, as the receiving client does not know when or if it has a message waiting [34]. In addition, the server can become a bottleneck and presents a single point of failure. Multiple-server nets are possible, but they are very cumbersome to set up, synchronize, manage, and reconnect when failures occur. The

multiple-server resolves bottleneck and point-of-failure exposures, but unfortunately increases inefficiencies and bandwidth consumption [35].

Client-Server architectures are often based on “object-centric” design. However, in distributed real-time applications, the information that needs to be communicated is quite often just data, rather than objects. Attempting to implement these “data-centric” systems with a client-server communications model frequently leads to unnecessarily complex system designs and a significantly degraded networking performance.

4.5.3 Publish-Subscribe

Publish-Subscribe architectures support *one-to-many*, *many-to-one*, *many-to-many* data-distribution. Publish-Subscribe adds a data model to messaging, with publish-subscribe nodes simply “subscribing” to data they need and “publishing” the information they create. Messages logically pass directly between the communicating nodes, and this fundamental communications model implies both: (i) discovery i.e. what data should be sent and (ii) delivery i.e. when and where to send it.

Publish-Subscribe systems are good at distributing large quantities of time-critical information quickly even in the presence of unreliable delivery mechanisms [34]. Publishers simply send data anonymously since they do not need any knowledge of the number or network location of subscribers. Subscribers simply receive data anonymously, without needing any knowledge of the number or network location of the publisher [35].

Within the distributed automation system, distributed command and control systems periodically send out data updates to controllers, loggers or other subscribers on the network. Publish-Subscribe is a necessity for these systems since the data are transmitted by the publishers to the subscribers when new data are produced. There is no request and no polling. The nodes interact in a similar way to the Event-Driven mode [36]. The advantages and disadvantages of each communication technologies is summarized in Table 4-7.

To conclude, publish-subscribe clearly offers advantages over point-to-point and client-server for delivering data in distributed and real-time environments. Publish-subscribe does not request traffic and the direct data transfer makes it much more efficient. In contrast, the client-server architecture requires all the information, if it is being generated at nodes, to be transferred to the server for later redistribution to clients. This adds an unnecessary unknown delay to the system, problematical for a real-time system; there needs to be control of the trade-off between reliable delivery and delivery timing. It is the concern with a guarantee of reliable delivery that destroys timing determinism, due to each retry taking up time.

Table 4-7: Summary of Communication Technologies

Type	Communication	Pros	Cons	Middleware
Point-to-Point	<ul style="list-style-type: none"> • One-to-One 	<ul style="list-style-type: none"> • given high bandwidth • simple straightforward model 	<ul style="list-style-type: none"> • does not scale well beyond a few nodes 	<ul style="list-style-type: none"> • telephone • TCP model
Client-Server	<ul style="list-style-type: none"> • Many-to-One 	<ul style="list-style-type: none"> • works well with data centric systems 	<ul style="list-style-type: none"> • irregular delay due to indirect data transmission • a bottleneck and presents a single point of failure • high bandwidth loaded with too many nodes • unnecessarily complex system designs and significantly degraded networking performance 	<ul style="list-style-type: none"> • Client-Server

Publish-Subscribe	<ul style="list-style-type: none"> • One-to-One • One-to-Many • Many-to-One 	<ul style="list-style-type: none"> • distributing large quantities of time-critical information quickly • mapping well with data distributed environments because the data flows directly from source to sink without requiring intermediate servers. • reliability and no single point failure • anonymous communications where publishers and subscribers do not need to know each other's physical network address 	<ul style="list-style-type: none"> • response time needs to be considered in strict real-time environments and time constraints 	<ul style="list-style-type: none"> • J2EE, • NET, • SOA
-------------------	--	---	--	--

Core technologies required for building distributed automation systems have been reviewed, and in general, automation systems are implemented from these standard technologies in manufacturing systems. Despite concerns of achieving real-time performance and security, Ethernet has gained much attention from researchers and industrial network vendors looking to overcome integration problems, in order to develop Ethernet TCP/IP for industrial automations. There is great potential to implement Ethernet with intelligent embedded microcontroller devices and a real-time publish/subscribe communication, to enable *flexible* and *non-proprietary* automation systems for agile manufacturing. Further enabling technologies and implementation approaches for distributed flexible automation systems are reviewed in the following sections.

4.6 A Distributed Automation System

The main focus of the author's research is the creation of future automation systems that enable more open, agile, and flexible manufacturing paradigms. There has been substantial ongoing research in the field of agent-based technology, service-oriented architectures and component-based design methodologies implemented with emerging powerful network communications. Ethernet, Modbus, and Profibus form a backbone for information exchange between intelligent devices and business planning level and a cost-effective way to support the lifecycle of manufacturing systems.

In this section, the key enabling approaches and technologies will be discussed to support the author's research for the potential tool development in the field.

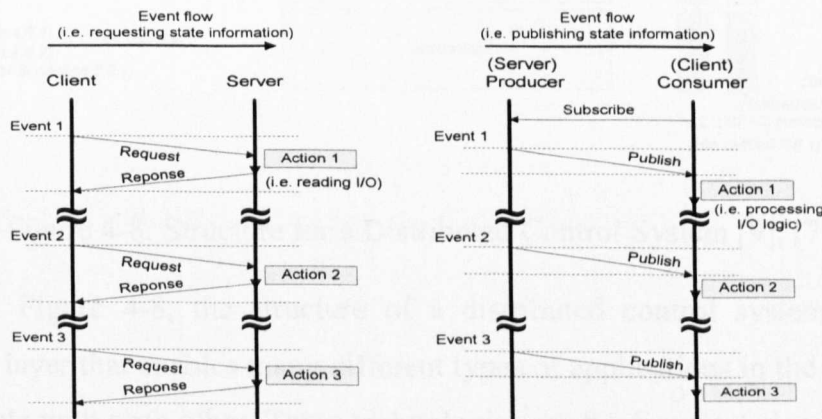


Figure 4-7: Request/Response and Publish/Subscribe Data-Flows for a Distributed Control System [71]

A general picture of a distributed automation system is depicted as in Figure 4-7. A node (event sink and event source) represents a controller, a micro processor or an embedded device which is distributed in the machine system to control its connected I/O devices locally. As discussed in section 4.3, Publish-Subscribe applications are most suited for distributed applications, with endpoint nodes that communicate with each other (peer-to-peer communication) by sending and receiving data anonymously.

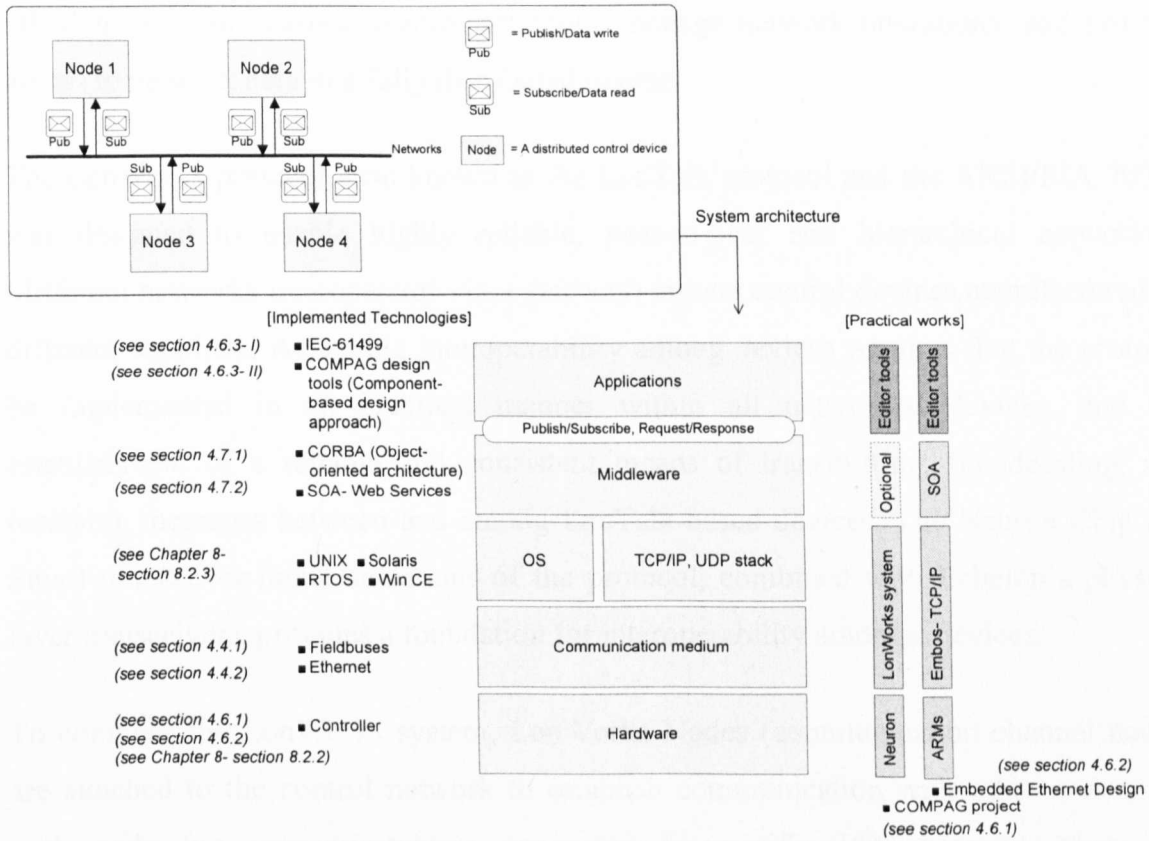


Figure 4-8: Structure for a Distributed Control System [9], [71]

As shown in Figure 4-8, the structure of a distributed control system includes an infrastructure layer that enables many different types of applications in the control nodes to communicate with each other. These technologies are fundamental elements that have been implemented in many research projects (see Chapter 5, section 5.3) in order to enable open and distributed manufacturing systems. Details of each implemented technology will be discussed later to highlight the suitable tools for the author’s research.

In the following section, the author presents a concise background of distributed control solutions from a previous study implementing the Lonworks system, and the potential solution of emerging miniature embedded devices regarding this research.

4.6.1 LonWorks System with Fieldbus

LonWorks system, developed by Echelon Co-operation in the USA, is an open solution for controlling distributed automation devices in home automation, industrial and transportation control systems. Its philosophy is to utilize a control network with a peer-to-peer communication (to allow intelligent devices to communicate directly to each

other) to monitor sensors, control actuators, manage network operations, and provide access to network data in a fully distributed manner.

The LonWorks protocol, also known as the LonTalk protocol and the ANSI/EIA 709.1, was designed to enable highly reliable, peer-to-peer and hierarchical networking (different networks interoperate via a gateway) among control devices manufactured by different suppliers. Achieving interoperability among devices requires that the protocol be implemented in an identical manner within all networked devices, and the establishment of a reliable and consistent means of transmitting, broadcasting, and receiving messages between and among LonTalk-based devices. The Neuron Chip and Smart transceiver implementations of the protocol, combined with Echelon's physical layer transceivers, provides a foundation for interoperability amongst devices.

To compose the LonWorks system, LonWorks Nodes (communication channel nodes) are attached to the control network to establish communication with other nodes, and each node has sensors/actuators connected to specific I/O channels. These are responsible for computing the data obtained from the sensors and passing the output command to activate actuators, according to the application programme loaded inside the node memory. The implementation of the LonWorks control system can be found in [33].

The heart of the node is the Neuron Chip C, and this includes three processors that provide both communication and application processing capabilities. The device manufacturer provides application codes to run on the Neuron Chip and I/O devices to be connected to the Neuron Chip. The programming applications are written Neuron C, based on ANSI C, and then this is compiled into binary bits "0", "1", etc, as understood by the Neuron Chip, and loaded into the node's memory.

The LonWorks system has developed a good high-level programming environment on Windows platform, to ease the building of networks. It has a generic network management tool and easily usable GUI (Graphical User Interface) for project administration, graphically visualized network variable binding, network variable browsing and adaptation to user needs by writing device specific control plug-ins.

It is the author's view that, even though the LonWorks system has developed control networks in advance of the open system architecture, flexibility and interoperability of other vendors, the proprietary nature of solutions is not fully resolved because devices outside the LonWorks agreement, i.e. LonMark, may not comply with the standard system due to the difference in the implemented protocol of bus systems, as discussed in section 4.4. The solution to interoperability among vendors' devices lies in a single control network standard like Ethernet. This should be introduced to free end-users from a few suppliers, so that they are able to respond to changes quickly.

4.6.2 Embedded Modules with Ethernet

Recently, embedded microprocessors have been rapidly improved, in terms of small size, low price, high processing speed, real-time performance and Ethernet connectivity, in order to facilitate an open and seamless integration of automation systems. Currently, there are some embedded solutions available on the market, such as NetSilicon Microprocessors (ARMs microprocessor) and Rabbit Microprocessors (Rabbit core, Dynamic C processing unit). These are capable of 10/100Based standard Ethernet connection and contain a broad set of industry standard peripherals, such as UBS, I2C, serial ports and an LCD controller. These embedded devices are designed to support various types of OS platforms including e.g. Linux, RTOS, Win CE, and Win XP.

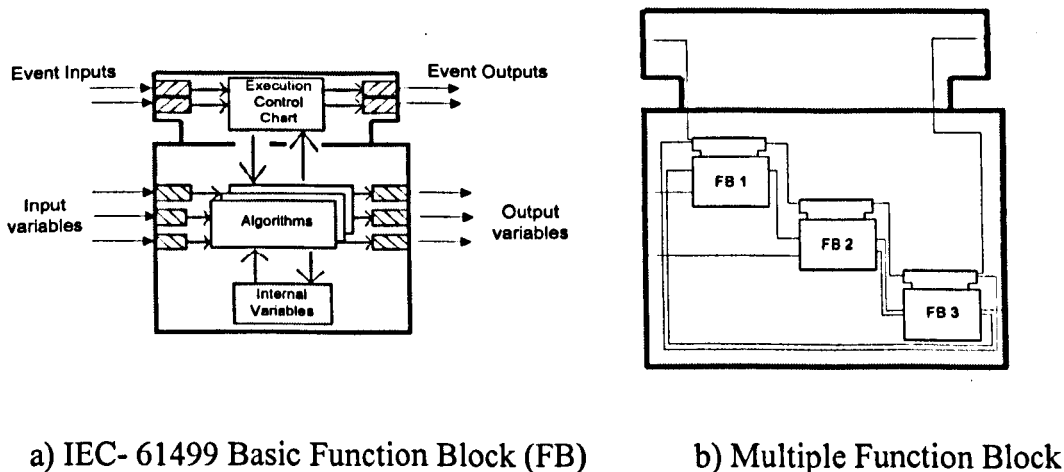
To conclude, the author has the novel idea of implementing this distributed control system infrastructure with embedded devices, Ethernet networks, the SOAP architecture and Web Services, publish-subscribe models and component-based design tools, in order to simplify and enhance the performance of automation systems (the implemented framework will be presented and discussed in *Chapter 7*).

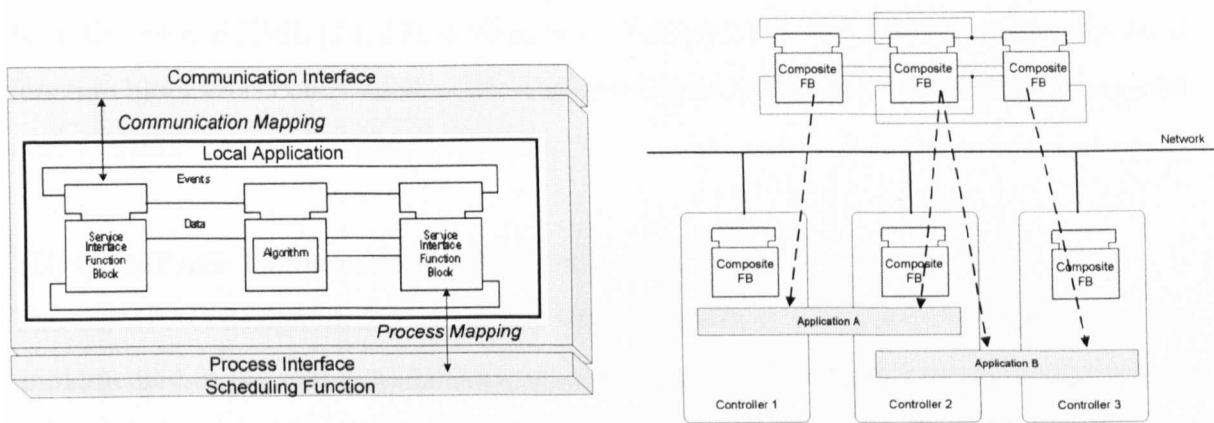
The author appreciates the potential of these embedded devices as the solution for open, flexible automation systems and ease of integration to higher control levels and ease of installation, in order to reduce development time and enable agile manufacturing. However, other distributed control infrastructures need to be chosen and developed as appropriate so as to effectively support the integration of these smart embedded devices in automation systems. The technologies for distributed automation systems will be outlined, according to the structure for a distributed control system (Figure 4-8), in the following sections:

4.6.3 Distributed Control Applications

(I) IEC -61499 Standard

IEC-61499 standard was proposed for the application of function blocks in distributed industrial-process measurement and control systems [51]. The developing standard of IEC-61499 presents an approach for distributed process control systems, whose components are function blocks. The control applications may be distributed among devices of a system, with each of these applications using one or more resources: these resources are defined as “containing one or more function blocks that may be activated by one or more control flows” [80]. In distributed control domains, the coherence of the actuation/sensing actions and the execution time of control loops are very important. The information exchange between the resources is defined by the specification of event and data variables: events are used to ensure the control flow of an application, and data variables are updated when executing an algorithm, and can be associated to an event. The arrival of a new event at the input of a function block launches the mechanism for the execution of algorithms, based on the ECC invocation. The Event-Driven concept of IEC 61499 is illustrated in Figure 4-9.





c) Composite FB and Process Interface

d) IEC Integration of the Distributed System

Figure 4-9: Event-driven IEC 61499 Execution Controls [51]

Event-driven state machine control of an IEC 61499 execution is illustrated in Figure 4-9. Basic function block types (as in Figure 4-9/a) are defined by declaring:

1. Execution Control Chart (ECC),
2. The algorithms, whose execution may be invoked by the ECC, are pre-defined function block (FB) behavior with external (interlocking) and internal variable.
3. Internal Variables (Local state variable of sensors and actuators).

The basic function block may be used to build a more complex application, as depicted in Figure 4-9/b. Figure 4-9/c shows details of the composite FB of the service interface function block for the device network communication, with other services provided by the resource's operating system and control algorithm performing I/O execution.

To compose the distributed system, in which devices may communicate with each other over one or more communication links and may interface to controlled and processed machines, applications should be distributed among one or more devices interconnected by event connections and data connections, to form the integration of the distributed system, as shown in Figure 4-9/d.

At present, there are a number of researchers implementing IEC 61499 function blocks in the design of distributed control systems (DCS) with high level programming such as

Java, C/C++ and XML [22, 27] on fieldbus or Ethernet networks. The implementation of function blocks has contributed to the improvement of reusability, re-configurability, and interoperability among different vendors.

(II) COMPAG- Component-based Design Tools ⁽⁵⁾

The COMPAG design methodology has been proposed at Loughborough University and implemented in the real automation system on a Ford Test Rig at Loughborough and an industrial test machine at Krause in Bremen, as reported in [73]. The concept of component-based design methodology will be further developed in this project, with the design of other engineering tools (software applications) to aid the machine commissioning and installation.

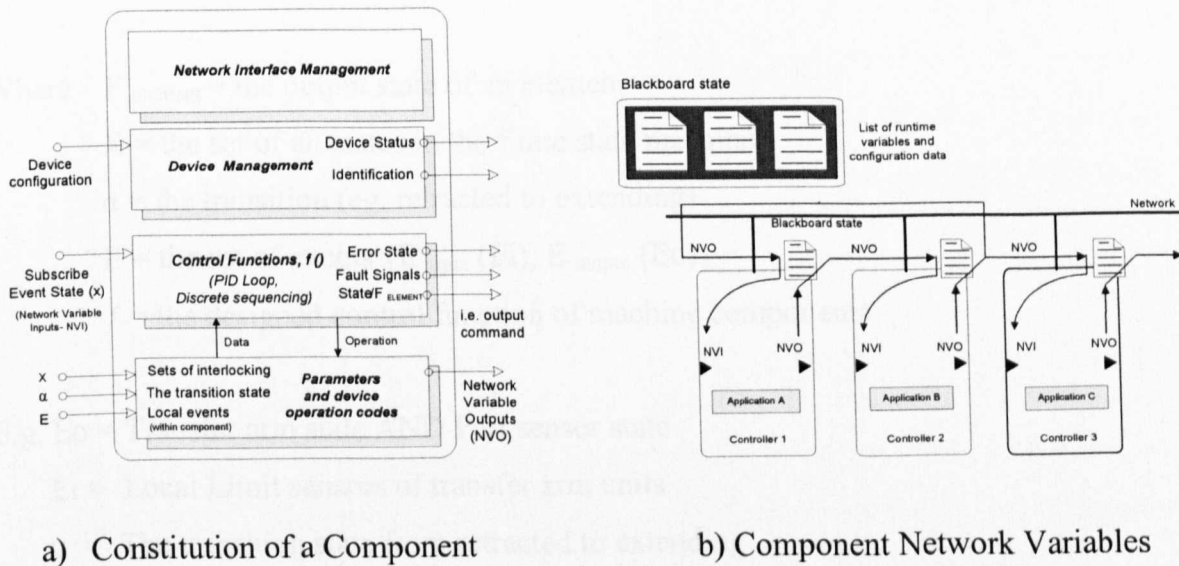


Figure 4-10: Functionalities of a Component (adopted from [33])

The term “component”, used throughout this research, is defined as the previous research at MSI. The component is viewed as an autonomous unit consisting of the automation devices (sensors and actuators), computing hardware (processor, memory, communication interface, electronic interface) and control software (application program, OS, and communication protocol) [11]. This definition of a component is illustrated as in Figure 4-10/a). The adoption of this approach is presented in Chapter 6-section 6.8.3.

5- COMPAG design tool, PDE (Process Definition Environment), is a tool which supports the design and integration of the machine from a component library. The tool was developed by MSI research group for the COMPAG project based on the component-based design approach

In the development of automation systems, a system is built from sub-system units that normally contain many components. Communication between components is event-driven, in which the states of devices are defined as inter-connected logic related to other devices. In addition, the states can be published to desired subscribe nodes as a network variable, by sending/receiving event messages through the output and input network variable interfaces (NVI, NVO), respectively shown in Figure 4-10/b).

The behaviour of the component is represented by using finite state machines, F_{element} , as a set of functions of component states, transitions, and a combination of events. The finite state machine for the element adopted by S.M. Lee [11] is as follows:

$$F_{\text{element}} = f(X, \alpha, E) \quad [1]$$

Where F_{element} = the output state of an element
 X = the set of all states in the finite state machine
 α = the transition (eg. retracted to extending)
 E = the set of events $\{E_{\text{input}}(E_i), E_{\text{output}}(E_o)\}$
 f = the designed control function of machine components

E.g. E_o = Transfer arm state AND Part sensor state

E_i = Local Limit sensors of transfer arm units

α = The transition state from retracted to extending

X = Other associated unit states

f = When $\{(X \text{ AND } \alpha) == 1 \text{ OR } (E == 0)\}$ then action; *discrete functions*

This approach enables the generic operation of devices to be pre-programmed and encapsulated in the component. The system operation can be configured by interlocking the event condition of the control elements through its states and the states of other elements, as specified in the function [1]. The composed functionality of a component has been developed to support the development of generic control functions of the system, system installation and independent reconfiguration without any prior knowledge of the application.

In general terms, the component can physically be seen as the controller device, with its own physical resource (sensors and actuators) and the control application to perform manufacturing tasks. The functional constituent of such a component has the capability to interface, in order to process applications such as device binding, simulation tools, on-site and remote monitoring. This is achieved by manipulating the component data obtained from output state variables of the encapsulated function entities. Furthermore, the basic operation of devices may be pre-programmed by the component suppliers and encapsulated into the component, in order to hide away the abstract functionalities and complexity from users. This “black-box’ implementation approach allows changes with minimal disruption to the system [73].

In the building of a component-based automation system, the component has predefined physical resources within the component boundary and is not accessible across components i.e. the component is independent of other components [11]. This increases the flexibility of the system as shown in Figure 4-10/b) in which components with their own applications (A, B, C) are distributed across the system and communicate via the black board state.

4.7 Middleware Servers

With the adoption of distributed objects and the heterogeneous nature of computing systems over the past few years, the middleware programming architecture has evolved to provide support. Middleware is the key engine of development which that acts as the glue to connect diverse computer systems. In the history of object-based programming, Microsoft’s Component Object Model (COM) and the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG) were two leading distributed-object technologies that were widely used. A comparison of these two middleware approaches can be found in [g4].

Extensive reviews on agent-based manufacturing, including the holonic approach (*section 4.8.1*), have shown that most researchers favoured the CORBA middleware, due to its wide use and acceptance. Therefore, in the literature review section, the CORBA middleware will be compared with an emerging middleware technology of Service-oriented architecture (SOA).

4.7.1 CORBA

The concise mechanism of the object-oriented invocation in CORBA is summarised in this section, with full details to be found in [g4, g5]. CORBA can be conceptualized as a communication bus for client-server objects, and, since CORBA is a three-tier distributed objected mechanism, the terminology “Client-Server” is applied within the context of a specific request, as in Figure 4-11.

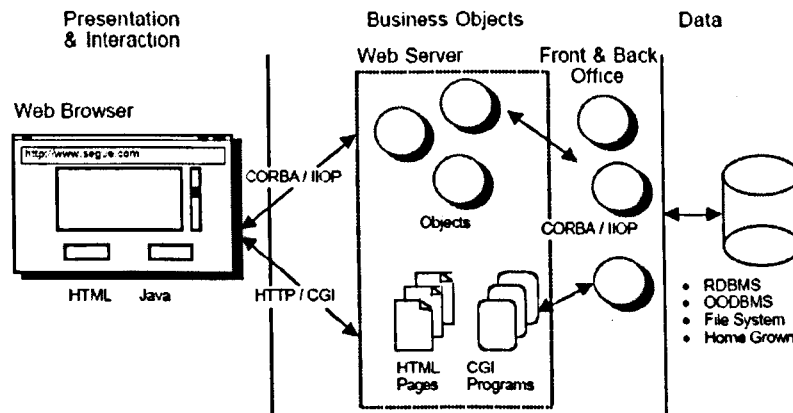


Figure 4-11: Three Tier e-Business Architecture [g4]

The first tier in the architecture is the presentation and interaction layer; for example, a web browser or a client. The middle tier consists of the application logic, which can be constructed from multiple components, such as web and application servers. The final tier includes data repositories such as object-oriented databases. With the growing need to integrate multiple heterogeneous systems in many areas, such as businesses and manufacturing, CORBA is increasingly used as the platform for integrating distributed objects [g4].

A client invokes a method on a server through the interface and object request broker (ORB). Exported server interfaces must be specified in the CORBA standard Interface Definition Language (IDL). IDL is part of the CORBA standard and permits interfaces to objects to be defined, independent of an object’s implementation. IDL is used as input to an IDL compiler that produces source code. Source code can be compiled and linked with an object implementation and its clients, which enable a program or object written in one language communicate with another unknown programming language. IDL also enables distributed applications to invoke operations transparently on remote networked hosts. IDL files are similar to C header files, except for the actual code implementation

(behind the IDL definitions) being located on a host remote to the caller (see [83, 87] for details).

An IDL interface description is mapped, using IDL compiler, to native language bindings such as Java, C++ and others. This allows each programmer to write source code independently in the most appropriate language [g5].

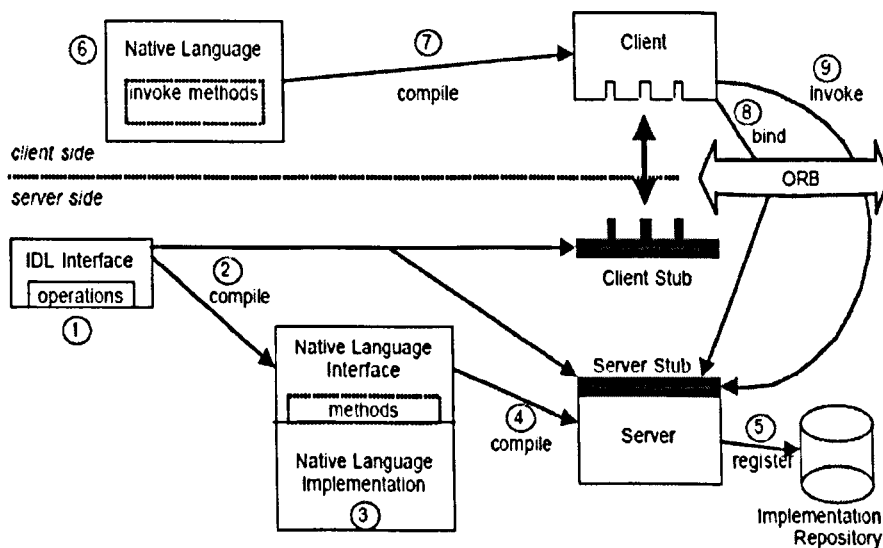


Figure 4-12: The CORBA Client/Server Invocation Methods [g5]

Central to the CORBA architecture is the Object Request Broker (ORB). The ORB serves as an object bus that transparently handles all client-server interactions between objects. The ORB is responsible for locating the object, establishing a communication channel, invoking the request, and managing the reply on the behalf of the client [g4], as shown in Figure 4-12. (Steps 8-9)

The details of CORBA invocations as presented in “A CORBA Primer”, Segue [g5] are

Step 1: Identify an IDL interface

Step 2: An IDL compiler is used to generate a server stub, a client stub that gets linked to a program wishing to invoke statically a server method through the associated interface.

Step 3: Implement the server

Step 4: Compile the server program and link to a server stub to generate the executable server program that can be invoked via a CORBA method

Step 5: Register the server in the implementation repository.

Step 6: Server object method calls as if they are local.

Step 7: Compile the client program and link in the client stub

Step 8-9: ORB binding to the server object and obtain a reference for the client to invoke the method call.

The CORBA developer group has attempted to advance CORBA components, so that they seamlessly fit into the standard infrastructure provided by the web. There are some deployments that use JavaBeans implemented with CORBA as the standard for component objects, to enable an open and independent operating system.

The internet Inter-ORB Protocol (IIOP) is the CORBA standard that guarantees interoperability between ORB implementations, as well as allowing applications built with different vendors' ORBs to communicate and share objects for distributed object invocations.

4.7.2 SOA Middleware

SOAP is a protocol for the key Web Service standards, WSDL and UDDI (see Figure 4-13) (Chapter 6- section 6.7.1). These are all based on the XML messaging format that is used to send information from one application to another. SOAP allows the integration of application-to-application transactions over the web. Unlike previous middleware technologies (e.g., DCE, CORBA, DCOM, MOM), SOAP middleware is available on any platform and it supports many programming languages.

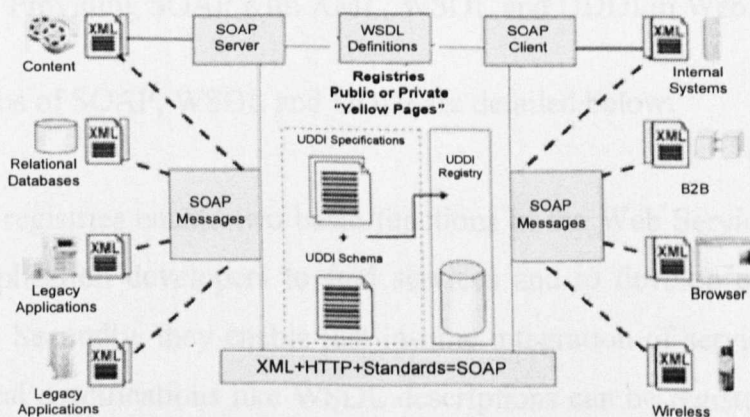


Figure 4-13: SOA Client –Server Middleware Model

SOAP defines a framework for message structure and a message processing model. It does this by providing an XML-based messaging framework that is: 1) extensible, 2) usable over a variety of underlying networking protocols and 3) independent of programming models.

SOAP also defines a set of encoding rules for serializing data and a convention for making remote procedure calls (RPC) [g1]. SOAP provides a rich and flexible framework for defining higher-level application protocols that offer increased interoperability in distributed, heterogeneous environments. The extensibility features built into SOAP allow the various Web Services protocols to be integrated individually and incrementally, as well as to be improved and versioned in isolation, without affecting the rest of the protocol stack [9].

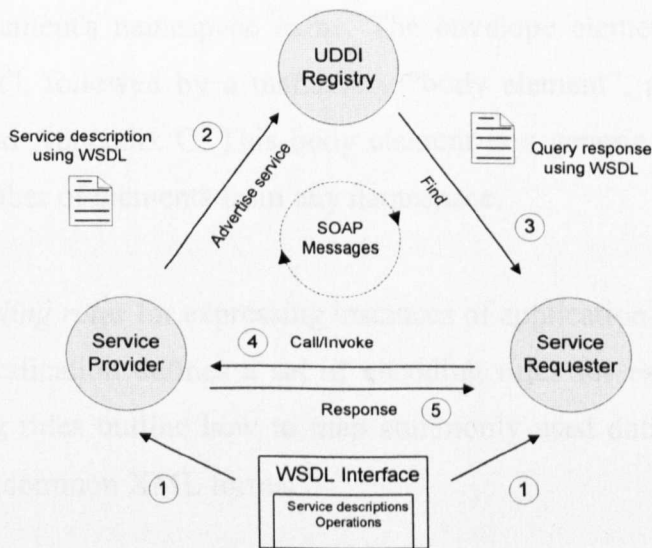


Figure 4-14: Providing SOAP with XML, WSDL and UDDI in Web Service [49]

The mechanisms of SOAP, WSDL and UDDI are detailed below.

UDDI service registries enable two basic functions in the Web Services model. Firstly, they allow application developers to find services and to develop code that relies on those services. Secondly, they enable just-in-time integration of service components. In UDDI, technical specifications like WSDL descriptions can be registered and then used to qualify the registry description of a compliant service [g19].

The Web Services provider registers its applications and links by subscribing them to UDDI registry at the time of development. At the client site, the Web Services requester can look up the services from providers by using WSDL file, in order to find out the location of the service, the function calls, and how to access them. Web Services mechanism relies on the SOAP message on HTTP protocol in this service invocation method. The practical implementation of this WS model can be viewed in [88].

The SOAP protocol is XML-based and consists of three parts:

1) *A SOAP envelope* for describing the message content and how to process it:

The envelope element is always the root element of a SOAP message. This makes it easy for applications to identify "SOAP messages" by simply looking at the name of the root element. Applications can also determine the version of SOAP being used by inspecting the envelope element's namespace name. The envelope element contains an optional "header element", followed by a mandatory "body element", as shown in Chapter 9-section 9.2.1 and Appendix C. This body element is a generic container in that it can contain any number of elements from any namespace.

2) *A set of encoding rules* for expressing instances of application-defined data types:

The SOAP specification defines a set of encoding rules for exactly this purpose. The SOAP encoding rules outline how to map commonly used data structures (like structs and arrays) to a common XML format.

3) *A convention for representing remote procedure calls and responses:*

Although the SOAP specification has evolved away from objects, it still defines a convention for encapsulating and exchanging RPC calls using the messaging framework described above. Defining a standard way to map RPC calls to SOAP messages makes it possible for the infrastructure to translate automatically between method invocations and SOAP messages at runtime, without redesigning the code around the Web Services platform [g6].

To conclude, SOAP defines a simple and extensible XML messaging framework that can be used over multiple protocols with a variety of different programming models, although the specification codifies how to use SOAP with HTTP and RPC invocations.

SOAP also defines a complete processing model that outlines how messages are processed as they travel through a path. In general, SOA middleware and SOAP message processing can be affiliated with other development tools such that Microsoft .NET Framework, Visual Studio.NET, C/C++ tools, Java, so as to support the development of Web Services applications.

4.7.3 Debate: CORBA VS. SOA Middleware

It has been an ongoing debate as to which of these middleware technologies is better suited for business-to-business and business-to-shop floor integration. Having studied and reviewed a number of papers, there is no straightforward and convincing answer to this question. Selection depends on opinions and experience. It is hard to justify the answers, especially without a specific scope for comparison. In this respect, the author has scoped the discussion by considering the relevant content to agility features required in manufacturing systems, as addressed in Chapter 3. The assessment of both middlewares is presented below.

Assessment of CORBA and SOA in the Requirement of Agile Manufacturing Systems

<i>(Q) Open standards and the pervasiveness of these middlewares in vendor solutions</i>	
SOA	CORBA
<p>[Advantages] ([g22])</p> <ul style="list-style-type: none"> • SOA-WS are based on emerging standards, and SOAP, WSDL, UDDI, have achieved broad acceptance in the industry. • WSDL also fully supports transport neutrality, as it allows separate specifications of the abstract service interfaces and their bindings for each specific transport protocol. • SOAP is the key to supporting a transport-neutral infrastructure for the actual production and consumption of messages, as it supports binding to different transports. 	<p>[Advantages]</p> <ul style="list-style-type: none"> • Promotes interoperability <p>[Disadvantages]</p> <ul style="list-style-type: none"> • Although CORBA has been implemented on various platforms, the reality is that any solution built on these protocols will be dependent on a single vendor's implementation. In the case of CORBA, every node in the application environment would need to run the same ORB product ([g21]).
<p>Points: For SOA and WS, heterogeneous applications running on different platforms are allowed to interoperate through a consistent, well-defined interface.</p>	

(Q) Requirements from an IT and Business perspective – widely distributed and highly integrated, loosely coupled and yet manageable applications

SOA	CORBA
<p>[Advantages]</p> <ul style="list-style-type: none"> • SOA is mainly for integrating existing systems in a decoupled way [g23]. • The service interfaces are defined in WSDL (which is itself an XML application), and XML technologies can be applied to interrogate the service capabilities and integrate discovered functions such as Web Services orchestrations. Web Services provide a semantically rich integration environment: they make it much easier to build business process management solutions that “orchestrate” multiple business functions from disparate applications and allow the system to apply “business rules” dynamically, e.g., for content-based routing of messages. They also provide semantic mappings between multiple XML business documents in a declarative fashion [g22]. 	<p>[Disadvantages]</p> <ul style="list-style-type: none"> • With CORBA, the focus is on Object Request Brokering. This results in tight coupling to well-defined interfaces, a broker infrastructure and multi-language mappings. These types of coupling are needed for almost the same reasons as Java-Interfaces for compile-time checking, implementation hiding, richly typed interface and easy client-programming [g25]. • CORBA can be distributed with different sets of interfaces if any prove too cumbersome for the task. It was difficult to manage different versions of CORBA interfaces, and these interfaces consumed excessive computing overheads [g26].
<p>Points: From this viewpoint, CORBA as OO needs more effort to commission than SOA, which provides consistent API's and homogeneous technology with loose coupling between existing applications.</p>	

(Q) Integration of applications from business services and partners

SOA	CORBA
<p>[Advantages] (g22)</p> <ul style="list-style-type: none"> • SOA will enable faster application integration using the WS standard (WSDL, UDDI and XML message types) and widely used SOAP protocols compatible with HTTP. WSDL also fully supports transport neutrality, as it allows separate specifications of the abstract service interfaces and their bindings for each specific transport protocol. • SOAP is the key to supporting a transport-neutral infrastructure for the actual production and consumption of messages, as it supports binding to different transports. • The self-describing XML documents and SOAP messages make it possible to build a loosely coupled, document-style integration environment. 	<p>[Advantages] ([96])</p> <ul style="list-style-type: none"> • The CORBA environment is best suited for applications developed and controlled by itself, in which all or most of the programming language is C, C++, or Smalltalk. • A bridge between the Java environment and CORBA has been available for a significant period of time. • It is a mature technology that still has its use in high-volume, highly secure, object-oriented applications within an enterprise. <p>[Disadvantages]</p> <ul style="list-style-type: none"> • CORBA has failed on the Internet and it is not used for public integration amongst companies. Rather, CORBA is typically used for communication among application components developed by the same team, but it is not used by companies to offer a public remote API that anyone could utilize [g23].
<p>Points: In regards to business programming, CORBA is best suited to tightly coupled transactional systems requiring high security.</p>	

<i>(Q) Cross-platform and cross-programming language interoperability</i>	
SOA	CORBA
Points: Both CORBA and Web Services provide interoperability across programming languages, operating systems, and hardware platforms.	
[Advantages] <ul style="list-style-type: none"> • Web Services are based on several (emerging and de facto) standard technologies, primarily SOAP, WSDL, and UDDI. • SOAP-XML is neutral with respect to the network access protocols, and so these data types and service interfaces can be mapped to different languages and middleware interfaces, thereby providing language/platform neutrality. Web Services standardize the messages exchanged by the interacting entities, which can then be mapped to an arbitrary object model. • WSDL also fully supports transport neutrality as it allows separate specifications of the abstract service interfaces and their bindings for each specific transport protocol [g22]. 	[Disadvantages] <ul style="list-style-type: none"> • Regarding to interoperability, CORBA assumes that all interacting entities conform to a standardized object model.

<i>(Q) Integrated middleware in embedded control devices</i>	
SOA	CORBA
Points: As reported by Roy Bell [g27], CORBA is 3 times bigger than the SOA middleware on the device, and about 6 times faster.	
[Advantages] <ul style="list-style-type: none"> • Smaller memory footprints [Disadvantages] <ul style="list-style-type: none"> • Slower processing speed, thus require more powerful processor specifications 	[Advantages] <ul style="list-style-type: none"> • Faster processing speed [Disadvantages] <ul style="list-style-type: none"> • CORBA is still a heavyweight solution for many smaller embedded systems, since the overhead of C++ was overcome by a combination of careful use and cheaper computing power [g24].

<i>(Q) Engineering tools and manufacturing application integration</i>	
SOA	CORBA
Points: Regarding this point, there are no real advantages or disadvantages. Both middleware integrations have been implemented in the same manner. CORBA uses IDL for the application interface, whilst SOA uses WSDL.	

Overall, it has been addressed by Sayjay [g22] that CORBA is better suited for building distributed applications in controlled environments. Such environments make it possible to share a common object model among all distributed entities, there are no restrictions on the granularity or volume of the communications between distributed entities, and deployment is more or less permanent, so that the system may find a benefit in mapping the network addresses directly to object references. However, the integration scenarios described above require a loose coupling, where CORBA may not be the best fit.

The definition of SOA includes the usage of technologies such as WSDL, UDDI, SOAP and XML technologies. With these technologies, developers can build applications in business domains or automation domains for control devices, using these standards to build components in the form of Web Services. In addition, Web Services semantics are standardized in the form of message definitions and service interfaces, promoting a wider variety of new applications [g22].

However, there are some concerns in implementing SOA and Web Services on the automation systems including:

1. Methodology and reliability of discovering the required services (*discussed in Chapter 6B*)
2. Providing acceptable performance (*discussed in Chapter 9*)
3. Messaging reliability and missing packet recovery (*discussed in Chapter 9*)
4. Security (*addressed future work*)
5. Fault handling, in order to maintain the reliability of the transaction whenever the service is unavailable due to changes or being closed (*discussed in Chapter 9*)

In this research, these issues need to be taken into consideration and resolved by experimentation and investigation on the implemented industrial test rig.

4.8 Agent-Based Manufacturing System

A distributed system consists of a number of components, which are loosely coupled and capable of performing simple local operations, such as data conditioning and local feedback controls without a master controller. The components are connected by some sort of communication medium, such as fieldbus or Ethernet, and applications are executed by using a number of processes in the different component systems. These processes communicate and interact to achieve productive work within the application. It is envisioned that by removing the need for a master controller and enabling local computing and control capabilities within each intelligent device, there would be no need to develop rigid conventional control programs.

There is considerable research proposing hypotheses in the domain of distributed open control system design (i.e. [6], [12], [70], [84], [85], [118]). However, many approaches are similar in concepts and key enabling technologies. In general, the key methodology to enable such an environment can be broadly classified into two approaches:

- 1.) Object-Orientated architecture
- 2). Service-Orientated architecture and Web Services

4.8.1 Object-Orientated Architecture (OOA)

This approach was proposed as a new distributed manufacturing control paradigm, presenting distributed structures based in autonomous and co-operative entities that have the ability to respond promptly and correctly to external changes. This differs from conventional approaches, in that there is an inherent capability to adapt to change without external intervention [15]. The concept of this approach is centred around the development of control software, based on a formal modelling of the entities involved in a cell. Regarding interactions, the cell controller takes care of co-ordination and synchronization issues, while individual objects are responsible for their own activities [16].

A significant development in the object-oriented concept has attracted many research consortiums and institutions. The Holonic approach, which is based on multi-agent technology, has been presented as the best outcome of machine controls, shop floor

controls, scheduling and planning, by means of autonomous, distributed decision making smart entities called “*holons*”. These entities interact via co-operation protocols within the manufacturing cell to perform their responsible tasks, in order to support the runtime reconfiguration demanded by shop floors. A typical holonic framework is seen in Figure 4-15.

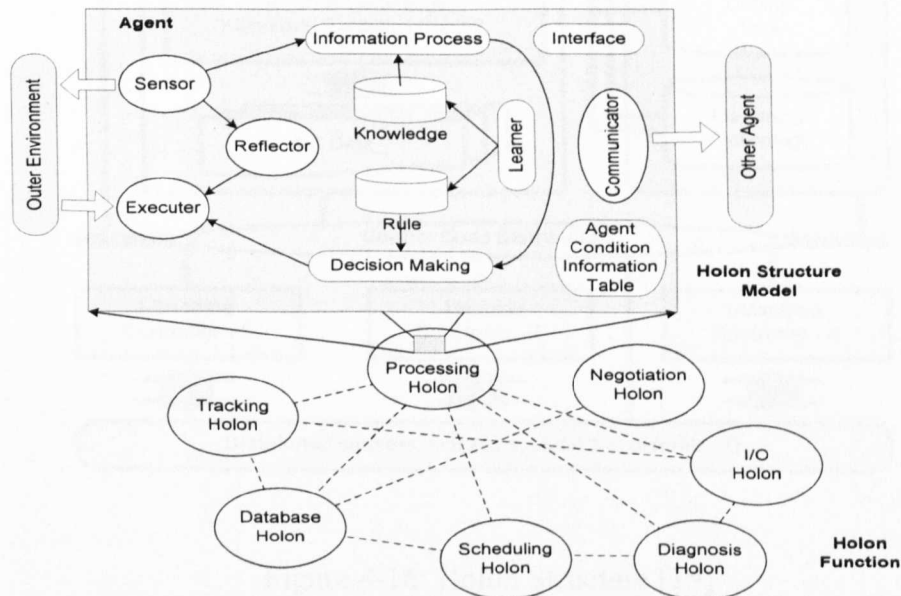


Figure 4-15: Holonic Framework via Mobile Agent (adopted from [17], [50])

The manufacturing cell (the so called “holon”) consists of a number of individual mobile agent units corresponding to specific functionalities in the physical configuration. Each holon is a dynamic system with input, processor, output and a controller. Structurally, a holon has:

- a) A physical processing part that is associated with an item of shop-floor machinery to process artifacts
- b) An information processing part that handles knowledge management and executes software algorithms pertaining to the holon’s control system specifications [18]. An example of the holon architecture can be seen in Figure 4-16

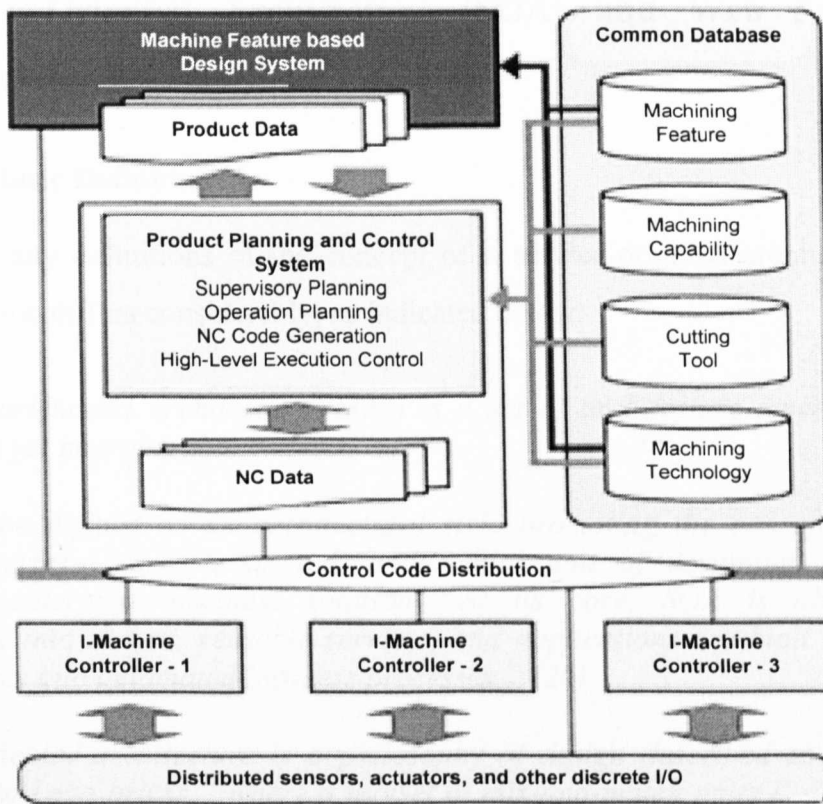


Figure 4-16: Holon Structure [19]

After the substantial number of reviews regarding the practical implementation of the Holonic design approach in the manufacturing control system design, many researchers have adopted this approach in the domain of programming object-oriented software. This has been implemented with standards of the emerging *Function block-based control IEC 6149*, to enable flexible, reconfigurable automation systems. The practice of using holonic manufacturing systems demonstrates how holons act autonomously and co-operatively through the interaction of the software components inside its software and mobile agents. This automatically produces function block applications that implement the desired manufacturing service within the scope of the IEC 61499 architecture [18].

However, it is the author's point of view, and indeed other researchers such as François Jammes and Harm Smit [9], that the Holonic approach has not made significant inroads in manufacturing plants, due to a lack of widely accepted standards, proprietary standards and complexities of the approach.

4.9 Service-Oriented Architecture (SOA) and Web Services for Manufacturing Systems

4.9.1 SOA Basic Definitions

There are many definitions of the concept of a service-oriented architecture (SOA). Each definition differs considerably, as indicated below:

“A service-orientated architecture (SOA) is a set of architecture tenets for building autonomous yet interoperable systems.” [9]

“SOA can be defined as an architectural style promoting the concept of business-aligned enterprise services as the fundamental unit of designing, building, and composing enterprise business solutions. At its core, SOA is about factoring functionality into shared, reusable services, and applications are built by assembling those services into automated business processes.” [20]

“Service-oriented-architecture is a philosophy of design described as “the software equivalent of Lego bricks,” where a toolset of mix-and-match units (“services”), each performing a well-defined task, can reside on different machines (including geographically separated ones), ready to be used when needed.” [21]

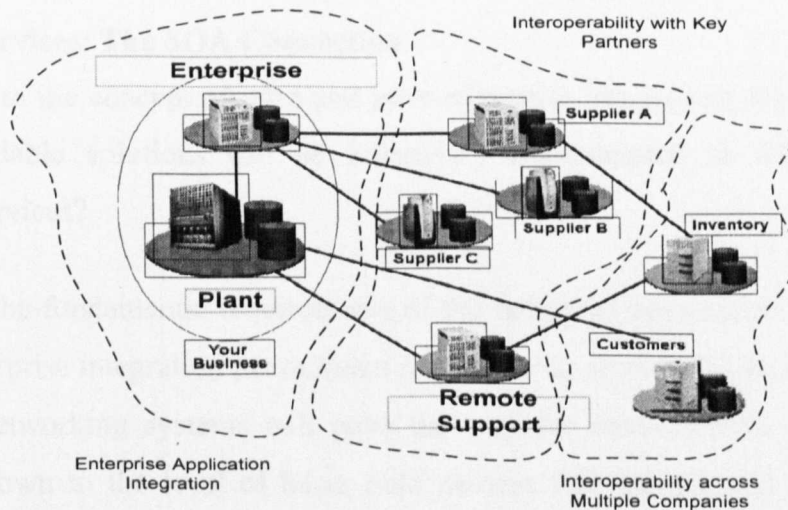


Figure 4-17: Business Ecosystems [g1]

As depicted in Figure 4-17, the service-orientated architecture in the global aspect may be seen as integrated applications within the generic enterprise, in order to extend the reach of businesses to partners and customers. This creates business efficiencies and exposes companies to new sources of revenue.

As reported in [g1], the adaptation of this technology would split the business ecosystems into three tiers:

Tier1- Enterprise Application Integration

This is the starting point for most companies. It allows them to expose legacy applications to business applications in heterogeneous environments, without having to rewrite large amounts of applications code.

Tier2- Interoperability with Key Partners

Required to integrate the business in association with key partners. SOA has been implemented because it allows for interoperability among applications across communication mediums, such as the public Internet.

Tier3- Interoperability across Multiple Companies

Companies want to extend their computing out to more partners and customers, in order to build business ecosystems.

4.9.2 Web Services: The SOA Connection

With regards to the concept of intra and inter-enterprise integration, the main question is what available solutions can be effectively implemented in SOA to connect ubiquitous services?

Considering the fundamental requirements of the industrial automation system and the business enterprise integration (*as outlined in Chapter 2, section 2.3.4*), the evolution of the device networking systems will pave the way for cost-effective communication paradigms, down to the level of basic field devices like sensors and actuators. As a consequence, the upcoming SOA and Web Services for manufacturing systems would bring the following requirements and challenges [9]:

Interoperability: Automation system shall be implemented independent of any vendor specific operating systems or programming languages, thus maximizing use of resources.

Reduced complexity: Devices shall make the automation system simple and easy to commission and diagnose by non-expert persons.

Ease of installation and preparation: WS shall enable efficient plug-and-play connectivity.

Reusability: The design of WS enabled devices shall enable the programming code to be reused at different architecture levels and in different devices.

Seamless integration: A device shall present a high-level management interface (typically graphical) in order to facilitate configuration, monitoring, fault diagnosis and maintenance.

In these sets of end-user requirements, as shown in Figure 4-18, the utilization of Web Services within the manufacturing system should facilitate the integrated sets of applications through the standard Web Services interface (discussed in Chapter 6B) These will be used by various interested parties to support production throughout the manufacturing lifecycle.

Based on this framework, the modularization of the production system depends on the decomposition of the present “control-orientated structure” into function modules, with a “manufacturing-task-orientated structure” [117]. At the automation level, the I/O states of the component (i.e. events variables) and the device functionality is exposed to the manufacturing process as values and services, that can respectively be used and managed by higher level applications. Dynamic service discovery and composition of the manufacturing process tasks are achieved by Service orchestration engine (Chapter 8- section 8.5.2).

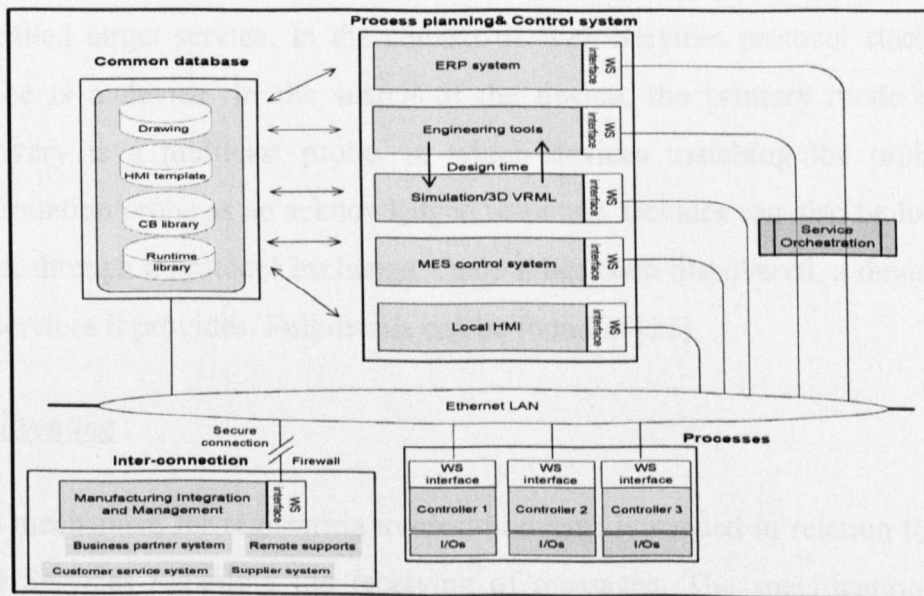


Figure 4-18: SOA Web Services Structure in Manufacturing Systems

As presented in [9], the core Web Services architecture required to achieve the above requirements (i.e. the Devices Profile for Web Services (DPWS)) is shown in Figure 4-19 and contains the following protocols and capabilities. The details of SOAP, UDDI and WSDL were presented in section 4.7.2.

WS-Discovery	WS-Eventing
WS-Addressing WS-Metadata Exchange WS-Policy WS-Security	
SOAP 1.2 WSDL 1.1, XML Schema	
UDP	HTTP 1.1
	TCP
IPv4/IPv6	

Figure 4-19: Devices Profile for Web Services (DPWS) Protocol Stack. [9]

XML Schema

The XML Schema is the definition of the data formats constructed that allow developers to create precise descriptions used for the message addressed to and received from services [g13].

WS-Discovery

This defines a multicast protocol to search and locate plug-and-play discovery, the so-called target service. In the context of Web Services protocol stack, a target service is a device. In the search of the device, the primary mode of service discovery is a multicast probe, in which devices matching the probe send a confirmation/probe as an acknowledged response. Devices can also be localized by name, through a protocol exchange. Once it has been discovered, a device exposes the services it provides. Full details can be found in [23].

WS-Eventing

This mechanism for registering interest in events is needed in relation to the set of Web Services regarding the receiving of messages. The specification defines a protocol for one Web Services (called a “subscribe”) to register interest (a

“subscription”) with another Web Services (an “event source”) in receiving messages about events (“notifications” or “event messages”). The subscriber may manage the subscription by interacting with a Web Services (“subscription manager”) designated by the event source [24].

WS-Eventing is intended to enable the implementation of a range of applications, from device-oriented to enterprise-scale publish-subscribe systems [9].

WS-Addressing

This mechanism provides transport-neutral methods for addressing Web Services and messages. Specifically, this defines XML (Extensible Markup Language) elements to identify Web Services endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks which include processing nodes, such as endpoint managers, firewalls, and gateways in a transport-neutral manner (i.e. HTTP, SMTP, TCP, UDP) [25].

WS-Metadata Exchange

Web Services use metadata to describe what other endpoints are required (e.g. description, schema, and policy) in order to interact with them, thus providing a web service introspection mechanism. The interactions defined in the WS-Metadata Exchange are intended for the retrieval of metadata only. They are not intended to provide a general purpose query or retrieval mechanism for other types of data associated with a service, such as state data, properties and attribute values [26].

WS-Policy

WS-Policy gives generic instructions of how senders and receivers can specify their requirements and capabilities in the form of policy assertions [9].

WS-Security

WS-Security provides quality of protection through message integrity, message confidentiality, and authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies. WS-Security also provides a general-purpose mechanism for associating security tokens with

messages. No specific type of security token is required by WS-Security. It is designed to be extensible, in order to support multiple security token formats. A client might provide proof of identity and proof that they have a particular business certification [g2].

The introduction of DPWS paves the way for the use of a unique technology base, via Web Services, across the entire heterogeneous enterprise applications, from the sensors/actuators level up into ERP/MES level [9]. However, the use of Web Services needs to be carefully considered in the deterministic real-time performances of automation systems and the security on the open network, as shown in the following section.

4.9.3 Web Services Consideration Issues

Although Web Services satisfy most of the manufacturing requirements, there are major concerns in implementing Web Services based on SOA solutions, as follows:

- a) **Security:** integrating business applications outside companies with partners and customers over the internet requires secure connections that prevent hackers spam, viruses, and unauthorized users gaining access to an operation terminal. Sufficient tools need to be implemented effectively, in order to secure the system completely in a cost effective way.
- b) **Reliability:** Characteristics of internet technologies are time delays and uncertain responses. Developers need to take this into account; how can they maintain the integrity of transactions over the internet?
- c) **Performance:** Every packet must be counted on sending and receiving data with regards to the impact on performance degradation, as a result of overheads and bandwidths by concurrent users using the network.

Initial solutions to some of the issues mentioned have been presented, but more detailed solutions will be discussed throughout the course of this thesis.

For example, the network connectivity can be secured by encrypting and decrypting each data transmission, using private networks with access protection as a secure token for remote login, as reported in [74]. In addition, the reliability and performance of implementing Web Services for device synchronisation is another factor that needs to be taken into consideration for connecting real-time communication between devices. If the delay associated with Web Services message exchange is too long for a *hard* real-time application, then the implementing of Web Services in this automation system needs to be scoped accordingly. However, for the work undertaken in this thesis, further investigation of the specification of device communications and time constraints was carried out with the project collaborators, in order to identify the suitability of Web Services as a low-level message exchanging for typical automotive automation systems.

4.10 Conclusion

The conceptual framework of agile manufacturing has been presented and the manufacturing requirements in relation to the automotive domain have been addressed from a survey of a primary collaborative car manufacturer, Ford. In addition, the general needs of advanced control equipment and approaches to agile manufacturing have been identified and chosen as appropriate, based on the reviews and discussions of other researchers. Suitable technologies for the development of automation systems have been selected, based upon the measures associated with the industrial requirements and the needs of agile manufacturing. The constituent web-services technologies for automation systems required to support the requirements of agile manufacturing are illustrated in the Table below.

It is predicted that the development of Web Services, when combined with Ethernet networking and other developing programming applications at all levels of the manufacturing system (i.e. from business enterprises down to automation devices), can result in a new way of building and integrating automation systems to higher control levels.

The derivation of distributed control technologies and paradigms is summarised as follows:

Table 4-8: Summary of Distributed Enabling Technologies Appraisal

Distributed enabling technologies	A distributed paradigm		Middleware		Networking		A distributed application	
	SOA	OOA	SOAP	CORBA	Ethernet	Fieldbus	IEC 61499	CB design Tools
Industrial requirements								
A high degree of reusability (software/hardware)	√√	√	√√	√				√
A high degree of flexibility (software/hardware)	√		√					√
Seamless integration at enterprise level	√		√		√			
Non-proprietary control solutions	√		√		√			√
A distributed, heterogeneous control architecture	√√	√	√√	√	√		√	√√
An encapsulation and abstraction for re-configurability								√
Visual modelling and simulation prior to installation							√	√√
Integrated support capabilities and expert assistance	√		√		√			
Ease of installation and reconfiguration								√
Low development cost	?	?	?	?	√			
Data logging and Diagnosis	√√	√	√√	√	√			√
Research implementation	SOA		SOAP		Ethernet		CB design tools	

□ Not applicable to the industrial requirements

√√ Advantage choice over √ in the same category if both are marked

(Note that '□' does not imply it does not meet the industrial requirements, but that it is less suitable in comparison)

CHAPTER 5

Research Focus and Design

In this chapter, the problems and end-user requirements in the control system are outlined. The discussion will also focus on state of the art literature derived from relevant research works in automation platforms and business-shop floor integration. The research objective and area of work are identified with regards to the SOA's web services approach within distributed automation systems (*see Chapter 4*).

5.1 Problem Statement

The broad scope of the automotive manufacturing problem has been outlined and discussed in detail in Chapter 1-3. However, this chapter is focused on the narrower problem domain and set of requirements of the end user's (FORD Motor Company) manufacturing systems, in relation to their production strategy and management. The main research questions can be stated as follows:

1. What are the main obstacles that prevent the development and installation time of the automotive powertrain production system being achieved within the required time scale of 40-42 weeks?
2. Which state of the art agile manufacturing systems, implemented for real industrial manufacturing applications, could contribute to the research focus?

5.2 Problem Definition and End-User Requirements

The current global market intense competition and manufacturing trends towards mass customisation, driven by customer demand, were discussed in Chapter 2. These conditions have forced manufacturers to produce various types of products for the market in a shorter time-span. In the automotive sector, the usual development time for the production machinery of car engines (i.e. powertrain) is about 53 weeks. End users and competitive pressures are demanding that this be reduced to 40-42 weeks [104]. Traditional manufacturing systems cannot deliver the required time to market, for the reasons discussed in Chapter 2. The problems within the manufacturing domain, and particularly for the automation control system can be summarised as:

- Rigid control system structure with centralised control
- Lack of re-usability and re-configurability in the control system
- Experience based design
- Lack of remote support and diagnostic systems
- Late verification and change after system design
- Lack of process simulation

The requirements of next generation manufacturing systems to support agility concepts have been presented in Chapter 2. The key enablers for agile manufacturing systems focused on automation systems have been reviewed and discussed in term of technologies and limitations. The author has proposed the use of a Web Services-based automation system which could better support the development of agile manufacturing systems. However, the proposed solution also needs to encompass the industrial requirements from the end-users in the automotive industry i.e. at the FORD Company. The major industrial requirements that have been identified are:

- Remote expert assistance for fault diagnosis and troubleshooting / maintenance support
- Data logging and monitoring
- Tools to support machine reconfigurations
- Data collection for business planning / plant to enterprise integration
- User friendly machine systems, with simple visual aids for operators
- Early evaluation and validation of control system design
- Low cost of automation design and build
- Ease of maintenance and upgrade

The summary of problems, requirements and key agile enablers can be seen in Chapter 3- section 3.8.

There is a large amount of previous research, from both the academic and industrial perspectives, working on these requirements, such as the distributed control system, the component-based design approach, business-process application integration and the development of process engineering tools. The relevant background research is outlined in the following section.

5.3 Related Automation Research

The state of the art in the context of key enabling collaborative manufacturing systems, using object-orientated approaches, agent-based technologies and service-orientated architectures are outlined in this section. Ongoing projects are reviewed in detail to highlight major achievements and major obstacles. Work from consortia and organisations have provided a great deal of background knowledge in developing the manufacturing system for the future summarised in the following:

ITEA SIRENA (collaborative project with Schneider Electric Company) proposed a novel approach using Web Services, based on a SOA standard, to create an open, flexible and agile environment with “plug-and-play” connectivity. This project applied the XML-based Web Services paradigm for interconnecting distributed heterogeneous applications through Ethernet TCP/IP, which demonstrated the possibility of a universal, platform, and language-neutral connectivity. ITEA SIRENA proposed the idea of building advanced functionality, embedded into devices, to enable new distributed application paradigms based on self-reliant smart devices [9].

The **PABADIS** (Plant Automation Based on Distributed Systems) consortium was interested in developing a dynamic structured design of automation systems by implementing agent-based technologies to enhance the flexibility and re-configurability of business enterprises and production sites. The main contribution of PABADIS was centred on a methodology for establishing information streams between office level and field level systems by using mobile software agents as the communicators. The implementation framework was focused on developing loosely coupled agents in the distributed environment. Integrated Manufacturing Execution Systems (MES) were designed to facilitate the plant activities, composed of diverse control system technologies such as HMI's, remote monitoring and motion control. The PABADIS approach used XML message passing between plant agents and ERP functionality for sending and receiving manufacturing orders. In addition, the project has demonstrated a measure of flexibility at the automation level, with a simple lookup service developed on JINI middleware for “Plug-and-Play” device discovery. The full description of this approach can be found in [g14].

Tampere University in Finland, has continuously contributed to the development of agent-based distributed automation, remote configuration, and wireless communication in control systems mainly within the electronics-manufacturing domain. With regards to agent-based research, the group has designed modular structures of software agents and used generic XML formats for messages, in order to simplify automation system flexibility and reusability. Their design framework provides access to the data source used to create business processes (e.g. data analysis functionality and modeling), and the group's area of expertise falls within the using of wireless communication, bluetooth, WAP, and wireless LAN to support maintenance and remote machine diagnostic systems.

The **OMG** (Object Management Group) has been progressively working on Data Distributed Services (DDS) for Real-Time Systems Specification by developing modern software standards, including CORBA and UML. The new DSS standard [114] addresses the communication needs of real-time systems via a network middleware that allows computer programs to communicate and readily exchange information over the network, based on publish-subscribe technology. DDS achieves flexibility and precision through the pervasive use of Quality of Services (QoS) parameters, in which information flow between these nodes is specified. It has been suggested by [71] that DDS is well suited for heterogeneous networks, as it handles format conversion across operating systems, processor architectures and programming languages, hence supporting interoperability amongst different distributed enterprises. It also provides a state propagation model that allows nodes to update only when they change state in the global data space. The OMG middleware platform is CORBA, which includes the OMG IDL and the protocol IIOP, used for real-time systems with both large applications or small embedded systems from various vendors.

Rockwell Automation (RA) has focused on developing a flexible and reconfigurable distributed platform with plug-and-play automation systems, based on agent-based technologies [41, 51]. The agent-based approach is implemented with real-time control agents and information transfer (i.e. data from sensors, diagnostic subsystems,) between agents implemented on PLC's (ControlLogix™). The developed platform follows object-oriented principles, in which agents are responsible for local control of particular manufacturing equipment. These agents are

implemented in object-orientated languages like C++ and Java. The group developed the Autonomous Co-operative System (ACS) with C++ based agents that are executed on the PLC's. The agent management is responsible for registration and services look up, and ensures the transport of messages among agents. Recently, RA has considered Java as an alternative to C++, due to the portability of Java programs between different hardware platforms, operating systems and web-browsers [75]. This opens up the chance of integrating more applications on the platform, including real-time monitoring, process simulation, and device simulation.

The **MSI Research Institute** at Loughborough University has focused on the lifecycle support of distributed automation systems by replacing centralised PLC controllers with distributed control nodes (LonWork controllers) and a component-based (CB) design approach, where the control functionality is embedded into the component modules [123]. The finite state machines and CB design have been implemented and evaluated in real industrial automation systems, in order to create the design of generic and modular device components and determine industrial feedback on performance and capability [72]. The implemented distributed automation system has been conceived as a key approach towards an agile and responsive manufacturing system. The work has contributed to an improvement in flexibility, reusability and ease of use in the control domain. The COMPAG project has also contributed to next generation distributed automation systems in improving performance via improvements in visualization, remote support, diagnosis and HMI's.

Recently, the implementation of a service-orientated architecture (SOA) and Web Services in automation systems have gained attention, promising enhanced support for connectivity with high-level applications (e.g. for remote configuration and data acquisition). In addition, object-orientated architectures have been intensively researched with agent-based approaches and CORBA middleware, similar to the SOA deployment. However, the object-orientated approach has not had a significant impact in manufacturing, due to the complexity of its implementation and the diverse range of tools that need to be supported ([9], [118]). Novel service focused solutions are focused on the implementation of a SOA and Web Services with smart embedded devices and the Ethernet network (used to replace proprietary fieldbus networks).

In the author's opinion the success in the development of these approaches will greatly benefit industry, in that businesses and processing entities will become more integrated, adaptable and agile.

5.4 Research Objectives

The main objective of the research outlined in this thesis is the design, implementation, test and critical evaluation of conceptual framework of Web Services for the automotive domain. This framework has been proposed as the result of a survey of requirements at Ford Motor Company in order to identify the needs of end users of plant controllers and related business management applications.

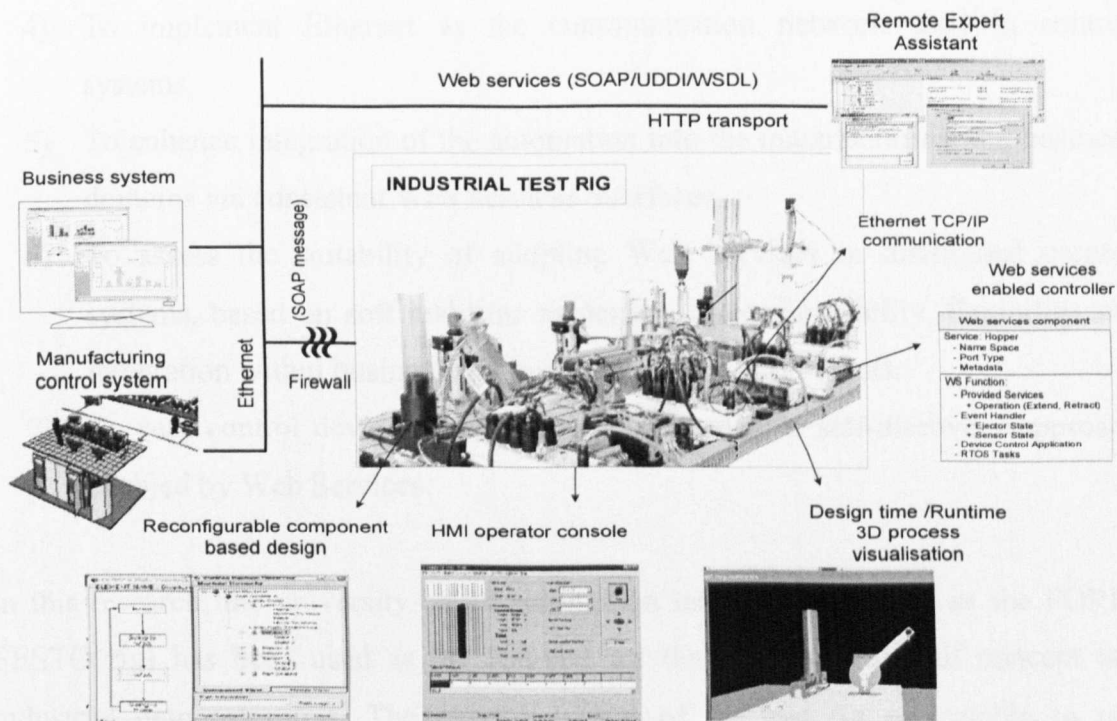


Figure 5-1: A Conceptual Model of the Test Rig used in this Thesis with Web Services

As depicted in Figure 5- 1, the focus of this research is the development of a web-services and component-based design methodology focused on an industrially specified portable test rig located at the university. The purpose of this is to facilitate:

(i) design, (ii) business and control application integration and (iii) lifecycle support of automation systems in the agile manufacturing business.

The supporting research objectives have been defined as follows:

- 1) To design, specify, implement and test a service-oriented architecture and Web Services within the design framework of component-based control methodologies.
- 2) To build a modular automation platform that effectively facilitates other areas of development, such as remote monitoring and support applications, data acquisition and process planning.
- 3) To implement Web Services on embedded devices. In order to provide the set of desired services, units needed to compose manufacturing tasks.
- 4) To implement Ethernet as the communication network used in control systems.
- 5) To enhance integration of the automation into the manufacturing and business domains via consistent Web Services interfaces.
- 6) To assess the suitability of adopting Web Services in distributed control systems, based on soft real-time responses, re-configurability, flexibility and integration within business and manufacturing control levels.
- 7) To ease control device installation through a device self-discovery approach enabled by Web Services.

In this research, the university based automation test rig (referred to as the FORD-FESTO rig) has been used as the test-bed for the research proof of concept and industrial demonstrations. The control system of the test rig is scalable to real machine applications, and is used by FORD to demonstrate the capability of novel control systems prior to implementation on real powertrain assembly machines. It enables a true demonstration of machine sequences and steps, with a machine controller for processing (i.e. transferring, buffering, checking position and drilling) vehicle parts. The result of this research on this rig is therefore considered applicable to real manufacturing applications.

5.4.1 The Area of Development and Novel Contributions

The general overview of the manufacturing system integration is shown in Figure 5-2 including a model framework implementing a SOA architecture with Web Services technologies for both the enterprise entities and real-time control device levels. This approach has the potential to have substantial benefits to many companies throughout the lifecycle of their production systems. In addition, the local control functionality and the device management information are collected and passed to the on-site diagnostics server, which mimics the activities of maintenance engineers in accurately monitoring, documenting, and analyzing the causes of machine breakdowns. All diagnostic information i.e. device types, error codes, device status and fault symptoms are stored in the local database and can be retrieved by local engineers or remote expert assistants for the purposes of proactive and reactive maintenance.

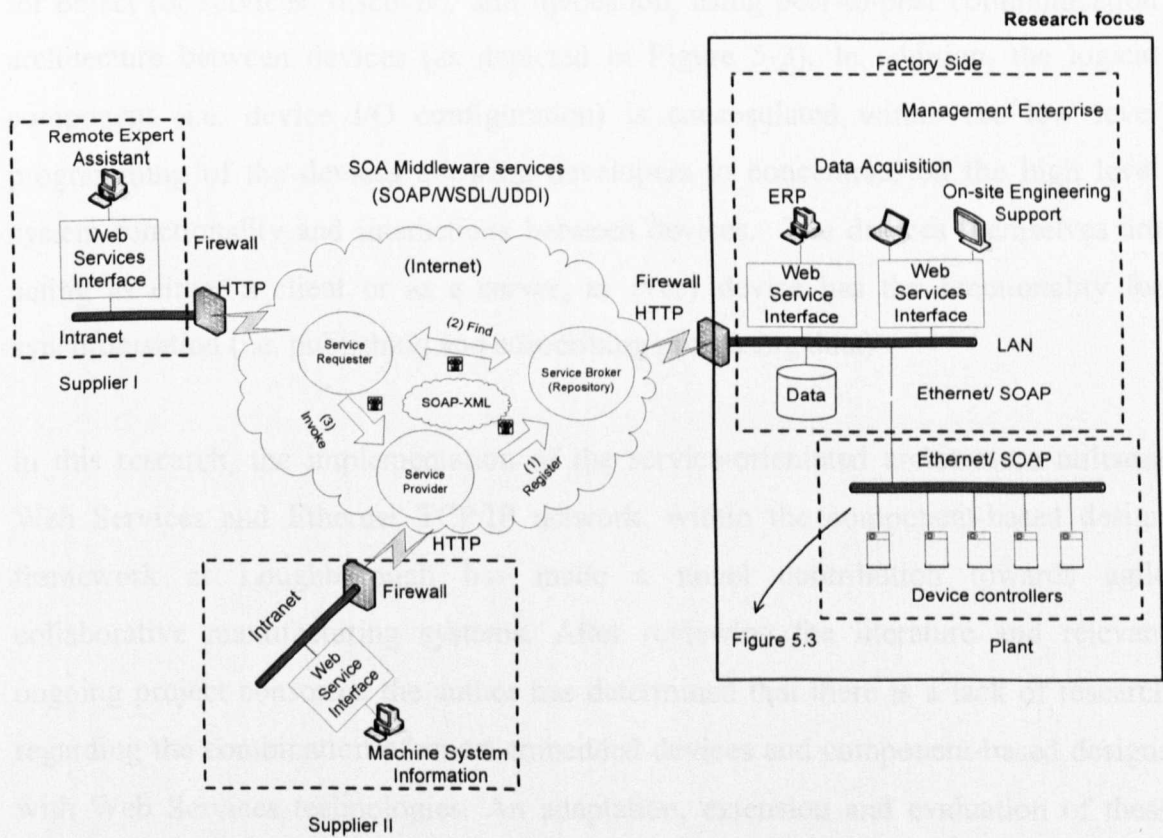


Figure 5-2: SOA-Web Services Integration Framework [adopted from 29]

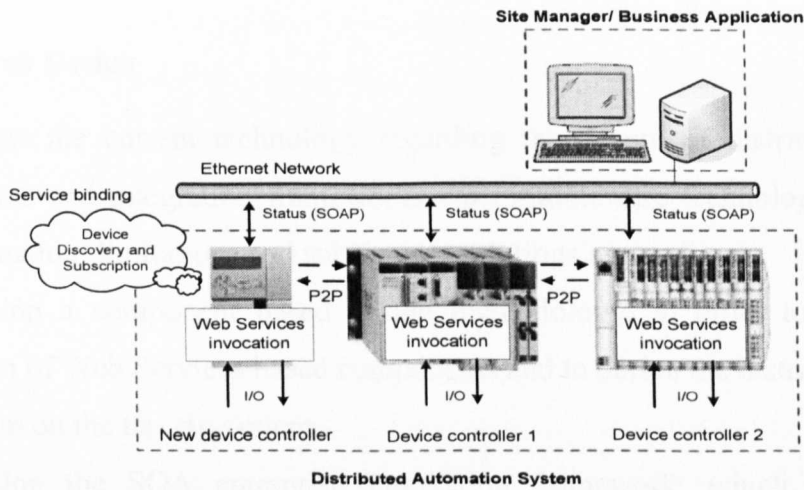


Figure 5-3: Peer-to-Peer Web Services Enabled Control System

Within this automation model, the Web Service codes are embedded into the devices for object (or services) discovery and invocation, using peer-to-peer communication architecture between devices (as depicted in Figure 5-3). In addition, the logical component (i.e. device I/O configuration) is encapsulated within the low level programming of the devices allowing developers to concentrate on the high level system functionality and interactions between devices. The devices themselves are acting as either a client or as a server, as every device has the functionality for synchronisation (i.e. publishing and subscribing processing data).

In this research, the implementation of the service-orientated architecture utilising Web Services and Ethernet TCP/IP network, within the component-based design framework at Loughborough has made a novel contribution towards agile collaborative manufacturing systems. After reviewing the literature and relevant ongoing project consortia, the author has determined that there is a lack of research regarding the combination of smart embedded devices and component-based designs with Web Services technologies. An adaptation, extension and evaluation of these technologies need to be further developed, as they are not mature enough to realise the full benefits in the domain of automation and business systems. To summarise the design of the research programme follows the structure outlined below:

5.4.2 Research Design

- 1) Review the current technology regarding the design of distributed control systems and integration frameworks with middleware technologies for agile automation: limitations and subsequent solutions are outlined.
- 2) Develop a component based design methodology, in order to support the design of Web Services based components and to derive the distributed control system on the test rig system.
- 3) Develop the SOA enterprise integration framework, which involves the adoption of WS in automation systems.
- 4) Commission the distributed control system test rig to conduct the experiment on the component based design.
- 5) Outline control system specification and implement DPWS, RTOS and TCP/IP stack on the embedded device, as per the industrial case study.
- 6) Conduct testing on control system performance regarding real time response and cycle time, ease of system design, application integration and changes to control system re-configurability. All of these evaluations are compared with the standard commercial PLC system.

5.5 Conclusion

The general problems and requirements of manufacturing systems have been derived from a review of the literature and previous MSI research group's case studies at the Ford Motor Company. The state of the art in distributed automation systems and enterprise integration within this domain has been studied in order to determine a position on the key enabling technologies and limitations of implementing approaches.

The concept of a Web Services- based framework, capable of connecting various heterogeneous platforms and diverse equipments so that they may be integrated into a unified system and interact in a co-operative way, has been outlined in this chapter. The concept of utilising the Web Services protocol stack offers the potential for manufacturing automation *to evolve*, enabling a new paradigm of open standard, technology neutral and interoperability components from various device vendors. The

development of device descriptions, embedded into the component and the driving of system intelligence down to the device level, ultimately offers the potential to eliminate the need for system integrators to undertake low level programming. The focus is on shifted towards building higher-level control applications and improving efficiency.

It is a novel contribution by the author in proposing Web Services on the CB automation system using standard embedded microprocessor controllers to effectively improve manufacturing system agility (*presented in Chapter 3*). The aim of this work is also to investigate the configurability, and re-usability of control systems and seamless integration to business levels, thus enabling companies to become more agile and collaborative.

CHAPTER 6

A Web Services Component-Based Automation Design

This chapter is divided into two parts:

The concept of the component-based (CB) software design methodology in support of the COMPAG framework is discussed in **Part I**. The role of machine users, builders and component suppliers is defined, based upon the design of reconfigurable automation systems and the design methodology of a CB automation system for the power train assembly machine is presented. The design is considered in accordance with the needs and required performance of automation systems from an end user perspective and based upon a requirements study with the Ford Motor Company Ltd. In order to utilise Web Services-based component interaction the component-based (CB) design approach of previous research needs to be redefined.

The concept of enterprise integration in a heterogeneous environment using Web Services is explored in **Part II**. The key function of Web Services technology is captured and adopted for building industrial control systems including control application integration. The selected Web Services toolkit provides a lightweight code generation technique, designed for C/C++ embedded microcontrollers to enable XML Web Services-based device communication and service discovery to be embedded in industrial control systems.

6.1 Problem Statement

This research is focused on the modular design of the control system, enabled by the component-based (CB) design approach to allow reconfigurable and reusable automation platforms, as one of the key requirements of agile automation (*Chapter 3- section 3.2*). The CB approach has previously been researched at MSI (via the COMPAG project), and the objective of this research is to apply Web Services to the design of CB automation, in order to enhance the degree of modularity and integration capability of the automation devices. The research questions relating to the adoption of Web Services within CB automation design are as follows:

1. What is the design framework of a component-based approach to enable reusable and reconfigurable automation systems? How do associated parties (i.e. machine builders, component builders and end-users) interact to undertake component design and reconfiguration?
2. How are modular designs achieved within the CB approach for automation devices?
3. What are the issues with applying Web Services in the automation domain?
4. What is the most effective way of mapping the WS to component functionality?

Part I- Distributed CB Automation Systems

Modularity is typically introduced into a manufacturing system to increase flexibility, both in terms of functionality and also provide an ability to be easily reconfigured [115]. Within this context, modularity is focused on the intelligent, autonomous and loosely coupled entities that are distributed throughout the subsystem. Modularity concepts can be found in many related areas in manufacturing, for example: reconfigurable manufacturing systems, agent-based manufacturing systems and holonic manufacturing systems. All these have adopted the modular design approach, particularly within automation systems, to increase the responsiveness of the manufacturing process to both external and internal disruption.

A number of researchers, (see [72], [95], [100], and [115]), have developed modular manufacturing production approaches, focusing on rapidly adaptable and reusable machine systems to support their lifecycle needs. The automation system is dissected into a set of mechatronic modules for the intended application domain. This approach aids the modular decomposition / composition of the control system. In contrast to proprietary object-orientated programming paradigms, this simplistic integration approach is effective in supporting distributed application development, particularly suitable when composing a set of distributed control functionalities to match the required physical modularity of machines [72]. The basic concepts behind the CB software development process are outline in the next section.

6.2 Component-Based (CB) System Development

The basic idea in component based software development is to structure a desired system around *components*, a well-defined *component framework*, *interfaces* and an appropriate *contract* to ensure proper system construction and operation. It is reported by [110] that the software component-based design contributes to increases in software productivity by reducing the amount of effort needed to develop, update and maintain systems. Regarding development of control applications in this research, the component-based software engineering (CBSE) provides the development platform that facilitates an evolving automation system during the manufacturing lifecycle.

In the following section, the framework and component software interfaces are outlined, in order to enable system integration. The objective is to enable the development of reusable control applications, considering not only the creation of components but also the lifecycle management of such reusable software units as part of an evolving automation system.

6.2.1 Component-based Construction Principles

In order to build software application systems from subsystems/components, such components or subsystems must be integrated through well-defined infrastructures. This infrastructure incorporates components from different sources to form the required system [33]. The materials used in the construction of a component-based design can vary widely in character, but may be classified into four main categories (i.e. Component, Component framework, Interface and Contract) based on [g36] and [107] and shown in Figure 6-1.

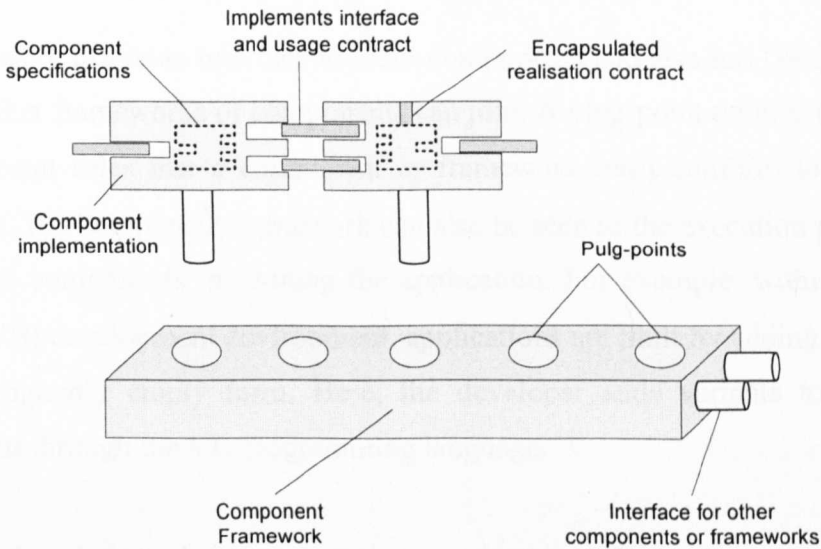


Figure 6-1: The Component-based Design Principle

A *component* is a piece of software or software design with a well-defined interface and hidden internals. The component is based on a concept that is recognisable and of value for its user which may be another component, a software system or a human user [g36]. It provides specific services to its environment across interfaces. In the software engineering discipline, a component is a self-contained part or subsystem that can be used as a reusable building block in the design of a larger system or so-called “construction”. A component may be integrated with other components or users through its interface, which contains e.g. services, attributes, events and times to show what the component can deliver [110]. In component-based software engineering, a component package may contain lists of provided and required interfaces, executable codes, validation codes and design documentation.

A *component framework* is a pre-built assembly platform of components, together with the “logical glue” that binds them together. The framework is designed to be extended. Frameworks are also defined as units for sharing and reusing architectures i.e. a framework can be viewed as the reusable component and also the platform for component integration.

A framework offers an interface to other components and also has “plug-points”, at which other frameworks or components can join. A plug-point defines the roles and development rules that a component or framework must conform to in order to integrate. The component framework can also be seen as the execution platform that facilitates components in running the application. For example, within the Visual Basic (VB) development environment, applications are built by adding components to an originally empty form. Here, the developer adds variants to component behaviour through the VB programming language.

An *interface* defines the access points to components. To be precise, an interface is a collection of operations used to specify the services of a component, and these operations, or actions, are defined through a set of software codes that can communicate with each other. Components can export or offer one or more interfaces to other components, which use or import these interfaces. The component that *offers* an interface is responsible for realising the action of that interface, while the component that *uses* the action of an interface only needs to know what the action achieves, not how it is achieved. In addition, a component interface consists of a signature part describing the name of operations, together with the parameters and types provided by a component and a behaviour part that describes the components' behaviour [97, 98].

A *contract* is a construct for explicitly specifying interaction among objects. Contracts formalise these collaborations and behaviour relationships. In component-based software design, the contract can be classified into two categories [97]: the first type of contract is a *realisation contract*, used during the design of the component to describe the component specifications such as quality of service, functionality, interaction methods and behaviours. This helps developers understand components, in order to use them. The second type of contract is a *usage contract*. Once the implementation of component specifications has been done, the usage contract defines the interaction amongst components during run time execution.

For the definition of these component software elements in the context of automation systems, this research maps component software functionalities into automation terms used by the control system. The component is considered as an encapsulated unit of machine operation software, which contains the operational interface, the contract, and the low level programming elements for actuator and sensor tasks. The control algorithms are implemented in programming languages, such as IEC 61133 (Function block diagram, structure text for example), C/C++, or JAVA, for the execution of I/O devices. For the purpose of reusability as well as re-configurability, the component needs to be implemented in a generic manner, so that it may be reused for building other components without changing the internal control algorithms. A major benefit of the component-based approach is that the developer does not need to have specific knowledge or experience in dealing with an encapsulated low-level device program. Rather, a knowledge of altering the behavior of the system through well-defined interfaces and contracts, as illustrated in Figure 6-2, is employed.

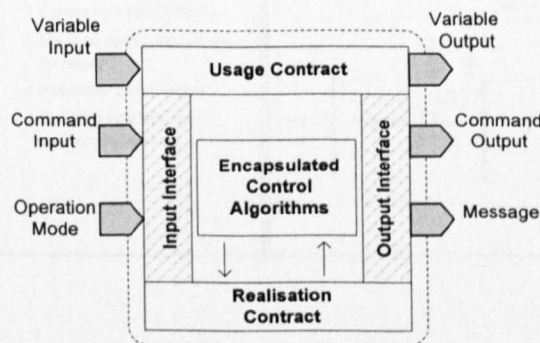


Figure 6-2: Automation Software Component Construction

In the control system, interaction between components is achieved through the interlocking of the devices' state variables, in terms of the input and output interfaces (see Figure 6-2). The interaction in this case is described by the usage contract. In addition, the realisation contract is the unit used to define component behaviors and the quality of device services (e.g. device parameters, I/O trigger delay time, memory registers and task priorities). In the design of the control application on the component,

device control logic in terms of I/O operations and state sequences are encapsulated and exposed to component users through the component interface.

Based on the CBSE construction in the control system, this research proposes the platform of the component integration to embody the novel development of the component-based design approach for automation systems. As illustrated in Figure 6-3, the component-based design framework in the automation domain shows the required component functionalities that aim to facilitate the design and integration of reconfigurable machine applications.

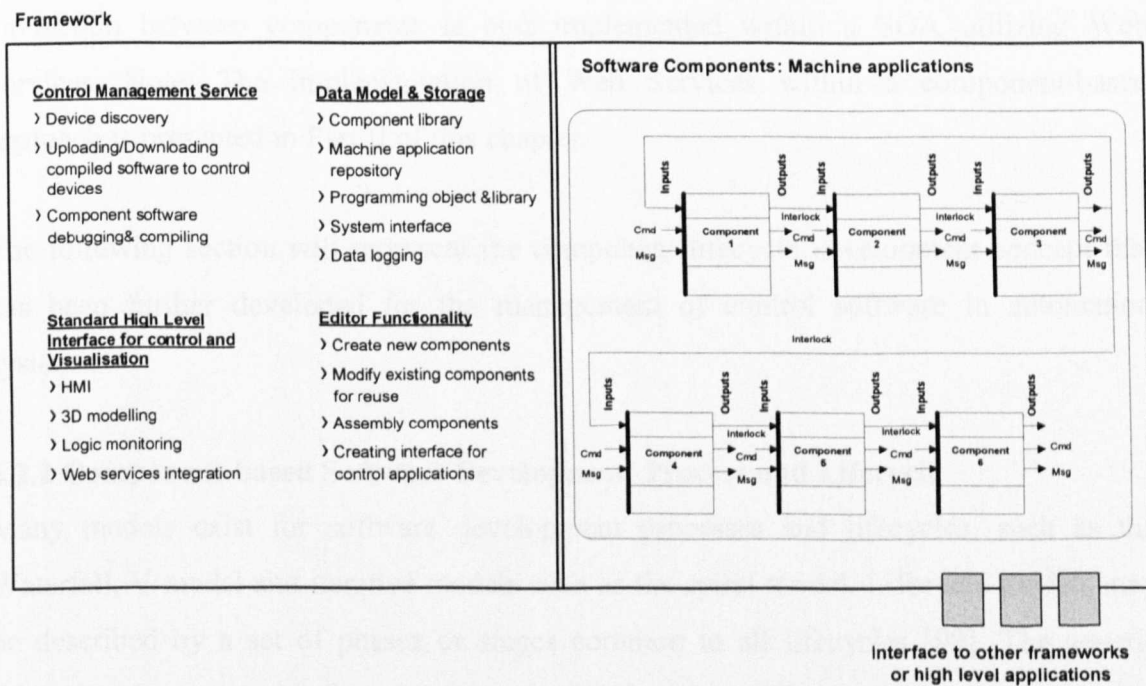


Figure 6-3: The Component-based Design Framework for Automation Systems

The CB design framework provides components with the platform to enable process editor functionality, data storage for system data logging and the component library, an interface to support manufacturing applications and control component design & build. Within this framework, the component developer is able to create the new component (software) and store it in the containment library for future (re)use. The control components for device operations, as well as the supported application interfaces (e.g.

HMI and Web Services integration), are debugged and downloaded to the target device by the component user or the control system integrator. The control system integrator is only concerned with building the control application through component interfaces that allow the integration of components with other supported applications, such as 3D modeling visualisation, HMI, and control configuration data for machine operations. This machine application, built by the system integrator, can be saved and reused for later changes in new machine configurations. The engineering roles associated with this design and use of the component are detailed in section 6.2.2.

The aim of this research is to further develop the CB approach and to define how such interaction between components is best implemented within a SOA utilizing Web Services. Note: The implementation of Web Services within a component-based approach is presented in Part II of this chapter.

The following section will represent the component lifecycle development concept that has been further developed for the management of control software in automation systems.

6.2.2 Component-based Software Development Process and Lifecycle

Many models exist for software development processes and lifecycles, such as the Waterfall, V model and iterative models such as the spiral model. Lifecycle models may be described by a set of phases or stages common to all lifecycles [99]. The generic lifecycle of a component-based system can be shown at different phases, as depicted in Figure 6-4.

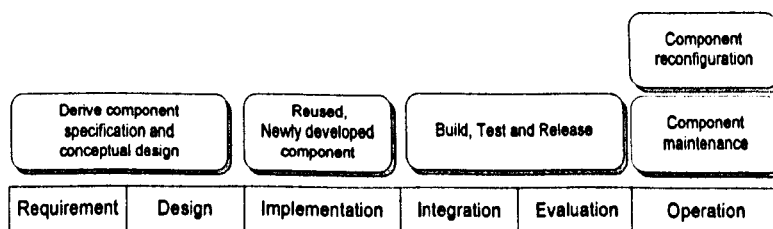


Figure 6-4: A Generic Component-based Lifecycle (adopted from [9])

Frequently, in the initial development phase, organisations perform the same activities in each evolution cycle. Thus, an existing software product will evolve into the next version by repeating the same sequence of phases. Regarding the process of component implementation, existing software components may be reused to build new components. These will be stored in a common data storage area, where they can be retrieved for future use.

The relationship between component and control system design is illustrated in Figure 6-5. A major development of the CB control system involves the retrieval of components for system integration. This process follows a similar procedure but deals with more integration units (e.g. hardware, HMI, high level engineering tool editor). Since the components (i.e. device control software) are separated from other units such as mechanical units, electrical wiring and I/O devices and controller modules, the process of building the component and the control system progresses in a concurrent and independent manner.

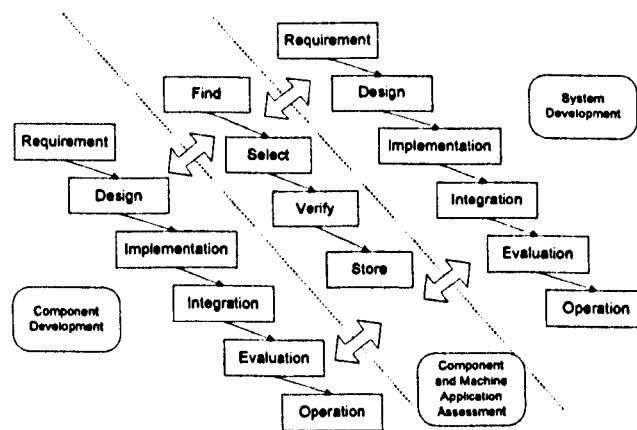


Figure 6-5: The Parallel Process of Component-based Development [99]

This parallel process development [9], allows component design and system development to be carried out independently of each other. A new component may be built a adapting a similar component, if available, and then adding to the component repository. This

component database system bridges the process in which components are subjected to searching and verifying, in the initiation of the system development process. Likewise, the component being developed will be provided with the based component, as related to requirements, by searching the component repository.

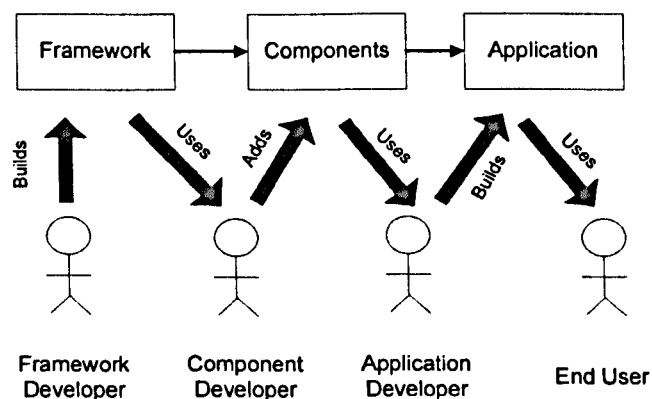


Figure 6-6: Various Developers and End User Roles in a Component-based Framework
(adopted from [107])

To support the concurrent design of component-based applications, all developers and users can access the component assessment to search for and build components and systems related to their roles and requirements as shown in the basic model in Figure 6-6. The framework is provided by component builder tools. Component developers then build new components and load them into the component project. The application developer then customises these components, in order to produce a custom application as expected by the end user. In this model, the CB application is developed concurrently with new components (and the framework if required by the application / control system integrator) and the complexity of the software component is hidden from the application developer who only integrates these components to build the control system through a well-defined component interface. In addition, the component developer is able to reuse component code via constructs of the component development language, such as inheritance, or templates provided by component building tools to develop new components. The application developer employs the framework's meta-data mechanisms,

provided by process engineering tools, to reuse components for building new machine applications (*for further discussion, see section 6.3*).

6.2.3 Component-based Development Design Issues

In adopting component-based technology for manufacturing automation systems, careful consideration needs to be given to the industrial systems requirements e.g. real time performance, safety issues and operating environment.

In addition, the design of current component-based technologies for distributed automation systems needs to be considered in the light of the constraints on automation devices, particularly small embedded devices which could have limited processing unit capacity and restrictions on communication bandwidth and memory space. To resolve these issues in the context of automation systems, the concept of component-based software needs to be redefined to fit into the automation domain, enabling real-time system engineering technologies.

In the following section, a hierarchical structure for mechatronic systems is addressed, and the development of component-based automation systems using the methodology of composing encapsulated software components to form the machine operations is also outlined.

6.3 A Component-Based Design for Manufacturing Systems

The pervasive adoption of component-based software has benefited developers and users by reducing development costs, system maintenance provision and the time taken to market and the realisation of more reusable systems. However, in the automation domain, the adoption of component-based approaches has been limited by the commonly-used technology in the domain. This technology is largely based on ad-hoc software design and manual coding techniques for control applications and has resulted in very limited reuse. Additionally the process is extremely time consuming.

Previous research [33] on the COMPAG project at the MSI Research Institute proposed the implementation of the reusable and reconfigurable control component (software) for automation devices (*see Chapter 4- section 4.6.3*). It is important to note that the adoption of Web Services to the automation domain inherits the proposed CB approach for the design of control applications.

6.3.1 Encapsulated Industrial Component Based Systems

In CB manufacturing systems, machine functions are defined from prefabricated components with known and validated properties. The system integrator does not need to have any knowledge of the internal design and implementation of the components.

Each component in the automation system needs to possess functionalities to support the production system effectively. The component functionality for the development of control systems in this research has been established: (i) in response to the user requirements obtained from the Ford Motor Company Ltd. (*see section 6.4*), and (ii) published academic papers regarding the design of component-based automation systems [95]. Components in automation systems can be viewed as:

- **A Unit of Service Provision.** A component encapsulates its manufacturing function, defined during the component design process, and this service functionality can be accessed through well-defined interfaces. The user does not know the internal implementation, only the service it provides. For instance, a

“feeder” actuator component could have only two encapsulated services: extend and retract.

- **A Unit for Validation.** The operation of the component can be properly evaluated prior to deployment. For example, simulation of the component in the control application may be viewed with a virtual modelling tool. In this case, the component needs to provide the modelling tool with the appropriate service i.e. providing operating information (i.e. the device state information) and its interface.

- **A Unit of Error Containment.** All errors that occur inside a component must be detected before the consequences of these errors propagate to other components in the system [14]. Component developers must implement the unit of codes needed to detect and handle the error internally.

- **A Unit of Reuse.** As a result of the generic design of component functionality, other components in the system that provide the same service/function can re-use generic function codes. For instance, in industrial programming (i.e. IEC 61131), the control function block of one unit that provides simple extend and retract functions may be used for another actuator component, as long as it is defined by the same functionality.

- **A Unit of Design and Maintenance.** Machine system design comprises sub-systems built from components that are independent of each other. Hence the upgrade and maintenance of separate components in the sub-system has no direct impact on the others. The management of the machine lifecycle is enhanced through the reduction of time needed to deploy the system.

The implementation of these components is detailed later. In order to support the development of a component-based approach for industrial control systems, the framework supporting CB integration has been studied to identify the tasks of each stakeholder in building and integrating encapsulated components. Each individual

stakeholder, such as the end-user, machine builder and component supplier, has their own distinct role in the development of CB manufacturing systems, as shown in Figure 6-7.

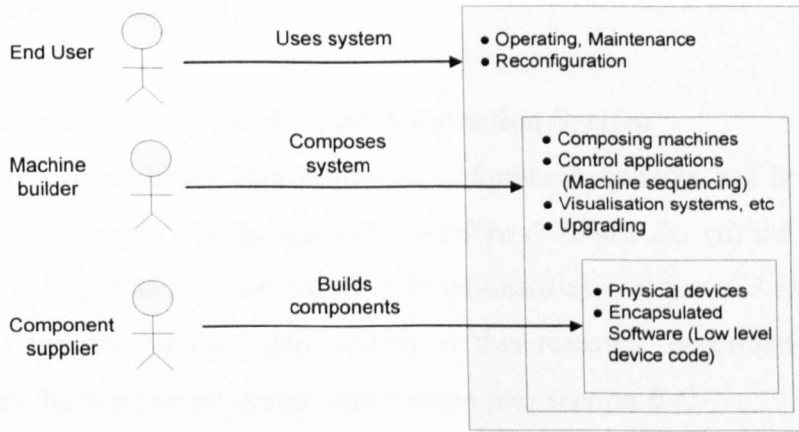


Figure 6-7: Stakeholder Roles Constituted in the COMPAG Project Framework

In the component-based design automation system, *component suppliers* (i.e. the component developer), supply automation parts to the machine builder and may be involved in training system developers on how to integrate components for required control functionality. Component suppliers develop the encapsulated control functionality and the low-level programming required by system integrators. *Machine builders* perform the role of system integrator. Instead of developing a component from scratch, the focus on developing applications for automation systems by configuring components and logically coupling them, in order to create the required control functionality for the end-user. The *End-user* operates, monitors and maintains the system. In the lifecycle management of the control system, upgrading may be performed by machine builders or component developers, without any effects on the integrity of the system.

The implementation of the CB approach to support stakeholder roles, as described in this section, has recently been developed for the FORD- FESTO test-rig assembly machine at Loughborough University. The development of the components for the test-rig considered the roles of component supplier, the machine builder and the end-user, and the generic component function blocks (as done by the component supplier) for each control

unit were developed. The components were later integrated to complete the automation system, including the HMI (as specified by the machine builder). The end user operates the system, as well as changing machine configurations. Details of the development are presented in Chapter 7- case study 1.

6.3.2 Specification of a Component-based Automation System

The scope of component-based approaches for automation systems has been identified for: (i) the design concepts and management (*sections 6.2.1-6.2.2*), (ii) the development issues (*section 6.2.3*), and (iii) the required functionalities (*section 6.3.1*). The design specification of the component-based system in this research, described later in this chapter, includes the component design information (*see section 6.4*).

Machine systems are decomposed into subsystems, which are composed of modules (components) and organised hierarchically when adopting the CB architecture. Each module can be seen as a group of components, which contain elements at the atomic level that describe the state of the manufacturing environment (i.e. sensors/ actuators).

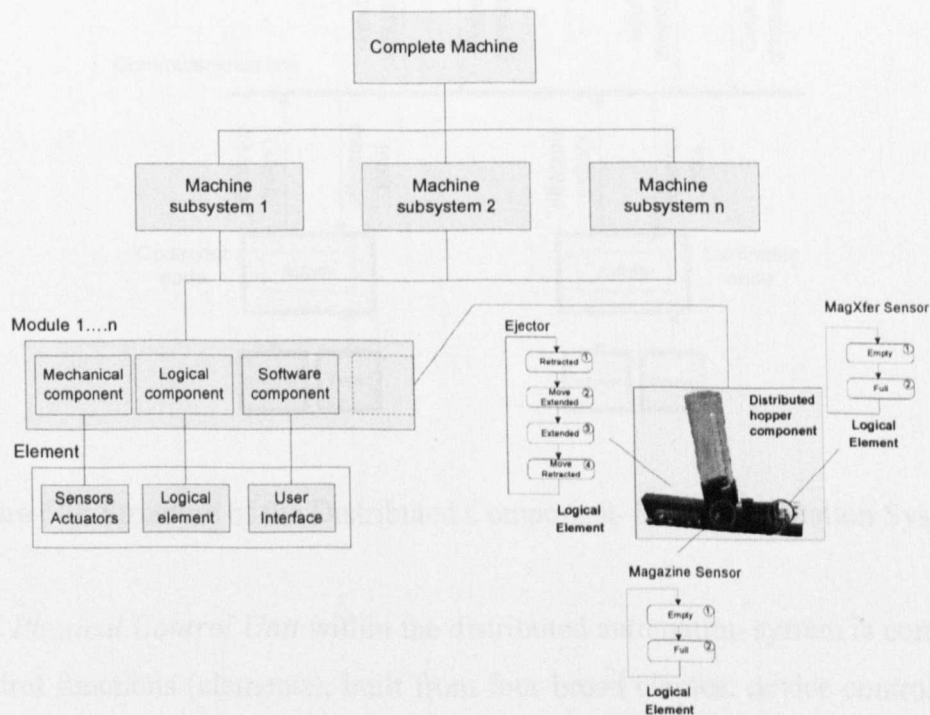


Figure 6-8: Component-based Machine Control Hierarchy

In this research each module comprises mechanical components, electrical components, logical control components, software components (e.g. 3D VRML modelling component) and other ancillary components relating to the machine system, as shown in Figure 6-8. In addition, the integration and configuration of these components is achieved with the Process Definition Editor (PDE), part of an engineering toolkit which manages the lifecycle of control systems.

In the design of component-based manufacturing systems with distributed control devices, an automation system will be composed of three main units:

- 1) The *physical control unit*,
- 2) The *control activity* and
- 3) The *control function block*.

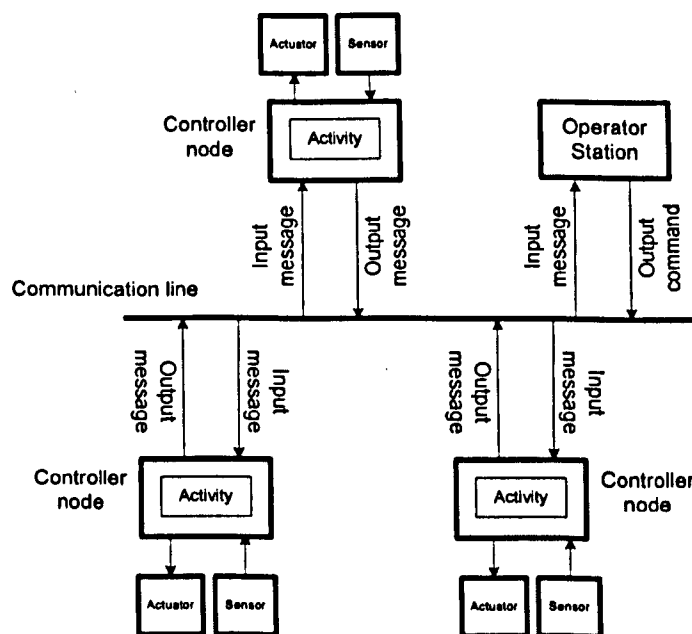


Figure 6-9: Structure of the Distributed Component-based Automation System

The *Physical Control Unit* within the distributed automation system is composed of control functions (elements), built from four broad classes: device control, process control, operator control station and service control (machine safe guard), as depicted in Figure 6-9.

All elements within the control function unit interact by exchanging messages as state variables (i.e. state transitions, positions, pressure) within various types of distributed transactions, such as producer-consumer and client-server interactions. Signals or state variables are exchanged through a suitable distributed communication protocol, typically achieved via Ethernet or a suitable fieldbus.

The *Control Activity* is the defined activity that reacts to internal and external events, in order to trigger such events. External events, such as input to the activity unit are obtained from signals provided by sensors. The control function activity, as shown in Figure 6-10/a, encapsulates the formal rules that define the output reactions by invoking sets of low-level programmed function blocks.

In the design of discrete automation systems, the behaviour of the component is defined by a set of transformational rules, invoked in response to the changes of state of corresponding event-driven mechanisms within finite state machines.

The *Control Function Block*, shown in Figure 6-10/b, contains information such as input variables, output variables (through state variables) and internal variables. Function blocks are used to specify the properties of a user defined function, such as START(), STOP(), EXTEND() and RETRACT(), in a generic way that may be used by other components. To form the state behavior of each component, function blocks need to be connected in sequence, to define the relation of the input/output state variables. This is normally done via variable reference (e.g. ejector_retracted allocated to %mw110.1 in the PLC memory) in the application tool, using pointers to allocate the corresponding variables to the memory addresses at the time of design.

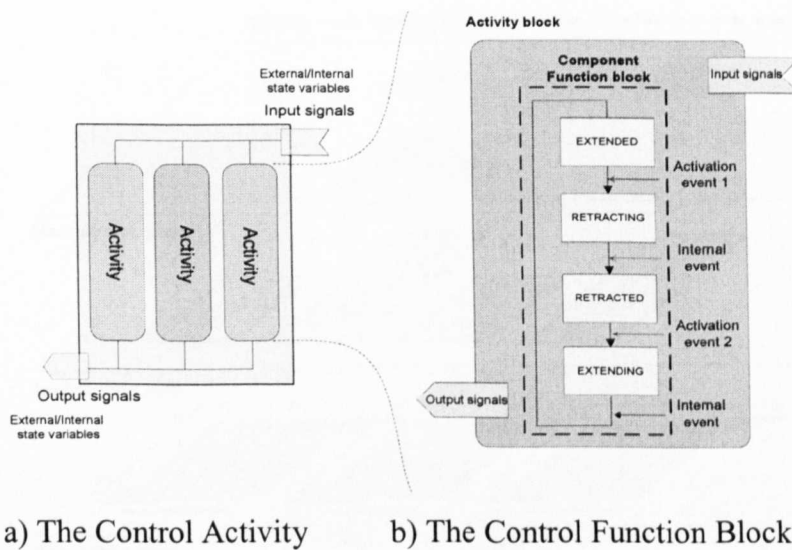


Figure 6-10: The State Transition Diagram in the Activity Function Block

6.3.3 The CB Control Design Model

In the design of the component-based automation system, components do not only exist as software functions in the development environment. They also form an integrated part of the mechatronic deployed system, where each component has a software-based logical implementation and a physical integration unit. Component implementation is encapsulated through a set of control elements that expose the abstracted control behaviour through state transition diagrams. The interlocking nature of a component may be implemented via this state transition diagram, in order to define control behaviour for the system. The interlocks are designed and tested with the debugging tool and the high-level logic simulation during design time. The control applications are then downloaded to the runtime component.

In the design of component-based automation systems, a design pattern is categorised into four different layers: (i) mechatronic layer, (ii) application layer, (iii) resource layer, and (iv) mechanical process layer. Details of each layer are based on the composition and specification of mechatronic modules, as depicted in Figure 6-11.

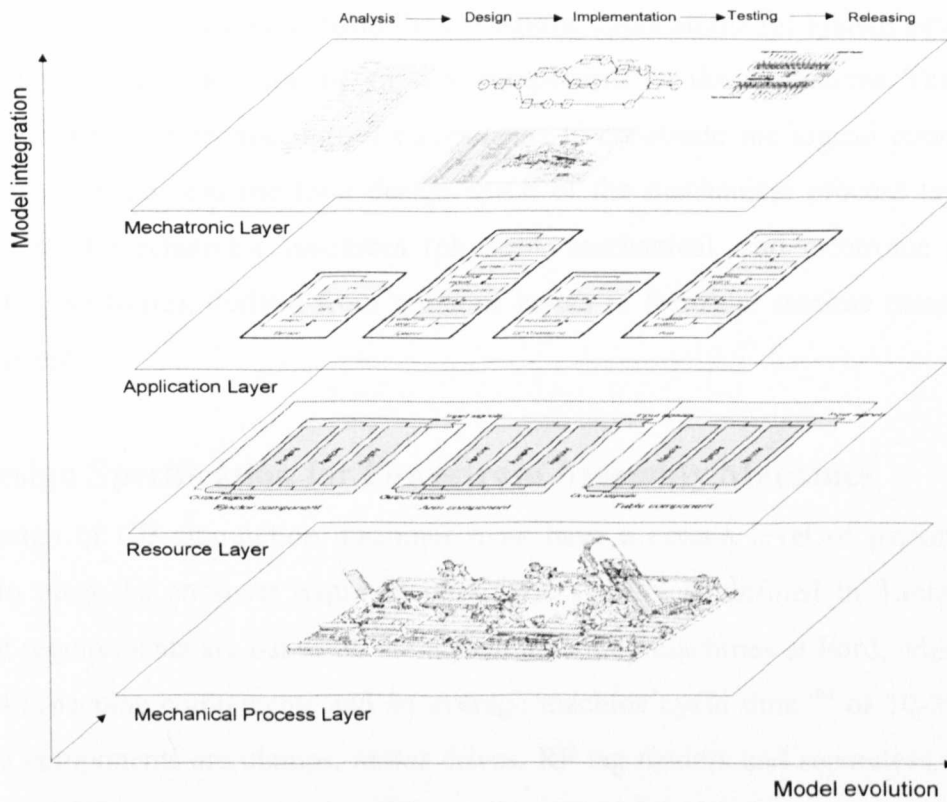


Figure 6-11: A Component-based Design Model

At the top level, the mechatronic-layer is projected onto the three sub-layers: the system application, the component resource and the mechanical process layer. In the design of mechatronic systems machine subsystems are derived from the units of electrical, mechanical and software components. These units, formed in the mechatronic-layer, are derived from customer requirements (interpreted by product processing, cycle charts, machine operations and specifications) and then mapped onto the required machine operation.

The application layer is used to form the controlling applications for system development by building the state transition diagrams of components for the subsystems. The resource layer defines the interconnection of components to constitute the logical connection of the control system, and the final design space of the mechanical process layer is the integration of mechatronic resources (physical, mechanical, and electronic parts and controlling software), derived from previous layers to form the runtime manufacturing environment.

6.4 Design Specification for Powertrain Assembly Machines

The design of CB automation machines must have a certain level of performance, in order to meet the end-user requirements of the system, as defined in Table 6-1. The derived requirements are based on the typical assembly machines at Ford, which contain 10 to 40 machine components and an average machine cycle time ⁽⁶⁾ of 30-35 seconds. Typical components are clamps, motor drives, RF tag readers and separators. Individual component I/O response time (*see Chapter 9- section 9.2.3*) needs to meet soft real-time criteria under 30 ms. However, it may be noted that the design of a CB system in this research is based on the embedded device, which needs to meet these machine timing specifications as well as the additional requirements from the end users (e.g. process description, reuse, reconfiguration, pre-programming, configurability, intelligence, ease of build / use, high level representations, unified interfaces, cheaper than competition, standardised technologies, effective solutions for reconfiguration, agility) detailed in Table 6.1.

(6)- An average product processing time in each machining process derived from time of the input to the next output stage

Table 6-1: FORD Requirements of the Component- based Automation System

End-user requirements	CB specification	Description
Process description and reuse of components to build and (re)configure machining systems	Supports well defined machine components and finite state transitions	In the event-based automation system, the component operation desires to operate on the changing state of other components defined in the device interlock during design time. Building of machine applications from well defined components helps integrators directly relate to the previous design and resource for reuse.
Meeting the target production cycle-time and response time to supported production system	Real-time communication	The response time of I/O devices needs to meet the soft real time requirements and hard real time conditions in critical tasks, such as safety and alarm systems. The implementation of RTOS (kernel OS) at the device required for the CB system.
Support for pre-programming and configurability of system modules	Hide the complexity of low level coding	Low- level program of devices is pre-fabricated and encapsulated in the hardware components, ready to be integrated by the application builder and end users. They can then focus on building and reconfiguring automation from high- level control applications.
Smart component modules for immediate and automated warning system	Self error-diagnostic	Diagnostic error checking routine need to be designed and pre-built in the device by the component builder for the end user. The error diagnostic routine helps the user in the automated monitoring of the device for active maintenance.

Design simplicity, ease of installation and use	Device discovery and initialisation	Automatic device discovery and installation to ease device installation within the system. The CB development platform needs to support the device discovery and dynamically allocates the port location for the device. There is no requirement to amend the hard code device programme for manual installation.
High level process description with unified application interface	Exposure of abstract device functionality	Ease of integrating the manufacturing system. The components will provide basic instructions on how to activate the interface for device communication and interaction. It does not require custom interfaces for every integrated application.
Cost equivalent to or cheaper than the current control devices	Embedded controller design with light-weight memory consumption and CPU resource	The compiled binary code of the control application, including RTOS and protocol stack, should be compact for the embedded device. The code should scale effectively in kByte rather than MByte , due to limited memory side of devices.
Standardised technologies and interface for heterogeneous devices	A unified and language - neutral framework	The component based design needs to support the interoperable on various devices. Low- level component coding is in the form of chosen devices, but will need to support interoperability and interaction among different devices through a unified interface (Web Services interface).

<p>Effective solution to build and re-configure automation systems in the lifecycle, taking into consideration cheap costs, short development time, and early error detection</p>	<p>High level machine configuration using Process Engineering Editor Tools that support virtual engineering and control logic simulation</p>	<p>In the design of machine applications and operations, it is foreseen that the current automation integration tools are not the most easy to use in current industrial practices. Control engineers are still dealing with the logic, sequencing and timing of components at the base level of programming, and a more user-friendly tool should be used in the presence of complexity in the commissioning production process. Editor engineering tools in compliance with the CB design are needed, which are capable of managing complexity during their lifecycle.</p>
<p>High degree of agility for system reconfiguration</p>	<p>Loosely coupled among other components</p>	<p>Regarding system reconfiguration, changing one component in the software or hardware units will not affect other components. It is desirable to have the device coupling defined through a more flexible device interlocking approach.</p>

6.5 The CB System Model

A UML class diagram of the CB system model as defined for this research is shown in Figure 6-12. The diagram illustrates the activity of component actors operating in a design time and runtime environment. These component actors are the required modules needed to compose a flexible and reconfigurable automation system.

Each component has been implemented to support a lookup service for automatic device discovery and initialisation, in association with the TCP/IPv4 port connection during the design time. The component functionality, programmed in native code (i.e. C/C++ is the language used in this project due to the wide support on embedded hardware), includes each devices ID (i.e. TCP/IP and MAC address), execution commands, I/O port

specification and operating interface in order to work with other actors. During design time, the operation logic of the component (i.e. machine operation sequences, device interlocking and contract parameters) is achieved with a high-level graphical drag-and-drop tool. The operational logic then downloaded to the target component through an I/O mapping interface.

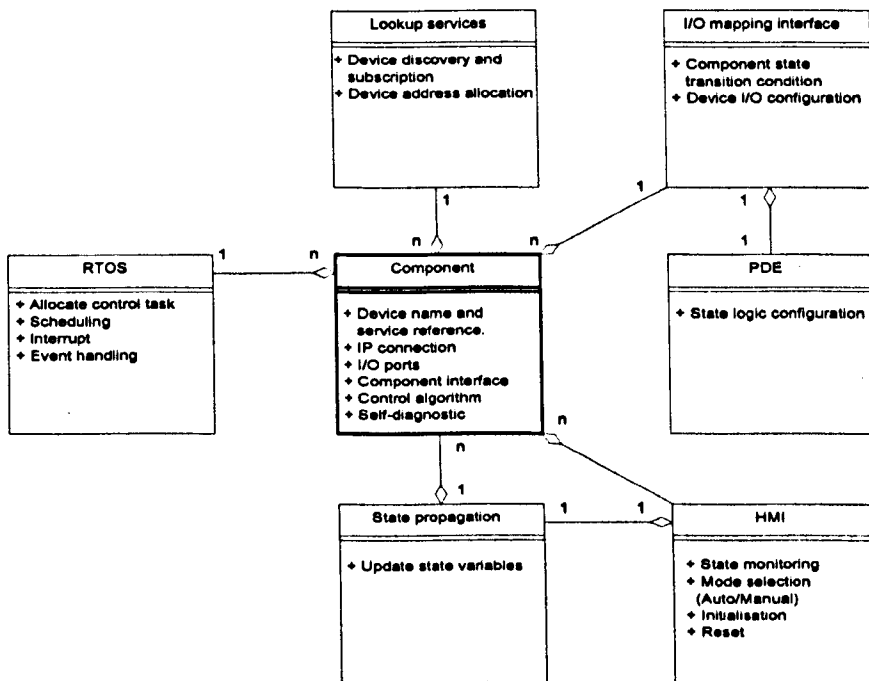


Figure 6-12: UML Class Diagram of the Component-based System Design

During runtime operation, the operator console controls the operating mode (i.e. auto/manual), initialisation and resetting of the system. In the design of an event-based asynchronous system, the RTOS is implemented in the embedded device at the kernel level in order to provide task eventing, interruption and scheduling. It should be noted that the use of scan-based mechanisms (such as PLC based automation systems) has proved inefficient, due to high memory consumption and a lack of CPU resources on the small device. The RTOS is utilised to handle the activated tasks when the state change happens. Otherwise the embedded component is idle.

The component state actor module serves as the broadcaster of the device information to other interlocked devices and the HMI console monitors the system. The broadcaster sends the state information via variables to other devices, in order to process the stage change as an input resource for the consequential output actions. In some cases, a high priority state (such as error or alarm) needs to be sent to the receiver in real time. In this case the RTOS ensures that these messages are handled within the specified time frame.

Having discussed the CB automation framework from the design and management perspectives of the component software architecture, the next part of the chapter is focussed the orchestration of Web Services for the sequencing and synchronised execution of a CB manufacturing process at the message passing level. Web Services provide an open, standards-based approach for connecting applications enabling multi-device co-operation and knowledge sharing, in order to create higher-level processes.

6.6 Part I Conclusion

The modular design of automation systems enabled by component-based design for mechatronics modules has been discussed in this chapter. The construction of component software integration has also been reviewed, in addition to its lifecycle development. The concept of embedded automation devices has been presented, where functionality is encapsulated into devices at the stage of commissioning components. With this, the machine builder only needs to focus on integrating the system through a knowledge of its required overall behaviour and what functionality the components provide. Knowledge of how individual programs are managed at low-level device configurations is no longer required. In addition, the requirements and provided functionalities of the CB design approach have been addressed from end user perspectives on building and reconfiguring the control system. In the next section, the core design, implemented technologies, and integration approaches based on Web Services will be presented.

Part II- Web Services-Based Automation System

The component-based automation system, as previously discussed, was conceived to enable modular automation for agile manufacturing systems. Despite the development of modular and reconfigurable manufacturing systems, the achievement of agile manufacturing was only partially realised. The requirements of agile manufacturing are centred on the close integration of manufacturing systems as part of the wider business workflow, as discussed in Chapter 3. However, seamless integration between high level applications and automation systems at the shop-floor level has not been achieved, mainly due to shop-floor environments being composed of many different independent automation systems, all using different technologies. This presents a real challenge, in terms of legacy system integration that is often linked to further use of proprietary integration technologies and solutions from different vendors. The management of integration is thus made difficult, both horizontally (interoperating among themselves) and vertically (in operation with higher level systems).

In order to aid integration and the development of a new generation of distributed applications in the automation domain, this aim of this research is to develop a technologically neutral platform for business and automation integration. This is achieved by building on the technology commonly used in the integration of business systems, namely Web Services technology, with the aim being to support enterprise and factory floor integration via the creation and integration of a vendor-neutral platform automation system.

Web Services implementations have been proposed in the building and medical domains, in the context of device monitoring. However, no research has demonstrated the real application of implementing Web Services on embedded control devices within the CB design and TCP/IP networking in the discrete-machine operation. The proposed Web Services solution regarding embedded devices will be applied to the integration of control systems, especially for machine assembly lines. Therefore, this work aims to demonstrate the feasibility of the proposed Web Services solution in component-based manufacturing systems, with the use of embedded devices to enable reconfigurable automation systems.

6.7 An Introduction to Web Services Enterprise Integration

As presented in section 6.3.2, the adoption of the component-based design for business and automation systems aims to reduce the design and development time of both business workflows and control systems, through the application of reusable software components.

In terms of current component-based software development, companies are working with different kinds of applications, which are based on different de-facto standard technologies. This leads to difficulties in the integration of business processes and applications. It has been reported by Michael Stal [65] that the various component software and object-orientated programming technologies used in Enterprise Application Integration (EAI) have some limitations in terms of flexibility and agility. This is because the available EAI solutions are proprietary, complex to use, and do not interoperate well with each other. The reason for this is the heterogeneity caused by differing standards in:

- Network technologies and devices
- Middleware solutions and communication paradigms
- Programming languages
- Interface technologies
- Data and document formats

Recently, SOA and Web Services (WS), used in common standard communications such as HTTP and XML, have emerged as technologies that may be implemented to integrate various types of middleware, in order to resolve the heterogeneity of these different technologies.

Regarding the SOA, all software components are modelled as services. SOA and Web Services approaches are similar to that of component-based software engineering, but a major difference is that the focus of application design is shifted to composing services invoked over a network. With the SOA approach, the designer is not building a program, a functional unit for purpose/use only. Designers are building a service that may be used in multiple business contexts that has a well-defined set of interfaces [76].

Regarding the design of component-based automation systems in this research, SOA and WS are implemented to facilitate the integration of control applications, using components with well-defined interfaces. Web Services provide the service descriptions (e.g. operation, state update, and error diagnosis service) for the automation device. In this context, control applications are commissioned with these services as the interface to the native program (C/C++) of control devices. The adoption of SOA and WS for automation systems will be discussed later on in this chapter.

Integrating existing component-based technology (i.e. object-orientated middleware) into the web had not resolved interoperability problems [95], as a result of the middleware not being implemented with the web in mind. Hence, such middleware needed to be transformed in order to allow method requests seamlessly over HTTP.

In order to resolve such interoperability issues, Web Services adopted a targeted Service-Orientated Architecture (SOA). It is common within the SOA to implement service brokers (as in CORBA and DCOM middleware) using standard internet protocols and platform neutral XML (Extensible Markup Language). As shown in Figure 6-13, the service provider publishes available services to the service broker, and this acts as an aid in searching the associated repositories. Clients or service requesters find the provided services from the broker.

The details of the key SOA mechanisms, as shown in Figure 6-13, can be found in Chapter 4- section 4.7.2

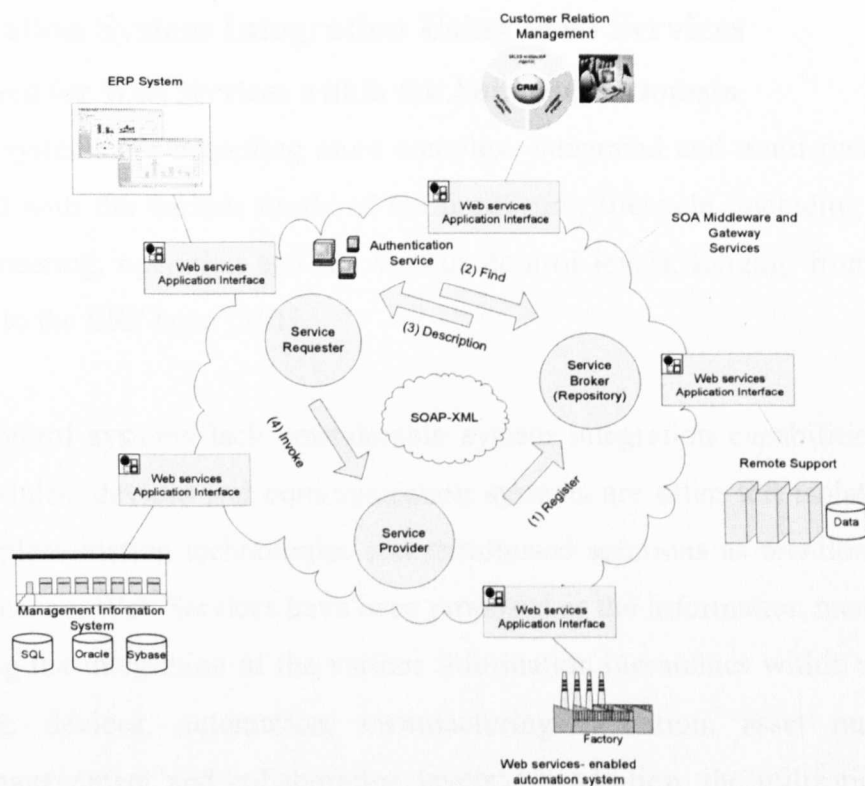


Figure 6-13: A Basic Principle of SOA Architecture

As characteristic of Web Services in building, publishing and locating services over the network, Web Services integration technology has design patterns or architectures appropriate for event-driven, composite, and autonomous distribution of applications at the business level. Regarding these design patterns, Web Services have some similar functions (i.e. autonomous, distributed control systems and event-based tasks) to those required in the automation domain. It is therefore anticipated that adopting Web Services for manufacturing systems could reduce the complexity of distributed automation systems and hence improve overall performance.

The following section will present the need for Web Services and where they fit into the automation system.

6.8 Automation System Integration Using Web Services

6.8.1 The Need for Web Services within the Automation Domain

Automation systems are becoming more complex, integrated and multi-functional, in order to deal with the various needs of the production lifecycle, including purchase, design, engineering, operation and the various control levels, ranging from the field device layer to the ERP layer [101].

However, control systems lack considerable system integration capabilities. Several machine modules, devices and communication systems are often left isolated, due to differing implementation technologies and customised solutions as mentioned earlier. As a consequence, Web Services have been proposed as the information model capable of facilitating the integration of the various information hierarchies within automation systems (i.e. devices, automation, manufacturing execution, asset management, enterprise management and collaboration layers). In addition, the utilization of Web Services within the automation system enables open standards (as in platform and language-neutrality), which are technologically neutral, in order to provide interoperability between diverse peripheral devices and different vendors. XML is used as the standard regarding the means of automation device communication. There are several ongoing projects [i.e. RIMACS, SOCRADES, SODA] and researches [83, 68, 60] implementing Web Services technologies, but the structure of the implementation of Web Services for the reconfiguration of machine systems is still unclear. This research aims to define the clear role of WS functionality within control systems and in accordance with a component- based design approach.

The transformation of Web Services for the purposes of device operation will be discussed next, in terms of requirements and ontologies.

6.8.2 Transformation of Web Services for CB Automation Systems

Web Services should have the capacity to achieve seamless automation integration and the creation of flexible and agile manufacturing systems. The role of Web Services in facilitating business and manufacturing tasks must be specified to scope development areas and, in accordance with the requirements of future manufacturing systems, to consider the design of automation systems in the areas (i.e. openness, re-

configurability, performance, reliability, visualisation and data monitoring, maintenance and integration to higher level business systems) as shown in the Table 6-2.

Table 6-2: The Adoption of Web Services within Automation Systems

Requirement	Methodology	Web Services Provision
Open automation platform	<ul style="list-style-type: none"> • Non- vendor specific devices • Unified interface for interoperability between different vendor devices 	<ul style="list-style-type: none"> • Developing the XML message as the standard communication for devices.
Reconfigurable automation systems	<ul style="list-style-type: none"> • Distributed component-based design • State-transition machine design • UPnP and Device discovery 	<ul style="list-style-type: none"> • Component functionality is prefabricated and interfaced to the Web Services defined by WSDL standard. • Web Services provide dynamic device discovery functionality.
Performance and Reliability	<ul style="list-style-type: none"> • Response time between I/O in inter-connected controllers, under 30 ms in soft real-time or under 	<ul style="list-style-type: none"> • RTOS will be implemented in the Web Services development for a soft real-time ^(a) capability. • Deterministic Web Services communication at TCP/IP stack
Visualisation and Data monitoring	<ul style="list-style-type: none"> • HMI interface • VRML 	<ul style="list-style-type: none"> • Reliable state propagation from Web Services using WS-Eventing mechanism for (near) real-time operation and visualisation.
Maintenance	<ul style="list-style-type: none"> • Support remote diagnostic applications • Data history logging • Dynamic device meta-data for device lifecycle, health monitoring and maintenance routine 	<ul style="list-style-type: none"> • Web Services metadata devices reports to the monitoring system and keeping log files for data analysis. • Web Services will provide interface to the local and remote data monitoring system.
The integration to higher-level applications	<ul style="list-style-type: none"> • Seamless integration 	<ul style="list-style-type: none"> • Web Services interface provided by WSDL for enterprise and manufacturing supervisory control integration.

(a) In this constraint, the machine task is allowed to miss the deadline (i.e. input to output response time) for sometimes, but it should not affect the robustness of the machine operation and synchronisation between machine components

Web Services Functionality in the CB applications

The encapsulated functionality of components, as identified in section 6.3, and the user requirements outlined in section 6.4 form the specification for Web Services, which will be consumed by other integrated applications at a higher level.

The Web Services element is implemented within the component, in order to provide the automation system with

- A device execution service (*details in Chapter 8- section 8.5.1*)
- A communication service (*details in Chapter 8- section 8.5.2 and 8.5.3*)
- A device information and description service (*details in Chapter 8- section 8.5.2 and 8.5.3*)
- A device “plug-and-play” service (*details in Chapter 7- section 7.4.2*)

The *device execution service* is directly related to the operation of the I/O device on the component. The WS provides an interface for device operations, such as reading sensors and executing actuators in response to requesters, which can be manufacturing execution systems or business applications. The *communication service* is concerned with device state propagation over the network to specific addresses. This service enables device-device and device-business/manufacturing support application communications. The *device information and description service* is initialised during a device start up, and can be found, upon request from other interested devices or applications, to provide information and services including part name, serial number, programme version and network address. The *device plug-and-play service* facilitates self-initialisation functionality during the control system installation or start-up. Newly added or existing devices within the control system autonomously announce themselves to other participating devices and applications, when they start up.

6.8.3 Building of Control Applications

Having defined the device operations as the services of each component, the complete machine operations and work flows are built from *ontologies* and *service orchestration*. These ontologies are concerned with the orientation of process work flows described from the service execution and message passing at the device level. The combination

and orientation of provided services will form the wider work process of manufacturing systems, and an example of such work is demonstrated in Figure 6-14.

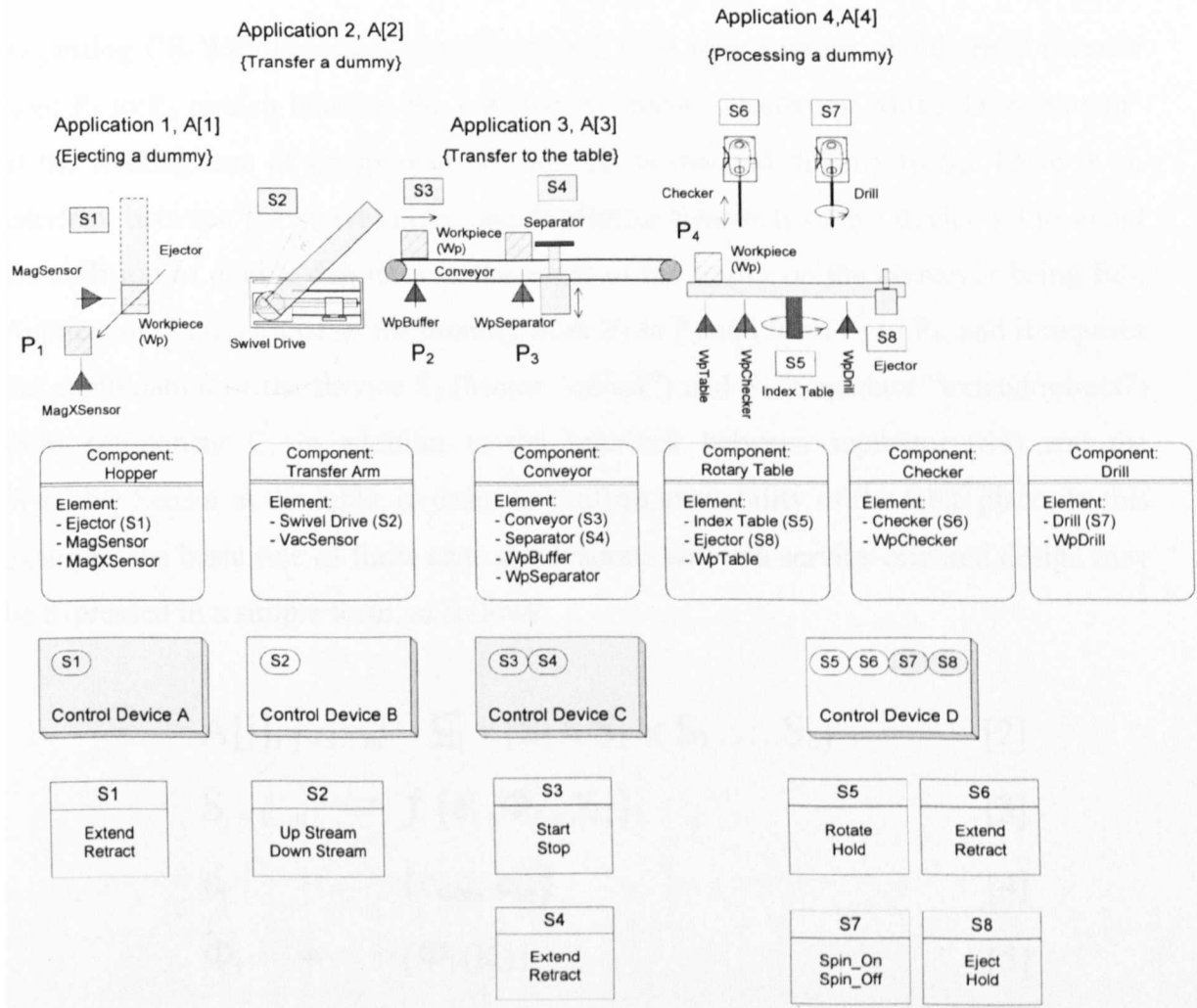


Figure 6-14: Service Orchestration Ontology

If the components (control devices) A, B, C, and D represent the physical modules of the assembly system, they contain operating functionalities as services $[S_i, i = 1..n]$, which will be used to build the control applications $[A_j, j = 1..m]$. The control application term is used to manipulate the desired control objective, such as “eject a dummy” and it is important to note that, in this research, the devices are encapsulated with distinct services unique to the control system. For example, S_1 is only for control device A, it cannot be included within other control devices. Thus, the boundaries of CB design are clear and complexities are easily managed. The control devices in the machine are composed of single or multiple services, depending on the decomposition

of the machine system. As shown in Figure 6-14, control device D provides four services: table rotating (S_5), part checking (S_6), drilling (S_7), and ejecting (S_8).

Regarding CB-Web Services, if application 2 (A_2) is to perform the dummy transfer from P_1 to P_2 (which matches the function S_2 “Move_Upstream/ Move_Downstream” of the rotating arm of component B), then A_2 is mapped directly to S_2 . There is an interlock between the swivel drive and WpBuffer Sensor (Control device C) to avoid the collision of double dummies in the event of the buffer on the conveyor being full. Application 3 (A_3) transfers the dummy from P_2 to P_3 and from P_3 to P_4 , and it requires the combination of the service S_3 (Motor “on/off”) and S_4 (Separator “extend/retract”) from component C, in addition to the interlock between separator (S_4) and the WpTable Sensor at the table, in order to confirm availability of the table place. In this example, the basic rule of finite state applications within a service-oriented design may be expressed in a simple form, as follows:

$$A[j], j=1\dots m \subseteq \{S_1 \times S_2 \times S_3 \dots S_n\} \quad [2]$$

$$S_{i=1\dots n} = f(\varepsilon_i, \Phi_i, X_i) \quad [3]$$

$$\varepsilon_i = \{e_{out}, e_{in}\} \quad [4]$$

$$\Phi_i = \{\Phi_i(K)\} \quad [5]$$

$$X_i = \{X\} \quad [6]$$

Where

A = Applications

S = Provided services

Φ = Component transition states

$\Phi_i(K)$ = Current component i transition state

ε = Local event(s) and condition(s)

e_{out}, e_{in} = Output event(s) and input event(s)

m, n = Number of applications and services

j, i = Application and service ID

K = State ID

X = Interlocking elements

The instance of built applications for the power train machine (i.e. a basic control definition), can be written, as follows:

$$\begin{aligned}
 A [1] & \subseteq \{S_1\} \text{ where } S_1 = f(\epsilon_1, \Phi_1, X_1) \\
 A [2] & \subseteq \{S_2\} \text{ where } S_2 = f(\epsilon_2, \Phi_2, X_2) \\
 A [3] & \subseteq \{S_3, S_4\} \text{ where } S_3 = f(\epsilon_3, \Phi_3, X_3) \\
 & \quad S_4 = f(\epsilon_4, \Phi_4, X_4) \\
 A [4] & \subseteq \{S_5, S_6, S_7, S_8\} \text{ where } S_5 = f(\epsilon_5, \Phi_5, X_5) \\
 & \quad S_6 = f(\epsilon_6, \Phi_6, X_6) \\
 & \quad S_7 = f(\epsilon_7, \Phi_7, X_7) \\
 & \quad S_8 = f(\epsilon_8, \Phi_8, X_8)
 \end{aligned}$$

A [3] – Moving a work piece from C to D = S_3 (ON) x S_4 (Extend)

$$S_3 (\text{Motor}) = f_3 (\epsilon_3, \Phi_3 (K), X_3)$$

$$S_4 (\text{Separator}) = f_4 (\epsilon_4, \Phi_4 (K), X_4)$$

$$f_3 = (\epsilon_3 (\text{WpBuffer} == \text{On}) \text{ AND } \Phi_3 (\text{Motor} == \text{stop}) \text{ OR } X_3 (\text{Separator} == \text{Extended}))$$

then Set the service $S_3 = \text{ON}$;

$$f_4 = (\epsilon_4 (\text{WpSeparator} == \text{On}) \text{ AND } \Phi_4 (\text{Separator} == \text{Retracted}) \text{ AND } X_4 (\text{Table} == \text{set AND WpTable} == \text{Off})) \text{ then Set the service } S_4, = \text{EXTEND};$$

This form of equation illustrates how the control application is built into a finite state machine in order to compose the complete machine application. In the real machine environment, this control sequence is more involved with the multiple interlocking of states and transitions between devices, in addition to the impact of device operation timing logic (i.e. time expired).

As mentioned earlier, the machine application is the combination of services formed by trigger events and sets of conditions. The reconfiguration of the process, therefore, is achieved via the alteration of the application design associated with the state behaviour of the component, as represented by $f(\epsilon_i, \Phi_i (K), X_i)$. To support flexibility and re-configurability, user applications are composed by a process definition editor tool (i.e.

the PDE software has been previously developed for the COMPAG project), which provides a platform for finite state machine development and interlocking, with graphical interfaces to the device service and the state machine. This tool replaces the need for manual hard-coding of applications (i.e. manually-programmed control logics). This research outlined in this thesis requires the definition of a Web Services device interface to the PDE tool in order to support the translation of the tool's graphical process workflow representations onto real control devices (*details in 6.8.4*).

6.8.4 Implementation and Enabling Technology

The technology and methodologies for the design of discrete automation systems according to the CB approach and Web Services within embedded devices is presented in this section. The main design process of the control application is involved in creating the state transitions and control algorithms (*as discussed in 6.8.3*), Web Services interfaces, I/O interfaces for physical I/Os and integration with the RTOS kernel, all of which support reconfigurable in automation systems. The implementation model of these elements is shown in Figure 6-15.

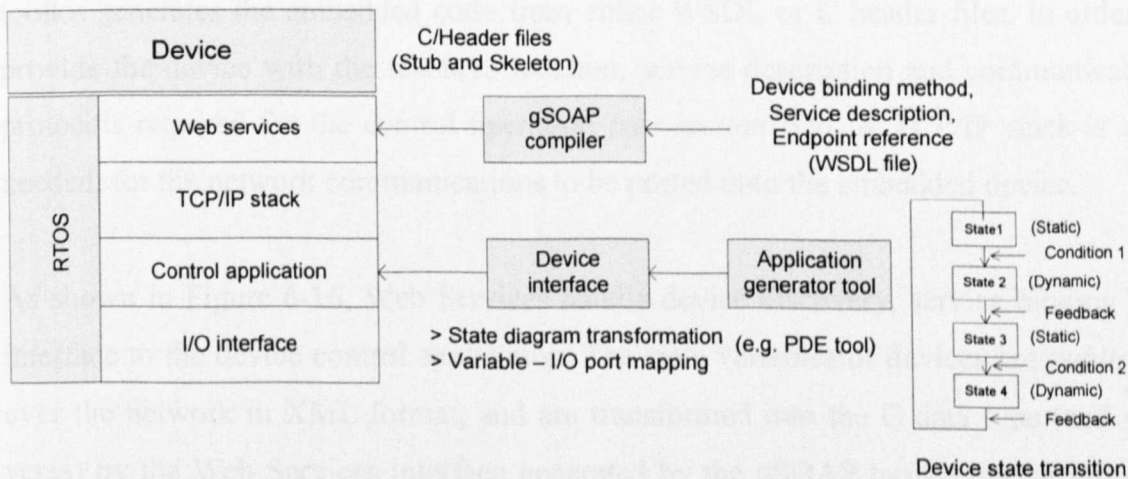


Figure 6-15: Component-based Web Services Implementation Model

It is important to note that, in this research, the target control device is an embedded micro-processor utilising a C/C++ compiler. In addition, the programming of all modules (i.e. RTOS, Control application, TCP/IP stack, and Web Services) of the control device is with C/C++ code.

It has been noted that scan-based operating systems are not the most efficient way to run an I/O “check and execute” command, especially for embedded devices. In this research the RTOS is required, at the kernel level, to handle the event-driven mechanisms of the control system, such as task scheduling, interrupt, idle, and multiple interlocks.

The hard-coding of the device operation is generically programmed using C/C++ code and stored in flash memory. Low-level programmes handle the I/O operations in accordance with the higher-level command requests. The system integrator does not need to rewrite this embedded code in order to build the system. The control configuration (i.e. set of state behaviour functionality defined by interlocks) of devices is created or altered within the PDE engineering tool. An interface is required: (i) to interpret the state logic diagram of the PDE into C/C++ structure required for the component, and (ii) map physical device I/Os to the state variables used in the control application.

The gSOAP toolkit is used for binding Web Services capabilities into the device. The toolkit generates the embedded code from either WSDL or C header files, in order to provide the device with the resource location, service description and communication protocols required for the control operation (*see section 6.9*). A TCP/IP stack is also needed, for the network communications to be ported onto the embedded device.

As shown in Figure 6-16, Web Services handle device discovery, service binding and interface to the device control application. The state variables of devices are *published* over the network in XML format, and are transformed into the C data type (and vice versa) by the Web Services interface generated by the gSOAP toolkit during runtime. External and local state variables are then passed to the control application of the devices. Activation of the service is dependant on meeting the set of conditions defined by logic and linked to the input gathered from the interfaces to I/O devices. It is noted that the WS-CB design architecture proposed in this research aims to enable the Peer-to-Peer (P2P) control systems. The details are presented in section 6.9.2-2.

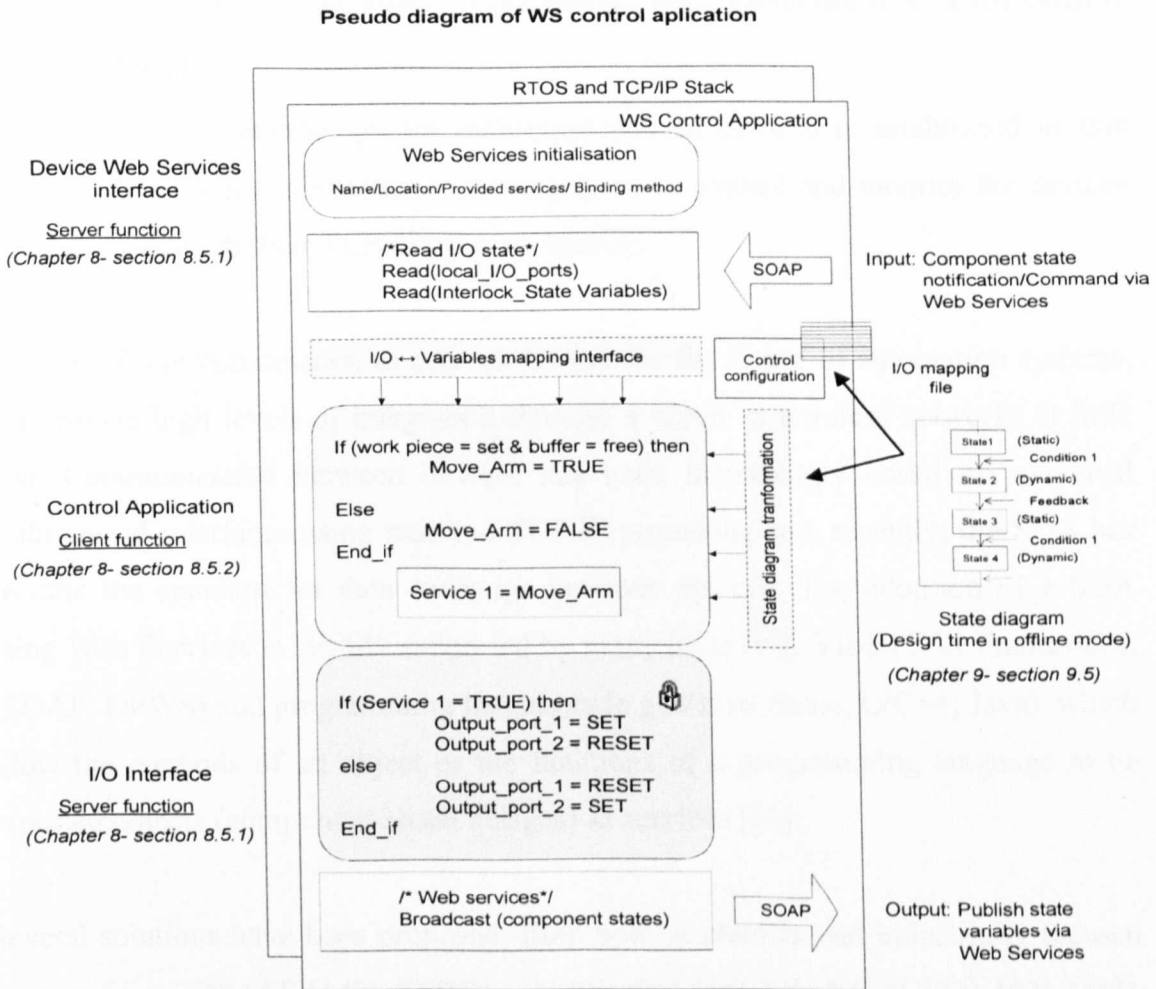


Figure 6-16: The Implementation of the WS-CB Automation System Design

Although the gSOAP code generation technique is widely adopted for Web Services applications, the real-time use of the gSOAP tool on embedded devices within automation systems in terms of performance, reliability and feasibility has not yet been demonstrated in any research.

In the following sections, the implementation of the framework of Web Services in devices binding and communication techniques using the gSOAP tool is developed and discussed. The implemented methodology of developing Web Services and code generation techniques for embedded control devices is also detailed.

6.9 Web Services Design Approach for Distributed Embedded Control Devices

The WS design methodology for embedded control devices is established in this section with Ethernet connections allowing users to control and monitor the devices specifically through their TCP/IP network address.

The use of microprocessors, as control devices for the future of automation systems, can provide high levels of integration through a wired or wireless networks at little cost. Communication between devices has been increasingly based on universal multifaceted interfaces using standard TCP/IP protocols, and, recently, the SOA has become the standard for data exchange between devices. The adoption of a SOA using Web Services is already supported by many tools (e.g. Visual .Net Framework, gSOAP, DPWS) and programming languages (e.g. Visual Basic, C/C++, Java), which allow the methods of an object or the functions of a programming language to be exported objects (component-based designs) as services [60].

Several solutions have been proposed, from both academics and industrially focused projects (e.g. RIMACS [147], SERINA, [148], JINI [56], UPnP [69], [89], [83], [58]), regarding the implementation of Web Services and SOA platforms on the different types of automation device, embedded microcontrollers, PLC-based and PC-based languages (e.g. C/C++, Java, Ladder). Although these researches have proposed Web Services solutions within various approaches and technologies (e.g. JINI [56], UPnP [69], DPWS [54]), most involve applying Web Services for the purpose of addressing devices for monitoring and data access. At the time of writing, the author has not seen any real practice in building a control application for machine operations based on the Web Services architecture. Therefore, this research aims to demonstrate the feasibility of using Web Services on embedded automation devices.

In the following section, the key integration technologies and architectures that support Web Services automation infrastructures will be presented.

6.9.1 Localisation and Standard Discovery Lookup

Regarding the design of control systems, the networked automation devices (meaning the embedded controllers which are directly connected to the communication network in this thesis) need to be located by an address, such as TCP/IP. This addressing schema is required for device set up and communication between nodes, in order to exchange state information.

There are two common ways of defining device localisation. The first is to use a Peer-to-Peer (P2P) search and the second involves building up a central directory as the device registry.

Peer-to-Peer (P2P)

The P2P mechanism is illustrated in Figure 6-17. Within the P2P environment, the device is self-organised [94]. Controller nodes hold information about other nodes, corresponding to their operation within the system through direct node interaction. Interaction is done via the sending of a multicast message (probe) to neighbouring devices, in order to find interested devices to work with (for example, exchange message, invocation). Also, if the new device is installed into a system, it propagates its specification and location, in order to let other specific devices know how to collaborate and work with it. Within the P2P network, nodes interact directly, rather than passing messages to the central server like a client/server network type. A failure of one node would not cause the failure of the whole system.

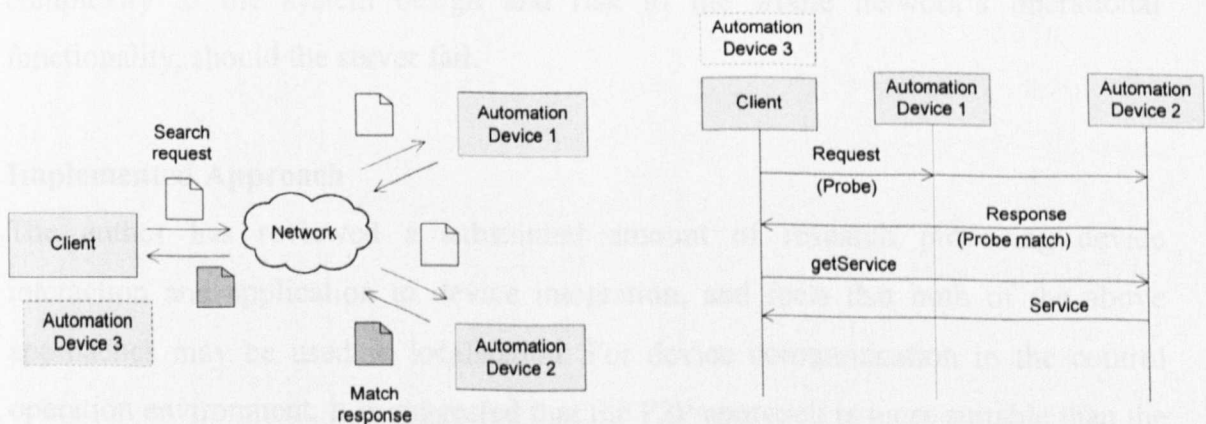


Figure 6-17: P2P Search Procedure

Central Directory Lookup

This is a typical client/server network mechanism. A central server is used to store information regarding automation devices and each automation device has to register its information to the server so that other nodes or remote client applications can look them up in the directory. In this environment, device interaction is reliant on the server organising communication, as illustrated in Figure 6-18.

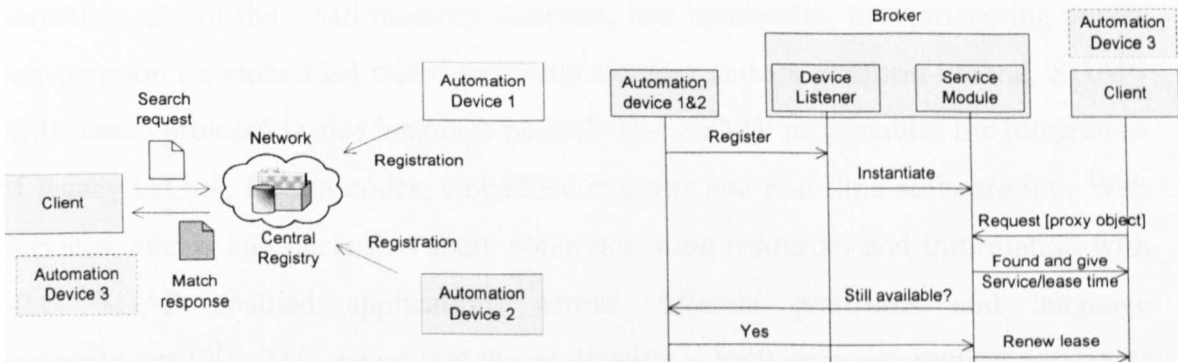


Figure 6-18: Central Directory Lookup

Comparing these two methods, the P2P method does not scale well as the numbers of nodes increase. Searching for target nodes may not be fast and accurate hence response time performance will be compromised. However, the advantage is that failure of one peer would not cause a failure of the whole network. In contrast, the localisation of automation devices via a central directory has a fast and definitive localisation [94]. With this method, however, the additional registration server adds complexity to the system design and risk to the whole network's operational functionality, should the server fail.

Implemented Approach

The author has reviewed a substantial amount of research proposing device interaction and application to device integration, and feels that both of the above approaches may be used in localisation. For device communication in the control operation environment, it is suggested that the P2P approach is more suitable than the central lookup. Devices can interact directly to each other without passing the command and information through the central server. Regarding the integration of

client applications, such as remote monitoring, data acquisition and modelling, the central lookup is more suitable as the interface (proxy server).

6.9.2 Lightweight Code Development for Embedded Devices

In this section, the code generation techniques using the gSOAP toolkit will be presented. The gSOAP toolkit has been chosen for deploying Web Services onto embedded control devices because it brings two benefits. Its can be used to provide portable code for the small memory footprint, low bandwidth, low processing power consumption on embedded micro-processor devices and its platform-neutral, SOAP-XML based protocol is also language neutral. The gSOAP tool enables the integration of legacy C/C++, fortran codes, embedded systems and real-time software into Web Services, clients and peers that share communication resources and information with other SOAP enabled applications across different platforms and language environments [90]. This means that the application is built on programming constructs that can inter-operate with another over a network, through the SOAP-XML methods generated by the gSOAP tool.

The gSOAP tool effectively allows the runtime library to travel back and forth between a SOAP message and the C/C++ formats for embedded C/C++ application communication. Conceptually, the design of Web Services code generation by gSOAP is reflected in Web Services communication within:

Transportation - at the lowest level of the model and handles HTTP communication.

Packaging - provides the XML-based SOAP protocol which facilitates the SOAP message in the Web Service environment, such as the RPC call mechanisms, SOAP encoding styles and meta-data information. All these functions are referred to as the SOAP envelope, header, and body.

Information - carries the XML-formatted SOAP message. This layer is used to establish a SOAP RPC request-response message exchange, using stubs and skeletons for encoding and decoding the invoked operation remotely (as shown in Figure 6-19).

Service - this is considered as a WSDL-defined layer that abstracts the service functionality and access mechanisms of Web Services.

Discovery - deals with service registration and discovery through the UDDI specifications.

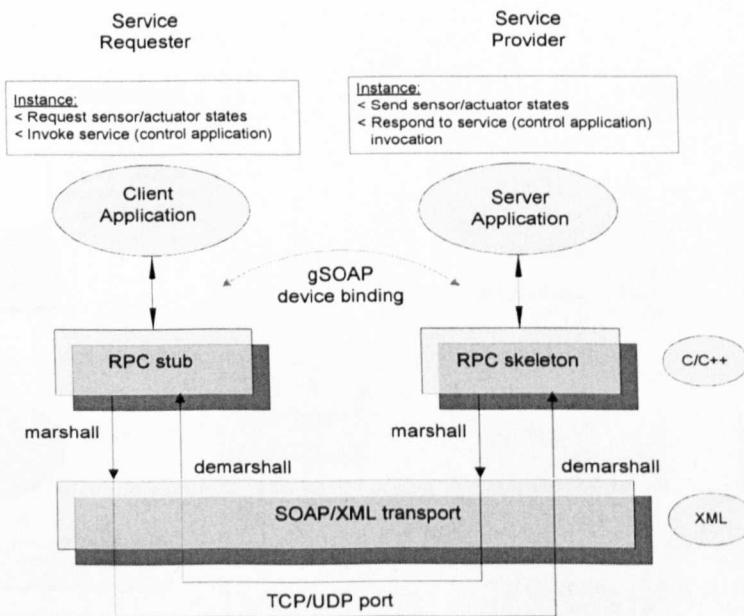


Figure 6-19: Remote Procedure Call in Device Binding

The Web Service communication and service binding mechanisms, including the integration of service (server) and consumer (client) applications (as presented in Figure 6-19), are implemented via generated RPC stubs and skeleton files included in these applications. The routine involves transforming a C and C++ application to a Web Services operation on the network, via SOAP/XML request and response (or publish and subscribe) messages at runtime. In the case of control operations, a client can invoke the service (i.e. read sensors or drive actuators) on the server through the parsing of SOAP commands (i.e. a response is sent back to acknowledge the client).

6.9.2-1 The gSOAP Stub and Skeleton Implementation

The development of Web Services clients and server applications at design time (Development) and runtime (Deployment) is depicted in Figure 6-20 below:

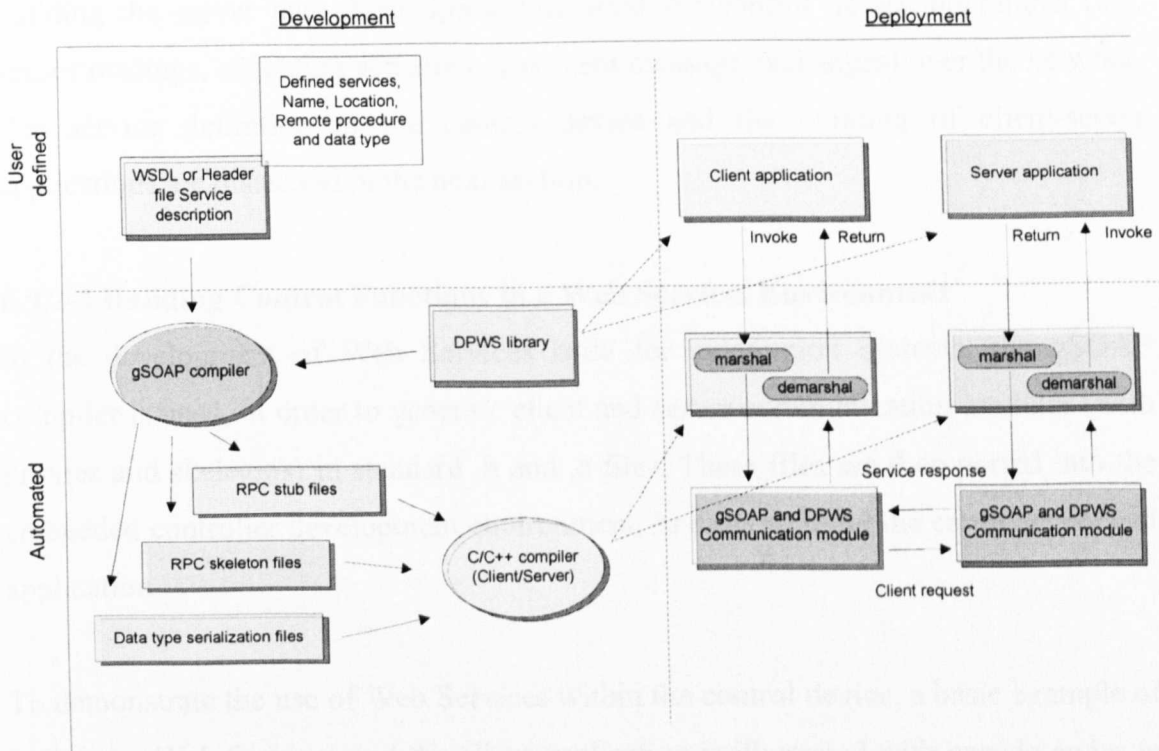


Figure 6-20: Design Time and Run Time of a Server and a Client (extended from [52])

To start building the Web Service functionality of control devices, the control component builder will need to define a header file that contains the service function declaration, with parameters and data types for the automated code generation of the stub and skeleton files by the gSOAP compiler. Alternatively, these files may be obtained from the WSDL document description of the service, remote methods, and the type of parameter data, as defined in the header files. The WSDL and header files generate the same library files and both methods will be presented here.

The Web services functionality defined in the DPWS stack for the control device (*Chapter 4- section 4.9.2*) is enabled by including the DPWS library in the DPWS-gSOAP code generation package provided by Schneider Electric. It is noted that this tool is an open development application which is freely distributed to any developers.

The DPWS library is later used to build the client and server application. This provides the functionality of meta-data devices, such as discovery, addressing, and eventing, which are all used in the building of the control system (*presented in the following section*). The RPC files are integrated with the DPWS runtime library, building the server and client application used for control device operations (e.g. sensor readings, executing actuators, and event message exchanges) over the network. The service definition of the control device and the building of client-server applications are discussed in the next section.

6.9.2-2 Building Control Functions in a Web Services Environment

In the development of Web Services code for automation systems, the gSOAP compiler is used, in order to generate client and server communication modules (with proxies and skeletons) in standard .h and .c files. These files are then ported into the embedded controller development environment, in order to build the complete control application.

To demonstrate the use of Web Services within the control device, a basic example of building a Web Services and the client application is illustrated with pseudo codes in Figure 6-21. Here, the services for turning ON and OFF (PowerState) a light by a service reference name (SwitchPower) and states (PowerState) are identified with the use of WSDL files. However, this WSDL file has been simplified to demonstrate its usage with the gSOAP toolkit. The full WSDL description is presented in Chapter 8-section 8.3. Having compiled the WSDL file, generated the .c and .h files, the gSOAP, and DPWS runtime libraries are then linked with the service and client applications. The standard functions used in the main application of clients and servers are defined in the generated code and these functions (i.e. device metadata, device lookup service, service invocation method) are programmed in the control application.

The implemented codes of the WSDL file, the server and the client project for this example can be found in the attached CD (./WS Automation/Chapter 6).

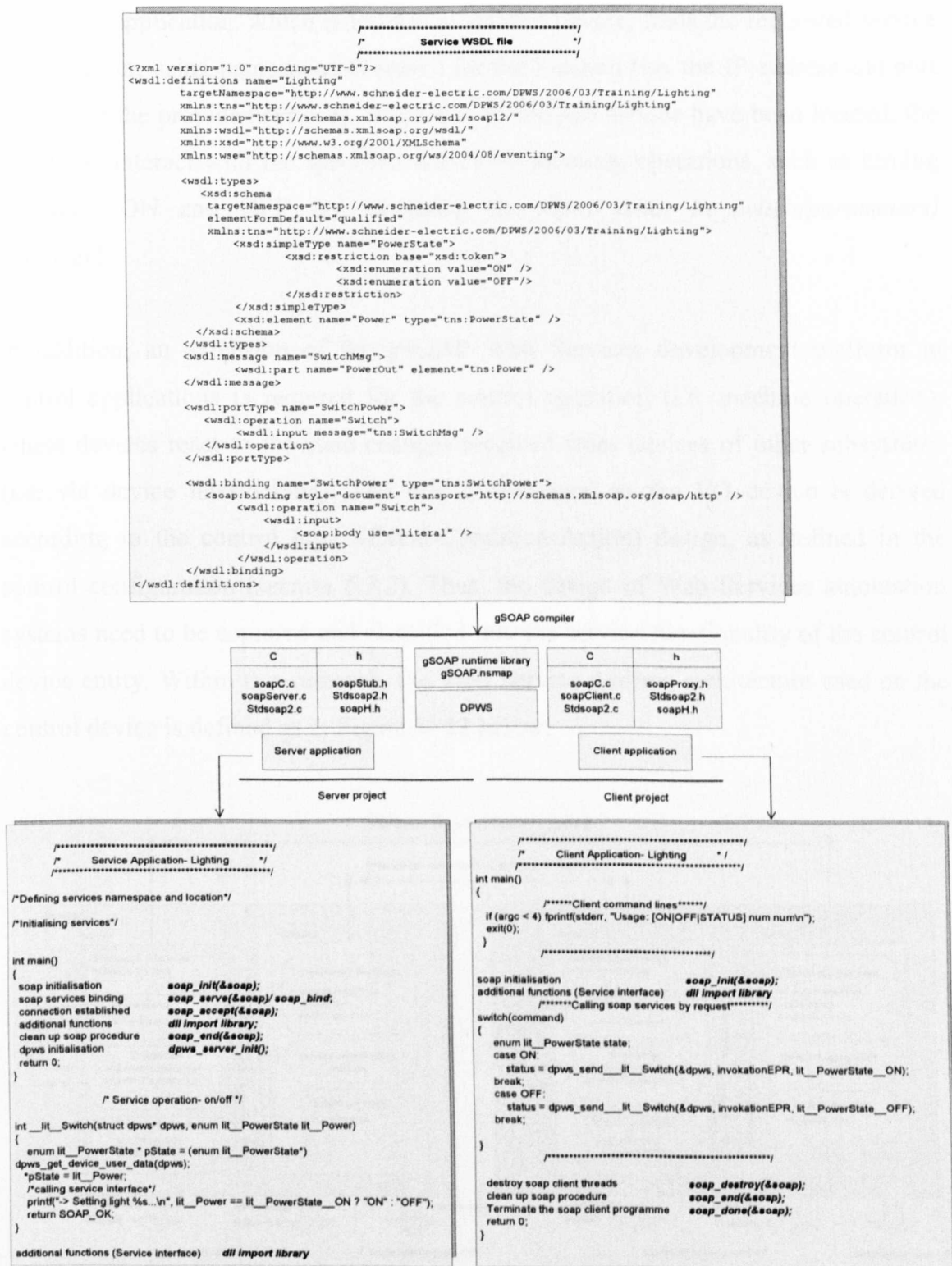


Figure 6-21: Web Services- a Client and Server Project Implementation

The services and associated XML schemas are initialised with unique names and locations on the network. The server provides the service of turning the light ON or OFF, and reading the status of the light. These services are seen as the interface to the local device input and output and can be remotely called by the client application.

The client application, which is located in another device, finds the requested service on the server, using namespace to search for the location (i.e. the IP address and port number of the provided service). When the device and service have been located, the client can interact with the specified device by invoking operations, such as turning the light ON and OFF, by executing the `dpws_send_lit_switch(parameters)` command.

In addition, an extension of the gSOAP Web Services development platform in control applications is required for the control operation (i.e. machine operation), where devices react to the state changes received from devices of other subsystems (i.e. via device interlocking). The output command to the I/O device is derived according to the control logic (Event-Condition-Action) design, as defined in the control configuration (section 6.3.2). Thus, the design of Web Services automation systems need to be captured and classified into the service functionality of the control device entity. Within this research, the Web Service runtime architecture used on the control device is defined as in Figure 6- 22 below:

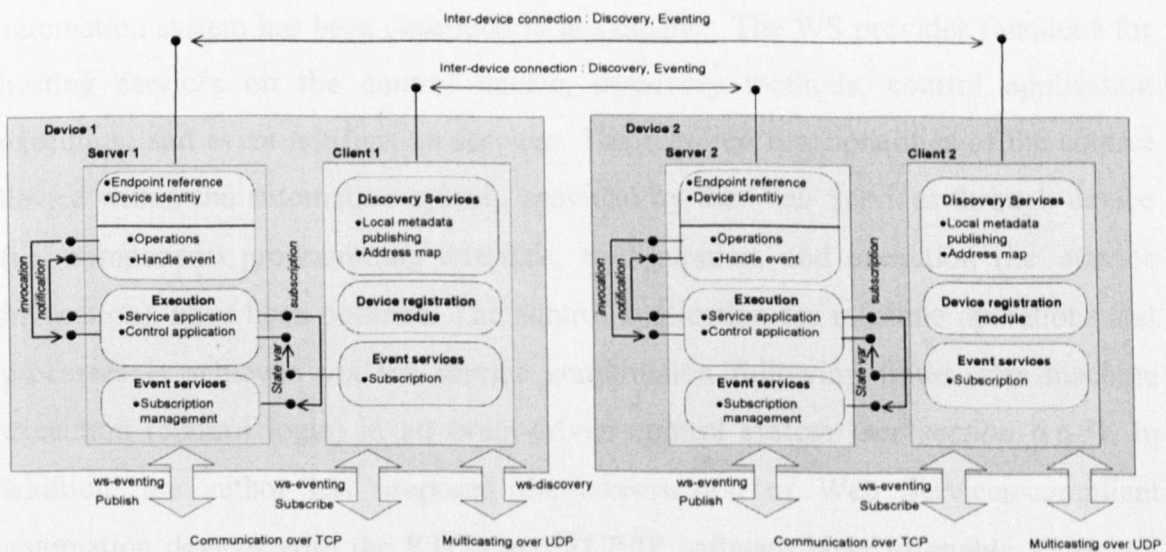


Figure 6-22: Web Services Compliant Control Device Architecture (in line with DPWS)

To enable the P2P model of the control system, the control device contains both client and server capabilities. The client portion of the device is directly associated with discovery services (i.e. WS-Discovery) and registration, in which the device is connected to a network discovery service on other devices. In the case of an automation system, the client acts as an event sink to receive the information (e.g. state variables) from the server of different devices and pass the value to the local server for the control application.

The server is associated with services definition (i.e. WS-Metadata Exchange) and the locations provided for dynamic device discovery and access from clients. In addition, the server provides the publish/subscribe services (i.e. WS-Eventing). In this respect, the server allows clients from other devices to subscribe to the asynchronous message exchange (e.g. state variables), produced by the state changes in the control operation. The server is also included in the control application for I/O device operations.

6.10 Part II: Conclusion

The methodology of applying Web Services to the component-based design automation system has been described in this chapter. The WS provides functions for hosting services on the control device, discovery methods, control application execution, and event-notification services. The required functionalities of the control device within the automation system, provided by the Web Services through device (i.e. component) programming interface, configuration, and execution (i.e. service invocation), have been outlined. The control application for machine operations and processes is achieved via the service combination following finite state machine execution (control logic) in an event-driven control system (*see section 6.8.3*). In addition, the author has proposed the construction of Web Services-compliant automation devices with the RTOS and TCP/IP software layer to enable a peer-to-peer communication in the distributed control systems. In this research, the implementation of Web Services on control devices aims to enable the creation of a neutral platform, where devices from different vendors can interoperate via XML message passing. System flexibility and re-configurability could be enhanced through effective plug-and-play discovery and software interlocking mechanisms via the use of remote procedure calls enabled by WS (*see section 6.9.2-1*).

CHAPTER 7

Implementation of CB-WS

The implementation of the CB design and Web Services approach in the distributed automation system are outlined in this chapter. These design approaches were presented in Chapter 6. Two separate case studies have been carried out in this research. The first has involved a study of distributed component-based automation systems and the second a study of Web Services-based automation devices. The first has involved looking at the primary design work of the CB approach, based on the PLC-based system environment, and the second has involved demonstrating the adoption of WS within the CB design approach.

7.1 Problem Statement

The functionality of the university-based automation test rig control system, which is deemed to be applicable to real industrial machinery and control applications experienced by the Ford Motor Company, needs to be detailed in order to appreciate the case studies discussed in this chapter. In addition the research questions to be addressed in this chapter are based around the design of distributed control systems utilising this test rig, as follows:

1. How is the test rig system broken down into components and elements?
2. How is encapsulated component functionality programmed and reused?
3. What scope of WS functionality is required for component design, operation and communication?
4. Which available technologies may be used for assembling WS control functionalities on the designed component?

7.2 Introduction

In the first case study on CB design, the concept of distributed automation systems is implemented on the FORD-FESTO test rig. The aim is to demonstrate the CB approach in practice and assess the feasibility of the distributed control system, based on an event-driven approach. The test rig is a replica of FORD assembly machine lines, implemented with a controller from Schneider Electric and a distributed I/O interface module connected to sensors and actuators. The control applications of the

machine components are designed using the Unity ProXL engineering tool provided by Schneider Electric. This tool supports the IEC- 61131 PCL programming language standard for control applications. Regarding the design of reconfigurable and reusable control applications, the control application of each component is implemented using Function Block Diagrams (FBD). Using these constructs the internal control logic, relating to I/O operations, can be readily reconfigured and reused for other components.

The author has been responsible for building the FBD of components and the state behaviour and interlocking of components for the control application of the test rig. In addition to the CB control software, other developments were undertaken for the test rig automation system, including:

- The integration of VRML simulations
- The development of control configurations and finite state machines with the Process Editor Engineering Tool
- Operator console and monitoring via HMI screens

The second case study is the progression of the FORD test rig control system, to supporting SOA and Web Services automation according to the methodologies presented in Chapter 6, Part II. In the design *experiment*, the preliminary testing of Web Services-based automation systems involved code generation for the automation components and demonstration of the use of Web Services on a PC that simulates the operation of real control application in the embedded automation device. In this implementation, the gSOAP toolkit is used to generate the remote procedure call (RPC) stubs and skeletons from the WSDL file, which are then used to define the component description for the control application (*see Chapter 6- section 6.3.2*). Web Services server and client applications are built, in order to show the message passing of state variables and executed commands that simulate the real control environment. In the communication process, TCP/UDP ports are used for client and server components to interact using XML message transport. This interaction involves device discovery, execution and message exchange. In addition, device operation, such as actuator manipulation, is realised by the execution of additional dynamic linked library (i.e. dll) files. For example these files support the interfaces that write the commands to ports (e.g. RS232) on the service provider (server) where required.

7.3 Case Study 1: Distributed Automation System (FORD-FESTO Rig)

7.3.1 FORD-FESTO test rig specifications

The test rig system has been divided into four subsystems (i.e. units or stations): (i) a distributed hopper unit, (ii) a buffer unit, (iii) a processing table unit, and (iv) a handling arm unit, as shown in Figure 7-1. Each of these subsystems contains one or more mechanical components that are connected to field devices (i.e. sensors and actuators) through a distributed I/O module (such as an Ethernet interface I/O module with specific TCP/IP addresses, as in Figure 7-2). These device components communicate via the state variables within the finite state machine behaviour of the complete system. The sequence of machine operations is defined by the interlocking of components in the design phase (i.e. using the engineering application tool-Unity ProXL). The control application of each component is then uploaded to a PLC controller which is responsible for the real-time control of distributed the components within the subsystems.

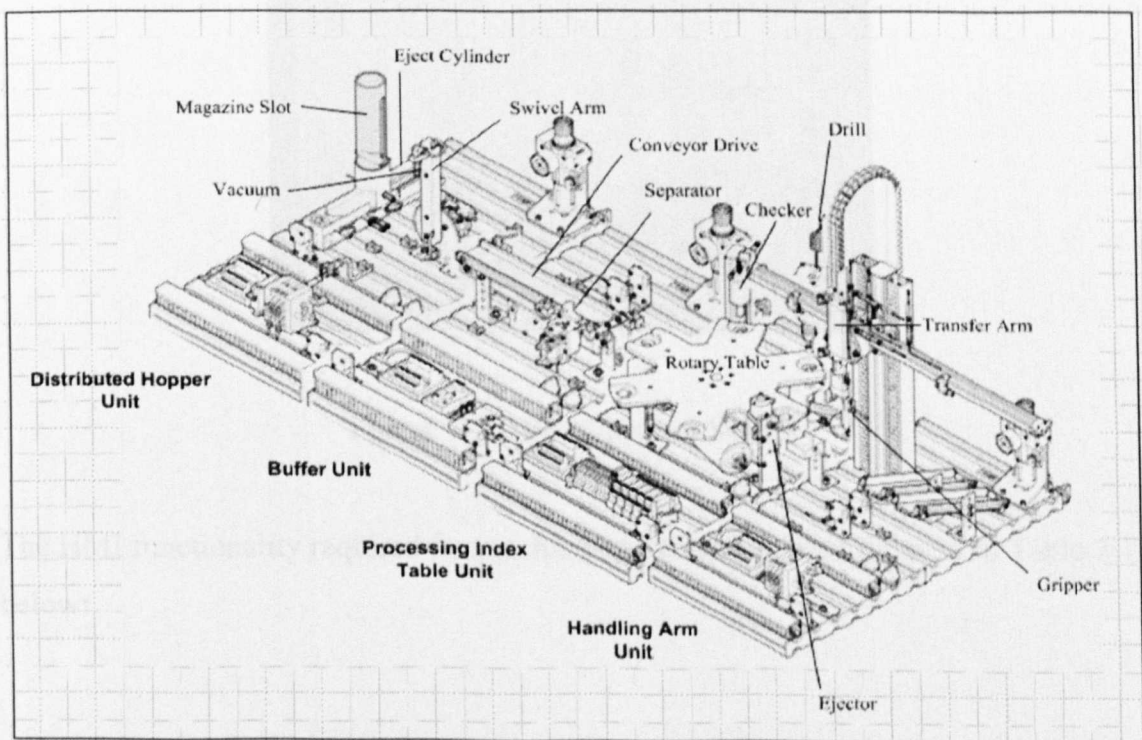


Figure 7-1: The FORD-FESTO test rig platform

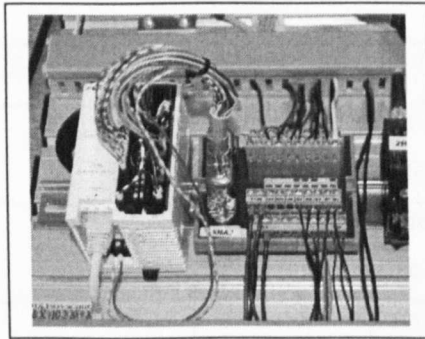


Figure 7-2: Ethernet Interface I/O Module

In addition to the control system, an operator console (i.e. HMI screen) is required for machine operation e.g. to set the operating mode (i.e. Auto/Manual), monitor alarms, reset and system initialisation and monitoring of the component state progression. The HMI control application has been implemented with the Vejo designer tool from Schneider Electric, which enables customisation of the graphical control screen, as shown in Figure 7-3.

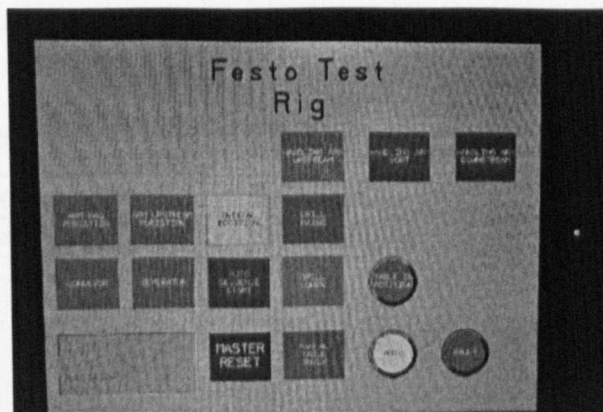


Figure 7-3: HMI Operator Screen

The HMI functionality required for machine operation has been detailed in Table 7-1 below:

Table 7-1 Test Rig HMI

HMI INTERFACE	FUNCTION
AUTO/MANUAL	Operating mode of the test rig to manually control or automatically run units
INITIALISATION	Safely returns all devices to their initial positions, ready for operation
ALARM	Malfunctioning alert to the operator
RESET	Releases the alarm and move devices to safety state before the resumed operation
DEVICE STATUS INDICATOR	Displays the status of devices on-screen, in association with the state variables

The design and integration of the component-based system development is detailed in the following section.

7.3.2 System Development and Integration

The architectural (i.e. Subsystem – Component – Element – State) decomposition of the FORD-FESTO test rig is shown in Table 7-2 below:

Table 7-2: Decomposition of the Test Rig Assembly

Subsystem	Component	Element	State
Station 1: Distributed Hopper Unit	Distribution_Hopper	Eject_Cylinder	Error
			Retracted
			Move_Extended
			Extended
		Move_Retracted	
		Magazine	Error
	Magazine_Empty		
	Magazine_Full		
	Mag_Xfer_Ready	Error	
	Magazine_Empty		
	Magazine_Full		
	Transfer_Arm	Swivel_Drive	Error
Downstream_Position			
Move_Upstream			
Upstream_Position			
Move_Downstream			
Vacuum		Error	
Off			
On			
Eject			
Gripper	Error		
	Not_Gripped		
	Gripped		

Station 2: Buffer Unit	Conveyor	Conveyor	Error
			Off
			On
		Separator	Error
			Retracted
			Move_Extended
			Extended
			Move Retracted
		Workpiece_Available	Error
			No Workpiece
			Workpiece Available
		Workpiece_at_Separator	Error
			No Workpiece
			Workpiece Available
		Workpiece_atConveyor_End	Error
No Workpiece			
Workpiece Available			
Station 3: Processing Index Table Unit	Rotary_Table	Indexing_Rotary_Table	Error
			Table in position
			Table Indexing
		Ejector	Error
			Received Workpiece
			Eject Workpiece
		Workpiece_Available	Error
			No Workpiece
			Workpiece Available
		Workpiece_Available_at_Checking_Unit	Error
			No Workpiece
			Workpiece Available
		Workpiece_At_Drilling_Station	Error
			No Workpiece
			Workpiece Available
	Component_Checker	Checking_Actuator	Error
			Actuator Off
			Drill Hole OK
			Drill Hole Fail
	Drilling_Unit	Drill	Error
			Retracted
Move_Extended			
Extended			
Move Retracted			
Drill_Spindle		Error	
		Drill_off Drill On	
Workpiece_Clamp		Error	
		Unclamped Clamped	

Station 4: Handling Arm Unit	Handling_Arm	Arm	Error
			Downstream_Position
			Move_Sort
			Sort_Position
			Move_Upstream
			Upstream_Position
		Move_Downstream	
		Gripper_Extend_Cylinder	Error
			Retracted
			Move_Extended
			Extended
			Move_Retracted
		Gripper	Error
			Closed
			Open
		Workpiece_Is_Not_Black	Error
			Workpiece_Is_Not_Black
			Workpiece_Is_Black
		Workpiece_Receptacle	Error
No_Workpiece			
Workpiece_Available			

7.3.2-1 Work Flow and Sequencing of the Test Rig System

The functionality of the test rig is representative of typical assembly line operations used to assemble automotive engines. In the automotive process engine parts are inserted to the main body of the engine blocks at each stage along the processing line.

In this test rig, the engine block is represented by a plastic *workpiece* that is processed by a number of tasks such as transferring, buffering, slot checking, drilling and sorting. The assembly sequence is as follows:

1. The workpiece (WP) or dummy is pushed from the magazine slot (*hopper*) by the *ejector* and waits for the *transfer arm* to pick up and transfer it to the *conveyor*.
2. Each WP is then conveyed to the *separator*, which will stop the WP going to the *processing table* if a free slot is not available (i.e. if table is moving or there is already a WP in the input slot).
3. The WP is released from the *separator* to the rotating *processing table*.
4. The *processing table* moves the WP to the *component checker*, which is used to confirm that the work piece is positioned correctly before the *drill* operation occurs. If not, the WP will skip the *drill* operation and raise an *alarm* to the operator.

5. After the *component checker*, if located correctly the WP is then transferred to the *drilling unit*.
6. After drilling, the WP is then moved to the last stage of the indexed *processing table*, the *ejector*. The *ejector* pushes the WP to the *buffer* of the *handling arm* unit.
7. When the WP is located in the *buffer*, the *handling arm* grasps the part and transfers it into the exit *slot*. There are two *slots*: one for black WP's and another for coloured WP's. Colour differentiation is sensed by the *gripper* element of the *handling arm* sensors.

The complete machine application, following the workflow of the WP via machine sequences, is implemented by the state transition diagrams of each element associated with the system components. Appropriate interlocks between elements provide the correct synchronisation.

7.3.2-2 State Transition Diagrams

The component state behaviour is normally described by generic two-, four- and six-state relationships of actuator and sensor functionality. An example of common state transitions can be viewed in Figure 7-4 below:

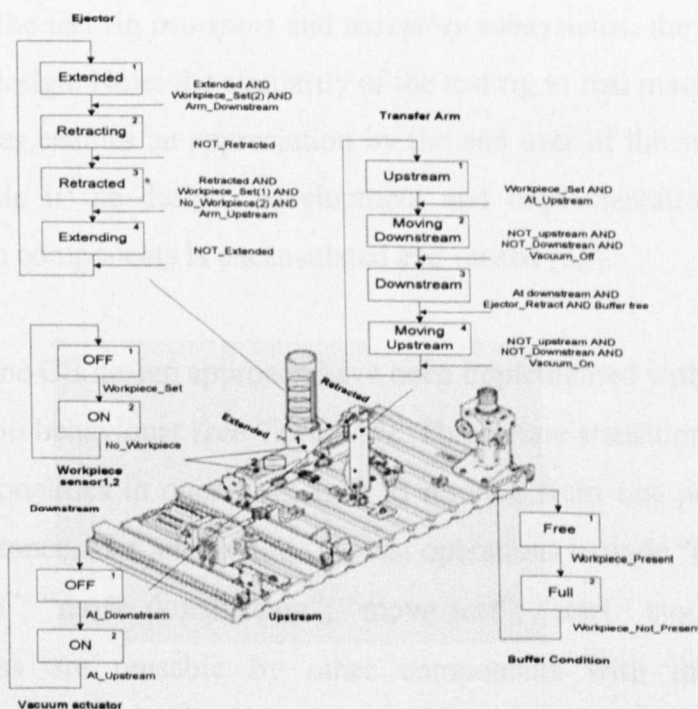


Figure 7-4: Common four-state Transition Behaviour of the swivel *Transfer Arm* Component

Regarding the *transfer arm* component's state transitions, the arm has been implemented with four states:

- State: 0 = *Error*
- State: 1 = *Upstream*
- State: 2 = *Moving Downstream*
- State: 3 = *Downstream*
- State: 4 = *Moving Upstream*

with the initial position at *Upstream*. The *transfer arm* transition from *Upstream* to *Downstream* depends on meeting the condition of the work piece sensor (*Mag_Xfer_Ready*: ON), and with the *ejector* state *Extended* in order to allow the *transfer arm* to move. Likewise, the transition from *Downstream* to *Upstream* requires the *ejector* state *Retracted* and the *buffer* condition (*workpiece_available*: OFF) sensor of the *conveyor* component, in order to enable the action. The dynamic states, such as *moving upstream* and *moving downstream*, are determined by the internal logic of the *transfer arm* element, from its sensors and its previous state, in order to track the moving direction. Full details of the state diagrams can be found in Appendix A.

7.3.2-3 Component Software Development

With regards to the test rig *transport* and *assembly* subsystems, the control hardware is of a standard design. Note: the similarity of the test rig to real machine architectures and functionalities enables an appreciation by the end user of the substantial savings that can be made if the design, development and implementation of the control software for such components is encapsulated and reused [33].

Components in the CB design approach have been implemented with basic two-, four-, or six- transition behaviours (see Table 7.2). These state transitions generally share the same commonalities in operation, such as moving from one position to another regardless of distance, axis and timing. Typical operations include “extend”; “retract”; “move upstream”; “move downstream”; “move sort”; “start : stop” and “on : off”. These operations are reusable by other components with the same required functionality (and state progression i.e. numbers). For instance, the 4- state actuator of the *ejector* element has the same state functionality (i.e. 2 static and 2 dynamic states) as the 4-state swivel *transfer arm*. These states represent the movement of the output

from one position to another regardless of axis and speed. The only differences between these elements are the variable names relating to the I/O ports and sets of interlocks and conditions that allow the action to occur.

In order to utilise the design of the reusable component-based automation system within this project, generic FBD's (Function Block Diagrams) have been derived with encapsulated low-level ladder logic programming, structured text (e.g. `if_then_else`, AND/ OR functions) and standard function blocks (e.g. timing, delay, negate function block). As these derived FBD's are implemented and tested, they are exported to the machine library for future reuse. In addition the exposed functional structure of the FBD has to be implemented in a standard format, so that other components can comply with it. The design is shown in Figure 7-5.

Different components can reuse the previously implemented function blocks of other components by altering the transition condition, and the input and output state variables that are used for the interlocking without having to build the new function block. In this way, substantial time is saved when the system developer is commissioning the machine system.

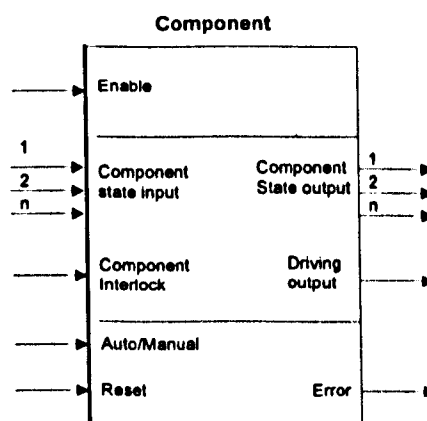


Figure 7-5: Generic FBD Structure

The next stage of developing the CB test rig system is the sequencing of components (via interlocking state variables) to commission the completed machine control application corresponds to assigning the *triggering events* of each component, as previously specified in the state transition behaviour. The complete reference control programme of the test rig is given in *Appendix B*.

7.4 Case Study 2: Web Services Device Control

An approach to the component-based design of embedded devices implemented within the Web Services environment is described in this section. The gSOAP toolkit is used to generate the remote procedure call (RPC) stubs and skeletons from the component description (i.e. the WSDL file) for device binding and interaction. An experiment evaluating control device interaction and execution (via manipulating I/O operation) in a request/response mode is carried out utilising the dynamic linked library (DLL) files, which are located in the WS application. This control application will be deployed in the client controller and SOAP message commands are transmitted to the server application in order to perform the controller task.

7.4.1 Development Platform

The initialisation of Web Services on embedded devices requires the gSOAP compiler and DPWS library to initiate component functionality. The DPWS protocol stack is required for device binding and operation on the network. A gSOAP and DPWS code library has been developed by Schneider Electric to support light-weight XML Web Services that are suitable for real-time embedded devices.

The Schneider DPWS toolkit contains: (i) a WSDL parser (`wsdl2h.exe`) to convert WSDL specifications into a gSOAP header file, (ii) a stub and skeleton generator and (iii) a gSOAP runtime library file (`stdsoap2.c` and `stdsoap2.h`), to be linked with the Web Services applications. In this research, the applications for embedded devices are programmed in C and the additional DLL files are used to add extra integration functionality for the C application. Visual studio .NET with a C/C++ compiler is used as the preliminary testing platform to support the Web Services- client and server applications. For example, the Web Services stub and skeleton source files of the components and the DPWS runtime library are imported into the Visual Studio.NET development project in order to implement the client and server application of the component.

Regarding the server development for a component (i.e. a simple light bulb operation in the case illustrated in Figure 7-6), the initialisation of the device (i.e. the light), provided service (i.e. power on/off), and service namespace and location is shown in

Figure 7-6. These component descriptions are used to support interaction (i.e. device/service lookup, subscription and invocation) with the client application.



Figure 7-6: Visual Studio .NET Web Services Application Platform

(see the attached CD in `..\WS Automation\Chapter 7\dpws_project\dpwscore\samples\Home\server.c`)

The component service (i.e. operation) of the control device is defined in the server application through the DPWS interface (i.e. call function), as shown in Figure 7-7.

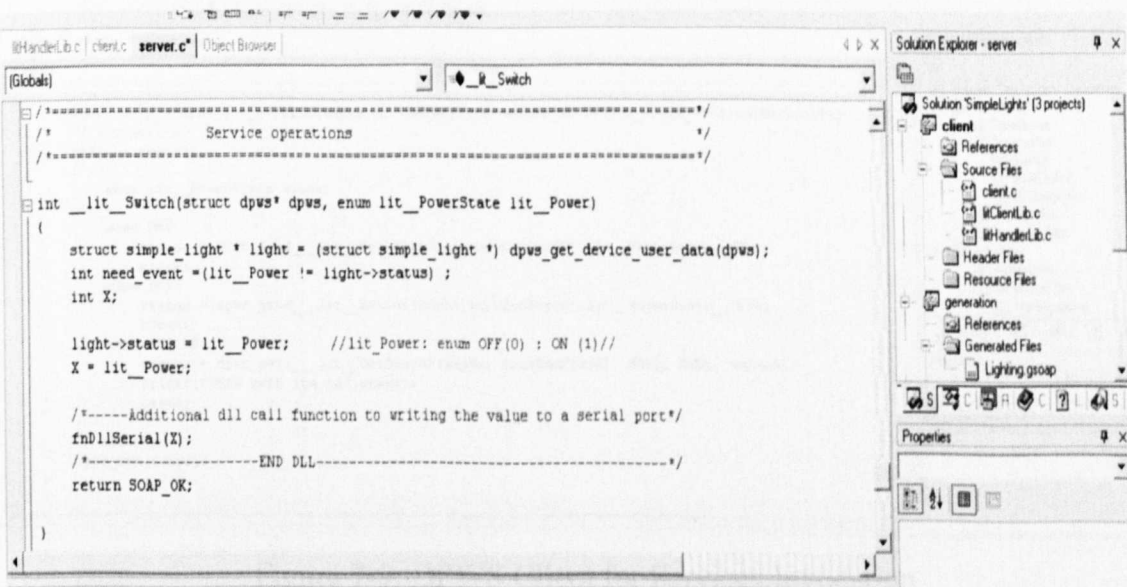


Figure 7-7: The Component Service

(see the attached CD in `..\WS Automation\Chapter 7\dpws_project\dpwscore\samples\Home\server.c`)

As illustrated, the DPWS function `__lit_Switch(arguments)` provides the interface to the device operation for turning the light on and off, by passing the *enum* argument of `lit_PowerState` (On:Off). In this case, the DPWS acts as the interface to the operation `fnDllSerial(X)`, where `X` is the on/off switch command passed on the serial port to simulate the output on and off operation.

```

client.c* | server.c* | Object Browser
[Globals] | main

dpws_init(); // Doesn't matter if this is done twice as this is already done on Win32

// lookup (US-Discovery PROBE message) simple light device
dpws_client_init(&dpws,NULL);

// finding device endpoint
devEndPts = dpws_lookup(&dpws,TRAINING_NS, SIMPLE_LIGHT_TYPE,
                       getDiversifiedScope(scope_suffix) ,&nbDevEndPt );

device = devEndPts[0];

// finding service endpoint
servEndPts = dpws_get_service(&dpws,devEndPts[0],LIGHTING_NS, SWITCH_POWER_TYPE, &nbServEndPt);

// print available devices
devMetadata = dpws_get_device_metadata(&dpws, device);
if (devMetadata) {
    printf("\n-> Friendly name: %s\n Model: %s\n", devMetadata->device_info.friendly_name, devMet

```

Figure 7-8: DPWS Client Initialisation

(see the attached CD in ..\WS Automation\Chapter 7\dpws_project\dpwscore\samples\Home\client.c)

On the client side application, the specific device and service lookup is initiated (*servEndPts = dpws_get_service(namespace arguments)*), in order to discover the location and descriptions (i.e. Device Meta-data) through matching on the namespace from *dpws_LookUp()*, as shown in Figure 7-8.

```

client.c* | server.c* | Object Browser
[Globals] | main

servEndPts = dpws_get_service(&dpws,devEndPts[0],LIGHTING_NS, SWITCH_POWER_TYPE, &nbServEndPt);
// invocation
switch(command)
{
    enum lit_PowerState state;

    case ON:
        status = dpws_send__lit_Switch(&dpws,servEndPts[0],lit_PowerState_ON);
        break;
    case OFF:
        status = dpws_send__lit_Switch(&dpws,servEndPts[0],lit_PowerState_OFF);
        break;
    case STATUS:
        status = dpws_call__lit_GetStatus(&dpws, servEndPts[0], NULL, NULL, &state);
        printf("ORHHH error its %d",state);
        break;
}
// cleanup
dpws_end(&dpws);
}

```

Figure 7-9: DPWS Client for Services Invocation

(see the attached CD in ..\WS Automation\Chapter 7\dpws_project\dpwscore\samples\Home\client.c)

As shown in Figure 7-9, the client application deploys the Web Services command, sending SOAP messages to execute the server application by means of *dpws_send__lit_Switch(service endpoint reference and command arguments)*. A response is then expected back from the device to report the current new state. The *dpws call* command is transformed from this C structure to SOAP messages through

gSOAP and the DPWS runtime library and it is sent over HTTP to the specific endpoint reference of the server application.

In the next section, an illustration of the Web Services-client and server applications of the light switch component is presented.

7.4.2 Web Services Applications

The implementation of the Web Services application, detailed above, has been prototyped on three PC's to demonstrate communication and interaction scenarios for I/O device control. In the test scenario, PC1 is running the server with the DLL interface, *fnDllSerial(X)*, in order to write the data ON and OFF to the connected serial port. PC's 2 and 3 run the client applications, which turn the light ON and OFF and gets the status of the light component from PC1.

The implemented DLL (Dynamic-Link Library) project and source codes for a serial port communication can be found in CD-ROM:

..\WS Automation\Chapter 7\DLLTest

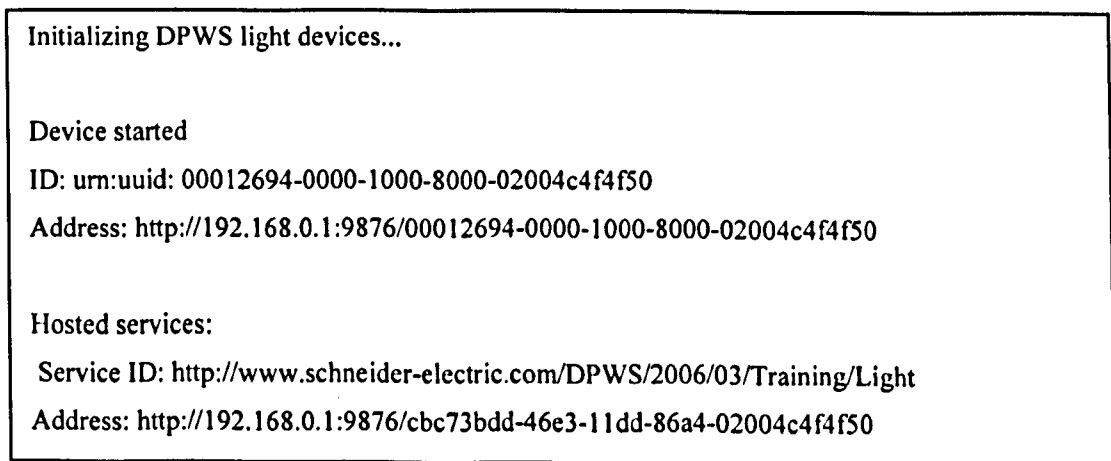


Figure 7-10: DPWS Components and Services Initialisation

During the component initialisation process, the device and service on the server application are initialised with a Universally Unique Identifier (UUID) on the specific IP address, as shown in Figure 7-10. These UUID's are used as the endpoint reference locations for any clients wanting to find the device running on the server during the device and service look up for device binding and service invocation. In

the test, as shown in Figure 7-11, the server provides two clients (i.e. PC2 and PC3) with the service required to control the light switch. The clients can invoke the operations (`__lit_Switch(Power_On or Power_Off)`, and `__lit_GetStatus()`) on the server side, via the DPWS SOAP message command, and the server then responds to the commands.

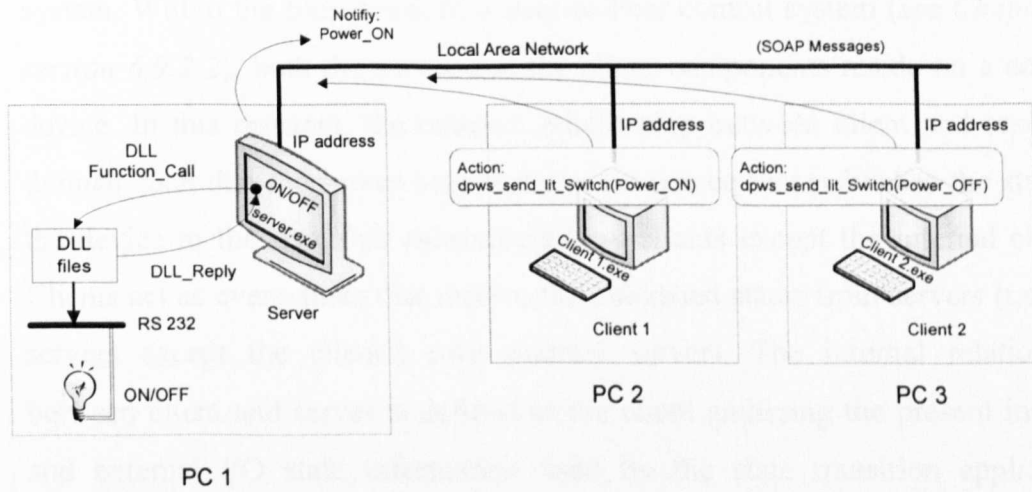


Figure 7-11: Web Services Test Scenario

In addition to the remote service invocation and communication, the execution of the control application on the server (PC 1) has been implemented DDL, in order to support control of the output devices. To demonstrate this, the DLL file was built with Visual Studio C++, to enable the call function to write the control action message through the RS232 serial port. The invocation of the DLL file on the server corresponds to the received command from the client, in which its function's attributes specify the light operation state as "ON" or "OFF".

The clients and the servers (operating on the PC's) were designed to interact in a similar fashion to an automation system, where controllers exchange state information and react according to state changes of interlocked devices. The behaviour of the component on the control device is based on interlocked state transition behaviour, (as presented in Chapter 6- section 6.3.2). Note: propagation of device state changes is enabled by peer communication using the Publish and Subscribe approach in this research (Chapter 8- section 8.5.2).

Adoption of Peer-to-Peer Automation Systems

In the design of an intelligent distributed automation system, the controller device is required to act autonomously, according to changes of the interlocked device in the event-driven based system. This behaviour has enabled the support of Peer-to-Peer communication amongst intelligent devices within the control system. Within the framework of a Peer-to-Peer control system (see Chapter 6-section 6.9.2-2), both the server and the client components reside on a control device. In this research, the external relationship between client and server is defined such that the server acts as the event-source that publishes the state of the device to the specified subscribers (any clients except the internal client). Clients act as event-sinks that receive the published states from servers (i.e. any servers except the client’s own internal server). The internal relationship between client and server is defined as the client gathering the present internal and external I/O state information used by the state transition application running on the client and interfacing to the local server which is driving the output device. The client and server interaction architecture is illustrated in Figure 7-12 below:

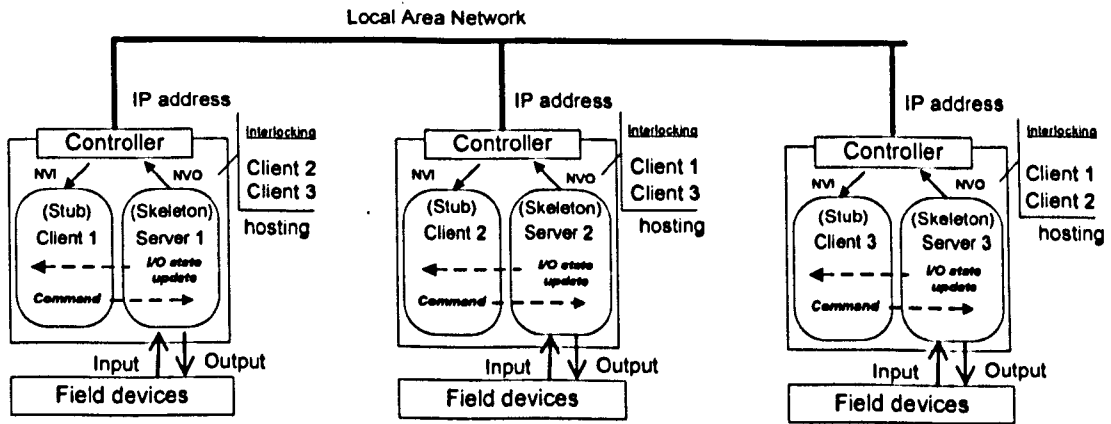


Figure 7-12: Client and Server Devices Interaction

The author has implemented a prototype of the DPWS client/server model as outlined above. The sample source codes can be found in the attached CD:

```
..\WS Automation\Chapter 7\P2P\Dpws_ClientServerModel.c.
```

7.5 Conclusion

With regards to the design of the FORD-FESTO test rig, the CB design of the automation system has been composed with the reusable software components, which are mapped to the distributed physical devices (i.e. system components) of the test rig to perform manufacturing tasks. The synchronisation between devices is performed by the interlocks between components of subsystems, defined via state diagrams. The state variable exchange for device interlocking and synchronisation is done via the Ethernet I/O interface module and the operator control and monitoring systems of the test rig system are implemented through the HMI console connected to the PLC control system via TCP/IP.

The modification of machine control applications has been achieved by reconstructing the synchronization (i.e. interlocks) between component function blocks. To enable the re-configurability of software components within the control system, component function blocks have been built and stored in editable forms, to be modified and re-used later for other component designs. Component alteration may be required, to change the variable names of specific I/O ports for the new components.

In addition to the component-based design, the utilisation of Web Services offers the means to evolve manufacturing automation towards an open standard, which is the technologically-neutral automation integration platform that provides interoperability between various device vendors via XML messages. With the Web Services approach, device description is embedded into the component and exposed to the higher management level with a homogeneous Web Services interface for seamless integration.

Regarding the design of Web Services within this research, a clear distinction of functionality between the server and client applications for the control system has been drawn. During runtime, the initialisation of the server application and broadcast of discovery messages from the client application are used to establish the devices' relationships for the control application and operation. This Web Services initialisation of the client and the server are generated by the DPWS code of the

component that is used to define component services and their location, using initialized namespace and port types.

In addition to the process control and monitoring capability, Web Services may be configured to collate and broadcast diagnostic and process information to assist enterprise personnel. These may include maintenance engineers (who could use the information to monitor, analyze, and document the information to schedule maintenance work or examine machine failure modes for proactive and reactive maintenance), or managers (who may examine machine throughput / uptime in order to predict production volumes). The information may also help the machine builder in providing remote expert assistance.

The next phase of the research is to implement Web Services on embedded microprocessor devices within the component-based design approach on the FORD-FESTO test rig to investigate the feasibility of Web Services control of real-time embedded devices in terms of the performance and reliability of message passing between control devices in under soft real-time constraints. The modularity of the system also needs to be assessed as well as the degree of re-configurability enabled by the WS control architecture.

CHAPTER 8

Web Services Automation Rig Design and Implementation

The design framework of the CB approach and Web Services for control systems, which were presented in Chapters 6 and 7, has been used to outline the design methodology for embedded Web Services. In this Chapter, the implementation of embedded Web Services on microprocessor devices will be presented. As in Chapter 7, the work has been implemented and evaluated on FORD-FESTO test rig in order to demonstrate the feasibility of the approach in a *real* industrial setting. The description of implemented designs will include details of the controller hardware, software specification, design tools, system operation and business system integration.

8.1 Problem Statement

The concept of Web Services for control functionality has been proposed in the previous chapter. In a real industrial application, the implementation of Web Services on the automation device (i.e. embedded microcontroller) needs to address the hardware and software architectures of the real-time control system. Also, a feasible design and implementation platform must be clearly outlined, along with the use case scenarios in: (i) control system build and (ii) manufacturing and business integration. The following questions are to be addressed in this Chapter:

1. What is the embedded microprocessor operating system functionality needed to meet the real-time performance requirements?
2. What is the WSDL description for the design of the component?
3. How is device state behaviour implemented on a WS control platform?
4. What are the WS implemented scenarios that demonstrate the performance of the complete manufacturing system covering machine execution, operational functionality and business integration?

8.2 Test Rig Design Specification

8.2.1 Overview

The distributed control system on the FORD-FESTO test rig is designed as a distribution of *controller nodes* which are instantiated as the software components that interact to perform the manufacturing tasks. As illustrated in Figure 8-1, there are four *controller nodes* responsible for the control tasks of each subsystem within the test-rig. These prototype embedded microprocessor controller devices (i.e. referred to as Field Terminal Block (FTB) modules) have been designed and provided for this research by the Schneider Electric Company.

Each FTB has been designed to contain multiple components, classified by the decomposition of the test rig, as defined in Chapter 7- section 7.3.2. The test rig I/O channels are connected to the FTB I/O connectors through the industrial standard I/O module and each I/O module is only connected to local sensors and actuators of its subsystem / station. There is no I/O cross connection between stations.

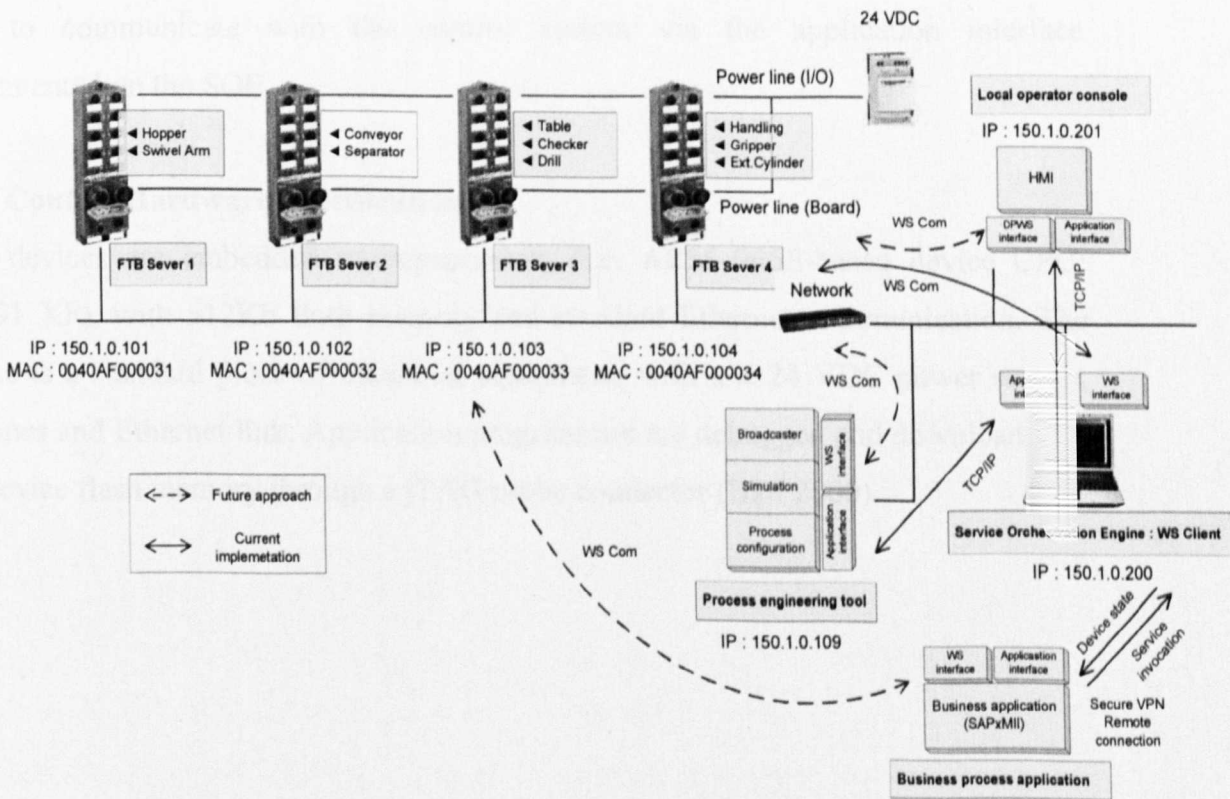


Figure 8-1: Web Services- based Automation System and Integration Platform

For the control system network, each FTB is assigned with a unique static IP address and MAC address, used solely for Web Services device communication through the Ethernet-LAN network. Regarding system operation, the FTB devices when operating as servers provide the DPWS control functionalities that are interfaced to the local input and output of the FTB device. The execution of the DPWS on the four FTB's is initiated by the Service Orchestration Engine (SOE), according to the defined state transitions of components on the Web Services client applications. It is the application integration model proposed in this research, which aims to enable seamless application integration by embracing the WS interface and communication. As shown in Figure 8-1, the DPWS interface implemented on FTB devices allows the process HMI and the process engineering tool to interact directly with the control system via standard WS interface. However, this research of WS automation system is an early work. The full implementation of common SOA middleware for seamless WS integration is on going work with the project collaborators. Therefore, for non-DPWS enabled applications, integration can be achieved through an application interface via TCP/IP connection. In this case, for example, the SAP application (*see Figure 8-1*) is able to communicate with the control system via the application interface implemented on the SOE.

8.2.2 Control Hardware Specification

FTB devices are embedded microprocessors (i.e. ARM 966E-based device CPU: STR91 XF), with 512Kb flash memory and standard Ethernet communication. The device is a standard piece of industrial equipment, with a ± 24 VDC power supply, I/O lines and Ethernet link. Application programmes are debugged and downloaded to the device flash memory through a jTAG probe connector (BDI 2000).

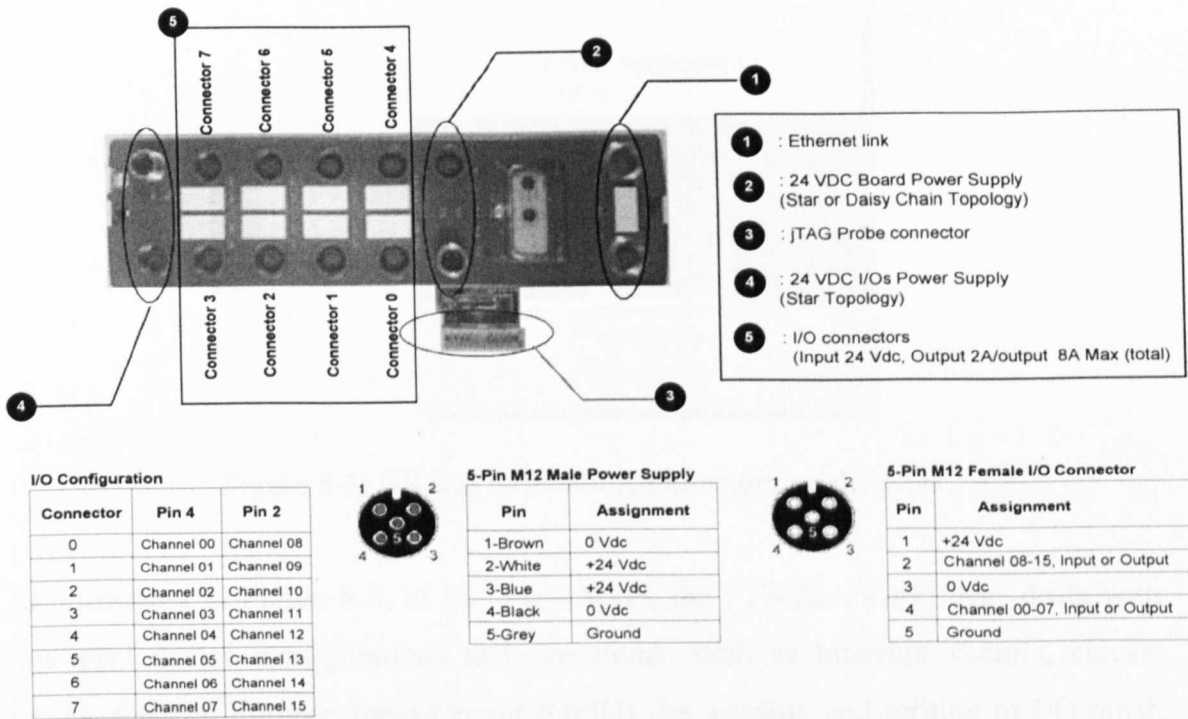


Figure 8-2: Field Terminal Block (FTB) Hardware Module

As shown in Figure 8-2, the FTB hardware specifications state the following:

- Each device has 8 “connectors”, configured as either input or output. There are two I/O pins on each connector that form a combination of 16 I/O lines (8-inputs and 8-outputs).
- There are 2 board power supply connectors: one for input and one for output. It is the same above, in terms of I/O line- power supply.
- There are 2 Ethernet connectors: daisy chain or star topology
- LED’s indicate the activity each input and output channel

8.2.3 OS and Software Architecture

To support the development of real-time applications and TCP/IP communications on the control device, a portable RTOS (i.e. embOS) and TCP/IP stack (i.e. Nexgen) are implemented on top of the standard functionality (i.e. I/O operation), as provided by the embedded hardware. Also, the incorporation of DPWS on the FTB device provides an abstract software layer, in order to support interfaces to DPWS applications, on the control hardware.

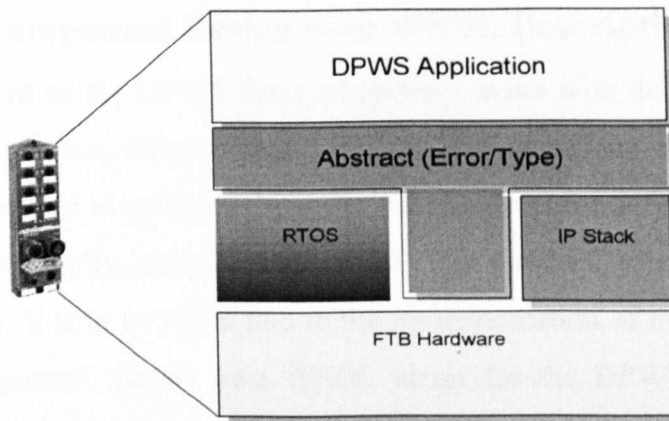


Figure 8-3: OS and Software Architecture on the FTB

As illustrated in Figure 8-3, at its lowest level, the *FTB hardware layer* deals with low-level device configurations and operations, such as interrupt vectors, clocks, timers, General Purpose Input-Output (GPIO) (i.e. reading and writing to I/O pins), UART and serial port access. In the embedded system, the operation task is created and managed by the *RTOS layer*, which is required to support real-time operations such as task scheduling and task prioritisation.

The *IP stack layer* is implemented to support the TCP/IP communication as required by the DPWS application. The IP and MAC address can be specifically set on the FTB when the device is configured. The *abstract layer* is a common interface that bridges the DPWS application task to the RTOS and the IP Stack layer. At the top of the protocol stack is the *DPWS application layer* where the deployment of the Web Services occurs. From the design perspective, the DPWS is created as a task with a defined task priority which can be managed by the RTOS.

All of these software layers have been implemented and debugged using the CodeWarrior development software in the ARM Realview Development Suite. The *abstract*, *IP Stack* and *RTOS* software layers are developed as a sub-project and then ported into the main DPWS project (i.e. the *DPWS application layer*) for each component. In the development of the control software (with the exception of the *DPWS application layer*, which is specifically designed and implemented for each component), the remaining layers remain the same for every developed component. The design process for a DPWS component for embedded control devices is described in the following section.

8.3 DPWS Component Design with WSDL Description

The development of the DPWS for a component starts with defining the component names, element names, element state variables and operations within the WSDL file. This file is later used to generate the stubs and skeletons for developing the server and client applications (physically located on the FTB's and PC respectively as presented in *Chapter 6B*). It is to be noted that in the implementation of the FORD-FESTO test rig, each component has its own WSDL script for the DPWS application. In the control application development, each subsystem (i.e. station) has a corresponding FTB as a container for the components. Hence, each FTB is responsible for the operation of multiple components within the subsystem.

Each component contains the operations and device state variables of contained elements. In the WSDL description, the WSDL file of one component defines all element operations (i.e. execution command, element state publication), state variable names, a device service name and location. The WSDL syntax provides a grammatical structure and building block for XML documents adopting the following structure:

```
<definition>
  <type></type>
  <message></message>
  <portType></portType>
  <binding></binding>
  <service></service>
</definition>
```

The sample of the WSDL file can be view from the CD in the following directory:

```
..\WS Automation\Chapter 8\ sdHandlingArm.wsdl
```

A description of WSDL elements for the component (*Handling Arm*) is given below:

1.0 Defining XML Version and Encoding style with Component Target Namespace for contained elements

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="sdlHandlingArm"
targetNamespace="http://www.soda-itea2.org/Demonstrator/HandlingArm"
xmlns:tns="http://www.soda-itea2.org/Demonstrator/HandlingArm"
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://www.soda-itea2.org/Demonstrator/wsa.xsd"
xmlns:wdp="http://www.soda-itea2.org/Demonstrator/wdp.xsd"
xmlns:wse="http://www.soda-itea2.org/Demonstrator/wse.xsd"
xmlns:hla="http://www.soda-itea2.org/Demonstrator/HandlingArm"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Note: An illustration of the generation of the unique target namespace where XML elements and attributes refer to their definitions and declarations. This is seen as a reference name (i.e. not an actual specific file location) added to each element to distinguish the name from those similar elements on other components [g11].

1.1 DPWS Elements, Sensor State Names and Invocation Commands in <Types>

```
<types>
<import namespace="http://www.w3.org/2003/05/soap-encoding"/>
<simpleType name="handlingArmStatus">
<restriction base="xsd:string">
<enumeration value="ARM-ERROR"/>
<enumeration value="Downstream-Position"/>
<enumeration value="Move-Sort"/>
<enumeration value="Sort-Position"/>
<enumeration value="Move-Upstream"/>
<enumeration value="Upstream-Position"/>
<enumeration value="Move-Downstream"/>
</restriction>
</simpleType>
<simpleType name="handlingArmAction">
<restriction base="xsd:string">
<enumeration value="Move-Downstream"/>
<enumeration value="Move-Upstream"/>
<enumeration value="Move-Sort"/>
<enumeration value="Status"/>
</restriction>
</simpleType>
<element name="Arm" type="hla:handlingArmStatus"/>
<element name="HandlingAction" type="hla:handlingArmAction"/>
</types>
```

Note: Variable (Arm) states and action (HandlingAction) names of the element are defined with enumeration values by referring to hla:handlingArmStatus type and hla:handlingArmAction type accordingly (see 1.2 below).

1.2 Request and Response message of each element, including sensors and actuators in `<message name>`

```

<message name="handlingCmdRequest">
  <part name="HandlingAction" element="hla:HandlingAction"/>
</message>

<message name="handlingCmdResponse">
  <part name="Arm" element="hla:Arm"/>
</message>

```

Note: This message definition defines the SOAP xml message for input (handlingCmdRequest) and output (handlingCmdResponse) messages of DPWS call functions defined in the `<portType>` tags.

1.3 Defining the DPWS Web Services operation commands (e.g. service invocations, state notifications) of each element in the component by `<portType>`

```

<portType name="handlingArm">
  <operation name="handlingCmd">
    <documentation>Service definition of function hla__handlingCmd</documentation>
    <input message="tns:handlingCmdRequest"/>
    <output message="tns:handlingCmdResponse"/>
  </operation>
</portType>

```

Note: The portType tag is used to define the DPWS call function (hla__handlingCmd) definition with the input and output message attributes. This function is used in the client and server applications (*see Section 8.5*).

1.4 Defining the binding name of each previously defined `<operation>` to the corresponding target namespace of the component in `<binding name>`

```

<binding name="sdI handlingArm" type="tns:handlingArm">
  <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="handlingCmd">
    <SOAP:operation/>
    <input>
      <SOAP:body use="literal" namespace="http://www.soda itca2.org/Demonstrator/I handlingArm"/>
    </input>
    <output>
      <SOAP:body use="literal" namespace="http://www.soda itca2.org/Demonstrator/I handlingArm"/>
    </output>
  </operation>
</binding>

```

Note: The WSDL binding element provides the unique name for a particular DPWS function for each operation hosted on the specific location in the control system. Thus, the functions can be differentiated and unique on the same control system or control device.

1.5 The service name of the component binding to the specific local host port described in <service>

```
<service name="sdHandlingArm">
<documentation>gSOAP 2.6.2 generated service definition</documentation>
<port name="sdHandlingArm" binding="tns:sdlHandlingArm">
<SOAP:address location="http://localhost:80"/>
</port>
</service>
```

Note: This service element is a collection of ports defined in the <binding> tag that exposes a particular binding name and address (port reference).

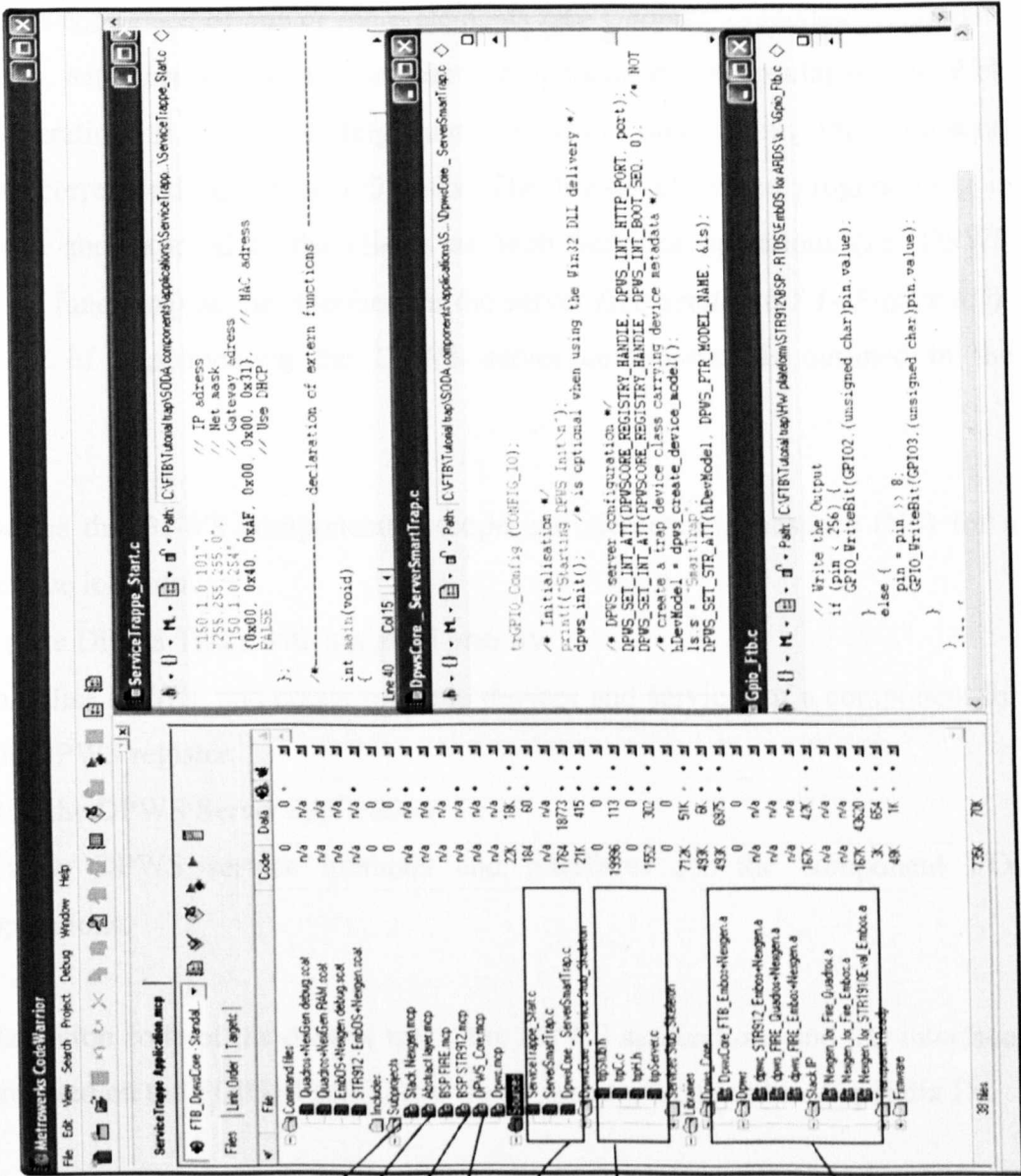
The descriptive details of WSDL can be found in [57]. The building of the client and server applications for the control system, using generated DPWS and gSOAP stub and skeleton files is described in the next section.

8.4 Design Tools

8.4.1 SERVER-Multiple Services on the FTB

The Development Platform of the ARM Real View Development Suite (ARM-RVDS), which has been used to build and deploy the DPWS-enabled components on the FTB's, is illustrated in Figure 8-4.

The skeleton files, generated from the WSDL description by the DPWS-gSOAP compiler toolkit for the DPWS server application, are imported into a project created on the ARM-RVDS, together with the RTOS and TCP/IP stack (sub-projects for the server application of each station). In addition, the DPWS project for the components is uploaded to the FTB control devices for debugging and deployment of the components' control applications through the jTAG probe provided for each FTB.



Stack IP Library

Abstract Layer Library

Real-time OS Library

Dpws Library

Embedded project is composed of

Source files + generic launcher file

Stub/Skeleton files +

marshaling/unmarshaling codes

Project link files

Figure 8-4: ARM Real View Development Suite (ARM- RVDS)

As mentioned earlier, each station contains one or more components, and each component is comprised of one or more elements (*see Chapter 7- section 7.3.2*). The elements (i.e. actuators and sensors) of each component are encapsulated with Web Services operations (i.e. commanding output actuators and reading input sensors) performing corresponding device I/O tasks. The low-level device programming is encapsulated and exposed to the clients as Web Services operations (i.e. DPWS services call functions) as the interface on the server (*see section 8.5.1- Figure 8.5*). The process of implementing the DPWS server components is outlined in the following:

1. Define the DPWS component(s) scope and elements namespace (NS) for a service location.
2. Create DPWS Tasks with assigned priority.
3. Initialise DPWS, and create multiple devices and services of a component for the DPWS register.
4. Run the DPWS Server application.
5. Utilise DPWS service methods and interfaces for the component I/Os operations.

The initialisation code of the control tasks, the DPWS applications and I/O interfaces as implemented on the FORD-FESTO test rig servers can be found in Appendix D.

From the above discussions it can be appreciated that implementing embedded WS on devices requires a significant amount of extra functionality to be associated with each device on top of the basic state machine operation of individual devices. From the memory perspective the compiled size of the RTOS and DPWS runtime library (i.e. the software layers to support the control application presented in section 8.2.3) on the FTB device have been derived as follows:

FTB resources

The RTOS, TCP/IP stack, abstract layer and DPWS layer are located in the FTB on board flash (ROM) memory (capacity: 512 Kbytes), using the following memory resource:

Lib C	22.7	Kb
EmbOS (RTOS)	19.4	Kb
Nexgen (TCP/IP)	85.2	Kb
Abstract Layer	4.8	Kb
DPWS Layer	<u>239.2</u>	Kb
	371.3	Kb

(Resource: Schneider Electric)

However, a server (i.e. control) application for a single DPWS component, which contains 1 actuator and 2-3 sensors, has a calculated memory footprint of **39.5 Kb**. Note: if there are three DPWS control components (i.e. server applications) on the FTB controller, the memory consumed = $371.3 + (3 \times 39.5) = 489.8$ **Kb** in total, with only **22.2 Kb** remaining. In conclusion each FTB could hold up to 3 components with the current flash memory capability.

8.5 Industrial Demonstration

The FORD-FESTO test rig demonstration developed by the author has been undertaken as part of the SOCRADES EU research project at the MSI, Loughborough. This is the first industrial demonstration of the realisation of Web Services in automation systems. The WS based control system as detailed above has been integrated with business applications (e.g. SAPxMII - Manufacturing Integration and Intelligence) and a broadcaster application (e.g. state propagation software that provides integration with a distributed HMI console) for process monitoring and data acquisition (*see Chapter 9- section 9.5 for details*).

It is important to note that the majority of author's research has contributed to the development, evaluation and analysis of the WS embedded control system on the FORD-FESTO test rig, including the collaborative work in interfacing the Web Services enabled devices with external business / monitoring applications. The

implementation details of the business and monitoring Web Services implementations are outside of the scope of the author's research and will be published elsewhere.

The following sections provide the implementation details of the complete WS implementation for the FORD-FESTO test rig, in order to demonstrate the full working capability of the system.

8.5.1 Web Services- based Control System Integration

In order to achieve the requirements of agile automation systems as outlined in Chapter 2, the aim of the Web Services system design is to develop the corresponding tools and methods, in order to support the development of reconfigurable automation systems and interoperable network-enabled manufacturing collaboration between business, control and shop-floor systems.

As presented in Chapter 3, the adoption of Web Services at the device level benefits the manufacturing system in terms of ease of system integration by allowing business and process applications to be directly integrated to the distributed WS functionalities defined at the device (i.e. component) level. The main reasons for adopting Web Services at the device level in this research is not only the seamless integration capability but also the perceived ease of design, analysis, deployment and re-configurability of integrated automation systems.

In the design of reconfigurable WS-based automation systems, the design of software components and the requirements of end-users to influence component reconfiguration must be taken into consideration. As demonstrated by S.M. Lee in his component-based (CB) design approach [33], I/O device code must be pre-programmed and encapsulated within call functions whose interfaces are presented to control builders as the means to instantiate device operations. System builders only need concern themselves with commissioning the higher-level automation *process* by defining device interlocks and sequences (i.e. device state logic conditions).

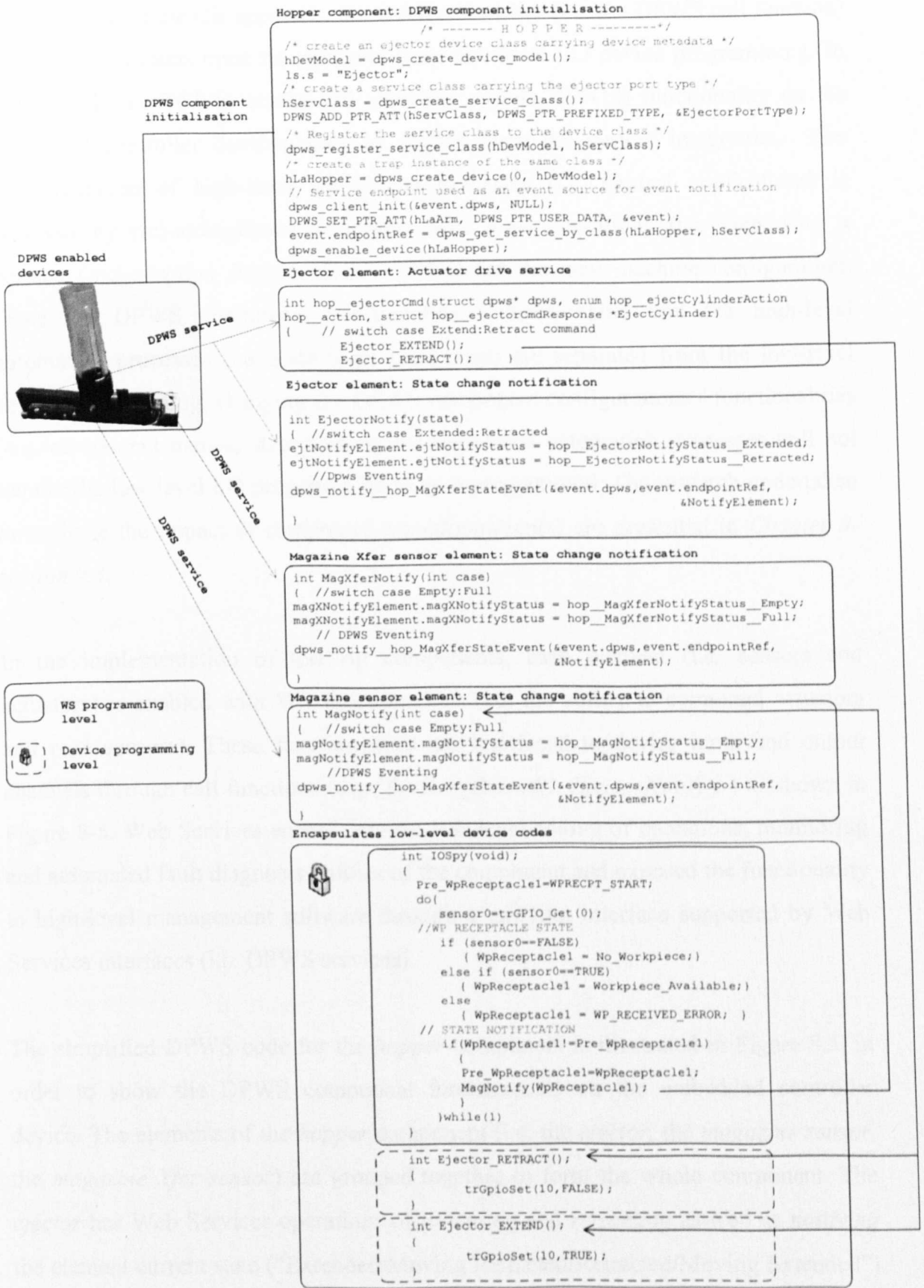


Figure 8-5: Control Code for the DPWS Enabled Component

(For the programming details, please see the attached CD in

In the context of the CB approach, the WS functionality (i.e. the DPWS call function) of device operations must be separated from low-level I/O device programming. In addition, device TCP/IP stacks and operating system (RTOS) functionality on the embedded controller devices must be hidden from the system integrators. The reconfiguration of high-level automation processes (i.e. control applications) is achieved by (re)-arrangement of device state logic conditions (i.e. *employing a Service Orchestration Engine- Section 8.5.2*) for the new machine configuration. Since the DPWS component configuration / functionality and the high-level automation processes (i.e. state logic conditions) are separated from the low-level device programming, changing the DPWS component configurations / functionalities (e.g. component names, state names) and high-level automation processes will not require the low-level I/O programming to be reprogrammed. The research undertaken to evaluate the impact of component reconfiguration(s) are presented in *Chapter 9-section 9.3*.

In the implementation of test rig components, each element (i.e. sensors and actuators) is enabled with WS functionalities (i.e. the ability to command actuators and read sensors). These functionalities are interfaced to device input and output channels through call functions (e.g., *Ejector_Extend()*, *EjectorNotify()*) as shown in Figure 8-5. Web Services encapsulate the low-level coding of operations, monitoring and automated fault diagnosis utilities of the component and exposed the functionality to high-level management software through a unifying interface supported by Web Services interfaces (i.e. DPWS services).

The simplified DPWS code for the *hopper* component is illustrated in Figure 8.5, in order to show the DPWS component fundamentals on the embedded controller device. The elements of the *hopper* component (i.e. the *ejector*, the *magazine sensor*, the *magazine Xfer sensor*) are grouped together to form the whole component. The *ejector* has Web Services operations of *extending* and *retracting* as well as *notifying* the element current state (“Extended/Moving Retracted/Retracted/Moving Extended”) and the two sensors provide the Web Services operation of identifying the workpiece availability in the hopper unit (“Empty / Full”). The detailed representation of component source code is given in Appendix D. In addition to the element state notification, each element is also assigned an internal error state which presents a miss

positioning situation diagnosed by defined sensor conditions in the I/O watchdog routine (see section 8.5.3-2). This *error* state is used as a preliminary report presented to higher control levels for diagnostics and maintenance through the Web Services state notification.

In this research, the FORD-FESTO test rig has provided an opportunity to evaluate the migration paths for the adoption of WS control systems for industrial automation. The overall aim is to replace eventually the existing centralised PC and PLC-based automation systems with systems composed of fully distributed embedded devices. Whilst WS on the control devices have not been fully evaluated with regards to reliability and performance, they fully support integration with other engineering applications (e.g. HMI, business applications and control system design tools) used in industry and it is considered a reasonable assumption that WS enabled devices will be able to co-exist and operate alongside current PLC and PC control devices. Therefore, the migration path for WS automation that has been initiated in this research with the execution and decision making (based on state logic conditions) being initially supported the PC and the PC acting as a service orchestration engine running distributed control applications provides a useful stepping stone towards fully distributed heterogeneous WS based automation. In the current automation model the PC based service orchestration engine acts as a client to a service on the FTB control device for device operation. The details of the test rig operation and client-server event interaction are presented in the following section.

8.5.2 Service Orchestration Engine

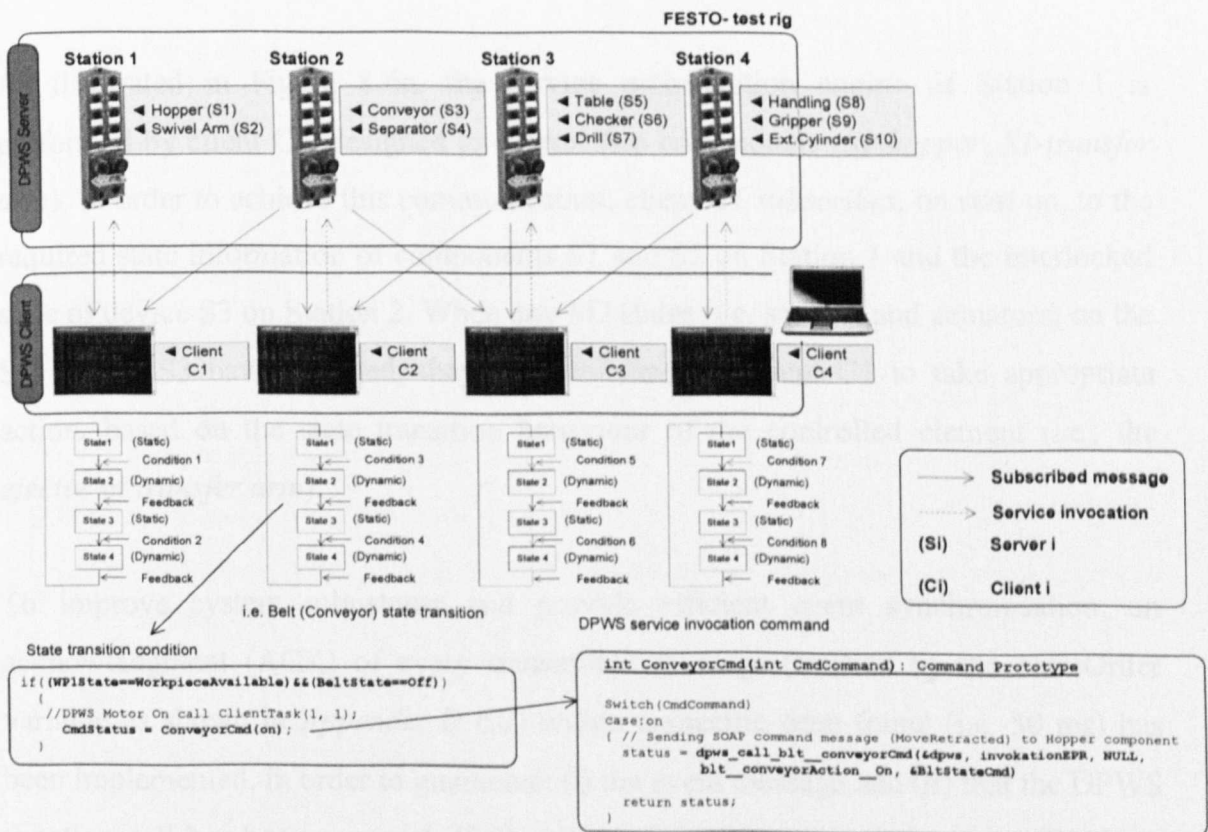
A typical feature of manufacturing systems is that several processes are composed and executed in given sequences in order to create complex higher-level processes. This pattern is repeated at several levels via the composition of field devices to create machines, the composition of machines to create work cells and lines, and the composition of work cells and lines to create manufacturing systems and factories [14].

The composition of component sequences and synchronisations at the device (i.e. component) level, referred to as “*Service Orchestration*”, determines the operation of the complete machine application and must accurately reflect the process workflow.

The *Service Orchestration Engine* is implemented within the application logic (i.e. the machine finite state machine), and *orchestrates* the services on the components.

In the discrete event-based automation environment, the system is conceived as a composition of interacting components, defined by means of state transition conditions and interlocked variables among dependent devices. The role of the process engineer is to manage the task of implementing the control logic code that forms the real manufacturing operation i.e. the complete machine application. The activity involves the translation of the process machining sequences (see Chapter 9-section 9.3), to interlocks on corresponding embedded hardware devices.

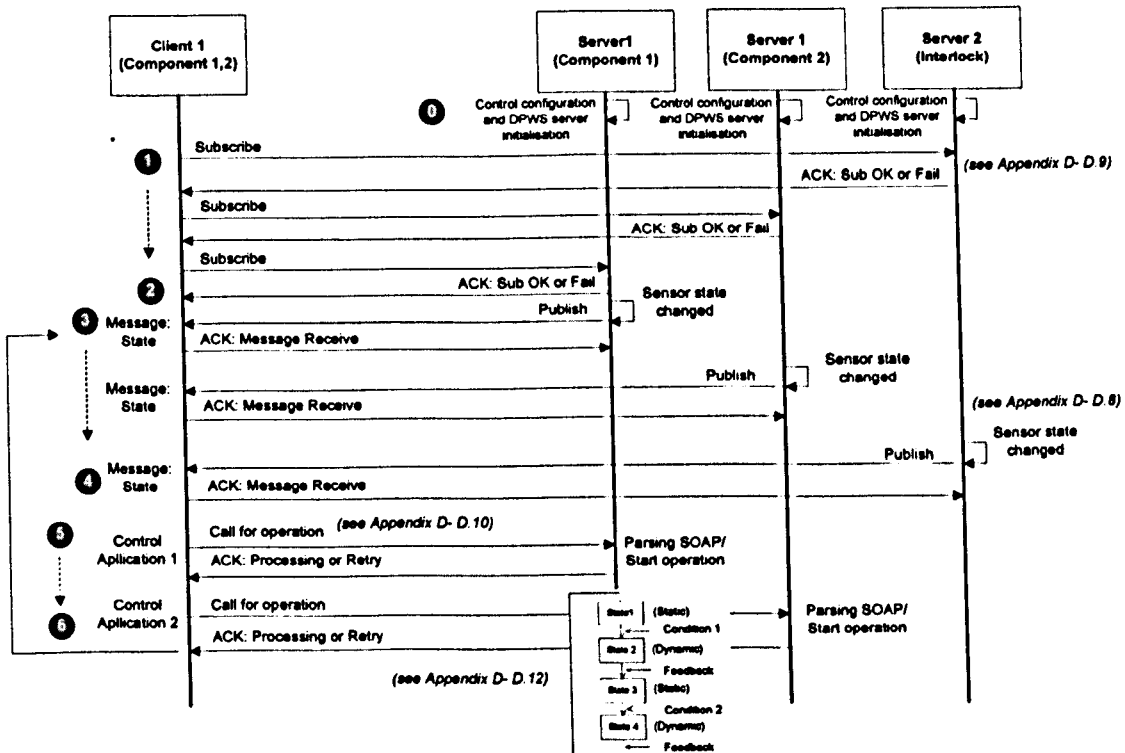
The control communication architecture in this research is based on the Publish and Subscribe approach to provide an event-driven architecture. This architecture is seen as the best suited to automation systems with good performance and utilisation of the network, especially on limited resource embedded devices (see section 8.4.1).



a) Execution of the Distributed Control Nodes

(For the programming details of the implemented client application, please see the attached CD in

.. \WS Automation\Chapter 8\Dpws_ClientStation1.c)



b) Operation of Test Rig Control Elements

Figure 8-6: Distributed Web Services-based Control System

As illustrated in Figure 8-6a, the service orchestration engine of Station 1 is performed by client C1, designed to control two components (*S1-hopper*, *S2-transfer arm*). In order to achieve this communication, client C1 *subscribes*, on start up, to the required state information of components S1 and S2 on Station 1 and the interlocked state of device S3 on Station 2. When any I/O states (i.e. sensors and actuators) on the S1, S2, or S3 have changed, they are published for client C1 to take appropriate action, based on the state transition behaviour of the controlled element (i.e., the *ejector* or *transfer arm*).

To improve system robustness and provide efficient event synchronisation, an acknowledgment (ACK) of every transmitted message (tracked by the NumOrder variable as shown in *Appendix D 6.2*) within a specific time frame (i.e. 50 ms) has been implemented, in order to guarantee: (i) the event message and (ii) that the DPWS function call has been received. If the acknowledgment message is not received, an alarm will be raised and handled. In this case, the automatic halt utility and fail-safe routine, implemented on the server and the client applications, will be activated to suspend specific components and client execution control applications from further

operations that may cause damage to the control system. Cause-Effect Investigations and maintenance can then be carried out. In addition to fault detection, I/O device miss-operation and error handling system is required for the safety of the control system operation. This *error-handling* scenario is discussed in section 8.5.3-2.

The instance of the client and the server synchronisation model for the machine application during “Start Up” and “Operation” is illustrated by the use case in Figure 8-6b.

The component operations and the service orchestration implemented on the servers and clients, proceed in the following way:

Step 0-Server applications of components start up. The controller operating system (i.e. RTOS), hardware configurations (i.e. I/O channels, timer, IP and MAC address) and DPWS of components initialised (*see section 8.5.1- Figure 8.5*).

Step 1-The client application (e.g. C1) starts up. All required components presented by server applications (e.g. S1, S2 and S3) are located for the state information and services invocation. In this case, the *probe* message has been broadcast over a network and only the matched component replies to the client with its location. The client then subscribes to the specific components for state information and also invokes the DPWS service to be run during the control session.

Step 2-Input or output channels on components change their states. Server applications publish state information to subscribers as states update. The client acknowledges publishers with a return ACK message.

Step 3-State information of the interlocking device is sent for the update on the client application. The client acknowledges publishers with a return ACK message.

Step 4-The control logic application on the client interacts with the operation on the server based on state information and transformation conditions of components.

Step 5-The component state transition conditions have been met. The client invokes the control operation on the server by sending the DPWS message (i.e. a SOAP message) to execute the DPWS call function running on the control device server.

The SOAP message regarding to the test rig operation can be found in Appendix C. It is observed that the sequence of control tasks depends upon the current state of work pieces, devices and interlocks, which need to meet the state transition conditions of the component so that the DPWS client invokes the operation on the component. Steps 2-5 will be run repeatedly through out a complete machine cycle. In this implementation, the client logs the state of all elements in the components during the operation in the log file which can be used for system analysis and maintenance. In addition to the service orchestration, the component state and operating information are broadcast to the higher-level control for process monitoring and business management. Details of this business application integration are presented in the following section.

8.5.3 Business Application Integration

Currently, intelligent shop-floor systems using distributed embedded devices concentrate the programming of behaviour and intelligence on a handful of large monolithic computing resources accompanied by large numbers of distributed devices. Intelligence and behaviour are tailored and individually programmed for each application [47, 48] and, within this manufacturing environment, it has been reported by [46] that integration of an individual control system is achieved via the various sets of interfaces and drivers required to connect to automation devices individually. As a consequence, there is limited co-operation between the business and shop-floor levels, due to a proliferation of inconsistent interfaces that prevent the integration of the complete range of distributed automation devices. This research aims to create improved co-operation between shop floor and business, through the use of WS-enabled automation systems. It is envisaged that WS will improve the transparency of the connection between back-end systems (ERP systems) and shop floor production. A detailed discussion and evaluation is presented in Chapter 9- section 9.6.

The business integration demonstration has been undertaken in collaboration with SAP GmbH. The business application integration has been demonstrated with the SAP's SAPxMII (Manufacturing Integration and Intelligence) application for shop-floor activity monitoring and fault diagnosis. This business application has been designed to allow business systems to obtain a real-time view of industrial processes, supporting business activity monitoring, maintenance optimisation and overall equipment effectiveness. In the evaluation, the test rig provides the SAPxMII application with control device data (i.e. real-time device states), all work piece status, the number of processing units and the machine operational and idle times for data manipulation and analysis.

8.5.3-1 SAP xMII and WS test rig Integration Platform

The SAP xMII provides functionality which allows users to collate data from multiple systems via a simple Web browser interface. The software does not need to be managed at the client site (i.e. the broadcaster in this case) and there is no requirement for complex data warehousing or data models [45].

The SAP xMII application does not currently support the complete of DPWS protocol, such as the *discovery* and *eventing* features reported by [47] and [45]. Hence, the SAP application was configured to interact with the FORD-FESTO test rig using the *service orchestration engine*. The data within SAPxMII are populated from the *state publication* utility in the *service orchestration engine* via a TCP/IP connection. Likewise, the SAP application is able to invoke services on the FORD-FESTO test rig through the *service invocator* on the service orchestration engine. In the evaluation trials, the state publication and service invocator had direct communication with the test rig via a DPWS interface that enabled them to *subscribe* to the required devices' states and to *invoke* the operations on these devices. It is noted that the DPWS interface of the *service invocator* and the *state publication* is achieved by using the stub (client) files, generated from the WSDL definitions for the component (see Chapter 6- section 6.9.2).

Business integration through single Web Services interfaces is ongoing with the project collaborators. Note: this research has implemented the integration of the *service orchestration engine* and the SAP xMII application (as provided by the WS application interface defined in Figure 8-7) using integration middleware enabled by WS rather direct integration to the test rig via DPWS interface.

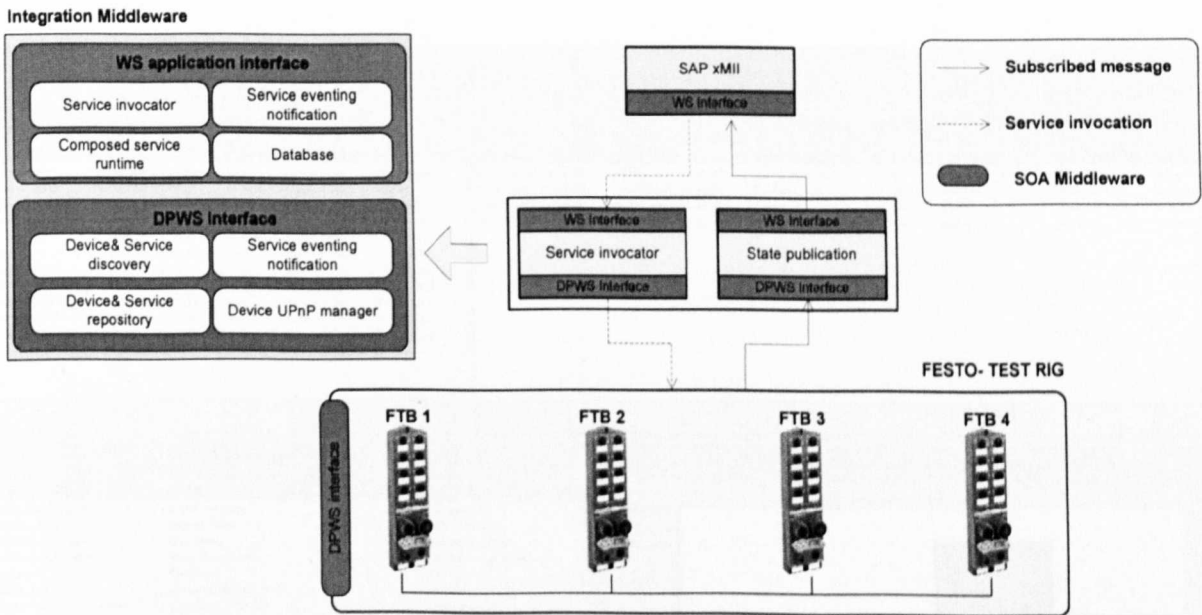


Figure 8-7: Business Integration Scenario with the SAP xMII application

It can be observed in Figure 8.7 that the middleware operates on two levels: (i) at the device level interfaced to DPWS-enabled devices and (ii) at the application level, interfaced to Web Services applications. The functionality of the DPWS interface are represented by the DPWS protocol, (see Chapter 4- section 4.9.2), and the functionality of the WS interface has been tailored for the SAP integration. The interface to SAP xMII has been developed as a service point for remote applications to invoke services on and receive notifications of the performance of the FORD-FESTO test rig. However, the implementation of the SOA middleware proposed for seamless integration is presented in Chapter 9- section 9.6. The integration architecture has been modelled on the SOCRADES middleware architecture [45].

8.5.3-2 The SAP xMII Process Monitoring and Error Diagnostic Application

In the business-test rig integration evaluation, the capture of process activities and operational performance data has been demonstrated with the SAP xMII application, which provides graphical visualisations of device data, the current status of work piece information and machine performance data in real-time, as shown in Figure 8-8.

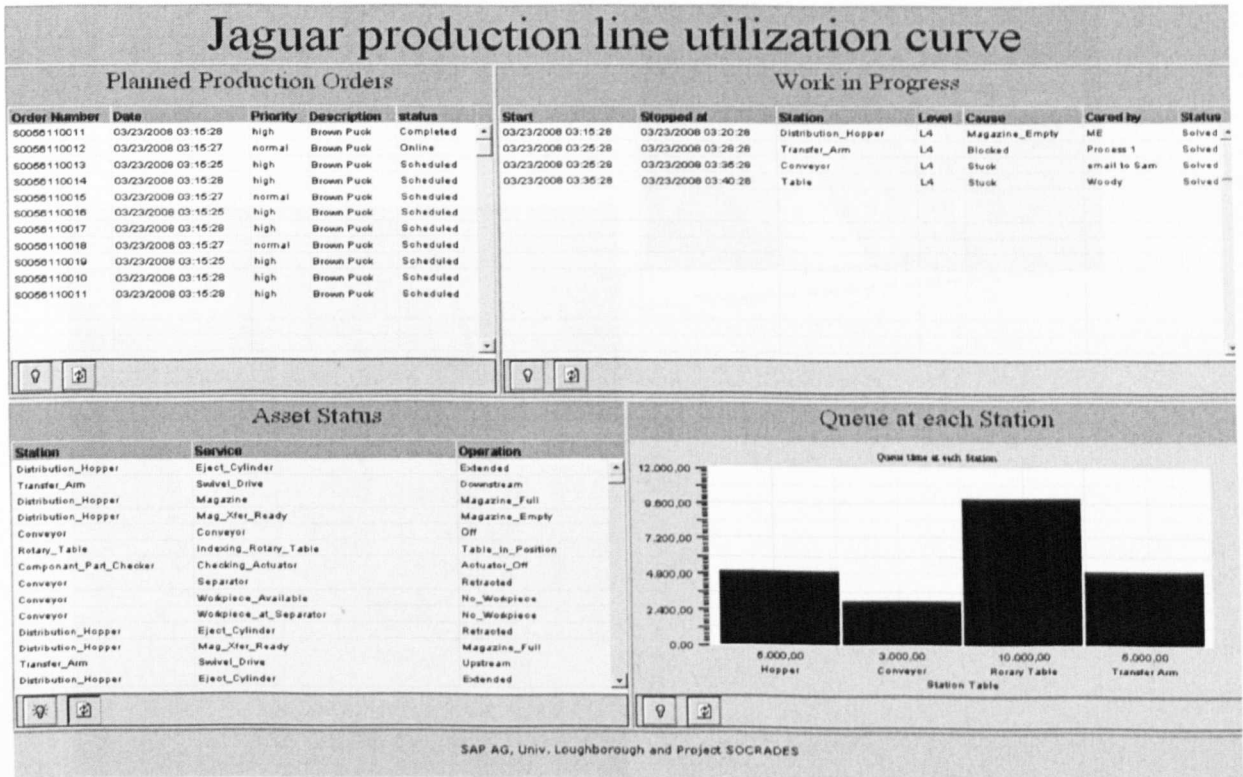


Figure 8-8: SAP- Production Line Monitoring Utility using SAP xMII

The operational status of the test rig is provided by the state variables determined from the logical state of local sensors. Based on the *eventing* communication model, these variables are transmitted to the SAP application via the broadcaster as a state change basis. The SAP application can be configured to extract shop floor data into useful business contexts, so that the end-user may obtain an appropriate business oriented global view of the process.

The business and manufacturing management levels can hence react accordingly and in a timely fashion to plant-floor information (e.g. machine up/down time, production capacity, and current work in process (WIP)). In addition to the process-monitoring evaluation detailed above, the FORD-FESTO test rig has been implemented with a

fault diagnosis and error handling support system that has been integrated with the SAP SAP xMII application. A test scenario has been demonstrated in which a faulty sensor of the *transfer arm* unit on station 4, where manual intervention cause the sensor to fail occasionally to detect and stop the arm moving into the correct position. This error resulted in the arm being slightly out of alignment with the work piece slot. The test scenario is shown in Figure 8-9.

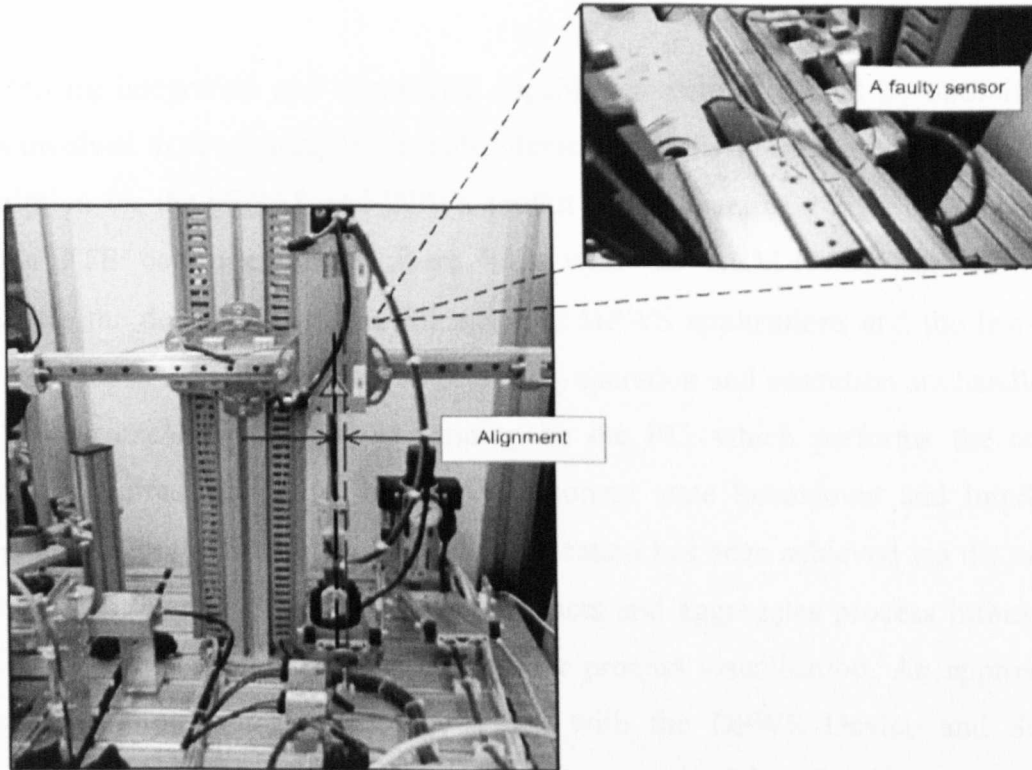


Figure 8-9: Handling Arm Fault Diagnostic Test Scenario

(For the programming details of error diagnostic, please see the attached CD in

..\WS Automation\Chapter 8\Dpws_ServerStation4.c)

Where this fault was injected into the system, the errors were logged locally on the test rig. When the number of these errors reached 10, the error summary was reported to the SAP application via the *state publication utility*, in order to bring attention to the error and enable appropriate action to be determined by personnel monitoring the SAP application. The SAP application can halt the faulty unit by passing the *WS stop* command through the *service invocator*.

8.6 Conclusion

The details of how the full implementation of the WS automation platform was achieved on the FORD-FESTO test rig, demonstrating the full potential of using Web Services technology in industry have been presented in this Chapter. Descriptions of the hardware and software design platforms for the embedded controller, in particular the FTB (ARM 9) device, have been outlined. This design platform is also generically applicable to any other embedded devices.

The test rig integration and operational capabilities were presented by outlining the steps involved in developing WS enabled devices along with the design of the WSDL description for the gSOAP and DPWS toolkit code generator. Multiple components on the FTB controller device were built with the ARM-RVDS toolkit, which facilitates the debugging and downloading of DPWS applications and the low-level device codes to the target devices. The test rig operation and execution are handled by the *service orchestration engine* running on the PC, which performs the control operations corresponding to designed component state behaviours and interlocks. Integration with the SAPx MII business application has been achieved via the *service execution engine*. The SAP application extracts and aggregates process information and transforms it into a business context for process visualisation. An approach to shop floor integration has been proposed with the DPWS Device and Service Application Interface utilised in the developed SOA Middleware. Direct communication to devices is achieved via the DPWS interface, and the WS application interface has been provided for non-DPWS enabled applications to allow them to connect to the WS based control systems.

CHAPTER 9

Evaluation and Discussion

The *implementation* details of the Web Services-based automation system for the industrial-based FORD-FESTO test rig were outlined in the previous chapter. In this chapter, an *analysis* of the Web Services-based automation system approach will be undertaken. Analysis and discussion is centred on an in-depth discussion of the machine performance, in relation to the TCP/IP communication mechanisms, packet structure and sizes and Web Services message delivery time. The requirements of agile automation systems by the end-user, as defined in *Chapter 6- section 6.4.2*, are initially assessed by determining the degree of reusability, re-configurability, and business integration of the automation system.

9.1 Problem Statement

The implementation of the WS control approach has been demonstrated on the FORD-FESTO test rig. The drivers of research require that the WS approach is also assessed in terms of performance and suitability (i.e. support for agility) for adoption by industry. The main research questions addressed throughout this chapter are:

1. The quantitative measures of TCP/IP network determinism, bandwidth and utilisation capture the performance and reliability of the implemented control system.
2. Does the WS performance, in terms of Ethernet packet speed and I/O response time, meet the soft-real time (under 30 ms) requirement for the automotive powertrain domain?
3. What is the relative cost and performance metric of the WS control system in comparison with PLC based and LonWorks based control systems?
4. In the key evaluation areas for agile manufacturing, what is the degree of reusability, reconfigurability and business integration in the implemented WS-based control system in comparison with PLC based and Lonwork control system?

9.2 DPWS Performance Analysis

9.2.1 DPWS- SOAP Message Structure

In order to determine the impact of the DPWS-SOAP protocol on system performance a protocol analyser was employed. The protocol packet analyser used in this test was the CoCoaPacketAnalyzer. This tool offers the ability to capture messages on a LAN network and transform the packet from binary into both Hex form and a readable format (i.e. the SOAP Message). For the purposes of the test, the CoCoaPacketAnalyzer utility was installed on the PC, running the *service orchestration engine* for the control operation. The analyser captures every DPWS message coming into and leaving the PC (Network card) during the rig operation.

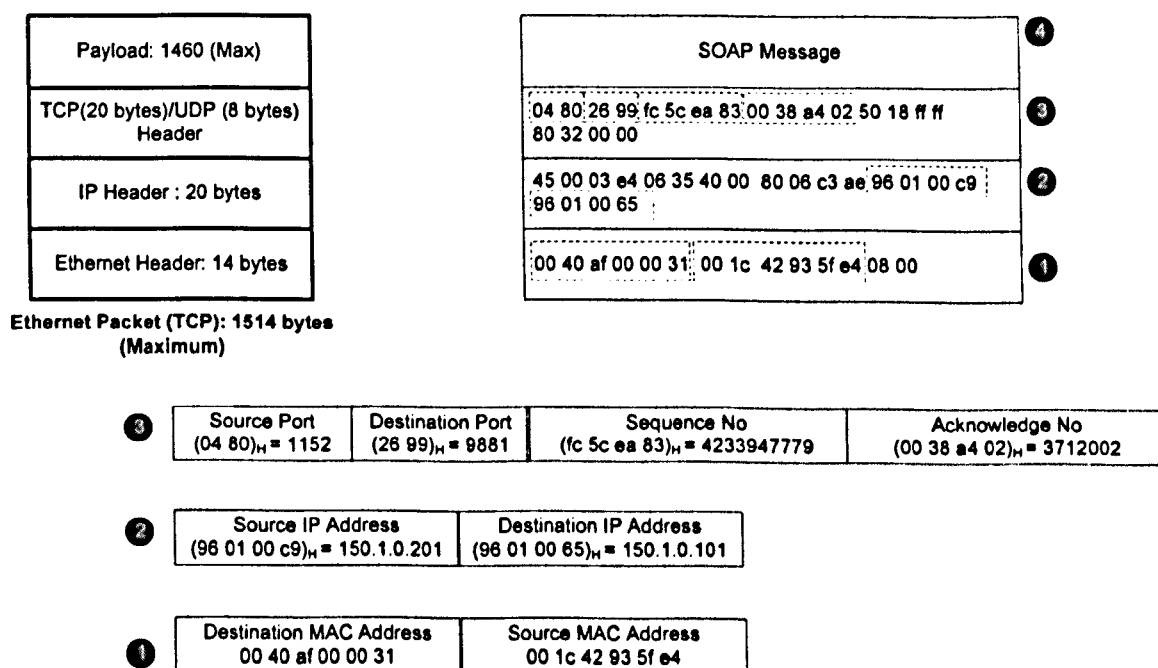


Figure 9-1: Ethernet TCP/UDP Packet Structure

The captured packet in the TCP/IP structure (*Chapter 4- section 4.4*) is illustrated in Figure 9-1. This packet is divided into 4 layers and the content of each layer conforms to the TCP/IP standard. Firstly, the data or payload (*see Figure 9.1- (4)*) i.e. the SOAP message (i.e. the XML based content), made up of source, destination port, sequence and acknowledgement number (*see Chapter 9- section 9.2.2*) is represented at the TCP/UDP layer (*see Figure 9.1- (3)*). The source and destination IP addresses are then added to the message (*see in Figure 9.1- (2)*). The destination of the packet and

source are located precisely through the means of the destination and source MAC address, added at the final layer (*see Figure 9.1- (1)*) before the packet is transmitted. It should be noted that the payload size has a maximum limit 1460 bytes and if the data load exceeds this limit, then it needs to be split into 2 or more packets, in order to fit into the Ethernet datagram.

The content of a SOAP message packet is in XML format which is decoded by the receiver to extract the information into an appropriate format (*e.g. gSOAP see Chapter 6- section 6.9.2*). With regards to the rig operation, the DPWS messages comprise: *discovery probe* (via UDP); *probe match reply* (via UDP); *request metadata* (via TCP); *return metadata* (via TCP); *device status* (via TCP); *service invocation command* (via TCP) and *service acknowledgment* (via TCP). The full description of these messages can be found in Appendix C.

The XML prologue `<?xml version="1.0" encoding="UTF-8" ?>` contains only an XML declaration, specifying the XML version and the character encoding of the XML message [g16], whilst the SOAP envelope tag `<SOAP-ENV:Envelope ... >` in the request and reply message of the device (component) specifies that the SOAP messages' encoding styles follow the schema and relevant WS methods [g9] as defined in the WSDL description. The SOAP envelope contains the SOAP header and the SOAP body. The SOAP Header contains application-specific information regarding the SOAP message [53] and, in this DPWS application, the header contains the following elements:

- `<wsa:To>` the destination (using a namespace or specific URI ⁽⁷⁾)`</wsa:To>`
- `<wsa:RelatesTo>` the source (a namespace or specific URI) `</wsa:RelatesTo>`
- `<wsa:Action>`the action namespace `</wsa:Action>`
- `<wsa:MessageID>`the sent message ID using UUID schema`</wsa:MessageID>`

The SOAP body is the content of the Web Services message. This encapsulates a single method tag i.e. the name of the method call or the device state information of the reply message as samples shown below.

7- Uniform Resource Identifier (URI) is a string of characters used to identify or name a resource on the Internet (<http://en.wikipedia.org/wiki/URI>). In the SOAP message obtained from this Web services implementation, the URI is a combination of IP address: Port/UUID (Please refer to Appendix C for examples).

<arm:armCmd> the required action (the service name)**</arm:armCmd>**

<arm:armCmdResponse> the return content**</arm:armCmdResponse>**

The method tag is typically prefixed by the component target namespace (for example, **arm:**, **blt:**, **hop:**) for the service name that ensures uniqueness. These details are described in [g16].

The sending and responding probe message contains slightly different attributes than the SOAP body of SOAP method calls (i.e. service invocations, reading device states). The attribute elements are as follows:

<wsa: EndpointReference>

the endpoint reference using UUID ⁽⁸⁾

</wsa: EndpointReference>

< wsa:Scope> the device name **< /wsa:Scope>**

< wsa:Type> the element name **< /wsa:Type>**

<wsa:Xaddress>the device address created from IP/Port/MAC**< wsa:Xaddress>**

This SOAP message structure, predefined with the WSDL description, is encoded and decoded by the stub and skeleton files on the implemented DPWS application, in order to extract the required information into the format processed by the control application.

The next section outlines the TCP/IP synchronisation approach for sending and receiving SOAP messages during machine operation. The network performance is also detailed.

9.2.2 Ethernet TCP/IP network communication

The DPWS communication in the FORD-FESTO test rig, based on the publish/subscribe model, is required to perform two DPWS functions: (1) Service invocation and (2) Event notification. The packet synchronisation details for these DPWS functions are presented in Tables 9-1 and 9-2 respectively.

8- Universally Unique Identifier (UUID) is used to uniquely identify information which will never be unintentionally duplicated by others for anything else. The UUID is formed by 32 hexadecimal digits, displayed in 5 groups separated by hyphens. In this implementation, the UUID is based on the MAC address generation scheme (Please refer to http://en.wikipedia.org/wiki/Universally_Unique_Identifier and Appendix C for examples)

Table 9-1: DPWS Server Operation Invocation- Ethernet TCP/IP Network Communication

Packet	Packet size (Bytes)	Data load (Bytes)	Packet received time (s)	Initial time (ms)	Time different between packets (ms)	Event	Seq No. (s)	Ack No. (s)	Diagram		Packet Synchronization (Seq,Act)	Ack Flag	Fin Flag
									HOST A	HOST B			
788	78	0	56.198956	0	0	Host A sends a TCP synchronize packet to Host B to initiate a connection	3976 +1	0	send	t+1	Host B received the packet from Host A to start the connection	0	0
789	60	0	56.199326	0.37	0.37	Host B replies Host A to acknowledge the request on the connection	6001 +1	3977	recv	t+3	Host B replies Host A with New Seq and Ack=Seq(A)+1	1	0
790	60	0	56.200026	1.07	0.70	Host A sends Acknowledge to Host B; The connection is established	3977 0: Data load	6002	send	t+4	Host A replies with Seq= Ack(B) and Ack= Seq(B)+1	1	0
791	669	915	56.200038	1.062	0.012	Host A sends a SOAP message to Host B	3977 915: Data load	6002	send	t+6	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B	1	0
792	60	0	56.200042	1.066	0.004	Host A sends another packet to Host B as the one to be acknowledged and also a request to close the connection from Host A to B	4892 +1	6002	send	t+8	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B	1	1
793	60	0	56.200083	1.874	0.788	Message received; Host B acknowledges Host A	6002 0: Data load	4893	recv	t+11	Host B acknowledges Host A with Seq= Ack(A) and Ack= Seq(A)+1	1	0
794	810	756	56.209668	10.912	9.038	Host B starts sending a response SOAP message to Host A and confirm the connection closed from Host A to B	6002 756: Data load	4893	send	t+12	The next sent a message from B to A with the Seq = Seq(B)+ data load and the same Ack	1	0
795	60	0	56.209675	10.919	0.007	Host B sends a packet to Host A that requests a termination of the connection from Host B to A	6758 +1	4893	recv	t+15	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A	1	1
796	60	0	56.212492	13.536	2.617	Message received; Host A acknowledges Host B to terminate the connection from Host B to A	4893	6759	send	t+16	Message received: Host A replies B with Seq= Ack(B) and Ack= Seq(B)+1	1	0
						Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection	8603	0	send	t+18	Generate the new Seq and set Ack=0	0	0

(b)- For the simplicity of demonstrating the packet synchronization, packet sequence number (Seq No.) and acknowledgment number (Ack No.) are only displayed by the last 4 digits of a full number.
 (c)- ACKnowledgment flag set to 1 is used to notify the successful receipt of packets.
 (d)- FINished flag set to 1 is used to acknowledge the request of a connection (from a requester to a receiver) termination. It is reset to 0 when the termination process is completed.

(The raw data of Table 9-1 can be viewed from the CD in

..\WS Automation\Chapter 9\Table 9.1 raw data.pcap)

Table 9-2: DPWS Server State Notification- Ethernet TCP/IP Network Communication

Packet	Packet size (Bytes)	Data load size (Bytes)	Packet received time (s)	Initial time (ms)	Time different between packets (ms)	Event	Seq No.	Ack No.	Diagram		Packet Synchronization (Seq,Act)	Ack Flag	Fin Flag
									HOST B	HOST A			
15	60	0	3.804595	0	0	Host B sends a TCP synchronize packet to Host A to initiate a connection	4001 +1	0	send t-1	recv t-1	Host A received the packet from Host B to start the connection	0	0
16	60	0	3.804969	0.374	0.374	Host A replies Host B to acknowledge the request on the connection	0225 +1	4002	recv t-3	send t-2	Host A replies Host B with New Seq and Ack=Seq(A)+1	1	0
17	60	0	3.805289	0.694	0.32	Host B sends an acknowledgment to Host A. The connection is established	4002	0226	send t-4	recv t-5	Host B replies with Seq= Ack(A) and Ack= Seq(A)+1	1	0
18	794	740	3.807433	2.838	2.144	Host B sends a SOAP message to Host A	0: Data load	0226	send t-6	recv t-7	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A	1	0
							740: Data load	0226	send t-8	recv t-9	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A	1	1
20	60	0	3.808047	3.452	0.609	Message received. Host A acknowledges Host B	0226	4743	recv t-11	send t-10	Host A acknowledges Host B with Seq= Ack(A) and Ack= Seq(A)+1	1	0
21	168	114	3.810723	6.128	2.676	Host A sends another packet to Host B to confirm the closing connection from Host B to A	0: Data load	4743	recv t-13	send t-12	The next sent a message from A to B with the Seq = Seq(A)+ data load and the same Ack	1	0
							114: Data load	4743	recv t-15	send t-14	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B	1	1
23	60	0	3.811115	6.520	0.000379	Message received. Host B acknowledges Host A to terminate the connection from Host A to B	4743	0	send t-16	recv t-17	Message received. Host B replies A with Seq= Ack(A) and Ack= 0	1	0
						Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection	8001	0	send t-18	recv t-18	Generate the new Seq and set Ack=0	0	0

(The raw data of Table 9-1 can be viewed from the CD in

..\WS Automation\Chapter 9\Table 9.2 raw data.pcap)

9.2.2-1 TCP/IP Communication Approach

Details of Ethernet packet synchronisation, as captured by the protocol analyzer during the test-rig operation of the *service invocation* method, are presented in Table 9-1. The *service orchestration engine* running on the PC (Host A) calls the provided service on the FTB (Host B) to operate the actuator. The synchronisation of sending and receiving packets between Host A and Host B to achieve this *service invocation* over the Ethernet network is represented by: (i) establishing the connection between hosts by means of the three-way handshake approach [9] (Packet 788th-790th) and (ii) sending and receiving the packets (Packet 791st- 796th) as the mechanism provided by the TCP byte oriented sequencing protocols [82].

First step: The Three-way Handshake

The three-way handshake mechanism [g38] is designed so that two systems attempting to initiate a connection for communication can negotiate one connection at a time independently of each other. The TCP 3-way handshake is associated with the sequence and acknowledgment number of the message, and assures that the message is transmitted and received in the correct order. As illustrated in Table 9-1,

- Host A negotiates the connection with Host B by sending the packet (packet 788th) with a TCP header, which is set *Syn Flag = 1*, a unique sequence number (*Seq No*) and acknowledgement No. (*Ack No*) = 0 to initiate the connection.
- Host B replies (packet 789th) with the synchronised sequence (newly generated *Seq No*) and acknowledgement number received from Host A (*Ack No (received) + 1*).
- Finally, host A sends the third packet (packet 790th) to host B with the synchronised sequence: *Seq No (sent) = Ack No (received)* and acknowledgement number: *Ack No (sent) = Seq No (received) + 1* in order to establish the connection.

Second step: The TCP byte-orientated Sequencing Protocols

When connected,

- Host A starts transmitting the message (packet 791st) with the Sequence and Acknowledgement numbers, to ensure that message synchronisation is in the correct order (a record of the above numbers are kept in the host, prior to receiving the acknowledgement back from the receiver). After acknowledgment, the packets are discarded, and with the TCP scheme, the sequence number of the next sent message will feature the sequence number of the previously sent message plus payload: $\text{Seq No (sent)} = \text{Seq No (received)} + \text{data load}$ and acknowledgement number remain the same: $\text{Ack No (sent)} = \text{Ack No (received)}$, as demonstrated in packet 790th-792nd.
- Having sent the message (packet 792nd), the sender negotiates the termination of the one-way connection from Host A to Host B by sending the next packet (packet 793rd), with the Fin Flag in the TCP header set to 1. Host B acknowledges receipt of the packet (packet 792nd) and the request of the termination (packet 793rd) in one message that reaches Host A. Host A keeps a record of the last sent message, thus it knows the value of the next expected Seq and Ack No, of which $\text{Seq No (sent)} = \text{Ack No (received)}$ and $\text{Ack No (sent)} = \text{Seq No (received)} + 1$. Re-transmission is required if the acknowledgment is not received in time or not in the right sequence, before the previous message from Host A is discarded. From packet 793rd onwards, the connection from Host A to Host B is already terminated, but that from Host B to Host A is still connected.
- Host B sends packet 794th to Host A, followed by a request of the connection termination by packet 795th (*Fin Flag = 1*).
- Host A replies to Host B with acknowledgement (packet 796th) of the receipt packets and terminates the connection between Host B and A, in order to free the connection for other hosts.

Further reference to TCP communication may be found in [g37] and [82].

It should be noted that the packet synchronisation in the state change notification (Table 9-2) was achieved in the same manner as above.

9.2.2-2 Ethernet Packet Delivery Time and Deterministic

Based on the implemented WS control system, the protocol analyzer has been installed to capture the network performance present on the network during the test-rig operation. Within this WS environment, SOAP messages for DPWS operations (i.e. state information, service call functions, discovery probe and metadata return) are transmitted. SOAP messages range in size from 750 to 1514 (max) bytes and the delivery time of one packet varies from 0.05 to 1 millisecond. During TCP transmission, there are 9 packets sent and received for one DPWS operation. These packets are responsible for establishing the TCP connection (3-way handshake), sending the SOAP message (a TCP byte – oriented sequencing protocols) and connection termination (full-duplex mode ⁽⁹⁾).

Table 9-3: Network Performance Analysis and Comparison

	LonWork Fieldbus [33]	FTB 10BasedT Ethernet
Packet size (Bytes)	12.8	750-1514
I/O interval reaction speed ^(e) (ms)	56.7-200	22-55 <i>(see section 9.2.3)</i>
Network bandwidth (KBits/sec)	4.01	495
Network utilisation (%)	5.13 %	8 %

(e)- The time delay between the occurrence of an input event (signal) and the corresponding of an output event.

This time is not included mechanical and electrical time loss caused by mechatronic devices' stiffness and friction.

A comparison of the Ethernet network performance enabled by the FTB controller and the LonWorks distributed control system [33] is shown in Table 9-3. The values of the FTB test rig are averaged from 10 runs of the full operation. In this research, the *network bandwidth* determines the speed of the network or throughput for data transfer (Kbits/second) over the connection, whilst *network utilisation (%)* indicates the current network traffic load. An ideal performance is indicated by a higher network bandwidth and a lower network utilisation (lower traffic). As seen from the

9- Two directions of dataflow; Two directions of the connection from Host A to Host B and vice versa

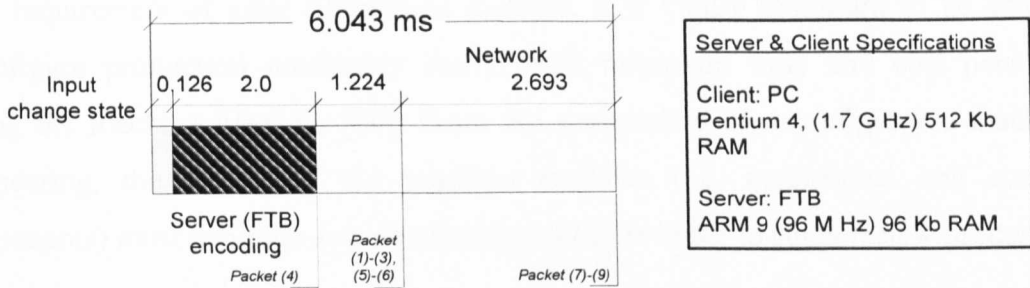
table, the large SOAP packet from the FTB controller results in a higher network traffic rates in comparison to the LonWork fieldbus system. Both network rates are considered normal if they are not over 50% after which performance starts to degrade (via packet loss and collision). In terms of speed, as defined by the I/O response time (i.e. I/O transaction time), the FTB control system has a faster response, although considering the larger packet size of the SOAP message this is due to a substantially higher network speed.

9.2.3 DPWS Processing Time and Component I/O Interval Reaction Time

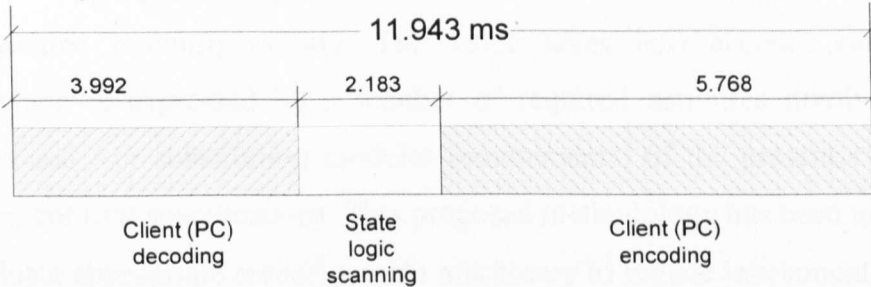
Regarding the DPWS processing time (i.e. parsing-encoding/marshalling and decoding/de-marshalling) on the FTB device and the *service orchestration engine* (on the PC), the millisecond and microsecond resolution timer functions were utilised on both the FTB and the PC application respectively, in order to measure accurately the time consumed by the DPWS functions. Note: the average value of the packet time for the DPWS operation is presented in Appendix E. The analysis data in this test was derived from approximately 900 packets accumulated from 10 experiments. Note that the protocol analyzer used in this evaluation provides time accuracy to microsecond resolution (the sample fragment codes can be viewed from the CD in the following directory, ..\WS Automation\Chapter 9\Stopwatch.c).

The full analysis of the DPWS functions is shown in Figure 9-2 which presents the analysis of the I/O interval reaction (i.e. response) time for: (i) DPWS event notification, (ii) orchestration logic processing and (iii) DPWS service invocation. The ARM 966 embedded control device with CPU speed 96 MHz is capable of processing the DPWS application in approximately 7 milliseconds for decoding and 2 millisecond for encoding SOAP messages. However, by taking into account the packet delivery time of all messages, local I/O processing, and the DPWS processing time, the estimated event notification and the service invocation time associated with the DPWS approach on the FTB device are 6.04 milliseconds and 13.08 milliseconds, respectively (*see Appendix E*). In addition to I/O response time analysis, the orchestration logic processing time on the PC (i.e. client applications) has been captured for decoding notification messages, state condition scanning and encoding for service invocation messages. Overall these processes take 11.94 milliseconds to complete.

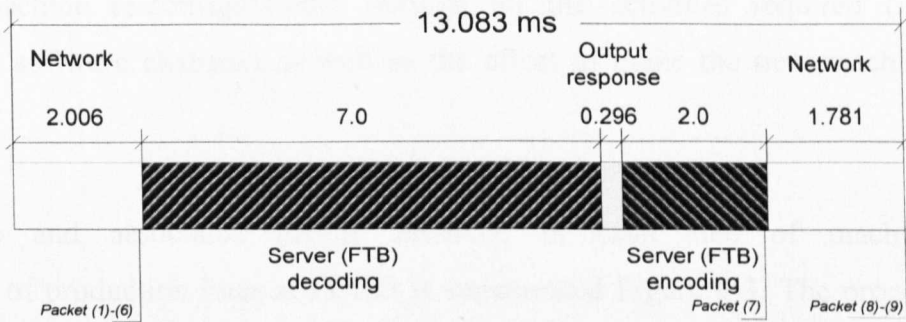
DPWS notification time analysis



Orchestration engine processing time analysis



DPWS service invocation time analysis



$I/O \text{ interaction time} = \text{Event notification time} + \text{Orchestration engine application} + \text{Service invocation time}$

Figure 9-2: The DPWS Service Invocation and Notification Time Analysis

It should be noted that the DPWS and I/O interaction time have been measured for the full test-rig operation of 4 distributed controller nodes comprising 10 components and 21 elements. As observed from Figures 9-2, the DPWS-enabled device is capable of delivering the I/O interaction time in around **31.06** milliseconds (Note: **6.04** ms: the DPWS event notification + **13.08** ms: the DPWS service invocation + **11.94** ms: the orchestration engine processing). As a result based on the calculated I/O intervention time, using the DPWS approach is 31.06 milliseconds just over the typical target I/O response time (under 30 ms as required by the end users). However, the WS approach has the potential to achieve the target time with further improvements (*discussed in Chapter 10*).

9.3 Ease of Machine Re-configurability Assessment

As a requirement of agile automation systems, it is vitally important to be able to reconfigure production machinery easily, with minimum time and cost penalties during the machine lifecycle [86]. From the perspective of reconfigurable machine engineering, the design of the machine modules (i.e. mechanical and control components) must facilitate system reconfiguration in order to support new products.

A.Urbani and S.P. Negri [102] have proposed a method to define a qualitative value of modular machine re-configurability. The value takes into account ease of modification, which is expressed by a number of required activities involved in making changes and / or substituting modules (components) of the present control system to implement new specifications. This proposed methodology has been used to design and evaluate appropriate reconfigurable machinery to reduce investment costs in the manufacturing lifecycles [102]. The methodology has been utilised to evaluate the ease of machine re-configurability through all the activities required (i.e. mechanical and software changes) as well as the effort to make the new machine configurations.

The procedure and associated people involved in each step of machine reconfiguration of production lines at FORD is summarised Figure 9-3. The process of machine reconfiguration for a new product (Job 2) is presented by the engineering tasks and associated engineering roles. During the reconfiguration phase, the system engineer is involved in deriving the new system characteristics and identifying new components, if required, for new control functionalities. Machine structures are then changed to the new configuration by the process engineer. The control engineer also builds required components from existing components and the control software is verified prior to subsystem integration when the machine sequence, control application and operator consoles are integrated. The verified subsystems are assembled with other subsystems to form the complete machine application, and complete system validation is carried out by the system engineer, to ensure that new machine configuration is functioning correctly and that the performance meets specifications prior to being delivered to the end user.

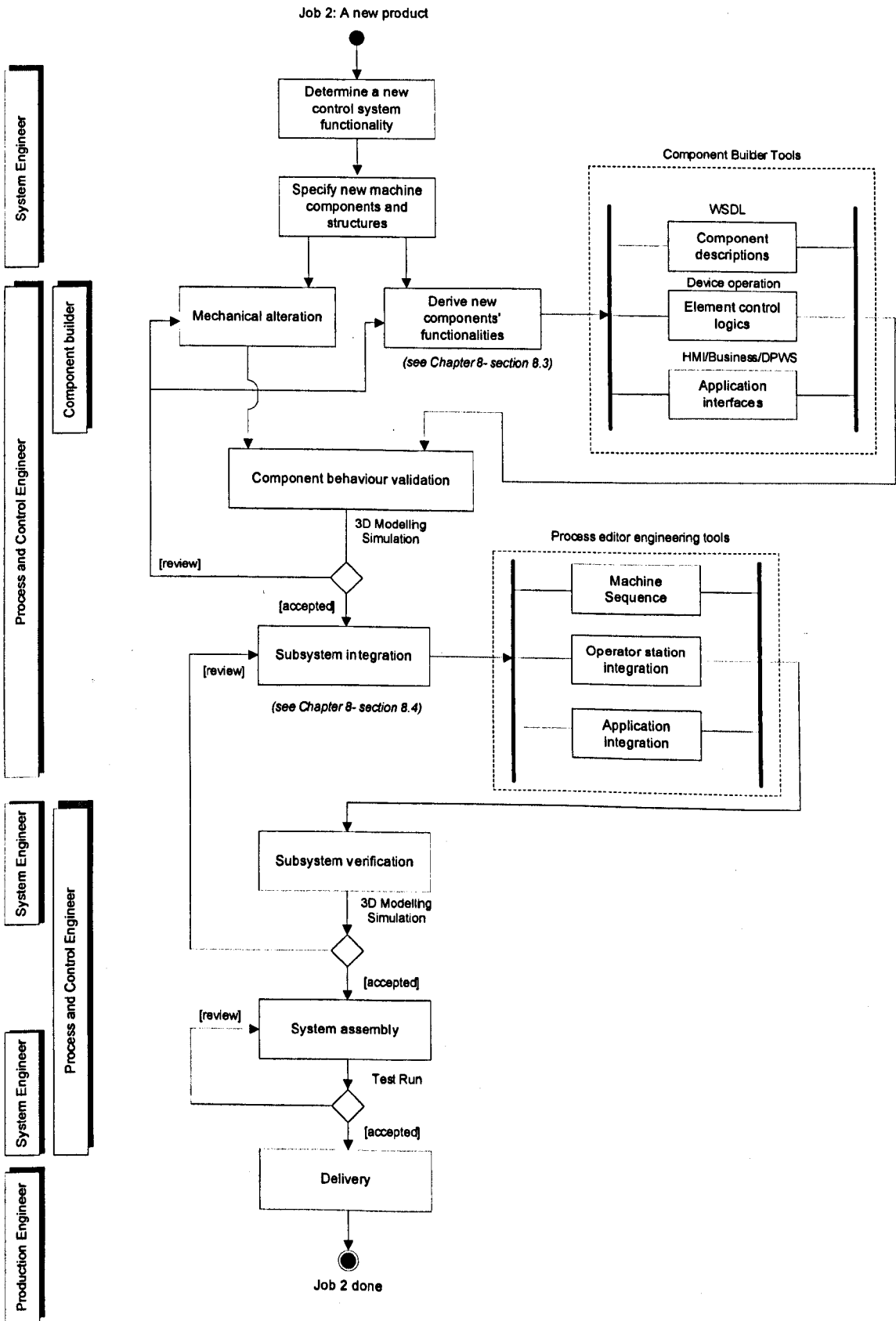
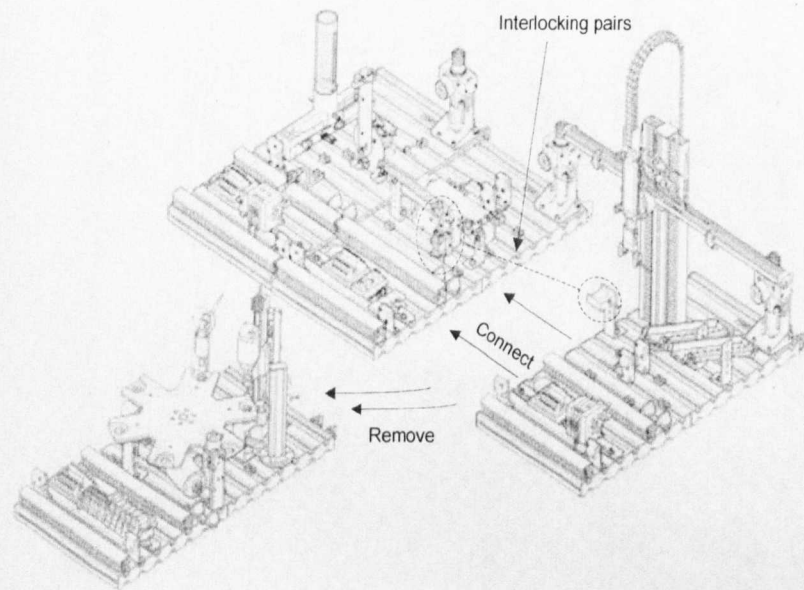
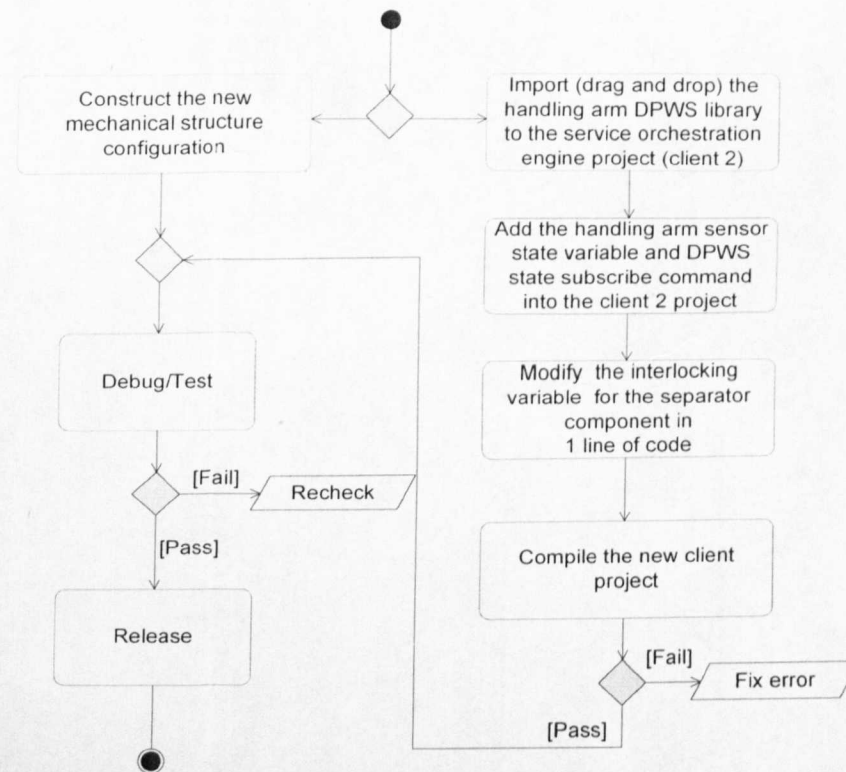


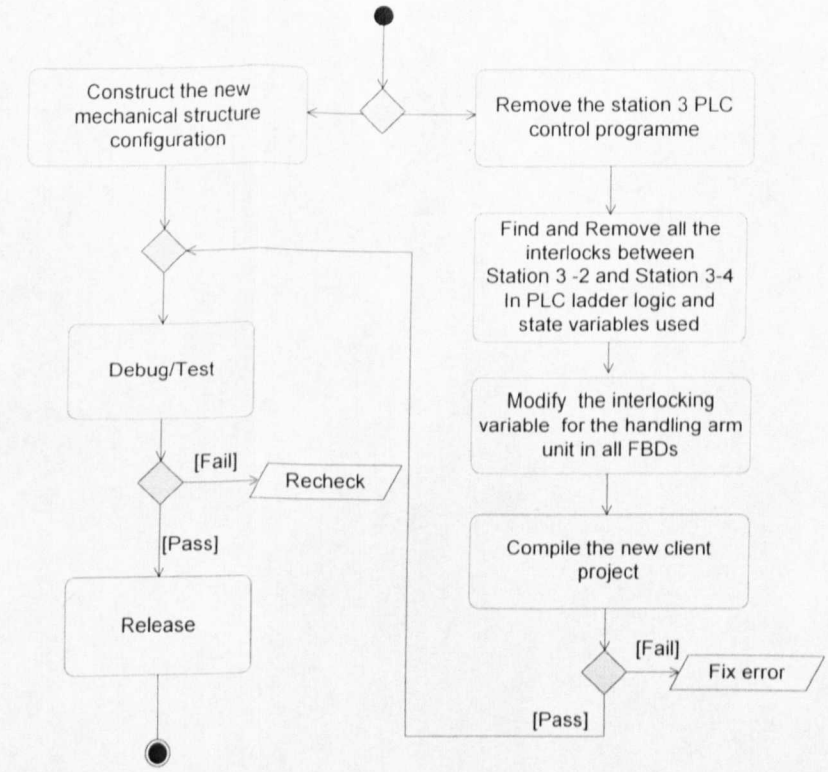
Figure 9-3: Process Reconfiguration Workflow



a) Altering a Process Workflow Scenario

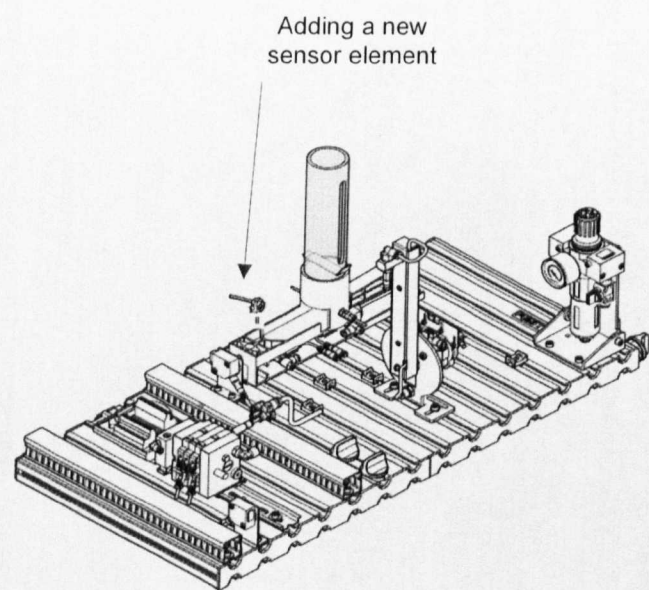


b) Web Services Reconfiguration Procedure

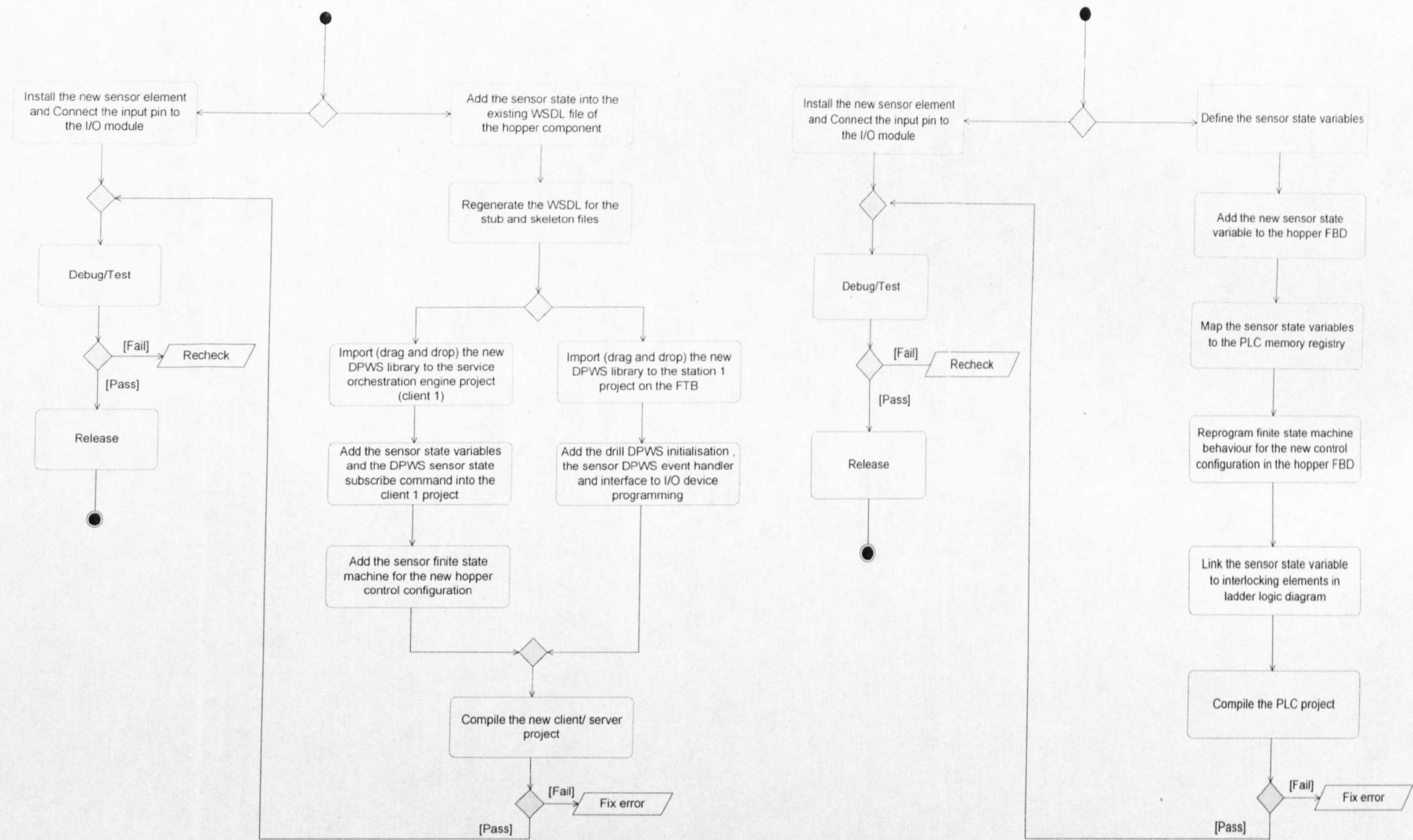


c) PLC Reconfiguration Procedure

Figure 9-4: Workflow Alteration: Station 1/2/3/4 → 1/2/4



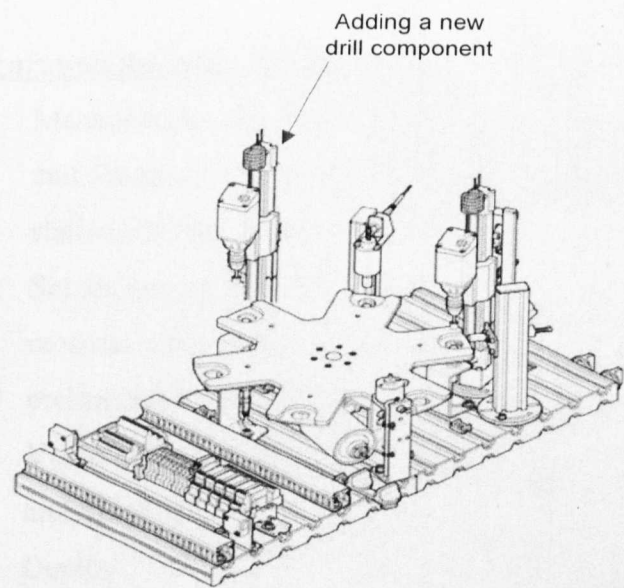
a) Adding a New Element Scenario



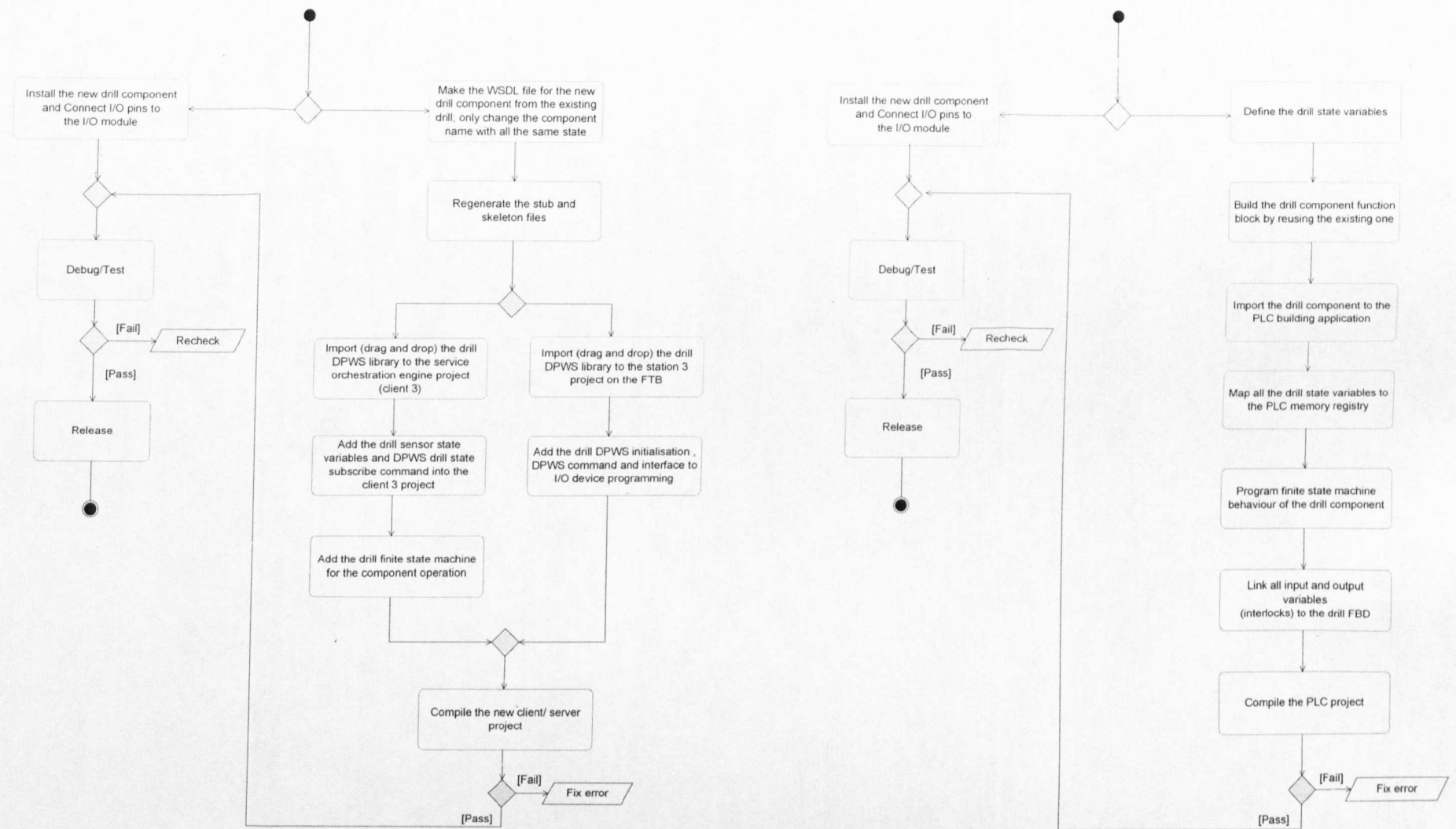
b) Web Services Reconfiguration Procedure

c) PLC Reconfiguration Procedure

Figure 9-5: Adding a Sensor in the Hopper Component



a) Adding a New Component Scenario



b) Web Services Reconfiguration Procedure

c) PLC Reconfiguration Procedure

Figure 9-6: Adding a New Drill Component

In assessing the degree of re-configurability of automation systems, test scenarios were determined to capture the activities involved in hardware and software reconfiguration based on the Web Services-based automation system implemented on the FORD-FESTO test rig. The tests assigned were to modify the process workflow and add a new process unit. It is noted that the test scenarios represent real-life activities in the reconfiguration examples occurring within FORD process lines. The following three scenarios were set up for evaluation:

9.3.1 Modifying a Process Work Flow

The Test Scenario Description:

Removing the processing unit (station 3) from the current configuration to bypass the assembled work piece directly to the handling unit

Configuration Steps for the WS Design Approach:

1. Mechanically rearrange the station, removing the processing table station 3 and fixing the handling station 4 to station 2 (i.e. current configuration station 1/2/3/4, new configuration station 1/2/4)
2. Set the new interlocking pairs between the separator (i.e. station 2) and the receptacle sensor (i.e. station 4) for the station synchronisation on the service orchestration engine.
3. Validate the new configured system (i.e. Simulation, Console Debugging, and Test Run)
4. Deploy

Note: On this Web Services-based test rig control system, there are no changes on the I/O configuration (i.e. device programming) and control applications (i.e. interface and DPWS call functions) on the controller. Only device interlocks need to be modified for the new machine application.

Description of WS Component Reconfiguration:

This change is very common at FORD process lines. The modification in this case was simple and thus imposed no major changes on the control software. The component behaviour, implemented with Web Services on the controller, remains unchanged. Only the old interlocks has been altered and replaced by the

new ones within the *service orchestration engine* in order to form the new machine specification. The modification of the control software in this test scenario was completed in a couple of hours and the time taken in changing the machine structure was dependent on the structure, size and fixing mechanism of physical machinery.

PLC-Based Reconfiguration Discussion:

In comparison with the PLC-based control system, changing the state transition behaviour is achieved by reconfiguring the interlock between components and the internal implementation of finite state machine behaviour of the component. The required change is subtle because the interlocking variables are difficult to trace throughout the application since there are linked to various points. Consequently, this task of control reconfiguration has to be assigned to control specialists, who have an understanding of the ladder logic program needed to implement the changes.

9.3.2 Adding a Sensor Element

The test scenario description:

Adding a new sensor at the *hopper* downstream position to identify the work piece colour, where a red one is allowed to pass through to the next step but black ones are stopped at the unit

Configuration Steps for the WS Design Approach:

1. Install the new sensor element
2. Connect the sensor input lines to available channels of the I/O module
3. Add the sensor state into the existing hopper WSDL file description and regenerate the stub and skeleton files
4. On the FTB device-Station1 project: add the DPWS control application (DPWS initialisation and sensor event handler) for the new sensor interface to device low-level programming by reusing the existing control application of the existing sensor as reference
5. Debug the new component software and upload to the controller
6. On the PC- Client 1 project: define the new sensor state variables for the DPWS sensor state subscription used by the hopper operation sequence.

7. Add the new sensor sequence to the service orchestration engine application for the new hopper control configuration
8. Compile the client service orchestration engine project
9. Validate the hopper component operation (Simulation, Console Debugging, and Test Run)
10. Deploy

Note: The new sensor has the same number of states (WP_Black and WP_NotBlack), as the existing one.

Description of WS Component Reconfiguration:

In step 3 of adding the new sensor element, the Web Services interface (DPWS) is implemented by simply adding the sensor state variables in the existing *hopper* WSDL file and regenerating the new stub and skeleton files, which are imported to the current project of the control device (i.e. the server application of the station 1) and the *service orchestration engine* (i.e. the client 1 application). The Web Services functionality of the sensor is then programmed and uploaded to the controller of the same station. In step 7, the new machine sequence is implemented separately on the service orchestration engine by defining a new set of interlocks for the *hopper* component, including the new sensor functionality. In this test, there is no requirement to program the low-level device code of the new sensor in step 4 if the interface to the DPWS call function is previously implemented. The interface maps the sensor input (channel) to the function operated by the DPWS state notification (*see Chapter 8- section 8.5.1*) in the control application on the FTB.

PLC-Based Reconfiguration Discussion:

Adding the new sensor element in the PLC-based system is achieved in the similar manner to the WS control system where the machine builders deal with creating the new element functionality as well as altering machine sequence. However, the PLC-based system requires the application builders to work with more technical detail to determine the specific I/O channels (connected to the new sensor) and memory allocation referred by device state variables. The new machine application is then changed by modifying the sequence in the ladder logic or function block with the new element functionality. In this way of the

process reconfiguration, any changes require careful attention and understanding of control sequences in the machine application.

Note: The PLC-based control system development discussed in this research is based on the implementation platform carried out by MSI Research Group at Loughborough University. The current machine and control application has been implemented by both ladder logic and FBD's.

9.3.3 Adding a New Component

The Test Scenario Description:

Adding another drill for a 2-stage bore hole of a different drill size on the work piece

Configuration Steps for the WS Design Approach:

1. Install the new drill component
2. Connect the I/O lines (2-actuators of a drill spindle and an axis, 2-limit switch sensors) to available channels of the I/O module
3. Create the WSDL file description for the new drill component by reusing the script from the existing drill and generate the stub and skeleton files
4. On the FTB device- Station 3 project: add the DPWS control application (DPWS initialisation and all drill state event handlers and commands) for the new drill sensor and actuator interfaces to device low-level programming by reusing the existing control application source code of the existing drill as reference
5. Debug the new component software and download to the controller
6. On the PC- Client 3 project: define the drill state variable (of sensors and actuators) and the DPWS state subscription for the drill component
7. Add the new drill sequence to the service orchestration engine application for the new control configuration of the station 3
8. Compile the client service orchestration engine project
9. Validate the drill component operation (Simulation, Console Debugging, and Test Run)
10. Deploy

Note: The new drill component has the same 4-state behaviour and operation as the existing one. Most of the code can be reused from the existing drill component (*see section 9.4- Table 9-6*).

Description of WS Component Reconfiguration:

In step 3 of building the new drill component, the Web Services interface (DPWS) for the drill was implemented by building the drill descriptions from WSDL file and generating the stub and skeleton files, which are imported into the current project of the control device (i.e. the server application of the station 3) and the *service orchestration engine* (i.e. client 3 application). The WSDL description of the existing drill is fully re-used (re-configured) by the new one, as they have the same functionality. Only the drill component names need to be changed for the unique names in the control system. In step 4, the Web Service functionality of the component is programmed and downloaded to the controller of the same station, and in step 7 the new machine sequence is implemented separately on the service orchestration engine by defining a new set of interlocks for the drill component. As mentioned in the earlier test scenario, a component user, such as a control engineer, does not need to be concerned with the I/O of the device to modify the process configuration. This is because adding or removing the component configuration on the controller devices will not pose any changes and involve any programming to this level of the device.

PLC-Based Reconfiguration Discussion:

Adding a new component is achieved in the same manner as adding a new element as discussed previously where the new component functionality and the control sequence in the ladder logic and function block diagram are added. Reconfiguration in this case is managed by reusing the existing drill function block, but however, the difficulty of changing the logic inside the function block module with the right variables and connecting the block to the right point in form of the ladder logic for the right sequence is still an huge issue and requires experience engineers for its successful implementation.

It may be observed from these test scenarios that the component's internal implementation is separate from the application specification corresponding to machine tasks defined by Web Services applications. This allows the system integrator/control builder to develop and verify the control application without having to understand the complexity of the low-level implementation details. The application is clearly organised and visualised at the higher "process" level. In this research, system behaviour is defined through the components' state-transition conditions via the state interlocks and this is kept separate from their low-level implementations. The WS application regarding component operations is interfaced to the low-level device operations using a call function as an interface. However, with the PLC-based system (*as presented in the case study 1 Chapter 7- section 7.3*), the changing of the system behaviour has to be made through the ladder logic program at the PLC code programming level, both inside and outside of the component function block. Time and close attention is required to ensure that all low-level implementations related to the modification have been reviewed and changed.

With regards to the implementation of the WS based components, the reconfiguration of production lines for different workflows, i.e. adding new elements and components, is considered to be a relatively easy task as it occurs at the system level. Modifying the process workflow and deploying the new system configuration only takes a couple of hours. When compared with the PLC-based application, there are significant differences. Although the reconfiguration of the PLC-based system involves fewer steps than that of the WS based design approach (as shown in the reconfiguration work flow in Figure 9-4b/4c, Figure 9-5b/5c and Figure 9-6b/6c), its reconfiguration tasks require considerably more effort in identifying all the required changes and ensuring that they are consistent, complete and correct. Also, these changes require particular expertise in the specific types of the PLC employed within the system.

Adding the new machine component and element, the control application in the WS environment is developed by reusing the control code (i.e. DPWS component initialisations and operations) of the existing components in the project. The assessment of modularity and reusability of components is presented in the following section.

9.3.4 Comparison of Distributed WS-CB and Conventional Centralized PLC Automation

Previous sections have presented the re-configurability of the distributed WS and PLC based control systems. However, this section provides a concise comparison between the implemented WS approach and a more conventional centralized PLC based control system (*Chapter 4- section 4.2*). The discussion on advantages and disadvantages of both approaches is based around the reconfiguration scenario (see below), considering system performance, system design and cost. The construction of the specific machine control system is based on the model of the FESTO test rig (*Chapter 7- section 7.3.1*) as shown in Figure 9-7 and 9-8.

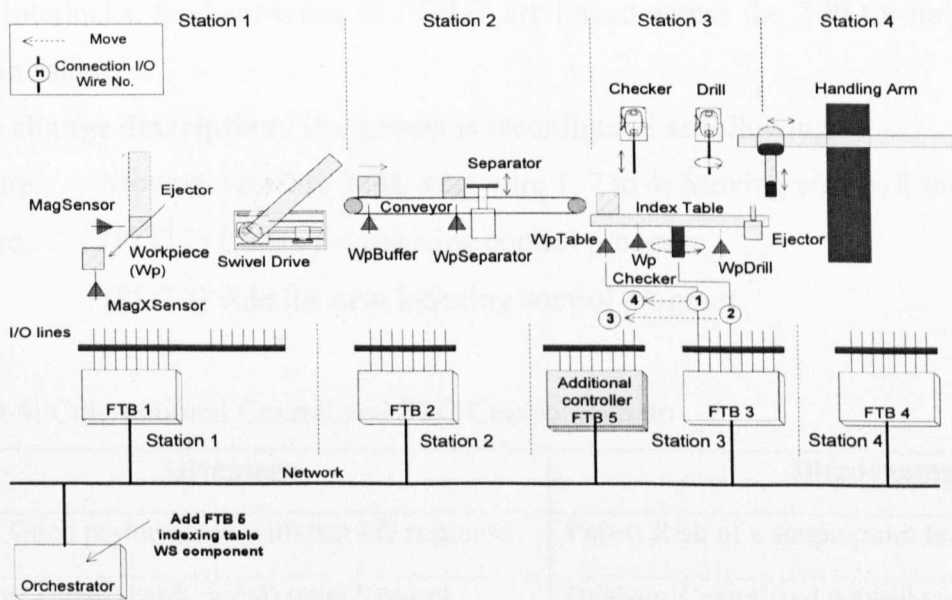


Figure 9-7: Distributed WS- based Control System

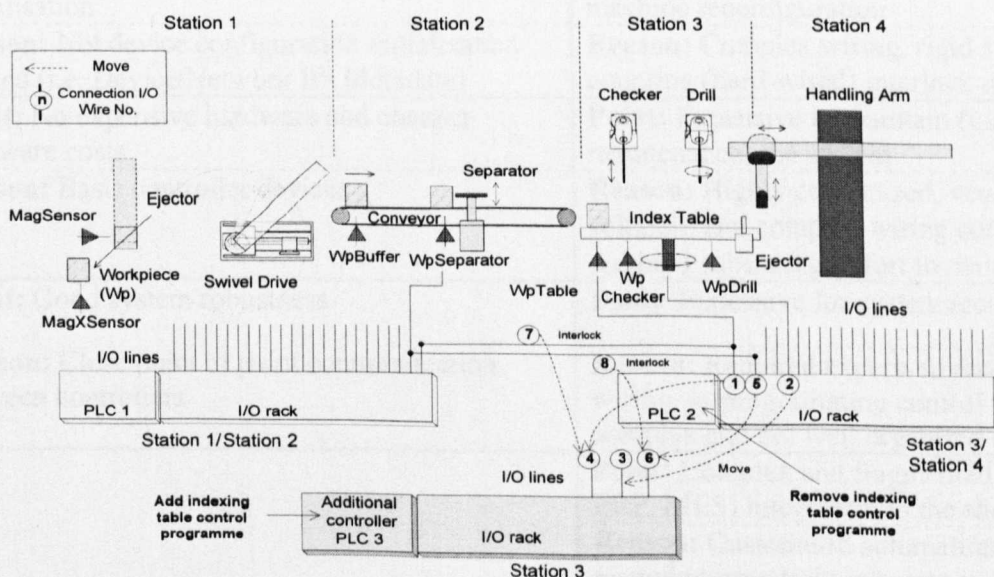


Figure 9-8: Conventional Centralized PLC- based Control System

System change scenario: The new controller is added to the control system in order to reduce the work load of the controller on Station 3 by reassigning the table rotating task to the new controller. In this example, it affects the function of 2 input sensors (work piece interlocks) and 1 output actuator (indexing the table), which need to be moved to the new controller.

Conventional Centralized PLC Control System (Figure 9-8)

System description: The controller functions are customised to specific machining tasks. In this example, the machine system (divided into 4 stations) contains 2 PLCs, each of them control 2 stations accordingly. All the input and output I/O lines are directly connected to the specific PLC I/O channels as designed. However, for the system interlocks, the hard-wires of INPUT are linked across the 2 PLCs for system synchronisation.

System change description: The system is reconfigured as following

- Hardware: Moving wire 2 to 3; Moving wire 1, 7 to 4; Moving wire 5, 8 to 6
- Software: (PLC 2) Delete the indexing control program;
(PLC 3) Add the new indexing control program

Table 9-4: Conventional Centralized PLC Control System

Advantage	Disadvantage
Point: Good performance with fast I/O response	Point: Risk of a single-point failure
Reason: Direct (hard-wired) point to point communication	Reason: Centralized control systems
Point: Faster system boot-up time and controller initialisation	Point: Time consuming and error prone process in machine reconfiguration
Reason: Not device configuration initialization needed (i.e. Device Network IP/ Metadata)	Reason: Complex wiring, rigid structure and tight coupling (hard-wired) interlock of a control system
Point: No expensive hardware and cheaper hardware costs	Point: Expensive to maintain (i.e. upgrade, maintenance) the system
Reason: Basic controller devices	Reason: Highly customised, vendor specific solution, and complex wiring control system that is required substantial effort to maintain
Point: Good system robustness	Point: Expensive for system reconfiguration
Reason: Close point to point communication between controllers	Reason: Required experts to reconfigure (re-wiring, re-programming control software) the complex and not well organised system
	Point: Complex and fragmented application (i.e. ERP, MES) integration to the shop-floor
	Reason: Customised automation system to certain manufacturing tasks requires customised application interfaces

Decentralized WS-CB Control System (Figure 9-7)

System description: In the distributed control system, each of the controllers controls one station accordingly. All the input and output I/O lines of each station are only connected to the I/O channel on the specific controller via its I/O interface module.

There are no cross I/O connections between controllers.

System change description: The system is reconfigured as following

Hardware: Moving wire 1 to 4; Moving wire 2 to 3;

Software: (FTB 3) Delete the indexing table control program

(FTB 5) Add the new indexing table control program

Table 9-5: Decentralized WS-CB Control System

Advantage	Disadvantage
<p>Point: Easier to reconfigure with lower labour costs (less time consuming)</p> <p>Reason: (see <i>Easier to maintain and upgrade</i>) in this case less time is required to change and debug the new configuration</p>	<p>Point: Expensive hardware</p> <p>Reason: More sophisticated controllers with integrated TCP/IP network functionality</p> <p>Suggestion: The system design cost could be reduced with cheap and powerful embedded micro-controllers in the future, and greater software/hardware reuse.</p>
<p>Point: No risk of a single-point failure</p> <p>Reason: Loosely coupled distributed control allowing task distribution</p>	<p>Point: Slower to start up</p> <p>Reason: Requires controller initialization (i.e. network, software component configurations)</p> <p>Suggestion: Although the slow start will not pose a major time loss to production, improvement could be achieved with greater processing speed and dedicated WS processors (<i>see Chapter 10 Future work</i>).</p>
<p>Point: Easier to maintain and upgrade</p> <p>Reason: Well organised (separated component software and clear hardware wiring structure) system; Engineers know exactly where to work on the control system (control programme and I/O wiring changes)</p>	<p>Point: Slower speed</p> <p>Reason: Time delay in the variable exchange on the network as well as data parsing</p> <p>Suggestion: Faster network speeds and SOAP in binary form (<i>see Chapter 10 Future work</i>)</p>
<p>Point: Simple system integration with only a single application interface per device</p> <p>Reason: The system is built within standard technologies (i.e. SOA, WS and Ethernet) where various applications could be simultaneously integrate to the shop-floor system without adding more interfaces (i.e. reduces complexity)</p>	<p>Point: Possibility of data loss if network load is substantially too high (affecting robustness)</p> <p>Reason: Open loop communication (dynamic plug and play devices) in the distributed control system</p> <p>Suggestion: Additional message acknowledgement method and the separate network router could be employed for more organised and scalable control systems.</p>
<p>Point: Possibly, shorter time to build</p> <p>Reason: Beneficial software (component and control application) reuse</p>	

9.4 Component Modularity and Reusability Assessment

Modularity

Modular architectures are typically defined as having one-to-one mapping from function elements in the function structure of the physical components. The modules are useful for design reuse, as already-designed modules with well-defined interfaces may be used again in other designs. This applies to both software and hardware components and product change, upgrade and variety can be achieved by replacing or adding a module in a system, without having to make changes to the overall production platform [109].

It has been reported [112] that an appropriate level of granularity within the component-based system architecture is important in order to support effective reuse and reconfiguration. Appropriate modularity makes systems easier to build, reconfigure, repair and manage. Good machine modularity is characterised by *minimal interaction between modules/stations (coupling)* and *maximum interaction within modules/stations (cohesion)*. The coupling and cohesion terms are associated with changeability, referred to as what changes can be accommodated and the number of tasks involved. For example, regarding the modularity description outlined in [112] the change of system operation within coupling modules would involve a modification of the interaction between control nodes. However, changes at cohesion level would only see changes within the node that involved less activity.

Regarding the development of the WS automation platform, a modular design has been achieved, featuring 4 couplings (interlocks between stations) and 10 cohesions (internal interaction within stations). The test rig supports the modular design, with one-to-one control functions to physical components mapping. The more complex designs of the test-rig system were formulated by defining the combination of components services and interlocks at a high-level programming, rather than writing application specific codes for the components at low-level device programming. In this CB Web Services design approach, it is not only the control application which is defined as the modular component. Additional services, such as state publication and error diagnostics are also designed for modularity and are well organised in the constitution of components. This approach enables the best practice for the effective

reusing and reconfiguration of control, diagnostic, and maintenance software components during the lifecycle.

Reusability

With regards to the assessment of control system reusability, [33] has estimated component reusability at the subsystem level, based on measuring the number of reused components for building / integrating a new control system approach (generally carried out by the system integrator and the control builder) with the support of engineering tools and component libraries. However, from the component builder / supplier perspective (which involves the hardware device programming), the evaluation of component reusability needs to measure the code reuse at the DPWS application level for creating new components within the WS development platform. Given the new component design, as previously demonstrated in the test scenario of adding the new drill component (*see section 9.3.3*), conceiving a new component for component users requires the commissioning of:

- 1) The WSDL description (to generate WS server stub and client skeleton files),
- 2) DPWS interfaces for device operations on the FTB project-server application.

In addition to commissioning a new component, component users need to orchestrate the new control configuration for newly-required manufacturing tasks, as defined in the client application via new sets of interlocks for the service orchestration.

How would code reuse be justified in this case? It should be noted that the analysis of code reuse in this research has been considered via the process of “copy-paste” and a variable name modification of existing code for new components. Within this work, these modifications are normally carried out in the DPWS and component variable initialisation and in standard utilities such as error diagnostic routines and heartbeat systems. Reprogramming these items for a new component is a relatively easy task, with the reused components having the same state behaviour and element names. However, programming new lines of code is justified, taking into account the effort required in programming reused code in some circumstances, such as creating the new set of interlocks and building new interface call functions to the low-level device

operations. These are seen as additional new programme lines, and these tasks require careful attention in the design and involved specific control operation of devices.

Table 9-6: Adding the New DPWS Drill Component Scenario

Required software element	All codes (Lines)	New codes developed (Lines)	% reusable codes 1 - <u>New codes</u> All codes
WSDL definition	146	0	100 %
Server application (FTB)	86	13	84.88 %
Client application (Service orchestration engine-PC)	127	28	77.95 %

Regarding component code reusability in building the new drill component with the Web Services approach, shown in Table 9-6, WS call operations and drill state variable names remain unchanged, thus the WSDL definition for the new drill component has been slightly altered via a new component name for the unique Web Services target namespace. The WSDL definition can therefore be reused with confidence. For the drill operation of a second drill on the embedded device (FTB), only the DPWS interface to the device operation (13 lines of codes) needs to be written. Note that the component control logic for the low-level device functionality, provided by the component builder, is encapsulated as “black box” and the I/O already exists. Therefore there are no changes to the internal implementation of the local controller application. On the client application, the 28 lines of device interlock programming code have been added to support the new system operation.

9.5 The Implementation with Process Engineering Tools

It is noted that this area of the author's research focuses on the development of a WS automation platform and considers the integration of third party engineering tools (i.e. control application builders, PDE toolkit), manufacturing applications (i.e. 3D simulation, HMI) and business applications (i.e. Process visualisation). The development of these applications has been carried out by other researchers. In conjunction with the development of process engineering tools for system (re)configuration, the author's research has outlined the description of control building and reconfiguration activities in detail. However, the WS approach of process building and configuration involves a small level of complexity in coding and debugging components and control applications (i.e. WSDL/ DPWS code generation and embedded system design) but required skills for non-experts. Therefore, it is important that this research outlines the effective strategy of building a control system in this WS environment.

As end-users (i.e. the system integrator/ control builder) require ease of process design and reconfiguration, the process engineering tools have a significant contribution in accommodating the design, integration of the control system and process reconfiguration by hiding the complexity of component programming and minimising the level of manually coding tasks. As stated in the previous section on component reusability, component users do not expect to have to deal with the manual coding of component interlocks and state transition behaviours implemented on the client application to create the machine application. It has been identified by this research that the PDE tool should provide the platform for configuring the components, (e.g. device Web Services namespaces and the finite state machine applications for manufacturing tasks). These component configurations can be saved for later (re)use and reconfiguration for new machine applications. Given the PDE tool, it can be observed that the component reusability in the design for a new machine is high. This is due to the modular design of the DPWS application in the component, where a component requiring the same or similar functionality (i.e. via component states, operations) can be readily reused.

The platform of the WS-enabled device needs to be outlined in order to accommodate the PDE tool development. The WS process engineering tools platform as detailed by [46], encapsulates functionality covering control software editing and control code generation, and is aimed at assisting control and process engineers to manage the task of implementing control code for complex and large-scale systems. This is achieved by:

- a) Providing a direct translation of process production sequence information into usable code, potentially in distributed form, at the embedded control hardware level, and
- b) Providing the means to maximise the re-use and re-configuration of the control application.

The development of WS-based control components has to support a platform, in order to accommodate process engineering tools in the development of new components and the uploading of device runtime control applications and data configurations. An architecture, as shown in Figure 9-9, has been derived in this research by considering the extension of DPWS capability to support the integration of the control builder applications and the PDE tool.

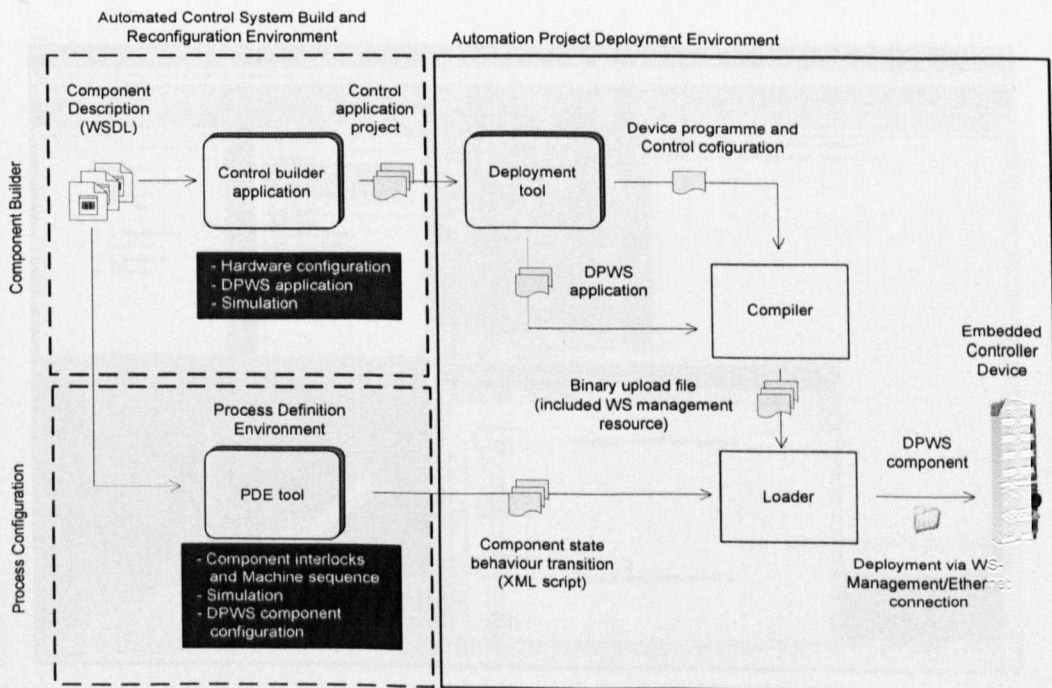
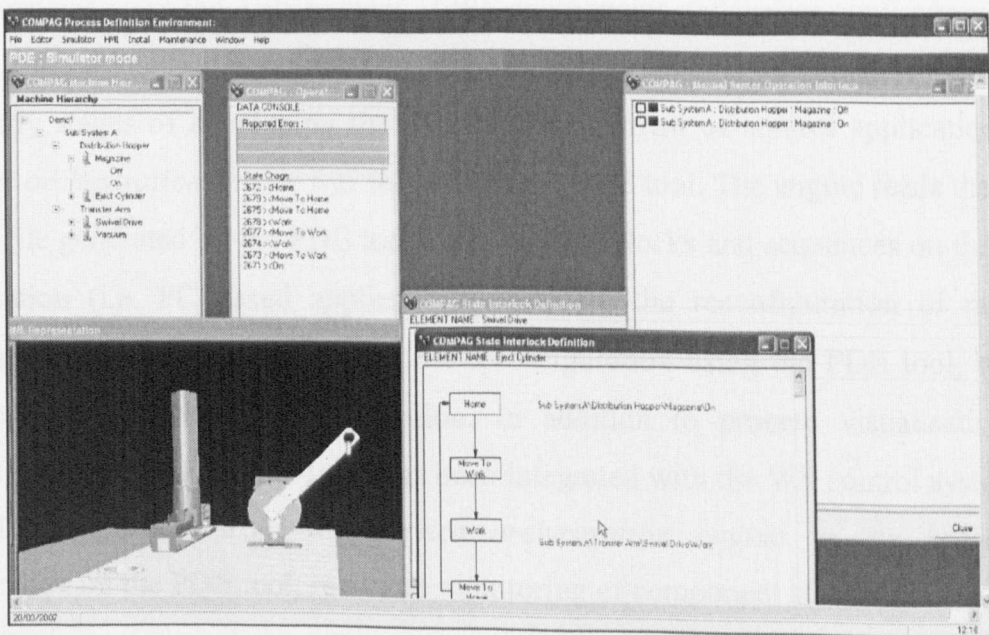


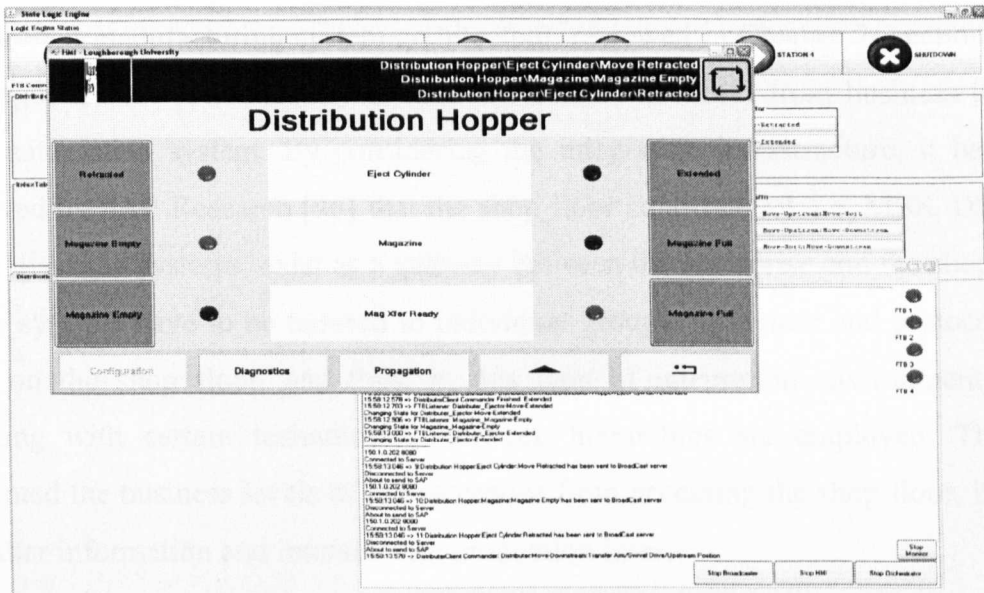
Figure 9-9: Automation System Development and Reconfiguration Framework

The DPWS configuration and control application are built and deployed by the control builder application and the deployment tool (in this case developed by Schneider Electric) in a visual environment, rather than manually programming at the low level of the control applications. This is achieved by compiling the device and service resource management (via XML device description and device configuration script files) in support of PLC open tool (Control Build Tool) onto the embedded control device for building and (re)configuring the control system. At the time of writing, ease of device interlocking (as implemented on the *service orchestration engine*) aspects has not been fully realised, and thus at this stage, the gap can be filled by the PDE tool to support the device interlocking for the control application (see Chapter 6- Section 6.8.3) implemented by the *service orchestration engine application*.

In addition to the dynamic deployment of DPWS components (see Chapter 8- section 8.5.1), the runtime (re) configuration of component control applications is achieved via the *WS-Management* service [g12]. Note: the *WS-Management* service is ongoing work on the SOCRADES research project, and when completed this package will be included in the DPWS standard, in order to support the management (i.e. editing) of the component and its configuration on the embedded control devices (i.e. FTB's) without recompiling the control application.



a) PDE tool



b) HMI and State Logic Applications

Figure 9-10: MSI Developed Suite Engineering Tools

As demonstrated in Figure 9-10/a, the PDE tool supports the *control builder* functionality, which enables the generation of the XML script files for the runtime control configuration. The PDE exports the component content (i.e. the component configuration data and state transitions) in an XML format. Note that the current PDE tool does not assist the development of WS components and device configuration, but rather contributes to the definition component sequences and manufacturing tasks. In the early stages of developing the dynamic deployment of control applications, the *service orchestration engine* was tested with the PDE tool. The engine reads the XML script file generated by the PDE tool for device interlocks and sequences on the client application (i.e. PC based application), therefore the reconfiguration of machine sequences can be implemented via the reconfiguration using the PDE tool, without any changes to the client application. In addition to process visualisation, the prototype of HMI (Figure 9-10/b) has been integrated with the WS control system, via TCP/IP connection from the *service orchestration engine* to the broadcaster application on the PDE tool, real-time monitoring of component states.

9.6 Seamless Integration Assessment

The present manufacturing system architecture, reviewed in Chapter 2- section 2.3.1, is categorised into different layers in a hierarchical network, from business to shop floor automation system. By considering the integration infrastructure, it has been observed by SAP Research [45] that the shop floor control level, i.e. MES, DCS and Plant Historian systems, exist as a gateway between the enterprise and the shop floor. These systems have to be tailored to individual groups of devices and protocols that exist on the shop floor, and thus, in this type of integration environment, close coupling with certain technologies between hierarchies are employed. This has prevented the business levels of the enterprise from accessing the shop floor, in order to gather information and interact with production lines.

Using the component-based design framework, with LonWorks and the PLC- based systems, the integration of the control system with the business application requires a communication protocol driver. This driver translates the device data into a specific format, in compliance with integrated applications and the communication protocol driver acts as a gateway between control devices and integrated applications, such as HMI's, broadcasters, manufacturing and business applications. An example of a gateway approach is the LonWorks -based system: the *XLON-USB* [g35] (Figure 9-11) and Premium PLC-based systems use *Monitor Pro XL* specification [g18] (Figure 9-12) as the communication protocol driver.

However, in the previous CB implementation of the LonWorks system test rig, the XLON device was used to interface the broadcaster and HMI applications to LonWorks nodes.

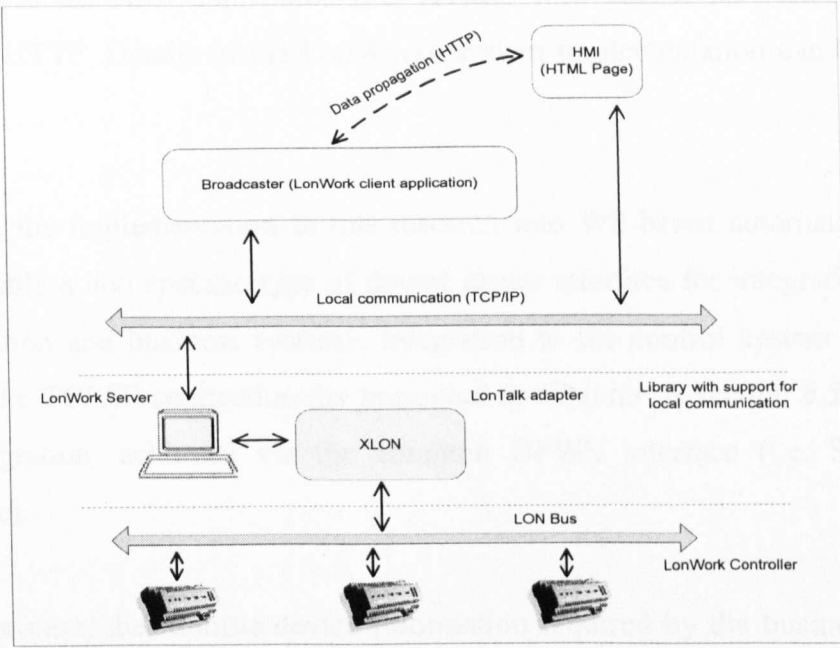


Figure 9-11: XLON Communication Protocol Interface [g35]

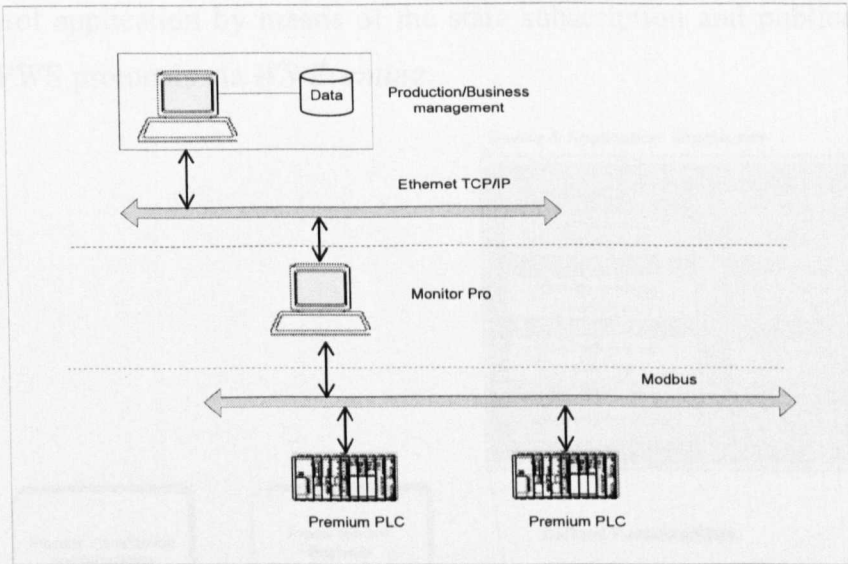


Figure 9-12: Monitor Pro PLC Communication Protocol Interface [g18]

Integration within the environment as shown above is tailored to specific target devices with different sets of device drivers. As illustrated in Figure 9-11, LONBUS (a Fieldbus system) is connected to the LonWorks client application running on a PC via *XLON-USB* which provides the Application Programming Interface (API) to communication devices. During runtime, the client application broadcasts state

information to the HMI application (i.e. HTML Web pages) for visualisation and control via HTTP. Details of the LonWorks system implementation can be found in [33].

In contrast, the implementation in this research into WS-based automation systems aims to enable a non specific type of device driver interface for integration between the automation and business systems. Integration to the control system is achieved either via the TCP/IP connection (*as presented in Chapter 8- section 8.5.3*) or using direct integration, achieved via the common DPWS interface (i.e. SOA device middleware).

In the latter case, the runtime device information required by the business level are already implemented on the WS enabled device and provide rich device information and live states to integrated applications throughout the manufacturing and enterprise system. Also, state and error information from devices can be directly sourced to the higher control application by means of the state subscription and publication, using standard DPWS protocols via *WS-Eventing*.

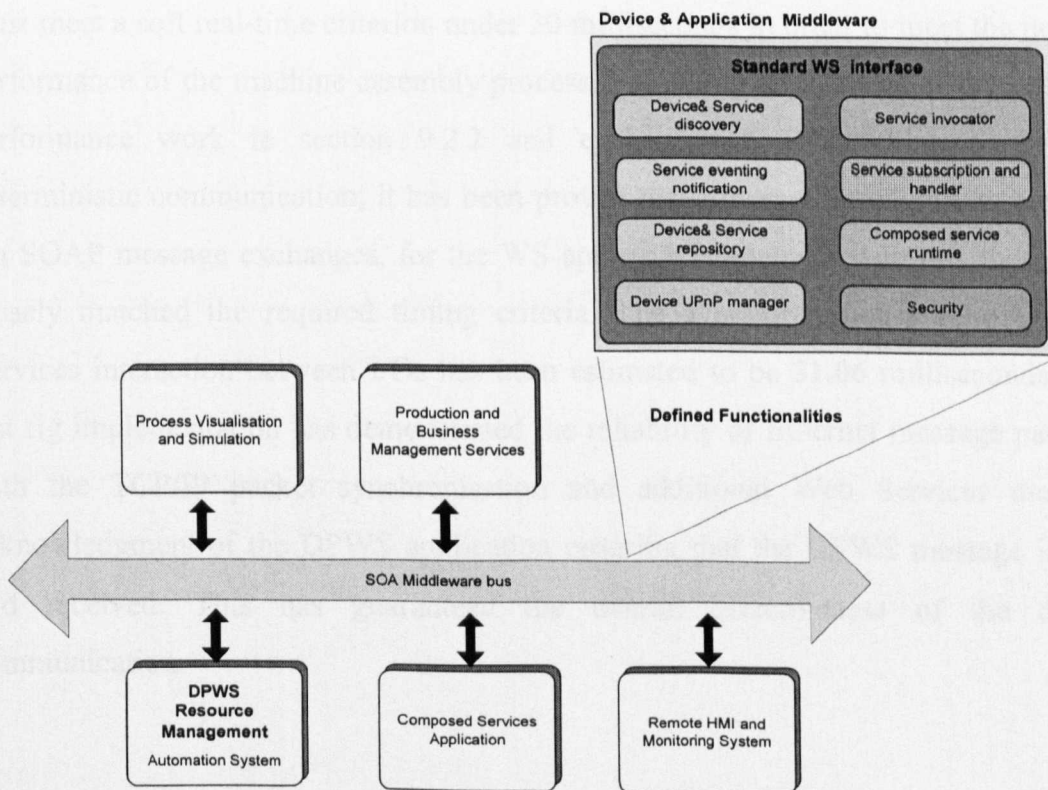


Figure 9-13: The SOA Middleware Integration Bus

As illustrated in Figure 9-13 the integrated architecture is not layered in different hierarchies but is indeed level. For seamless integration, Web Services on the devices via DPWS provide direct communication through the device discovery service to inter-connected applications for browsing device meta-data as well as invoking the device services and resource management via the *WS-Management* service (section 9.5). The integration of third party applications is achieved via the common mediator of SOA middleware. This allows more sophisticated application integration such as business planning, machine vendors, production, maintenance section and control systems. Also, the provision of SOA middleware has enabled direct integration to automation devices through the standard WS interface. Therefore, the control platform is not subject to increased complexity from several integrated applications, with various interfaces for example MES/ERP system integration.

9.7 Suitability of the WS Based Automation in Industry

I/Os intervene time

The responsiveness of input and output reaction times between inter-connecting controllers in production at the FORD motor company specifies that response time must meet a soft real-time criterion under 30 milliseconds in order to meet the normal performance of the machine assembly process. Following an analysis of the network performance work in section 9.2.2 and even though Ethernet supports non-deterministic communication, it has been proven that Ethernet based communication via SOAP message exchanges, for the WS approach implemented in this thesis, has closely matched the required timing criteria. The average response time of Web Services interaction between I/Os has been estimated to be **31.06** milliseconds. The test rig implementation has demonstrated the reliability of Ethernet message packets, with the TCP/IP packet synchronisation and additional Web Services message acknowledgment of the DPWS application ensuring that the DPWS message is sent and received. This has guaranteed the overall effectiveness of the device communication.

DPWS Application and Embedded Controller Resources

Another factor to be considered is the size of the DPWS resource and memory usage on the device, as this limits device capability and the number of DPWS components that can be co-located on a single control device which affects control system development costs. As detailed in *Chapter 8-section 8.4.1*, the DPWS library consumed almost 50% (240k bytes) of the currently available disk (ROM) space (512k bytes): a significantly high figure. Regarding other resources, each controller device can hold a maximum of three components which is reasonable for the implementation outlined in this thesis since the application on each device only performs the server task. The client task, e.g. the service execution, runs separately on the PC. However, both the end-user and industrial machine automation suppliers would prefer a fully-distributed control system, enabled by peer-to-peer automation devices (i.e. device to device interaction), rather than a PC-based orchestration and centralised PLC- based control system. To satisfy these requirements, the embedded control device needs a larger memory specification, in order to host both the DPWS server and client application of (possibly) multiple components co-located on the device. Increasing the memory specification on the embedded control device in order to support the Web Services application is not a critical issue, as the price of memory is reducing. Any increase in costs associated with upgrading memory sizes will not pose a major increase in control system cost. Also, the cost of embedded devices is much lower than the conventional PLC-based control systems.

Note: although the 512 Kbyte memory is sufficient to hold one component (with the server and the client application in this implementation), the level of component granularity needs to be taken into account, due to the trade-off between a degree of coupling and cohesion that will affect the cost of the control system development. As reported in [112], if the level of granularity is too fine, the integration cost is high but development cost is low, and vice versa. The optimum level needs to be carefully considered.

Hardware Cost within the Control System

A consideration of hardware costs within the control system involves addressing the scalability of the WS based approach. In this case, the cost comparison between the WS enabled device and the PLC (Schneider Electric Premium) -based automation

system can be used to assess the feasibility of migrating to WS automation systems in the future. Although the PLC products are varied, it is noted that the compare prices of PLC's intend to assess on the standard and commercial use equipments in the industry.

Table 9-7: FTB and PLC Cost Comparison

Case	Control Architecture	Control Hardware	CPU Unit	Remote I/O	Total Cost
4 machine stations					
1	Centralised PLC	PLC x 1	1	4	£5,305 ^(f)
		£825/unit	£3480/unit	£250/unit	
2	Distributed FTB	FTB x 4	Included	Included	£896 ^(g)
		£224/unit	-	-	
2 additional machine stations					
3	Centralised PLC	PLC x 1	1	2	£4,805
		825/unit	£3480/unit	£250/unit	
4	Distributed FTB	FTB x 2	Included	Included	£448
		224/unit	-	-	

(f), (g) - The calculated prices (checked on February, 2009) are obtained from RS Component Supplier, Online catalogue available from www.rs-components.co.uk

As detailed in Table 9-7, the cost of the FTB embedded device control system is much lower than the PLC- based control system, as the FTB hardware itself is relatively cheap. Also, a centralised PLC is designed to accommodate a large amount of I/O's (e.g. *Modicon TSX premium PLC*: 2048 discrete and 256 analog) it is generally the case that full I/O capacity is never used. However, each of the distributed FTB devices is designed to handle I/O functionality in smaller volumes, with 16 channels for the distributed automation environment. This means that the FTB is more cost efficient, in terms of capacity use/device, directly affecting the control system cost if machine capacity or new functionalities are added to the existing system. For example, if the end-user requires more process capacity by adding a new machine, which has 2 new stations composed of 4 components, with 22 input and output channels (c.f. the FORD-FESTO test rig), assuming that the end-user needs separate hardware for the control system, the comparison of costs between adopting a PLC or an FTB-based control system performing the same functionality is illustrated by cases 3 and 4 in Table 9-6 i.e. the cost for the new system within the centralised PLC

system is about 10 times higher than the FTB- based control system. This shows that the implemented WS approach is particularly cost efficient, is more scalable due to the finer grain of cost paid per required function.

9.8 Fulfilling the End User Requirements within Agile Automation

This research has demonstrated the feasibility of adopting WS for automation systems via an industrial test rig demonstration. The analysis work and the test scenarios have shown that the WS-based automation approach is able to meet the requirements of agile automation systems, and the WS approach as adopted in this research can be exploited for the manufacturing systems within the following areas:

To Enable the Design of Reconfigurable Automation Systems:

Web Services complement component-based (CB) design approaches within the DPWS via the encapsulation of low-level device programming. This allows the control builder to alter the component and process configuration without having to address low-level device control issues.

Enable the Design for Reuse:

In the generic design of the DPWS device, where the component is differentiated by the target namespace, the incorporation of a new component into the system could reuse the control applications from common / similar components. Also, current components can be reused as programming references, since the development of the control systems within the WS approach is clearly defined and follows similar patterns.

Enable Self-Contained Component Information:

The DPWS provides rich information about each component through the *WS-Metadata* services initialised on each control device. Initialised information, such as device names, firmware version, date of deployment and expiration can be obtained by the DPWS client applications through device look up and discovery services. This metadata information enables the use of process management and preventive maintenance software to keep track and monitor the control system throughout its lifecycle.

Enable Process Visualisation and Data Monitoring:

The test rig automation system has been integrated with production and business applications e.g. the SAP xMII application. Web Services provide the live state of the components and work piece status. These data can be used to capture process activity and to determine the process performance. Web-Services have been implemented with diagnostic capability, in order to report automatically errors, such as actuator malfunction and communications loss to the monitoring and execution control systems.

Integrate 3D Machine Visualisation:

The process visualisation of the test rig on the Web Services platform is ongoing and included in the PDE suite of engineering tools. At this stage, integration of 3D modelling of the live process on the test rig have been prototyped as a demonstration of capability, with the component state linked via UDP multicast from the *service orchestration engine* to the PDE tool.

Enable Component Validation:

As work concerning the PDE is ongoing, the WS component validation of the device operation has been undertaken on a simulator using an MS-DOS console. The client projects for device operation and execution have been implemented in the Visual Studio.Net platform on Windows, for debugging and testing.

Open Automation Platform:

The Web Services automation system utilises a SOAP-XML message exchange, as a common communication that enables inter-operability between different vendor devices implemented within the DPWS. In this research work, it has been established that WS implementation within the DPWS standard can be ported to any embedded microcontroller devices or PLC's within the compliant DPWS gateway, as the DPWS application has been developed on the C/C++ standard to these control devices.

Adopt Local Data Logging and Historian Utility:

As the embedded device does not scale for saving data due to memory constraints, the data record utility has been implemented on the integrated application running on the PC. As the *service orchestration engine* directly interacts with the control system, the

data is captured into a logged file within the application. In this research, the data log is implemented in the WS control system, and records the component state information during runtime. This capability provides track record and data history, in order to support effectively maintenance in the event of a machine breakdown.

Unifying Platform for Higher-Level Production/Business Application Integration:

The DPWS device and the WS application interface provide a seamless application integration platform, achieved by the unifying SOA middleware for all Web Services enabled applications. This WS middleware eliminates the need for custom interfaces tailored for specific control systems and integrated applications. However, further development work, in collaboration with business application providers such as the SAP Company, is required to enhance the seamless integration by directly integrating their applications to the control device within the DPWS.

Distributed Autonomous Automation System:

The design of Web Services-based control devices supports the distribution of component functionalities into local control devices. In the discrete-control system design of the FORD-FESTO test rig, the distributed component reacts accordingly to its environment, defined by the set of component state transitions and interlocks. The implementation of the control system has been demonstrated on the test rig, and the system performance has met the operational requirements of automation systems. However, due to memory resource constraints on the embedded device, the decision and execution making of the embedded device is done separately on the *service orchestration engine*. Further research needs to be carried out on the implementation of autonomous automation systems enabled by the peer-to-peer communication approaches.

9.9 Conclusion

The summary of all the evaluations of quantitative and qualitative measures, as demonstrated in this research, are shown in the table below:

Control system	PLC	LonWorks	FTB
Scalable system cost (Controllers, I/O modules)	£££	££	£
Machine processing speed ^(l)	29.7 seconds	N/A ^(h)	32.5 seconds
Input to Output response time	9-11 ms ⁽ⁱ⁾	57-200 ms ⁽ⁱ⁾	22-55 ms ^(k)
Component (Software) reusability	√√	√√	√√
Seamless integration and complexity (Business- Shop floor applications)	√	√	√√
System reliability	√√	√√	√
Control system development time	√	√	√√
Ease of system reconfiguration	√	√√	√√
Interoperability among different control systems (Open platform)	√ Required gateway/translator	√ Required gateway/translator	√√ No additional gateway/translator

√√ Advantage choice over √ in the same category

(h)- It is not applicable for a comparison due to the LonWork control system was implemented on the different machine.

(i)- The value is obtained from [15] where the same PLC and remote I/O modules have been used.

(j)- The value is obtained from S.M. Lee dissertation [33].

(k)- The value is derived from the minimum and maximum response time boundary by the experiment as presented in section 9.2.3.

(l)- Time to process the workpiece from the first to the last station

Analysis of the TCP/IP communication approach, in relation to message reliability and the SOAP message structure (based on the component design) as the WS communication between client and the service applications in the control system have been detailed in this chapter. The responsiveness of interacting I/O has been measured via a timing analysis of Network, DPWS processing and I/O processing. The qualitative features of the control WS system have been assessed by their performance, with respect to various parameters such as network bandwidth, utilisation rate, machine cycle time and the cost of control hardware. The DPWS is able to deliver the real-time capability of event interaction in around 0.031 seconds, and the implemented WS control system is able to achieve a work piece processing

time of 33 seconds (the PLC- based processing time is 30 seconds on the same FORD-FESTO test rig with the same machine configuration). The cost of developing the embedded control platform (e.g. the FTB controller) is substantially lower than the PLC, and this is due to the combined module of the Ethernet communication interface, in addition to the low market prices for embedded controllers.

Evaluation of the qualitative features of WS were measured in terms of reusability, re-configurability and business integration of the WS control system, and compared with the PLC based and Lonwork systems. These are the key evaluation areas characterised by agile manufacturing (*Chapter 3*). In this research, the WS automation system has demonstrated a good level of re-configurablity and reusability, in terms of activity and skills involved when compared with the PLC and LonWorks based control systems. WS's also provides a better quality of process/shop floor system integration as required by end-users to support collaborative manufacturing frameworks. Although the implementation of WS on automation devices requires effort, in terms of improving reliability, robustness and support tools for building a more user-friendly control system, the evaluation undertaken in this thesis has shown that the WS approach is indeed feasible and suitable to be adopted for the next generation of competitive automation systems in the industrial domain.

CHAPTER 10

Conclusion and Future Work

This chapter concludes the author's research findings and details the contributions of the proposed Web Services and component-based design approach for agile automation systems. A summary of the core studies, research demonstrations and analysis results is provided, in addition to research achievements and suggested future work.

10.1 Research Conclusion

The research has been focused on adopting a SOA and WS approach for automation systems incorporating the component-based design approach previously developed at the MSI Research Institute at Loughborough University. However, the author's original work and novel contributions in extension and collaboration with the other MSI work have been presented in Figure 10-1.

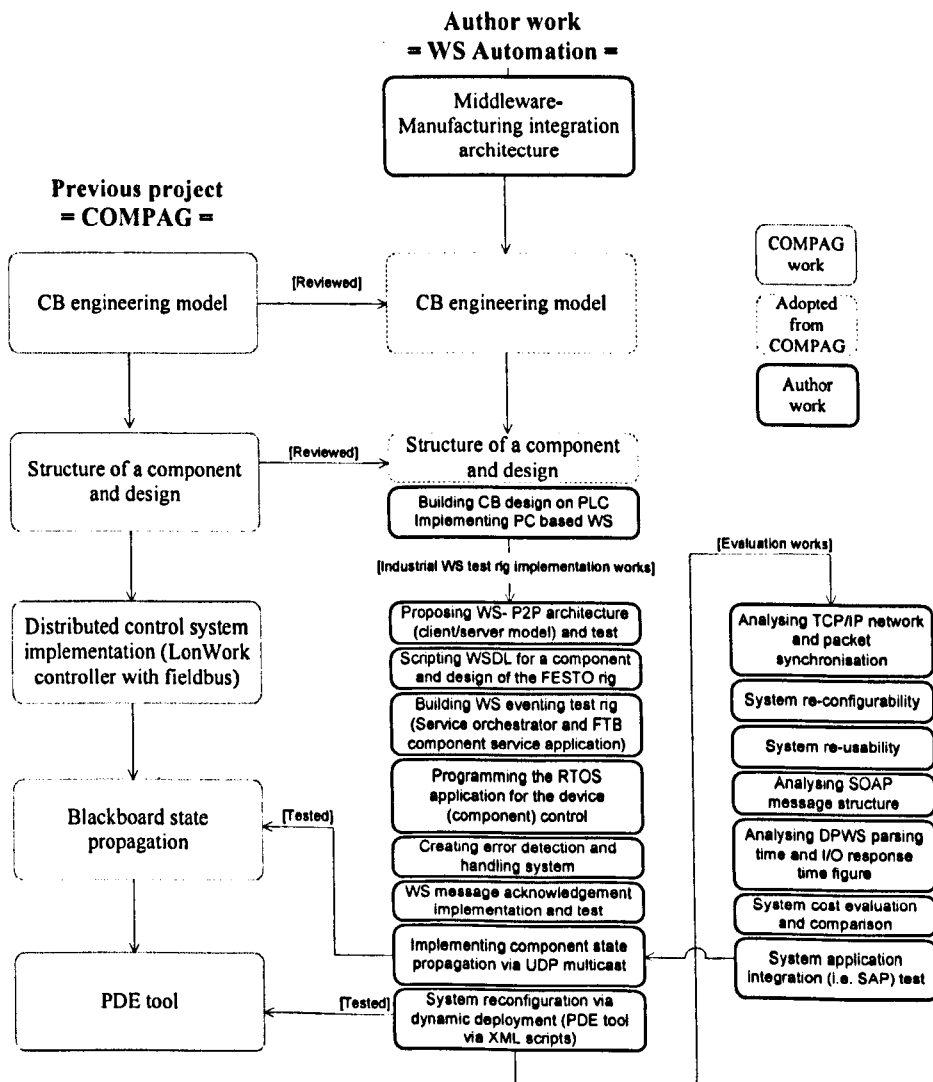


Figure 10-1: The Author Original Work and Novel Contributions

The details are summarized as follows:

10.1.1 Research Findings

An approach for the design and implementation of agile automation systems, with respect to system reconfiguration and seamless business application integration has been researched in this thesis. Reviews of reconfigurable manufacturing systems and process-business integration have provided a useful references and a framework for the conceptual design of WS based automation systems. The core strategy behind the component-based design for the reconfigurable and reusable machine components, has been identified and a focus on seamless connectivity and heterarchical enterprise architectures adopted to support business integration with shop floor system.

10.1.2 Contribution to Knowledge

This research has proposed the novel idea of integrating the CB design approach with WS, and the work has provided a clear structure of how to implement the WS approach on the automation devices. The approach may be used as a core design paradigm for future automation systems in similar domains. The research has highlighted robust evidence of positive WS performance on embedded control device platforms when compared with PLC and alternative distributed systems technology, demonstrating the feasibility of adopting the SOA approach within manufacturing systems.

In summary, this research has developed a unique design framework for WS enabled control devices, which brings the component-based (CB) design approach and the proposed SOA middleware (i.e. DPWS and WS application interface) technology into the component constitution. The WS in this research utilise the CB approach in order to facilitate the design of control applications for reconfigurable and reusable manufacturing systems. This approach allows the decomposition of components and their assembly into a loosely coupled system with SOAP-XML message exchange formats utilised to generate the manufacturing application tasks.

10.1.3 Implementation and Evaluation

The implementation of the WS based automation system has been conducted on an industrial test rig for the purposes of demonstration and evaluation. The evaluation research on the proposed WS control system has been carried out in detail, in order to illustrate the knowledge and findings behind the Web Services approach in the following areas:

Ethernet Network Performance

The primary concerns of WS within the automation system i.e. the delivery of robust timing performance and message reliability have been assessed. The approach of WS communication, in respect of the TCP/IP packet synchronisation, has been identified, in addition to a SOAP text based structure within the DPWS operation. Also, an analysis of the DPWS processing time has been undertaken, using a protocol analyzer and set timers on the DPWS servers and client applications. The results have demonstrated that the speed of the DPWS (I/O response time 22-55 ms) on the selected embedded platform could be able to meet the expectations of end-users (in particular, the FORD Company) in terms of soft real time criteria and the required I/O response time (under 30 ms).

Process Re-configurability

The capability of WS based automation, in terms of reusability and re-configurability, has been evaluated and compared with the conventional PLC based system, in a number of industrial test scenarios. In this context, WS facilitated the modular design of the component and the management of dependencies between components (i.e. component interlocks) and state transition conditions. The advantages of reconfiguration and parameterisation for process adaptation have been explicitly demonstrated by the test scenarios in Chapter 9-section 9.3.

Process Integration

The contribution of WS towards seamless integration has been demonstrated and assessed using the SAP xMII application and DPWS enabled middleware. This middleware was implemented as the unifying interface that allows diverse production and business supported applications to be connected throughout the enterprise. As the research work has demonstrated in Chapter 8- section 8.5.3, non-DPWS applications can be integrated into the WS based automation environment, via a mediator that participates in both the WS application and DPWS interface. A business application, such as SAP, can access the control system information at mediator level, where it is enabled with the DPWS interface to interact directly with the control devices.

Production Support System

WS-Metadata, initialised on WS enabled components, provides rich component information regarding device names, software versions, manufacturer, starting date, expected end of life. This information can be retrieved by the implemented client DPWS applications for component monitoring and servicing applications used in maintenance systems. In this research, the process monitoring (remote HMI) and simulation (3D modelling) applications, both part of the PDE tool development at MSI research group, have been tested in conjunction with the WS-based automation system in order to provide visualisation of automation systems for control application validation and control (*Chapter 9- section 9.5*). A proof-of-concept data logging utility has also been implemented, to record the device operating history. This can be used by maintenance staff for preliminary analysis of machine capability.

10.2 Research Achievements

This research has contributed to the development and investigation of a new automation methodology that incorporates the use of the Service Orient Architecture and Web Services technologies (SOA-WS) to enhance the lifecycle of manufacturing systems. The research work has been focused on the adoption of a WS based automation paradigm for reconfigurable and reusable automation systems supporting the seamless connection of the manufacturing automation and business systems.

Evaluation research (in relation to the end-user requirements of agile automation) has been carried out to determine quantitative parameters. The analysis work carried out in this research have provided a clearer understanding of the WS automation approach in addition to suitability of Ethernet communication mechanisms within the automation research field.

The objectives of the research were achieved, regarding the industrial test rig demonstration and evaluation, in the following areas:

- An innovative design approach for WS enabled control devices for the industrial distributed control systems. The WS based automation systems adopted in this research are radically different from conventional automation systems, with an emphasis placed on the approach of building control applications and the device consistent interfaces for higher level application integration. The implementation of the WS-based component programming allows low-level control hardware programming to be separated from building and reconfiguring the machine application, and this approach significantly reduces engineering effort, allowing engineers to focus their core competencies at process focused web service level. Also, the complexity of building control applications, in respect of device interaction and message exchange, is managed by the DPWS functionality through unique device namespace for resource allocation, discovery, and invocation.
- The novel development and implementation of the event-driven decentralised control systems with DPWS enabled control devices. The research has determined the communication specifications regarding message exchange among components and distributed service orchestration applications. From a business process workflow perspective, this research has proven the implemented WS approach in its ability to control and manage production lines from the process control level within the manufacturing system. From a shop floor automation perspective, the demonstrated prototype has provided a manufacturing platform for the migration path to fully distributed and peer-to-peer automation systems, which will allow components to react autonomously to the manufacturing environment.

- The open automation platform effectively facilitates the ease of heterogeneous application integration, such as remote monitoring, data acquisition and business process planning applications. The provision of the DPWS and WS application interface in the SOA middleware has enabled a consistent interface for seamless integration between automation and other manufacturing supported applications.
- Novel findings of the architecture and implementation of the WS based automation system on embedded microprocessor devices with the office standard Ethernet network have been discovered. For the deployment of Web services to other automation and manufacturing applications, the software platform implemented in this research can be directly applied to any other general controller equipment that runs on C/C++ compilers.
- A formal experimental evaluation has been used to assess the feasibility of using Web Services enabled control devices within industrial automation systems, particularly the powertrain assembly of automotive automation. The experimental and data analysis work of the WS based automation has emphasised the Ethernet network communication approach, performance, and speed and explicitly outlined and justified the adopting of Web Services within the distributed control system.
- Ease of process design and reconfiguration. Although integration within the process-engineering environment has not been demonstrated due to this being outside the scope of this thesis, a machine and process reconfiguration framework and strategy have been outlined. The research work provides a common representation of the process reconfiguration procedure and reuse of the WS software component. Also, the strategy for the dynamic deployment framework through engineering tools has been provided, in order to support the realisation of the integrated PDE tool and other control build applications.

Table 10-1: The Research Implementation Summary

		Key agile enablers																					
		Information sharing system	Open control platform	Seamless network connection	Control system simulation/validation	Component-based design automation	Process engineering tools	Distributed control system															
End-user requirements	Reconfigurable automation system and Dynamic deployment	✓						✓	✓	•	•	•											
	Ease of design and installation	✓							✓	•	•												
	Design for reuse of automation components	✓							✓	•	•												
	Pre-design evaluation		✓											•									
	Unifying application interface for business- shop floor system integration			✓				✓														•	
	Open, non-vendor specific automation solutions			✓				✓															•
	Process monitoring and automated error warning system		✓				✓																•
	Lower control device cost								✓														
	Meet the soft-real time performance								✓														
		Generic design of WS control components	Device state propagation (via direct TCP/IP binding and DPWS subscription)	SOA Middleware (devices and higher level applications)	Local device error and MES level diagnostic	Ethernet based control system	Low cost embedded controller	Distributed WS based automation	Integration of state orchestration engine application and the PDE tool														
		Research implementation																					

The summary of the results delivered by this research, in relation to the end user requirements and key agile enablers addressed in Chapter 6 and Chapter 3 respectively, is presented in Table 10-1.

10.3 Future Work

This research has shown a promising result for adopting WS based automation systems, in order to achieve an agile automation system within the automotive manufacturing sectors. However, research in this field is still in progress, and this can be extended in various domains to enhance networking performance and maximise usability within the process-engineering environment. As the result of the analysis

work in this research, there are issues concerning the capability of WS that remain to be investigated.

- Most of the time taken regarding DPWS communication is expensively consumed by the DPWS parsing (i.e. encoding and decoding the SOAP DPWS message- 20.76 ms). This is around 70% of the embedded I/O device response time (31.08 ms) capability. This further limits performance significantly.
- Although the Web Service platform has demonstrated a fundamental ability to support the ease of building and reconfiguring the machining process, it is desirable, from the end-user point of view, to have a process engineering tool with graphical user interface that fully supports integration of WS devices, in order to avoid the low level coding of the WS application. Such tools are not yet available.
- The realisation of fully distributed autonomous automation systems with WS enabled control devices has not been fully developed and tested, due to hardware constraints. In particular of the specification from the CPU and sufficient memory at a suitable cost.

Related future work can be undertaken in the following areas:

Binary SOAP-based Message

SOAP messages use a text-based representation [67], as outlined in this research in Chapter 9- section 9.2.1. As the format is ASCII text, there are associated costs of conversion, from binary to ASCII and vice versa [62]. It is reported by [62], [66], and [67] that the SOAP protocol (in a text-based format) has often been regarded as relatively poor, requiring significant amounts of processor time and bandwidth in comparison with the binary fixed format implemented in CORBA. There is ongoing research and proposed solutions concerned with optimising the SOAP parsing speed, particularly in the SOAP binary format. Some proposed solutions, such as Compact XML tag [67], Byte Sequence Memorization/ Pattern recognition [63], Table Driven XML [66], and XML-binary Optimized Packaging [61] have shown promising results in improving the SOAP message processing performance. However, these approaches are based on the PC applications generally used in the business and Web application

domains. Adoption of these approaches or similar concepts within the automation domain is still dependent on future research, in order to find the most suitable and feasible approach for embedded control devices.

Implementation of a Peer-to-Peer DPWS-based Automation System

As the functions of the server and client have been scoped in the automation system, the realisation of the Peer-to-Peer approach for autonomous control systems is reflected by deploying the producer (server) and the consumer (client) onto the same device. This enables direct interaction amongst distributed control devices, and the architecture is presented in Chapter 6- section 6.9.2-2.

Integration of the PDE Suite tool and WS-Management Support

This research has illustrated the activities involved in component design, control system build and reconfiguration and has provided the guidelines and a platform for research work on process engineering tool integration at the MSI research institute. As far as the WS enabled control devices are concerned, in relation to the process engineering environment for building/validating the component logic and process design and simulating the process runtime system, the design of the Web Services control device needs to support this engineering environment. Thus, future works are needed in the following areas:

- **Process configuration:** An interface between WS-Management, Control builder application and the PDE tool is required for control data configuration download. The control logic could then be uploaded via the XML control data configuration format from the PDE, which that can be managed by the embedded WS-Management functionality on the control device.
- **Design time visualisation:** For the accuracy of component validation, the logic of WS component design (i.e. component states and state transition conditions) is required to match the simulation logic running on the PDE tools. This requires mapping between the DPWS component description (WSDL) and the PDE tool simulator logic.

- **Runtime visualisation:** This is conducted by the broadcaster application in the PDE integrated tool. As presented in this research, work has been done on integration of the broadcaster via the multicast state propagation from the service execution engine. Future work could be the direct integration between the broadcaster and DPWS-enabled control devices, via the DPWS interface, for seamless connection.

System robustness and reliability

The further improvement on the system reliability regarding to lost messages and error recovery needs to be addressed and managed. Additional message handling at the application level is proved to be sufficient, but however, this adds extra messages and complexity to the control system. In this case, the simultaneous event notification based on multicast approach could potentially resolve this problem by reducing the amount of messages and traffics on the network. However, this approach needs to consider additional message guarantee of the multicast approach which is not include in the standard Ethernet.

Implementation of the DPWS on other Generic Embedded Control Devices

This research has proven that Web Services can be deployed onto the standard embedded microprocessor controller, in order to operate in the industrial manufacturing task environment. Considering that the cost of the embedded device is becoming cheaper (substantially cheaper than conventional devices such as the PLC), the full exploitation of Web Services as an open automation platform on low-cost embedded devices could yield significant savings within automation. Moreover, the software platform in this work is applicable to other control platforms, as the application on the embedded device contains the same functionality and concepts as in the design of control automation. Although the results of a WS common platform were delivered in this research, the integration of the DPWS within the various RTOS and TCP/IP layers on different devices requires further investigation and implementation studies.

REFERENCES

- [1] A. Molina, et al., "Next-generation manufacturing systems: key research issues in developing and integrating reconfigurable and intelligent machine," in *International Journal of Computer Integrated Manufacturing*, vol. 18, pp. 525-536, Oct-Nov 2005
- [2] A. Gunasekaran, and E.T. Nagib, "Build-to-order supply chain management: a literature review and framework for development," in *Journal of Operations Management*, vol. 23, no. 5, pp. 423-451, July 2005
- [3] P. R. Dean, D. Xue, Y. L. Tu, "Prediction of manufacturing resource requirements from customer demands in mass-customisation production," in *International Journal of Production Research*, vol. 47, no. 5, pp. 1245 – 1268, January 2009
- [4] M.F. van Assen, E.W. Hans, and S.L. van de Velde, "An agile planning and control framework for customer-order driven discrete parts manufacturing environments," *International Journal of Agile Management Systems*, vol. 2, no. 1, pp. 16 – 23, 2000
- [5] L.J. Lee, "A Next Generation Manufacturing Control System," *PhD dissertation*, MSI Research Institute, Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, 2003
- [6] P. Leitão, A.W. Colombo, and F. Restivo, "A Formal Validation Approach for Holonic Control System Specifications," a book on *Holonic and Multi-Agent Systems for Manufacturing*, Publisher: Springer Berlin / Heidelberg, 2004, vol. 2744/2004
- [7] G. Büyüközkan, T. Dereli, and A. Baykasoğlu, "A survey on the methods and tools of concurrent new product development and agile manufacturing," in *Journal of Intelligent Manufacturing*, vol. 15, pp. 732-751, 2004
- [8] A. Gunasekaran, "Agile manufacturing: enablers and an implementation framework," *int. journal prod. res.*, vol. 36, no. 5, pp.1223-1247, 1998
- [9] F. Jammes, and H. Smit, "Service-Orient Paradigms in Industrial Automation," in *IEEE Transactions on Industrial Informatics*, vol. 1, no. 1, pp. 62-70, 2005
- [10] A.W. Colombo, R. Schoop, and P. Leitão, "A Collaborative Automation Approach to Distributed Production System," in *2nd IEEE International Conference on Industrial Informatics*, pp. 27-32, 2004
- [11] S.M. Lee, R. Harrison, and A.A. West, "A Component-based Distributed Control System," in *2nd IEEE International Conference on Industrial Informatics*, pp. 33-38, 2004
- [12] D.C. McFarlane, and S. Bussman "Holonic manufacturing control: rationales, developments and open issues," in *Deen, S.M., (ed.) Agent Based Manufacturing - Advances in the Holonic Approach*, Springer, 2002
- [13] T. Wagner, "An Agent-oriented Approach to Industrial Automation Systems," in *3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Business MALCEB'2002*, Erfurt/Thuringia, Germany, 2002
- [14] A.W. Colombo, et al., "Service-oriented architectures for collaborative automation," in *31st Annual Conference of IEEE (IECON 2005)*, November 2005
- [15] B. Denis, et al., "Measuring the impact of vertical integration on response times in Ethernet fieldbuses," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA'07)*, pp 532- 539, September, 2007
- [16] L. Lin, M. Wakabayashi, and S. Adiga, "Object-Oriented Modeling and Implementation of Control Software for a Robotic Flexible Manufacturing Cell," in *Journal of Robotics and Computer-Integrated Manufacturing*, vol. 11, no. 1, pp. 1-12, 1994
- [17] Y.M. Hu, R.S. Du, and S.Z. Yang, "Intelligent Data Acquisition Technology Based on Agents," in *International Journal of Advanced Manufacturing Technology UK*, vol. 21, no.10-11, pp. 866-73, 2003

- [18] **M. Fletcher, R.W. Brennan, and D.H. Norrie**, "Modeling and Reconfiguring Intelligent Holonic Manufacturing Systems with Internet-based," in *Journal of Intelligent Manufacturing*, vol. 14, pp. 7-23, 2003
- [19] **L. Wang**, "Integrated design-to-control approach for holonic manufacturing systems," in *Journal of Robotic and Computer- Integrated Manufacturing*, vol. 17 pp. 159-167, 2001
- [20] **B. M. Shahzada, et al.**, "Creating Customer Value through SOA and Outsourcing: A NEBIC Approach," in *International Journal of Social Sciences*, vol. 4, no. 1, 2009
- [21] **P.M. Nadkarni, and R. A. Miller**, "Service-oriented Architecture in Medical Software: Promises and Perils," in *Journal of the American Medical Informatics Association: JAMIA*, vol. 14, no. 2, pp. 244-246, 2007
- [22] **O.J. Orozco, and J.L. Lastra**, "Adding Function Blocks of IEC 61499 Semantic Description to Automation Objects," in *IEEE Conference on Emerging Technologies and Factory Automation*, pp. 537-544, September 2006
- [23] **D. Cachapa, et al.**, "An Approach for Integrating Real and Virtual Production Automation Devices Applying the Service-oriented Architecture Paradigm," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 309-314, September 2007
- [24] **V. M. Trifa, D. Guinard, and M. Koehler**, "Messaging Methods in a Service-Oriented Architecture for Industrial Automation Systems," in *International Conference on Networked Sensing Systems (INSS)*, pp. 35-38, June 2008
- [25] **H. Liu, M.L. Li, and X. Lin**, "Mapping Web Services Standards to Federated Identity Management Requirements for m-Health," in *International Conference on Internet Computing in Science and Engineering*, pp. 459-466, 2008
- [26] **F. Jammes, A. Mensch, and H. Smit**, "Service-Oriented Device Communications using the Devices Profile for Web Services," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, 2007
- [27] **M. Hirsch, V. Vyatkin, and H.M. Hanisch**, "IEC 61499 Function Blocks for Distributed Networked Embedded Applications," in *IEEE International Conference on Industrial Informatics*, pp. 670-675, August 2006
- [28] **B. Dutta**, "Semantic Web Services: A Study of Existing Technologies, Tools and Projects," in *DESIDOC Journal of Library & Information Technology*, vol. 28, no. 3, pp. 47-55, May 2008
- [29] **M.H. Hung, F.T. Cheng, and S.C. Yeh**, "Development of a Web-Services- Based e-Diagnostics Framework for Semiconductor Manufacturing Industry," in *IEEE Semiconductor Manufacturing*, vol. 18, no.1, February 2005
- [30] **R. Fan, L Cheded, and O. Toker**, "Java plus XML; A powerful new combination for SCADA systems," in *IEEE Computing & Control Engineering Journal*, October/November 2005 issue
- [31] **D.M. Dilts, N.P. Boyd, and H.H. Whorms**, "The Evolution of Control Architectures for Automated Manufacturing Systems," in *Journal of Manufacturing Systems*, vol. 10, no. 1, pp. 354, 1991
- [32] **J. Yu and K. Krishnan**, "A conceptual framework for agent-based agile manufacturing cells," in *Info System Journal*, vol. 14, pp. 93-109, 2004
- [33] **S.M. Lee**, "A Component-Based Distributed Control Paradigm for Manufacturing Automation System," *PhD dissertation*, MSI Research Institute, Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, 2004
- [34] **V. Issarny, M. Caporuscio, and N. Georgantas**, "A Perspective on the Future of Middleware-based Software Engineering," in *International Conference on Software Engineering*, pp. 244-258, 2007

- [35] **M. A. Mastouri, and S. Hasnaoui**, "Performance of a Publish/Subscribe Middleware for the Real-Time Distributed Control systems," in *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 7, no.1, January 2007
- [36] **J. Schlesselman and B. Murphy**, "Publish-Subscribe Approach Aids Military Comms," in *COTS Journal, Real-Time Innovations*, March 2003
- [37] **J. Park, S. Mackay, and E. Wright**, "Practical Data Communications for Instrumentation and Control," *IDC Technology*, Publisher Oxford: Newnes, 2003
- [38] **P.G. Ranky**, "Modular fieldbus design and applications," in *MCB Journal of Assembly Automation*, vol. 20, pp. 40-45, November, 2000
- [39] **K. Bothamley and J. Rodgerson**, "Emerging Ethernet Protocols," Department of Electrical and Electronic Engineering, Manukau Institute of Technology, Auckland
- [40] **D. Heffernan, and P. Doyle**, "Time-triggered Ethernet based on IEEE 1588 clock synchronization," in *Journal of Assembly Automation*, vol. 24, no. 3, pp. 264-269, 2004
- [41] **V. Marik, and D. McFarlane**, "Industrial Adoption of Agent-Based Technologies," in *IEEE Intelligent Manufacturing Control*, vol. 20, no. 1, pp. 27-35, January/February 2005
- [42] **J. Camerini, A. Chauvet, and M. Brill**, "Interface for Distributed Automation: IDA," in *IEEE International Conference on Emerging Technologies and Factory Automation*, vol.2, pp. 515-518, 2001
- [43] **W. Zhang, W.A. Halang, and C. Dietrich**, "Specification and Verification of Applications Based on Function Blocks," *a book on Component-Based Software Development for Embedded Systems*, Publisher: Springer Berlin / Heidelberg, pp. 8-34
- [44] **I. Plaza, C. Medrano, and A. Blesa**, "Analysis and implementation of the IEC 61131-3 software model under POSIX Real-Time operating systems," in *Journal of Microprocessors and Microsystems*, vol. 30, pp. 497-508, June 2006
- [45] **L.M Sa de Souza, et al.**, "SOCRADES: A Web service based Shop Floor Integration Infrastructure," in *The Internet of Things Book*, Publisher: Springer Berlin / Heidelberg, vol. 4952, March 2008
- [46] **R. Harrison, et al.**, "D7.2-Integration of Enabling Technologies, Methods and Tools for Application Engineering," *SOCRADES project internal document*, October, 2007
- [47] **S. Karnouskos, et al.**, "D6.1- Services integration concept for field related data into business processes," *SOCRADES project internal document*, April, 2007
- [48] **SOCRADES FP6**, "Annex I- Description of Work," *SOCRADES project internal document*, April, 2006
- [49] **S.C. Yu and R.S. Chen**, "Web services: XML based system integrated techniques," in *Emerald Journal Electronic library*, vol. 21, no. 4, pp. 358-366, 2003
- [50] **H. V. Brussel**, "A conceptual framework for holonic manufacturing: Identification of manufacturing holons," in *Journal of Manufacturing Systems*, vol. 18, No. 1, pp. 35-52, 1999
- [51] **L. Wang, Y. Songa and Q. Gaoa**, "Designing function blocks for distributed process planning and adaptive control," in *Engineering Applications of Artificial Intelligence Journal*, November 2008
- [52] **R. A. van Engelen**, "Code generation techniques for developing light-weight XML Web Services for embedded devices," in *ACM SIGAPP SAC Conference (Embedded Systems track)*, 2004
- [53] **P. Kearney**, "Message level security for web services," in *Journal of Information Security Technical Report*, vol. 10, no. 1, pp. 41-50, 2005
- [54] **A. Mensch, and S. Rouges**, "DPWS toolkit version 2 user guide," *Schneider Electric*, October 2005

- [55] **M. Gardoni, C. Frank and F. Vernadat**, "Knowledge capitalisation based on textual and graphical semi-structured and non-structured information: case study in an industrial research centre at EADS," in *Computers in Industry Journal*, vol.56, pp. 55–69, 2005
- [56] **H. K. Wei**, "Design a Jini-based Service Broker for Dynamic Service Combination Framework," in *Journal of Software*, vol. 1, no. 3, September 2006
- [57] **Y. Wang, and E. Stroulia**, "Semantic Structure Matching for Assessing Web-Service Similarity," in *Service-Oriented Computing - ICSOC 2003 Book Journal*, Publisher: Springer Berlin / Heidelberg, vol. 2910, 2003
- [58] **G. B. Machado et al.**, "Integration of embedded devices through Web Services: Requirements, Challenge and Early results," in *11th IEEE Symposium on Computers and Communications (ISCC'06)*, pp. 353-358, 2006
- [59] **K. Ishii**, "Life-cycle Engineering Design," in *ASME Journal of Mechanical Design*, vol. 117, pp.42-47, 1995
- [60] **R. Glaschick and B. Oesterdieckhoff**, "Service-Oriented interface design for embedded devices," in *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'05)*, vol. 2, pp. 288-295, 2005
- [61] **M. Girardot, and N. Sundaresan**, "Millau: an encoding format for efficient representation and exchange of XML over the Web," in *International Journal of Computer and Telecommunications Networking*, vol. 33, no. 1-6, pp.747-765, June 2000
- [62] **K. Devaram and D. Andresen**, "SOAP Optimization via Parameterized Client-side Caching," in *the Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, pp. 785- 790, November 2003
- [63] **T. Takase, H. Miyashita, T. Suzumura, and M. Tatsubori**, "An Adaptive, Fast and Safe XML Parser Based on Byte Sequence Memorization," in *WWW2005 Conference*, May, 2005
- [64] **P. T. Kidd**, "Agile Manufacturing: a strategy for the 21st century," in *IEE Colloquium on Agile Manufacturing*, pp. 1-6, Oct 1995
- [65] **M. Stal**, "Web Services: Beyond component-based computing," in *Communication of the Association for Computing Machinery (ACM) Journal*, vol. 45, no. 10, Oct 2002
- [66] **K. S. Alex**, "Optimising Web Services Performance with Table Driven XML," in *the 2006 Australian Software Engineering Conference (ASWEC'06)*, IEEE Computer Society, April 2006
- [67] **C. Kohlhoff and R. Steele**, "Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems," in *WWW2003 Conference*, May, 2003
- [68] **N. Kakanakov and G. Spasov**, "Adaptation of Web services architecture in distributed embedded systems," in *International conference on computer systems and technologies (CompSysTech)*, 2005
- [69] **W. Chen, S. Y. Kuo, and H. C. Chao**, "Service integration with UPnP agent for an ubiquitous home environment," in *Journal Information Systems Frontiers*, Publisher: Springer, June 2008
- [70] **W. Sperling, and P. Lutz**, "Design applications for an OSACA control," in *Proceedings of the International Mechanical Engineering Congress and Exposition*, Dalles/USA, November 16-21, 1997
- [71] **G. P. Castellote**, "DDS Spec Outfits Publish-Subscribe Technology for the GIG," Real-Time Innovations, in *COTS Journal*, April 2005
- [72] **R. Harrison, A.W. Colombo, A.A. West and S.M. Lee**, "Collaborative automation from rigid coupling towards dynamic reconfiguration production systems," in *Proceedings of 16th IFAC World Control Congress*, Prague, Czech Republic
- [73] **S M Lee, R Harrison, and A A West**, "A component-based control system for agile manufacturing," in *Journal of Engineering Manufacture, IMech*, vol.219, Part B, 2004

- [74] **J. E. Boritz, and W. G. No** "Security in XML-based financial reporting services on the Internet," in *Journal of Accounting and Public Policy*, vol. 24, No. 1, pp. 11-35, January-February 2005
- [75] **V. Marik, P. Vrba, K. H. Hall and F. P. Maturana**, "Rockwell Automation Agents for Manufacturing," in *4th International joint conference on Autonomous agents and multiagent systems (AAMAS)*, July 25-29, 2005, Utrecht, Netherlands
- [76] **S. Graham, et al.**, "Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI," *Sams publishing*, Second edition,
- [77] **K. Cheng, P. Y. Pan, and D. K. Harrison**, "The Internet as a tool with application to agile manufacturing: a web-based engineering approach and its implementation issues," in *Int. J. Prod. Res.*, vol. 38, no. 12, pp. 2743- 2759, 2000
- [78] **T. J. Williams and H. Li**, "PERA and GERAMA- Enterprise Reference Architecture in Enterprise Integration," in *Information Infrastructure Systems for Manufacturing II*, Kluwer Academic Publishers, 1998
- [79] **R. Harrison, S. M. Lee and A.A. West**, "Lifecycle Engineering of Modular Automated Machines," in *2nd IEEE International Conference on Industrial Informatics (INDIN '04)*, June 2004
- [80] **M.P. Stanica and H. Guéguen**, "A Timed Automata Model of IEC 61499 Basic Function Blocks Semantic," in *7th International Workshop on Discrete Event Systems, IFAC*, pp 385-90, September 2004.
- [81] **K. B. Lee and R.D. Schneeman**, "Distributed Measurement and Control Based on the IEEE 1451 Smart Transducer Interface Standards," in *IEEE Transaction on instrumentations and measurement*, vol.49, no.3, June 2000
- [82] **G. Thomas**, "TCP and UDP for Control," in *Industrial Ethernet Book*, Issue 4:29
- [83] **G.B. Machado and R. Mittmann**, "Embedded systems integration using Web Services," in *IEEE International Conference on Networking*, 2006
- [84] **C.R. McLean, H.M. Bloom, and T.H. Hopp**, "The Virtual Manufacturing Cell," in *Proceeding of the IFAC/IFIP Conference on Information Control Problems in Manufacturing Technology*, Gaithersburg, October, 1982
- [85] **R.D. Quinn, et al.**, "Design of an agile manufacturing workcell for light mechanical applications," in *IEEE International Conference on Robotics and Automation*, vol.1, pp. 858-863, 1996
- [86] **R. Harrison and R. Monfared**, "D7.1- User requirements for the application systems engineering and lifecycle support of distributed smart embedded devices," *SOCRADES project internal document*, March, 2007
- [87] **M. H. Selamat, et al.**, "Software Component Models from a Technical perspective," in *International Journal of Computer Science and Network Security (IJCSNS)*, vol.7, no.10, October 2007
- [88] **J. Nern, et al.**, "INFRAWEBs semantic web service development on the base of knowledge management layer," in *International Journal Information Theories and Applications*, vol. 13, no. 1, pp. 161-168, 2006
- [89] **U. Topp, P. Muller, J. Konnertz, and A. Pick**, "Web based services for embedded devices," in *Web Databases and Web Services*, Publisher: Springer-Verlag Berlin Heidelberg, pp. 141-153, 2003
- [90] **R. A. van Engelen and K.A. Gallivan**, "The gSOAP toolkit for Web Services and Peer-to-Peer Computing Networks," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, 2002
- [91] **A. Vaughn, P. Fernandes and J.T. Shield**, "Manufacturing System Design Framework Manual," *Massachusetts Institutes of Technology Lean Aerospace Initiative*, 2002

- [92] **U. Topp, P. Muller, J. Konnertz, and A. Pick**, "Web Based Services for Embedded Devices," in *Web Databases and Web Services Journal*, Springer-Verlag Berlin Heidelberg, LNCS 2593, pp.141-153, 2003
- [93] **T. H. Ess**, "Accessing Devices Using a Web Services," in *IEEE Southeast Conference*, pp 463-467, 2002
- [94] **N. Jazdi and J. Konnertz**, "Localization of distributed internet ready automation devices," *Institute of Industrial Automation and Software Engineering*, University of Stuttgart, Online publications
- [95] **H. Kopetz**, "Component-based design of large distributed real-time systems," in *Control Engineering Practice 6*, Publisher: Elsevier Science, pp. 53-60, 1998
- [96] **K. West**, "SOA vs. CORBA: Not Your Father's Architecture," in *Optimize Magazine*, 2004
- [97] **Z. Kiziltan, T. Jonsson, and B. Hnich**, "On the definition of concepts in component based software development," *Online publications*, Institutionen för Datavetenskap,
- [98] **A. Bracciali, A. Brogi, and C. Canal**, "A formal approach to component adaptation," in *The Journal of Systems and Software*, vol. 74, no. 1, pp. 45-54, 2005
- [99] **I. Crnkovic, M. Chaudron, and S. Larsson**, "Component-based development process and component lifecycle," in *International Conference on Software Engineering Advances (ICSEA)*, October 2006
- [100] **S. C. Lee, and A. I. Shirani**, "A component based methodology for Web application development," in *Journal of Systems and Software*, vol. 71, pp. 177-187, 2004
- [101] **Z. Hu**, "Using ontology to bind Web Services to the data model of automation systems," *Web, in Web-Services, and Database Systems Journal*, Springer Berlin, vol. 2593, pp. 154-168, 2003
- [102] **A. Urbani and S.P. Negri**, "Production System Modelling for the Evaluation of the Degree of Reconfigurability," in *Reconfigurable Manufacturing Systems and Transformable Factories Journal*, Springer, pp. 239-257, June 2007
- [103] **M.H. Ong**, "Evaluating the Impact of Adopting the Component-Based System within the Automotive Domain," *PhD dissertation*, MSI, Loughborough University, October 2004
- [104] **Dr. D.M. Anders**, "MASS CUSTOMIZATION, the Proactive Management of Variety," in *Build-to-Order & Mass Customization Casebook*, CIM Press, 2004
- [105] **C.K. Fan, and T.N. Wong**, "Agent-based architecture for manufacturing system control," in *Journal paper of Integrated Manufacturing Systems*, MCB UP Limited, pp. 599-699, 2003
- [106] **T. Sturgeon**, "Globalization and Jobs in the Automotive Industry," in *Proceeding to International Journal of Technological Learning, Innovation and Development*
- [107] **D. Parsons, A. Rashid, A. Telea and A. Speck**, "An architectural pattern for designing component-based application frameworks," in *Software- Practice and Experience, Wiley InterScience*, vol. 36, pp. 157-190, 2006
- [108] **J. Spatz and P. Nunnenkamp**, "Globalization of the Automobile Industry- Traditional Locations under Pressure?," in *Aussenwirtschaft - The Swiss Review of International Economic Relations*, vol. 57, no. 4, 2002
- [109] **K. H. Otto**, "Modular Product Platform Design," *Doctoral Dissertation*, Department of Mechanical Engineering Machine Design, Helsinki University of Technology, 2005
- [110] **B. Jonsson, et al.**, "Component-based design and integration platforms," in *Embedded Systems Design: The ARTIST Roadmap for Research and Development Journal*, vol. 3436, Springer 2005
- [111] **J. Mortimer**, "Krause plans low-volume assembly hall for Ricardo," in *Assembly Automation Journal*, vol. 21, no.4, pp. 333-335, 2001

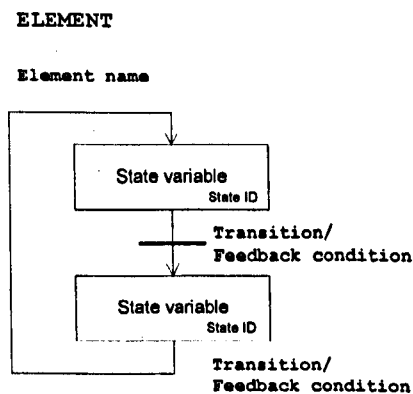
- [112] **R. Harrison, A.W. Colombo, A.A. West and S.M. Lee**, "Reconfigurable modular automation systems for automotive power-train manufacture," in *Journal of Flexible Manufacturing System*, Springer, vol.18, pp.175-190, 2006
- [113] **D. Ceglarek, et al.**, "Time-Based Competition in Multistage Manufacturing: Stream-of-Variation Analysis (SOVA) Methodology—Review," in *International Journal of Flexible Manufacturing Systems*, vol.16, pp. 11–44, 2004
- [114] **M. Ryll, and S. Ratchev**, "Application of the Data Distribution Service for Flexible Manufacturing Automation," in *Proceedings of World Academy of Science, Engineering and Technology*, vol. 31, July 2008
- [115] **D. McFarlane**, "Modular Distributed Manufacturing System and the Implication for Integrated Control," in *IEE Colloquium on Choosing the Right Control Structure*, London, UK, 1998
- [116] **D. A. Vera**, "Innovative Approach to the Design and Realisation of a Virtual Prototyping Environment for Manufacturing System Engineering," *PhD dissertation*, Mechanical Engineering, Loughborough University, December, 2004
- [117] **C. Abadie and R. Neubert**, "The mechatronic automation framework and its architecture requirements," *RIMACS Deliverable D 2.1 Internal Document*, August 2006
- [118] **R.H. Weston**, "Model-driven, component-based approach to reconfiguring manufacturing software systems," in *International Journal of Operations & Production Management*, vol. 19, no. 8, pp. 834- 855, 1999
- [119] **Y.Y. Yusuf, M. Sarhadi, and A.Gunasekaran**, "Agile manufacturing: The drivers, concepts and attributes," in *International Journal of Production Economic*, vol. 62, pp. 33-43, 1999
- [120] **G. Xiong, A. Litokorpi, and T. R. Nyberg**, "Middleware-based Solution for Enterprise Information Integration," in *8th IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 2, pp. 687-690, 2001
- [121] **M. Piyush, T. N. KIEN, and E. Abdelkarim**, "QoS-Based Message-Oriented Middleware for Web Services," in *Journal of Web Information Systems – WISE 2004 Workshops*, Publisher: Springer Berlin / Heidelberg, vol. 3307, October 2004
- [122] **L. Kutvonen, T. Ruokolainen, and J. Metso**, "Interoperability Middleware for Federated Business Services in Web-Pilarcos," in *International Journal of Enterprise Information Systems*, vol. 3, no.1, pp. 1-21, 2007
- [123] **R.H. Weston, R. Harrison, A.H. Booth, and P.R. Moore**, "Universal machine control system primitives for modular distributed manipulator systems," in *International Journal of Production Research*, vol. 27, no. 3, pp. 395-410, 1989

APPENDIX A

FESTO- Test Rig State Transition Diagram

The following document shows the finite state machine behaviour of the FORD-FESTO machine components implemented on the PLC and the Web services- based control system. The diagram illustrates element states and transition conditions of the components within the machine system. The state transitions are formulated by inputs and outputs of field devices (sensors/actuators) as well as element timer. The style of a diagram is presented in A.1 figure. Element names, state variables, state ID and transitions are followed the standard defined by the MSI research team.

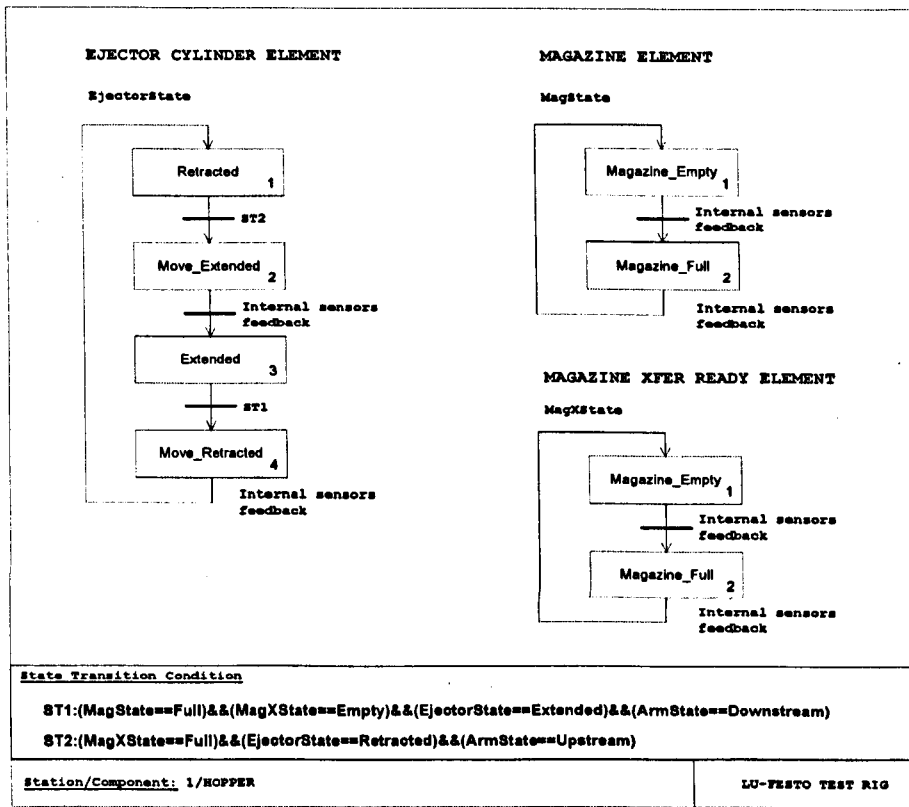
A.1: State Transition Representation Diagram



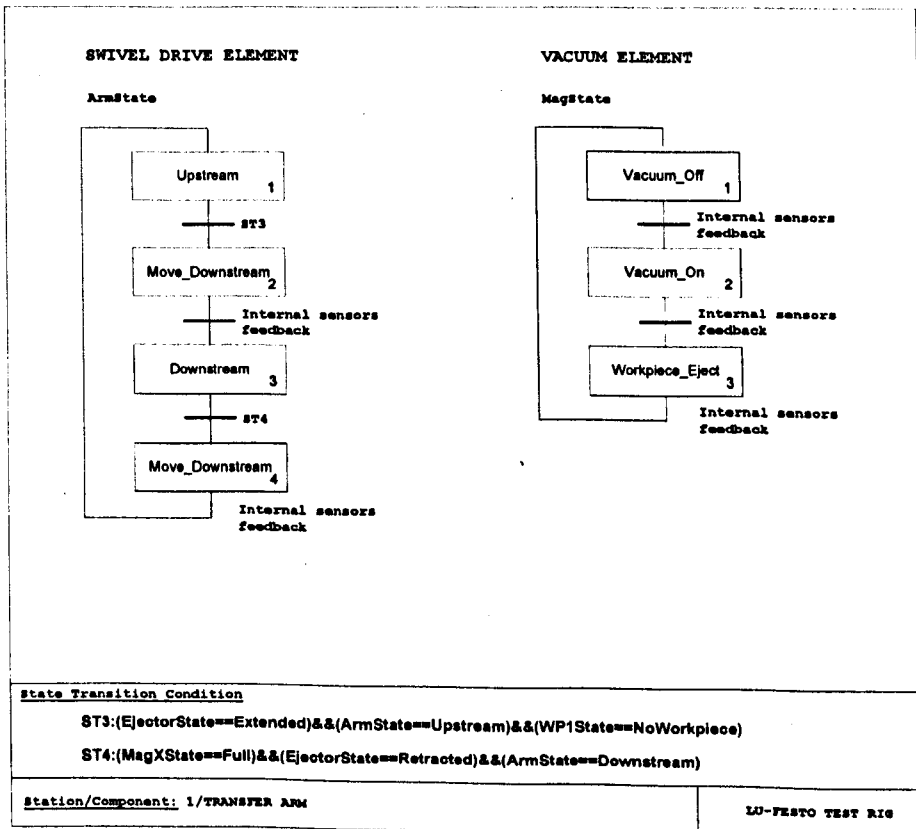
The element state is defined by the co-operated sensors attached to elements; the state is expressed according to the simple combination of sensor logics (1 or 0) defined in the system. However, a more complex component contained 3 elements or more can be simplified to a single element by the state combination as shown in A.8. The three elements have been grouped together to form the combination of the 12-state element. This approach makes the state transition condition to be simpler and more manageable.

In associate with following state transition diagrams, some elements may contain dynamic states (e.g. Move_Extended) as well as static states (e.g. Extended). The state transition (ST) is a specific condition for the state change only from the static state to either the dynamic or static state. The transition condition will never originate from the dynamic state. The movement expressed by the dynamic state causes the coupled sensors to automatically change their states, thus producing the feedback signal along the transition.

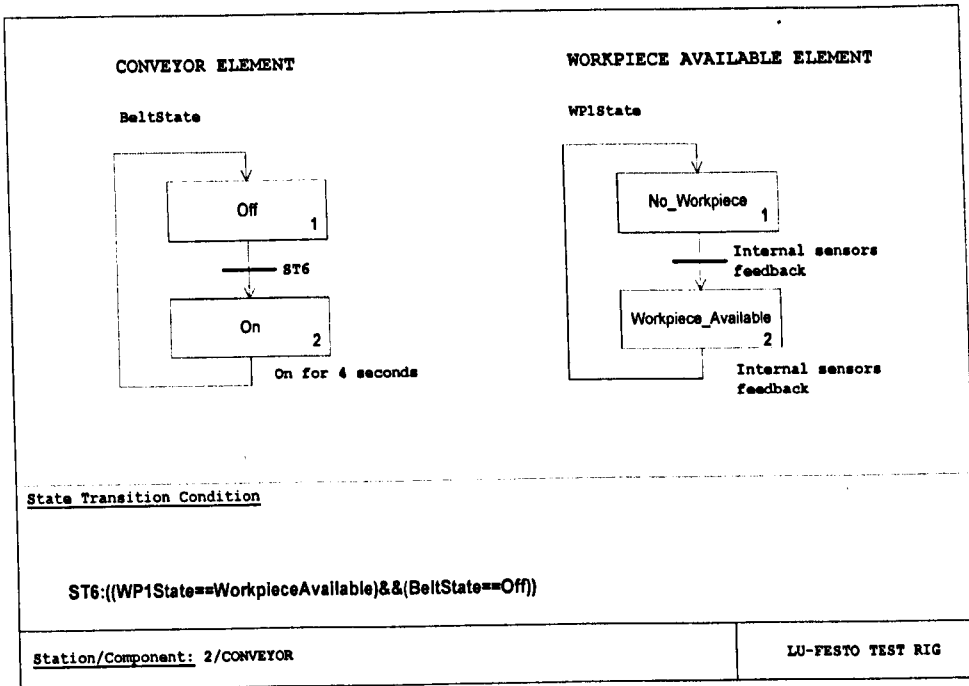
A.2: Hopper Component State Transition Diagrams



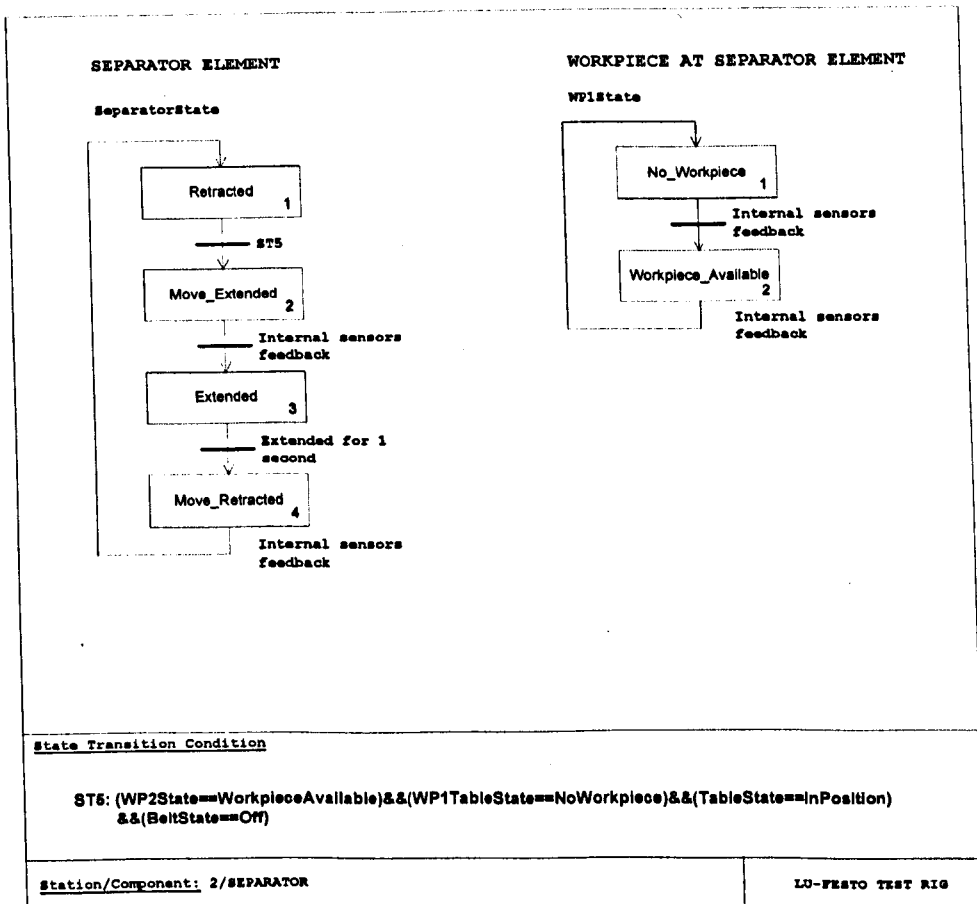
A.3: Swivel Drive Component State Transition Diagrams



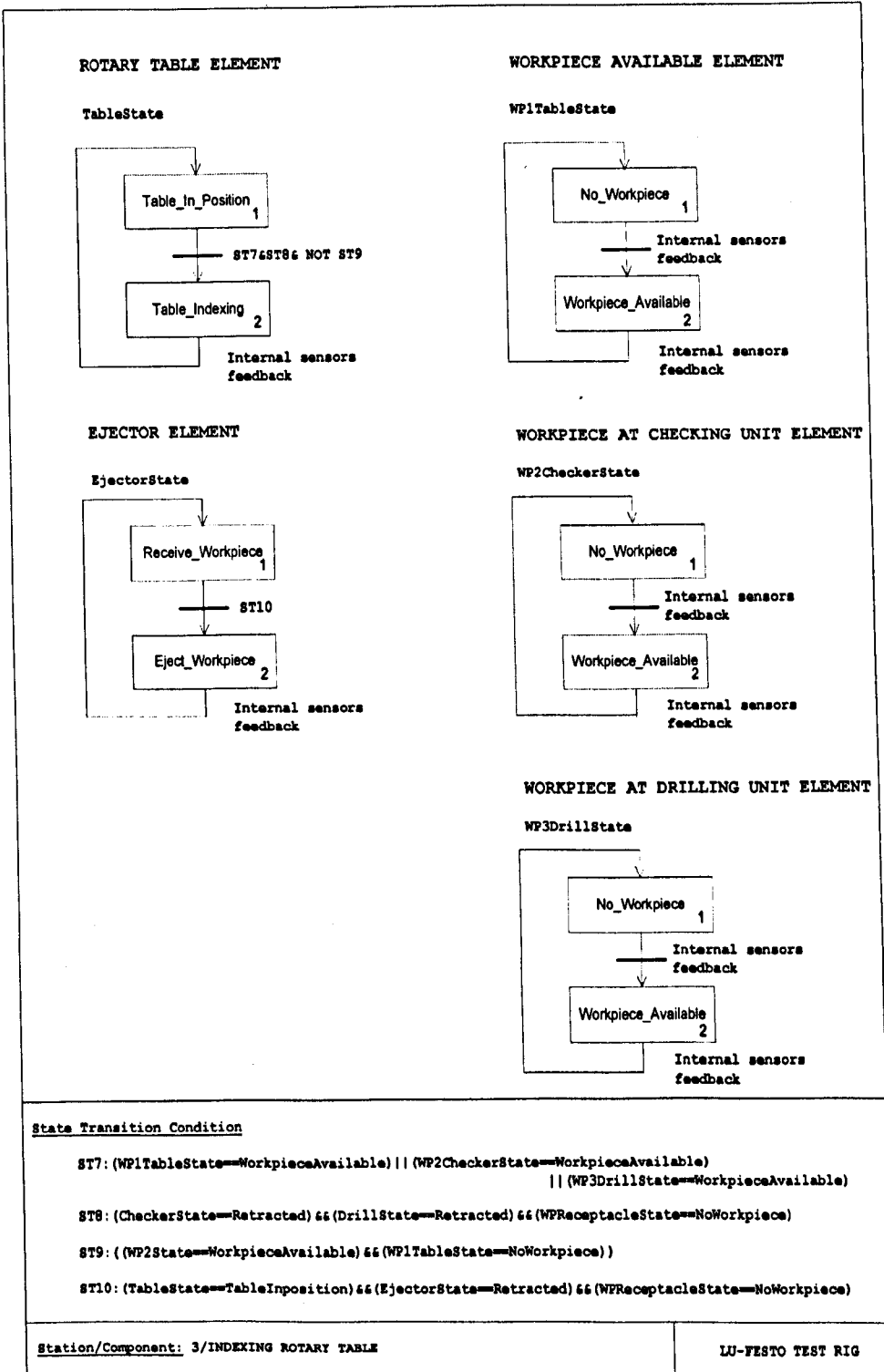
A.4: Conveyor Component State Transition Diagrams



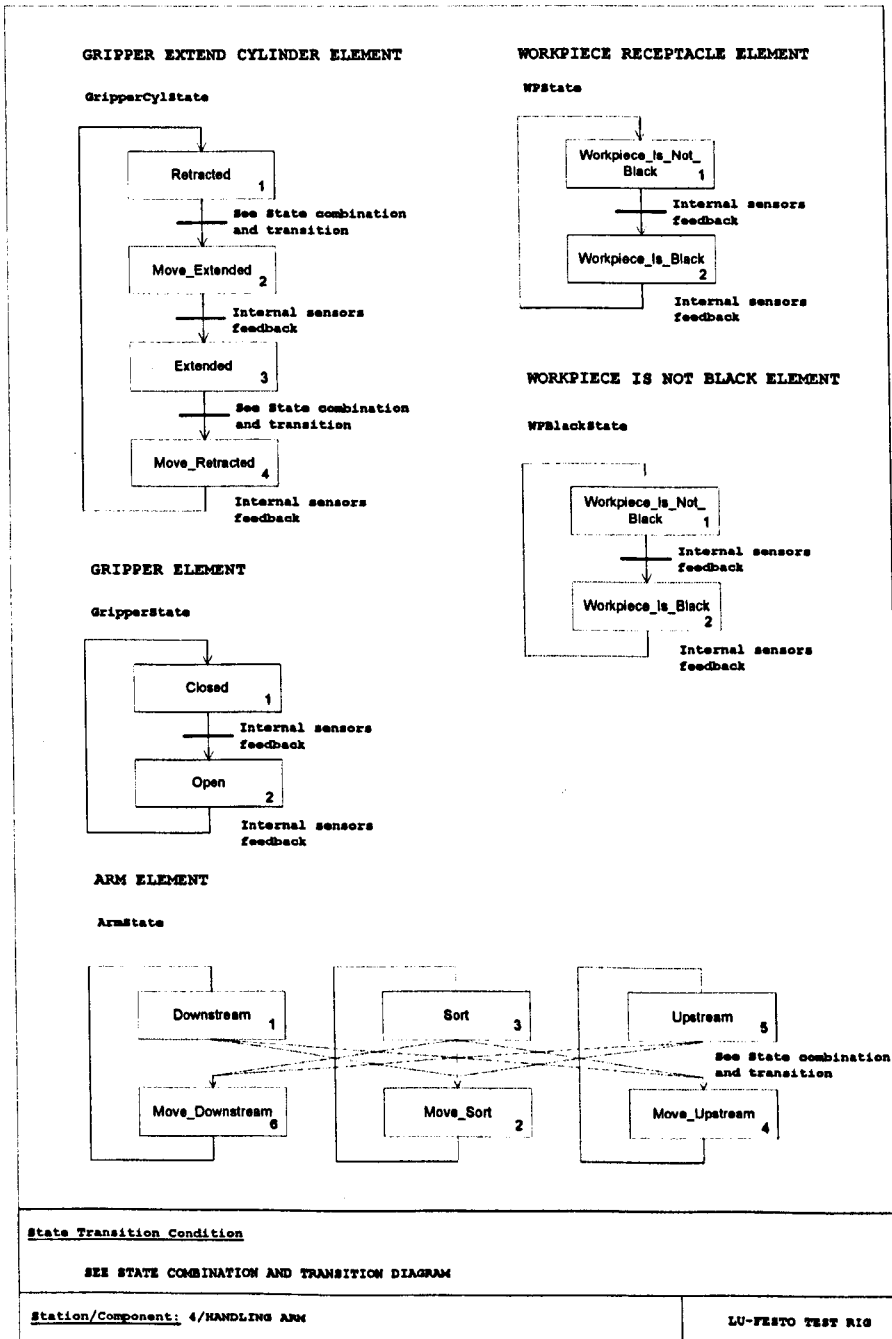
A.5: Separator Component State Transition Diagrams



A.6: Indexing Table Component State Transition Diagrams



A.7: Handling Arm Component State Transition Diagrams



A.8: Handling Arm Component State Combination

```

HANDLING ARM COMPONENT STATE COMBINATION (static state)

[ARM (3 states) x GRIPPER (2 states) x GRIPPER CYLINDER (2 states) = 12 states(static)]

/*-----Defining Handling Arm States- 12 States + 1 Error States-----*/
//STATE 1
    if ((ArmState==Upstream) && (GripperCylState==Retracted) && (GripperState==Open))
    {
        Handling=1;
    }
//STATE 2
    else if ((ArmState==Upstream) && (GripperCylState==Extended) && (GripperState==Open))
    {
        Handling=2;
    }
//STATE 3
    else if ((ArmState==Upstream) && (GripperCylState==Extended) && (GripperState==Closed))
    {
        Handling=3;
    }
//STATE 4
    else if ((ArmState==Upstream) && (GripperCylState==Retracted) && (GripperState==Closed))
    {
        Handling=4;
    }
//STATE 5
    else if ((ArmState==Downstream) && (GripperCylState==Retracted) && (GripperState==Closed))
    {
        Handling=5;
    }
//STATE 6
    else if ((ArmState==Downstream) && (GripperCylState==Extended) && (GripperState==Closed))
    {
        Handling=6;
    }
//STATE 7
    else if ((ArmState==Downstream) && (GripperCylState==Extended) && (GripperState==Open))
    {
        Handling=7;
    }
//STATE 8
    else if ((ArmState==Downstream) && (GripperCylState==Retracted) && (GripperState==Open))
    {
        Handling=8;
    }
//STATE 9
    else if ((ArmState==Sort) && (GripperCylState==Retracted) && (GripperState==Closed))
    {
        Handling=9;
    }
//STATE 10
    else if ((ArmState==Sort) && (GripperCylState==Extended) && (GripperState==Closed))
    {
        Handling=10;
    }
//STATE 11
    else if ((ArmState==Sort) && (GripperCylState==Extended) && (GripperState==Open))
    {
        Handling=11;
    }
//STATE 12
    else if ((ArmState==Sort) && (GripperCylState==Retracted) && (GripperState==Open))
    {
        Handling=12;
    }
//ERROR STATE 0
    else
        Handling=0;

```

State Transition Condition

STATE TRANSITION DIAGRAM

Station/Component: 4/HANDLING ARM

LU-FESTO TEST RIG

A.9: Handling Arm Component State Combination Transition Condition

HANDLING ARM COMPONENT STATE TRANSITION

```

/*-----STATE TRANSITION CONDITION-----*/
if(Handling==8)
  { //Goto 1(Initial Position)-Move UPstream
  }
else if(Handling==12)
  { //Goto 1(Initial Position)-Move UPstream
  }
else if((Handling==1)&&(WPstate==NoWorkpiece))
  { //Standby for WP
  }
else if((Handling==1)&&(WPstate==WorkpieceAvailable))
  { //Goto 2-Extended
  }
else if((Handling==2)&&(WPstate==WorkpieceAvailable))
  { //Goto 3-Close Gripper
  }
else if(Handling==3)
  { //Goto 4-Retracted
  }
else if(Handling==4)
  {
    if(WPBlackState==Black)
      { //Goto 9-Move Sort if Backe WP
      }
    else if(WPBlackState==NotBlack)
      { //Goto 5-Move Downstream if Back WP
      }
    else
      {
        //Unidentified Workpiece!! ERROR at STATE 4
      }
  }
}
/*WP Not Black Cycle state 5 to 7*/
else if(Handling==5)
  { //Goto 6-Extended
  }
else if(Handling==6)
  { //Goto 7-Open Gripper
    GripperState = ClientTask(1,7);
  }
else if(Handling==7)
  { //Goto 8-Retracted
  }
}
/*WP Black Cycle state 9 to 11*/
else if(Handling==9)
  { //Goto 10-Extended
  }
else if(Handling==10)
  { //Goto 11-Gripper Open
  }
else if(Handling==11)
  { //Goto 12-Retracted
  }
}
else
  { //Error Recover //Handling = 0 (Normally at sort)
    //Start error recover procedure:Restart to STATE-1(by Extended-Open-Retracted-
    MoveUpstream)
    //*****Recover done*****//
  }
}

```

State Transition Condition

STATE TRANSITION DIAGRAM

Station/Component: 4/HANDLING ARM

LU-FESTO TEST RIG

APPENDIX B

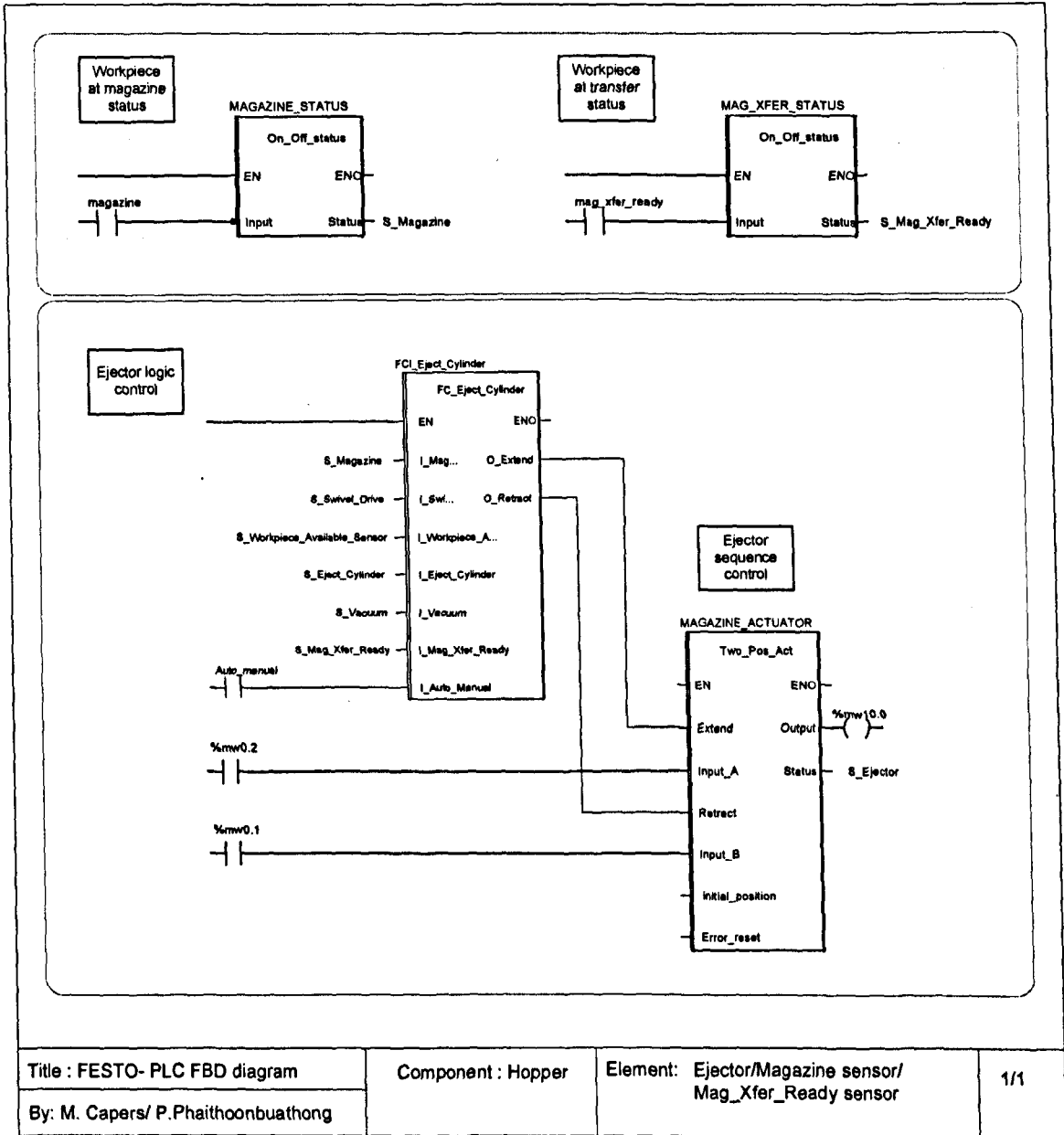
Test Rig PLC Function Block Control Application

The control programme for the rig operation is built using the function block diagram (FBD) contains I/O operations, state transition conditions, network variables, and communication and HMI interface of components. The control function in FBDs is implemented with the structure text and sub function blocks which are enclosed inside the main function block. The machine application is conceived by forming the interaction among components in corresponding to the defined state transition condition of components (*Appendix A*). In the PLC- based control system implemented in the research, the machine application is commissioned via the PLC ladder logic sequence and the function block of control elements for the machine operation in both automatic and manual mode and the operator console.

The following function blocks contain the input and output state variables of sensors and actuators which are assigned to specific I/O elements via the PLC system memory allocation (e.g. %mw). These state variables are used by the state transition condition of the element, which is encapsulated inside the function block, to perform the I/O operation task upon the current I/O state. In this implementation, the function block has been built and saved into the PLC database which can be accessed for reuse by other new components, provided the new state variables need to be modified.

In the PLC programming applications, the diagrams have been generally organised into 3 units labelled by Status, Logic Control and Sequence Control. Status FBD defines the state of elements by mean of converting I/O memory location (%mw), which connected to the local sensors, to a global state variable used by the HMI console and the PLC application. Logic Control FBD defines the control logic (finite state transition condition) of components in associated with contained elements (actuator and sensor states) and interlocking components. Sequence Control FBD is used to control component timing, sequences and actuating the PLC output channels after the component transition in Logic Control FBD is met.

B.1: Hopper Component PLC Programming Application



Title : FESTO- PLC FBD diagram

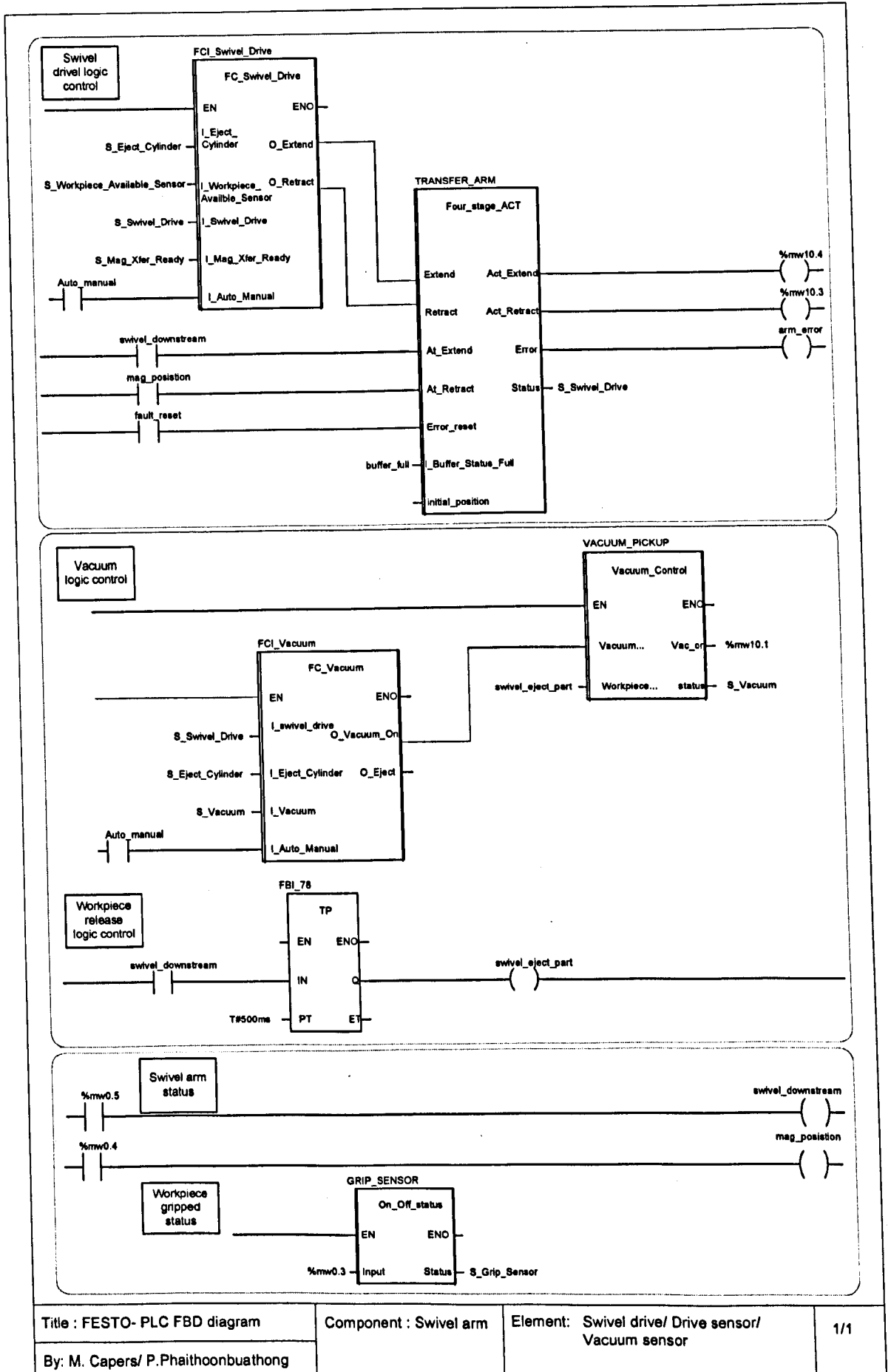
Component : Hopper

Element: Ejector/Magazine sensor/
Mag_Xfer_Ready sensor

1/1

By: M. Capers/ P.Phaithoonbuathong

B.2: Swivel Drive Component PLC Programming Application



Title : FESTO- PLC FBD diagram

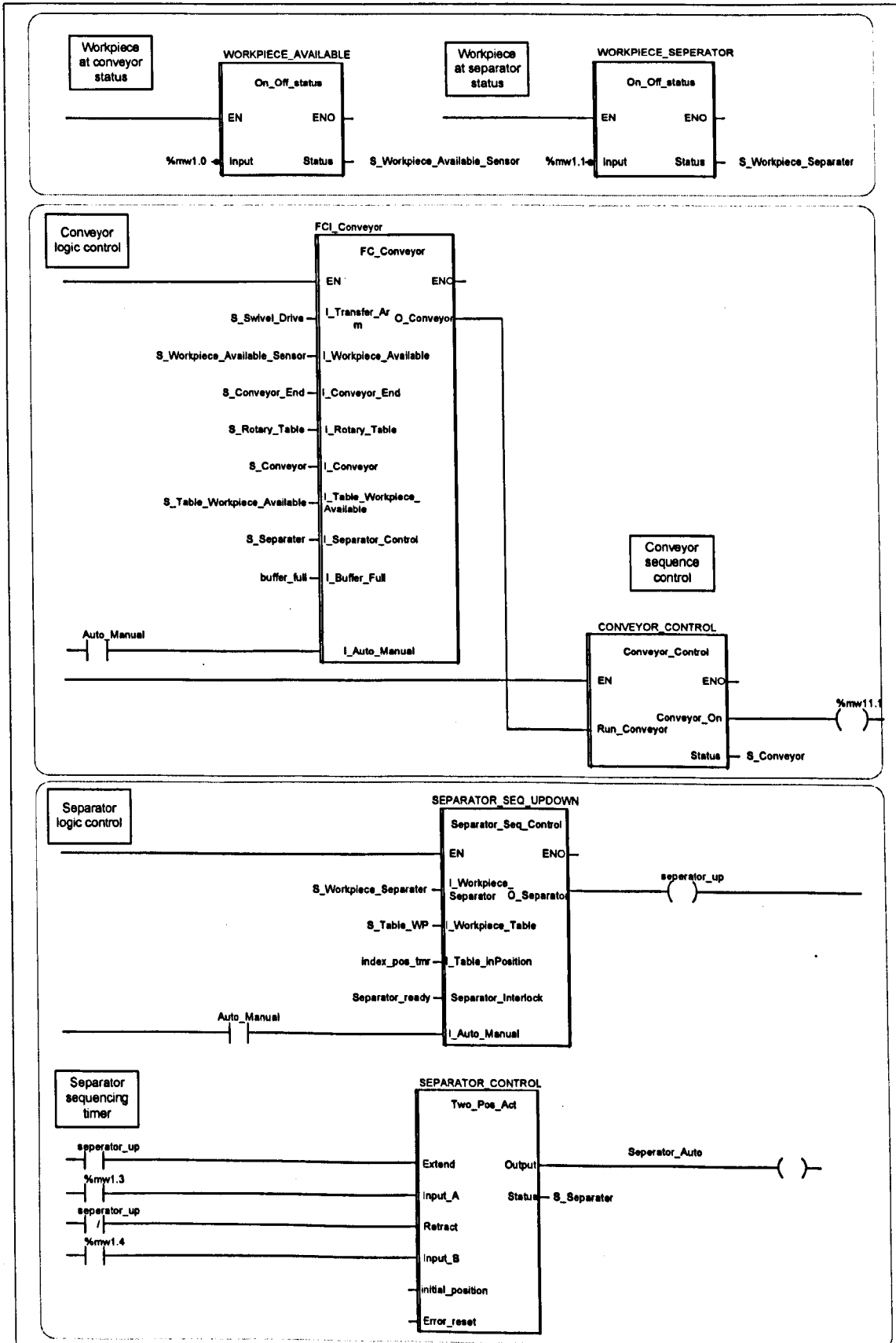
Component : Swivel arm

Element: Swivel drive/ Drive sensor/
Vacuum sensor

1/1

By: M. Capers/ P.Phaitoonbuathong

B.3: Buffer Conveyor Component PLC Programming Application



Title : FESTO- PLC FBD diagram

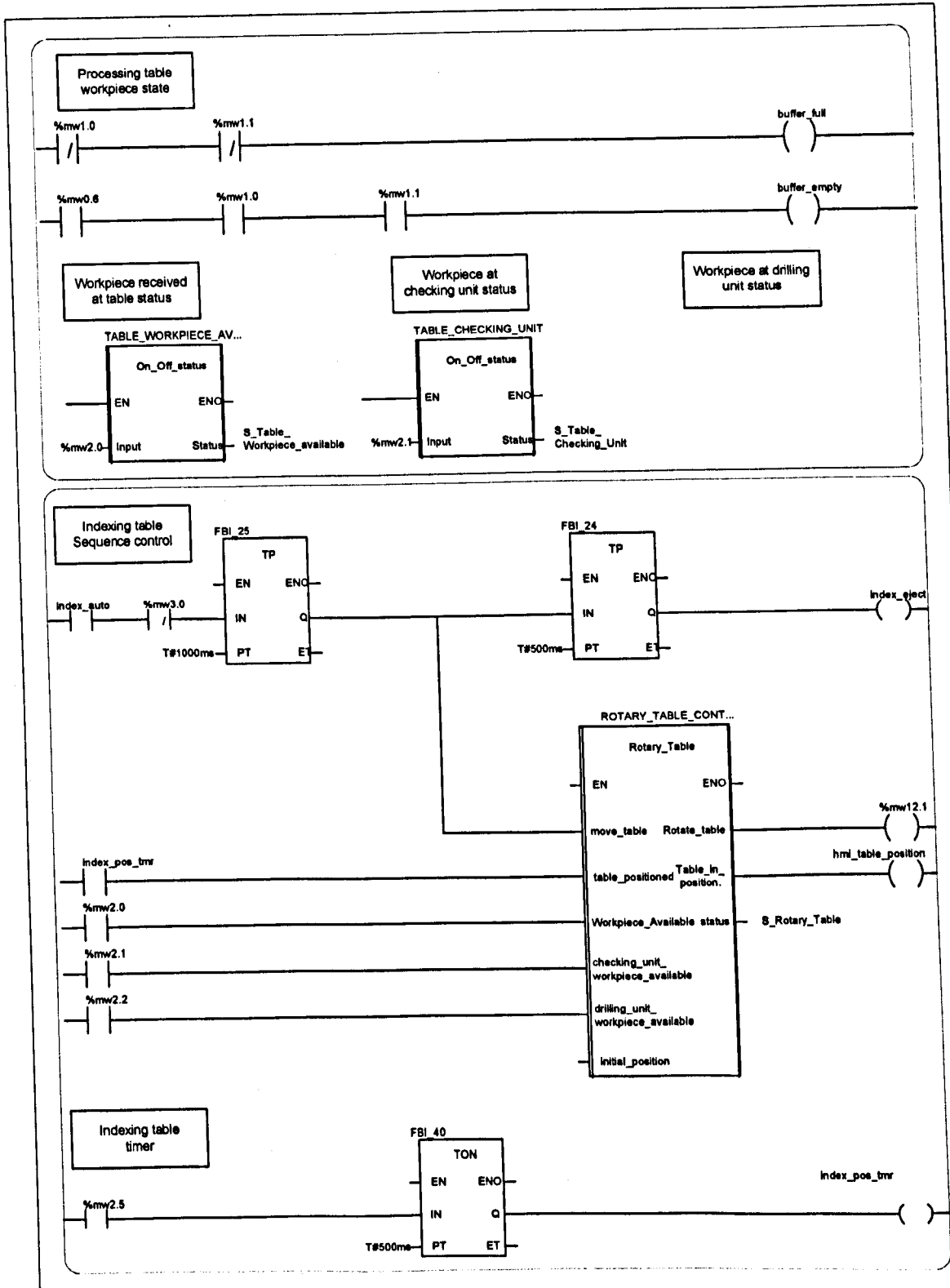
Component : Buffer conveyor

Element: Conveyor/ Separator
Workpiece at conveyor sensor/
Workpiece at separator sensor

1/1

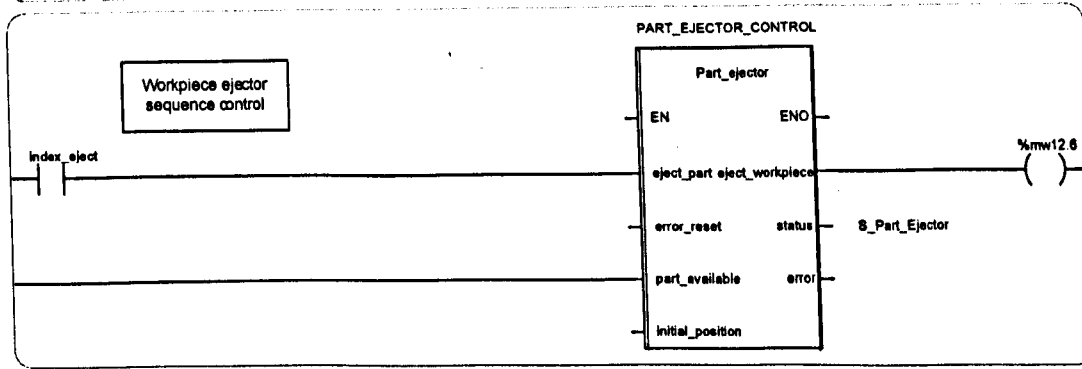
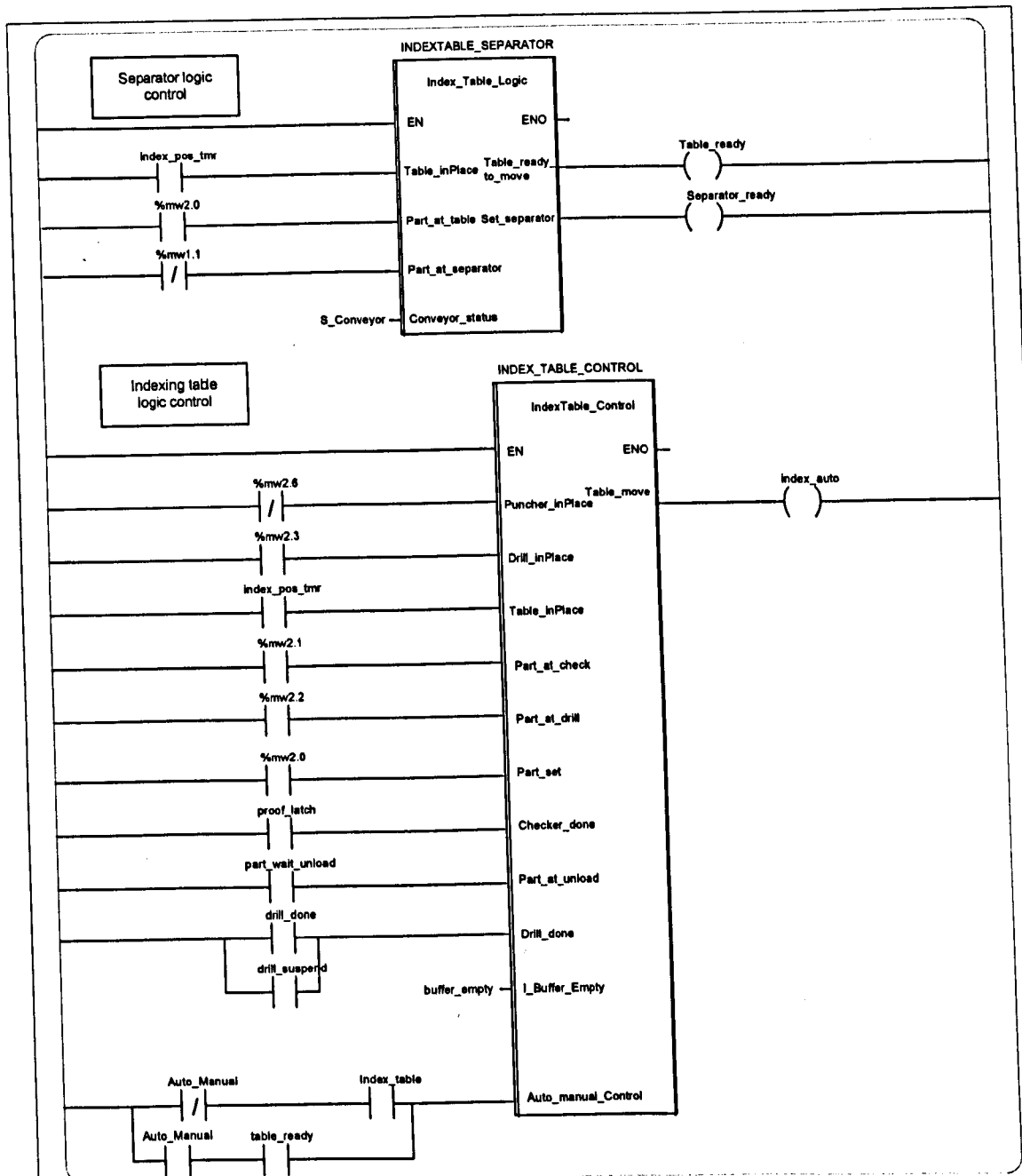
By: M. Capers/ P.Phaithoonbuathong

B.4-1: Processing Table Component PLC Programming Application



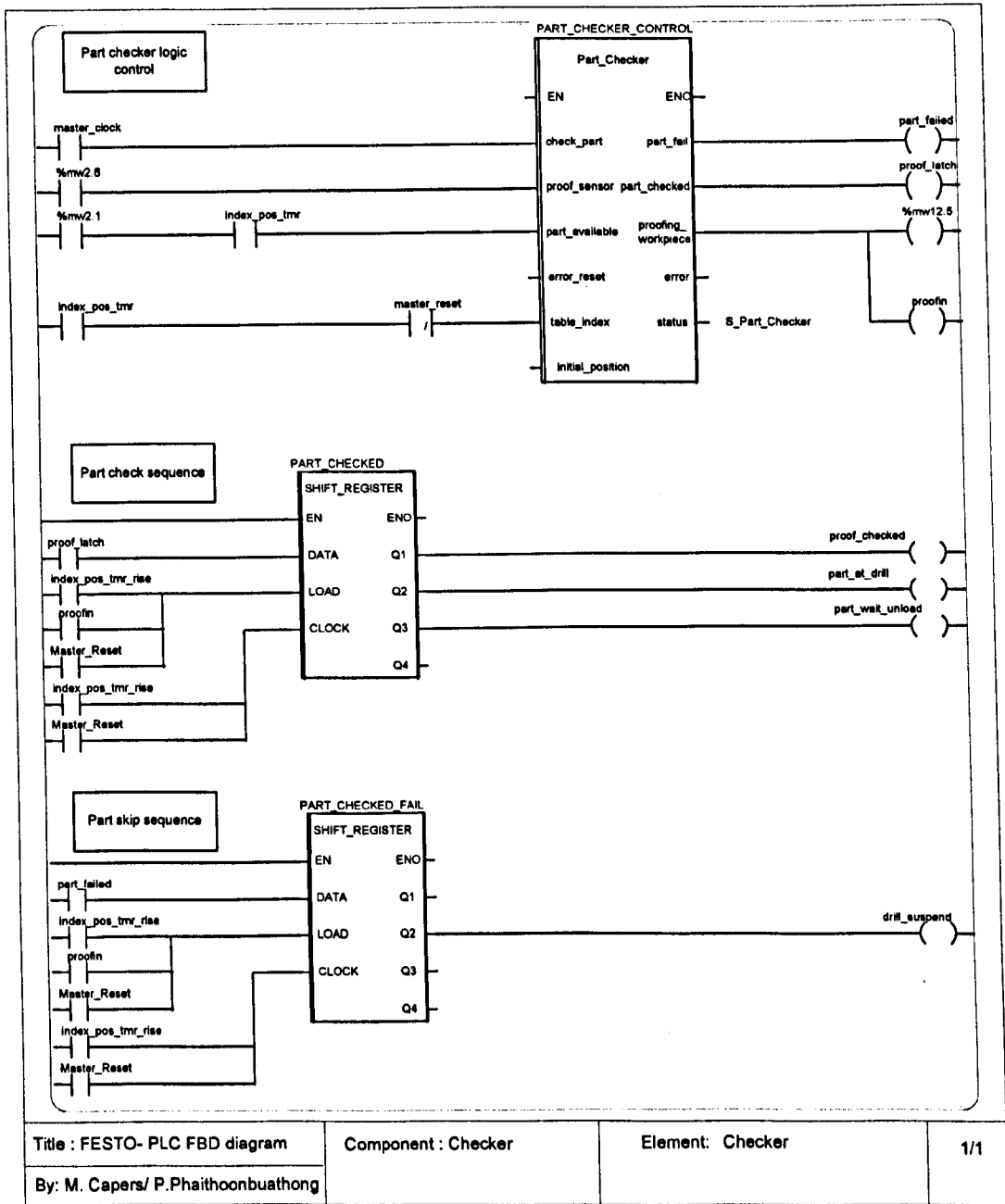
Title : FESTO- PLC FBD diagram	Component : Processing table	Element: Indexing table/Ejector Workpiece received at table sensor/ Workpiece at checking unit sensor/ Workpiece at drill unit sensor	1/2
By: M. Capers/ P.Phaithoonbuathong			

B.4-2: Processing Table Component PLC Programming Application

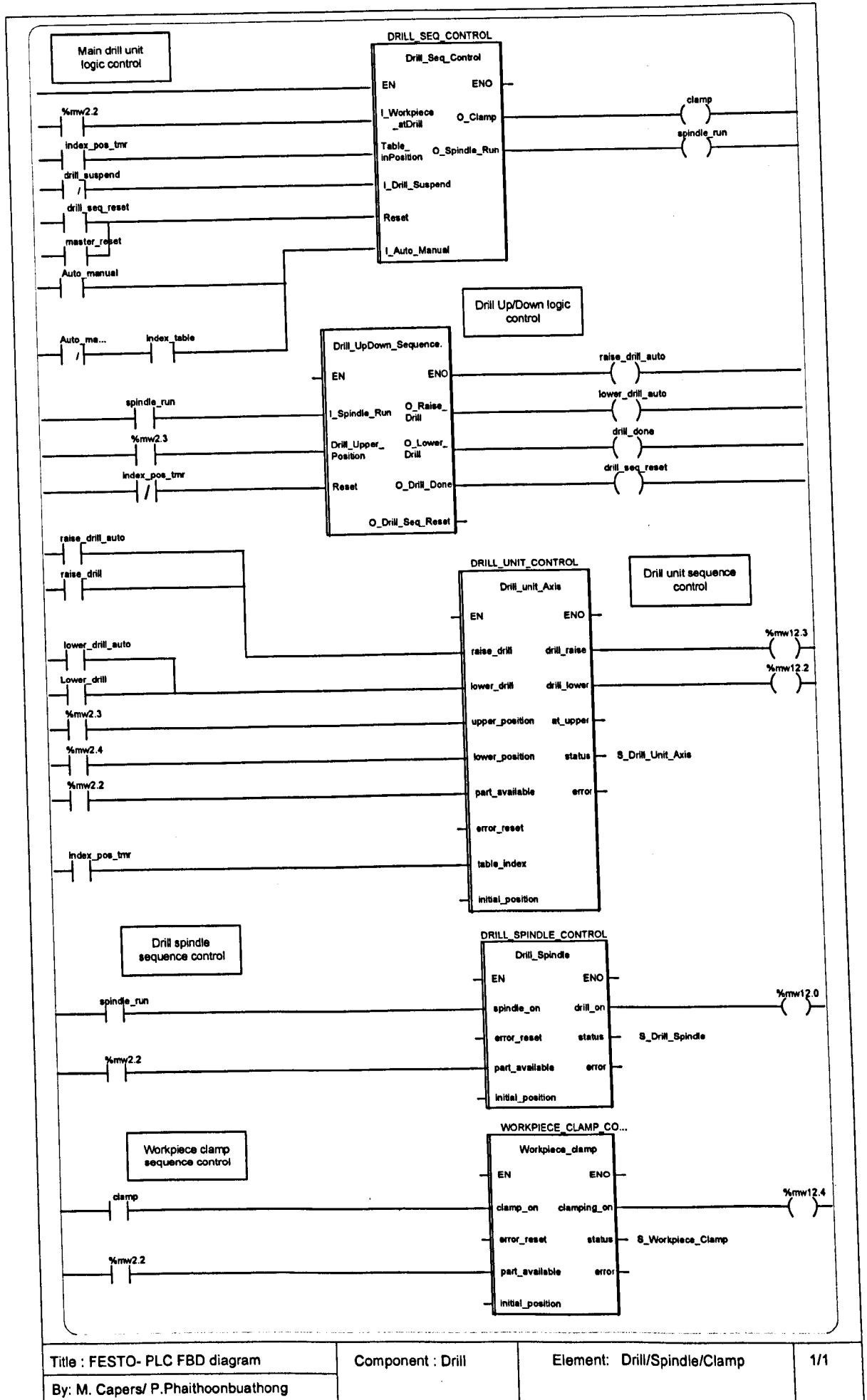


Title : FESTO- PLC FBD diagram	Component : Processing table	Element: Indexing table/Ejector Workpiece received at table sensor/ Workpiece at checking unit sensor/ Workpiece at drill unit sensor	2/2
By: M. Capers/ P.Phaithoonbuathong			

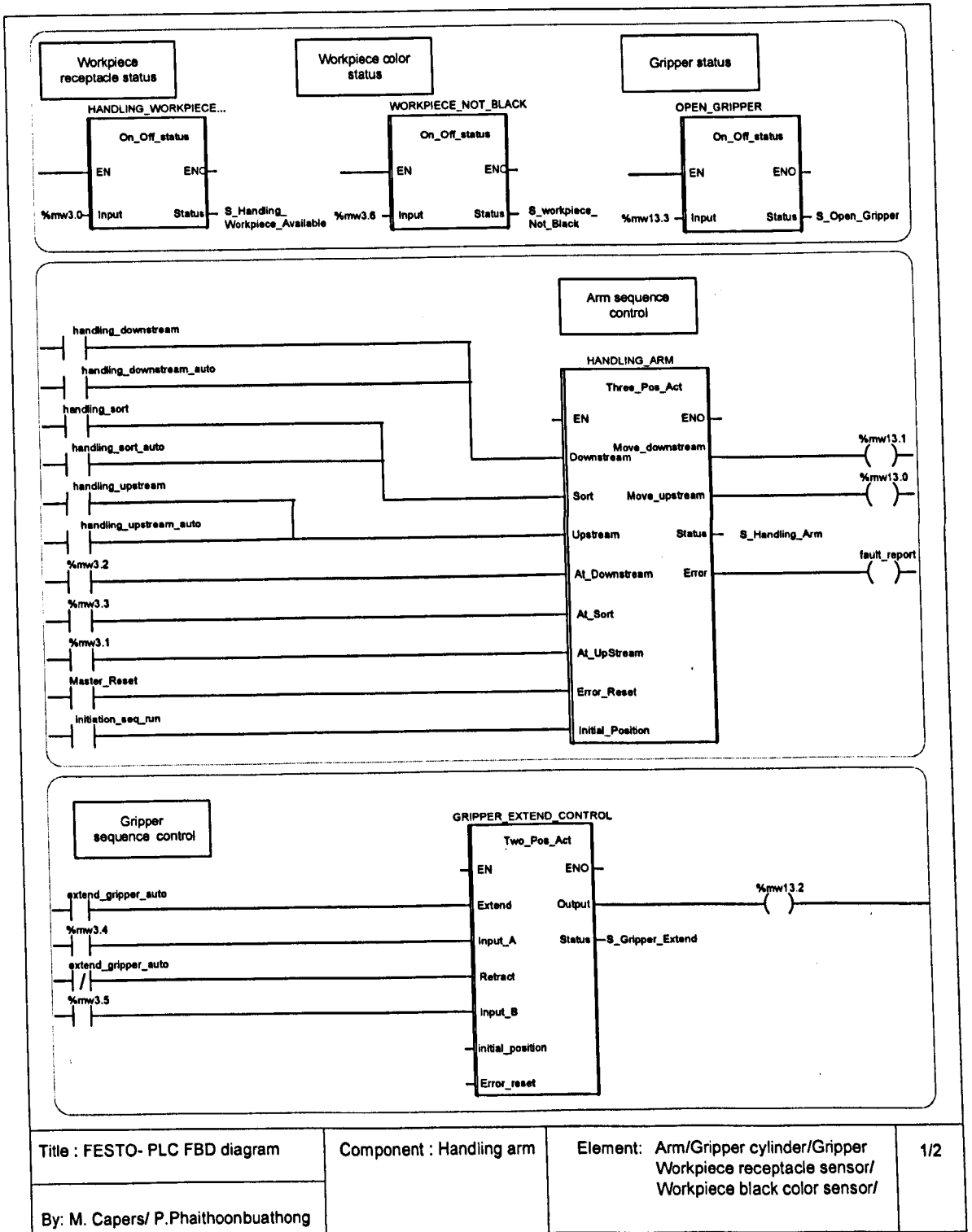
B.5: Checker Component PLC Programming Application



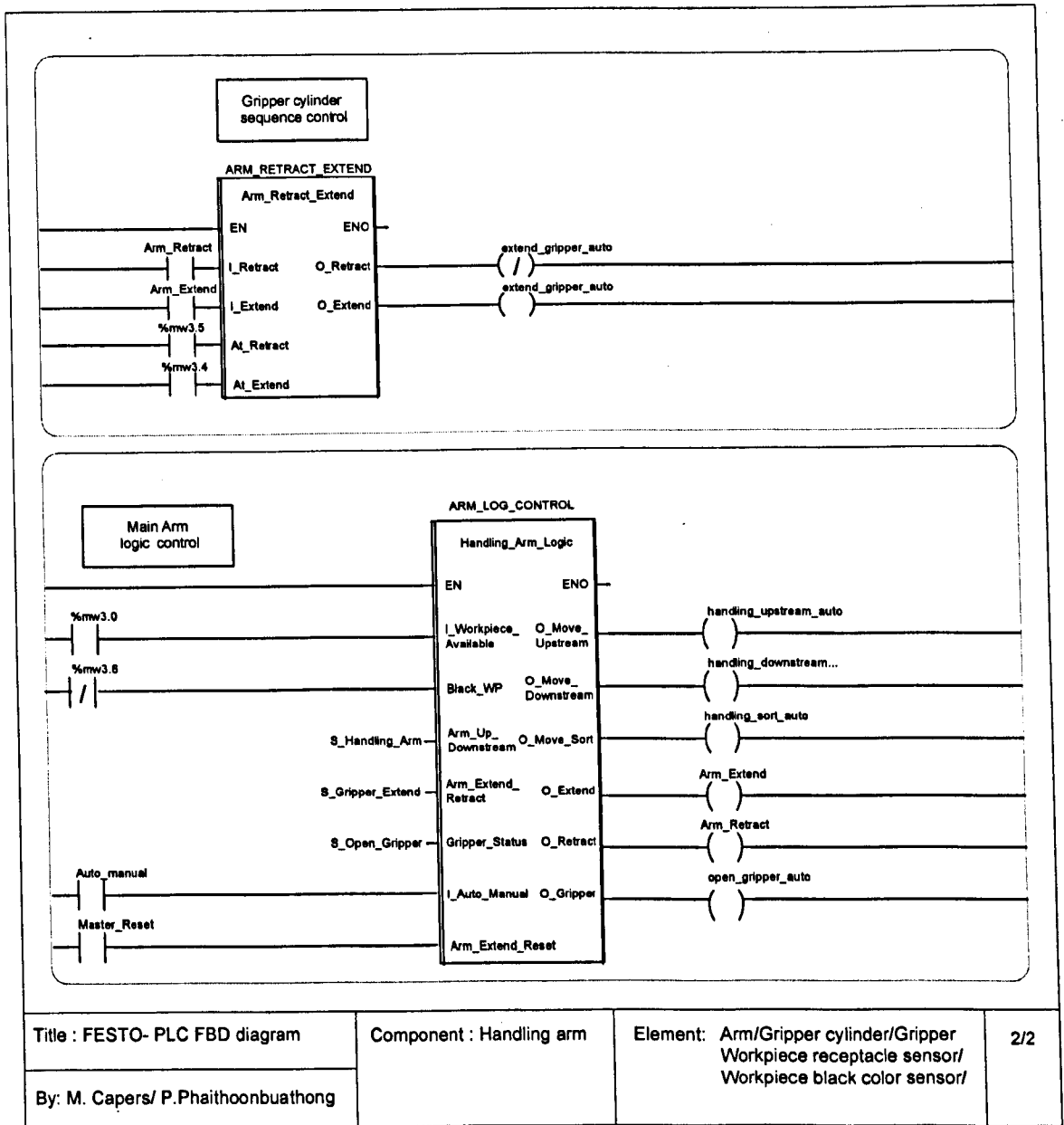
B.6: Drill Component PLC Programming Application



B.7-1: Handling Arm Component PLC Programming Application



B.7-2: Handling Arm Component PLC Programming Application



Title : FESTO- PLC FBD diagram	Component : Handling arm	Element: Arm/Gripper cylinder/Gripper Workpiece receptacle sensor/ Workpiece black color sensor/	2/2
By: M. Capers/ P.Phaithoonbuathong			

APPENDIX C

SOAP – XML Message Contents

The SOAP text-based message for Web Services state notifications and operations on the implemented test rig is captured by the TCP/IP protocol analyser as demonstrated in Chapter 9. In the implementation, four of the distributed embedded control devices (FTB's) have been assigned with the static IP address (150.1.0.101 to 150.1.0.104), MAC address (00-40-AF-00-00-31 to 00-40-AF-00-00-34) and Port number (9881 to 9884). A PC (running Service orchestration engine application to execute the services on the control devices) has the static IP address: 150.1.0.201 and MAC address: 00-1C-42-93-5F-E4. These values are referenced in the SOAP message to form the unique message identification and address for sending and receiving SOAP messages by the DPWS client and server application (*Chapter 9- section 9.2.1*). The captured SOAP messages from the test rig operation (*as outlined in Chapter 8- section 8.5.1 and 8.5.2*) are shown as follows:

C.1 Multicast Probe Discovery Message (from PC to all FTB devices) - 650 bytes

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery">

  <SOAP-ENV:Header>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe</wsa:Action>
    <wsa:MessageID>urn:uuid:aaad075a-3ebf-11dd-96d0-001c42935fe4</wsa:MessageID>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <wsd:Probe>
      <wsd:Types xmlns:_0="http://www.soda-itea2.org/Demonstrator/Hopper">_0:Ejector</wsd:Types>
    </wsd:Probe>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The client sends the probe message (attached with the corresponding location in MessageID) over multicast protocol to look up the ejector element (_0: Ejector) on the network.

C.2 Probe Match Reply Message (from matched FTB device to PC) - 1203 bytes

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:ns0="http://www.soda-itea2.org/Demonstrator/Hopper"
  xmlns:wdp="http://schemas.xmlsoap.org/ws/2006/02/devprof">

  <SOAP-ENV:Header>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches</wsa:Action>
    <wsa:MessageID>urn:uuid:13814007-1dd2-11b2-bc3d-0040af000031</wsa:MessageID>
    <wsa:RelatesTo>urn:uuid:aaad075a-3ebf-11dd-96d0-001c42935fe4</wsa:RelatesTo>
    <wsd:AppSequence MessageNumber="4" InstanceId="0"/>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <wsd:ProbeMatches>
      <wsd:ProbeMatch>
        <wsa:EndpointReference>
          <wsa:Address>urn:uuid:00002699-0000-1000-8000-0040af000031</wsa:Address>
        </wsa:EndpointReference>

        <wsd:Types>ns0:Ejector wdp:Device</wsd:Types>
        <wsd:Scopes>Hopper</wsd:Scopes>
        <wsd:XAddr>http://150.1.0.101:9881/00002699-0000-1000-8000-0040af000031</wsd:XAddr>
        <wsd:MetadataVersion>1</wsd:MetadataVersion>
      </wsd:ProbeMatch>
    </wsd:ProbeMatches>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The matched element replies the client with the probe match message contained information on element type, component scope, and address (endpoint reference) which are used by DPWS to retrieve component metadata and invoke services.

C.3 Subscribe Message (from PC to the FTB device) – 1261 bytes

```

POST /13814001-1dd2-11b2-bc3d-0040af000031
HTTP/1.1..Host:150.1.0.101:9881..User-Agent: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 1013..Connection: close....

<?xml version="1.0" encoding="UTF-8"?>
< SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">

  <SOAP-ENV:Header>
    <wsa:To>http:// 150.1.0.101:9881 /13814001-1dd2-11b2-bc3d-0040af0 00031</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/08/ eventing/Subscribe</wsa:Action>
    <wsa:MessageID>urn:uuid:5e14087e- e24f-11dd-8ff5-001c42935fe4</wsa :MessageID>
    <wsa: ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ ws/2004/08/addressing/role/anyon ymous</wsa:Address>
    </wsa:ReplyTo>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body >
    <wse:Subscribe>
      <wse:Delivery Mode="http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push">
        <wse:NotifyTo>
          <wsa:Address>http://150.1.0.201:8871/5b2d6358-e24f-11dd-8ff 5-001c42935fe4</wsa:Address>
        </wse:NotifyTo>
      </wse:Delivery>
      <wse:Expires>PT1H</wse:Expires>
    </wse:Subscribe>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>

```

After the look up process, the client can subscribe (for 1 hour- PT1H parameter) to the specific element for state changed notification which is sent to the specific address given in the <wsa:Address> tag.

C.4 State Changed Notification Message (from the FTB device to PC) – 794 bytes

```
POST /158cb99e-693b-11dd-8abd-001c42935fe4
HTTP/1.1..Host:150.1.0.201:8873..User-Agent: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 546..Connection: close....

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soapenvelope"
                    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                    xmlns:ejt="http://www.soda-itea2.org/Demonstrator/Ejector">

  <SOAP-ENV:Header>
    <wsa:To>http://150.1.0.201:8873/158cb99e-693b-11dd-8abd-001c42935fe4</wsa:To>
    <wsa:Action>http://www.soda-itea2.org/Demonstrator/Ejector/ejector/ejtNotifyEvent</wsa:Action>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <ejt:NotifyElement>
      <ejt:NotifyStatus>EXTENDED</ejt:NotifyStatus>
    </ejt:NotifyElement>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

The changed state (EXTENDED) is sent to the subscriber on the service orchestration engine addressed at SOAP envelope header <wsa:To> section.

C.5 Service Invocation Message (from PC to the FTB device) – 1020 bytes

```
POST /13814002-1dd2-11b2-bc3d-0040af000032
HTTP/1.1..Host:150.1.0.102:9882..User-Agent: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 773..Connection: close....

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
                    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                    xmlns:blt="http://www.soda-itea2.org/Demonstrator/Belt">

  <SOAP-ENV:Header>
    <wsa:To>http://150.1.0.102:9882/13814002-1dd2-11b2-bc3d-0040af000032</wsa:To>
    <wsa:Action>http://www.soda-itea2.org/Demonstrator/Belt/belt/separatorCmdRequest</wsa:Action>
    <wsa:MessageID>urn:uuid:41c616ba-3e14-11dd-8430-001c42935fe4</wsa:MessageID>
    <wsa:ReplyTo><wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address></wsa:ReplyTo>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <blt:separatorCmd>
      <blt:SeparatorAction>MoveExtended</blt:SeparatorAction>
    </blt:separatorCmd>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

The service orchestration engine invokes the (extend) service on the separator element by sending the SOAP message with a command attribute- MoveExtended to separator endpoint reference addressed at <wsa:To> tag.

C.6 Service Acknowledge Message (from the FTB device to PC) – 808 bytes

```

HTTP/1.1 200 OK..Server: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 625..Connection: close...
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:blt="http://www.soda-itea2.org/Demonstrator/Belt">
  <SOAP-ENV:Header>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
    <wsa:Action>http://www.soda-itea2.org/Demonstrator/Belt/belt/separatorCmdResponse</wsa:Action>
    <wsa:RelatesTo>um:uuid:41c816ba-3e14-11dd-8430-001c42935fe4</wsa:RelatesTo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <blt:separatorCmdResponse>
      <blt:Separator>MoveExtended</blt:Separator>
    </blt:separatorCmdResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

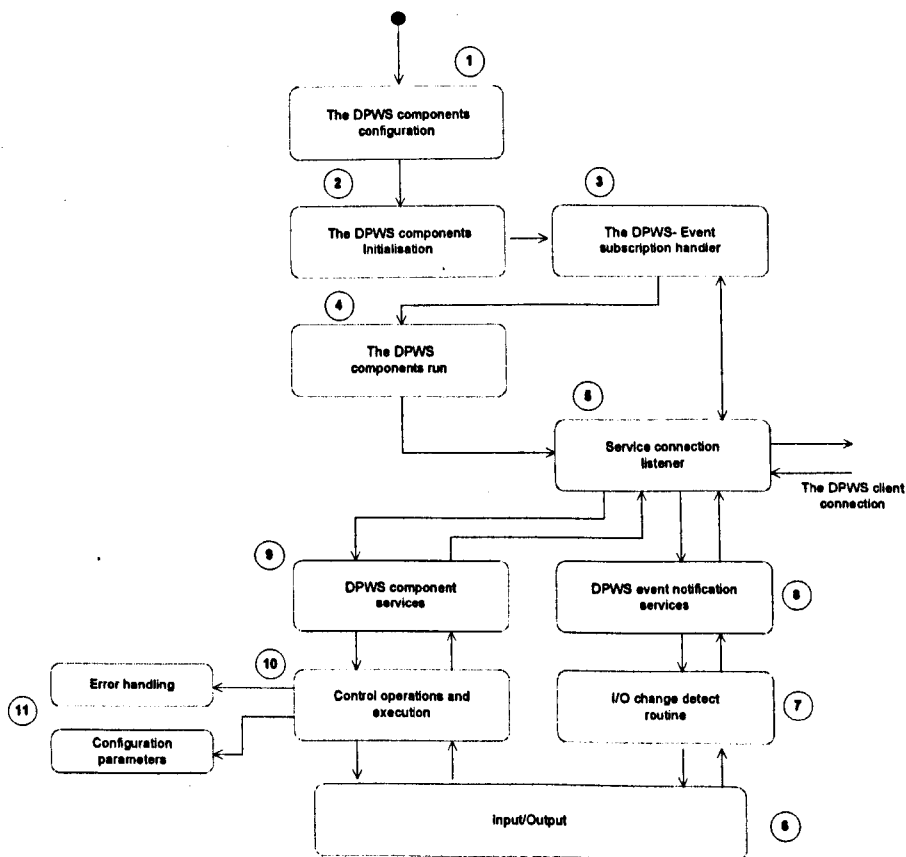
The DPWS acknowledgement/response message is replied to the service invocator as addressed at the <wsa:RelatesTo> tag to confirm the receipt of the message for the device operation (MoveExtended). In this SOAP message, the reply address in <wsa:RelatesTo> is a copy of <wsa:MessageID> obtained from the invocator.

APPENDIX D

Web Services Application Programming

The language written in the DPWS client and the DPWS- FTB device (server) operation including I/O execution, I/O timer, I/O state change report and diagnostic is based on C. The programming workflow of the client and the server application is schematically presented as follows:

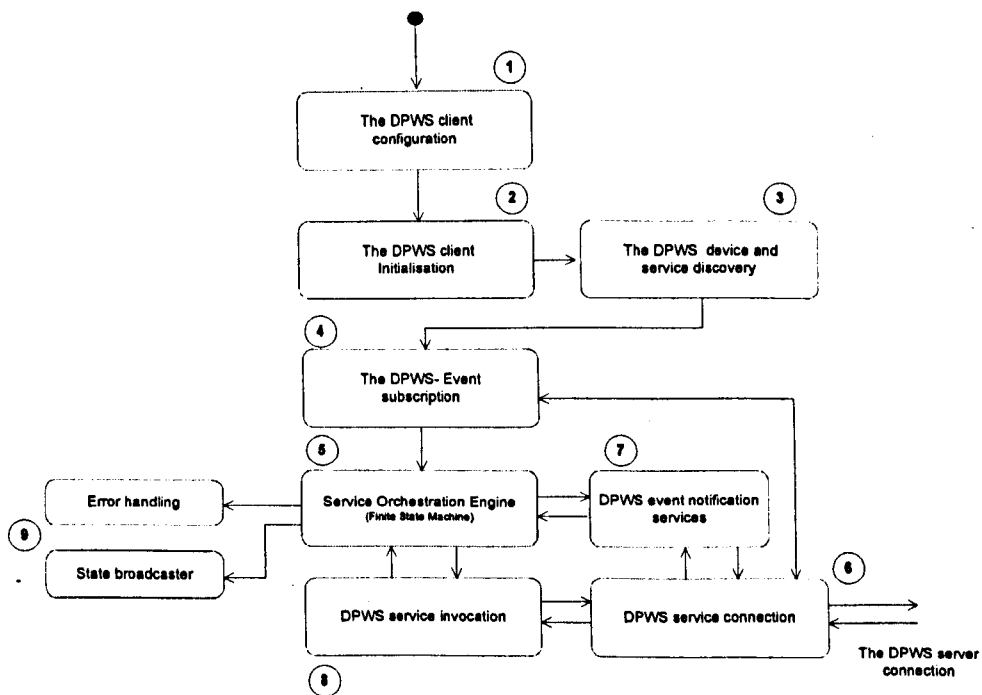
D.1 DPWS Server Application



- 1-2: The server application loads the network configuration (IP and MAC Address) and initialises control tasks for hosted components.
- 3-4: Then the DPWS components and the DPWS event handler for client subscriptions are initialised and started on the control device.
- 5: Having completed the start-up process, the server application is waiting for a connection to clients preceded by the client call or I/O tasks (state notification).
- 6-8: When device inputs or outputs change their states, this process calls the DPWS event handler to publish the device states to the subscribers.

- 9-10: The client application sets the connection to the server application for sending the DPWS command. Then the application parses the received SOAP message in order to initiate the specific DPWS call function interfaced to I/O operations on the control device.
- 11: Whilst the DPWS is running, the additional control tasks such as diagnostic routine, writing a log file and setting control parameters, are operating.

D.2 DPWS client application



- 1-2: The client application loads the network configuration (IP and MAC Address) and initialises DPWS command and notification utility for hosted components.
- 3-4: The client looks up the device by the service endpoint reference for the subscription process.
- 5-6: Having completed the start-up process, the client application is running the service orchestration engine which is interface to the DPWS event notification and service invocation. Then it is waiting for the connection from the server.
- 7: The client application (subscriber) receives the device state change notification when device inputs or outputs change their states through the DPWS event handler process.
- 8: The client application sends the DPWS command (SOAP message) in order to initiate the DPWS call function on the control device.
- 9: Whilst the DPWS is running, the additional control tasks such as diagnostic routine, writing a log file and setting control parameters, are operating.

The following figures show the sample codes implemented on the client and server application including DPWS configuration and component initialization, DPWS lookup service and subscription, DPWS state notification and service invocation, Service orchestration engine, and I/O state change detection and device operation.

Server Applications

D.3 DPWS Component Server Initialization

```

/*----- HOPPER * COMPONENT -----*/
/* create an ejector device class carrying device metadata */
hDevModel = dpws_create_device_model();
ls.s = "Ejector";
DPWS_SET_STR_ATT(hDevModel, DPWS_PTR_MODEL_NAME, &ls);
DPWS_SET_STR_ATT(hDevModel, DPWS_STR_MODEL_NUMBER, "ST-101");
DPWS_SET_STR_ATT(hDevModel, DPWS_STR_MODEL_URL, "http://www.schneider-electric.com/SmartTrap/ST-101/factsheet.html");
DPWS_SET_STR_ATT(hDevModel, DPWS_STR_PRESENTATION_URL, "index.html"); //ok!
ls.s = "Schneider Electric SA";
DPWS_SET_STR_ATT(hDevModel, DPWS_PTR_MANUFACTURER, &ls);
DPWS_SET_STR_ATT(hDevModel, DPWS_STR_MANUFACTURER_URL, "http://www.schneider-electric.com");

/* create a service class carrying the ejector port type */
hServClass = dpws_create_service_class();
DPWS_ADD_PTR_ATT(hServClass, DPWS_PTR_PREFIXED_TYPE, &EjectorPortType);
DPWS_ADD_PTR_ATT(hServClass, DPWS_PTR_WSDL, &hopperWsdL);
DPWS_ADD_PTR_ATT(hServClass, DPWS_PTR_HANDLING_FUNCTION, (const void*)&hop_serve_request);
DPWS_SET_STR_ATT(hServClass, DPWS_STR_ID, "urn:sdhopper");

/* Register the service class to the device class */
dpws_register_service_class(hDevModel, hServClass);
/* create a trap instance of the same class */
hLaHopper = dpws_create_device(0, hDevModel); /* the id must be unique in the physical local device */
DPWS_ADD_PTR_ATT(hLaHopper, DPWS_PTR_TYPE, &EjectorType);
DPWS_SET_INT_ATT(hLaHopper, DPWS_INT_METADATA_VERSION, 1); /* NOTE: should change everytime a change on the device impacts DPWS
metadata */
DPWS_SET_STR_ATT(hLaHopper, DPWS_STR_SCOPE, HOPPER_SUFFIX_SCOPE);
ls.s = "Ejector Server";
DPWS_SET_STR_ATT(hLaHopper, DPWS_PTR_FRIENDLY_NAME, &ls);
DPWS_SET_STR_ATT(hLaHopper, DPWS_STR_FIRMWARE_VERSION, "2.1xx");
DPWS_SET_STR_ATT(hLaHopper, DPWS_STR_SERIAL_NUMBER, "56080001");
DPWS_SET_PTR_ATT(hLaHopper, DPWS_PTR_USER_DATA, &EjectCylinderStatus1); /* instance device specific information */

dpws_enable_device(hLaHopper); /* prepares hello and activates the device */

```

BY: P. Phaitoonbuathong

DPWS component server initialization

D.4 DPWS Event Subscription Handler

```

// Service endpoint used as an event source for event notification
dpws_client_init(&event0.dpws, NULL);
DPWS_SET_PTR_ATT(hLaHopper, DPWS_PTR_USER_DATA, &event0); /* instance device specific information
event0.endpointRef0 = dpws_get_service_by_class(hLaHopper, hServClass);

```

BY: P. Phaitoonbuathong

DPWS event subscription handler

D.5 DPWS Event Notification Service

```

/*-----DPWS NOTIFICATION-----*/
int dpwsArmNotify(HandlingArmStatus1)
{
    switch (HandlingArmStatus1)
    {
        case Downstream:
            handlingNotifyElement.handlingArmStatus = hla__handlingArmStatus__Downstream_Position;
            break;
        case Upstream:
            handlingNotifyElement.handlingArmStatus = hla__handlingArmStatus__Upstream_Position;
            break;
        case Sort:
            handlingNotifyElement.handlingArmStatus = hla__handlingArmStatus__Sort_Position;
            break;
        case ARM_ERROR:
            handlingNotifyElement.handlingArmStatus = hla__handlingArmStatus__ARM_ERROR;
            break;
    }

    dpws_notify__hla__handlingNotifyEvent(&event0.dpws, event0.endpointRef0, &handlingNotifyElement);
    return 0;
}

```

BY: P. Phaitoonbuathong

DPWS event notification service

D.6 DPWS Component Provided Service

```

/*-----HANDLING ARM Cmd operation Utility-----*/
int __hla_handlingCmd(struct dpws* dpws, enum hla_handlingArmAction hla_handlingArmActionElement, struct hla_ResponseElement *hla_ResponseElement)
{
    NumOrder++;
    hla_ResponseElement->OrderID = NumOrder;
    hla_ResultElement.OrderID = hla_ResponseElement->OrderID;
    hla_ResultElement.allRequestStatus = hla_RequestStatus_ACK;
    //Message received acknowledgement
    dpws_notify__hla_handlingCmdAck(spState->dpws, pState->endpointRef0, &hla_ResultElement);

    switch (hla_handlingArmActionElement)
    {
        case hla_handlingArmAction_Move_Downstream:
            MoveArmDownstream();
            //Result acknowledgement
            hla_ResponseElement->allOperationResultStatus = hla_OperationResultStatus_DONE;
            dpws_end(spState->dpws);
            break;

        case hla_handlingArmAction_Move_Upstream:
            MoveArmUpstream();
            //Result acknowledgement
            hla_ResponseElement->allOperationResultStatus = hla_OperationResultStatus_DONE;
            dpws_end(spState->dpws);
            break;

        case hla_handlingArmAction_Move_Sort:
            MoveArmSort();
            //Result acknowledgement
            hla_ResponseElement->allOperationResultStatus = hla_OperationResultStatus_DONE;
            dpws_end(spState->dpws);
            break;

        default:
            //Result acknowledgement
            hla_ResponseElement->allOperationResultStatus = hla_OperationResultStatus_FAILURE;
            dpws_end(spState->dpws);
            break;
    }

    return SOAP_OK;
}

```

BY: P. Phaitoonbuathong

DPWS component provided service

D.7 Component I/O Operation

```

/*-----DEVICE I/O OPERATION-----*/
int MoveArmDownstream()
{
    trGPIO_Set(8, FALSE); //channel 08 Upstream pos
    trGPIO_Set(9, TRUE); //channel 09 Downstream pos
    /*Reading Arm input sensor*/
    do
    {
        sensor1=trGPIO_Get(1); //channel 01 At upstream sensor
        sensor2=trGPIO_Get(2); //channel 02 At downstream sensor
    }while(!(sensor1==FALSE&sensor2==TRUE));

    /*Break the ARM when position reached*/
    trGPIO_Set(8, TRUE); //channel 08 Upstream pos
    trGPIO_Set(9, TRUE); //channel 09 Downstream pos

    return 0;
}

```

BY: P. Phaitoonbuathong

Component I/O operation

D.8 I/O State Change and Error Detection

```

/*-----Detecting IO changes (CALLED FROM THE MAIN ROUTINE)-----*/
int IOSPY(void)
{
    Pre_HandlingArmStatus=ARM_START;

do
{
    sensor1=trGPIO_Get(1);
    sensor2=trGPIO_Get(2);
    sensor3=trGPIO_Get(3);
    // HANDLING ARM STATE

    if (sensor1==FALSE&&sensor2==TRUE&&sensor3==FALSE)
    {
        HandlingArmStatus1 = Downstream;
    }
    else if (sensor1==TRUE&&sensor2==FALSE&&sensor3==FALSE)
    {
        HandlingArmStatus1 = Upstream;
    }
    else if (sensor1==FALSE&&sensor2==FALSE&&sensor3==TRUE)
    {
        HandlingArmStatus1 = Sort;
    }
    else if (sensor1==FALSE&&sensor2==FALSE&&sensor3==FALSE)
    {
        if((HandlingArmStatus1==Move_Downstream)||((HandlingArmStatus1==Move_Sort)||
            (HandlingArmStatus1==Move_Upstream))
            //still in moving state(previously from this loop):ignore!
        )
        else//Start from static position
        {
            //at Downstream(direction=2)
            if((HandlingArmStatus1 == Downstream)&(Direction==3))
            {
                HandlingArmStatus1 = Move_Sort;
            }
            else if((HandlingArmStatus1 == Downstream)&(Direction==1))
            {
                HandlingArmStatus1 = Move_Upstream;
            }
            //at Sort(direction=3)
            else if((HandlingArmStatus1 == Sort)&(Direction==1))
            {
                HandlingArmStatus1 = Move_Upstream;
            }
            else if((HandlingArmStatus1 == Sort)&(Direction==2))
            {
                HandlingArmStatus1 = Move_Downstream;
            }
            //at Upstream(direction=1)
            else if((HandlingArmStatus1 == Upstream)&(Direction==2))
            {
                HandlingArmStatus1 = Move_Downstream;
            }
            else if((HandlingArmStatus1 == Upstream)&(Direction==3))
            {
                HandlingArmStatus1 = Move_Sort;
            }
        }
    }
    else
    {
        HandlingArmStatus1 = ARM_ERROR;
    }
}
//-----Entering Notification-----*/

if (HandlingArmStatus1 != Pre_HandlingArmStatus1)
{
    Pre_HandlingArmStatus1=HandlingArmStatus1;
    dpwsArmNotify(HandlingArmStatus1);
}
}
}
return 0;
}

```

BY: P. Phaithoonbuathong

I/O state change detection

Client Applications

D.9 DPWS Client Initialisation and DPWS Event Subscription

```

// DPWS client initialization
dpws_init();
dpws_client_init(dpws, NULL);

// ----- Begin EVENTS initialize
// initialize DPWS server for event reception
DPWS_SET_INT_ATT(DPWSCORE_REGISTRY_HANDLE, DPWS_INT_BOOT_SEQ, 0);
DPWS_SET_INT_ATT(DPWSCORE_REGISTRY_HANDLE, DPWS_INT_HTTP_PORT, port);
DPWS_SET_PTR_ATT(DPWSCORE_REGISTRY_HANDLE, DPWS_PTR_CALLBACK_HELLO, device_joining);
DPWS_SET_PTR_ATT(DPWSCORE_REGISTRY_HANDLE, DPWS_PTR_CALLBACK_BYE, device_leaving);

htkEndpoint = dpws_create_endpoint();
DPWS_ADD_PTR_ATT(htkEndpoint, DPWS_PTR_HANDLING_FUNCTION, hop_handle_event);

if (initServer(&listen_dpws)) {
    fprintf(stderr, "Could not initialize DPWS server...\n");
    exit(1);
}
tkEndpoint = dpws_get_local_endpoint_ref(dpws_get_default_service_port(htkEndpoint));

if (bootServer(&listen_dpws)) {
    fprintf(stderr, "Could not boot server...\n");
    exit(1);
}
// <----- End EVENTS initialize

(device lookup and Service endpoint discovery)
...
//Service subscription
invokationEPR = dpws_get_default_endpoint_reference(dpws, serviceProxy);
subsManager = dpws_endpoint_ref_dup(dpws_event_subscribe(dpws, invokationEPR, tkEndpoint, NULL, NULL, &duration));

```

BY: P. Phaithoonbuathong

DPWS client initialization& DPWS event subscription

D.10 DPWS Service Invocation

```

/*
 * DPWS ClientTask function invoking serverices
 */
int DPWSClientTask(int CmdCommand)
{
    if (CmdCommand == 0)
        command = MoveExtended;

    switch(command)
    {
        //-----HOPPER-----//
        case MoveExtended:
            // Call Open Trap
            if ((status = dpws_call_hop_ejectorCmd(&dpws, invokationEPR, NULL,
                hop_ejectCylinderAction_MoveExtended, &HopStateCmd) == DPWS_OK)
                {
                    printf("\n<- EXTENDED COMMAND -----> ");
                    printf("\n Ejector Extend command response :");
                    printf("\n Number ID : %d", ResponseElement.OrderID);
                    PrintMessage(ResponseElement.allOperationResultStatus);
                    //command responded from a server: DONE or FAILURE
                }
            else
                {
                    printf("\n<- Communication ERROR with Ejector ");
                }
            break;
        ...
    }

    command=Idle;
    dpws_release_proxy(serviceProxy);
    dpws_end(&dpws);
    return 0;
} //End ClientTask function

```

BY: P. Phaitoonbuathong

DPWS service invocation

D.11 DPWS Device and Service Discovery

```

// device lookup, discovery
deviceProxies = dpws_lookup(&dpws, SD_HOPPER_NS, HOPPER_DEVICE_TYPE, NULL, &nbDevices);
//service endpoint reference
serviceProxies = dpws_get_services(&dpws, deviceProxy, SD_HOPPER_NS, HOPPER_PORT_TYPE, &nbServices);
// invocation service endpoint reference
invokationEPR = dpws_get_default_endpoint_reference(&dpws, serviceProxy);

```

BY: P. Phaitoonbuathong

DPWS device and service discovery

D.12 Service Orchestration Engine (Finite State Machine)

```

/*-----//
 * FINITE STATE TRANSITION *
 *-----//
int Main ()
{
    do
    {
        //EJECTOR ELEMENT Eject WP by Retracting Cylinder
        if((MagState==Full)&&(MagXState==Empty)&&(EjectorState==Extended)&&(ArmState==Downstream))
            {
                //Move_Retracted call ClientTask(1, 1);
                EjectorState = ClientTask(1,1); //Moving Ejector
            }
        //SWIVEL DRIVE ELEMENT Pick up WP by Moving Arm Downstream Cylinder
        if((MagXState==Full)&&(EjectorState==Retracted)&&(ArmState==Downstream))
            {
                //Move arm to Mag call ClientTask(2, 5);
                ArmState = ClientTask(2,5); //Moving Arm
            }
    }
    }while(1)
}

```

BY: P. Phaitoonbuathong

Service orchestration engine

APPENDIX E

Packet Analysis of DPWS Operations

The average packet received time analysis of a single DPWS operation (E.1- Service invocation, E.2- Device state notification) is illustrated in the following tables. The data has been captured by a packet sniffer running on the PC to analyze the packet size and received time between the DPWS client (Host A- a PC) and server (Host B- FTB) application.

E.1 Average Time Analysis of the DPWS Service Invocation with a Reply Message

Packet	Packet size (Bytes)	Data load (Bytes)	Initial packet received time (ms)	Time different between packets (ms)	Event	Seq No.	Ack No.	Diagram		Packet SYNchronization (Seq,Ack)
								HOST A	HOST B	
1	78	0	0	0	Host A sends a TCP synchronize packet to Host B to initiate a connection	3976 +1	0	send t	t+1	Host B received the packet from Host A to start the connection
2	60	0	0.369	0.369	Host B replies Host A to acknowledge the request on the connection	6001 +1	3977	recv t+3	t+2 send	Host B replies Host A with New Seq and Ack=Seq(A)+1
3	60	0	1.116	0.727	Host A sends Acknowledge to Host B; The connection is established	3977 0; Data load	6002	send t+4	t+5	Host A replies with Seq= Ack(B) and Ack= Seq(B)+1
4	669	915	1.149	0.033	Host A sends a SOAP message to Host B	3977 915; Data load	6002	send t+6	t+7	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B
5	60	0	1.153	0.004	Host A sends another packet to Host B as the one to be acknowledged and also a request to close the connection from Host A to B	4892 +1	6002	send t+8	t+8	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B
6	60	0	2.006	0.853	Message received; Host B acknowledges Host A	6002 0; Data load	4893	recv t+11	t+10 send	Host B acknowledges Host A with Seq= Ack(A) and Ack= Seq(A)+1
7	610	756	11.301	9.295	Host B starts sending a response SOAP message to Host A and confirm the connection closed from Host A to B	6002 756; Data load	4893	recv t+13	t+12 send	The next sent a message from B to A with the Seq = Seq(B)+ data load and the same Ack
8	60	0	11.307	0.005	Host B sends a packet to Host A that requests a termination of the connection from Host B to A	6758 +1	4893	recv t+15	t+14 send	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A
9	60	0	13.083	1.776	Message received; Host A acknowledges Host B to terminate the connection from Host B to A	4893	6759	send t+16	t+17	Message received; Host A replies B with Seq= Ack(B) and Ack= Seq(B)+1
					Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection	8603	0	send t+18		Generate the new Seq and set Ack =0

E.2 Average Time Analysis of the DPWS State Change Notification

Packet	Packet size (Bytes)	Data load (Bytes)	Initial packet received time (ms)	Time different between packets (ms)	Event	Seq No.	Ack No.	Diagram		Packet SYNchronization (Seq,Ack)
								HOST B	HOST A	
1	80	0	0	0	Host B sends a TCP synchronize packet to Host A to initiate a connection	4001 +1	0	send t	t+1	Host A received the packet from Host B to start the connection
2	80	0	0.402	0.402	Host A replies Host B to acknowledge the request on the connection	0225 +1	4002	recv t+3	t+2 send	Host A replies Host B with New Seq and Ack=Seq(A)+1
3	80	0	0.728	0.324	Host B sends an acknowledgment to Host A; The connection is established	4002 0; Data load	0226	send t+4	t+5	Host B replies with Seq= Ack(A) and Ack= Seq(A)+1
4	794	740	2.852	2.126	Host B sends a SOAP message to Host A	4002 740; Data load	0226	send t+6	t+7	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A
5	60	0	2.856	0.005	Host B sends another packet to Host A as the one to be acknowledged and also a request to close the connection from Host B to A	4742 +1	0226	send t+8	t+9	Host B sends a packet with Seq= Seq(B) + data load and the same Ack as expected by A
6	60	0	3.348	0.483	Message received; Host A acknowledges Host B	0226 0; Data load	4743	recv t+11	t+10 send	Host A acknowledges Host B with Seq= Ack(A) and Ack= Seq(A)+1
7	168	114	5.845	2.296	Host A sends another packet to Host B to confirm the closing connection from Host B to A	0226 114; Data load	4743	recv t+13	t+12 send	The next sent a message from A to B with the Seq = Seq(A)+ data load and the same Ack
8	60	0	5.857	0.012	Host A sends a packet to Host B that requests a termination of the connection from Host A to B	0340	4743	recv t+15	t+14 send	Host A sends a packet with Seq= Seq(A) + data load and the same Ack as expected by B
9	60	0	6.042	0.385	Message received; Host B acknowledges Host A to terminate the connection from Host A to B	4743	0	send t+16	t+17	Message received; Host B replies A with Seq= Ack(A) and Ack= 0
					Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection	8001	0	send t+18		Generate the new Seq and set Ack =0

APPENDIX F

Publication Papers

1. P. Phaithoonbuathong, R. Harrison and C. S. Mcleod, "A Web Services Based Automation Paradigm for Agile Manufacturing," EPSRC 4th International Conference on Responsive Manufacturing (ICRM) on Agile Manufacturing System organised, 2007, School of Mechanical, Materials and Manufacturing Engineering, The University of Nottingham
2. P. Phaithoonbuathong, T. Kirkham, C. S. Mcleod, M. Capers, R. Harrison, R. P. Monfared, "Adding Factory Floor Automation to Digital Ecosystems; tools, technology and transformation," IEEE International Digital Ecosystems and Technologies Conference (IEEE- DEST), February 2008
3. T. Kirkham, D. Savio, H. Smit, R. Harrison, R.P. Monfared, P. Phaithoonbuathong, "SOA middleware and automation: Services, applications and architectures," 6th IEEE International Conference on Industrial Informatics 2008, (INDIN 2008.), 13-16 Jul 2008, Industrial Informatics, page 1419-1424 (2008)
4. P. Phaithoonbuathong, R. P. Monfared, T. Kirkham, R. Harrison, A.A West, "Web Services- based Automation for the Control and Monitoring of Production Systems," Paper has been accepted to International Journal of Computer Integrated Manufacturing, 2009
5. Y.S. Park, T.D. Kirkham, P. Phaithoonbuathong, R. Harrison, "Implementation Agile and Collaborative Automation using Web Service Orchestration," IEEE International Symposium on Industrial Electronics (ISIE) Conference, July, 2009

APPENDIX G

Website Reference

- [g1] "Global XML Web Service Architecture," Microsoft Corporation© Whitepaper, October, 2001, from:
<http://www.gotdotnet.com/team/XMLwebservices/Global%20XML%20Web%20Services%20Architecture%20White%20Paper.doc>
- [g2] C. Kaler, "WS-Security," Microsoft Corporation, Inc, April 2002, from:
<http://www-128.ibm.com/developerworks/library/ws-secure/>
- [g3] "Open Ethernet connectivity via Modbus/TCP – Product note," Honeywell Inc. Online articles, from:
http://www.dcab.se/HC900/Manuals/hc900_modbus_tcp_interface.pdf
- [g4] A. Yee, "Making Sense of the COM vs. CORBA Debate," SAGA software, 1999, from: <http://www.omg.org/news/whitepapers/index.htm>
- [g5] "a corba primer," Segue Software, Technical Whitepaper, from:
<http://www.omg.org/news/whitepapers/seguecorba.pdf>
- [g6] A. Skonnard, "Understanding SOAP," Microsoft©, March 2003, from:
<http://msdn.microsoft.com/webservices/webservices/>
- [g7] S. Schneider, G.P. Castellote, and M. Hamilton, "Can Ethernet Be Real Time," Real-Time Innovation (RTI) Whitepaper, from: <http://www.vmecritical.com/article/id?607>
- [g8] Dr. G. Turnbull, "Is Ethernet the answer to the Fieldbus Dilemma?," Online Articles, from:
<http://128.242.40.200/articles/george.asp?index=ieintro%2Etxt&title=Introducing%20Indu>
- [g9] N. Quaine, "SOAP Basic 2: Web Services and the Service Web," Online articles, from: <http://www.soapuser.com/basics2.html>
- [g10] N. Matta, B. Eynard, L. Roucoules, and M. Lemerrier, "Continuous capitalization of design knowledge," Acacia Publication, France, from:
<http://www-sop.inria.fr/acacia/WORKSHOPS/ECAI2002-OM/Actes/Matta.pdf>
- [g11] A. Radiya and V. Dixit, "The basic of using XML Schema to define elements," IBM ©, Online articles, August 2000, from:
<http://www.ibm.com/developerworks/xml/library/xml-schema>
- [g12] David Excoffier, Fabien Couble and Ludovic Mussier "WS-Management module for DPWS stack- Technical Specifications," Soda- ITEA Project, December 2008, from:
<http://www.soda-itea.org/Documents/objects/file1180018090.54>
- [g13] "XML Schema," W3C® Online articles, from: <http://www.w3.org/XML/Schema>
- [g14] "Revolutionising Plant Automation –The PABADIS Approach," PABADIS Consortium Whitepaper, October 2001, from:
http://www.uni-magdeburg.de/iaf/cvs/pabadis/downloads/final_deliverable_6_3.pdf
- [g15] "OPC Overview- OLE for process control," Emerson Process Management © Whitepaper, March 2007, from:
http://www.easydeltav.com/pd/WP_OPC_Overview.pdf
- [g16] N. Quaine, "SOAP Basic 3: SOAP Messages," Online articles, from:
<http://www.soapuser.com/basics3.html>
- [g17] Dr D. M. Anderson, "The end of the line for mass production: No Time for Batches & Queues," Online articles, 2003, from: www.build-to-order-consulting.com
- [g18] "Monitor Pro supervision software," Telemecanique, Online articles,
http://www.idom.ru/files/Schneider/Info/Monitor_Pro/Catalogues/Catalog_e.Pdf
- [g19] F. Curbera, W.A. Nagy and S. Weerawarana, "Web Services: Why and How," IBM T.J. Watson Research Center, August, 2001, from:
<http://www.research.ibm.com/people/b/bth/OOWS2001/nagy.pdf>
- [g20] "Industrial Ethernet: A Control Engineer's Guide," Cisco Systems Inc. Whitepaper, 2008, from:
http://www.cisco.com/web/strategy/docs/manufacturing/industrial_ethernet.pdf

- [g21] D. Gisolfi, "Web services architect, Part 3: Is Web services the reincarnation of CORBA?," IBM®, Online articles, 2001, from:
<http://www.ibm.com/developerworks/webservices/library/ws-arc3/>
- [g22] S. Patil, "Integration Approaches: Web Services vs Distributed Component Models PART II," IONA Technologies, Online articles, April 2003, from:
<http://soa.sys-con.com/node/39719>
- [g23] M. Henning, "Middleware Matter: Rise and Fall of CORBA," IONA Technologies, Online Forum, June 2006, from: <http://blogs.iona.com/vinoski/archives/000307.html>
- [g24] N. Murphy, "Introduction to CORBA for Embedded Systems," Online articles, from:
<http://www.embedded.com>
- [g25] M. Link, "SOA? CORBA?," Online forum, May 2007, from:
http://www.theserverside.com/news/thread.tss?thread_id=45613
- [g26] T. Baer, "SOA in the Real World," CBR, Online articles, February, 2006, from:
http://www.cbronline.com/article_cbr.asp?guid=FC75BE31-C48E-45A5-A74F-18A43F601727
- [g27] R. M. Bell, "An SOA Alternative to CORBA," Ratheon Network Centric Systems Whitepaper, June 2007, from:
www.omg.org/news/meetings/workshops/RT-2007/08-1_Bell.pdf
- [g28] T. Sturgeon, "Globalization and Jobs in the Automotive Industry," the Alfred P. Sloan Foundation research project, November 2000,
<http://web.mit.edu/ipc/publications/pdf/00-012.pdf>
- [g29] L. Tiezheng, "Global auto makers customize China products," China Economic Net, 2007, from: http://en.ce.cn/Insight/200712/10/t20071210_13863417.shtml
- [g30] S. Metzger, "Next Steps in the Small, Task-Oriented Vehicle Revolution," International Market Solutions, 2007, from:
<http://www.iuvmag.com/articles/jan07-3.htm>
- [g31] Professor A. P Graves, "Global Competition and the European Automobile Industry: Opportunities and Challenges," Online articles, from:
<http://imvp.mit.edu/papers/93/Graves/graves-1.pdf>
- [g32] D. Oetinger, "Automotive Supplier Excellence: Achieving Continuous Cost Reduction," Oracle Whitepaper, October 2002, from:
[http://www.oracle.com/industries/automotive/AutomotiveSupplierExcellence\(C14223-01\).pdf](http://www.oracle.com/industries/automotive/AutomotiveSupplierExcellence(C14223-01).pdf)
- [g33] M. Holweg, and A. Greenwood, "Product Variety, Life Cycles, and Rate of Innovation –Trends in the UK Automotive Industry," Lean Enterprise Research Centre, Cardiff University, from:
<http://www.3daycar.com/mainframe/publications/library/ProductVariety.pdf>
- [g34] "Open System Architecture for Controls within Automation Systems," OSACA II Final Report, April 1996, from: http://www.osaca.org/related_projects/osaca_2.htm
- [g35] "XLON XLDV32.DLL Programmer's Guide," DH electronics, Online articles, October 2005, from: <http://www.dh-electronics.de/eindex.htm>
- [g36] Prof. D.B. Rijsenbrij, "Component-based software development," Advance Solution Development, Online articles, from:
<http://home.hetnet.nl/~daanrijsenbrij/progx/eng/chapter8.htm>
- [g37] "TCP Connections," InetDaemon Enterprise, Online articles, from:
<http://www.inetdaemon.com/tutorials/internet/tcp/connections.shtml>
- [g38] "TCP 3-way Handshake," InetDaemon Enterprise, Online articles, from:
http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml