A service-oriented approach to embedded component-based manufacturing automation

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/9445

Publisher: © Navjot Kaur

Please cite the published version.
This item was submitted to Loughborough's Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.

![Creative Commons License](https://creativecommons.org/licenses/by-nc-nd/2.5/)

**Attribution-NonCommercial-NoDerivs 2.5**

You are free:

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

Under the following conditions:

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

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

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

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

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

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

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
A Service-Oriented Approach to Embedded Component-Based Manufacturing Automation

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

By

Navjot Kaur

February 2012

Wolfson School of Mechanical & Manufacturing Engineering
Loughborough University
United Kingdom
©Navjot Kaur 2012
To the Almighty...
Acknowledgement

I would like to take the opportunity to express my gratitude to my supervisors Prof. Robert Harrison and Prof. Andy West for their persistent motivation, support and farsighted guidance from the beginning of my research work. Their broad knowledge and deep insights have been an important resource to me in writing this thesis. The discussions and critiques made by them were of essence to the progress of this work. I would like to express my sincere thanks to Director of Research, Prof. David Whalley, for providing me encouragement and support in various ways.

Also special thanks to Pilkinton Library Staff, for the books, support and material needed to write this thesis. A special thanks to my department IT administrator Clive Turner, research administrator Mrs Jo Mason for their valuable support. Foremost, I would like to thank Ford Motors staff, for their trust and financial support, without which it was not possible to complete this project.

My deepest gratitude goes to my family for their unflagging love and support throughout my life, without which this dissertation is simply impossible. I am deeply thankful to my family for their moral encouragement without which this study would be difficult to complete. I would like to thank all colleagues and friends in the Department of Mechanical & Manufacturing Engineering for all the help and encouragement, I have received. As a result, research life became smooth and rewarding for me.

Last but not least, thanks to Almighty GOD (Devi Matta) for making my life more bountiful. May your name be exalted, honored and glorified!

Loughborough
February, 2012

Navjot Kaur
Abstract

This thesis is focused on the application of Component-Based (CB) technology to shop floor devices using a Service Oriented Architecture (SOA) and Web Services (WS) for the purpose of realising future generation agile manufacturing systems. The environment of manufacturing enterprises is now characterised by frequently changing market demands, time-to-market pressure, continuously emerging new technologies and global competition. Under these circumstances, manufacturing systems need to be agile and automation systems need to support this agility. More specifically, an open, flexible automation environment with plug and play connectivity is needed. Technically, this requires the easy connectivity of hardware devices and software components from different vendors. Functionally, there is a need of interoperability and integration of control functions on different hierarchical levels ranging from field level to various higher level applications such as process control and operations management services.

A potential solution is to realise a modular and reconfigurable automation system, based on a platform of reusable components; while simultaneously reducing the number of unique components. For designing such reusable components, a component-based technology has been used in current thesis. The automation components created are active and distributed entities interacting through their data, event and service ports. To communicate these entities, this thesis focuses on applying SOA and WS at device level. SOA and WS have already been proven successful in linking business applications. If SOA can be applied at shop floor using the embedded devices, it can enable entirely new automation architecture based on peer-to-peer interactions between autonomous devices. The adoption of this SOA-WS approach at shop floor level would enable a seamless integration of higher level business applications and shop-floor level system. This will also provide a loosely coupled message-oriented service in embedded device networks and geographically distributed automation system. SOA can be implemented using Web Services on the embedded devices, which will provide even greater flexibility and interoperability because WS is platform neutral. This SOA-WS approach will enable end users to operate and maintain the supplied system easily. A key aim of this thesis is to examine if the adoption of SOA-WS at the embedded control devices can provide the same level of message speed and reliability as the current control systems. It is expected that the desired distributed, loosely coupled and reconfigurable automation
system can be formed by a network of these collaborative autonomous SOA-WS based devices using an open control platform.

This approach has been experimentally evaluated both in terms of quantity and quality using various parameters involved in the design, implementation, evaluation and reconfiguration of SOA-WS based automation systems. This has been done using the Ford Festo test rig located at the Manufacturing System Integration Research Institute of Loughborough University. The mechanisms on this test rig represent control problems typically associated in engine assembly and handling machines. Therefore, the result of experimental studies performed on this test rig can be considered applicable to real manufacturing applications.

**Keywords:** Automation System, Component-Based Design, Service Oriented Architectures, Web Services, Device Profile for Web Services, Distributed Systems and Agile Manufacturing Systems.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Tables</strong></td>
<td>vii</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td>xii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Automation in globalised economy</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Reconfigurable components: A solution to industrial automation</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Research aim</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Research objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Contributions to knowledge of the thesis</td>
<td>5</td>
</tr>
<tr>
<td>1.6 Thesis structure</td>
<td>6</td>
</tr>
<tr>
<td><strong>2 Review of Current Manufacturing Systems</strong></td>
<td>8</td>
</tr>
<tr>
<td>2.1 Manufacturing technologies in Automotive Industry</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1 Mass customisation - Future of Automotive Industry</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Architecture of traditional automation systems</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Future manufacturing systems</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Overview of agile manufacturing</td>
<td>19</td>
</tr>
<tr>
<td>2.4.1 Related approaches toward agile manufacturing systems</td>
<td>21</td>
</tr>
</tbody>
</table>
## Contents

5.4 Deploying the software components ............................................ 91
  5.4.1 Physical resources .......................................................... 91
  5.4.2 Establishing communication channel ...................................... 92
    5.4.2.1 Communication models ............................................... 93
5.5 Stakeholders in implementing CB systems .................................... 94
5.6 Summary ................................................................................. 96

6 Engineering Methods for Web Services ............................................ 97
  6.1 Need of Web Services in CB automation systems ......................... 98
  6.2 Engineering requirements for CB design of agile automation systems . 99
  6.3 Integrating distributed CB automation systems ........................... 101
  6.4 Implementing SOA using Web Services ...................................... 103
    6.4.1 Embedded device level SOA ............................................. 104
    6.4.2 Device level SOA protocols ............................................. 105
  6.5 DPWS for implementing device level SOA .................................. 106
  6.6 Designing a CB automation system using Web Services & SOA ......... 109
    6.6.1 Building a control application using Web Services & SOA .......... 109
    6.6.2 Implementation ............................................................ 113
  6.7 Designing & deploying applications using ControlBuild .................. 115
    6.7.1 Implementing Peer-to-Peer interactions using ControlBuild ....... 122
  6.8 Summary ................................................................................. 125

7 Component-Based Web Services Test Set Up ..................................... 126
  7.1 Introducing Ford-Festo test rig ............................................... 126
    7.1.1 Decomposing Festo test rig ............................................ 128
    7.1.2 Work flow process of Festo test rig .................................... 132
  7.2 Selecting an embedded device .................................................. 134
    7.2.1 Smart Terminal Blocks .................................................... 136
  7.3 Programming STBs with ControlBuild ....................................... 140
  7.4 Summary ................................................................................. 152
A.6 Client code of belt component designed in ControlBuild . . . . . . . . . 230
A.7 IO configuration of device 1 using ControlBuild . . . . . . . . . . . . . 231
A.8 IO configuration of device 2 using ControlBuild . . . . . . . . . . . . . 232
A.9 UUID parameters of sensor component in ControlBuild . . . . . . . . . 233
## List of Tables

1.1 Contributions to knowledge. .................................................. 5

2.1 Problems associated with traditional manufacturing automation systems. 15

2.2 Requirements of the future manufacturing systems (after Phaithoonbuathong et al. (2008)). .................................................. 16

2.3 Key technologies, projects & consortiums focusing on agile manufacturing systems. 21

2.4 Comparison of key manufacturing automation systems technologies. ............ 31

3.1 Key Fieldbus technologies for designing automation systems. .................... 39

3.2 Key Industrial Ethernet technologies for designing automation systems. .......... 40

3.3 A comparison of peer-to-peer, client-server and publish-subscribe communication technologies. .................................................. 43

3.4 A comparison of traditional business-to-business applications and Web Services. 50

3.5 Assessment of CORBA and SOA in meeting the requirements of agile manufacturing systems. .................................................. 52

5.1 Example of assigning state numbers to control elements in DFSM approach. .... 85

5.2 State transition table reporting all the combinations of states of control elements of component 1. .................................................. 87
List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>State transition table reporting all the combinations of states of control elements of component 2.</td>
<td>87</td>
</tr>
<tr>
<td>6.1</td>
<td>Ford requirements for the CB design of an automation system (after Phaithoonbuathong et al. (2008))</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Role of Web Services &amp; SOA in meeting the end user requirements.</td>
<td>108</td>
</tr>
<tr>
<td>7.1</td>
<td>Decomposing Festo test rig.</td>
<td>130</td>
</tr>
<tr>
<td>7.2</td>
<td>Assigning state numbers to sensor and belt elements.</td>
<td>141</td>
</tr>
<tr>
<td>8.1</td>
<td>Comparing key characteristics of FTB &amp; STB.</td>
<td>155</td>
</tr>
<tr>
<td>8.2</td>
<td>Assigning state numbers to control elements of conveyor and separator components.</td>
<td>172</td>
</tr>
<tr>
<td>8.3</td>
<td>State transition table reporting all the combinations of states of control elements of conveyor component.</td>
<td>174</td>
</tr>
<tr>
<td>8.4</td>
<td>State transition table reporting all the combinations of states of control elements of separator component.</td>
<td>174</td>
</tr>
<tr>
<td>8.5</td>
<td>Assigning state numbers to control element of station 3.</td>
<td>180</td>
</tr>
<tr>
<td>8.6</td>
<td>State transition table reporting all the combinations of states of control elements of separator component of station 2.</td>
<td>180</td>
</tr>
<tr>
<td>8.7</td>
<td>Comparison with earlier studies of MSI Group.</td>
<td>191</td>
</tr>
<tr>
<td>8.8</td>
<td>Suitability in meeting the end user requirements.</td>
<td>193</td>
</tr>
</tbody>
</table>
## List of Figures

2.1 Key manufacturing technologies in the automotive industry (Anderson, 2004) ........................................ 9
2.2 Types of mass customisation (after Gilmore & Pine (1997)) ................................................................. 12
2.3 Generic architecture of an automation system (Pinceti, 2002) .............................................................. 14
2.4 The ARC advisory group collaborative manufacturing management model (after Mick & Polsonetti (2003)). Collaboration goes from shop floor to business level on the enterprise axis, from suppliers to customers on the value chain axis, and from design to support on the lifecycle axis. .................................................. 17
2.5 The role of collaborative automation ................................................................. 18
2.6 Architecture of intra-enterprise domain of the agile manufacturing system shown in Figure 2.4 .................................. 20
2.7 Architecture of RIMACS technology for integrating higher level business applications to shop floor applications (Cucinotta et al., 2009) .................................................. 23
2.8 Architecture of OPC technology for integrating higher level business applications to shop floor applications (Leitner & Mahnke, 2006) .................................................. 24
2.9 Architecture of OSACA technology for integrating higher level business applications to shop floor applications (Kosmopoulos, 2007) .................................................. 25
2.10 Framework of SIRENA technology .................................................................................................. 27
2.11 SOA and Web Services as a solution to future manufacturing system .................................................. 29
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The typical structure of a distributed automation system (after Castellote (2005)).</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Types of control architectures of automation systems (after Lee (2004)).</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>Schematic of client-server interactions.</td>
<td>42</td>
</tr>
<tr>
<td>3.4</td>
<td>Holonic elements and interfaces (Bussmann &amp; Schild, 2001).</td>
<td>45</td>
</tr>
<tr>
<td>3.5</td>
<td>Structure of holon manufacturing system (Fletcher et al., 2000).</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>The CORBA client-server interaction model (after (OMG, 2002)).</td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td>The role of Web Services in implementing SOA.</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>A comparison of conventional system and LonWorks system.</td>
<td>55</td>
</tr>
<tr>
<td>3.9</td>
<td>Basic structure of IEC 61499-1 function block (after Lopez &amp; Lastra (2006)).</td>
<td>56</td>
</tr>
<tr>
<td>3.10</td>
<td>Component-Based approach used in the COMPAG project (after Lee et al. (2004b)).</td>
<td>58</td>
</tr>
<tr>
<td>4.1</td>
<td>Key benefits of using Component-Based development.</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>SOA and Web Services integration framework (after Hung et al. (2005)).</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>Overview of research reported in this thesis.</td>
<td>68</td>
</tr>
<tr>
<td>5.1</td>
<td>Physical representation of an automation component.</td>
<td>71</td>
</tr>
<tr>
<td>5.2</td>
<td>Functional representation of the elements of an automation component (Lee, 2004).</td>
<td>72</td>
</tr>
<tr>
<td>5.3</td>
<td>Hardware part of an automation component.</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>States of a simple binary switch.</td>
<td>74</td>
</tr>
<tr>
<td>5.5</td>
<td>Functional representation of the software part of an automation component.</td>
<td>77</td>
</tr>
<tr>
<td>5.6</td>
<td>An example of designing control systems using distributed finite state machines.</td>
<td>80</td>
</tr>
<tr>
<td>5.7</td>
<td>Part 1: A stage by stage representation of a subsystem (Figure 5.6) consisting a distribution hopper &amp; swivel arm to illustrate the concept of DFSM.</td>
<td>81</td>
</tr>
<tr>
<td>5.8</td>
<td>Part 2: A stage by stage representation of a subsystem (Figure 5.6) consisting a distribution hopper &amp; swivel arm to illustrate the concept of DFSM.</td>
<td>82</td>
</tr>
<tr>
<td>5.9</td>
<td>An example of designing STDs of control elements.</td>
<td>85</td>
</tr>
<tr>
<td>5.10</td>
<td>An example of making a DFSM model of a system using STDs of its control elements.</td>
<td>86</td>
</tr>
<tr>
<td>5.11</td>
<td>Component-Based design framework for automation systems (Phaithoonbuathong, 2009).</td>
<td>90</td>
</tr>
<tr>
<td>5.12</td>
<td>(a) Client-server and (b) Peer-to-Peer communication model (after Sukanen (2002)).</td>
<td>94</td>
</tr>
<tr>
<td>5.13</td>
<td>Role of the stakeholders in an automation system.</td>
<td>95</td>
</tr>
<tr>
<td>6.1</td>
<td>Middleware layer in context.</td>
<td>102</td>
</tr>
<tr>
<td>6.2</td>
<td>Web Services based Service-Oriented Architecture.</td>
<td>104</td>
</tr>
</tbody>
</table>
List of Figures

6.3 DPWS protocol stack (Jammes et al., 2007). 106
6.4 An example of service orchestration ontology to build a control application. 112
6.5 The process of designing the CB automation system using SOA & Web Services. 113
6.6 Logical view of a physical device (Karnouskos et al., 2007). 116
6.7 Dynamic deployment toolchain of ControlBuild (Guerard, 2009a). 116
6.8 (a) FBD representation of control application in ControlBuild and (b) its corresponding automatically generated PLCopen XML file. 119
6.9 (a) Component view and (b) its corresponding I/O configuration file in ControlBuild. 120
6.10 (a) Web Services interface with its corresponding automatically generated WSDL file, (b) DPWS library function (in C) for a WS based component in ControlBuild. 121
6.11 Peer-to-Peer approach for device interaction. 122
6.12 (a) A client-server model for implementing peer-to-peer device interactions in a distributed automation system and (b) its implementation in ControlBuild. 124

7.1 Schematic diagram of the Ford-Festo test rig (Kirkham et al., 2008a). 127
7.2 Physical representation of the Ford-Festo test rig located at MSI laboratory of Loughborough University. 128
7.3 Distributed I/O interface module of the Festo test rig. 129
7.4 State transition diagram of the elements of distribution hopper. 130
7.5 State transition diagram of the elements of transfer arm. 131
7.6 State transition diagram of the elements of conveyor. 131
7.7 Workflow process of Festo test rig. 133
7.8 Architecture of a DPWS based embedded device (Jammes et al., 2007). 135
7.9 STB hardware module (Schneider, 2010). 137
7.10 (a) Network interface module, (b) power distribution module, (c) input module and (d) output module of STB 2311 module shown in Figure 7.9. 139
7.11 Island of the STB 2311 module. 140
7.12 (a)(b) Workflow process, (c) physical decomposition and state exchange, and state transition diagrams of (d) sensor and (e) belt of the conveyor system. 142
7.13 Designing sensor and belt components (Mac) in ControlBuild. 144
7.14 Sub-components, FBD of sensor component displaying belt state received from belt component in ControlBuild editor. 145
7.15 Sub-components, FBD of belt component interlocked with sensor state of sensor component in ControlBuild editor. ......................................................... 146
7.16 Web Service interface of (a) sensor and (b) belt components in ControlBuild. . . . 148
7.17 (a) Embedded tree view with WS utility and (b) Upload manager in ControlBuild. . 151
7.18 Materna (DPWS) explorer showing the list of connected devices in a network. . . 152
8.1 Layout of test setup used on Ford-Festo test rig to implement a WS based fully distributed automation system. ..................................................... 157
8.2 Layout of test setup used on Ford-Festo test rig to implement a WS based automation system using a centralised orchestrator (Phaithoonbuathong, 2009). .......... 158
8.3 OS and software architecture of FTB (Phaithoonbuathong, 2009). ................. 159
8.4 OS and software architecture of STB. ..................................................... 159
8.5 DPWS interface for establishing client-server relationship on STB devices. ........ 161
8.6 Client-server interactions within the STB devices connected to form a feedback control system. ................................................................. 163
8.7 Ethernet TCP/UDP packet structure. ..................................................... 164
8.8 Ethernet TCP/IP network communication. ............................................. 166
8.9 Adding a separator component on the conveyor belt station. ....................... 171
8.10 STD of sensor and separator control elements. ....................................... 173
8.11 A DFSM model of station 2 using STDs of control elements. ....................... 173
8.12 Component view of station 2 in ControlBuild before adding the separator component. 177
8.13 Component view of station 2 in ControlBuild after adding the separator component. 177
8.14 Adding the indexing rotary table (station 3). ....................................... 179
8.15 STD of WP1TS sensor control element of rotary table component. ............... 179
8.16 Logical view of station 2 and station 3 on STB2 & STB3 devices respectively. . . 180
8.17 DFSM model of station 2 & station 3 using STD's & interlocking of corresponding control elements. ......................................................... 181
8.18 Component view of station 3 (in ControlBuild) publishing its WP1TS sensor state to station 2. ................................................................. 182
8.19 Modifying the workflow process by removing station 2. ............................. 185
8.20 Logical view of (a) stations 1/2/3 on STB1,2,3; and (b) logical view of station 1/3 on STB1,3 after removing the station 2. ................................. 185
8.21 Component view of station 3 in ControlBuild after removing station 2. .......... 186
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.22</td>
<td>Component view and FBD of (a) WP1 sensor and (b) WP1TS sensor elements.</td>
<td>187</td>
</tr>
<tr>
<td>8.23</td>
<td>Integration of the designed WS based control system and SOA middleware.</td>
<td>188</td>
</tr>
<tr>
<td>8.24</td>
<td>A framework to integrate designed control application with PDE tools.</td>
<td>189</td>
</tr>
<tr>
<td>9.1</td>
<td>Author’s original work and novel contributions.</td>
<td>200</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter presents an overview of the challenges, faced by automation systems, in meeting the demands of future generation manufacturing paradigms; enabled by the shift of mass production systems toward mass customised systems. As a solution to these challenges, a Component-Based design approach using Service-Oriented Architectures and Web Services has been introduced. At the end, it summarises the aim, research objectives and the outline of thesis.

1.1 Automation in globalised economy

In recent years, competition in manufacturing industries has significantly increased due to globalisation and is expected to intensify further in future. Because of this increasing competition, industries are facing significant difficulties staying in business, growing and maximising their profit. As a solution, they are looking forward to respond customer’s requirements and market products more efficiently and quickly. They are trying to bring new products at reduced price, improved quality and better working life. To do so, most manufacturing industries adopted a strategy to design and manufacture products across geographical boundaries, and distribute and market these
worldwide (Shimokawa, 2000). Industries are focusing on integrating product engineering and manufacturing engineering, i.e., knowledge sharing between suppliers and customers/end users during the conception and design stages. More specifically, they are looking forward to integrate and improve the geographically distributed manufacturing operations, with increased flexibility and efficiency (Harrison et al., 2006). This will help to meet the global market demands by having shorter product development cycles, improved product quality and reduction in cost both to manufacturer and customer.

To achieve this, most manufacturing industries are focusing on improving and increasing automation in the manufacturing processes. This involves moving away from the traditional automation systems based on machine controllers such as Programmable Logic Controllers (PLC) and Centralized Control Systems (CCS) (Setchi & Lagos, 2004). It is because these systems are based on a rigid centralised approach and their reconfiguration is currently a complicated and time consuming process (Fan & Wong, 2003; Patrick & Fardo, 2009). This is why, the traditional design approaches present many deficiencies, especially when used as a basis for an intelligent manufacturing control system. Some of these includes: low flexibility, reusability and interoperability; high initial installation cost and the high cost associated with the frequent product development and redesign of operations (Bollinger, 1998). For example, according to Harrison et al. (2006), in most of the present manufacturing plants, approximately one third of the total cost over their lifetime is spent on the installation and set-up of the automation system. The above mentioned adverse effects will be expected to get even worse in future. This is because, in future, the industries will be expected to face frequently changing market demands, greater time-to-market pressure, continuously emerging new technologies and, above all, global competition. This will lead to the requirement for distributed and open control automation systems, with considerably more flexibility and adaptability to change, than the present automation systems can afford. Therefore in this thesis, a Component-Based design approach is proposed as a solution to yield a suitable distributed automation system. In particularly, this approach is based on using reconfigurable components as building blocks for designing distributed automation systems. It is expected to give end users a more effective way of reconfiguring their automated machinery with higher level of flexibility at lower
cost. Therefore, the research reported in this thesis investigates the applicability and appropriateness of adopting Component-Based re-configurable automation systems.

1.2 Reconfigurable components: A solution to industrial automation

A solution to increase flexibility and adaptability in industrial automation systems is to use a reconfigurable component-based design (CBD) approach. A CBD approach for automation systems had already been researched at the Manufacturing Systems Integration (MSI) research institute of Loughborough University, in collaboration with the Ford Motor company. This research aimed to investigate the potential of a CBD approach in improving the flexibility and re-configurability within the engine assembly lines of Ford Motor company (Anon, 1998; Lee, 2004). The key outcomes of the research studies at MSI institute are: an 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 the automation components can be configured, simulated and deployed. To date, these are not fully matured for industrial applications and are under development, and have been only tested under controlled laboratory conditions. From the results of the research studies of the MSI institute, it has been anticipated that this work will provide a new design approach for a highly flexible and reconfigurable automation system, simultaneously reducing the development cost and time to market for new products. However, the reported work to date is based on a centralised approach with a request-respond handshaking principle and is limited to very small scale integration. For example, distributed control system based on LonWorks, previously developed by Lee (2004), implicit network variables and initial Web Services based research done by Phaithoonbuathong (2009) is based on centralised control, request-response and partial eventing. The reported work has several shortcomings such as limited access and low flexibility, which needs to be addressed before making it suitable for industrial applications. Therefore, the research presented in this thesis is an extension of the studies made by MSI institute on the CB reconfigurable distributed systems.
In particularly, the current thesis presents an innovative way of integrating the reconfigurable components using a peer-to-peer approach, with Service Oriented Architectures and Web Services as communication platform. In this approach, the whole manufacturing system is divided into components, where each component can be easily redesigned, changed and accessed through SOA & Web Services technologies. SOA and Web Services have already been widely researched to support business-to-business integration (Marks & Bell, 2006). Therefore, research has now been targeted on whether these technologies can be applied in industrial sectors to integrate business to shop-floor applications. Thus, the research reported in this thesis is focused on communicating the shop-floor level devices using the application of SOA and Web Services. It is supposed to give comparatively increased flexibility, and low cost to manage and redesign individual components than the complete traditional automation systems. It will also provide an open, flexible and agile environment with plug-and-play connectivity.

1.3 Research aim

The research presented in this thesis aims to develop next generation business-driven automation systems using predefined component based modules through investigation of the suitability of applying SOA & Web Services to shop floor level devices.

1.4 Research objectives

The research reported in this thesis is carried out as a sub-project of the work undergoing in MSI research group at Loughborough University; with the following objectives:

1. To review manufacturing industry automation requirements, standards and technologies, in situ operational restrictions and problems associated in the presence of globalised production characteristics.

2. To understand the role of automation systems in benefiting manufacturers; especially in terms of the cost savings associated with the design & installation of such systems, and the reduction in production lead times.
Chapter 1: Introduction

3. To investigate a strategy to decompose manufacturing operations into user manageable component modules appropriate for designing & establishing automation systems.

4. To develop a SOA & Web Services based prototype distributed automation model using reconfigurable components.

5. To perform a pilot study trial of the developed model, aiming to investigate its efficiency in fulfilling the user requirements (such as reconfigurability and interoperability) for agile manufacturing system.

In summary, the focus of the research reported in this thesis is on investigating the feasibility of applying SOA and Web Services to low level devices for shop floor applications.

1.5 Contributions to knowledge of the thesis

In the achievement of the above research objectives, the contributions made are reported in Table 1.1.

<table>
<thead>
<tr>
<th>Contributions to knowledge</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensions for a methodology to decompose automated industrial operations into user manageable components</td>
<td>5</td>
</tr>
<tr>
<td>Extensions for a methodology to design reconfigurable and reusable components based on the state variables</td>
<td>5</td>
</tr>
<tr>
<td>A methodology to design reconfigurable and reusable components with Web Services interface using graphical based tools</td>
<td>6</td>
</tr>
<tr>
<td>A new method of integrating devices using choreography based peer-to-peer interaction through SOA &amp; Web Services interfaces</td>
<td>6,7</td>
</tr>
<tr>
<td>A client-server architecture using graphical based tools for enabling peer-to-peer interactions within devices</td>
<td>6,7,8</td>
</tr>
<tr>
<td>A novel procedure of integrating these Web Services based SOA components with higher level applications</td>
<td>8</td>
</tr>
</tbody>
</table>
1.6 Thesis structure

The main body of this thesis comprises nine chapters; with their outline provided below:

- **Chapter 1: Introduction**

  This chapter presents an overview of the importance of automation systems in manufacturing industries; along with their associated problem in meeting the demands of future generation manufacturing systems. As a solution to these problems, a component-based distributed automation system using SOA and Web Services is briefly introduced. Finally, the chapter ends presenting the aim, objectives and outline of thesis.

- **Chapter 2: Review of Current Manufacturing Systems**

  This chapter presents the literature review related to the drivers of change of current manufacturing systems towards agile manufacturing. A review of traditional automation systems is presented and their problems associated in meeting the requirements of future generation agile manufacturing system are highlighted. The key enabling technologies both in software and hardware corresponding to automation system requirements are also detailed. Finally at the end, it presents a summary of the requirements for developing the next generation automation systems.

- **Chapter 3: Current Manufacturing System Technologies**

  This chapter discusses the technologies required for implementing collaborative automation systems. It presents a discussion of available technologies for the implementation of distributed control systems, their logic applications, and middleware. Finally, a comparison of these technologies is done and a selection is made to support the development of agile manufacturing systems.

- **Chapter 4: Research Methodology and Design**

  This chapter presents a new component based automation system with the application of SOA and Web Services. It highlights the challenges associated with developing this approach and the research methodology employed to overcome these challenges in the development and implementation of component-based automation systems.
Chapter 1: Introduction

• Chapter 5: Developing Component-Based Automation Systems
  This chapter presents the physical and functional representation of automation components; which can be used individually or integrate with others using SOA and Web Services to make an automation system.

• Chapter 6: Engineering Methods for Web Services
  This chapter presents the selection of a suitable engineering method, for designing software based control applications and integrating these using SOA and Web Services to make an automation system.

• Chapter 7: Component-Based Web Services Test Set Up
  This chapter presents a test rig system used for implementing the concept of CB automation designed using the engineering method selected in Chapter 6. It also presents the selection of an embedded device on which the designed software based components can be deployed. Finally, the chapter ends by presenting an example of designing a CB control application with a WS interface using the selected method and deploying it on the selected device.

• Chapter 8: Implementation & Evaluation
  This chapter investigates the feasibility of SOA & Web Services based CB automation systems within real control applications, supporting engineering tools through investigating the system evaluation on the test rig mentioned in Chapter 7. It evaluates the performance of the design and its re-configuration capabilities against the requirements of manufacturing automation system.

• Chapter 9: Conclusions and Recommendations for Future Work
  Finally, this chapter presents the main conclusions drawn from the thesis and recommendations for future work based upon the exploitation of Web Services in industrial control systems.

  This thesis also contains an appendix providing supporting documentation related to the coding (e.g. XML & WSDL) of the implemented system and application example.
Chapter 2

Review of Current Manufacturing Systems

This chapter presents an overview of traditional manufacturing technologies and their shift to mass customisation; primarily due to globalisation, unstable markets and growing competition. It describes the concept of mass customisation of production and lists the requirements of next generation manufacturing automation systems capable of supporting mass customisation. A review of various existing and emerging manufacturing technologies is also presented. Finally, a comparison of these technologies is made to identify their threshold and potential in meeting the requirements of next generation manufacturing automation systems.

2.1 Manufacturing technologies in Automotive Industry

Since the industrial revolution (a period from 18th to 19th century), many manufacturing technologies have evolved; however, only key relevant technologies are discussed in this chapter. These are summarised in Figure 2.1 and detailed below:

- **Mass production**: Mass production was originally introduced in late 18th century. However, it was popularized in 19th century by Henry Ford’s Ford Motor
Company, which introduced electric motors to the then-well-known technique of chain production (Shiwanand et al., 2006). This usage of electric motors allowed mass production manufacturers to then produce relatively more per worker-hour and simultaneously reduce the manufacturing cost of end product. It happened because then market was characterised by stable demand with little variation in products and few competitors. The manufacturers simply made profit by producing large quantities of a product in order to minimise production costs. These characteristics made mass production successful. However on the downside, mass production based systems are inflexible as it is difficult to alter a design or production process after a production line has been implemented. Mass production is also capital and energy intensive, as it uses a high proportion of machinery and energy in relation to workers.

![Figure 2.1: Key manufacturing technologies in the automotive industry (Anderson, 2004).](image)

- **Lean production**: Lean production is also known as the Toyota production system or just-in-time production (Anderson, 2004). It is an assembly-line production practise based on the principle of getting the right things to the right place at the right time in the right quantity to achieve perfect work flow; while minimising waste, and increasing flexibility and ability to change.

The principles of lean production were developed originally by Toyota’s engineer Taiichi Ohno, after World War II (Shiwanand et al., 2006). These principles include: pulling production from customer demand, perfect first-time quality, waste minimisation, minimising inventory, continuous improvement, flexibility,
design for rapid changeover, empowering workers, building and maintaining a relationship with suppliers, load leveling and maximising production flow and visual control.

Lean production proved to be a successful technology for Toyota; however, it had several shortcomings and challenges (Abdullah, 2003; Shiwanand et al., 2006). For example, the implementation of lean manufacturing requires a high level of support from employees (especially leaders); and additional training, which usually increases expenses. Another challenge of lean manufacturing is the task of maintaining resources and building good partnership with suppliers. For example, a shipment of defective goods or parts from a supplier can result into a low-quality final product or even no product at all.

- **Mixed model production**: Mixed model production was also developed by Toyota in 1960’s as a solution to the problems created by line changeovers and long runs on the assembly lines. It is based on the principle of assembling several distinct models of a product on the same assembly line without changeovers and then sequencing these models in a way that smoothes the demand for upstream components. It uses the Demand Flow Technology (DFT), which offers many benefits such as simplicity in use, provision of step-by-step guide for converting production from a scheduled-push to a demand-pull and flow system, and a customer-centric approach (Miller & Kampouris, 1994). In the customer-centric approach, DFT places customer at the center of operation to guide behaviour in the organisation and aligns business and customer goals. It also helps unifying financial and customer objectives for managing operating capital and business growth. However on the downside, its scope is limited to production planning, and requires extensive manual work for production process definition.

- **Flexible manufacturing system (FMS)**: FMS refers to a manufacturing system, which can flexibly react to both predicted and unpredicted changes (Carlsson, 1995). However, this flexibility is limited and can be divided into two categories: *machine flexibility* and *routing flexibility*. The *machine flexibility* covers the system’s ability to change to produce new product types, and ability to change the order of operations executed on a part. *Routing flexibility* refers to the ability to use multiple machines to perform the same operation on a part, as
well as the system’s ability to absorb large-scale changes (e.g. in volume or capability). In FMS, these flexibilities are introduced by using advanced machines and systems such as robots, computer-controlled machines, numerical controlled machines (CNC) and sensors. The key advantages of FMS include lower cost/unit, higher productivity, improved machine efficiency, improved quality, increased system reliability, reduced parts inventories and shorter lead times. However on the downside, it is expensive to implement.

- **Mass customisation:** Over the last couple of decades, organisations all over the World face significant transformational changes due to global competition, low volume customised products, reduced product time to market, high market turbulence, fast growing customer demands, technological obsolescence, mergers, acquisitions and globalisation (Kazan & Baydar, 2007). The frequency of these changes is expected to further increase in future. As a result, these changes have created customer oriented environments, in which customer satisfaction is the prime success factor in addition to efficiency, effectiveness, productivity, quality and competitiveness. Therefore in recent years, a growing number of researchers, such as Pine (1993) and Feitzinger & Lee (1997), are looking towards the emerging paradigm of mass customisation as an attractive approach to meet these challenges. By definition, the term mass customisation refers to the production and distribution of customised goods and services quickly and cost effectively on a mass basis (Davis, 1989). Mass customisation calls for flexibility and quick responsiveness using computer aided manufacturing. In addition, it reconfigures the environment, people, processes, units and technology to give customers exactly what they want at relatively low cost and reduced time scale.

In summary, mass customisation

at its core offers a tremendous increase in variety and customisation without a corresponding increase in costs. At its limit, it is the mass production of individually customised goods and services. At its best, it provides strategic advantage and economic value.
Thus, from the above discussion, it can be inferred that mass customisation is the key technology to satisfy the challenges of today’s and future manufacturing industries. This is why the main focus of this thesis is to adopt mass customisation in designing manufacturing automation systems.

2.1.1 Mass customisation - Future of Automotive Industry

From the above discussion it is clear that mass customisation is suitable in situations of today’s and future unstable environments and markets. In automation industry, mass customisation can be better expressed as the ability to provide uniquely individualised products and services which can satisfy any manufacturing requirement in a cost-effective way (Pine, 1993). Mass customisation itself is of several types.

According to Gilmore & Pine (1997), there are four types of mass customisation as shown in Figure 2.2. These are described below:

- **Adaptive customisation**: In this approach, a firm produces a standardised product; however, this product is customisable in the hands of end-user, i.e., the customers alters the product themselves.

- **Transparent customisation**: The firms adopting this approach provide individual customers with unique products, without explicitly telling them that the
products are customised. This is done by accurately assessing the customer needs.

- **Cosmetic customisation:** In this approach, the firms produce a standardised product; however, market it to different customers in unique way.

- **Collaborative customisation:** This is the most flexible approach. In this approach, the firms talk to individual customers in order to determine the precise product offering that best serves the customer’s needs. This information is then used to specify and manufacture a product that suits that specific customer.

Among the above four different types of mass customisation, the collaborative mass customisation offers most flexibility and thus future generations of manufacturing automation systems should aim to support its needs. The following section describes the traditional automation systems and how collaborative mass customisation impacts on future generation automation systems.

### 2.2 Architecture of traditional automation systems

The typical generic architecture of an automation system based on the standard architecture ANSI/ISA-95 is shown in Figure 2.3 (Dalwalla *et al.*, 2007; Pinceti, 2002). This is generally applicable in both the manufacturing and process industries. It looks like pyramid and is generally composed of following four integrated levels:

- **Management level:** This is the top level of pyramidal architecture. It represents the communication and information system (CIS), using which different sub-divisions of company interact to play their specific roles e.g. sales, research, purchasing and finance. The key roles of this level are to manage enterprise-level finance, resource planning and distribution, workflow planning and order management and fulfilment.

- **Process/supervisory control level:** This is the second level from top and is also called area control unit. It is usually involved in operations such as quality management, order tracking, manufacturing operations & control, dispatching, product scheduling and reliability assurances. It comprises of network workstations such as SCADA (Supervisory Control And Data Acquisition) for supervising
sites and monitoring the automation process control at the field level and supporting data acquisition processes.

- **Field/machine control level:** This represents local control units employing different technologies such as PLCs and intelligent I/O controllers. Their purpose is to interact with field devices (e.g. sensors and actuators).

- **Actuator-Sensor level:** This is the base of pyramid hierarchy and usually made by conventional and/or intelligent devices (sensors/actuators) in the field. These devices interact with the physical manufacturing environment to perform the tasks and/or to collect the data required for higher level data monitoring and acquisition operations. The devices at this level are usually connected to higher field level through physical communication mediums such as fieldbuses and/or industrial networks.

From the above description, it can be interpreted that the pyramid model is based on a rigid and hierarchical structure. From the literature, such as Lee (2003, 2004); Phaithoonbuathong et al. (2008) and Carlsson (1995); it has been found that this model...
Chapter 2: Review of Current Manufacturing Systems

will offer many deficiencies (e.g. poor flexibility and reconfigurability) when used as basis for future generation manufacturing systems. These problems are tabulated in Table 2.1.

Table 2.1: Problems associated with traditional manufacturing automation systems.

<table>
<thead>
<tr>
<th>Problems</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of flexibility</td>
<td>• Rigid hierarchical structure; thus, difficult to re-plan, reschedule and after after implementation, if required.</td>
</tr>
<tr>
<td></td>
<td>• Poor flexibility &amp; agility, i.e. the ability to respond in a time and cost efficient manner to both planned &amp; unplanned changes.</td>
</tr>
<tr>
<td></td>
<td>• Suitable for only centralised and sequential manufacturing systems.</td>
</tr>
<tr>
<td>Poor reusability and reconfigurability</td>
<td>• Reusability and re-configurability are difficult to support and the whole system is usually replaced, when new designs are required.</td>
</tr>
<tr>
<td></td>
<td>• Tight coupling between hardware and their control software.</td>
</tr>
<tr>
<td></td>
<td>• Supporting any change is usually time consuming and expensive.</td>
</tr>
<tr>
<td>Difficult to design &amp; implement</td>
<td>• High complexity as the system being composed of diverse automation peripherals.</td>
</tr>
<tr>
<td></td>
<td>• Design and development is reliant upon experienced professionals and usually involve a large amount of their effort and time, which is generally very expensive.</td>
</tr>
<tr>
<td>Engineering tools &amp; late verification of</td>
<td>• Lack of engineering tools that are capable of supporting concurrent engineering tasks throughout the lifecycle and design phase of automation systems.</td>
</tr>
<tr>
<td>automation systems</td>
<td>• Verification of system(s) cannot be undertaken until the completion of all the required elements (e.g. mechanical, electrical, fluid elements and control software).</td>
</tr>
<tr>
<td>Cost effectiveness</td>
<td>• On the shop floor, centralised manufacturing systems are not favored because they involve high investment costs at implementation stage.</td>
</tr>
<tr>
<td></td>
<td>• For any alteration, the whole system is usually changed because devices or elements under the central control unit are reliant upon the central controller.</td>
</tr>
<tr>
<td></td>
<td>• This type of the system is also at risk of a single point failure that can result in the whole system being inoperable.</td>
</tr>
<tr>
<td>Diversity and complexity of automation</td>
<td>• Automation systems are usually composed of various types of vendor specific systems such as Human Machine Interfaces (HMI's), data monitoring systems and control systems. These vendor specific units work on different operational platforms such as Microsoft Windows and UNIX with different communication mediums such as Ethernet and Fieldbus. This makes the overall automation system very expensive and complex to maintain as it requires the use of heterogeneous tools and experts with knowledge of diverse technologies.</td>
</tr>
<tr>
<td>devices</td>
<td></td>
</tr>
<tr>
<td>Lack of remote diagnostics and support</td>
<td>• These systems lack remote support. This usually result as end-users not getting immediate support from machine vendors when machines breakdown. 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.</td>
</tr>
</tbody>
</table>
2.3 Future manufacturing systems

As discussed before, the typical characteristics of future manufacturing systems include adapting to frequently changing market demands, time-to-market pressure, continuously emerging new technologies and, above all, global competition. The requirements for such systems are mentioned in Table 2.2. These requirements are listed based on the above mentioned issues and problems with traditional automation systems.

Table 2.2: Requirements of the future manufacturing systems (after Phaithoonbuathong et al. (2008)).

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seamless integration</td>
<td>Systems should be capable of reacting to changes with less (or ideally without) effect on other processes within the manufacturing system.</td>
</tr>
<tr>
<td>Simplicity in design</td>
<td>The various units such as mechanical, electrical, fluid and control units of system should be designed independently. However, these need to be available in common formats so that these can be easily integrated to build the complete system.</td>
</tr>
<tr>
<td>Reconfiguration</td>
<td>The control platform should have built-in re-configurability at runtime in order for the system updates to be carried out on-line. It means that the control devices and automation components can describe themselves to designers, human operators and other automation components.</td>
</tr>
<tr>
<td>Reusability</td>
<td>To improve design and quality of products, applications should, as much as possible, be based on proven solutions. This will help reusing features/modules of existing system within the new automation system. This will save time and development costs.</td>
</tr>
<tr>
<td>Interoperability and non-vendor specific platforms</td>
<td>There is a need for interoperability between control products, as these are usually supplied from different vendors. The software based components should be designed on open-platforms in order to improve flexibility, capability and reliability of system.</td>
</tr>
<tr>
<td>Intelligent machines</td>
<td>To increase flexibility in plant operations, the manufacturing equipments should be intelligent and capable of performing multiple tasks in co-operation.</td>
</tr>
<tr>
<td>Visualisation and simulation</td>
<td>The functional and behavioral capability of control system needs to be evaluated prior to installation. This will minimize time for doing late validation.</td>
</tr>
<tr>
<td>Exception handling, fault tolerance and recovery</td>
<td>In addition to changes in products and production schedules, control systems should cope with other types of abnormal situations, such as unexpected disturbances originating from process fluctuations, failure of process equipment or faults in control system hardware and software. In addition, a single failure should have a minimal impact on the rest of system functionality.</td>
</tr>
</tbody>
</table>

Based on the requirements mentioned in Table 2.2, a typical model of future manufacturing system is shown in Figure 2.4. It is commonly called collaborative manu-
turing management (CMM) model (Leitao et al., 2005).

CMM was originated from the concepts of computer integrated manufacturing (CIM) and plant wide systems (PWS). These systems (CIM & PWS) were originally based on centralised and sequential manufacturing planning, scheduling and control mechanisms. Due to this, systems built using these concepts (CIM & PWS) were found insufficiently flexible and agile (Rathwell, 2001). Comparing with traditional manufacturing systems, they also have the same problems of huge investment, long lead times, large size rigid structure and high risks of shutting down of the whole system by the occurrences of single failure at one point.

Therefore, a decentralised approach was introduced. It is based on the architecture of having a conglomerate of distributed, autonomous, intelligent, fault tolerant, reusable production units, which operate as a set of co-operating entities (Colombo et al., 2004). Each entity is capable of dynamically interacting with others using the CMM paradigm. Through CMM, manufacturers optimise the processes for overall enterprise-wide effectiveness rather than individual plant efficiency.
Chapter 2: Review of Current Manufacturing Systems

This concept of CMM is depicted in Figure 2.4 and is built around three intersecting domains: Enterprise, Value Chain, and Lifecycle. These three intersecting domains help manufacturers in establishing the relationships among plant and enterprise applications, markets, value chains and manufacturing nodes. The spheres represent the manufacturing units which are connected by material, information and process flows to yield a collaborative manufacturing environment. Above the central disc are business functions; while the production functions are below it.

This CMM model is used to derive a collaborative industrial automation system paradigm as shown in Figure 2.5. This paradigm is the result of the integration of emerging technologies like smart agent based control technology, holonic control systems and mechatronics. This model will help the information to flow quickly and accurately in both directions in the enterprise domain, i.e. between business and automation systems.

These collaborative automation systems depend on suitable embedded control software, and a reliable control and automation architecture. In particular, its reliability, agility, and degree of flexibility are significantly dependent on the reliability and flexibility of the embedded control and automation architecture. The key advantage of a production system managed and controlled by such a collaborative automation system
Chapter 2: Review of Current Manufacturing Systems

is its dynamic reconfigurability\(^\text{1}\). Because of its advantages, collaborative automation system is considered as the most suitable solution to yield future agile manufacturing systems.

The following section discusses the concept of agile manufacturing mentioned above and the collaborative automation based existing agile manufacturing approaches.

2.4 Overview of agile manufacturing

Agile manufacturing was initially introduced in early 1990s. Since then, it has been associated with many definitions. For details, see Gunasekaran (1998); Kidd (1995); Yusuf \textit{et al.} (1999) and Elkins \textit{et al.} (2004). However, it can be broadly defined as

\begin{quote}
\textit{a business and an operational strategy aiming to accelerate and enhance flexibility in a customer’s oriented make/configure-to-order production process with minimal changeover time and interruptions.}
\end{quote}

In addition, it also includes defining processes, tools and training required to quickly respond to customer needs and market changes while still controlling costs and quality. The key features of agile manufacturing include low volume/high mix product order, speed, flexible production processes with the environment where customised, configurable, or specialised products can be manufactured. According to Lee \textit{et al.} (2004a), to physically implement this concept and 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. More specifically, manufacturers have to integrate the design and production information with their business partners.

In future automation systems, agility is required in many areas throughout the manufacturing lifecycle e.g. business applications, product design, manufacturing and reconfiguration production processes. The general framework of agile manufacturing system is presented in Figure 2.6. This is a wide area of research; therefore, the research reported in this thesis is only focused on the production and the systems integration

\(^{1}\)The term dynamic reconfigurability refers to the ability to attain efficiency and versatility by producing a wide range of different product families and/or different types of a product with a minimal effort in changing the involved manufacturing environment (Leitao \textit{et al.}, 2005)
facets of agility. More specifically, it is focused on the intra-enterprise domain of the system (Figure 2.6), i.e. to develop flexible and reconfigurable production lines and connect to higher level business applications. In the Intra-Enterprise domain (Figure 2.6), business applications, manufacturing execution systems and other control applications are integrated into manufacturing system through middleware and various interfaces. The term middleware refers to the software applications specifically designed to integrate disparate software applications in heterogeneous environments (Xiong et al., 2001). The network communication is established using LANs at business level while at shop floor level; gateways are used to interface with different types of shop floor networks (e.g. Fieldbus and Industrial LAN).

Figure 2.6: Architecture of intra-enterprise domain of the agile manufacturing system shown in Figure 2.4.
Chapter 2: Review of Current Manufacturing Systems

2.4.1 Related approaches toward agile manufacturing systems

To physically implement the concept of agile manufacturing, several research groups, industries and standard bodies have attempted to create solutions. Some of the key technologies, projects & consortiums in the scope of research presented in this thesis are reported in Table 2.3.

Table 2.3: Key technologies, projects & consortiums focusing on agile manufacturing systems.

<table>
<thead>
<tr>
<th>Technology/Project/Consortium</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Radically Innovative Mechatronics and Advanced Control Systems (RIMACS)</strong></td>
<td>RIMACS was an European FP6 project initiated in 2005, with the aim to define a radically innovative manufacturing control open architecture based on the state-of-the-art information, wireless communication technologies and modular mechatronics. Its details are presented in Cucinotta et al. (2009), where it was claimed to achieve partial success in making manufacturing plants agile and reconfigurable, by using multi-agent systems, open platforms, wireless technology and virtual engineering design methods.</td>
</tr>
<tr>
<td><strong>Object Linking and Embedding (OLE) for Process Control (OPC)</strong></td>
<td>OPC, developed by OPC Foundation, specified a set of standards based on the OLE, COM and DCOM technologies developed by Microsoft for its Windows operating system family. More specifically, the specification defined a standard set of objects, interfaces and methods for use in process control and manufacturing automation applications to facilitate interoperability. Its details are presented in OPC (2003) &amp; Mahnke et al. (2009).</td>
</tr>
<tr>
<td><strong>Object Linking and Embedding for Process Control - Unified Architecture (OPC UA)</strong></td>
<td>OPC UA is the enhanced version of OPC. It focuses on services based framework, as the originally used Microsoft’s COM and DCOM technologies are now officially legacy technologies. Therefore, it focuses on using Web Services for data transport and communication with shop-floor devices (Hadlich, 2006).</td>
</tr>
<tr>
<td><strong>Open System Architecture for Controls within Automation Systems (OSACA)</strong></td>
<td>OSACA consortium employs open control architectures using a client/server protocol. It uses an object-oriented approach for developing communication system required for supporting open data exchange between software modules within control systems. OSACA enables flexibility and supports interoperability between different vendor solutions, by defining the control application as an object (i.e. a device); as mentioned in Kosmopoulos (2007).</td>
</tr>
<tr>
<td><strong>Service Infrastructure for Real Time Embedded Networked Applications (SIRENA)</strong></td>
<td>This project focused on defining unique domain overlapping and platform independent architecture for distributed embedded systems, based on service-oriented architecture. For this purpose, an XML-based Web Services paradigm was applied for interconnecting heterogeneous applications through Ethernet TCP/IP. Primarily, it focused on providing more general service definitions for embedded devices and applications (Bohn et al., 2006; Jammes &amp; Smit, 2005).</td>
</tr>
</tbody>
</table>
## Chapter 2: Review of Current Manufacturing Systems

Table 2.3 – Continued from previous page

<table>
<thead>
<tr>
<th>Technology/Project/Consortium</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSI Research Institute Loughborough University, UK</strong></td>
<td>This research group has focused mainly on the lifecycle support of distributed automation systems by replacing centralised PLC controllers with distributed control nodes (LonWork controllers) and using component-based design approach, where the control functionality is embedded into the component modules. Its details are presented in Lee <em>et al.</em> (2004a), where the work has been claimed to contribute improving flexibility, reusability and ease of use in the control domain. The COMPAG project has also been claimed to contribute to next generation distributed automation systems in improving performance via improvements in visualisation, remote support, diagnosis and HMI’s.</td>
</tr>
<tr>
<td><strong>Tampere University, Finland</strong></td>
<td>The research group at Tampere University, has contributed to the development of agent-based distributed automation, remote configuration, and wireless communication in control systems mainly within the electronics-manufacturing domain (Villasenor <em>et al.</em>, 2009). 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.</td>
</tr>
<tr>
<td><strong>Siemens Totally Integrated Automation (TIA)</strong></td>
<td>Totally Integrated Automation (TIA) is an open system architecture developed by Siemens. The framework of TIA has been claimed to combine all the automation software tools for controllers, HMI and drives in a single development environment. This is reported in Siemens (2011); in which Siemens mentioned that the unique system uniformity of TIA ensures interaction of all components that benefits machine and plant builders, system integrators and end customers with optimized processes reducing the total cost of ownership, shorten the time to market and improve quality.</td>
</tr>
<tr>
<td><strong>Service-Oriented Device and Delivery (SODA)</strong></td>
<td>The SODA project focus on building a service-oriented ecosystem that can be used throughout a system’s life cycle. It incorporates ITEA SIRENA framework for high-level device communications that exploits service-oriented architecture. This application of SOA paradigm for communicating and interworking between embedded software components at the device level is claimed to offer many advantages. For example, it offers a platform, language and network-neutral way, applicable to a wide variety of networked devices for diverse applications in domains such as industrial automation (Bobek <em>et al.</em>, 2008).</td>
</tr>
<tr>
<td><strong>Service-Oriented Cross-layer Infrastructure for Distributed Embedded Systems (SOCRADES)</strong></td>
<td>SOCRADES project focused on implementing, testing and piloting prototypes of DPWS-enabled devices in the industrial automation domain. The primary objective of the project was to develop design, execution and management platform for next-generation industrial automation systems by exploiting the Service Oriented Architecture paradigm both at the device level and at the application level, as reported in Karnouskos <em>et al.</em> (2007).</td>
</tr>
</tbody>
</table>
 Chapter 2: Review of Current Manufacturing Systems

To investigate the limitations of technologies reported in Table 2.3, these are broadly classified into two groups depending on their functional operation. The first group comprises of RIMACS, OPC and OSACA; whereas, second group consists of SOA based technologies such as SODA and SOCRADES. These are discussed below.

- **RIMACS**: RI-MACS is the acronym for Radically Innovative Mechatronics and Advanced Control Systems. RIMACS was initiated in 2005, to define, develop and introduce the radically innovative manufacturing technologies (Abadie & Neubert, 2006). RIMACS used open architecture standards, collaborative automation paradigm, modular mechatronics and virtual engineering environments. These are used with the aim to yield a flexible and agile manufacturing system.

![Figure 2.7: Architecture of RIMACS technology for integrating higher level business applications to shop floor applications (Cucinotta et al., 2009).](image)

Its principle is based on considering the set of production entities as a conglom-rate of distributed, autonomous, intelligent and reusable units. These production entities are usually called Intelligent Autonomous Mechatronic Components (IAMC). IAMC comprises of three units: The Embedded Components (TEC), the Embedded Machines (TEM) and the Embedded Production Systems (TEPS). TEC refers to the control devices and mechanical parts. TEM refers to the machine subsystems consisting of independent sub-components and computational units. TEPS refers to the overall machine systems or production lines to represent abstract functionality of physical components. TEC units are or can be part of TEM, which further are or can be part of TEPS. The final automation system is realised by collaborating these intelligent, distributed production units.
In RIMACS technology, the higher level business applications are integrated to shop floor applications through the RIMACS interface and middleware as shown in Figure 2.7. In Figure 2.7, the RIMACS interface stack comprises of various interfaces such as Web Services, Modbus and TCP/IP, in order to allow interoperability of different applications (Cucinotta et al., 2009). The key feature of RI-MACS approach was extending the use of networks through these interfaces for the real-time production control and automation. This helps enabling a new range of products and services.

- **OPC Foundation**: The OPC foundation stands for Object Linking and Embedding (OLE) for process control. It specified a set of standards in 1996. These standards are based on OLE, Common Object Model (COM) and DCOM technologies developed by Microsoft (Mahnke & Leitner, 2009). The specification also defined a standard set of objects, interfaces and methods for use in process control and manufacturing automation applications. The architecture of OPC technology is shown in Figure 2.8; which depicts the method of accessing data from a source and communicate it to any client application. The method remains the same regardless of type and source of data. Its middleware provides a set of software drivers suitable for different components.

The purpose of OPC technology is to define a common interface or driver that is written once and then reused by any business, SCADA, human machine inter-
face, or custom software packages. Once an OPC server is written for a particular device, it can be reused by any application that is able to act as an OPC client (Mahnke et al., 2009). However for designing automation systems from the life-cycle perspective, there is no well structured approach from OPC to support flexibility and re-configurability. This depends on the capabilities of vendors to implement this functionality within the OPC framework.

- **OSACA:** OSACA stands for the Open System Architecture for Controls within Automation systems (Kosmopoulos, 2007). This consortium is working on open control architectures (Figure 2.9) based on a client/server protocol. It uses an object-oriented approach for developing communication system required for supporting open data exchange between software modules within control systems.

![Figure 2.9: Architecture of OSACA technology for integrating higher level business applications to shop floor applications (Kosmopoulos, 2007).](image-url)

To enable flexibility and support interoperability between different vendor solutions, the OSACA architecture defines the control application as an object (i.e. a device) (Figure 2.9). The object encapsulates the logic control, motion control and process control and its generic functionality is exposed and accessed through the object interface (API). These APIs act as an interface between the application
object and underlying infrastructure in the form of communication, operating system and electrical components. This gives vendors of control software the freedom to implement the software in their own fashion. However, the solutions need to be complied with OSACA interface standard in order to have interoperability. The business system acts as client application and the shop floor level applications as servers and their integration is done through the OSACA middleware. More specifically, the client finds and invokes the services through COM, Application Services System (ASS) and Microsoft Transaction Server (MTS) middleware layers. The physical communication is established through LAN and/or gateway if different networks and protocols are used on the shop floor.

From the above discussion, it can be interpreted that OPC, OSACA and RIMACS technologies offer many benefits; however, none of these are able to fully meet the requirements of future automation systems. Their key limitations are mentioned below:

- **Complexity:** The above mentioned technologies are very complex to implement in practise. It is because these technologies use middleware and APIs to integrate various software applications, especially when running on different operating systems. To integrate new applications into systems, a new set of drivers and interfaces is required for each application. This increases the complexity of system, especially with increase in number of applications. In addition, different types of networks are generally used at shop floor such as Fieldbus, Modbus and Industrial Ethernet. These networks are usually different from the top business level, which leads to the requirements of additional hardware such as gateways; which increases the complexity of system.

- **Vendor specific solutions:** These technologies use different middleware vendors; thus, interoperability among these middlewares is another concern for the development of an open platform. Therefore, the integration of business and shop floor applications is limited to a number of vendor specific middleware solutions.

- **Rigid structure:** In the above mentioned technologies, the co-operating devices are interconnected in object-oriented design patterns. System configuration and management is provided as a building block. These system configurations have tight coupling in terms of communication and object binding within client
and server applications. However, the future automation systems require loose coupling of components, in order to have high flexibility.

From the above discussion, it can be interpreted that the RIMACS technology can only partially support the requirements of future automation systems. Therefore, new technologies (e.g. SIRENA and SOCRADES), based on service-oriented architectures came. Their overall aim is to introduce SOA and Web Services at shop-floor device level in order to facilitate integration with higher level business applications. There are several technologies and projects in this domain as reported in Table 2.3. Among these, the ITEA-SIRENA and SOCRADES is found to claim some success; whereas, the others such as SODA are in development phase. In addition, Siemens has recently launched a new commercial technology, named TIA. It is evaluation phase and not much related literature is available; therefore, it is not discussed further in this thesis.

The ITEA SIRENA project was started in 2003 by Schneider Electric, to leverage Service-Oriented Architectures to seamlessly interconnect (embedded) devices inside and between four distinct domains - the industrial, telecommunication, automotive and home automation domain. A framework was developed to achieve this aim as well as to assure interoperability with existing devices and extensibility of networks based on SIRENA technology. This is shown in Figure 2.10. The core of the framework is the Devices Profile for Web Services (DPWS), which is discussed in detail in Chapter 6. The first DPWS stack, worldwide, for embedded devices was developed by the SIRENA consortium using an open source gSOAP package and C/JAVA (Jamous & Smit, 2005).

![Figure 2.10: Framework of SIRENA technology.](image_url)
Chapter 2: Review of Current Manufacturing Systems

The DPWS stack is designed in such a way that any kind of transport protocol could be used; however, IPv4 and IPv6 still form the underlying protocols in the SIRENA project because of their wide acceptance. The SIRENA framework defines the basic service oriented technology used for device integration and interaction. All SIRENA enabled devices comply with the basic framework. Devices and services from other SOAs may be attached to the SIRENA framework by using the SIRENA framework extension interface, shown in Figure 2.10. The key benefit of using the DPWS technology is that it provides an open standard with plug and play connectivity, security, technology neutral platform regarding programming language, operating system and network media. However, the project lacks to provide real-time functionality using a real-time IP stack that manages Quality of Service (QoS) aspects in the SIRENA basic framework (Bohn et al., 2006).

SODA and its two sub-projects SOA4D and WS4D are other examples of similar work with SOA, in development with ITEA. These projects are focusing on enhancing SIRENA’s features such as ease of the development, integration, deployment, maintenance and management of devices and services in a SIRENA based network.

The other project SOCRADES also focused on employing SOA and WS, for implementing, testing and piloting prototypes of DPWS-enabled devices in the industrial automation domain. The SOCRADES project extended the SIRENA project framework by applying SOA using Web Services technologies to facilitate the adoption of a unifying technology for all levels of the enterprise (Colombo et al., 2009; Phaithoonbuathong et al., 2010). The project had created a middleware technology to provide common communication standards (HTTP, XML, SOAP), thus enabling homogeneous linkage from control levels to enterprise business processes. The project implemented modular CB manufacturing approach using SOA-WS technology based on a central orchestrator, to provide an open standard for non-vendor specific control systems as well as enabling seamless enterprise integration. Looking into the advantages of using this SOA and WS based approach; the same is used in the research reported in this thesis. This is discussed in detail below.

2.4.2 SOA and Web Services as proposed solution

It has been claimed by researchers such as Colombo et al. (2005); Lee et al. (2004a) and Kirkham et al. (2008a) that service-oriented architectures can improve interoperability
Chapter 2: Review of Current Manufacturing Systems

among a wide array of applications in business and manufacturing systems. SOA has already been proven in business-to-business integration by successfully overcoming the problems related to increasing complexity of inter-connecting applications in the heterogeneous software environment. This success of SOA’s in business-to-business integration can be extended into manufacturing systems, i.e., business-to-shop floor integration. This can be done through the combination of SOA middleware and Web Services interfaces as shown in Figure 2.11.

By definition, SOA describes information technology (IT) infrastructure which allows different applications to exchange data with one another, as they participate in business processes (Marks & Bell, 2006). More specifically, SOA is an approach to the design, implementation and deployment of information systems in such a way that the complete system is created from individual components. In this way it refines the discrete and complex business functions. In SOA technology, the term components is replaced by the term ‘services’. The services communicate with each other either by passing data from one service to another, or by coordinating an activity between two or more services. SOA allows loose coupling of services, distributed computing and modular programming. SOA distributes services across geographical boundaries, enterprises, and can reconfigure these into new business processes as needed. This helps to make the applications accessible both inside and outside the company, thus providing greater flexibility and uniformity.

Figure 2.11: SOA and Web Services as a solution to future manufacturing system.
Chapter 2: Review of Current Manufacturing Systems

For integrating business to shop floor applications (Figure 2.11), SOA uses Web Services for communication. It is because Web Services are becoming a common standard network for industry and its combination with SOA can be used to form a homogenous platform to ease the enterprise integration as depicted in Figure 2.11. In addition, the flexibility and agility will be enhanced by the implementation of Web Services on distributed control devices. This will give loose coupling between services, which enables the various local run time and design time applications to seamlessly integrate through a single set of standard Web Services interfaces.

To enhance the reconfigurability of control system, a novel framework of Web Services, SOA and Component-Based design approaches have been proposed and utilised in the research reported in this thesis. These are discussed in detail in following chapters. In this approach each mechanical part has 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. This will help engineers and designers to easily design and integrate the components to yield the complete control system.

The benefits of adopting the proposed SOA and Web Services based approach over the existing approaches (OPC, RIMACS and OSACA) is tabulated in Table 2.4. The data provided in Table 2.4 have been collected from the literature and research publications such as Colombo et al. (2005); Kirkham et al. (2008a); Lee et al. (2004a); Marks & Bell (2006) and Jammes et al. (2007). It can be interpreted from Table 2.4 that in some areas the existing solutions do not support agility. This is because the middleware of existing technologies is generally composed of different sets of drivers for each different application. In addition, various 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. However, the proposed WS-SOA based automation system supports every aspect of agile system requirements as detailed in Table 2.4. Therefore, the WS-SOA approach allows the creation of an overall homogenous platform. This is discussed in detail in further chapters of this thesis.
## Chapter 2: Review of Current Manufacturing Systems

Table 2.4: Comparison of key manufacturing automation systems technologies.

<table>
<thead>
<tr>
<th>Agility Requirements</th>
<th>OPC</th>
<th>OSACA</th>
<th>RIMACS</th>
<th>WS-SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non vendor specific control platform to increase modularity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reconfigurability</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flexibility</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Visualisation and simulation</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Component based reusable design</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Seamless network integration</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Well Integrated information sharing</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Information retrieval and utilisation tools (HMI system)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Common database and diagnostic system</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 2.5 Summary

This chapter presented a discussion on key manufacturing technologies. From the discussion, it is concluded that collaborative mass customisation offers maximum degree of flexibility and thus offers the promise to meet the requirements of future agile manufacturing systems. A discussion on traditional manufacturing systems is also presented and their shortcomings in meeting the requirements of today’s and future manufacturing systems are highlighted. As a result, agile manufacturing has emerged as the favoured manufacturing paradigm. The state of the art for agile manufacturing and the key enabling technologies has been reviewed. From the review, each considered approach is found to provide only partial agility features and has limitations in realising a complete agile manufacturing paradigm. A common problem is the poor level of interoperability between vendor specific technologies. In order to improve interoperability and also increase flexibility, a new design of the control and manufacturing platform using standard technology of SOA and Web Services is proposed. SOA and Web Services have already been proven successfully in business-to-business integration. However, their role in integrating business to shop floor manufacturing system, especially the real-time distributed control automation systems, has not been defined in detail as yet. Therefore, the research outlined in this thesis is focused on the application of SOA and Web Services at the shop floor level and its integration to business level so that it enables the achievement of agility in the complete manufacturing system and supply chain.
Chapter 3

Current Manufacturing System Technologies

This chapter presents the second part of Literature review. It presents a discussion on technologies required for implementing collaborative automation system, mentioned in Chapter 2. More specifically, it presents a discussion on available technologies for the implementation of distributed control system, their logic applications, and middleware. A comparison of these technologies is done and a selection is made to support the development of agile manufacturing systems.

3.1 Distributed automation systems

As discussed in literature review presented in Chapter 2, the future manufacturing paradigms demand for an open, agile and flexible distributed automation system. The structure of such an automation system, including various implementing technologies is presented in Figure 3.1. There are many research projects (undertaken and/or in progress) focusing on these technologies to yield the open and distributed manufacturing automation systems. These technologies will be discussed and compared in the following sections.
Chapter 3: Current Manufacturing System Technologies

Figure 3.1: The typical structure of a distributed automation system (after Castellote (2005)).

3.2 Control system

In Figure 3.1, the element *hardware* mainly refers to processors (such as Intel, Motorola Neuron and ARM microcontrollers) of control systems. These control systems are used to send commands to physical devices (such as actuators) to perform specific manufacturing tasks. These commands depend upon their software based control logic and the input data from physical devices such as sensors. The control systems are generally application-specific devices, so these are associated with many definitions depending on their applications. However with regard to automation systems in manufacturing industry, a control system can be broadly defined as

> a device or set of device(s), which can be used to manage, command, direct or regulate the behavior of other devices or systems (Lee, 2004).

A typical example of control system is an automatic sequential control system, which triggers a series of electric and/or pneumatic transducers and/or mechanical
actuators in correct sequence to perform a task, say, moving a conveyor belt. Generally, the control systems can be divided into two classes: logic or sequential controls, and feedback or linear controls (Patrick & Fardo, 2009). In manufacturing industries, automation systems mostly comprise of logic control systems. In the very beginning, these logic control systems were implemented for industrial and commercial machinery using interconnected relays operating at mains voltage and the control logic was designed using ladder logic. However over the last few decades, most of the control systems are constructed using programmable logic controllers (PLCs) and/or embedded microcontrollers. PLCs were introduced in early 1970’s and since then these are the most common choice for constructing control systems. PLC itself comprises of a hardware part (mostly relays/switches and a processor) and a software part which defines the control logic. In principle, PLC uses the binary logic; whereas the binary status “TRUE” and “FALSE” is used to indicate the status (e.g. “ON” and “OFF”) of switches connected to operate relays. For programming PLCs, the graphical language ladder logic is traditionally used. However, there are several drawbacks of programming PLCs using ladder logic (IEC-61131-3, 2003). Some of these include: lack of modularity, limited reusability of programming code and poor support for complex programming structures. Ladder logic also requires highly experienced programmers for the implementation and commissioning of its software. To overcome these shortcomings, IEC 61131-3 standards were introduced. The aim is to provide a common, interchangeable, structured and open framework for PLC software architectures. IEC 61131-3 comprises of five language constructs: Instruction List (IL), Structured Text (ST), Ladder Diagram (LD), Function Block Diagram (FBD) and Sequential Function Charts (SFC). Their programming details and functional capabilities are detailed in IEC-61131-3 (2003).

3.2.1 Control system architectures

The second element is architecture of control systems; which defines the ways to organise devices (e.g. PLCs) in automation systems. The control architectures can be classified into four categories as shown in Figure 3.2. These are explained below.

1. **Centralised control architecture:** This is the traditional design approach based on a single processing unit, which controls the whole system (Figure 3.2).
This unit processes all the data and/or commands, and also controls all the connected physical devices such as sensors and actuators. Because of a single controlling unit, the systems build using this architecture are often very rigid. Any changes/reconfigurations required in hardware and/or software of these systems is usually very complex. It is because this requires a skilled engineer with complete system-wide knowledge in order to make any modification(s) to hardware/software. In addition, as all the physical I/O devices are dependent on a single unit; therefore, if this unit fails, the whole system fails.

2. **Hierarchical control architecture:** In hierarchical control architecture, the workload of single unit in centralised control system is distributed to some extent in hierarchy as shown in Figure 3.2. In this architecture, each control system is referred as a node. The node in upper level is responsible for job management. However, the nodes in lower level perform the tasks specified in schedules, and any variance is sensed and processed by upper nodes. This is also referred as Master-Slave system. In comparison to centralised control architecture, it offers better stability and higher flexibility. However on the downside, the systems build using this architecture are slow in I/O response and their robustness is also limited to a single point of failure. They also have rigid structure, so it is not easy to incorporate late changes or to do any reconfigurations.
3. Modified hierarchical control architecture: The modified hierarchical control architecture was introduced to overcome the problems identified with true hierarchical architecture. In this architecture (Figure 3.2), the processing units are located at machining stations to provide local control to physical devices such as sensors and actuators. These control units at the same level of hierarchy can communicate with one another in a collaborative manner; thus, improving flexibility and overall system performance. However, the communication is still restricted to a small level. In this architecture also, the master controller unit provides the overall control and sequence of operations. Therefore, the overall architecture is still rigid and has the drawback of a single point of failure. In this architecture also, it is difficult to do any future changes and reconfigurations.

4. Heterogeneous control architecture: The fourth and latest architecture is heterogeneous architecture. These are also called fully distributed control system. In this architecture, there is no central controller as shown in Figure 3.2. Instead, the control autonomy is assigned to local control units. These control units have encapsulated processors which allow these units to act autonomously with loose coupling with its peers. This makes the system more flexible as it no longer relies on a central controller. In this case, failure of a control node may not affect the whole system, which makes the system more robust.

Because of these advantages, heterogeneous architecture is considered as most suitable for future manufacturing systems and is thus used in research presented in this thesis.

3.3 Communication mediums

After defining the control architecture, the next step is to establish the physical communication channel between control devices, physical devices and the higher level process control devices and applications. This communication channel is commonly called industrial network (Decotignie, 2009). Historically, the industrial networks came in early 1980’s and comprised of different types of fieldbuses. However, these fieldbus based network systems have many limitations and is still not widely used. It is because of lack of fieldbus protocol standards, which can ensure complete interchangeability and interoperability between products from different suppliers. Therefore in recent years,
Web Services (e.g. Ethernet) is fast becoming the standard for establishing device communication. The details of fieldbus and Ethernet are presented below:

- **Fieldbus**: By definition, fieldbus is an industrial network system for real-time distributed control between field devices and master device such as PLC (Lewis, 1997). There are many types of fieldbus in use in existing automation systems. The specifications of some of the most common fieldbus technologies are presented in Table 3.1. These technologies support daisy-chain, star, ring, branch and tree network topologies. In principle, fieldbus technologies are based on the Open Systems Interconnection (OSI) reference model. This model describes how information from a software application in one processing unit moves through a network medium to a software application in another unit. The model was developed by the International Organisation for Standardization (ISO) in 1984, and its details are presented in Stallings (1993). It consists of seven layers: Physical layer (1), Data link layer (2), Network layer (3), Transport layer (4), Session layer (5), Presentation layer (6) and Application layer (7). The lower layers (1-4) focus on data-transport functions, and the upper layers (5-7) focus on the applications. Some fieldbus technologies use three layers: physical layer (1), data-link layer (2) and application layer (7); whereas some fieldbus technologies also incorporate network layer (4) in addition to the above. To physically implement the fieldbus networks, the fieldbus connections require only one communication point at the controller level and allow multiple analog and digital points to be connected at the same time. This reduces both the length and number of the cables required. However, fieldbus has many disadvantages. Some of these include: lack of standards, complexity to operate, higher cost to use, longer reaction times and poor interoperability between different fieldbus based devices. Therefore, device manufacturers make different versions of their devices (e.g. sensors, actuators) compatible with different fieldbus standards. This usually adds to the cost of devices and the difficulty of device selection and availability.

- **Industrial Ethernet networks**: Industrial Ethernet is an emerging technology based on using intelligent switching technology. It offers a variety of advantages compared to traditional industrial fieldbus networks. Some of these include: low
cost to establish and operate, ease of use even in critical manufacturing applications, network security and performance. Ethernet allows open system communication and interoperability among various vendor specific devices; which is why Ethernet becomes popularised in recent years (Potter, 1999). There are various types of Industrial Ethernet technologies, as mentioned in Table 3.2. Industrial Ethernet is also based on the OSI model; however, it is different from the standard Ethernet used in the computer networking. Standard Ethernet is designed around two layers of the OSI model (i.e. physical layer and data-link layer); however, Industrial Ethernet also encompasses layers 3 and 4. In principle, the layer 3 is used for Internet Protocol (IP) addressing and routing, i.e. which way to send data. Layer 4 is the transport layer; which defines the IP suite (Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)) to ensure that data is delivered error-free and in correct sequence.

To physically establish the Ethernet LAN, organizations can choose from a variety of devices and architectures. Devices range from simple hubs, to unmanaged switches, to intelligent, managed switches. In current research, switches are used. It is because switches offers a number of important advantages compared to hubs and other devices. Some of these advantages include predictable performance, low latencies (<100 microseconds, which is well below the limit and 100 times faster than most applications require), and management, diagnostics and security capabilities. Switches also avoid collisions when multiple devices communicate at the same time. Thus, it makes it possible for several users to send information over a network at the same time without slowing down each other. This is particularly beneficial for automation networks, where real-time, predictable performance is required.

The key benefit of industrial Ethernet is that it can unite a company’s administrative, control-level, and device-level networks to run over a single network infrastructure. It is not based on custom or proprietary standards; therefore, it is more interoperable with other network(s) and their equipments. This is why industrial Ethernet is used in the research presented in this thesis. More details are available in further chapters of this thesis.
<table>
<thead>
<tr>
<th>Field Characteristics</th>
<th>Bus Company</th>
<th>Actuator Sensor Interface (AS-i)</th>
<th>Control Area Network (CAN)</th>
<th>DeviceNet</th>
<th>Interbus-S</th>
<th>Profibus DP/PA</th>
<th>LonWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>AS-I Consortium</td>
<td>Bosch</td>
<td>Allen-Brandley</td>
<td>Phoenix Contact</td>
<td>Siemens</td>
<td>Echelon Corp.</td>
<td></td>
</tr>
<tr>
<td>Physical Medium</td>
<td>Twisted-pair or fiber</td>
<td>Twisted pair</td>
<td>4 conductor cables</td>
<td>Twisted-pair, fiber and slip-ring</td>
<td>Twisted-pair &amp; fiber</td>
<td>Twisted-pair, fiber, power line</td>
<td></td>
</tr>
<tr>
<td>Max. Cable Length</td>
<td>100m 300 with repeater</td>
<td>25-1000m (baud-rate dependent)</td>
<td>100-500m 5Km with repeaters</td>
<td>400m/segment 12.8 Km total</td>
<td>100m between segments@12Mbaud 24Km (fiber)</td>
<td>2000m@78kbps</td>
<td></td>
</tr>
<tr>
<td>Communication Methods</td>
<td>Master/Slave with cyclic polling</td>
<td>Master/Slave, peer-to-peer, multi-cast, multi-master</td>
<td>Master/Slave, Multimaster, peer-to-peer</td>
<td>Master/Slave with total frame transfer</td>
<td>Master/slave, peer-to-peer</td>
<td>Master/slave, peer-to-peer</td>
<td></td>
</tr>
<tr>
<td>Network Topology</td>
<td>Bus, ring, tree star</td>
<td>Trunkline/Dropline</td>
<td>Trunkline/dropline with branching</td>
<td>Segment with “T” drops</td>
<td>Line, star &amp; ring</td>
<td>Bus, ring, loop, star</td>
<td></td>
</tr>
<tr>
<td>Media Access Algorithm</td>
<td>Cyclic polling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max. Data Size</td>
<td>8 bits with 4 inputs, 4 outputs</td>
<td>8-byte variable message</td>
<td>8-byte</td>
<td>1-64 Bytes data 246 Bytes Parameter 512 bytes h.s. Unlimited block</td>
<td>0-214 bytes</td>
<td>228 bytes</td>
<td></td>
</tr>
<tr>
<td>Max. Devices</td>
<td>31 slaves</td>
<td>127 nodes</td>
<td>64 devices</td>
<td>256 nodes</td>
<td>127 nodes + 3 masters</td>
<td>32000/domain</td>
<td></td>
</tr>
<tr>
<td>Diagnostics</td>
<td>Slave fault, Device fault</td>
<td>Error control &amp; emergency messages</td>
<td>Bus monitoring</td>
<td>Segment of CRC error and cable break</td>
<td>Station, module &amp; channel diagnostics</td>
<td>Database of CRC errors and device errors</td>
<td></td>
</tr>
</tbody>
</table>
## Table 3.2: Key Industrial Ethernet technologies for designing automation systems.

<table>
<thead>
<tr>
<th>Ethernet Network Characteristics</th>
<th>Modbus-TCP</th>
<th>Ethernet Powerlink</th>
<th>EtherCAT</th>
<th>EtherNet/IP (CIP)</th>
<th>Profinet I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Modicon</td>
<td>Bernecker + Rainer</td>
<td>Beckhoff</td>
<td>Rockwell Automation</td>
<td>Siemens and PROFIBUS user Organisation</td>
</tr>
<tr>
<td>Physical Medium</td>
<td>IEEE 802.3 Copper Cable, Fibre Optics and Wireless standard</td>
<td>IEEE 802.3, 100BaseX</td>
<td>IEEE 802.3, 100BaseTX, Twisted Pair</td>
<td>IEEE 802.3, Twisted Pair or Fibre Optics</td>
<td>IEEE 802.3 Switched Ethernet, Shielded Twisted Pair Cables or Fibre Optics</td>
</tr>
<tr>
<td>Governing Standards</td>
<td>IEC 61158 and IEC 61784</td>
<td>EC 61158, EPSG (Ethernet Powerlink Standardization Group)</td>
<td>IEC 61158, IEC/PAS 62407, IEC 61784-3, ISO 15745-4</td>
<td>IEC 61158 and IEC 61784; ODVA Ethernet/IP standard</td>
<td>IEC 61158 and IEC 61784</td>
</tr>
<tr>
<td>Network Communication speed</td>
<td>Ethernet with 10/100 Mbit/s to 1Gbit/s</td>
<td>Ethernet 100Mbit/s</td>
<td>Ethernet 100Mbit/s</td>
<td>Switches, router and wireless, from 100 Mbit/s up to 1 Gbit/s</td>
<td>Switches, router and wireless, from 100 Mbit/s up to 1 Gbit/s</td>
</tr>
<tr>
<td>Protocol</td>
<td>TCP/IP</td>
<td>TCP/IP, UDP/IP</td>
<td>EtherCAT, EtherCAT/UDP/IP</td>
<td>TCP/IP; UDP/IP</td>
<td>Isochronous real time protocol (IRT), Real time protocol (RT), Real time over UDP(RTU)</td>
</tr>
<tr>
<td>Network Topology</td>
<td>Star, Tree</td>
<td>Star</td>
<td>Line, Daisy Chain, Tree</td>
<td>Ring</td>
<td>Line, Branch, Tree</td>
</tr>
<tr>
<td>Media Access Algorithms</td>
<td>Polling</td>
<td>Polling and time-slicing mechanisms</td>
<td>Time-slot organisation Broadcast or multicast communication with message prioritisation</td>
<td>Time-slicing approach using special switch ASICs</td>
<td>Time-slicing approach</td>
</tr>
<tr>
<td>Communication Method</td>
<td>Message-oriented</td>
<td>Message-oriented</td>
<td>One frame for all communication partners (one frame per node per cycle)</td>
<td>Message-oriented</td>
<td>Message-oriented</td>
</tr>
<tr>
<td>Communication Architecture</td>
<td>Client/Server</td>
<td>Master/Slave</td>
<td>Master/Slave</td>
<td>Master/Slave and distributed control architecture</td>
<td>Master/Slave</td>
</tr>
</tbody>
</table>
Chapter 3: Current Manufacturing System Technologies

3.4 Communication modes

The fourth most important element is to determine the mode of interaction between the autonomous control devices in automation systems. This is particularly important for complex distributed systems; where a large number of devices usually exist. Several types of technologies have emerged to meet this need; however, the three key communication technologies are: Point-to-Point, Client-Server and Publish-Subscribe. These are described below and compared in Table 3.3.

- **Point-to-Point:** A point-to-point connection refers to a communication link restricted to two systems or processes (Issarny et al., 2007). This is also called peer-to-peer communication mode. In this communication model, the system/device at either side of link is responsible for any formatting of data transmitted between them. More than two devices can also be connected to form a shared network. In case of shared networks, all devices listen to signals on the communicating medium from broadcasting devices. However, the data is received by only one device to which it is addressed. This is how the devices are engaged in point-to-point communications across the shared medium. This is a simple and straightforward approach that gives high-bandwidth; however, does not scale very well with many devices.

- **Client-Server:** The client-server communication model can be considered as a distributed application that partition tasks or workloads between the providers of a resource or service, called servers and service requesters, called clients (Kirkham et al., 2008b). This communication model is based on a many-to-one approach, i.e. one central server node and many client nodes (Figure 3.3). In this architecture, all the generated information is send to server for subsequent redistribution to clients. This indirect client-to-client communication is usually inefficient, particularly in real-time environments. This is because the central server usually adds an unknown delay, as the receiving client does not know when or if it has a message waiting. In this architecture, multiple-server nets are possible; but these are very difficult to set up, synchronize, manage and reconnect when failures occur. In this architecture, the server can become a bottleneck and presents a single point of failure.
Chapter 3: Current Manufacturing System Technologies

Figure 3.3: Schematic of client-server interactions.

- **Publish-Subscribe**: Publish-Subscribe architectures support one-to-many, many-to-one and many-to-many data distribution. In Publish-Subscribe model, the nodes simply subscribes the data they need; however, they publishes all the information they create (Mastouri & Hasnaoui, 2007). In this approach, senders (publishers) of messages do not program the messages to be sent directly to specific receivers (subscribers). Published messages are characterized into classes, without knowledge of what (if any), subscribers there may be. Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what (if any) publishers are there. Publish-Subscribe systems are beneficial in distributing large quantities of time-critical information quickly even in the presence of unreliable delivery mechanisms. In summary, publishers simply send data anonymously; which subscribers receive without needing any knowledge of the number or network location of publisher.

The comparison of above mentioned point-to-point, client-server and publish-subscribe technologies is done in Table 3.3. From Table 3.3 and above description, it can be concluded that the publish-subscribe approach offers several advantages over point-to-point and client-server technologies. For example, publish-subscribe approach do not require to request traffic and data transfer directly; which makes this approach more efficient. However, the client-server architecture requires all the information, which causes significant problems for a real-time system. It is because of the trade-off between reliable delivery and delivery timing. This is why publish-subscribe approach
Chapter 3: Current Manufacturing System Technologies

is considered as most suitable communication mode for the research presented in this thesis.

Table 3.3: A comparison of peer-to-peer, client-server and publish-subscribe communication technologies.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Point-to-Point</th>
<th>Client-Server</th>
<th>Publish-Subscribe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Approach</td>
<td>One-to-One</td>
<td>Many-to-One</td>
<td>One-to-One; One-to-Many; One-to-One</td>
</tr>
<tr>
<td>Middleware</td>
<td>Telephone and TCP model</td>
<td>Client-Server</td>
<td>J2EE, NET and SOA</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>High bandwidth</td>
<td>High bandwidth, when loaded with too many nodes</td>
<td>Slows down if too many nodes, high bandwidth</td>
</tr>
<tr>
<td>Scalability</td>
<td>Poor scalability</td>
<td>Works well with data centric systems</td>
<td>Best scalability due to features such as parallel operation, message caching and network-based routing</td>
</tr>
<tr>
<td>Installation cost</td>
<td>Low installation cost</td>
<td>Higher initial setup cost than peer to peer networks</td>
<td>Low installation cost</td>
</tr>
<tr>
<td>Simplicity</td>
<td>Simple straightforward model with non requirement of any additional coordination entity</td>
<td>Complex system designs usually having significantly degraded networking performance</td>
<td>Publishers are loosely coupled to subscribers, and do not even need to know of their existence</td>
</tr>
<tr>
<td>Message delay and collisions</td>
<td>Low delay in data transfers; however, high collisions rate</td>
<td>Irregular delay due to indirect data transmission</td>
<td>Can distribute large quantities of time-critical information quickly</td>
</tr>
<tr>
<td>Security and robustness</td>
<td>Peer-to-peer networks are less secure because security is handled by the individual devices, not controlled and supervised on the network as a whole</td>
<td>Single point of failure</td>
<td>Anonymous communications where publishers and subscribers do not need to know each other’s physical network address, so it offers high security</td>
</tr>
</tbody>
</table>

3.5 Middleware

As discussed before, a distributed system consists of a number of physical devices comprising of distributed software components, which based on the control logic handles the machine operations. These components are loosely coupled and capable of performing
local operations, such as data conditioning and local feedback controls, without a master controller. The components are usually connected by communication mediums, such as fieldbus or Ethernet, and applications are executed by using a number of processes in different component systems. These processes communicate and interact to achieve productive work. This is done through Middleware, which for automation systems can be defined as a software that connects various components and their applications. In general, Middleware consists of a set of services that allow multiple processes running on one or more machines to interact. This technology provides interoperability within coherent distributed architectures, which are most often used to support and simplify complex distributed applications.

Several researchers and consortiums have worked for some years on various technologies to design middlewares for distributed open control systems. For example, Lee (2003); McFarlane & Bussman (2002); Sperling & Lutz (1997) and Weston (1999). Some of these technologies include: Microsoft’s COM, Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG), XML, SOAP, Web Services and service-oriented architecture. Most of these technologies are similar in concepts and can be broadly classified into two design approaches: 1) Object-Oriented architecture (OOA) and 2) Service-Oriented architecture and Web Services, which will here after be considered in more detail.

3.5.1 Object-Oriented architecture

A distributed system consists of various autonomous and co-operative entities. The object-oriented distributed systems generally comprise of these entities; whereas, the behavior of these systems results from the collaboration of involved entities. In the Object-Oriented architecture, these entities possess the ability to change quickly and correctly without external interventions. In principle, this architecture is based on the development of control software; which involves modeling the entities involved in a cell and their interactions (Dessouky et al., 1995; Lin et al., 1994). The individual entities are responsible for their activities; however, the coordination and synchronisation between these is managed by cell manager.

The typical example includes holonic approach; which attracted many research consortiums and institutions. In the holonic approach, the entities are called holons (Wang, 2001). A holon is defined as an autonomous and cooperative building block
within a manufacturing system, which can be used for transforming, transporting, storing and/or validating information and physical objects. The major functional elements and interfaces of a holon is shown in Figure 3.4. A holon has the autonomy to create and control the execution of its own plans, and can cooperate with other holons to develop mutually acceptable plans for achieving system goals. In Figure 3.4, the holon consists of a number of individual agent units corresponding to specific functionalities in physical configuration. Physically, each holon is a dynamic system with input, processor, output and a controller. A holon exchanges information, material, or resources with other holons via its interfaces through negotiation and cooperation. It can also form a part of another holon (Leitao et al., 2005).

![Figure 3.4: Holonic elements and interfaces (Bussmann & Schild, 2001).](image-url)

The cooperation among holons is accomplished through an evolutionary self-organizing holarchy i.e. a system of holons. This holarchy which integrates the entire range of manufacturing activities, such as design and production is defined as Holonic Manufacturing System (HMS) as shown in Figure 3.5 (Christensen, 2003). The Multi Agent System (MAS) methodologies are used for modelling HMS due to similarities between the holonic and agent approaches and the availability of complete MAS methodologies. The term *agent* mentioned above is an autonomous and flexible computational system that is able to act in an environment (Bussmann et al., 2004).
Chapter 3: Current Manufacturing System Technologies

HMS deals with the overall structure of the manufacturing process and in particular with the integration of equipment, control and workers; whereas, MAS concentrate on the design of information processing in control system and its implementation. Because of the strong similarities of concepts, many researchers consider it advantageous to combine both paradigms and to use multi-agent systems as an enabling technology for the information processing in HMS.

Many researchers such as Lewis (1997); Martinez (2004) & Hirsch et al. (2006) are adopting this approach in the domain of programming object oriented software, and implement these with emerging Function block based IEC-61499 standards. However, in view of other researchers, such as Jammes & Smit (2005), the multi-agent and Holonic approach have not made significant inroads in manufacturing plants. It is due to lack of widely accepted standards and complexities of the approach. In addition, another reason is that their implementations only cover a part of the manufacturing paradigm. The other areas remain subjected to the reign of proprietary standards, methods and mechanisms; resulting in a rigid patchwork of technology islands with poor scalability.

3.5.1.1 Common Object Request Broker Architecture

It is abbreviated as CORBA and is developed by the Object Management Group (OMG, 2002). It is a vendor independent infrastructure that allows the integration of dis-
Chapter 3: Current Manufacturing System Technologies

distributed object based applications in a network. In principle, CORBA is a three-tier distributed object mechanism for the client-server objects. The first tier 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.

![Diagram of CORBA client-server interaction model](image)

To integrate multiple heterogeneous systems or applications, such as businesses and manufacturing, CORBA integration requires CORBA middleware in both ends of the connection. All applications need at least an Object Request Broker (ORB) which provides an Application Programming Interface (API) for basic remote method invocation (RMI\(^1\)). The ORB serves as an object bus that transparently handles all the 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 behalf of client as shown in Figure 3.6. The ORBs themselves communicate via an Inter ORB Protocol, which is implemented for the transport technology used. The internet Inter-ORB Protocol (IIOP) is a COBRA standard that is used to guar-

\(^1\)It is an object-to-object, two-way synchronous remote method invocation service, which involves getting a reference (stub) to a remote object, called stub's method and waiting until the method returns with related return values.
any interoperability between ORB implementations, as well as allow applications built with different vendors ORBs to communicate and share objects for distributed object invocations.

In the process of CORBA, a client invokes a method on a server through the interface and ORB as shown in Figure 3.6. The server interfaces are specified in the CORBA standard Interface Definition Language (IDL). IDL is part of 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 to communicate with another unknown programming language. This allows each programmer to write source code independently in the most appropriate language. Major benefits of CORBA include language and operating system (OS) independence, freedom from technology linked implementations, strong data-typing, high level of tunability and freedom from the details of distributed data transfers.

3.5.2 Service-Oriented Architecture

Service-Oriented architecture, abbreviated as SOA, is associated with many definitions, which varies considerably. The introduction of SOA has already been presented in Chapter 2; however, this section presents its comparison with OOA. In general, SOA can be considered as a methodology for the development and integration of system; where functionality is grouped around business processes and packaged as interoperable services (Thramboulidis et al., 2008). More specifically, SOA describes the IT infrastructure, which allows different applications to exchange data in business processes and allows loose coupling of services within operating systems, programming languages and other technologies. The term services here refer to the distinct units, which are accessible over a network and can be combined and reused in production of business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services.

SOAs build various applications using these services. To establish communication, protocols are defined which describe how one or more services can talk to each other. A business process expert then links and sequences services using a process known as
orchestration. A complementary concept to service orchestration is ‘service choreography’ (Jammes et al., 2005; Martinez, 2005; Peltz, 2003). In the process of orchestration, services are associated in a non-hierarchical arrangement by an engineer using special software tools. The process of orchestration is concerned with the workflow-oriented execution and sequencing of processes. It does not take into account the different conversation patterns required to invoke the services associated with those processes. However, choreography considers the rules that define the messages and interaction sequences required in order to execute a given process through a particular service interface. In addition to orchestration and choreography is metadata. It is used to describe not only the characteristics of these services, but also the data that drives them. The metadata is usually created using the Extensible Markup Language (XML), whereas the services are usually described by Web Services Description Language (WSDL) and communication protocols by Simple Object Access Protocol (SOAP). This is presented in detail in Chapter 6.

The overall goal of SOA is to allow fairly large amount of functionality to be combined together to form the adhoc applications which can be built almost entirely from existing software services. The larger the amount of functionality, the fewer the interface points required implementing any given set of functionality; however, very large amount of functionality may not be granular enough to be easily reused.

### 3.5.2.1 Web Services

SOA can be implemented using many technologies; however, Web Services is most widely used. It focuses on making services accessible over the standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network enabled (Tidwell, 2000). Web Services are self contained and self describing modular applications that can be published, located and invoked across the web (Harrison et al., 2006). Once a Web Service is deployed, other applications (and other Web Services) can discover and invoke the deployed services. Table 3.4 gives a comparison between traditional business-to-business applications and Web Services.
Table 3.4: A comparison of traditional business-to-business applications and Web Services.

<table>
<thead>
<tr>
<th>Traditional business-to-business applications</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralised</td>
<td>Decentralised</td>
</tr>
<tr>
<td>Contained and controlled</td>
<td>Open and unmonitored</td>
</tr>
<tr>
<td>Limited, defined user base</td>
<td>Unknown, unlimited user base</td>
</tr>
<tr>
<td>Secure (risk minimised)</td>
<td>Exposed (open to random events)</td>
</tr>
<tr>
<td>Proprietary</td>
<td>Shared</td>
</tr>
<tr>
<td>Fixed, well defined, compiled</td>
<td>Built dynamically, on-the-fly</td>
</tr>
<tr>
<td>Incremental scale based on known demand</td>
<td>Unlimited scale, based on unknown, unpredictable demand</td>
</tr>
<tr>
<td>Staged, periodic changes</td>
<td>Continuous, adhoc changes</td>
</tr>
</tbody>
</table>

In principle, Web Services use SOAP for transferring request and reply messages (on top of TCP/IP). Service interfaces are defined using WSDL and the Universal Description, Discovery and Integration (UDDI) is used for advertising and searching available services at runtime. These techniques can be used in various combinations over the public Internet or in a private network. As in recent years, TCP/IP and small embedded web servers are entering the plant floor, it is expected that web services will eventually expand to low level devices. The Devices Profile for Web Services (DPWS) gives details of the Web Services specifications applicable to these lightweight embedded devices (Jammes et al., 2007). This is presented in Chapter 6. This evolution of the device networking systems and DPWS will pave the way for cost-effective communication paradigms, down to the level of basic field devices like sensors and actuators.

The key benefits of adopting SOA and Web Services include:

- Interoperability within various vendor specific operating systems and/or programming languages, thus maximising use of resources.
- Simple to design and easy to commission and diagnose by even non-expert persons.
- Ease of installation using efficient plug-and-play connectivity.
- Reusability of the design code in different architecture levels and devices.
- High-level management interface (typically graphical) in order to facilitate configuration, monitoring, fault diagnosis and maintenance.
3.5.2.2 SOA middleware

To implement SOA, Web Services use SOAP protocol for exchanging messages. The role of SOAP, WSDL and UDDI in Web Services is explained in Figure 3.7. SOAP defines the framework for message structure and a message processing model by providing an XML-based messaging framework. SOAP also defines a set of encoding rules for serialising data and a convention for making remote procedure calls (RPC). SOAP is platform neutral and support many programming languages unlike other middleware technologies such as Distributed Computing Environment (DCE), CORBA and DCOM. SOAP also provides a flexible framework for defining higher-level application protocols. These features of SOAP offer increased interoperability in distributed, heterogeneous environments and allow the various Web Services protocols to be integrated individually and incrementally without affecting the rest of protocol stack.

![Figure 3.7: The role of Web Services in implementing SOA.](image)

As shown in Figure 3.7, the UDDI service is used to allow application developers to find services and develop code that relies on those services. UDDI registers the service descriptions using WSDL and then qualifies the registry description of a compliant service. The service provider registers its applications and links by subscribing them to the UDDI registry. However, the service requester looks up the services from providers by using WSDL file, in order to find out the location of service, the function calls and the ways to access them. The whole invocation method is done through SOAP
messages. Thus, SOAP defines a simple and extensible XML messaging framework that can be used over multiple protocols with a variety of different programming models.

### 3.5.3 Comparison of CORBA and SOA middleware

The assessment of both middleware technologies in meeting the agility features required in future manufacturing systems is presented in Table 3.5. From Table 3.5, it can be concluded that CORBA is better suited for building distributed applications in controlled environments, in which there are no restrictions on the granularity of the communications between distributed entities and deployment is more or less permanent. However, the future manufacturing system requires a loose coupling, so CORBA will not be suitable. With Web Services and SOA, it is possible to have loose coupling. With Web Services and SOA, developers can build applications in business domains or automation domains for control devices. As the main focus of research reported in this thesis is the creation of open and flexible manufacturing automation paradigms; therefore, SOA and Web services is preferred over CORBA.

Table 3.5: Assessment of CORBA and SOA in meeting the requirements of agile manufacturing systems.

<table>
<thead>
<tr>
<th>Features</th>
<th>SOA</th>
<th>CORBA</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open non-vendor specific solutions</td>
<td>SOAP and WSDL allows separate specifications of the abstract service interfaces and their bindings for each specific transport protocol; thus supports transport neutrality.</td>
<td>CORBA supports interoperability on various platforms; however, every node in the application environment would need to run the same ORB product.</td>
<td>SOA and WS allows heterogeneous applications running on different platforms to better interoperate through a consistent well-defined interface.</td>
</tr>
<tr>
<td>Loosely coupled and manageable applications</td>
<td>The self-describing XML documents and SOAP messages makes it possible to build a loosely coupled, document-style integration environment. Web Services provide a semantically rich integration environment using Orchestration of multiple business functions.</td>
<td>CORBA uses Object Request Brokering, which results in tight coupling to well-defined interfaces, a broker infrastructure and multi-language mappings. CORBA interfaces are difficult to manage as they consume excessive computing overheads.</td>
<td>CORBA needs more effort to commission than SOA, which provides consistent API's and homogeneous technology with loose coupling between existing applications.</td>
</tr>
</tbody>
</table>
### Table 3.5 – Continued from previous page

<table>
<thead>
<tr>
<th>Features</th>
<th>SOA</th>
<th>CORBA</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration of applications</td>
<td>SOA enables faster application integration using the WS standard (WSDL, UDDI and XML) and widely used SOAP protocols compatible with HTTP. It is because they support transport neutrality.</td>
<td>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. It is a mature technology that still has its use in high-volume, highly secure, object-oriented applications within an enterprise.</td>
<td>CORBA has failed on the Internet and it is not used for public integration amongst companies. CORBA is best suited to tightly coupled transactional systems requiring high security.</td>
</tr>
<tr>
<td>Interoperability within cross-platforms and cross-programming languages</td>
<td>Web Services 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.</td>
<td>CORBA assumes that all interacting entities conform to a standardized object model.</td>
<td>Both CORBA and Web Services provide interoperability across programming languages, operating systems and hardware platforms.</td>
</tr>
<tr>
<td>Integrated middleware in embedded control devices</td>
<td>SOA-WS devices have smaller memory footprints; however, slower processing speed and require more powerful processor specifications.</td>
<td>CORBA devices have faster processing speed; however, it is a heavyweight solution for many smaller embedded systems.</td>
<td>CORBA is 3 times bigger than the SOA middleware on the embedded device, and about 6 times faster.</td>
</tr>
</tbody>
</table>

### 3.6 Typical examples of distributed control systems

This section presents the typical examples of the existing technologies in use to realise a distributed control system. These technologies are fundamental elements that have been implemented in many research projects in order to enable open and distributed manufacturing systems. Many alternative distributed control solutions have been implemented both in industry and academic domain. For example ABB, Siemens, Honeywell, Rockwell Automation and Yokagava. Two example systems of particular relevance to this thesis are described below:

1. **LonWorks system using Fieldbus**: LonWorks system was developed by Echelon Co-operation, USA (ECHELON, 1999). It is an open solution for controlling distributed automation devices in home automation, industrial and transportation control systems. LonWorks support interoperability. It comprises of a

53
comprehensive set of tools, components, software and hardware, which helps to minimise development expenses and reduce the time required to bring products to market (Lee et al., 2004b). As shown in Figure 3.8, the heart of a LonWorks hardware device is a Neuron chip, an integrated circuit supporting communication protocol, three microprocessors, a multitasking operating system, and a flexible input/output system. The programming applications are written in Neuron C, based on ANSI C, and then it is compiled into binary bits ‘0’ and ‘1’ understood by the Neuron chip and loaded into the node’s memory. The Neuron chip based LonWorks devices can communicate directly to each other in a peer-to-peer fashion to monitor various intelligent devices such as sensors and control actuators, and to manage network operation and provide access to network data in a fully distributed manner. The LonWorks system consists of a generic network management tool and easily usable Graphical User Interface (GUI) for project administration, graphically visualised binding network variable browsing and adaptation to user needs by writing device specific control plug-ins.

In LonWorks, the interoperability within applications is supported by the LonWorks Network Operating System (LNS) standard, which is based on client-server architecture. LNS offers a common platform with a customised ‘front end’, which makes it possible for multiple vendors to supply interoperable tools. The LonWorks protocol is also known as the LonTalk protocol or the ANSI/EIA 709.1. It was designed to enable a reliable, peer-to-peer as well as hierarchical networking (different networks interoperates via a gateway) among control devices manufactured by different suppliers.

Thus, LonWorks offers many benefits such as increase in flexibility and interoperability. However, the system remains essentially proprietary because non Echelon devices are rare and may not comply with the system due to difference in the implemented protocol of bus systems. LonWorks has formed the basis for previous implementations of distributed component-based control systems at Loughborough university (Lee, 2004). However, a more open, vendor neutral solution is the goal of the new research presented in this thesis.

2. **Embedded modules with Ethernet:** In recent years, embedded microprocessors have significantly improved in term of size, cost, processing speed, real-time
performance and Ethernet connectivity for open and seamless integration of automation systems. Typical example include: ARMs NetSilicon microprocessors and Rabbit microprocessors, which are capable of operating at 10/100 Mbps based standard Ethernet connection and contain a broad set of industry standard peripherals such as USB, serial ports and a LCD controller. These embedded devices are designed to support various types of OS platforms such as Linux, RTOS, Win CE and Win XP. These embedded devices are potentially found suitable for open, flexible automation systems and provide ease of integration to higher control levels and ease of installation to enable agile manufacturing and reduce development time. Therefore, these are used in research presented in this thesis.

3.6.1 Distributed control models

A wide range of approaches have been utilised for the application logic modelling, specification and implementation of distributed control systems. Most notable in the context of the author’s research are:

1. **IEC-61499 standard**: IEC-61499 is an open standard for designing distributed control and automation applications (Orozco & Lastra, 2006). It comprises of
function blocks, which are used as basic building blocks to build the entire applications. There are two types of function blocks: basic function blocks and composite function blocks. A composite function block can contain other composite function blocks and/or basic function blocks (Figure 3.9). However, a basic function block contains algorithms and an execution control chart (ECC) (Hirsch et al., 2006). Each function block has event inputs and outputs as well as data inputs and outputs. In a basic function block the execution of an algorithm is triggered by the occurrence of an input event. The executed algorithm then from the input data produces new output data. When the algorithm has finished executing, an output event is generated. This output event might then be the input event to another function block. In this way, this architecture represents a light-weight component solution. It provides various features such as encapsulation, reusability, interoperability, portability, reconfiguration and a holistic view on distributed applications.

![Figure 3.9: Basic structure of IEC 61499-1 function block (after Lopez & Lastra (2006)).](image)

This model includes processes and communication networks as an environment for embedded devices, resources and applications. 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. At present, there are a number of researchers implementing IEC 61499 function blocks in the design of distributed control systems with high level programming such as Java, C/C++ and XML on fieldbus or Ethernet networks (Vyatkin & Hanisch, 2002). The implementation of function blocks has contributed to the improvement of reusability, re-configurability, and interoperability among different vendors (MEDEIA, 2008).

2. **COMPAG- Component-based design:** Loughborough University’s MSI research institute has been working on modular/component based distributed control system since mid 1990’s. A project titled Component based architecture for agile manufacture (COMPAG), undertaken at MSI research institute has developed a toolkit known as the Process Definition Environment (PDE) toolkit (Lee, 2004). It is based on the concept of component based design methodology as shown in Figure 3.10. It is under development, along with the design of other engineering tools to aid the machine commissioning and installation. A test rig is built from subsystem units which consists of device controller nodes (sensors and actuators) called components\(^1\), working under sequential, interlocking and discrete manner in the real-time environment. The communication between these components is event-driven in which the states of devices are defined as interconnected logic related to other devices. In addition, the states can be published to desired subscribed nodes as a network variable by sending/receiving event message through the output and input network variable interfaces. The behaviour of the component is represented by using finite state machines, \(F_{\text{element}}\), as a set of functions of component states, transitions and a combination of events. The finite state machine for the element is as follows:

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

where \(F_{\text{element}}\) = the output state of an element,
\(X\) = the set of all states in the finite state machine,
\(\alpha\) = the transition (e.g. Retracted to Extending),

\(^1\)The term *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).
Chapter 3: Current Manufacturing System Technologies

$E = \text{the set of events } E_{\text{input}}(E_i) \text{ and } E_{\text{output}}(E_o)$,

$f = \text{the designed control function of machine components.}$

For example, $E_o = \text{Transfer arm state AND part sensor state}$,

$E_i = \text{Local limit sensors of the transfer arm units}$,

$\alpha = \text{The transition state from Retracted to Extending}$,

$X = \text{Other associated unit states and}$,

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

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 control elements through its states and the states of other elements as specified in the function above.

![Component-Based approach used in the COMPAG project (after Lee et al. (2004b)).](image)

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.

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 can 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. 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. This in-
creases the flexibility of the system. This approach offers many benefits; thus it
is preferred over IEC-61499 in the research presented in this thesis.

Thus to realise the distributed automation system, a combination of embedded devices
associated with Ethernet networks, the SOAP architecture and Web Services, publish-
subscribe models and component-based design tools is proposed. This is expected to
provide ease of integration to higher control levels and ease of installation and will
reduce the development time and enable agile manufacturing.

3.7 Summary

In this chapter, the technologies involved for designing distributed open control sys-
tems are discussed and a selection is made. The concept of intelligent, autonomous and
cooperative holons is mentioned to make the next generation of intelligent manufact-
uring systems. However, these have certain requirements such as a mixed top-down
and bottom-up development approach. From the literature, it has also been found that
the object oriented architecture has also been intensively used with the agent based
approach and CORBA middleware in the similar manner as SOA. However, object-
oriented approach could not make significant inroads into manufacturing due to its
complexity of implementation and diverse implemented tools. SOA provide loosely
coupled service interfaces when implemented at device level. It is concluded that SOA-
based communications are of an asynchronous nature i.e. when an action is invoked
on a service, the result if any is returned to the invoking application entity without
the latter suspending its operations. Synchronous remote procedure call mechanisms
are comparatively less scalable than asynchronous communication, especially in the
presence of complex business processes where many operations with variable response
times run concurrently. From this discussion, suitable technologies for the develop-
ment of automation systems have been selected, based upon the measures associated
with the industrial requirements and the needs of agile manufacturing. It is predicted
that the development of Web Services, when combined with Ethernet networking and
suitable application modelling and implementation approaches (such as IEC 61499 and
COMPAG) at all levels of the manufacturing system (i.e. from business enterprises down to automation devices), has the potential to provide a new way of building and integrating automation systems.
Chapter 4

Research Methodology and Design

The literature review presented in Chapters 2 & 3, discussed the problems associated with the current automation systems. From this discussion and to meet the demands and challenges of future generation agile manufacturing, needs for a new automation system are identified. To fulfil this purpose, a new component based automation approach with the application of SOA and Web Services (e.g. Ethernet) is proposed. The aim is to provide higher flexibility and adaptability at lower cost, compared to traditional automation systems. However, there are several challenges in this approach, as mentioned in Chapters 2 & 3. Therefore, this chapter presents the research work to be done and the research methodology that has been employed in current thesis, to address these challenges.

4.1 A solution to future generation automation systems

The literature review presented in Chapter 2, discussed the key drivers responsible for the emerging needs of future manufacturing systems. Some of these include: globalisation, increasing competition, shifting of manufacturing trends towards mass customisation, and the frequent variations in market and customer demands. These conditions resulted in the need of a new manufacturing automation system. These new
manufacturing system(s) should have the ability to produce a variety of products subject to the market demands, in a shorter time-span compared to traditional systems. For example, it should possess the ability to meet the end users demand of significantly reducing the existing development time of around 53 weeks for the production systems for car engines (i.e. powertrain) to say 40-42 weeks (Anders, 2004). Traditional manufacturing systems are not capable to meet these conditions. They also exhibit limitations in their capability to effectively cope with production changes, throughout the development and production phases of their lifecycle. Reasons for this have been detailed in Chapter 2 and are summarised below. In particular they:

- Are based on a rigid centralised control system model.
- Possess poor re-usability and re-configurability.
- Usually incorporate experience based designs, which usually lack flexibility and adaptability to product changes.
- Lack features such as remote support and self error-diagnostics.
- Lack engineering visualisation and simulation tools, which are very useful for process simulation and virtual testing before being deployed in practise.

From the literature presented in Chapter 2, it has also been found that flexibility and adaptability at lower costs are key requirements of future manufacturing automation systems. Therefore, there is a need for next generation manufacturing systems, which can overcome the above shortcomings and support the agility concepts. The key requirements of these systems are detailed in Chapter 2 and are summarized below:

- They should possess a high degree of reuse and low degree of coupling. This will ensure their high adaptability to quickly reconfigure to meet the changing production needs.
- Their overall design should be simpler and easier to understand, develop and implement.
- The system should be developed and built from common system elements. These elements should possess the ability to be visually simulated and modelled. This
Chapter 4: Research Methodology and Design

will help to test and validate the overall virtual system prior to its installation in practice.

- The automation system should employ a distributed control architecture. This is to embed the knowledge and intelligence locally into control elements, which will improve the flexibility, reconfigurability, adaptability and reusability of the overall system.

In summary, future manufacturing paradigms should feature distributed, self-organising and co-operative entities or structures, which can ideally cope with any dynamic changes on demand. For this purpose, a design and implementation of such an intelligent system comprising distributed autonomous units is proposed in this thesis. These units will operate as sets of cooperating entities in a loosely coupled heterogenous control architecture (Dilts et al., 1991). This will provide high operational flexibility and capability to change. In this architecture, the overall aim is to incorporate intelligence into the above mentioned controlling devices (units), instead of just relying on the centralised master controller. This will lead to form intelligent actuators and sensors, which will be able to perform local operations such as local feedback control or data conditioning, without relying on centralised master controller. This will also help to attain a greater degree of modularity, flexibility and reconfigurability by distributing control autonomy to these local devices (Vyatkin & Hanisch, 2002). These intelligent devices/units will then be used as common building blocks to compose a manufacturing system, capable of meeting the requirements of agile manufacturing. Thus in summary, the functions and load of master controller(s) used in traditional centralised control systems is distributed to locally controlled intelligent devices. However, there are several challenges that need to be addressed to realise such a fully distributed approach. These include: distributing control to local devices; providing the required architecture and services in order to meet both the functional and real-time requirements of automation systems.

4.2 Research methodology

To realise the future generation automation system, discussed above in section 4.1, a component-based model is proposed as the fully distributed control approach with SOA
Chapter 4: Research Methodology and Design

as the underlying architecture based on the use of Web Services on an open Ethernet platform. Therefore, the focus of this research is to enhance the component-based design (CBD) methodology using SOA and Web-Services to yield a suitable automation platform to meet the need of future generation manufacturing paradigms such as mass customisation.

CBD originated from the manufacturing sector in the late 19th century. In the 19th century, this concept was introduced by making building parts to a predefined specification and using these standardised, interchangeable parts to assemble end products. However in the modern business context, the CBD approach typically supports system development by building on and reusing past experiences and knowledge. This benefits organisations in taking the advantages of technical advancements and market opportunities (Szyperski, 1997). This concept can help organisations to manage complexities in system design and adapt to changes rapidly at relatively small cost compared to traditional systems. This concept of using CBD in manufacturing automation systems will offer many benefits. These are reported in Figure 4.1 and summarised below:

- **Better functionality:** It will be easier and faster to compose automation systems using existing/pre-defined components. This will give rise to systems with better functionality compared to traditional systems.

- **Increased usability:** It will be easier and more effective to develop systems using standardised and common components.

- **Improved efficiency:** Every individual component will incorporate error/fault handling capabilities, which will help to easily identify and rectify errors without affecting the overall system integrity.

- **Improved maintainability:** The modular structure of CBD will allow individual components to be upgraded and replaced easily. This will aid system evolution, i.e. ability to rapidly adapt, make changes and extensions.

- **Improved reliability:** Every individual component can be individually validated and optimised prior to deployment; thereby, the overall reliability of the system can be enhanced.
• **Lower development cost:** Future development costs will be significantly reduced because systems can be built by reusing existing components. This will also reduce time-to-market as the new system using these components need not to be developed from beginning.

![Component-Based Development Diagram]

**Figure 4.1:** Key benefits of using Component-Based development.

This CBD approach has previously been researched at Manufacturing System Integration (MSI) research group of Loughborough University. This group is through the SOCRADES\(^1\), Business Driven Automation\(^2\) (BDA), COMPAG and MBODY\(^3\) projects. The research reported in this thesis was carried out as a sub-project of the work undertaken in the MSI research group. To implement this approach, the smart embedded devices will be used as components, and will be combined using SOA and Web Services to make the whole system. The evaluation and assessment of meeting agility features of this approach will be done using the experimental studies on the ‘Ford-Festo test rig’ in the MSI laboratory of Loughborough University. This test rig

---

\(^1\)SOCRADES was an European research project, with the primary objective to develop design, execution and management platform for next-generation industrial automation systems, exploiting the Service Oriented Architecture paradigm both at the device and at the application level.

\(^2\)BDA project aimed to establish new highly generic, end-user business-driven approach to automation systems development and support applicable to virtually all automation systems.

\(^3\)Modular build for distributed systems is an EU project with the primary objective to address the need for modular approach to the implementation and lifecycle support of automated machinery.
is established by Ford (co-sponsor of this project) and its specifications and functions are detailed in Chapter 7. The mechanisms on this test rig represent control problems typically associated in engine assembly and handling machines. The rig is designed in a manner that its control system is scalable to real machine applications. It has been used by Ford to demonstrate the capability of new control systems prior to their implementation and deployment on real powertrain assembly machines. It enables the true demonstration of machine sequences and steps for processing (i.e. transferring, buffering, checking position and drilling) parts. Therefore, the results of experimental studies performed on this test rig can be considered applicable to real manufacturing applications.

Thus, the system developed using this approach makes a novel contribution towards more flexible, reconfigurable and agile manufacturing systems. However, the design and selection of components, and their utilisation is a complicated task which needs to be done in a proper way to make an effective automation system. Also it should be noted that the CBD, SOA and Web Services technologies are not yet fully matured to realise the full benefit in the domain of automation and business systems, so these need further enhancement.

4.2.1 Research focus

A general overview of manufacturing system integration using SOA & Web Services is shown in Figure 4.2. It depicts the model framework implementing a SOA architecture with Web Services technologies at both the enterprise level entities and real-time control devices.

In 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. In addition, the logical component (i.e. device I/O configuration) is also encapsulated within the low level programming of the devices. This helps developers to concentrate only on the high level system functionality and interactions between embedded devices. The devices themselves can act as either a client or server, as every device has the functionality for synchronisation (i.e. publishing and subscribing to data).

From the literature presented in Chapter 2, it is noticed that there is a lack of research regarding the combination of smart embedded devices, component-based de-
design approaches, SOA and Web Services technologies. Therefore, the focus of current research is on SOA, the role and application of Web Services in automation; along with the investigation of potential of low level embedded device in supporting Web Services.

4.2.2 Research design and approach

The major steps involved in the research reported in this thesis are summarised in Figure 4.3 and are detailed below:

- Review of the traditional automation systems and their limitations in meeting the requirements of future manufacturing paradigms. This study has been reported as the literature review in Chapter 2. This review provided an overview of the research need.

- Review of the current technologies in use for designing the distributed control systems and integration frameworks using middleware technologies. This study has presented a comparison of the current technologies, including their limitations. This comparison has justified the capability of component based design approaches in meeting the requirements of future generation manufacturing paradigm.
Chapter 4: Research Methodology and Design

Review of Current Automation Technologies

Review of requirements of Agile Manufacturing & Future Generation Automation Systems

Define Needs & Research Objectives

Theory of Component Based Systems, Web Services & Service Oriented Architectures in context of Future Generation Automation Systems

Prototype Definition

Definition of Prototype System against needs

Prototype System Implementation

Theoretical Approach

Experimental Studies

Evaluation at Test Rig based on Reliability, Fault Tolerance, Reconfigurability & Flexibility, Interoperability with other devices

Findings, Conclusion & Future Work

Figure 4.3: Overview of research reported in this thesis.
• Develop a component based design methodology for implementing the concept of distribution of control to local field devices and its testing under laboratory conditions. This helps to gain new insights to analyse the feasibility and practicality of implementing such a distributed control concept in actual manufacturing automation systems.

• Develop the SOA based integration framework, which involves the adoption of Web Services at device level in automation systems. This helps to investigate the efficiency and performance of communicating embedded devices based components using Web Services within the distributed control concept mentioned above.

• Perform mapping of the above developed strategy using suitable embedded device. This helps to test and determine the performance of embedded device based control system. The performance will be evaluated in terms of the real time response and cycle time, ease of system design, application integration and changes to the control system to assess re-configurability.

• Finally, the developed component-based systems will be analysed to examine the practicality of using a component-based distributed control paradigm in future generation agile manufacturing systems.

4.3 Summary

In summary, this chapter reported the research focus and methodology used in the thesis. In particular, it is focused on implementing the component based technology using SOA, Web Services and the smart embedded devices. This approach is expected to significantly benefit manufacturing industry by enabling businesses and processing entities to become more integrated, adaptable and agile.
In Chapters 2 & 3, a literature review was presented on automation systems. From the literature, the need and requirements for the future generation agile automation systems were highlighted. To meet these requirements, a component based development concept for the future generation agile automation system with Web Services as the interaction method was proposed in Chapter 4. The aim is to divide the whole automation system into small reusable components; which individually and/or as a group can interact using Ethernet based Web Services. This chapter presents the automation components; which can be used individually or integrated with others to make an automation system, based on the paradigm developed at Loughborough University.

5.1 Component-Based development

In industrial automation systems, it is desirable to reuse well defined hardware and software sub-systems; aiming to reduce the overall cost and boost flexibility. However, presently this reuse is limited. These re-usable system elements are termed ‘components’ in this thesis and the concept of reusing components is called component-based
This approach originated from the manufacturing sector in the late 18th century (Green, 1997). In the domain of automation systems, the CBD approach has previously been researched at Manufacturing Research Institute (MSI), Loughborough University via COMPAG project (Anon, 1998). It has been found to be an attractive approach for future generation automation systems because it offers several advantages such as improved reliability, efficiency and maintainability. However at present, the design and selection of components, and their application utilization is a complicated task which needs to be done in a proper way in order to make an effective automation system. This is one of the key tasks of the research presented in this thesis and is discussed further in this chapter.

5.2 Physical representation of an automation component

In general, in a component based approach, an automation component can be defined as:

\[ A \text{ unit of composition with contractually specified interface and explicit context dependencies, that can be deployed independently and is subjected to composition by third parties (Szyperski, 1997).} \]

![Component representation](figure5_1.png)

Figure 5.1: Physical representation of an automation component.
Figures 5.1 & 5.2 shows the physical and functional representation respectively of an automation component of a component-based automation system. The automation component shown in Figure 5.1 comprises both hardware and software sub-systems, explained below.

![Functional representation of the elements of an automation component](image)

**Figure 5.2:** Functional representation of the elements of an automation component (Lee, 2004).

### 5.2.1 Hardware part of an automation component

The hardware part of an automation component is shown in Figure 5.3. It comprises a real-time embedded system often called a controller, interface electronics and the automation devices (Figures 5.2 & 5.3). The controllers are microprocessor based embedded systems; used for data processing, controlling the input/output (I/O) operations, accessing memory and interaction with external devices (Lee et al., 2004b). The characteristics of these embedded systems determine most of the properties, such
as timing and performance of the components. In Figure 5.3, the controller also has in-built memory, which stores the local data such as configuration parameters, diagnostic information and life cycle history of the component. This data can be accessed through various management tools via the network interface for various engineering, commissioning and maintenance purposes.

Figure 5.3: Hardware part of an automation component.

For communication purposes, there are two types of interfaces (Figures 5.2 & 5.3) in automation components used in current thesis. These are:

1. Physical I/O interface, which provides a communication interface with the physical devices such as sensors and actuators used to perform operational tasks in the manufacturing process.

2. Network Interfaces, used for communication with other components. For example, to exchange (publish & subscribe) the state variables of components.

The term ‘state’ can be better explained with the following example. Consider the Figure 5.4, which shows a simple switch. This switch performs two finite actions “OFF” and “ON”; which are termed ‘state’ in current thesis. The states can be represented in
many ways. In current thesis, integers are used to represent the states. For example, for the switch in Figure 5.4, the two states “OFF” and “ON” can be represented by integers 1 & 2 respectively. The transition from state 1 (OFF) to state 2 (ON) and vice-versa is governed by opening and closing the contact of switch.

![Figure 5.4: States of a simple binary switch.](image)

Finally, the interfacing electronics (Figures 5.2 & 5.3) is used to condition and translate the information (control signals) from the controller to the devices (sensors/actuators) connected to the output interface and vice versa (from the devices to the controller).

The key part of a component is the embedded system. These embedded systems can vary significantly from very small systems to very large systems; and thus have many constraints related to their properties (Crnkovic, 2001). For small embedded systems, non-functional properties, such as power and memory consumption, are as important as functional properties. For large embedded systems, the resource constraints, such as power and memory consumption, are not of primary concern as in case of the small embedded systems. In this case, the complexity and interoperability play a much more important role. It is because the complexity of large systems makes their development very expensive that reduction in the development costs is an important priority. Also, for large embedded systems the demands on reliability, robustness and availability are of prime importance. In general, most of the commonly stated requirements of embedded systems relate to non-functional characteristics and can be summarised as:

- **Real-time properties**: There are numbers of real-time properties such as response time (latency), execution time and deadline. These are related to time; therefore, any violation of time requirements can produce errors in the system functionality.
• **Dependability**: Dependability is defined as the ability of a system to deliver trusted service with minimum or no failures. A high degree of dependability is always preferred and it can be obtained by avoiding faults i.e. fault prevention, fault tolerance, fault removal and fault forecasting.

• **Resource consumption**: Most embedded systems have strong requirements for low and/or controlled consumption of different resources, such as power and memory consumption, execution time and computational power.

• **Life cycle properties**: Lifecycle of an embedded system is very important. During the lifetime of an embedded system, there can be several generations of supporting hardware and software technologies. This means the long life embedded systems should possess the ability to accommodate these changes introduced either into the surrounding environment or into the system(s) themselves.

Thus, a careful selection of the hardware components is required to meet these requirements. This is discussed further in Chapter 7, where a selection of commercial hardware component is made.

### 5.2.2 Software part of an automation component

The software part is an executable unit which can be deployed and composed in runtime. It provides the service perspective as well as the functional modularity aspect of a component. Its functional representation is shown in Figure 5.5. In Figure 5.5, the software part consists of logical software objects, called control elements, to represent and implement the logical control behaviour of a component. In practise, a component can contain one or more control elements depending upon its application in an automation system. The behaviour of each control element is governed by a distributed finite state machine\(^1\); comprising a finite number of states and their associated state transitional conditions. Each control element publishes its states, and monitors the states of other elements through the network interface. In this way, it establishes communication with others.

The communication channel is established by logically connecting the output network interface(s) of one component to the input network interface(s) of other components. This enables the components to publish (send) and subscribe (receive) event

---

\(^{1}\)See section 5.3 for details.
updates or other network messages with other components. The overall automation system application can be developed by logically connecting (interlocking) the control elements of various components and configuring the operational characteristics of each component using their configuration parameters. Thus in CB approach, each control element possesses the capability to execute autonomously by observing the states of other components in an automation system and can decide when it should invoke operations based on pre-configured interlock parameters.

As can be seen in Figure 5.5, a typical software based control element consists of three interrelated constituents: control abstraction, control parameterisation and control implementation; explained below:

- **Control abstraction**: Control abstraction describes the behaviour and functionality of a control element in terms of ‘states’ using the state transition diagrams (Lee, 2004). Control abstraction does not deal with the operational details of a control element. As shown in Figure 5.5, any change from one state to another is governed by specific conditions. The term ‘conditions’ refers to the logical conditions used for transition from any particular state to another. More specifically, these are the various combinations of logical AND, OR and/or NOT operators. These can be further classified into internal and external conditions. Internal conditions relate to local events from the control input (e.g. local feedback control signals from switches) or internal software events (e.g. expiry of a software timer). External conditions refers using the states of other control elements. This usage of states of other control elements is called ‘interlocking’ of states in this thesis. Actions are associated with relevant states. Actions can be control outputs (e.g. actuating an automation device), or internal software functions (e.g. starting an internal software timer or performing a computation).

- **Control parameterisation**: Control parameterisation refers to the control parameters used to express the operational details of a component. The two most important control parameters are:

  1. **Device specific operating parameters**: These parameters are used to represent the general behaviour of each control element as finite state machines and its specific operational properties as a set of values. These parameters can be changed to modify the performance characteristics of the
Figure 5.5: Functional representation of the software part of an automation component.
control element to suite particular system requirements. This will not affect the control abstraction.

2. **Lifecycle Information**: Lifecycle information is required for the maintenance, up-gradation and recycling purposes (Harrison *et al.*, 2004, 2006; Klausner *et al.*, 1998; Simon *et al.*, 2001). This is usually provided by the component manufacturer. However, the component-based approach can directly support lifecycle management through the encapsulation of product information associated with each control element. It will be advantageous as the CB approach will periodically provide this information instead of physically looking into manufacturers data sheet at various intervals (say every year) during lifecycle (say 10 years) of component. This will significantly facilitate marketing, reliability analysis, maintenance as well as recycling.

- **Control implementation**: Control implementation is used to encapsulate the specification and execution conditions of each control element. Encapsulation is the process of hiding the implementation code that drives each control element. It is advantageous as the users do not need to understand the implementation, as the functionality of the control element is accessible through the control abstraction and control parameterisation, explained above (Crnkovic & Larsson, 2002).

In general, component-based development has been an attractive approach in the domain of automation systems by many researchers such as Harrison & West (2000a); Harrison *et al.* (2003); Lee (2004); Phaithoonbuathong (2009). However, there are several challenges in this concept. For example, in recent years, the demand for flexibility in automation system has been significantly increased, which resulted into the requirement of more complex hardware (embedded) systems. This increasing complexity of embedded (real-time) systems further leads to increasing demands with respect to engineering requirements, high level design, early error detection, productivity, integration, verification and maintenance (Koren *et al.*, 2000; Mehrabi *et al.*, 2000). Therefore, a right strategy is required to use components effectively for making an automation system. These are discussed further in this chapter.
Chapter 5: Developing Component-Based Automation Systems

5.3 Designing control elements of a component

In section 5.2, the physical and functional representation of an automation component is described. The next step is to design the behaviour and logical conditions for state transitions of any control element. For this purpose, a finite state machine (FSM) model is used. By definition, a FSM is typically an abstract machine having a finite number of states; with the transition from one state to another governed by specific logical conditions (Girault et al., 1999; Hopkinson, 1998). Any action or output takes place when there is any change in state.

Thus, FSM can be considered as discrete-event based frameworks to specify, design and verify control systems. However, this approach has several limitations such as complexity and difficulties involved when redesigning a system. Therefore to overcome these problems, a modular finite state machine (MFSM) framework was proposed by Endsley (2004). This framework was based on decomposing the whole system into smaller interacting modules (finite state machines). Each of these modules can be independently developed and verified as a single logical functional block. Therefore, this framework allows the task of designing a complex control system by breaking down into smaller, manageable parts. However, this approach is not suitable for large automation systems. It is because large systems may result into finite state machines with potentially large number of states and associated conditions for state transitions, which can make the finite state machines difficult to comprehend. Therefore, this approach is not feasible in these situations because of the large size of the resulting control program and the associated complexity in developing, debugging and maintaining the program (Endsley et al., 2006).

Therefore, a distributed finite state machine (DFSM) is used in this thesis and in the Loughborough research groups distributed control work more generally. In this approach, instead of decomposing an automation system top-down (as in MFSM), the opposite bottom-up approach is used. In this bottom-up DFSM approach, an automation system can be realised by designing small finite state machines for common groups of control elements and then connecting these together in a specific way to achieve the common objective (Kappes et al., 2000). This approach is advantageous as the low-level control elements are often simple and, therefore, can be represented by simple FSMs comprising of a few states and transitions. The another key advantage is
that these FSMs will be generic (independent of any specific application) and thus can be used in other applications. A control system can be composed from these distributed finite state machines by interlocking the transitions of a FSM with the states of other FSMs. This approach can be better explained using the following example.

Figure 5.6: An example of designing control systems using distributed finite state machines.

Consider a subsystem shown in Figure 5.6, which comprises two components: Distribution hopper and Swivel arm. Distribution hopper consist of a pneumatically controlled ejector and two sensors. The ejector moves horizontally between two positions named home and work. Swivel arm consists two pneumatically controlled actuators and a sensor. These actuators controls the circular motion of the swivel arm between two positions named upstream and downstream, indicated by the upstream and downstream sensors respectively. The gripper sensor shows the status of a vacuum element in situations when it grips or releases a workpiece. The step by step actions performed by this subsystem on the work piece (Figure 5.6) are shown in Figures 5.7 & 5.8 and are explained below:

1. The workpiece arrives in the distribution hooper; which makes magazine sensor to change its state (OFF to ON) indicating that it is full (Figure 5.7(a)).

2. As soon as the magazine sensor gets full (ON) and the magazine transfer sensor state is empty (OFF), the pressure in the pneumatically controlled ejector gets activated. This makes ejector start pushing the workpiece towards magazine transfer sensor (destination) of the distribution hopper (Figure 5.7(b)).
Chapter 5: Developing Component-Based Automation Systems

Figure 5.7: Part 1: A stage by stage representation of a subsystem (Figure 5.6) consisting a distribution hopper & swivel arm to illustrate the concept of DFSM.
Chapter 5: Developing Component-Based Automation Systems

Figure 5.8: Part 2: A stage by stage representation of a subsystem (Figure 5.6) consisting a distribution hopper & swivel arm to illustrate the concept of DFSM.
3. When the workpiece reaches at the destination place, the magazine transfer sensor changes its state (OFF to ON) indicating that it is full (Figure 5.7(c)).

4. As the magazine transfer sensor changes its state to full, the Arm starts moving towards downstream position to pick up the workpiece (Figure 5.7(d)).

5. The arm reaches at the downstream position (destination of the distributed hopper) to pick up the workpiece. This is indicated by the downstream sensor, which gets switched ON. The vacuum element gets activated to grip the workpiece, which is indicated by the gripper sensor (Figure 5.8(a)).

6. When the arm grips the workpiece at the downstream position, the pressure of the pneumatically controlled ejector gets deactivated and it starts going back to its home position (Figure 5.8(b)).

7. When the ejector reaches at the home position, the magazine sensor gets switched OFF, indicating it is empty. The arm picks up the workpiece and starts moving towards the upstream position. The hopper destination position becomes empty, so the magazine transfer sensor gets switched OFF (Figure 5.8(c)).

8. Finally, the arm reaches the upstream position (indicated by upstream sensor) and the vacuum gets switched OFF to release the workpiece (Figure 5.8(d)).

To design the control elements of this subsystem (shown in Figure 5.6) using the proposed DFSM approach, the following steps are taken:

1. Firstly, the total number of elements are identified, which controls the behavior/actions of the system to be designed. In this case there are 3 sensors and 2 actuators (pneumatically controlled ejector and arm).

2. In the second stage, each element is given a distinct number as mentioned in Table 5.1. The total number of states of each element is defined and each state is assigned a number in serial order. For example, sensor has two states (ON and OFF) while ejector and arm have four states, with two representing their static positions (home & destination) and other two dynamic states (moving to home or destination). If, say, any sensor element having two states is represented by integer 1, then its corresponding state numbers will be 1.1 & 1.2.
3. In the third step, the elements with same behaviour are classified as one element. For example, there are 3 sensors which have same behaviour: ON and OFF. These can be considered as one sensor repeated three times. The finite state machines of all these selected elements is then designed using the state transition diagrams (STD). STD describes the complete behavior of an element. STDs of the three elements (sensor, ejector and arm) is shown in Figure 5.9.

4. In the fourth stage, a DFSM model is made for both the components using the STDs of individual elements, as shown in Figure 5.10. This will represent all the internal and interlocking conditions required to perform the actions/events.

5. In the next stage, all the possible combinations of states of the elements of a component are listed in a Table (See Tables 5.2 & 5.3 for components 1 & 2 respectively). Using the DFSM model in Figure 5.10, the interlocking of the states is defined in Tables 5.2 & 5.3, for the required events. It will be appreciated at this point that not all the state combinations are used. These are listed to allow the user to select different combinations in future, if required. This will make the overall system reconfigurable and flexible. The output state is set for the selected state combinations. For example, when the work piece arrives, then the elements of the component 1 will be in states 1.2, 2.1 & 3.1. The pressure (Output) of the pneumatically controlled ejector is set (HIGH) to move the ejector. So, the output is set to HIGH for state combination of 1.2, 2.1 & 3.1. In this way the complete Table can be realised. The key benefit of this approach is that user just needs to change only one variable (Output) for any modifications in the behavior of a component.

6. Finally, to establish the logic for the control elements, all the state entries of each row in Tables 5.2 & 5.3 are combined using logical AND operations and each output of these logical AND operations are combined using logically OR to get the desired final output.
Table 5.1: Example of assigning state numbers to control elements in DFSM approach.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Number</th>
<th>Number of States</th>
<th>State Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magazine Sensor</td>
<td>1</td>
<td>2</td>
<td>1.1, 1.2</td>
</tr>
<tr>
<td>Magazine Transfer Sensor</td>
<td>2</td>
<td>2</td>
<td>2.1, 2.2</td>
</tr>
<tr>
<td>Ejector</td>
<td>3</td>
<td>4</td>
<td>3.1, 3.2, 3.3, 3.4</td>
</tr>
<tr>
<td><strong>Component 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arm</td>
<td>4</td>
<td>4</td>
<td>4.1, 4.2, 4.3, 4.4</td>
</tr>
<tr>
<td>Gripper Sensor</td>
<td>5</td>
<td>2</td>
<td>5.1, 5.2</td>
</tr>
<tr>
<td><strong>Component 2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.9: An example of designing STDs of control elements.
Chapter 5: Developing Component-Based Automation Systems

Figure 5.10: An example of making a DFSM model of a system using STDs of its control elements.
Table 5.2: State transition table reporting all the combinations of states of control elements of component 1.

<table>
<thead>
<tr>
<th>Magazine Sensor (1)</th>
<th>Magazine Transfer Sensor (2)</th>
<th>Ejector (3)</th>
<th>Interlocking with Component 2</th>
<th>Output</th>
<th>Action/Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>2.1</td>
<td>3.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.1</td>
<td>3.2</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.1</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.1</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.2</td>
<td>3.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.2</td>
<td>3.2</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.2</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.2</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.1</td>
<td>3.1</td>
<td>5.1</td>
<td>HIGH</td>
<td>WP Arrives, Move to Work Position</td>
</tr>
<tr>
<td>1.2</td>
<td>2.1</td>
<td>3.2</td>
<td>HIGH</td>
<td></td>
<td>Moving to Work Position</td>
</tr>
<tr>
<td>1.2</td>
<td>2.1</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.1</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.2</td>
<td>3.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.2</td>
<td>3.2</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.2</td>
<td>3.3</td>
<td>5.3</td>
<td>LOW</td>
<td>Move to Home Position</td>
</tr>
<tr>
<td>1.2</td>
<td>2.2</td>
<td>3.4</td>
<td>LOW</td>
<td></td>
<td>Moving Home Position</td>
</tr>
</tbody>
</table>

Table 5.3: State transition table reporting all the combinations of states of control elements of component 2.

<table>
<thead>
<tr>
<th>Gripper Sensor (4)</th>
<th>Arm (5)</th>
<th>Interlocking with Component 1</th>
<th>Output</th>
<th>Output</th>
<th>Action/Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>5.1</td>
<td>2.2,3.3</td>
<td>HIGH</td>
<td></td>
<td>Move to Downstream</td>
</tr>
<tr>
<td>4.1</td>
<td>5.2</td>
<td></td>
<td>HIGH</td>
<td></td>
<td>Moving Downstream</td>
</tr>
<tr>
<td>4.1</td>
<td>5.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>5.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>5.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>5.3</td>
<td>3.1</td>
<td>HIGH</td>
<td></td>
<td>Move to Upstream</td>
</tr>
<tr>
<td>4.2</td>
<td>5.4</td>
<td></td>
<td>HIGH</td>
<td></td>
<td>Moving Upstream</td>
</tr>
</tbody>
</table>
Thus utilising this DFSM approach,

\[
\text{an automation system can be composed of a network of finite state machines which communicate asynchronously with each other over a distributed control network by sending and receiving state messages and operate autonomously according to their state-transition behaviours and interlocks.}
\]

The CB approach can offer significant advantages over contemporary methods of control system development; such as:

- **Generic structure:** The behaviour of the control elements as described in their STDs are not associated with any specific system implementation or control application. Therefore, these elements are generic and can be used as common building blocks for other applications or control system.

- **Reconfigurability:** The control system’s behaviour can be easily altered by just selecting the different combinations of states from the state transition table and by adding or removing the interlocking states as required. This makes the system reconfigurable.

- **Ease of use:** The behaviour of control elements can be implemented as simple finite state machines which can be encapsulated and reused as a black-box. Their functional representation can be easily expressed using STDs. It is much easier as the system specification and development process in this approach primarily concerned with specifying the interlocks between the control elements instead of developing programming code for the entire control system.

- **Reusability:** As mentioned above, DFSM effectively separates low-level control implementation from application specification and development; therefore, it allows control systems to be developed from reusable control elements at a higher level of abstraction.

- **Encapsulation:** In DFSM approach, the control behaviour of low-level devices/elements can be encapsulated using simple finite state machines. This will
help in the realisation and implementation of the component-based automation system.

- **Modularity:** The control elements are generic and can be used as common building blocks for other applications or control system. So the system designed using these elements will support modularity and ability to be easily reconfigured.

- **Flexibility:** As the control elements are generic and can be used as modules, so the overall system designed using the CB approach will be flexible for any decomposition/composition and integration with other systems or components. The framework supporting this interaction of components/systems in the domain of automation systems is presented in the next section.

### 5.3.1 Component-Based system development framework

In previous section, the design of software based control elements is presented. In practise, there can be many subsystems or components (control elements) in an industrial automation system. Therefore in order to build systems using DFSM approach, these components or subsystems needs to be integrated (Pasetti *et al.*, 1999). This presents the need of an integration framework to support and facilitate the design, implementation and operation of the component-based manufacturing automation system. In current research, this is done by using the concept of component-based software engineering (CBSE) (Hammer & Chaudron, 2001). In CBSE, a desired system is constructed using components, a well-defined component framework, interfaces and an appropriate contract for system construction and operation. This is reported to be beneficial as it increases the overall productivity by reducing the amount of effort needed to develop, update and maintain system (Antoniadis & Leopoulus, 2000).

To enable system integration in DFSM approach, using the CBSE concept, the framework including the component interfaces is shown in Figure 5.11. The CB design framework in Figure 5.11, provides components with a platform to enable process editor functionality, data storage for system data logging and the component library, an interface to support manufacturing applications and control elements design & build. This framework can be extended and integrated to other frameworks or higher level
applications using interfaces shown in Figure 5.11. Within this framework, the component developer can create new components (software) and store these in the containment library for future (re)use. The control system integrator can build the machine application(s) using the component interfaces. These interfaces allow the integration of components with other supported applications, such as human machine interface (HMI) and control configuration data for machine operations. These machine application(s), built by the system integrator, can be saved and reused for future changes in new/existing machine configurations.

![Component-Based design framework for automation systems](Image)

Figure 5.11: Component-Based design framework for automation systems (Phaithoonbuathong, 2009).

To establish communication in this framework, components use their *interfaces*. The term ‘*interface*’ can be referred to the collection of operations used to specify the *services* of a *component*. Components can have more that one interface through which they can interact with other components. To establish communication, a component exports its one or more interface(s) to other components. In control system(s), this in-
teraction between components is achieved by interlocking the state variables, exchanged through the input and output interfaces.

Thus, the component-based design framework presented in Figure 5.11, have the required component functionalities to facilitate the design and integration of reconfigurable machine applications in the automation domain. 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 utilising Web Services.

### 5.4 Deploying the software components

After the design, the next step in the CB approach is the composition and deployment of software applications (components) on targeted hardware devices. Firstly, the software components are composed in the form of source code written in languages (such as C or Java) or graphical based platforms such as functional block diagram or structured text (e.g. the standard IEC 61131 (Lewis, 1998) and IEC 61149 (IEC, 2000)), depending on the target environment provided. In the next stage, these software components are configured and interlocked to compose the overall system with the required functionalities. After the system has been composed, a code generator is used to translate the composed software based components into machine compatible languages; followed by making an executable application. This final executable application is then installed into the targeted hardware device and this phase is called deployment of component/system. This composition and deployment is detailed in Chapter 7. To properly compose and deploy the components, there are some requirements of a set of runtime services to support and enforce the execution of the component-based manufacturing system. These are described below:

#### 5.4.1 Physical resources

A key issue concerning CBSE is the amount of efforts involved in mapping of logical software components to physical resources such as processing power, memory and communication bandwidth. In addition, cycle time for executing a software application, quality of service and deadline are also of high concern as automation hardware devices are real-time systems. These issues can produce adverse affects on the performance of other components, if any/some components consume a significant amount of these
Chapter 5: Developing Component-Based Automation Systems

resources. Therefore to avoid such issues, the components in CB approach are assigned with pre-defined physical resources within the component boundary; which are not accessible to other components (Harrison & West, 2000b). Thus, each component has a limit and can only consume the resources allocated to it during the component development phase. This will help to individually estimate the performance of each component. This characteristic will be significantly beneficial in manufacturing automation systems as the systems can be developed and implemented without the need to conduct system wide performance analysis such as load balancing for ensuring the system can deliver the expected operational throughput. However on the downside, this will limit the size of software application which can be deployed in an individual component.

5.4.2 Establishing communication channel

As discussed before in section 5.2, components communicate via their network interfaces. A communication channel is established by connecting the output network interface(s) of a component to the input network interface(s) of other components, to which the communication is required. During communication, state variables are exchanged in a manner the values generated from the output network interfaces of a component are transmitted to the receiving input network interfaces of other component.

The transmitting component has no knowledge of who will be the recipient of its network data during the component development phase. This knowledge is specified after the component is delivered to the machine builder, who will assemble the automation system by specifying the interlocks. This process of logically interconnecting component(s) output network interfaces (transmitter) to targeted components input network interfaces (receiptor) after the components have been developed is called late binding. Late binding supports the interconnecting of components for communication purposes, even they are completely unaware of each other. This process is typically performed during the system installation and commission. The state based design of components presented in this chapter allows late binding by interlocking the state variables of the components to perform the desired actions (See example presented in section 5.2 for details).
Chapter 5: Developing Component-Based Automation Systems

5.4.2.1 Communication models

As mentioned above in section 5.4.2, the components communicate (exchange state variables) through their network interfaces. The two communication models which support this communication in automation systems are: client-server (Figure 5.12(a)) and peer-to-peer (Figure 5.12(b)). The client-server communication model is a centralised approach, in which a central server (Figure 5.12(a)) is used to store the information specific to all the automation devices communicating in a network (Tanenbaum, 2002). All other devices register to the network; communicate through this server. In this environment, device interaction is dependent on the server handling this communication. This approach is also called central directory lookup or many-to-one communication as the central node (server) stores and controls all information exchange. This approach is quite popular and some of the examples of this client-server communication models include Microsoft’s Distributed Common Object Model (DCOM) (Harrison, 1998) and Common Object Request Broker Architecture (CORBA) (Vinoski, 1997). However, this approach has a key problem of message collisions on server side, especially in situations of heavy traffic of messages.

In peer-to-peer communication model, the devices are self-organised. Each device can act as a controller node and hold information about all other nodes in a network. The interaction is done via sending a multicast message to neighbouring devices, in order to find the interested devices to communicate with. Within the peer-to-peer network, devices interact directly rather than passing message to central server like in client-server model. Thus, it is also called many-to-many approach. Network data distribution service (NDDS) (RTI, 2002) and LonTalk protocol (ECHELON, 1999) are two examples that support a peer-to-peer communication architecture.

Comparing these two models, failure of server in client-server model will cause failure of whole network; whereas, it will not affect the system in peer-to-peer model. There is also a probability of message collisions on server side in client-server model. The peer-to-peer model is a fully distributed approach; thereby, resolving some of the message collision issues via multicast. However, it should be noted that message collisions still have to be managed or avoided via suitable protocols in a peer-to-peer communication systems. Thus in the CB approach, the traditional client-server model is likely to be
relatively inefficient, require a high network bandwidth and lead to unnecessarily complex system designs and degrades networking performance (Phaithoonbuathong et al., 2008). Therefore in reported research in this thesis, a peer-to-peer model is employed for communicating components.

5.5 Stakeholders in implementing CB systems

This section, finally, presents the role of stakeholders, such as end-users, machine builders and component suppliers; involved in the implementation of CB system. The roles & responsibilities of each stakeholder is shown in Figure 5.13 and are summarised below.

- **Component suppliers:** In the component-based design of automation systems,
Chapter 5: Developing Component-Based Automation Systems

cOMPONENT suppliers (also called component developer) supply automation parts to the machine builder. They also provide training to the system developers on integrating components to build the system with the required control functionality. Component suppliers have the responsibility to develop the encapsulated control functionality (component abstraction and logical network interface) and the low-level programming required by system integrators, i.e., machine builder and end-user.

- **Machine builders**: Machine builders perform the role of system integrator. They are responsible for selecting components from independent component suppliers and then integrating these to form the automation system. Instead of developing low-level programs to control the automation hardware, the machine builders configure the components and logically (interlocking) couple these together to create the system control behaviour, required by end-users.

- **End users**: End-users operate, monitor and maintain the system. In the lifecycle management of control system, upgrading/maintenance may be performed by machine builders or component developers, without any effects on the integrity of system. Faulty components can be directly replaced (Harrison *et al.*, 2001). The component supplier and machine builder can independently perform upgrades to components or system respectively independently, without affecting the integrity of overall system.

![Figure 5.13: Role of the stakeholders in an automation system.](image)

Chapters 7 & 8 present the roles of stakeholders involved in implementing the CB approach on the Ford-Festo test-rig assembly machine at Loughborough University. The development of components for this test-rig considered the roles of component
supPLIER, MACHINE BUIldER AND THE ENd-USER. THE COMPONENTS WERE LATER INTEGRATED TO
COMPLETE THE AUTOMATION SYSTEM WHICH THE ENd USER(S) CAN THEN OPERATE.

5.6 Summary

This chapter presents the physical and functional representation of an automation
component. Based on the requirements of manufacturing automation systems, an au-
tomation component is defined as an autonomous unit incorporating its own physical
hardware, application software and local intelligence that enables it to know what,
how and when to execute its own automation function without the need for a central
controller. It comprises of the automation device, a local microprocessor, and its in-
terfacing electronics. The logical behaviour of the component is represented by control
elements using the distributed finite state machines represented by state-transition dia-
grams. Components interact with one another through well-defined network interfaces
and are configurable through configuration parameters. These parameters are used to
modify their specific operational characteristics as well as their interlocking relation-
ships with other components. The component based system is designed by configuring
these interlocking relationships. The services and requirements for the implementa-
tion and deployment of components have also been presented. Finally, the chapter
ends presenting the roles and responsibilities of stakeholders involved in the design and
development of manufacturing automation system, using the CB approach. This chap-
ter has hence characterised the CB system architecture which is to be realised on a
SOA/WS platform as a key goal of this research.
Chapter 6

Engineering Methods for Web Services

In Chapter 5, the physical and functional representation of an automation component is presented; where these components are used as building blocks to make an automation system. This CB concept offers many advantages over traditional design of automation systems. However, CB implementation approaches to date have been either vendor specific and/or have lacked the ability to be integrated openly with higher level business and engineering systems. Thus the system realisations to date have not fully meet the requirements of flexible automation system. However, Web Services can be used to potentially overcome these shortcomings, because Web Services are commonly used in higher business applications. Thus, if Web Services can be effectively used in CB automation systems, then this will provide a technologically neutral platform to integrate component based automation systems at shop-floor level to the higher level business and engineering applications. Therefore, this chapter presents a selection of engineering method, which can be used to design software based control applications and integrate these using Web Services.
6.1 Need of Web Services in CB automation systems

The literature review presented in Chapters 2 & 3, illustrates the needs of an agile automation system for manufacturing industries. The key requirements of such an automation system is its ability to provide: (a) the flexibility to support product variety dynamically and, (b) the reconfigurability to manufacture new generations of a given product economically. To meet these requirements effectively, a component based approach is presented in Chapters 4 & 5. In this approach, an automation system is built by using reconfigurable and reusable automation components as common building blocks. As detailed in Chapters 4 & 5, this CB approach offers several benefits over traditional design of automation systems. Some of these benefits include improved flexibility, fault tolerance, self-configuration and self organisation.

However, today’s automotive industries are looking forward to develop, deploy and support automation systems across the globe, in short timescales (Phaithoonbuathong et al., 2010). This means to manage these globally distributed systems; a readily configurable information exchange is required, not only between geographically dispersed higher level business partners and engineering teams but also with the shop-floor level embedded devices. More specifically, it means a requirement of seamless integration between higher level business applications and the shop-floor level automation components. This will be beneficial in enabling rapid reconfiguration and reusability of components when introducing new products.

Presently, industrial information and control systems are often fragmented, hard to manage, difficult to change or extend and isolated from higher level business systems (Colombo et al., 2005; Patrick & Fardo, 2009). It is because these systems being composed of a wide range of vendor specific components; along with their integration technologies and managing solutions also supplied by different vendors. These conditions usually lead to difficult situations of managing the integrity within these components at the shop-floor level and simultaneously with the higher level business applications. Therefore, this presented a need of distributed system capable of integrating these heterogeneous devices. So, the research work in this thesis is aimed to develop a technologically neutral platform for business and CB automation system integration.

This can be achieved by utilising Web Services for integrating components in automation systems. It is because Web Services (e.g. Ethernet) are commonly used
at the higher level business applications. Therefore, if Web Services can be used for integrating components in the automation systems; then this will give the required vendor-neutral platform for integrating enterprise and shop-floor automation systems. This application of Web Services aims to enhance the degree of modularity and integration capability of the automation devices. This approach is expected to benefit various developers and users by reducing development costs, system maintenance provision, time to market and realising more reusable systems. Therefore, the research reported in this thesis aims to demonstrate the feasibility of employing Web Services at embedded device level to enable the reconfigurable agile automation systems.

6.2 Engineering requirements for CB design of agile automation systems

The CB approach presented in Chapters 3 & 4, has already been implemented using orchestrator based WS approach on laboratory scale, at MSI Institute of Loughborough University (Lee, 2004). The details of this research is reported in Phaithoonbuathong (2009). However, this research has several shortcomings such as high memory consumption and delay in communicating devices (See Chapter 8 for details). Therefore, the research presented in current thesis is the extension of work done at MSI lab, with the aim of applying choreography based SOA-WS approach to the design of CB automation system.

To realise an agile automation system using Web Services and CB concept, it is essential that the design of CB automation systems must meet the end-user requirements. In the current research project, the end-user is Ford and the requirements specified by Ford are based on their assembly machines. These requirements are detailed in Table 6.1 and are adopted from literature presented by other researchers at MSI Institute (Phaithoonbuathong et al., 2008). From this literature, it can be seen that a typical assembly machine at Ford usually contains 10 to 40 machine components with an average machine cycle time of 30-35 seconds. Typical example of components includes clamps, motor drives, RF tag readers and separators. To effectively design and support the production of automation systems, a detailed knowledge of the functional properties of each component is required. Therefore, a proper selection of the functional properties
of a component is very important. In this thesis, the end user requirements obtained from Ford are used to define the component functionalities.

Table 6.1: Ford requirements for the CB design of an automation system (after Phaithoonbuathong et al. (2008))

<table>
<thead>
<tr>
<th>End-user requirements</th>
<th>CB specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconfigurable automation systems</td>
<td>Well defined components and their finite state transitions</td>
<td>Use of well defined components to build machine applications, in order to help integrators by providing the previous design and resources when reusing the existing components.</td>
</tr>
<tr>
<td></td>
<td>Loose coupling with other components</td>
<td>The overall system should be reconfigurable in a manner that changing a component will not affect other components.</td>
</tr>
<tr>
<td></td>
<td>Device discovery and initialisation</td>
<td>The CB development platform should support device discovery, and dynamically allocate the port location to new devices in order to ease device installation within the system.</td>
</tr>
<tr>
<td>Open automation platform</td>
<td>A unified and language neutral framework</td>
<td>The low level programming code in CB design should support interoperability and interaction among different devices through a unified interface (e.g. Web Services interface in current research).</td>
</tr>
<tr>
<td>Performance and reliability</td>
<td>Real-time communication</td>
<td>The response time of I/O devices, when accumulated should meet the targeted production system’s cycle time.</td>
</tr>
<tr>
<td></td>
<td>Embedded controller based design with low consumption of physical resources</td>
<td>The final executable code to be installed in the embedded device should be compacted enough to consume minimum processing memory and fast enough to meet the required I/O response time.</td>
</tr>
<tr>
<td>Visualisation and data monitoring</td>
<td>High level machine configuration using process engineering tools supporting virtual engineering and control logic simulation</td>
<td>A user-friendly process engineering tool in compliance with CB design is required to manage the complexity in commissioning production process.</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Self error diagnostic</td>
<td>Self error diagnostic routines needs to be designed and pre-installed in the device by the component builder for the end user. This will help the user in automated monitoring of the device for active maintenance.</td>
</tr>
<tr>
<td>Integration to higher level applications</td>
<td>Exposure of abstract device functionality</td>
<td>The components should be self explanatory in providing instructions for integrating within the manufacturing system and activating the interface for device communication and interaction.</td>
</tr>
</tbody>
</table>
Table 6.1 – Continued from previous page

<table>
<thead>
<tr>
<th>End-user requirements</th>
<th>CB specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulate the complexity of low level coding</td>
<td>Low level program code should be written in a manner to be encapsulated in the hardware device, and ready to be integrated by the application builder and end users. This will make it easier for application builders to focus on building and reconfiguring automation systems from high level control applications.</td>
<td></td>
</tr>
</tbody>
</table>

6.3 Integrating distributed CB automation systems

As discussed above in section 6.1, the distributed automation systems consist of a variety of hardware devices and their associated software; usually supplied by different vendors. These vendor specific devices and software are usually based on different defacto standardised technologies. Therefore, their integration with each other and/or with higher business processes is a complicated task. However, these complex and heterogeneous distributed systems can be integrated using various software technologies called middleware.

The middleware itself are of several types and are not all standardised, so it is difficult to give a definition to middleware. However, it can be broadly defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system (Figure 6.1). In simple words, middleware can be considered as a software that makes a distributed system programmable and manageable by masking some of the kinds of heterogeneity of distributed systems. Some of these include heterogeneity of networks, hardware; and heterogeneity of operating systems or programming languages, or both. Some of the examples of middleware include RPC, MOM, CORBA, DCOM and JavaRMI.

From the literature presented in Chapters 2 & 3, the existing middleware technologies are diverse and have limitations in terms of flexibility and agility. This is because these available technologies are proprietary, complex to use and do not interoperate well with each other. Several proposals have been put forward by a variety of consortia and standard bodies to integrate these middleware solutions. Some of these include multi-agent and holonic systems, which have been a subject of great attention by many researchers such as Fletcher et al. (2003); Leitao et al. (2004) and Yu & Krishnan...
Chapter 6: Engineering Methods for Web Services

Figure 6.1: Middleware layer in context.

(2004). However, these approaches are able to meet only a few requirements of agile manufacturing and have several shortcomings. Some of these shortcomings include the lack of widely accepted standards, poor scalability between various technologies, inflexible communication infrastructure among components, and the difficulty of porting application softwares to new configurations.

Therefore in an attempt to address the key limitations mentioned above, a service-oriented architecture is used to yield an open, flexible and agile environment with plug-and-play connectivity, in current research. It has been discussed in Chapter 3; however, it is reproduced briefly here to explain its implementation in CB automation systems.

SOA can be implemented in the domain of automation engineering by many technologies such as CORBA and Web Services (V. M. Trifa & Koehler, 2008). However in this research, Web Services is used because the open standards of the Web Services family allow the implementation of SOA in a technology neutral fashion. This makes SOA particularly applicable for a global multi-vendor environment where interoperability is essential. Thus in a nutshell, the approach of expanding the use of SOA to low-level real-time embedded devices paves the way for fulfilling the reconfigurability and flexibility requirements. This is discussed in detail in the following section.


6.4 Implementing SOA using Web Services

In the domain of automation systems, SOA can be defined as an application architecture within which all functions are defined as independent services with well-defined invokable interfaces. Unlike traditional object-oriented architectures, SOA services are loosely coupled\(^1\) and highly interoperable. In the current research, the term *services* refers to all the software components. A software component consists of an implementation code and interface. SOA follows the paradigm to explicitly separate the implementation code from the interface. In current research, these interfaces are defined using Web Service Description Language (WSDL) (Jammes & Smit, 2005).

The individual services of SOA is associated using orchestration (Jammes *et al.*, 2005). In the process of orchestration, the services are associated in a non-hierarchical manner using a software tool that contains a complete list of all available services, their characteristics, and the means to build an application utilising these sources. A complementary concept to service orchestration is ‘service choreography’ (Peltz, 2003); which considers the rules that define the messages and interaction sequences required in order to execute a given process through a particular service interface. To enable the services, SOA requires metadata, which refers to the data to describe the characteristics of services and also the data to drive them. The Extensible Markup Language (XML) is generally used to create metadata for interfacing services, WSDL for describing services, and Simple Object Access Protocol (SOAP) as communication protocol.

SOA can be implemented using a wide range of technologies such as RPC, REST, DCOM, CORBA, Jini, Web Services, DDS and WCF. However in the author’s research, Web Services is used to implement SOA, because Web Services standards (e.g. SOAP) provide greater interoperability by making functional building blocks (services) accessible over the standard Internet protocols (such as HTTP) that are independent from vendor specific platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network enabled (Tidwell, 2000; Topp *et al.*, 2003).

Figure 6.2 shows the basic principle of implementing SOA using Web Services for the purpose of integrating shop floor automation systems to the higher level business

\(^1\)The term loosely coupled means that services in a SOA can be connected without having knowledge of their technical details.
applications such as customer relation management (CRM) and management information system (MIS). In Figure 6.2, SOA distinguishes three different roles of services: service provider, service requester and service broker; explained below:

- **Service provider**: The service provider creates Web Service and publishes its interface and access information to the service registry. The service provider decides which services to expose for a given broker and the information related to their availability, security, price and ways to exploit them.

- **Service requester**: The service requester locates entries in the broker registry and binds to the service provider in order to invoke services. The brokers look for services they need and then bind themselves with respective service to use it. They can access multiple services if available.

- **Service broker**: The service broker is responsible for enabling service discovery.

### 6.4.1 Embedded device level SOA

As discussed in Chapters 3 & 4, in the author’s research, the embedded devices in the CB automation system should interact in a peer-to-peer fashion. To implement SOA
in such automation systems, the embedded devices can be categorised to play the role of either controlling devices or controlled devices or both, for enabling peer-to-peer interactions (Kaur et al., 2010). The peer-to-peer interaction patterns of this device-level SOA can be categorised according to the following five levels of functionality:

1. **Addressing**: It is the foundation for device networking and depends upon network protocol. For example, for IP-based networking, the addressing is provided by the IP protocol, either IPv4 or IPv6.

2. **Discovery**: After getting the IP address, the next step for the devices is to discover each other. When a controlled device is added to the network, a discovery protocol enables it to advertise its services on the network. However when a controlling device enters a network, it sends out a search request and then the device(s) that match the request sends a corresponding reply.

3. **Description**: Once a controlling device has discovered a controlled device, to learn more about the latter and its capabilities, the controlling device retrieves the controlled device description (metadata).

4. **Control**: Once it knows a controlled device, a controlling device can exert control over it. To invoke an action on a service, a controlling device sends a control message to the network endpoint for that service. In return, the service may or may not return a response message providing any command specific information.

5. **Eventing**: In addition, devices may communicate through asynchronous eventing, usually implemented by a publish-subscribe mechanism. In this mechanism, a service exposes events corresponding to internal state changes, to which controlling devices can subscribe in order to receive event notifications whenever the corresponding internal state change occurs.

### 6.4.2 Device level SOA protocols

To implement SOA in automation system, several device level SOA technologies have been proposed, most notably Jini and Universal Plug and Play (UPnP). Jini is based on Java; therefore, it lacks platform neutrality. The UPnP architecture leverages Internet and Web technologies including IP, TCP, UDP, HTTP, SOAP and XML; hence it is
platform-neutral. However, it uses specific protocols for device discovery and eventing, and a specific XML based language for device and service description. A very promising approach is Devices Profile for Web Services (Jammes et al., 2007), as described below in next section. It has the same advantages as UPnP, but additionally it is fully aligned with Web Services technology. Therefore, it is used in this research for implementing device level SOA and thus building the CB automation systems.

6.5 DPWS for implementing device level SOA

To implement SOA, this research project has utilised the DPWS protocol stack as shown in Figure 6.3 and explained below.

<table>
<thead>
<tr>
<th>Application-specific Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Discovery</td>
</tr>
<tr>
<td>WS-Security</td>
</tr>
<tr>
<td>WS-MetaDataExchange</td>
</tr>
<tr>
<td>SOAP 1.2</td>
</tr>
<tr>
<td>UDP</td>
</tr>
<tr>
<td>IPv4/IPv6</td>
</tr>
</tbody>
</table>

Figure 6.3: DPWS protocol stack (Jammes et al., 2007).

- **Web Services Description Language (WSDL):** It is used for the abstract description of service interfaces and their binding to transport protocols using XML. Interface definitions are made up of messages i.e. requests, responses and notifications.

- **XML Schema:** It is the definition of the data formats used for constructing the messages addressed to and received from services.
• **Simple Object Access Protocol:** SOAP is an XML based protocol to provide service related messages (request & reply) formatted in accordance with the corresponding WSDL definitions. SOAP defines a framework for describing the contents of message and their processing, a set of encoding rules for expressing instances of application defined data types, and a convention for representing remote procedure calls and responses. SOAP is designed independent of any particular programming language; thus it provides a flexible framework with high interoperability for distributed and heterogeneous environments.

• **WS-Addressing:** It provides transport neutral mechanisms to address Web Services and messages. It is closely associated to SOAP and moves all message addressing information into the header of SOAP message envelope, which allows the message content to be carried over any transport protocol (HTTP, SMTP, TCP, UDP). More specifically, this specification defines XML elements to identify Web Service endpoints and to secure end-to-end point identification in messages.

• **WS-MetadataExchange:** Web Services use metadata to describe which endpoints (description, schema and policy) need to know to interact with them.

• **WS-Policy:** WS-Policy gives the generic instructions to specify the requirements and capabilities of senders and receivers, required to complement the WSDL description of the service.

• **WS-Security:** WS-Security is an optional set of mechanisms for ensuring end-to-end message integrity, confidentiality and authentication.

• **WS-Discovery:** DPWS uses the WS-Discovery protocol for plug-and-play device discovery. WS-Discovery defines a multicast protocol to search and locate targeted service. In the context of Web Service protocol stack, this targeted service is a device. To search the targeted device, the primary mode of service discovery is to multicast a probe. Any devices matching to the probe will then send a confirmation probe as acknowledgement. Once it has been discovered, then it exposes the services it provides.

• **WS-Eventing:** WS-Eventing defines a protocol, in which Web Service (called subscriber) registers interest (called subscription) with another Web Service in
receiving messages about events (called notifications or event messages). The subscriber may manage the subscription by interacting with a Web Service (called the subscription manager) designated by the event source. Event notifications themselves are one-way messages, the content of which may include any data of any type. These are transported in the same way as any other SOAP message.

From the above discussion, Web Services build, publish and locate services over the network. These have already been adopted in integrating business applications, which include event-driven, composite, and autonomous distribution of applications. These are similar functions (i.e. autonomous, event-based and distributed control systems) required in the automation domain. Therefore, Web Services based SOA can be applied to the automation systems, where control application activities would proceed in a similar manner to WS at business level. This adoption of Web Services through DPWS will enable open standards, which are technologically neutral, in order to provide interoperability across the entire heterogeneous enterprise applications, down from the sensors/actuators level up into higher level business applications (Kirkham et al., 2008a). The role of using Web Services and SOA in meeting the end user requirements (Table 6.1) is mentioned in Table 6.2. The next section gives an example of practically implementing Web Services based SOA in the automation domain.

Table 6.2: Role of Web Services & SOA in meeting the end user requirements.

<table>
<thead>
<tr>
<th>End-user requirements</th>
<th>CB specification</th>
<th>Role of Web Services &amp; SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reconfigurable automation systems</strong></td>
<td>Well defined components and their finite state transitions</td>
<td>Distributed component-based design using state transition diagrams.</td>
</tr>
<tr>
<td></td>
<td>Loose coupling with other components</td>
<td>To aide loose coupling, component functionality is prefabricated and interfaced to the Web Services.</td>
</tr>
<tr>
<td></td>
<td>Device discovery and initialisation</td>
<td>Web Services provide dynamic device discovery functionality.</td>
</tr>
<tr>
<td><strong>Open automation platform</strong></td>
<td>A unified and language neutral framework</td>
<td>Use of XML message as the standard communication for devices will provide unified interface for interoperability between different vendor devices.</td>
</tr>
<tr>
<td><strong>Performance and reliability</strong></td>
<td>Real-time communication</td>
<td>Web Services support deterministic communication at TCP/IP stack. To improve performance and reliability, RTOS will be implemented in the Web Services development for a real-time capability.</td>
</tr>
</tbody>
</table>
### Table 6.2 – Continued from previous page

<table>
<thead>
<tr>
<th>End-user requirements</th>
<th>CB specification</th>
<th>Role of Web Services &amp; SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Embedded controller based design with low consumption of physical resources</strong></td>
<td>Web Services and SOA based design is expected to consume low resources. See Chapter 7 for details.</td>
<td></td>
</tr>
<tr>
<td><strong>High level machine configuration using Process Engineering Tools supporting virtual engineering and control logic simulation</strong></td>
<td>Web Services provide reliable source of state propagation using WS-Eventing mechanism for real-time operation and visualisation devices such as HMI.</td>
<td></td>
</tr>
<tr>
<td><strong>Self error diagnostic</strong></td>
<td>Support remote diagnostic applications, data history logging, dynamic device meta-data for maintaining device lifecycle, health monitoring and maintenance routine. Web Services provide interface to the local and remote data monitoring system.</td>
<td></td>
</tr>
<tr>
<td><strong>Exposure of abstract device functionality</strong></td>
<td>Seamless integration to higher level business applications using Web Services interface provided by WSDL.</td>
<td></td>
</tr>
<tr>
<td><strong>Encapsulate the complexity of low level coding</strong></td>
<td>CB design based on state transitions allows the builder to build machine applications without getting into the details of low level code.</td>
<td></td>
</tr>
</tbody>
</table>

### 6.6 Designing a CB automation system using Web Services & SOA

This section is divided into parts. The first part explains the building of control application; while the second part presents its implementation using SOA & Web Services. A detailed example is reported in Chapter 7.

### 6.6.1 Building a control application using Web Services & SOA

The process of building a control application using the CB approach is presented in Chapter 5. However, when using Web Services and SOA, the software components are considered as services. To build the whole system using services, the process work flow of the system is considered. In process work flow (detailed in Chapter 5), the whole machine operation is distributed into components; which further comprise elements. At each stage in the workflow, an individual element is responsible for a specific task.
Once the task is complete, the individuals responsible for the next task are notified and receive the data required to execute their stage of the process. An ontology is used to describe the components, their properties and relationships to define the whole process work flow (Popescu & Lastra, 2007). The ontologies are concerned with the orientation of process work flows described from the service execution and message passing at the device level. However, the overall system is built by associating the services in a non-hierarchical arrangement using orchestration.

To better explain, consider the example of a system comprising distribution hooper and swivel arm mentioned in Chapter 5, which is reproduced here in Figure 6.4. To build the control applications using Web Services for this system, the following steps are taken:

1. The complete work process flow of the system is defined. For example in Figure 6.4, the work process flow refers to the step of (a) ejecting the work piece by ejector to the destination of distribution hooper; (b) the action of picking this work piece by the swivel arm and finally (c) transferring it to the upstream position.

2. In the second step, the work process is divided into control applications \([A_j, j = 1 \ldots m]\); where each control application is handled by a control device. In Figure 6.4, the work flow process is divided into two control applications: (a) ejecting a work piece (distribution hooper) and (b) transferring a workpiece (swivel arm). These are controlled by devices D1 and D2 respectively.

3. Thirdly, the operating functionalities of each control device are defined as services \([S_i, i = 1 \ldots n]\). For example in Figure 6.4, the control device A has two services: \(\text{extend} \& \text{retract}\). It is important to note that, in current research, the devices are encapsulated with distinct services. For example, \(S_1\) is only for control device A, and thus is not included within other control devices. This helps in clearly defining the boundaries of CB design and managing the complexities with ease. However, a control device can have multiple services depending on the decomposition of functions of the machine system.

4. In the final stage, the interlocks between these services is defined as mentioned in Chapter 5, to build the complete system application.
In summary, the above steps of building system using CB design approach can be mathematically (Lee et al., 2004a) expressed as:

\[ A[j]_{j=1...m} \subset \{S_1 \times S_2 \times S_3 \ldots S_n\} \] (6.1)

\[ S_i=1...n = f(\varepsilon_i, \Phi_i, X_i) \] (6.2)

\[ \varepsilon_i = \{\varepsilon_{\text{out}}, \varepsilon_{\text{in}}\} \] (6.3)

\[ \Phi_i = \{\Phi_i(K)\} \] (6.4)

\[ X_i = \{X\} \] (6.5)

where

- \( A = \) Applications,
- \( \subset = \) Subset of,
- \( S = \) Provided services,
- \( \Phi = \) Component transition states,
- \( \Phi_i(K) = \) Current component transition state,
- \( \varepsilon = \) Local event(s) and condition(s),
- \( \varepsilon_{\text{out}}, \varepsilon_{\text{in}} = \) Output event(s) and input event(s),
- \( m, n = \) Number of applications and services,
- \( j, i = \) Application and service ID,
- \( K = \) State ID, and
- \( X = \) Interlocking elements.

Using these mathematical relationships, interlocking (already explained in CB approach in Chapter 5) between components (Distribution Hopper and Swivel Arm) shown in Figure 6.4 can be mathematically expressed as:

\[ A[1] = \text{Moving work piece from A to B } \subset \{S_1\} \]

\[ A[2] = \text{Transferring work piece from B to C } \subset \{S_2\} \]

\[ S_1 = f_1(\varepsilon_1, \Phi_1, X_1) \]

Where,
Chapter 6: Engineering Methods for Web Services

Figure 6.4: An example of service orchestration ontology to build a control application.

\[
f_1 = \left( \varepsilon_1 \left( \begin{array}{l}
(Magazine\ Sensor = \text{FULL}) \ AND \\
\phi_1 (Ejector = \text{Retracted}) \ AND \\
X_1 (Arm = \text{Upstream})
\end{array} \right) \ AND \right) \Rightarrow \text{Set the Service} \\
S_1 = \text{EXTEND}
\]

\[
S_2 = f_2 (\varepsilon_2, \phi_2, X_2)
\]

Where,

\[
f_2 = \left( \varepsilon_2 (\text{Magazine Transfer Sensor = FULL}) \ AND \\
\phi_2 (Arm = \text{Upstream}) \ AND \\
X_2 (Ejector = \text{Extended})
\right) \Rightarrow \text{Set the Service} \\
S_2 = \text{DOWNSTREAM}
\]

These logical equations explain how the machine applications are built using the finite state machines of the CB approach. In the real machine environment, the control sequence can have multiple interlocking of states and transitions between devices. The system can be reconfigured by altering the application design associated with the state behaviour of the component, as represented by \( f (\varepsilon_i, \phi_i(K), X_i) \). The next section explains how this is implemented in practise on embedded devices.
6.6.2 Implementation

The process adopted by the author for designing CB automation systems using SOA and Web Services within embedded devices is presented schematically in Figure 6.5. The main design process involves creating the state transitions and control algorithms, Web Services interfaces, I/O interfaces mapping to physical I/Os of the device and deployment on embedded controllers, with all of these need to be reconfigurable. These design steps are explained below:

1. **Defining device state transitions**: The first step is to define the behaviour of components in terms of states and their associated state transitions and inter-locking conditions using the state transition diagrams (finite state machines), as explained in Chapter 5. This is usually done through application generator tools. A graphical application generator tool called Process Definition Editor (PDE) is currently under development in MSI lab of the Loughborough University (Thomas et al., 2002). In current research, PDE software is proposed to be used to define
finite state machines and interlocking logic conditions. To support reconfigurability, these can be altered within the PDE tools and then can be mapped again to the control applications through an interface. However to date, state transitions and interlocking conditions are defined manually offline by reading the XML files, generated by graphical based PDE tools.

2. **Designing the control application:** The second step is designing the control application. More specifically, it means to design the control logic for reading/writing to the physical I/O lines of the embedded devices according to the actual machine sequences. The control application can be designed using various programming languages like C, C++ and IEC-61131. In current research, IEC-61131 is used through a graphical tool called ControlBuild, provided by Geensys\(^1\). It is discussed in detail in section 6.7. The code for a component is designed in a manner that the system integrator do not need to rewrite this embedded code in order to build the system (comprising many components).

3. **Defining service binding and Web Services description:** After designing the control application logic, the next step is to define the Web Services interfaces in order to make it compatible for communicating with other WS based applications. In current research, ControlBuild editor is used to define the Web Services interface. These WS interfaces handle the device discovery, service binding and interface to the device control application. ControlBuild automatically generates a WSDL file which is used to bind services onto the embedded devices. A service is activated on meeting the set of conditions defined by logic and linked to the input gathered from the interfaces to I/O devices. This helps to integrate the embedded device at shop-floor to the higher level business applications.

4. **Simulation and testing:** The next step is to test the designed control application prior to its deployment on the embedded device. ControlBuild, used in current research, allows simulation and testing of desired control application in order to verify that it meets the required machine logic sequence. In ControlBuild, the status of any input variable can be changed to visually inspect the corresponding desired change in outputs while simulation is running. This will help to determine errors prior to its deployment on embedded device.

5. **Physical I/O configuration generation:** Following the testing of control applications, the Inputs/Outputs of the designed components are mapped to the physical input/output lines of the device by generating a configuration file. The input lines will read the status of the connected physical devices such as sensors and pass it to the control application to run the logic accordingly and the output of this control application is then passed to output lines for controlling the connected devices such as actuators.

6. **Deploying control application into embedded device:** The final step is to deploy the tested and configured components into Web Services based embedded devices. A selection of the hardware device will be made in Chapter 7.

### 6.7 Designing & deploying applications using ControlBuild

In current research, ControlBuild (a graphical based tool provided by Geensys) is used for designing, testing and deploying control applications for the distributed automation systems. Geensys refer ControlBuild as an open automation software platform that allows seamless progress through all phases of the application development cycle, i.e. from the definition and validation of specifications to implementation and deployment (Guerard, 2009b). Some of the key features and functions of ControlBuild include a variety of graphical interactive editors for component-based design covering IEC 61131-3, C and electrical circuit diagrams, a set of component libraries and a simulation environment for control software testing & validation prior to integration and commissioning. ControlBuild also allows direct monitoring the lifecycle of system for the maintenance purposes.

The principle of ControlBuild is based on a recursive component model called MAC\(^1\), in which an application is divided into components and sub components assembled graphically (Guerard, 2009b). For designing the control application for distributed automation systems, Control Build supports the dynamic deployment of Web Services. The principal concept created with dynamic deployment is the WS component. In Control Build, a WS component is a top level Mac with a WS interface defined. From a “DPWS point of view”, a WS Component is a logical device hosted by a physical device as shown in Figure 6.6. Therefore, this component is able to be discovered

---

\(^1\)MAC refers to model of assembly of components.
and/or to discover other components and has access to the physical devices resources such as I/Os and counters. A physical device can host several WS components and it is possible to move WS components from a device to another without modifying the application.

Figure 6.6: Logical view of a physical device (Karnouskos et al., 2007).

The design, testing & deployment of the control application as mentioned above in section 6.6, is done through the dynamic deployment toolchain (Guerard, 2009a) of ControlBuild, as shown in Figure 6.7 and explained below:

Figure 6.7: Dynamic deployment toolchain of ControlBuild (Guerard, 2009a).
1. **ControlBuild editor:** ControlBuild editor is used for developing applications, organising components, defining the algorithm inside the components in IEC languages and defining the interfaces of components. It also possesses the ability to test, validate and simulate the application before its deployment. In current research, IEC 61131-3 functional block diagram (FBD) language (Figure 6.8(a)) of ControlBuild is used to design the control application using the state transition definitions provided by the application tool generator. ControlBuild editor automatically generates XML file for this graphical (FBD) design of the control application as shown in Figure 6.8(b). In the next stage, ControlBuild editor produces a configuration file (.ioconf file) which gives the details of physical lines of embedded devices to which the component input/output is mapped (Figure 6.9). These can be manually altered by the user. A WS interface of component is defined (Figure 6.10(a)) and the ControlBuild editor automatically generates a WSDL file (Figure 6.10(b)) for this component interface, which is used for establishing network communication with other WS components. A detailed example of generating control applications using FBD in ControlBuild corresponding to the system comprising distribution hopper and arm (mentioned in Chapter 5) is given in Appendix A.

2. **Dyndep generator:** The main goal of dyndep generator is to enable other tools (such as IEC compiler and configuration compiler) to generate the control application. This involves converting the .wsdl files generated by ControlBuild and WS Management files (devices and service classes) into the .plcopen files, and copy them at the right place for deploying into each device.

3. **I/O generator:** The purpose of I/O generator is to generate the configuration file (.ioconf file), which contains the information to map the input/output of the software components to the physical I/O lines of device.

4. **IEC compiler:** The purpose of IEC compiler is to convert the .plcopen files to the binary program (bytecode), required to finally install on the device.

5. **Configuration compiler:** The DPWS/IEC driver needs to be configured to make a link between the variables used by the IEC engine and the messages received and sent by the DPWS stack. It is also responsible for the I/O configura-
The role of configuration compiler is to extract information from the service class resource (DPWS configuration files) and the I/O configuration to generate a binary configuration file used by the driver.

6. **Manager:** The role of manager is to manage (upload & delete) the configuration of a device supporting dynamic deployment capabilities. It finds devices with the discovery protocol, in which each device is uniquely identified by it’s metadata’s serial number.

The control application design process presented above is for only a single embedded device. However in practise, there can be more than one embedded device in an automation system. Therefore to locate the individual devices in a network, each device is assigned with an address, such as TCP/IP, which is used for device set up (e.g. deploying the control application) and communication (e.g. exchanging state information) between them. In current research, a Peer-to-Peer approach is used for establishing communication between these embedded devices in a network and defining the device localisation. This has been discussed in detail in Chapter 5; however, it is reproduced here for the purpose of its implementation using ControlBuild.
Figure 6.8: (a) FBD representation of control application in ControlBuild and (b) its corresponding automatically generated PLCopen XML file.
Figure 6.9: (a) Component view and (b) its corresponding I/O configuration file in ControlBuild.
Figure 6.10: (a) Web Services interface with its corresponding automatically generated WSDL file, (b) DPWS library function (in C) for a WS based component in ControlBuild.
6.7.1 Implementing Peer-to-Peer interactions using ControlBuild

The peer-to-peer approach for interacting embedded devices is shown in Figure 6.11. In this approach, each device is self-organised and can interact directly to other without passing the commands and information through a central server (Jazdi & Konnertz, 2003). A device typically interacts with others by sending a multicast message to find interested devices to interact with. If a new device is installed (enters) into the system, then it propagates its details (specification and location) to let other devices know about its presence and conditions to interact with. A key advantage of this approach is that a failure of one device would not cause the failure of whole system. Other advantages relate to the scalability and re-configurability of the system. Therefore, the distributed control system presented in this thesis is implemented using peer-to-peer communication between the embedded control devices.

![Figure 6.11: Peer-to-Peer approach for device interaction.](image)

To realise a peer-to-peer approach using Web Services based communication and SOA, a client-server mechanism is used in this thesis as shown in Figure 6.12(a). In this approach, each embedded device in a system comprises both client and server capabilities as shown in Figure 6.12(a). The client portion of device is directly associated with discovery services (i.e. WS-Discovery) and registration by sending a multicast discovery messages over UDP. Each client acts as an event sink to receive information (e.g.
state variables) from the server of other connected devices and passes this information to its local server. The role of server is to associate services metadata, locations provided for dynamic device discovery and access state variables information from clients. The server is also involved in control application for I/O device operations and publishes the information related to its state variables over the network. This information exchange (SOAP messages) between client and server is done asynchronously using a publish/subscribe approach as shown in Figure 6.12(a). In this approach, a client invokes the service on server and the server then send an acknowledgement message to the client. To bridge the gap between the abstract service interfaces described through WSDL and the SOAP messages, the DPWS stack automatically generate files called stub and skeleton respectively for client and server. The skeleton file is used by server for exchanging messages required for control operations, while the stub files provide remote service interface used by client to invoke service operations on the server side. More specifically, these are used to perform the marshalling and demarshalling of the exchanged messages. The term marshalling and demarshalling refers to translating the contents of SOAP-XML messages into native objects of client and server; and vice versa.

To implement this peer-to-peer approach, ControlBuild has been utilised in this research to integrate the server and client applications as shown in Figure 6.12(b). The publish and subscribe mechanism for Web Services based communication is established through the SOAP messages at runtime. ControlBuild uses the WSDL file to generate client and server communication modules, through the automated code generation of the stub and skeleton files using these WSDL file. In the ControlBuild editor, for each WS component with server capabilities, a DPWS function is automatically generated and can be used for service invocation. This DPWS function is generated in specific libraries folder as shown in Figure 6.12(b). These files are then converted into a binary program as explained above and this is then deployed into embedded device, in order to build the complete control application. This has been discussed in detail in Chapter 6.

In summary, ControlBuild provides a graphical means to generate component based control applications for distributed automation systems using Web Services based devices.
Chapter 6: Engineering Methods for Web Services

Figure 6.12: (a) A client-server model for implementing peer-to-peer device interactions in a distributed automation system and (b) its implementation in ControlBuild.
6.8 Summary

This chapter presented the importance of employing Web Services at device level, for building the CB automation systems presented in Chapters 3 & 4. Web services are commonly used in higher level business applications; therefore, if the same technology can be used in shop floor devices, it provides a technologically neutral platform for integrating component based automation systems at the shop-floor to the higher level business applications. For implementing Web Services and SOA at device level, DPWS is considered as the most suitable protocol. Using DPWS, the WS provide functions for hosting services on the control device, discovery methods, control application execution and event-notification services.

To design and deploy WS based control applications using DPWS protocol, a graphical engineering tool called ControlBuild provided by Geensys is presented in this chapter. ControlBuild is found beneficial as it includes a variety of graphical interactive editors for component-based design covering IEC 61131-3, C and electrical circuit diagrams, a set of component libraries and a simulation environment for control software testing & validation prior to integration and commissioning. A detailed discussion on using ControlBuild (particularly FBD) has been presented in this chapter. It presented the procedure of generating the required functionalities of the control device within the automation system using ControlBuild. For a distributed automation system comprising many devices, a peer-to-peer approach is used for device level interactions. These have been implemented in ControlBuild using a client-server model, through the automatically generated stub and skeleton files.

Thus, the research presented in this chapter explains the feasibility of applying Web Services at device level in CB automation systems. This creates a technology neutral platform, where devices from different vendors can interoperate via XML message passing and thus enhance system’s flexibility and re-configurability through effective plug-and-play discovery.
In Chapter 6, ControlBuild was introduced as a potential engineering tool for designing the CB automation systems using Web Services based SOA within the embedded devices at shop floor. This design approach is an innovative way of designing components using graphical engineering tool with Web Services interfaces. This chapter presents a test rig used for implementing the concept of CB automation system designed using ControlBuild. It also presents a selection of DPWS based embedded device and its implementation to form a distributed automation system. Finally, the chapter presents an example of designing a CB control application with WS interface using ControlBuild and deploying it on the selected device.

7.1 Introducing Ford-Festo test rig

The component based approach for designing distributed automation systems, as described in Chapters 3, 4 & 5 comprises two stages. The first stage is of designing the component based control applications for the distributed automation system; whereas, the second stage is to integrate/communicate these components using Web
Chapter 7: Component-Based Web Services Test Set Up

Services and Service Oriented Architectures. These have been discussed in detail in Chapters 4 & 5. However, this Chapter presents a test rig named Ford-Festo test rig designed for the practical implementation of CB concept at laboratory scale under controlled conditions. The test rig is located in the MSI laboratory of Loughborough University. The schematic of this rig is shown in Figure 7.1 and its physical appearance is presented in Figure 7.2. This test rig is a replica of Ford powertrain assembly line and the functionality of this automation test rig is deemed to be applicable to real industrial machinery and control applications experienced by the Ford Motor Company.

![Ford-Festo test rig schematic diagram](image1)

Figure 7.1: Schematic diagram of the Ford-Festo test rig (Kirkham et al., 2008a).

To practically demonstrate the CB approach for designing control applications, the Ford-Festo rig has been considered as a distributed system in current thesis. The test rig comprises various sensors and actuators, which are electrically or pneumatically controlled and accessible through the distributed I/O interface module of test rig as shown in Figure 7.3. The embedded devices hosting the control applications are used to control the test rig. These devices read the sensors and control actuators by connecting the I/O interface module of embedded device to the distributed I/O interface module of test rig. The selection of embedded device is explained in section 7.2.
Chapter 7: Component-Based Web Services Test Set Up

The component based distributed control applications are designed using Control Build, an automation design engineering tool provided by Geensys (Guerard, 2009b). This has been explained in Chapter 6. This tool supports various programming languages such as C and IEC-61131 for designing control applications. In current research, for designing the reconfigurable and reusable CB based control applications, the control logic of each component is designed using Function Block Diagram language of IEC-61131. In ControlBuild, FBD is a graphical way of designing the distributed control application for reconfigurable and reusable components. In order to design the CB automation system using Web Services and SOA for this test rig, the first step is to decompose the test rig (as mentioned in Chapter 5). It is explained in the following section.

7.1.1 Decomposing Festo test rig

The first step to design the CB distributed control system is to decompose the test rig using the architecture (Subsystem - Component - Element) explained in Chapter 5.

Figure 7.2: Physical representation of the Ford-Festo test rig located at MSI laboratory of Loughborough University.
This is reported in Table 7.1, in which the Ford Festo test rig shown in Figure 7.1 is considered as a system and is divided into four subsystems called stations in current research. These are: (i) a distributed hopper station, (ii) a buffer station, (iii) a processing table station, and (iv) a handling arm station. These are shown in Figure 7.1, where each of these subsystems comprises one or more mechanical components that are connected to elements (i.e. sensors and actuators) through a distributed I/O interface module (Figure 7.3) of the test rig. Each of these elements is represented by a finite state machine, with their functions represented by state variables. For simplicity, the finite state machines of elements of station 1 (distribution hopper and transfer arm) and station 2 (conveyor) are shown in Figures 7.4 to 7.6. The sequence of operations of test rig is defined by interlocking the states of components in the design phase, i.e. when designing the control application in ControlBuild. The information related to this state behaviour and interlocking of components using their finite state machines is proposed to be provided by the PDE tool. The designed control application of each component is then uploaded to an embedded device, which is responsible for the real-time control of distributed components within the subsystems. The design and integration of the CB system for this test rig is detailed in the following section.
### Chapter 7: Component-Based Web Services Test Set Up

#### Table 7.1: Decomposing Festo test rig.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Component</th>
<th>Element</th>
<th>Actuator</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Station 1</td>
<td>Distributed_Hopper</td>
<td>Eject_Cylinder</td>
<td>Magazine</td>
<td>Mag_Xfer_Ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Swivel_Drive</td>
<td></td>
<td>Gripper</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vacuum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Station 2</td>
<td>Conveyor</td>
<td></td>
<td></td>
<td>Workpiece_Available</td>
</tr>
<tr>
<td></td>
<td>Separator</td>
<td></td>
<td></td>
<td>Workpiece_at_Separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Workpiece_at_Conveyor_End</td>
</tr>
<tr>
<td>Station 3</td>
<td>Rotary_Table</td>
<td>Indexing_Rotary_Table</td>
<td>Workpiece_Available</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Component_Part_Checker</td>
<td>Checking_Actuator</td>
<td>Workpiece_Available_at_Checking_Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Drilling_Unit</td>
<td>Drill</td>
<td>Workpiece_Available_at_Drilling_Station</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Drill_Spindle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workpiece_Clamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Station 4</td>
<td>Handling_Arm</td>
<td>Arm</td>
<td>Workpiece_Is_Not_Black</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gripper</td>
<td>Workpiece_Receptable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gripper_Extend_Cylinder</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 7.4: State transition diagram of the elements of *distribution hopper*. 
Chapter 7: Component-Based Web Services Test Set Up

Figure 7.5: State transition diagram of the elements of transfer arm.

Figure 7.6: State transition diagram of the elements of conveyor.
7.1.2 Work flow process of Festo test rig

The second step is to define the work flow process of test rig, which means the sequence of actions performed by various mechanical components including elements of the test rig. The functionality of this test rig is representative of typical assembly line operations used to assemble automotive engines. However in this test rig, the engine blocks processed in real assembly lines are replaced by a plastic workpiece (WP). Similar to the engine blocks in assembly lines, this workpiece undergoes a number of tasks such as transferring, buffering, slot checking, drilling and sorting in test rig. The sequence of these tasks is shown in Figure 7.7 and detailed below:

1. In the first step, the workpiece is inserted into the magazine of distributed hopper. The ejector gets pneumatically activated and pushes the workpiece from magazine slot to the transfer position of hopper and waits for the transfer arm to pick it up and transfer it to the conveyor.

2. Each WP transferred by the arm is then moved to the separator using the moving belt system called conveyor. The separator on conveyor checks for the free slot on processing table and stops WP going further to the processing table if a free slot is not available (e.g. in situations when table is moving or there is already a WP in the input slot).

3. When the input slot of table gets free, WP is released by the separator for the rotating processing table.

4. The processing table moves the WP to component checker, which confirms that the work piece is positioned correctly before the drilling operation can be performed. If it is not positioned correctly, the WP will skip drilling and the control system will raise an alarm to the operator.

5. After performing the position checking operation on WP, if the WP is found located correctly then it is transferred to the drilling unit for drilling operations.

6. After drilling, the WP is then moved to the ejector of indexed processing table, which then pushes the WP to the buffer of handling arm unit.
Chapter 7: Component-Based Web Services Test Set Up

7. When the WP is located in buffer, the handling arm grips it up and transfers it to the exit slot. There are two slots: one for black WP’s and another for coloured WP’s. The gripper element of handling arm identifies the color of WP and helps transfers it into its specific color slot.

This workflow of the WP illustrates the sequence of machine operations of the test rig and is used to define the appropriate interlocks between the states of elements of test rig.

Figure 7.7: Workflow process of Festo test rig.
7.2 Selecting an embedded device

The next step is to make a selection of an embedded device, supporting DPWS based Web Services interfaces. DPWS is presented in Chapter 6, where it seems a very promising approach fully aligned with Web Services technology, for use in building automation systems. A key aspect of the DPWS protocol stack is that all messaging is based on the use of SOAP and WS-Addressing. Therefore, in order to design the CB automation systems using the Web Services and SOA, the selected embedded device should be based on DPWS protocol. This has been discussed in Chapter 6; however, this section presents the general architecture of a device built using the DPWS toolkit. This is shown in Figure 7.8, which presents an architecture of a DPWS protocol based device. In this architecture, the device runs two types of services: hosting services and hosted services. Hosting services are directly associated to a device, particularly used in the device discovery process. Hosted services are mostly functional and depend on their hosting device for discovery. The execution services, eventing services and discovery services shown in Figure 7.8 are predefined services along with the embedded SOAP 1.2 engine. These are generally provided as run-time libraries in the DPWS toolkit.

In addition to these, the device architecture presented in Figure 7.8 specifies the following set of built-in services:

- **DPWS description & discovery services:** These are used by a device to advertise itself and/or to discover other devices using plug-and-play device discovery. This is done by using a multicast discovery protocol to search for and locate other devices connected on the network. In this protocol, a client sends a multicast message to search one or more devices. Hosted services do not participate in the discovery process, but can be individually addressed (through their respective end point references) once the hosting device has been discovered. The multicast message is sent using the SOAP-UDP binding in order to minimize network traffic overhead.

- **Eventing services:** These services are used for allowing other devices to subscribe to asynchronous event messages produced by a given service. More specifically, it defines a publish-subscribe event handling protocol allowing one Web Service (event sink) to register interest (subscription) with another Web Service.
Figure 7.8: Architecture of a DPWS based embedded device (Jammes et al., 2007).

(event source) in receiving messages about events (notifications). Event notifications themselves are one-way messages, the content of which may include any data of any type. These are transported in the same way as any other SOAP message.

- **App services and events:** These are user-defined services and events, provided as user-written code and generated code in the DPWS toolkit.

- **Network interfaces:** There are two types of network interfaces: primary and discovery interface. The primary interface uses the standard SOAP 1.2 over HTTP binding, while the discovery interface uses SOAP over UDP and a multicast address to broadcast and listen to the discovery messages.

- **Metadata exchange services:** This is used to provide dynamic access to the metadata of a device hosted services.

Based on this architecture and the requirement of controlling the digital input/output
elements (sensor & actuator) of Festo test rig, a selection of device called Smart Terminal Block (STB) is made. This is discussed in the following section.

7.2.1 Smart Terminal Blocks

The controller device selected for Ford-Festo test rig is Advantys Smart Terminal Block (STB) module, supplied by Schneider Electric\(^1\). There are several models of STBs; however, in current research STB 2311 is used. A selection of STB 2311 was made, as it closely matches the digital input/output and industrial fieldbus (Ethernet) requirements of Festo test rig.

Advantys STB is based on mechatronics\(^2\) and is an assembly of distributed I/O module, power distribution module (PDM) and a network interface module (NIM) residing in a structure called an island, as shown in Figure 7.9 (Schneider, 2010). The island (Figure 7.11) functions as a node on a fieldbus control network. Its modules are mounted on one or more DIN (Deutsches Institut für Normung) rails and are connected using a bus running from beginning to end of each island. The island bus provides power distribution, signal sensing and power management to all compatible modules in the form of a wiring management system. The island’s primary segment can be extended for further modules; however, a bus termination plate with 120 ohm resistance is used to end the segment.

The islands can function on a variety of different open industry standard fieldbus networks. Among these are Profibus DP, DeviceNet, Ethernet, CANopen, Fipio, Modbus Plus and INTERBUS. STB supports a NIM for each type of fieldbus. In current research, the Ethernet based fieldbus\(^3\) is taken and the respective NIM is NIP2311 (Figure 7.10(a)).

The individual modules of STB 2311 is shown in Figure 7.10 and are described below:

\(^1\)http://www.schneider-electric.co.uk/ (URL last cited on 25th January, 2011).

\(^2\)Mechatronics is a process of installing the STB island with distributed I/O modules as close as possible to the mechanical field devices it needs to control. This will help reducing the installation time and cable costs for sensors and actuators, while increasing system availability.

\(^3\)STB support a 10/100 Base-T LAN, defined by the IEEE 802.3 Ethernet specification, with contention for 10/100 Base-T networks resolved by using Carrier Sense Multiple Access with Collision Detect (CSMA/CD). STB supports both Ethernet II and IEEE 802.3 frame formats; however, the Ethernet II is the default frame type with an auto selection of transmission rate of 10 or 100 Mbit/s.
Chapter 7: Component-Based Web Services Test Set Up

Figure 7.9: STB hardware module (Schneider, 2010).

- **Network Interface Module (NIP2311):** STB supports different types of field-buses and for each fieldbus, there is an industrial NIM. In current STB module, Ethernet is the fieldbus and the corresponding NIM is NIP 2311. It is shown in Figure 7.10(a) and is installed in the leftmost location in primary segment. There is only one NIM in an island. It acts as a gateway to the island bus, i.e., all communications to and from the island bus passes through the NIM. It is the only module on the island that is fieldbus dependent. The rest of the I/O and power distribution modules on the island bus function works exactly the same, regardless of the fieldbus on which the island resides. This makes the STB island advantageous of being able to select the I/O modules to build an island independent of the fieldbus on which it will operate.

- **Power distribution modules (PDT3100):** The role of power distribution module (PDM) is to supply power to the island. In current STB, the PDM is PDT3100 module, which distributes 24 VDC across the island bus. The PDM is installed directly to the right of NIM. It is shown in Figure 7.10(b).

- **I/O Modules:** The two modules on the right of PDM are the input and output modules respectively. Their role is to send and receive signals from the connected
devices. The input module is DDI 3725 and the output module is DDO 3705. These are shown in Figures 7.10(c) & 7.10(d).

To control the Festo test rig, multiple components may be embedded on each STB, depending on the decomposition of test rig. The STB I/O modules are connected to the I/O module of test rig which are further connected to its local sensors and actuators of its subsystem/station. STB I/O connectors are of removable screw or spring type connectors, as shown in Figures 7.10(c) & 7.10(d). STB modules can be removed or replaced (hot swap) when the system is powered on. Each STB 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 STB devices when operating as servers provide the DPWS control functionalities that are interfaced to the local input and output of the STB device. It is the application integration model proposed in this research, which aims to enable seamless application integration by embracing the WS interface and communication. The following section presents an example of programming STBs with ControlBuild.
Figure 7.10: (a) Network interface module, (b) power distribution module, (c) input module and (d) output module of STB 2311 module shown in Figure 7.9.
7.3 Programming STBs with ControlBuild

To understand the whole process of designing a CB application using ControlBuild and finally deploying it on STBs, consider an example of a conveyor belt shown in Figure 7.12(a). In this example, the conveyor belt starts or stops moving respectively when the sensor switches to ON or OFF. The duration, for which the belt keeps moving, can be controlled using timers. However, for simplicity, timers are not taken into this example. The steps taken to illustrate the complete process of designing and deploying CB application with WS interface for this example are mentioned below:

1. **Workflow process:** The first step is to illustrate the workflow process. It is shown in Figures 7.12(a) & 7.12(b). When the workpiece arrives, the sensor gets ON and the belt starts moving. When there is no workpiece the sensor remains
OFF and so the belt remains stopped. In simple words, the combination of work-piece and sensor can be represented by a simple switch as shown in Figure 7.12(c). When the switch is ON or OFF, the belt moves or stops respectively.

2. Decomposing the system: In the second step, the conveyor system in Figures 7.12(a) and 7.12(b) is divided into smaller subsystems and these are further divided into component and elements as explained in Chapter 3. In this case, for simplicity, the sensor is assumed as a component and is controlled by one device; while, the belt is controlled by another device. Both components are part of a subsystem named sensor-belt. It is shown in Figure 7.12(c), where the device 1 will read the status of sensor and passes its state to device 2, which then switches OFF (stops) or ON (moves) the belt accordingly. In return, the device 2 passes the states of belt (moving/stopped) to device 1, which is displayed on the output (named belt status LED) of device 1. This is done to explain how the states are exchanged over the network between components deployed on different devices using the client-server model.

3. State transition diagrams: The third step is to make the state transition diagrams of all the elements, and define the interlocking conditions using the workflow process. In this example, the belt is interlocked with a sensor element deployed on another device. A detailed discussion for implementing interlocking conditions in ControlBuild and STB will be presented in Chapter 8. The state transition diagrams of sensor and belt are respectively shown in Figures 7.12(d) and 7.12(e).

4. State variables table: The fourth step is to make a Table of all the state variables and assign the state numbers as mentioned in Chapter 4. In this case, the state variables with their state numbers are reported in Table 7.2.

<table>
<thead>
<tr>
<th>Component</th>
<th>Element</th>
<th>Number of states</th>
<th>State Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Sensor</td>
<td>2</td>
<td>1.2</td>
</tr>
<tr>
<td>Belt</td>
<td>Belt</td>
<td>2</td>
<td>3.4</td>
</tr>
</tbody>
</table>
Figure 7.12: (a)(b) Workflow process, (c) physical decomposition and state exchange, and state transition diagrams of (d) sensor and (e) belt of the conveyor system.
5. **Designing control application in ControlBuild:** The next step is to design the control logic of components using ControlBuild. ControlBuild is based on a model of assembly of components (MAC) where the application is divided into components and sub-components assembled graphically. In this example, the main application *sensor_belt* has two components: *sensor* and *belt* to be installed on two different devices 1 & 2 respectively. These are represented by a MAC in ControlBuild as shown in Figure 7.13. The parameters such as IFSensor and OFsensor of MAC in Figure 7.13 are Web Services parameters, explained in later steps. These are further divided into sub-components, in which control logic is defined using FBD as shown in Figures 7.14 & 7.15 respectively. In ControlBuild, the components and their FBDs can be designed in one application e.g. *sensor_belt*. However, the different components in an application can be deployed on different STB devices; say, *sensor* component on device 1 and *belt* component on device 2 in this example. The key steps involved in designing the FBDs for these components are:

- Each component is assigned with sub-components (FBD in this case), e.g. *sensor_logic* for *sensor* component.

- The state variables of each component are defined as mentioned in Table 7.2. For example, the sensor has two states OFF & ON. It is represented by *Sensor_OFF* and *Sensor_ON* and is selected by a module *sel_int* depending upon the status (*IO_sensor*) of the sensor (Figure 7.14). The output of *sel_int* module represents the present state of sensor and is passed to the variable *Sensor_State_OUT*. This can be passed further to different components through the clients, as mentioned below in next step.

- After defining the states of variables, the control logic of components is designed using FBD in ControlBuild. This is based on workflow process and involves the state exchange between components. For example, for component *sensor*, the logic is defined in sub-component *sensor_logic* as shown in Figure 7.14. The sensor state is passed to the *belt* component through the variable (*Sensor_State_OUT*), while the *belt* status is received in the form of states using the variable *Belt_State_IN*. This is passed to a state comparator...
Chapter 7: Component-Based Web Services Test Set Up

Figure 7.13: Designing sensor and belt components (Mac) in ControlBuild.

(eq_int), which checks the state of belt and gives the logical HIGH (Output_high) or LOW (Output_low) output as shown in Figure 7.14, representing the belt moving or stopped respectively on the LED display, through the variable IO_Belt_Status. Similarly using the workflow process for belt component, the FBD of belt component is designed as shown in Figure 7.15. In this case, the sensor state is received through the variable Sensor_State and is passed to the state comparator (eq_int) to give the logic HIGH (belt moves) or LOW (belt stops) through the variable IO_Belt_Status. Depending on this logical HIGH or LOW, the (sel_int) module assigns the state value to the variable Belt_State which is passed to the sensor component. After designing the FBDs, the ControlBuild automatically generates PLCopen XML file for each of sensor & belt components seperately as reported in Appendices A.1 & A.2 respectively. These files will be later used by ControlBuild editor to finally generate the executable file to be installed on the STB device.
Figure 7.14: Sub-components, FBD of sensor component displaying belt state received from belt component in ControlBuild editor.
Chapter 7: Component-Based Web Services Test Set Up

Figure 7.15: Sub-components, FBD of belt component interlocked with sensor state of sensor component in ControlBuild editor.
6. **Defining the Web Services interface in ControlBuild:** After designing the control logic, the next stage is to design the Web Services interface of components in ControlBuild. This is designed individually for each component (e.g. *sensor* and *belt*), for the purpose to provide a pathway for interactions with the same fieldbus applications/devices. This is also done graphically in ControlBuild. The WS interface of *sensor* and *belt* components are respectively shown in Figures 7.16(a) & 7.16(b), where the *IFSensor* and *IFBelt* are the WS interface functions, and *IPSensorParam* and *IPBeltParam* are input parameters received as state information from other components over the network. *OFSensor* and *OFBelt* (not shown in Figure 7.16) are the response functions used for return receipt in two-way messaging. These are shown in Figure 7.13. ControlBuild automatically generates a WSDL file for each of these WS-interface as reported in Appendices A.3 & A.4. This contains all the above information regarding the WS interface of a component and is used to generate the final plcopen file.

7. **Designing the client-server CB application in ControlBuild:** As discussed in Chapter 5, a client-server architecture is used to integrate components distributed on different devices. This is also graphically designed in ControlBuild as shown in Figures 7.14 & 7.15 respectively for *sensor* and *belt* components; installed on devices 1 & 2 respectively. It should be noted that when designing the client-server model in ControlBuild, the client of components (e.g. *belt*) distributed on other devices is designed in the same window of component (e.g. *sensor*) under designing. More specifically to create a client for any server component, a DPWS function is simply dragged and dropped into the client component. This DPWS function is automatically created for each WS component with server capabilities. This is coded in C language and is reported in Appendices A.5 & A.6 for *sensor* & *belt* components respectively. An enable function is used to enable client for passing the state variables to its server operating on other device (Figures 7.14 & 7.15). In one-way messaging, it enables the client sending messages and other state variables to its server component, while in two-way messaging, it enables client sending an additional return receipt. ControlBuild automatically generates a C interface file for client including the skeleton file as reported in Appendices A.5 & A.6.
Figure 7.16: Web Service interface of (a) sensor and (b) belt components in ControlBuild.
8. **Defining embedded configuration and hardware architecture:** In this step, each component and its associated parameters are configured and the hardware architecture is defined to make the designed control application and the device holding this application unique in the network. This is done to avoid duplication in the network. In ControlBuild, this is done by generating an embedded configuration file that assigns each device, component and its variables with a unique identification number called UUID to resolve the same name duplication. For example, the UUIDs for sensor component is mentioned in Appendix A.9. Regarding the architecture of device, ControlBuild generates an embedded tree view (Figure 7.17(a)), which shows all the components and their parameters. It is shown in Figure 7.17(a), where it shows two types of devices: targeted device and the organic device. The targeted device is the actual physical device on which the application needs to be deployed; however, the organic device is the virtual device used by ControlBuild for the purpose of simulating and testing applications prior to deployment. More specifically, organic device corresponds to a WS component. For example in *sensor.belt* application, STB1 and STB2 are the target devices and STBO1 and STBO2 are the organic devices.

9. **Generating the I/O configuration file for devices:** This step involves generating the I/O configuration of the device using the ‘WS Utils’ droplist shown in Figure 7.17(a). The term I/O configuration refers to mapping component I/O parameters to the I/O modules of the physical device. This is done by generating the I/O configuration files called ioconf files. The ioconf files are simultaneously generated with the pcleopen file by dyndep generator in ControlBuild. These ioconf files contain information regarding the device’s physical input/output lines mapped to the software components. These are automatically generated in ControlBuild; however, these can be manually altered subject to the requirements of user. The ioconf files for devices 1 & 2 are shown in Appendices A.7 & A.8 respectively.

10. **Generating the device specific configuration files:** STB uses the binary instructions for its operation. Therefore in this step, ControlBuild generates the final binary version of configuration files to be deployed on devices. Firstly, it combines the XML file, WSDL file and the client C interface file (generated in
above steps) to generate the PLCopen file and then it internally converts it into a set of binary instructions before deploying on the STB. This is done by dyndep generator.

11. **Deployment of components:** The final step is to deploy the above generated files (distributed components) on physical devices. It should be noted that in ControlBuild, a WS component is loosely coupled to the device hosting it. This means that the application can be designed independent to a particular device. It will be decided at this stage, that the designed application needs to be deployed on which device. For example, the plcopen file generated for `sensor` can be deployed on either of devices 1 or 2. This is very beneficial in designing control applications and is called ‘hardware abstraction’ between the WS components and physical devices. The deployment process is done through a manager function in ControlBuild. It displays the list of devices available on the network uniquely identified by their metadata and serial number as shown in Figure 7.17(b). A selection of device(s) is made from the list and the manager deploys the DPWS configuration files and the binary files on the selected device as shown in Figure 7.17(b).

12. **Device management:** Finally, each STB device has a dyndep configuration embedded webpage, which is accessible through a tool called Materna/DPWS explorer as shown in Figure 7.18. Materna explorer is used to find and address DPWS devices i.e. it displays all the STB devices connected on the network as well as the components deployed on these devices. It is used to manage (delete and change) WS management services and parameters like serial number of STB. For example, to upload the new configuration in the device, the previous components are first deleted using its webpage in the Materna explorer.
Chapter 7: Component-Based Web Services Test Set Up

Figure 7.17: (a) Embedded tree view with WS utility and (b) Upload manager in ControlBuild.
Chapter 7: Component-Based Web Services Test Set Up

7.4 Summary

In summary, this chapter presented the layout of a test rig named Ford Festo test rig located at MSI laboratory of Loughborough University. This test rig mimics the operations of assembly lines of Ford and is thus used to determine the feasibility of using SOA and Web Services based CB design approach for distributed automation systems. It also presents a selection of DPWS based device for controlling the test rig. A selection of Advantys STB provided by Schneider is made and its integration with ControlBuild is presented. ControlBuild was introduced in Chapter 5, as a potential engineering tool for designing the CB automation systems and using Web Services based SOA within the embedded devices at shop floor. This design approach is an innovative way of designing components using graphical engineering tool, providing a platform to be integrated with higher level business applications using Web Services. Finally, the chapter presents an example of designing a CB control application with WS interface using ControlBuild and deploying it on the selected STB device. In next chapter, an investigation is made on implementing Web Services on STB devices.
within the component-based design approach on the Ford-Festo test rig, in terms of the performance and reliability of message passing between control devices under soft real-time constraints. The modularity and degree of re-configurability of the designed system using this WS control architecture are also assessed.
Chapter 8

Implementation & Evaluation

This chapter focuses on the implementation of the developed concept of integrating SOA-WS and Component-Based technology on the Ford-Festo test rig, described in Chapter 7. A comparison is made with the earlier studies done by MSI research group. The performance of this approach is evaluated by analysing and discussing various parameters in relation to the communication mechanisms, packet structure and sizes and Web Services message delivery time. Finally, the chapter ends making an assessment in meeting the requirements such as reusability and re-configurability of automation systems.

8.1 Test setup

To determine the feasibility of proposed SOA-WS based CB design approach, the said approach is implemented on the Ford-Festo test rig using the layout shown in Figure 8.1. As mentioned before, similar studies have also been made by the MSI research group of Loughborough University. Its details are presented in Phaithoonbuathong (2009) and the layout of test set up used in these studies is shown in Figure 8.2. The details of the Ford-Festo test rig is presented in Chapter 6; however, it is reproduced here briefly for explaining the layout of test setup. The test rig comprises of four subsystems called stations. The tasks performed by each station are controlled by
Chapter 8: Implementation & Evaluation

a controller node called device, which encapsulates software based control applications. In current research, STB (details presented in Chapter 7) is used as the embedded microprocessor based controller device; while Field Terminal Blocks (FTB’s) were used as controller devices in the research studies done by Phaithoonbuathong (2009). A comparison of the key features of STB and FTB is done in Table 8.1.

Table 8.1: Comparing key characteristics of FTB & STB.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>FTB</th>
<th>STB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>STR912 @ 96 Mhz</td>
<td>Coldfire 5208 @ 133 Mhz</td>
</tr>
<tr>
<td>RAM</td>
<td>96 Kb</td>
<td>Up to 8 Mb</td>
</tr>
<tr>
<td>Memory</td>
<td>512 Kb FLASH</td>
<td>Up to 8 Mb FLASH</td>
</tr>
<tr>
<td>Number of I/Os</td>
<td>16 digital configurable I/Os</td>
<td>up to 512 configurable I/Os</td>
</tr>
</tbody>
</table>

In test setup shown in Figures 8.1 & 8.2, each station is controlled by a single STB/FTB device. However, these devices may contain multiple software based components depending on the decomposition of station and the tasks performed by it. Each station has its own I/O channels, which are further connected to I/O connectors of STB/FTB as shown in Figures 8.1 & 8.2. This is done using the industrial standard I/O module; whereas, each I/O module is connected to local sensors and/or actuators of its station. There is no I/O cross connection between stations. To establish the Web Services based control systems network, a unique static IP address and MAC address is assigned to devices (STB/FTB). In Phaithoonbuathong (2009), a centralised orchestrator based approach was used; however in present studies, a choreography based peer-to-peer communication approach is used to interact the devices. Regarding system operation, the STB/FTB devices when operating as servers provide the DPWS control functionalities that are interfaced to local input and output of the STB/FTB device. The key differences between the two test setup are:

1. As shown in Figure 8.2, the execution of 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. However in current CB approach shown in Figure 8.1, the STB’s act as autonomous devices and use a
client-server model in association with a peer-to-peer communication approach, to initiate the execution of DPWS. This is beneficial as it eliminates the need of central orchestrator on which the whole system depends; and thus, reduces the risk of single point of failure.

2. As can be seen in Figures 8.1 & 8.2, eliminating central orchestrator has made a significant amount of reduction in hardware and associating complexity. This makes the overall system more simple, robust and easier to operate.

3. In research reported in Phaithoonbuathong (2009), an application integration model was used to enable seamless application integration by embracing the WS interface and communication. As shown in Figure 8.2, 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 and central orchestrator. However in research presented in current thesis, the devices such as HMI can be directly connected with STB’s using the switch. However, this research part is an early work; and the full implementation of common SOA middleware for seamless WS integration is on going work with the project collaborators.

8.1.1 Operating system & software architecture

The operating system and software architecture of FTB and STB is shown in Figures 8.3 & 8.4 respectively. Both architectures can be divided into four layers, which are very similar with the key difference is the logic engine and hardware. Physically, the lowest layer comprises of hardware and is different for both STBs and FTBs. The specifications of hardware used in the lower layer is mentioned in Table 8.1. However, the function of hardware is same for both devices, other than the specification. This hardware layer deals with the 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. The next layer comprises of RTOS (e.g. Quadros for STB) and communication stack (e.g. developed using Trek for STB), required to support the development of real-time applications and communications (e.g. TCP/IP/UDP) on the control device. The IP and MAC address of
Chapter 8: Implementation & Evaluation

Figure 8.1: Layout of test setup used on Ford-Festo test rig to implement a WS based fully distributed automation system.
Chapter 8: Implementation & Evaluation

Figure 8.2: Layout of test setup used on Ford-Festo test rig to implement a WS based automation system using a centralised orchestrator (Phaithoonbuathong, 2009).

the device (STB/FTB) required for communication purposes, is assigned during the device set up and configuration operations. The third layer is the abstraction layer, which is used to bridge the gap between the DPWS stack and the lower level RTOS and communication stack. The top layer is the DPWS application layer which further comprises of many sub layers and services. The deployment of Web Services occurs at the DPWS application layer. All of these software layers have been implemented and debugged using the CodeWarrior development software in the ARM Realview Development Suite. For FTB’s, 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 (Phaithoonbuathong, 2009). However, for STB’s these are already embedded and, thus, do not need to be specifically written for individual devices.
Figure 8.3: OS and software architecture of FTB (Phaithoonbuathong, 2009).

Figure 8.4: OS and software architecture of STB.
8.2 Developing DPWS in STBs using ControlBuild

In the test setup shown in Figure 8.1, each station of Ford-Festo test rig has a controlling device (STB), which acts as a container for WS interface based components. ControlBuild is used to design these components. ControlBuild is based on a recursive component model, in which an application is divided into components and sub-components, assembled graphically. This means each STB may be responsible for the operation of multiple components within the subsystem. Figure 8.5 shows the logical view of design approach used for defining the WS interfaces and designing the server and client components. In this model, a WS component is considered as a logical device hosted by a physical device; and therefore, this component is able to be discovered/to discover other components. It has access to the physical device resources and its comportment is managed by the IEC code created in ControlBuild. For this purpose of accessing the resources, DPWS interface is required. The development of DPWS for a subsystem starts with defining the component names, element names, element state variables and operations; and generating the corresponding WSDL file for the whole subsystem. It is to be noted that WSDL file is automatically generated in ControlBuild. Moreover, the architecture of ControlBuild allows an easy decomposition of components, and allows a direct and graphical links between Web Services concepts (i.e. request/response messages). In the implementation of the Ford-Festo test rig, each station has its own WSDL script for the DPWS application. The WSDL file of a subsystem defines the message request and response variables, a device service name and location. The WSDL syntax provides a grammatical structure and building block for XML documents, as shown in example in Appendix A.3. This WSDL file is later used to generate the stubs and skeletons, required for developing the server and client applications between the STB devices.

8.2.1 Implementing DPWS interface to create server/client on STB device

In ControlBuild, the low-level device logic/code is encapsulated in a top level WS interface. Its DPWS services can be invoked in other components through service interface, as shown in Figure 8.5. If, say, two STBs communicate in a peer-to-peer manner, then each STB device contains DPWS server interface for a server and a
DPWS service call functions for a client. This is shown in Figure 8.5. The actual control logic and publication variables are encapsulated in DPWS server application. For example, the DPWS interface for establishing client-server relationship between STB1 & STB2 devices is shown in Figure 8.5. The server component deployed on STB1 can be invoked in any other device by simply dragging its DPWS function (stored in library in Controlbuild) in other component. Any state change variables on station 1 will be published to station 2; which can be further used in control logic of station 2 to meet the interlocking conditions. Station 2 acknowledges the message reception by generating a response variable. The whole communication is done through Ethernet connection; however, for clarifying the interchange of logical message between STB1 & STB2, different lines are shown in Figure 8.5. The two devices communicate in a fully distributed approach with two way request and response messages.

This adoption of Web Services at the device level benefits 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. However, 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. The details of the test rig operation and client-server event interaction are presented in the following
8.2.2 Establishing client-server interactions

This section explains the client-server interactions within STB’s, communicating in a peer-to-peer fashion on the Ford-Festo test rig. To establish the client-server interactions, an event-driven communication architecture based on publish-subscribe approach is employed in this research. This decision is made from literature review; where, this communication approach is seen as the best suited to automation systems with limited resource embedded devices and requires good performance and utilisation of the network. In this 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.

A typical example to explain the physical implementation of the event-based client-server model is shown in Figure 8.6. For simplicity, two stations (1 & 2) of the test rig set up are considered in Figure 8.6. Station 1 and station 2 are deployed on STB1 and STB2 respectively. The server of station 1 (deployed on STB1) controls the two components: hopper and swivel arm in accordance to the pre-defined interlocking conditions. Similarly, the server on STB2 controls the two components conveyor and separator of station 2. The two stations 1 & 2 are interlocked with each other. The client-server interactions within STB devices is established as shown in Figure 8.6. It is to be noted that a client and its server can’t be hosted by the same device. Therefore practically, a feedback approach is used in the current research. In this approach, the client of a STB device (say STB2) is deployed on the previous device (STB1) in the loop.

For setting up event-based communication in example shown in Figure 8.6, the client (C2) on STB1, starts with subscribing to the pre-defined interlocked state(s) of station 2 (deployed on STB2). This subscription process follows after the device boot up and client-server initialisation process. Following this subscription process, the state publication process starts. In the publication process, if any I/O states on STB2 change then these are published for client C2, as shown in Figure 8.6. The client C2 then subscribes to these states and passes these to server S1 which then take appropriate action based on the state transition behaviour of the controlled elements (i.e. ejector or swivel arm). For instance, assume swivel arm of station 1 is interlocked with a sensor of conveyor component of station 2. Any change in sensor state on
Chapter 8: Implementation & Evaluation

Figure 8.6: Client-server interactions within the STB devices connected to form a feedback control system.

Station 2 is published to station 1. The swivel arm subscribes to this state and uses it to meet the interlocked state transition conditions, required to take appropriate actions. Similarly, further stations can be connected using this approach. It is to be noted that for Ford-Festo test rig, the states of a station are interlocked with the states of next station in order to make a feedback control system. More specifically, the states (if any) of the (i+1)th station are fed back to the ith station and are invoked through client C(i+1). This sequence of interlocking is defined specifically for Festo test rig; and it can be changed to any parallel operations as well.

In theory, to improve the system robustness and provide event synchronisation, an acknowledgement is sent back for every event message and DPWS function call. In situations of not receiving an acknowledgement message, a fail-safe routine implemented on the server and client applications is activated to suspend specific components from further operations that may cause damage to the control system. However in practise, for STBs, this feature (to date) is still in development with vendor. Therefore, this is not discussed and implemented in the research reported in this thesis. This can be
considered as future work. However, the client implemented on STB is aware of its associated server failure, and if this situation occurs, the communication messages are not further sent back to the IEC engine. A faulty interface is reported and no further messages are issued.

**8.3 DPWS performance analysis**

The performance of DPWS based network is evaluated as follows:

**8.3.1 DPWS-SOAP message structure**

The first thing is to determine the impact of DPWS-SOAP protocol on system performance. For this purpose, a network protocol packet analyser called Wireshark was used to determine the SOAP message structure. This tool offers the ability to capture messages on a LAN network and transforms the packet from binary into both Hex form and a XML based readable format. In the Ford-Festo test rig set up, the Wireshark utility was installed on a PC connected on the same subnet of STB devices.

<table>
<thead>
<tr>
<th>Payload (1460 bytes (Max))</th>
<th>SOAP Message</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Header (20 bytes)</td>
<td>Source Port</td>
<td>(04 4a 26 94 2b 53 5f 30 03 88 e5 5d 00 10 00 00 0c 00)</td>
</tr>
<tr>
<td>UDP Header (8 bytes)</td>
<td>Destination Port</td>
<td>(26 94 00 00 14 98 01 00 00 01 00 15)</td>
</tr>
<tr>
<td>IP Header (20 bytes)</td>
<td>Sequence No.</td>
<td>(7206844144)</td>
</tr>
<tr>
<td>Ethernet Header (14 bytes)</td>
<td>Acknowledgement No.</td>
<td>(c3 88 e5 5d)</td>
</tr>
</tbody>
</table>

![Ethernet TCP/UDP packet structure.](image)

The packets were captured and their structure is shown in Figure 8.7. This packet structure can be divided into 4 layers; where the content of each layer conforms to the TCP/IP standard. The first layer comprises of the data or payload i.e. the SOAP message in the XML format. It is to be noted that the payload size has a maximum limit of 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 second layer is the TCP/UDP header layer and is made up of source, destination port, sequence and acknowledgement number. In the third layer from top, the source and destination
IP addresses are mentioned. Finally, to precisely locate the destination and source of packet, the destination and source MAC address is added to the final bottom layer; before the packet is transmitted. The content of a SOAP message is in XML format which is decoded by the receiver to extract the information into an appropriate format. In context of the test rig, the DPWS-SOAP 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). This SOAP message structure, predefined with the WSDL description, is encoded and decoded by the stub and skeleton files on the implemented DPWS application. This is done to extract the required information into the format processed by the control application.

8.4 Ethernet TCP/IP network communication

In the proposed SOA-WS based CB approach, the DPWS communication is required to perform: (1) Service invocation and (2) Event notification operations. These are required to support publish-subscribe communication between devices. The details of packets exchanged to establish the DPWS connection, during the test-rig operation of service invocation method are presented in Figure 8.8. In Figure 8.8, device 1 (represents as Host A) calls the provided service on the device 2 (Host B) to operate the actuator. To achieve the service invocation, the synchronisation of sending and receiving packets between Host A and Host B over the Ethernet network is represented by: (i) establishing the connection between hosts using three-way handshaking and (ii) sending/receiving packets through TCP byte oriented sequencing protocols. These are explained below:

- **Three-way handshaking:** The principle of three-way handshaking is shown in Figure 8.8. In the three-way handshake mechanism, the two systems (say, Host A & B) attempting to initiate communication, can negotiate one connection at a time independently of each other. This is beneficial as it assures message is transmitted and received in correct order. In three-way handshake, this is done by associating a sequence and acknowledgment number for every message as shown in Figure 8.8, and explained below:
Chapter 8: Implementation & Evaluation

- Host A initiates establishing connection with Host B, by sending a TCP synchronisation packet. In this packet, the TCP header has SynFlag set to 1, acknowledgement number to 0 and a unique sequence number to initiate the connection.

- Host B replies with the synchronised sequence (newly generated sequence number) and acknowledgement number as sequence number received from Host A incremented by 1.

- Finally, Host A sends the third packet to Host B, with the synchronised sequence: Sequence number (sent) = Acknowledgement number (received) and Acknowledgement number (sent) = Sequence number (received) + 1, in order to establish the connection.

**Figure 8.8:** Ethernet TCP/IP network communication.

- **Byte-orientated sequencing protocols:** After establishing the connection, the next thing is to exchange messages. This is done through byte-oriented sequencing protocols. This is explained below:

  - Host A transmits messages with sequence and acknowledgement numbers, to ensure that message synchronisation is in the correct order. A record of
the above numbers is kept in the host till the acknowledgement is received back from the receiver. After acknowledgment, the packets are discarded.

- The sender then negotiates the termination of the one-way connection from Host A to Host B by sending the next packet, in which the Fin Flag in TCP header is set to 1. Host B acknowledges receipt of this packet. Host A keeps a record of the last sent message, thus it knows the value of the next expected sequence and acknowledgement number. If the acknowledgment is not received in time or not in the right sequence then re-transmission is required before the previous message from Host A is discarded. After this, the connection from Host A to Host B is terminated; however, Host B is still connected to Host A.

- Host B then sends packet to Host A, followed by a request of the connection termination. Host A replies to Host B with acknowledgement of the receipt packets and terminates the connection between Host B and A, in order to free the connection for other hosts.

It is to be noted that the packet synchronisation in the state change notification is also achieved in the same manner as in the service invocation method.

8.4.1 Ethernet packet delivery time and I/O reaction time

In the Ford-Festo test rig, a protocol analyser was installed using PC to determine network performance during the test-rig operation. Various SOAP messages for DPWS operations (i.e. state information, service call functions, discovery probe and metadata return) were noticed. In one DPWS operation, 9 packets were sent or received during TCP transmission. 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). From the results obtained from protocol analyser, SOAP messages were found in the size range of 750 to 1514 (max) bytes; with the delivery time of one packet varies from 0.05 to 1 millisecond. This packet size (750 to 1514 bytes) is same as reported in Phaithoonbuathong (2009).

The second important parameter is I/O response time. It is to be noted that the DPWS processing time (i.e. parsing-encoding/marshalling and decoding/de-marshalling) is not mentioned in research reported in this thesis. It is because the vendor has not
provided access to the encapsulated IEC engine code of the STB device. Therefore, only the I/O response time is measured, which includes the time required for DPWS event notification and service invocation. The I/O response time noticed for DPWS enabled STB device is 28.7 milliseconds, which is less than that of 31.06 milliseconds, as reported in Phaithoonbuathong (2009) for orchestrator based control system implemented using FTB devices. This is because additional time is required to execute an application on orchestrator while using FTBs. This means that the proposed STB devices based control system has a faster I/O response time for the similar size of SOAP messages and network speeds.

From the above results it is concluded that the proposed SOA-WS based CB approach, implemented using STBs is suitable to apply in automation industry. It is because:

- Ford motor company (co-sponsor of this project) specifies that the I/O response time between inter-connecting controllers in production must meet a soft real-time criterion under 30 milliseconds in order to meet the normal performance of the machine assembly process. The average response time of Web Services interaction between I/Os on the Ford-Festo test rig was estimated as 28.7 milliseconds. This meets the required timing criteria. In addition, the test rig implementation has demonstrated the reliability of Ethernet message packets, with the TCP/IP packet synchronisation. The additional Web Services message acknowledgment of the DPWS application guarantees that the DPWS message is sent and received.

- Another factor is the size of DPWS resource and memory usage on the device. This limits the device capability and number of DPWS components that can be installed on a single device, which further affects the development costs of control system. In STBs, the DPWS library is encapsulated and a memory space more than 7MB (depending on model) is provided by vendor for programming for user. This space is capable of supporting many components (e.g. designed using ControlBuild in current case) and supports the client-server model with peer-to-peer communication approach, leading to a fully distributed control system. However, FTB’s provide less memory of around 140KB for user programming applications and, therefore, are not able to interact in a peer-to-peer fashion and
needs an orchestrator. From the literature, both the end-user and industrial machine automation suppliers prefer a fully-distributed control system, enabled by peer-to-peer automation devices, rather than a PC-based orchestration and centralised PLC based control system. These requirements are satisfied by STBs as these have a larger memory to host both the DPWS server and client application of multiple components installed on the same device.

- There are many varieties of PLC products supplied by various vendors. For a typical PLC, FTB and STB device, the approximate cost noticed in Dec, 2011 on various UK based suppliers is 825/Unit, 224/Unit and 150/Unit respectively. From these cost figures, STB devices are found much cheaper than the conventional PLC and FTB devices. In addition, it is found from the specifications that STB’s have large memory space and can accommodate a large amount of I/O’s (e.g. Modicon NIP 2311 STB supports 256 discrete and 256 analog). However, FTB devices is designed to handle I/O functionality in smaller volumes, with 16 channels for the distributed automation environment. This means that the STB 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. This shows that the implemented WS approach is particularly cost efficient and is more scalable.

In the next section, the agility features of the proposed SOA-WS based CB approach are investigated.

8.5 Assessment in meeting agility features

This second part of current chapter is focused on assessing the agility abilities of proposed CB based SOA-WS approach implemented using STBs and ControlBuild. As discussed before, the key requirement of an agile automation system is its ability to reconfigure (both mechanical and control components) in minimum time and cost over the machine lifecycle. This agile feature is required to support new products and can be evaluated qualitatively. This can be done by determining the ease with which the system can be reconfigured for any desired changes in respect of time, effort and cost. More specifically, it takes into account the number of required activities (i.e. mechanical and
software changes as well as the effort to make the new machine configurations) involved in making changes and/or substituting modules (components) of the present control system to implement new specifications. To determine the degree of re-configurability and assessing other agile features of proposed SOA-WS based CB design method, a series of test scenarios have been implemented on Ford-Festo test rig. However for simplicity and ease of understanding, the following three test scenarios are presented in this thesis. As mentioned before, the actions performed on this test rig replicates the real powertrain assembly lines of Ford; therefore, the results of following tests will be applicable to real process lines. These experimental case studies are detailed below:

8.5.1 Case 1: Adding a component

This case is implemented on station 2 of Ford-Festo test rig. Initially, station 2 of test rig comprised of component named conveyor, with two elements conveyor belt and workpiece available sensor (abbreviated as WP1). The function of station 2 was that the conveyor belt turns ON for 4 seconds, when the work piece arrives at WP1 sensor position (i.e. when WP1 is FULL). However in this case, a component named Separator is installed on station 2 (conveyor belt) as shown in Figure 8.9.

The Separator component comprises of a binary sensor named Workpiece Available Sensor 2 (WP2) and a pneumatically controlled actuator named Separator as shown in Figure 8.9. This component with Separator and WP2 sensor elements is installed to control the work piece traffic (i.e. the number of work pieces at one time) on the conveyer belt and whole system is referred as buffer unit. The step-by-step actions performed to install this component physically and control it through application control logic encapsulated in STB device are described below:

1. The new separator component was physically installed on the conveyor belt as depicted in Figure 8.9. In particularly, the WP2 sensor element was installed on the left of Separator element in position. The WP2 sensor is used for changing the Separator’s state from static to dynamic, i.e. when WP2 is FULL, the separator extends.

2. The pneumatic controlled input lines of the separator were connected to the air pressure.
3. The input lines of WP2 sensor and Separator elements were connected to the available channels of the I/O module of station 2 of test rig. These were further connected to the I/O module of STB device (corresponding to station 2) and as described in the I/O configuration file (See section 6.7 and Appendix A.7 for example on generating I/O configuration file.)

4. In the next step, the states of Separator and WP2 sensor elements of the Separator component were defined. In this case, Separator element has four states as mentioned in Figure 8.10, so the STD of Ejector element (See section 5.3 for details.) was used. Also, the WP2 sensor element is a binary sensor, so the STD of any binary sensors (previously described in examples detailed in section 5.3 of Chapter 5) can be used. These are shown in Figure 8.10. This usage of STDs of elements with same states supports re-usability of CB design approach, and thus makes the design procedure simple, faster and easier. The states of these
elements are defined in Table 8.2.

5. The state transition tables (Tables 8.3 & 8.4) were made, which report all the combinations (for the desired workflow), of states of control elements of Separator component and Conveyor component of station 2. The procedure of defining the combinations of state transitions and output for the required workflow actions/events has been detailed in section 5.3 of Chapter 5.

6. In the next stage, a DFSM model of station 2 was designed using the STDs of elements. This is shown in Figure 8.11 and represents the process workflow including the interlocking conditions of components of station 2.

7. The functional operation of WP2 sensor and Separator elements are respectively same as binary sensors (e.g. Magazine sensor) and pneumatically controlled actuators (e.g. Ejector), previously detailed in Chapter 5. Therefore, the FBDs designed for these elements in Chapter 6 were reused here by just changing the names and the states as per Tables 8.3 & 8.4. This new Separator component in ControlBuild is shown in Figures 8.12 & 8.13. This is the benefit of reusing FBDs in ControlBuild for elements with same operational functions.

8. The I/O files were generated in ControlBuild as detailed in section 6.7 of Chapter 6 and the new component software is debugged, simulated and validated for testing purposes prior to its deployment on the device.

9. Finally the new software is deployed on the device.

Table 8.2: Assigning state numbers to control elements of conveyor and separator components.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Number</th>
<th>Number of States</th>
<th>State Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component 1: Conveyor</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WP1 Sensor</td>
<td>1</td>
<td>2</td>
<td>1.1,1.2</td>
</tr>
<tr>
<td>Conveyor belt</td>
<td>2</td>
<td>2</td>
<td>2.1,2.2</td>
</tr>
<tr>
<td><strong>Component 2: Separator</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WP2 Sensor</td>
<td>3</td>
<td>2</td>
<td>3.1,3.2</td>
</tr>
<tr>
<td>Separator</td>
<td>4</td>
<td>4</td>
<td>4.1,4.2,4.3,4.4</td>
</tr>
</tbody>
</table>

*Note: WP stands for Work Piece*
Chapter 8: Implementation & Evaluation

Figure 8.10: STD of sensor and separator control elements.

Figure 8.11: A DFSM model of station 2 using STDs of control elements.
Table 8.3: State transition table reporting all the combinations of states of control elements of conveyor component.

<table>
<thead>
<tr>
<th>WP1 Sensor</th>
<th>Conveyor Belt</th>
<th>Interlocking with Component 2 (Separator)</th>
<th>Output</th>
<th>Action/Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>2.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.2</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.1</td>
<td>3.1</td>
<td>HIGH for 4 seconds</td>
<td>Conveyor Belt ON for 4 seconds</td>
</tr>
<tr>
<td>1.2</td>
<td>2.2</td>
<td></td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.4: State transition table reporting all the combinations of states of control elements of separator component.

<table>
<thead>
<tr>
<th>WP2 Sensor</th>
<th>Separator</th>
<th>Interlocking with Component 1 (Conveyor)</th>
<th>Output</th>
<th>Action/Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>3.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>3.2</td>
<td>2.2</td>
<td>HIGH for 1 second</td>
<td>Extended for 1 second</td>
</tr>
<tr>
<td>4.1</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>3.1</td>
<td>2.2</td>
<td>HIGH</td>
<td>Extended</td>
</tr>
<tr>
<td>4.2</td>
<td>3.2</td>
<td>2.2</td>
<td>HIGH for 1 second</td>
<td>Extended for 1 second</td>
</tr>
<tr>
<td>4.2</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>
8.5.1.1 Results & observations

The key outcomes and observations made from this experimental case study are:

1. The STDs, FBDs and control logic of elements and/or components with same functional behaviour and same number of states can be reused anywhere in the workflow process, despite of different names. There is no need to design the low-level control logic code separately for the elements and/or components with same functional behaviour. For example, FBD designed for Ejector actuator can be used for the Separator actuator as these have the same number of states and functional behaviour; however, these have different names.

2. Existing CB based applications can be reconfigured without rewriting or disturbing the whole program. A change in combination of states (State Transition Table) as per the workflow process actions and interlocking conditions is only required to add or remove any element(s) and/or component(s). This makes the overall design process very easier and simpler.

3. When changing control logic applications in ControlBuild, specifically FBD in current case, there is no need to manually write/modify corresponding WSDL and DPWS functions, as these are automatically generated.

4. For adding a new component/element, say sensor and separator, the Web Services interface (DPWS) is implemented by simply adding the sensor and separator state variables in existing component (say conveyor) WSDL file and regenerating the new stub and skeleton files. These files also need not to be manually altered or generate, as these are available through server of corresponding subsystem (say station 2) for deploying on the STB device.

5. There is no need to redesign the WS interface (DPWS) of the whole control application in instances of adding or removing a component and/or element. It is because; the WS interface maps the input lines of elements using the state notifications. This means changing the control logic sequence (e.g. adding/removing a component and/or element and/or interlocking conditions) do not require changing the WS interface.
6. As reported in State Transition Tables 8.3 & 8.4, the workflow process sequence can be changed by just changing the interlocking of states, i.e. the state numbers. There is no need to redesign the logic/FBD for any changes in the workflow process. This also means that there will be no change in memory, provided the number of elements remains the same. With the addition of an element and/or component, additional memory may be required depending upon the number of states.

7. It is also observed that with increase in number of elements per component or addition of elements with large number of states will result into State Transition Table of large number of combination of states. It is observed that usually for any workflow process not all these combination of states is required. Thus, the unused state combinations if also used in designing control logic may consume a significant amount of memory. This is a drawback of CB approach with large number of combination of states. This can be overcome by eliminating the combination of states not required from FBD, and thus saving the memory. It is also observed that eliminating the unused combination of states do not significantly affect the process timings of device.

In summary, for adding a new element and/or component, the control logic (FBD) of existing elements (e.g. actuator and/or sensor) can be re-used (re-configured) for the new one, provided they have the same functionality. The reconfiguration is managed by reusing the existing elements function blocks and altering interlocking conditions (if any) by just changing the output to HIGH or LOW for the desired (new) combination of states as mentioned in the state transition table. However in case of FTB based WS approach presented by Phaithoonbuathong (2009), the logic needs to be modified on both the central orchestrator and device as well. In case of the PLC-based system, the application builders require 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. In this way of the process reconfiguration, any changes require careful attention and understanding of control sequences in the machine application. However in CB approach, the process can be done simply by just changing the combination of states in the State Transition Table, and thus 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.

Figure 8.12: Component view of station 2 in ControlBuild before adding the separator component.

Figure 8.13: Component view of station 2 in ControlBuild after adding the separator component.
8.5.2 Case II: Adding a new station

In this case, a new station named indexing rotary table, was installed on the test rig as shown in Figure 8.14. This station was controlled through the new STB3 device. This station has many elements; however for simplicity, its only one element named Work Piece Available Table State Sensor (WP1TS) is taken in this case study. The role of this sensor is to check if the work piece has arrived at the rotary table. The actions undertaken to install this station and the key observations made from this case study are:

1. The new station with its components and elements were physically installed as depicted in Figure 8.14. The pneumatic controlled input lines were connected to the air pressure; while the input lines of elements were connected to the available channels of the I/O module of station 3 of test rig. These were further connected to I/O module of the STB3 device, as described in the I/O configuration file.

2. The states of elements (in particularly WP1TS sensor) were defined (Table 8.5). In this case, WP1TS sensor element has two states as mentioned in Figure 8.15, so the STD of any binary sensors can be used. This usage of STDs of elements with same states supports re-usability of CB design approach and thus makes the design procedure simple, faster and easier.

3. The state transition table is shown in Table 8.6. The DFSM model (Figures 8.16 & 8.17) of station 3 was designed using the STDs of elements and the state transition table, which reports all the combinations of states of control elements of station 3, including the interlocking conditions. Control logic of station 2 is modified and it becomes a client of station 3.

4. In the next stage, the control logic of station 3 in ControlBuild was designed as shown in Figure 8.18. The functional operation of WP1TS sensor element is same as binary sensors such as Magazine sensor; therefore, the FBDs designed for these elements in Chapter 6 can be reused here by just changing the names and the states as per Tables 8.6. This is the benefit of reusing FBDs in ControlBuild for elements with same operational functions.
5. The I/O files were generated in ControlBuild and the new component software was debugged, simulated and validated for testing purposes prior to its deployment on the device. Finally the new tested software was deployed on the device.

Figure 8.14: Adding the *indexing* rotary table (station 3).

Figure 8.15: STD of WP1TS *sensor* control element of *rotary table* component.
Table 8.5: Assigning state numbers to control element of station 3.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Number</th>
<th>Number of States</th>
<th>State Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component 1: Rotary Table</td>
<td>WPTTS</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 8.6: State transition table reporting all the combinations of states of control elements of separator component of station 2.

<table>
<thead>
<tr>
<th>WP2 Sensor</th>
<th>Interlocking with WPITS Sensor</th>
<th>Interlocking with Conveyor</th>
<th>Output</th>
<th>Action/Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>3.1</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>3.2</td>
<td>2.2</td>
<td>HIGH</td>
<td>Extended</td>
</tr>
<tr>
<td>4.1</td>
<td>3.3</td>
<td>5.1</td>
<td>2.2</td>
<td>Move to retracted position</td>
</tr>
<tr>
<td>4.1</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>3.1</td>
<td>2.2</td>
<td>HIGH</td>
<td>Extended</td>
</tr>
<tr>
<td>4.2</td>
<td>3.2</td>
<td>2.2</td>
<td>HIGH</td>
<td>Extended</td>
</tr>
<tr>
<td>4.2</td>
<td>3.3</td>
<td></td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>3.4</td>
<td></td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.16: Logical view of station 2 and station 3 on STB2 & STB3 devices respectively.
Figure 8.17: DFSM model of station 2 & station 3 using STD’s & interlocking of corresponding control elements.
8.5.2.1 Results & observations

In addition to the benefits (such as reconfigurability and reusability) observed in the outcomes of the Case I scenario, following key outcomes and observations were made from this experimental case study:

1. The control logic of station 3 can be added to the same program of STB1 & 2 in ControlBuild. It is because in ControlBuild, the selection of device is made at the time of deploying the program. As the control logic design procedure is independent of specific device; so this deployment process is called dynamic deployment. Therefore, the control logic for station 3 can be designed independently of STB3; which makes the design procedure easier.

2. In the proposed CB approach, a client-server architecture is used to support peer-to-peer communication within the devices. Therefore on adding the control logic of station 3 on STB3 device, only the client of server 3 (STB3) needs to be dragged into the control logic application of station 2 (STB2). There will not be any change in the FBD of STB1. Thus for adding another device, there is no need to design or change the low-level control logic code of its predecessor device (e.g. STB1. However for STB2, the FBD of separator component will be modified according to the interlocking state conditions with WP1TS sensor of station 3.

3. STB3 can invoke server functions defined on another STB device (if connected next in the sequence) by simply invoking the DPWS call function, and thus act
as client of that device. There is no need to change its control logic, until any new sequences or interlocking conditions are defined. However comparing to studies presented by Phaithoonbuathong (2009), on adding or removing a device, the low-level program in central orchestrator and device needs to be changed. This makes the design process time consuming and complex.

4. When changing control logic applications in ControlBuild on STB2 or STB3, specifically FBD in current case, there is no need to manually write/modify corresponding WSDL and DPWS functions, as these are automatically generated.

5. Once the client-server WS interface is established within the devices (STB2 and STB3); there is no need to redesign the WS interface (DPWS) in situations such as adding/removing a component and/or element and/or interlocking conditions.

8.5.3 Case III: Modifying the workflow process

This case investigates the suitability of applying proposed SOA-WS based CB design approach in the environment of changing workflow process, which is very common in manufacturing industries (e.g. Ford process lines). For this purpose, the workflow process of test rig presented in Figure 8.14 is modified by removing the station 2 and directly connecting station 1 to station 3. In this case, the assembled workpiece from station 1 is directly passed to station 3 for further operations as shown in Figure 8.19. For simplicity, only WP1TS sensor of Rotary Table component of station 3 is considered. In the new workflow process, the Swivel Arm component of station 1 is interlocked to the WP1TS sensor of station 3. This is similar in situation prior to modifying the workflow process; where the Swivel Arm component of station 1 is interlocked to the WP1 sensor of Conveyor Belt component of station 2. Therefore, same interlocking logic can be used. However, the names of elements and components have been changed. The key tasks undertaken to perform this change in workflow process and the observations made in addition to those noticed in Cases I and II are detailed below:

1. Station 2 including its physical, electric and pneumatic connections was removed from the rig.

2. Station 3 was shifted towards station 1 and locked in the right position to maintain the alignment of physical elements.
3. The earlier configuration of station 1/2/3 of test rig is reconfigured to station 1/3 after removing the station 2. This scenario is shown in Figure 8.20. Therefore, the devices STB1 and STB3 will now directly communicate in a peer-to-peer fashion. This means the WS interface (DPWS function invocation) need modifications. More specifically, client function of station 2 on server 1 needs replacing by the client function of server 3 in ControlBuild, as shown in Figure 8.21.

4. As the interlocking conditions remain the same; so the names were only changed in the new control logic.

5. The new control system is configured, tested and validated. Following validation, control logics for station 1 & station 3 are respectively deployed on STB1 and STB3. The I/O files remain the same.

The key observations made from this case study are:

1. This modification in workflow process does not lead to significant changes in the control software. Comparing to Cases I & II, this case requires WS interface (DPWS function invocations) changing and this is done graphically. It is because the sequence of communicating devices was changed from STB 1/2/3 to STB 1/3.

2. The control logic in FBD needs not to be changed as no element and/or component have been added or removed. Only the old interlocks have been altered and replaced by the new interlocks to define the new machine sequence.

3. There are no changes in I/O configurations of station 1 on the device controller.

4. The modifications of control software in this 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.

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. These are found very easy to implement in proposed CB approach; however, in research presented by Phaithoonbuathong (2009), it is very subtle because the interlocking variables are difficult to trace throughout the application since these are linked to various points.
Chapter 8: Implementation & Evaluation

Figure 8.19: Modifying the workflow process by removing station 2.

Figure 8.20: Logical view of (a) stations 1/2/3 on STB1,2,3; and (b) logical view of station 1/3 on STB1,3 after removing the station 2.
In addition to the above, it can also be observed from above scenarios that the CB approach supports both modularity and reusability of designed components. The modularity feature is very beneficial for design reuse, as already-designed modules with well-defined interfaces may be used again in other designs. Ideally in agile manufacturing systems, the term ‘reuse’ applies to reuse of both software and hardware components. Through this approach, any product change, upgrade and variety can be achieved by simply replacing and/or adding a module in a system, without changing the overall production platform. This is more clear in Figure 8.22, which gives an example of component view of two previously defined elements: WP1 and WP1TS sensors. From Figure 8.22, it is clear that the FBD/component view of elements/components can be reused, provided the new elements/components have same number of states and functional behaviour. More specifically, these can be reused at another place by either copy-paste or export-import approach in ControlBuild, followed by changing variable names, as shown in Figure 8.22. Moreover, these elements or components can be deployed on any STB device, as ControlBuild supports dynamic deployment. It is also observed that 100% of the design of element (say sensor in this case) can be reused at the DPWS application level for creating new components within the WS development platform; apart from changing the names. Thus, the component’s control logic can be considered to be encapsulated as ‘black box’, with well defined I/O interfaces.

It has also been noticed that the proposed SOA-WS based CB design approach gives a high level of granularity within the component-based system architecture in
Chapter 8: Implementation & Evaluation

order to support effective reuse and reconfiguration. This is because there is minimal coupling interaction between modules/stations and maximum cohesion interaction within modules/stations. For example, the Ford-Festo test rig features 2 couplings (interlocks between stations) and 4 cohesions (internal interaction within stations) for current configuration, station 1/2/3, with station 3 having one component with one element. This makes system easier to build, reconfigure, repair and manage. In the CB approach, it is not only the control application which is defined as the modular component. Additional services, such as state publication are also designed for modularity and are well organised in the constitution of components.

8.5.4 Assessing integration with engineering applications

This section focuses on investigating the potential of integrating the proposed SOA-WS based CB approach with third party engineering tools (e.g. PDE toolkit (Lee, 2004)), manufacturing applications (e.g. HMI) and business applications. The WS based fully distributed control system, designed in this thesis, can be integrated with these applications through a common Web Services based SOA middleware, shown in Figure 8.23. The integration with the above three applications is described below.

- Process Definition Environment tools: These graphical based tools are un-
der development in MSI group. These are designed to provide end-users (i.e. the system integrator/control builder), the ease of process design and reconfiguration. This is done by hiding the complexity of component programming and minimising the level of manual coding. Through this approach, the PDE tools can significantly contribute in accommodating the design, integration of the control system and process reconfiguration.

The designed SOA-WS based CB approach can be integrated with PDE tools. For this purpose, a platform is required to accommodate PDE tools in developing new components and uploading device runtime control applications and data configurations. The architecture of such a platform is shown in Figure 8.24. In Figure 8.24, the control application (e.g. FBD in current case) along with DPWS configurations is designed in ControlBuild (as mentioned before in this thesis). This results in WSDL files, which can be used by PDE tools to gather information about the designed components. PDE tools use the information contained in WSDL files to generate the XML script files, required for the runtime configuration. These XML files define device specific interlocking conditions for
the control application.

![Diagram](image)

**Figure 8.24:** A framework to integrate designed control application with PDE tools.

The PDE tools do not support developing WS components and configuring devices. PDE functionality focuses only on defining component behaviour interlocks and manufacturing process sequences. In research reported in this thesis, ControlBuild is used as development tool. Therefore, there is a need of an interpreter/converter to read the XML script file generated by PDE tools and provide a device compatible PLCopen file. In addition to this dynamic deployment of DPWS components, the runtime (re) configuration of component control applications can also be achieved via the WS-Management service, shown in Figure 8.24. This is currently in development with the vendor. In this way, the reconfiguration of machine sequences can be implemented using the PDE tools. In summary, PDE tools can be considered as a graphical intermediate tool which can provide a direct translation of manufacturing process sequence into usable code for the embedded devices. This will help maximising the reusability and re-configurability of the control application.

- **HMI monitoring system:** In current test setup, a HMI (Schneider Magelis) is integrated with the WS control system (encapsulated in STB device) through the TCP/IP connection to mimic the real-time monitoring of component states.
In SOA-WS based CB approach, the information related to any I/O variable is passed in terms of states which are displayed on the HMI using Ethernet TCP/IP. The Magelis HMI is provided by Schneider Electric and can monitor data using DPWS based interface. The data monitoring application for HMI is designed and tested using Vijeo designer software. In this application, the I/O variables are mapped to the I/O register addresses of STB device, in order to display their status. In addition, STBs have embedded web pages to provide device configuration. These web pages can be accessed using DPWS/Materna explorer. It is possible to configure, reboot and visualise the device status through these web pages.

- **Business integration:** In the earlier studies presented by MSI group on LonWorks and PLC-based control systems (designed using the component-based design framework), the integration with business applications requires a communication protocol driver. This driver translates the device data into format, in compliance with integrated applications. The communication protocol driver acts as a gateway between control devices and integrated applications, such as HMI’s, manufacturing and business applications. However in contrast, the research reported in this thesis focuses on the WS-based automation systems with a non specific type of device driver interface for integrating the automation and business systems. This can be done either via the TCP/IP connection or directly using the common DPWS interface (i.e. SOA device middleware).

The runtime device information required by the business level can be obtained using WS enabled devices such as STB. The device specific information and live states can be provided 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.

For seamless integration, Web Services on the devices use DPWS interfaces to 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. Also, the provision of SOA middleware has enabled direct integration to automation devices through
the standard WS interface. However, this is beyond the scope of research as this is in development with vendors of STB devices and ControlBuild software, so it is not reported in this thesis.

8.5.5 Comparison with earlier studies of MSI group

A detailed comparison of the author’s distributed SOA-WS based CB design approach with previous orchestrator based CB approach (Phaithoonbuathong, 2009) and centralised PLC approach for designing control systems is presented in Table 8.7. The data presented in Table 8.7 is taken from research reported in Phaithoonbuathong (2009) and the outcomes of above mentioned case studies. From Table 8.7, it is clear that proposed SOA-WS based CB approach offers significant benefits such as graphical based design and peer-to-peer communication over the earlier research reported by MSI group.

Table 8.7: Comparison with earlier studies of MSI Group.

<table>
<thead>
<tr>
<th>Features</th>
<th>Conventional Centralized PLC Control System</th>
<th>Orchestrator SOA-WS based CB Control System</th>
<th>Distributed SOA-WS based CB Control System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Re-configurability</strong></td>
<td>Time consuming and error prone process, because of complex wiring, rigid structure and tight coupling interlock of control system.</td>
<td>Supports re-configurability with better ease than conventional centralised PLC control system. Time consuming to implement.</td>
<td>Fully distributed so easier to reconfigure both in terms of cost and effort. Less time is required to change and debug the new configuration.</td>
</tr>
<tr>
<td><strong>Operational performance</strong></td>
<td>Good performance with fast I/O response because of direct point-to-point communication.</td>
<td>Slower speed as time required to initialize devices, exchanging variables on network and delay due to data parsing.</td>
<td>Slower speed as time required to initialize devices. However, faster than orchestrator based approach because of faster embedded processor based devices.</td>
</tr>
<tr>
<td><strong>Robustness &amp; network performance</strong></td>
<td>High risk of a single-point failure.</td>
<td>Better than centralised approach. However, there is possibility of data loss if network load is substantially too high, which may affect robustness.</td>
<td>No risk of a single-point failure because of loose coupling within devices. Higher robustness because of additional message acknowledgement.</td>
</tr>
</tbody>
</table>
### Table 8.7 – Continued from previous page

<table>
<thead>
<tr>
<th>Features</th>
<th>Conventional centralised PLC Control System</th>
<th>Orchestrator SOA-WS based CB Control System</th>
<th>Distributed SOA-WS based CB Control System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Initialisation</strong></td>
<td>Faster in respect of system boot-up time and controller initialization as device configuration initialization is not required.</td>
<td>Slower to start up as it requires controller initialization (i.e. network and software component configurations).</td>
<td>Slower to start than centralised approach; however, faster than orchestrator based approach because of greater processing speed and dedicated WS processors and removal of central orchestrator.</td>
</tr>
<tr>
<td><strong>Software design &amp; implementation</strong></td>
<td>Coding and implementation methods are complex. Need skilled engineers.</td>
<td>Coding and implementation methods are complex; however, less than centralised approach.</td>
<td>Graphical based design approach makes it simple and easier to design &amp; implement control applications.</td>
</tr>
<tr>
<td><strong>Cost &amp; maintenance</strong></td>
<td>No expensive hardware; however, expensive to maintain, reconfigure &amp; upgrade the system.</td>
<td>Expensive hardware as it needs more sophisticated controllers with integrated TCP/IP network functionality. Requires an orchestrator and expensive to reconfigure as well.</td>
<td>Cheaper and easier to maintain, implement &amp; upgrade, as it uses cheap and powerful embedded micro-controllers with greater software/hardware reuse.</td>
</tr>
<tr>
<td><strong>Openness</strong></td>
<td>Highly customized &amp; vendor specific solution</td>
<td>Support openness</td>
<td>Support openness</td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td>Complex and fragmented application integration to the shop-floor customised automation system.</td>
<td>Simple system integration with only a single application interface per device with standard technologies (i.e. SOA and WS) where various applications can simultaneously integrate to the shop-floor system without adding more interfaces.</td>
<td>The system is built within standard technologies (i.e. SOA, WS and Ethernet) where various applications can simultaneously integrate to the shop-floor system without adding more interfaces.</td>
</tr>
</tbody>
</table>

### 8.6 Suitability in fulfilling the end user requirements

The outcomes of above research studies have demonstrated the feasibility of adopting SOA-WS based CB design approach for designing automation systems. The analysis work and test scenarios have shown that this approach is able to meet the requirements of agile automation systems. These details are provided in Table 8.8.
### Table 8.8: Suitability in meeting the end user requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing reconfigurable automation systems</td>
<td>SOA-WS complement CB design approaches within the DPWS via the encapsulation of low-level device programming. Also, ControlBuild is a graphical tool. Therefore, the control builder can alter the component and process configuration without looking into low-level device code. In particularly, component/element modules with same states and functions can be easily reconfigured by just changing the interlocking logic between components.</td>
</tr>
<tr>
<td>Reusable design</td>
<td>In the proposed CB approach, the component/element with same functionalities is just differentiated by variables namespace. Therefore, the incorporation of a new component/element into the system could reuse control applications (components/elements) from common / similar components by just changing the names of variables.</td>
</tr>
<tr>
<td>Visualisation and simulation</td>
<td>The Ford-Festo test rig was integrated with HMI in the above mentioned scenarios. Web Services use DPWS interfaces to provide the live state of variables of components; which can be used to capture process activity and to determine system performance. Also, in ControlBuild, the designed control application can be tested using graphical simulation tools for validation purposes prior to deployment.</td>
</tr>
<tr>
<td>Interoperability and non-vendor specific platforms</td>
<td>The SOA-WS based CB automation system utilises a SOAP-XML message exchange approach. SOAP uses a neutral platform and thus enables interoperability 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 other embedded microcontroller devices or PLC's within the compliant DPWS gateway.</td>
</tr>
<tr>
<td>Seamless Integration</td>
<td>The DPWS based 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, is required to enhance the seamless integration by directly integrating their applications to the control device within the DPWS.</td>
</tr>
<tr>
<td>Distributed Autonomous Automation System</td>
<td>The design of Web Services-based control devices supports the distribution of component functionalities into local control devices. In 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 distributed autonomous automation system has been demonstrated on the test rig, and the system performance has met the operational requirements of automation systems. The implementation of autonomous automation systems is enabled by the peer-to-peer communication approach.</td>
</tr>
</tbody>
</table>
Chapter 8: Implementation & Evaluation

Table 8.8 – Continued from previous page

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intelligent Machines</td>
<td>DPWS provides rich information about each component through the WS-Metadata services initialised on each control device. 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 can be used for process management and preventive maintenance software in order to keep track and monitor a control system throughout its lifecycle.</td>
</tr>
<tr>
<td>Simplicity in design</td>
<td>The software tool used to design and implement the system is a graphical based automation software design tool. It provides a comprehensive platform for developing and distributing IEC 61131-3 applications on different types of PLCs architectures (Schneider, Siemens, Rockwell).</td>
</tr>
</tbody>
</table>

8.7 Conclusion

This chapter has considered the suitability of applying the proposed SOA-WS based CB approach to an industrially representative application. This is done by implementing the said approach on the Ford-Festo test rig. The results obtained from the experimental studies done on this test rig have been compared with those mentioned in the studies presented by Phaithoonbuathong (2009). The test rig operation and execution was handled by the STB devices; interacting in peer-to-peer fashion using the client-server approach. The control applications were designed in ControlBuild and deployed on STB devices. The STBs perform the control operations corresponding to the designed component state behaviours and interlocks. The complete process realises a fully distributing system, as required.

The proposed SOA-WS based CB approach has been evaluated. The client-server based WS communication between STB devices is analysed in relation to the reliability and structure of SOAP messages exchanged in the TCP/IP communication. The qualitative features of the control WS system have been assessed in terms of their performance, with respect to various parameters such as I/O processing time and also the cost of the hardware devices. The I/O response time measured for DPWS enabled STB device is 28.7 milliseconds, somewhat less than that of 31.06 milliseconds, as reported by Phaithoonbuathong (2009) for a comparable orchestrator-based control system solution implemented using FTB devices. This performance is useable in an
industrial context, meeting the required timing criteria specified by Ford Motor Company (co-sponsor of this project); which specifies that the I/O response time between inter-connecting controllers in production must meet a soft real-time criterion of under 30 milliseconds in order to meet the normal performance of the machine assembly process. The cost of developing the embedded control platform using STB’s is found to be substantially lower than the FTB based control system mentioned in studies presented by MSI group. Due to the elimination of the need for an orchestrator, the author’s STB based implementation should be substantially cheap to implement since no central PC/PLC is required.

The qualitative features were evaluated by measuring the degree of reusability, reconfigurability and integration of the WS control system. These are the key evaluation areas in the context of agile manufacturing. In this research, the WS automation system has demonstrated a good level of re-configurablity and reusability, in terms of the activity and skills involved when compared with conventional PLC based control systems. Web Services also potentially provide a much better quality of process/shop floor system integration as required by end-users to support collaborative manufacturing frameworks. From these results, it is concluded that the proposed SOA-WS based CB approach is indeed feasible and potentially suitable to be adopted for the next generation of automation systems. It is important to note that further work is needed on error handling and recovery beyond the proof-of-concept work reported here.
Conclusions and Recommendations

In this chapter, a comprehensive summary of the research work undertaken in the thesis is presented, followed by the conclusions. At the end, key recommendations for the future work are given.

9.1 Overview of research objectives

This section summarises the research work done in this thesis to develop the SOA-WS and Component Based approach; to better support future generation agile manufacturing systems. To achieve the aim and research objectives defined in Chapter 4, the following specific tasks were reported in the thesis:

- **Objective 1:** To review existing manufacturing industry automation requirements, standards and technologies involved, in situ operational restrictions and problems associated.

  This objective is covered in the literature review reported in Chapters 2 & 3. A review of the traditional automation systems is done and their limitations in meeting the requirements of future manufacturing paradigms are highlighted. This study has been reported in Chapter 2. The reported literature provided
an overview of the research work to be done and the work already been done to meet the challenges faced by traditional automation systems. In Chapter 3, the current technologies in use for designing the distributed control systems, integration frameworks and middlewares are reviewed. This study has presented a comparison of the current technologies, including their limitations. This comparison has justified the capability of component based design approaches in meeting the requirements of future generation manufacturing paradigm.

- **Objective 2:** To investigate a strategy to decompose automated industrial operations, such as powertrain assembly line, into user manageable component based modules required for industrial automation.

To achieve this objective, a component based development concept for the future generation agile automation system with Web Services as communication medium is proposed in Chapter 5. It explains the methodology of distributing control to local field devices. A procedure for dividing the whole automation process into small reusable components; which individually and/or as a group can communicate using Ethernet based Web Services is highlighted. In this approach, the automation component is defined as an autonomous unit incorporating its own physical hardware, application software and local intelligence that enables it to know what, how and when to execute its own automation function without the need for a central controller. Its logical behaviour is represented by control elements using the distributed finite state machines represented by state-transition diagrams. The interaction within components is done through well-defined network interfaces; which are configurable through configuration parameters. These parameters are used to modify their specific operational characteristics as well as their interlock relationships with other components. The overall system is designed by configuring these interlocking relationships.

- **Objective 3:** To specify and develop prototype Web Services and SOA based industrial automation model using reconfigurable manufacturing components. To implement the said model on embedded devices and to provide the set of desired services and units needed to compose manufacturing tasks.
This objective is covered in the reported work in Chapter 6. It explains the importance of employing Web Services at shop floor devices to give a technologically neutral platform for integrating component based automation systems at shop-floor to the higher level business applications. This is done using the DPWS protocol; which provides functions for hosting services on the control device, discovery methods, control application execution and event-notification services. To design and deploy WS based control applications using DPWS protocol, a graphical engineering method called ControlBuild provided by Geensys is presented. ControlBuild is found beneficial as it includes a variety of graphical interactive editors for component-based design covering IEC 61131-3, C and electrical circuit diagrams, a set of component libraries and a simulation environment for control software testing & validation prior to integration and commissioning. A detailed discussion on using ContolBuild (particularly FBD) has been presented in Chapter 6.

- **Objective 4:** To perform pilot study trials on the developed model and to understand its efficiency in fulfilling user requirements for industrial automation.

In Chapter 7, the layout of a test rig named Ford Festo test rig is presented. This test rig is used for implementing the concept of CB automation system designed using ControlBuild. It also presents a selection of DPWS based embedded device and its implementation to form a distributed automation system. In particular, a selection of Advantys STB provided by Schneider is made and its integration with ControlBuild is presented. Finally to achieve the above objective, the chapter presents an example of designing a CB control application with WS interface using ControlBuild and deploying it on the selected STB device.

- **Objective 5:** To develop, reconfigure and test the developed model in context of requirements and efficiency targets of future generation automation systems.

To achieve this objective, an investigation is made on implementing Web Services on STB devices within the component-based design approach on the Ford-Festo test rig. The investigation is made in terms of the performance and reliability of message passing between control devices under soft real-time constraints. The modularity and degree of re-configurability of the designed system using this WS
control architecture is also assessed. Finally, the developed component-based system is analysed to examine the practicality of using component-based distributed control paradigm as the future generation agile manufacturing system.

9.2 Research contributions

The research reported in current thesis has proposed the novel idea of integrating the CB design approach with SOA & WS, using a choreography based peer-to-peer communication approach, in order to realise a fully distributed agile manufacturing system. 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. A well defined structure of implementing the said approach using embedded controllers (particulary STBs) and graphical based tools (ControlBuild) has been presented in current thesis. The outcomes of the performed experimental studies have highlighted robust evidence of positive WS performance on embedded control device platforms when compared with PLC, centralised orchestrator based system and other distributed systems technologies. The research has also been evaluated both in terms of the end-user requirements (such as reconfigurability and reusability) of agile automation and various quantitative parameters. These studies have demonstrated the feasibility of adopting the SOA-WS based CB approach within manufacturing systems. A part of this approach was previously developed at the MSI Research Institute, 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 9.1 and are detailed below. However, the key contribution of author is:

An innovative approach to design a unique novel framework using proposed SOA middleware technology, for the realisation of CB control system, which is truly distributed on a WS based embedded platform. The control/application logic designed during CB based design can be deployed on distributed embedded devices communicating in a peer-to-peer fashion, thus, making the automation platform reconfigurable and reusable, logically as well as physically.
Figure 9.1: Author's original work and novel contributions.
Chapter 9: Conclusions and Recommendations

The overall conclusions drawn and findings made from the research presented in this thesis are summarised here as follows:

- **Defining an agile automation system using unifying application interface (SOA-WS) to provide integration from shop-floor to higher levels of an enterprise.** A discussion on key manufacturing technologies is presented in Chapter 2. From the discussion, it is concluded that collaborative mass customisation offers maximum degree of flexibility and is thus will closely meet the requirements of future agile manufacturing systems. The existing technologies are found to provide only partial agility features and have limitations in realising a complete agile manufacturing paradigm. The common problem is the poor level of interoperability between vendor specific technologies. In order to improve interoperability and also increase flexibility, a new design of the control and manufacturing platform using standard technology of SOA and Web Services is proposed.

- **Implementing SOA at device level using common standard interoperable interface (DPWS and Ethernet).** The technologies involved for designing the distributed open control systems are discussed in Chapter 3. The concept of Intelligent, autonomous and cooperative holons is mentioned. However, these have certain requirements such as a mixed top-down and bottom-up development approach. This is why these are not found suitable for future agile manufacturing systems. From the discussion presented, it has also been found that the object-oriented approach could not make significant inroads into manufacturing due to its complexity of implementation and diverse implemented tools. SOA provide loosely coupled service interfaces when implemented at device level. It is concluded that SOA-WS based communications are of an asynchronous nature and are thus more scalable and suitable for complex manufacturing processes where many operations with variable response times run concurrently. From this discussion, it is predicted that the development of SOA-WS at shop floor device level can result in a new way of building and integrating automation systems to higher control levels.

- **Design and decomposition of automation system into components logically and physically to enable reusability and reconfigurability of the
system. An innovative way of designing the manufacturing systems using component-based technology in association with SOA and WS is presented in Chapter 5. The physical and functional representation of an automation component is presented. Based on the requirements of manufacturing automation systems, an automation component is defined as an autonomous unit incorporating its own physical hardware, application software and local intelligence that enables it to know what, how and when to execute its own automation function without the need for a central controller. The logical behaviour of the component is represented by control elements using the distributed finite state machines represented by state-transition diagrams. From the detailed discussion presented on CB technology in Chapter 5, it is concluded that it will offer significant benefits, especially simplicity, reusability and reconfigurability, in designing the future manufacturing systems.

- **Implementation of CB design framework, including encapsulation of DPWS and WS based interface, using graphical based design tool (ControlBuild).** A discussion on DPWS is presented in Chapter 6. It is found that 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. For implementing Web Services and SOA at device level, DPWS is considered as the most suitable protocol. Using DPWS, the WS provide functions for hosting services on the control device, discovery methods, control application execution and event-notification services. To design and deploy WS based control applications using DPWS protocol, a graphical engineering method called ControlBuild provided by Geensys is presented in Chapter 6. ControlBuild is found to be beneficial as it includes a variety of graphical interactive editors for component-based design covering IEC 61131-3, C and electrical circuit diagrams, a set of component libraries and a simulation environment for control software testing & validation prior to integration and commissioning.

- **Implementing WS peer-to-peer architecture (client-server model) to realise a truly distributed automation system.** For the distributed automation systems comprising many devices, a peer-to-peer approach is used for
device level interactions. These have been implemented in ControlBuild using a client-server model, through the automatically generated stub and skeleton files. This in combination with WS enabled devices is found to create a technology neutral platform, where devices from different vendors can interoperate via XML message passing and thus enhance system’s flexibility and re-configurability through effective plug-and-play discovery.

- **Evaluation and assessment of system performance, packet structure and agility of the WS based fully distributed system.** The proposed research has been experimentally evaluated using the Ford-Festo test rig located at MSI laboratory of Loughborough University. This test rig replicates the operations of assembly lines of Ford and is used to determine the feasibility of using SOA and Web Services based CB design approach for distributed automation systems. The results of the experimental work done has emphasised the Ethernet network communication approach, performance, and speed and explicitly outlined and justified the adoption of Web Services within the distributed control system. The use of Advantys STB embedded device provided by Schneider and its integration with ControlBuild has also justified the use of WS at device level. The modularity and degree of re-configurability of the designed system using this WS control architecture have also been assessed. In summary, the demonstrated prototype has provided a manufacturing platform for the migration path to fully distributed peer-to-peer automation systems, which will allow components to react autonomously to the manufacturing environment.

### 9.3 Recommendations for future work

The research presented in this thesis has shown promising results for applying CB technology in combination with SOA-WS for fully distributed peer-to-peer system, ControlBuild for programming and STBs as embedded devices. This as a whole has shown achievable results to yield an agile automation system within the automotive manufacturing sectors. However, research in this field is still in progress, and the following recommendations are made for the future work:
• **PDE suite tools:** These tools, developed at the MSI research institute, are based on component based technology. These are not fully matured and its direct integration with WS-Management functionality on the embedded devices is a complicated task. It is because the XML control data configuration format produced by the PDE tools is not in a format which can be managed by the embedded WS-Management functionality on all the control devices. In addition, to accurately validate the components designed using PDE tools, 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. Therefore, further work could be done on integrating the PDE tools and the DPWS-enabled control devices, via the DPWS interface, for seamless connection to provide process re-configuration and runtime visualisation.

• **Security, robustness and reliability:** Further work can be done to improve the reliability of system by minimizing the number of lost messages and using properly managed error recovery techniques. 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 included in the standard DPWS implementation.

• **Integrating various DPWS based 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.

9.4 Limitations

Although the research reported in current thesis has shown promising results; however, it has still some limitations, as mentioned below:

- Attempts have been made to realise a fully distributed autonomous automation systems with WS enabled control devices. However, only a limited number of embedded devices are utilised in this research due to the test rig constraints. However in practise, there may be a large number of devices or elements which can result in more complex programs requiring more memory. The complex programs if designed using ControlBuild may need a large number of graphical blocks, which at present is limited to 200 per program.

- Although the Web Services 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. Such tools are not yet available.

- The research reported in this thesis deals with only binary actuators and sensors.

- SOAP messages use a text based representation, and the format is ASCII text. Therefore, SOAP protocol requires significant amount of processing time and bandwidth in comparison with binary fixed format implemented in say CORBA. Adoption of binary SOAP in embedded control devices is still dependent on future research.


References


References


Appendix A

Designing CB automation systems in ControlBuild

This appendix presented an example of files generated at various stages of designing a CB control application in ControlBuild. In particular, the files presented in this Appendix corresponds to the example of conveyor belt presented in Chapter 7. This appendix includes the XML file generated for the FBD design of the control logic of components, WSDL file for the WS interface of component, I/O configuration file representing the physical I/O configuration of the device and the C interface file corresponding to the client of a server component.
A.1 XML file for sensor component designed in ControlBuild

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generator: ControlBuild -->
<fileHeader companyName="TNI-Software" productName="ControlBuild" productversion="4.10"
creationDateTime="2011-02-20T9:24:31+00:00"/>
<contentHeader name="sensor_logic">
<coordinateinfo>
<fbd>
<scaling x="100" y="100"/>
</fbd>
</coordinateinfo>
</contentHeader>
<types>
<dataTypes/>
<pous>
<pou name="sensor_belt.sensor_logic" pouType="functionBlock">
<interface>
<inputvars>
<variable name="Belt_state_IN">
<type>INT</type>
<initialValue>
<simplevalue value="0"/>
</initialValue>
</variable>
.documentation>CB.kind=inputvars</documentation>
</inputvars>
</pou>
</pous>
</types>
</project>
```
Chapter A: Designing CB automation systems in ControlBuild

```xml
<variable name="sensor_logic">
  <type>BOOL</type>
  <initialvalue>
    <simplevalue value="false"/>
  </initialvalue>
</variable>

<documentation>CB.kind=inoutvars</documentation>
<inoutvars>
  <variable name="Sensor_State_OUT">
    <type>INT</type>
    <initialvalue>
      <simplevalue value="0"/>
    </initialvalue>
  </variable>
  <variable name="Response">
    <type>BOOL</type>
    <initialvalue>
      <simplevalue value="false"/>
    </initialvalue>
  </variable>
</inoutvars>

<documentation>CB.kind=outputvars</documentation>
<outputvars>
  <variable name="Three">
    <type>INT</type>
    <initialvalue>
      <simplevalue value="3"/>
    </initialvalue>
  </variable>
  <variable name="Four">
    <type>INT</type>
    <initialvalue>
      <simplevalue value="4"/>
    </initialvalue>
  </variable>
  <variable name="Output_low">
    <type>BOOL</type>
    <initialvalue>
      <simplevalue value="false"/>
    </initialvalue>
  </variable>
  <variable name="output_high">
    <type>BOOL</type>
    <initialvalue>
      <simplevalue value="true"/>
    </initialvalue>
  </variable>
  <variable name="Sensor_OFF">
    <type>INT</type>
    <initialvalue>
      <simplevalue value="1"/>
    </initialvalue>
  </variable>
  <variable name="Sensor_ON">
    <type>INT</type>
    <initialvalue>
      <simplevalue value="2"/>
    </initialvalue>
  </variable>
</outputvars>
```

218
Chapter A: Designing CB automation systems in ControlBuild

sensor_logic

<documentation>CB.kind=localvars</documentation>
</localvars>
</interface>
<actions>
</actions>
<body>
<FBD>
<label localId="0" label="1">  
<position x="100" y="0"/>
</label>
<invariable localId="2">  
<position x="0" y="0"/>
<expression>IO_Sensor</expression>
</invariable>
<invariable localId="5">  
<position x="0" y="200"/>
<expression>Sensor_OFF</expression>
</invariable>
<invariable localId="6">  
<position x="0" y="300"/>
<expression>Sensor_ON</expression>
</invariable>
<invariable localId="8">  
<position x="0" y="400"/>
<expression>Belt_state_IN</expression>
</invariable>
<invariable localId="13">  
<position x="0" y="500"/>
<expression>Three</expression>
</invariable>
<invariable localId="14">  
<position x="0" y="600"/>
<expression>Output_low</expression>
</invariable>
<invariable localId="16">  
<position x="0" y="700"/>
<expression>Belt_state_OUT</expression>
</invariable>
<invariable localId="18">  
<position x="0" y="800"/>
<expression>Four</expression>
</invariable>
<invariable localId="19">  
<position x="0" y="900"/>
<expression>output_high</expression>
</invariable>
</variable>
<block localId="1" width="200" height="300" typeName="sel_int" instanceName="sel_int_1">  
<position x="300" y="100"/>
<inputvariables>
<variable formalParameter="G">  
<connectionPointIn>
</variable>
</inputvariables>
</block>
<reflocalId="2"/>
<variables>
<variable formalParameter="IND">  
<connectionPointIn>
</variable>
<variable formalParameter="INI">  
<connection>
</variable>
</variables>
<reflocalId="5"/>
<variables>
<variable formalParameter="INI">  
<connectionPointIn>
</variable>
<variable formalParameter="OUT">  
<connection>
</variable>
</variables>
<reflocalId="6"/>
<variables>
<variable formalParameter="OUT">  
<connectionPointIn>
</variable>
<variable formalParameter="OUT">  
<connection>
</variable>
</variables>
</FBD>
Chapter A: Designing CB automation systems in ControlBuild

```
sensor_logic
  <block localId="7" width="200" height="200"
  >
    <position x="300" y="400" />
    <inputVariables>
      <variable formalParameter="IND">
        <connectionPointIn>
          </connection>
        </variable>
    </inputVariables>
    <refLocalId="8" />
    </connectionPointIn>
  </block>

  <block localId="13" width="200" height="200"
  >
    <position x="300" y="700" />
    <inputVariables>
      <variable formalParameter="IND">
        <connectionPointIn>
          </connection>
        </variable>
    </inputVariables>
    <refLocalId="16" />
  </block>

  <block localId="18" width="200" height="200"
  >
    <position x="500" y="400" />
    <inputVariables>
      <variable formalParameter="e1">
        <connectionPointIn>
          </connection>
        </variable>
    </inputVariables>
    <refLocalId="7" formalParameter="OUT" />
  </block>

  <block localId="14" width="200" height="200"
  >
    <position x="500" y="700" />
    <inputVariables>
      <variable formalParameter="s3">
        <connectionPointIn>
          </connection>
        </variable>
    </inputVariables>
    <refLocalId="17" />
  </block>

<documentation>C.B.Porte=true</documentation>
```

220
Chapter A: Designing CB automation systems in ControlBuild

A.2 XML file for belt component designed in ControlBuild

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generator: ControlBuild -->
<project xmlns="http://www.pleopen.org/xml/tc6.xsd" xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.pleopen.org/xml/tc6.xsd TC6/XMLSchema-instance">
  <$fileHeader companyName="TNI-Software" productName="ControlBuild" productVersion="4.10"
    creationDateTime="2011-02-20T13:03:05+00:00" />
  <$contentHeader name="belt_logic">
    <$coordinateInfo>
      <$fbConnect x="100" y="100" />
      <$id>
        <$sfcConnect x="100" y="100" />
        <$sfc x="32" y="8" />
      </sfcConnect>
    </coordinateInfo>
    <$contentInfo>
      <$dataType>
        <$pou name="sensor_belt.belt_logic" pouType="functionBlock">
          <$interface>
            <$inputvars name="message_reception">
              <$type>BOOL</type>
              <$initialValue>true</initialValue>
            </inputvars>
            <$inputvars name="Sensor_State">
              <$type>INT</type>
              <$initialValue>0</initialValue>
            </inputvars>
            <$inputvars name="IO_Belt_status">
              <$type>BOOL</type>
              <$initialValue>false</initialValue>
            </inputvars>
            <$outputvars name="Belt_State">
              <$type>INT</type>
            </outputvars>
          </interface>
        </pou>
      </dataType>
    </contentInfo>
  </contentHeader>
</project>
```
Chapter A: Designing CB automation systems in ControlBuild

```xml
<documentation>CB.kind=outputvars</documentation>
</outputvars>
<localvars>
  <variable name="Three">
    <type> <INT />
    <initialvalue> <simplevalue value="3" />
  </initialvalue>
  </variable>
  <variable name="output_high">
    <type> <BOOL />
    <initialvalue> <simplevalue value="true" />
  </initialvalue>
  </variable>
  <variable name="Four">
    <type> <INT />
    <initialvalue> <simplevalue value="4" />
  </initialvalue>
  </variable>
  <variable name="output_low">
    <type> <BOOL />
    <initialvalue> <simplevalue value="false" />
  </initialvalue>
  </variable>
  <variable name="Two">
    <type> <INT />
    <initialvalue> <simplevalue value="2" />
  </initialvalue>
  </variable>
  <variable name="One">
    <type> <INT />
    <initialvalue> <simplevalue value="1" />
  </initialvalue>
  </variable>
</localvars>
</interface>
<actions>
  <body>
    <FBD>
      <label localid="0" label="1">
        <position x="100" y="0" />
      </label>
      <invariable localid="2">
        <position x="0" y="100" />
      </invariable>
    </FBD>
  </body>
</actions>
```
Chapter A: Designing CB automation systems in ControlBuild

```
belt_logic
<expression>sensor_State</expression>
</variable>
<variable localId="7"/>
<position x="0" y="200"/>
<expression>One</expression>
</variable>
<variable localId="8"/>
<position x="0" y="300"/>
<expression>Output_low</expression>
</variable>
<variable localId="10"/>
<position x="0" y="400"/>
<expression>Sensor_State</expression>
</variable>
<variable localId="12"/>
<position x="0" y="500"/>
<expression>Two</expression>
</variable>
<variable localId="13"/>
<position x="0" y="600"/>
<expression>Output_high</expression>
</variable>
<variable localId="17"/>
<position x="0" y="700"/>
<expression>Three</expression>
</variable>
<variable localId="18"/>
<position x="0" y="800"/>
<expression>Four</expression>
</variable>
<block localId="1" width="200" height="200"

typeName="eq_int" instanceName="eq_int_1">
<position x="200" y="100"/>
<inputVariables>
<variable formalParameter="INO">
<connectionPointIn>
<connection
</variable>
</inputVariables>
</variable>
<block localId="2"/>
</variable>
<variable localId="7"/>
<position x="0" y="900"/>
<connectionPointIn>
<connection
</variable>
</inputVariables>
</variable>
</block>
<block localId="9" width="200" height="200"

typeName="eq_int" instanceName="eq_int_2">
<position x="200" y="400"/>
<inputVariables>
<variable formalParameter="INO">
<connectionPointIn>
<connection
</variable>
</inputVariables>
</variable>
<block localId="10"/>
</variable>
<variable localId="12"/>
<position x="0" y="900"/>
<connectionPointIn>
<connection
</variable>
</inputVariables>
</variable>
</block>
```
Chapter A: Designing CB automation systems in ControlBuild

```
<block localId="3" width="100" height="200"
  <position x="400" y="100" />
  <inputVariables>
    <variable formalParameter="e1">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
    <variable formalParameter="e2">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
  </inputVariables>
  <refLocalId>"1" formalParameter="OUT" /></refLocalId>
</block>

<block localId="11" width="100" height="200"
  <position x="400" y="400" />
  <inputVariables>
    <variable formalParameter="e1">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
    <variable formalParameter="e2">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
  </inputVariables>
  <refLocalId>"9" formalParameter="OUT" /></refLocalId>
</block>

<block localId="14" width="100" height="200"
  <position x="500" y="100" />
  <inputVariables>
    <variable formalParameter="e1">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
    <variable formalParameter="e2">
      <connectionPointIn>
        <connection
      </connectionPointIn>
    </variable>
  </inputVariables>
  <refLocalId>"13" formalParameter="s3" /></refLocalId>
</block>
```

225
Chapter A: Designing CB automation systems in ControlBuild

typeName="sel_int" instanceName="sel_int_3">

	<position x="700" y="600" />

	<variable formalParameter="G">
		<connectionPointIn>
			<connection />
		</connectionPointIn>
	</variable>

	<variable formalParameter="s3" />

	<variable formalParameter="IM0">
		<connectionPointIn>
			<connection />
		</connectionPointIn>
	</variable>

	<variable formalParameter="IM1">
		<connectionPointIn>
			<connection />
		</connectionPointIn>
	</variable>

	<variable formalParameter="s3" />

	<variable formalParameter="OUT">
		<connectionPointIn>
			<connection refLocalId="4" />
		</connectionPointIn>
	</variable>

	<formalParameter="s3" />

	<documentation>CB.Affect=true</documentation>

	<formalParameter="OUT" />

	<formalParameter="OUT" />

	<documentation>CB.Affect=true</documentation>

	</outVariable>

</block>

</outVariables />

<outVariables />

</block>

</type>

<instances>

<configurations />

</instances>

</project>
A.3 WSDL file for sensor component designed in ControlBuild

```xml
<xml version="1.0" encoding="ISO-8859-1" >
<!-- Generator: ControlBuild -->
<wsdl:definitions name="sensor" targetNamespace="http://www.sensor_belt.eu"
  <wsdl:documentation>www.controlbuild.com</wsdl:documentation>
  <!-- TYPES -->
  <xsd:schema targetNamespace="http://www.sensor_belt.eu"
    elementFormDefault="qualified"
    xmlns:tns="http://www.sensor_belt.eu">
    <!-- Dummy Type -->
    <!-- Simple Types -->
    <!-- Composite Types -->
    <xsd:element name="IFSensor">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="IFSensorParam" type="xsd:short" />
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <!-- MESSAGES -->
  <wsdl:message name="IFSensorSensorReqMsg">
    <wsdl:part name="IFSensorSensorIn" element="tns:IFSensor" />
  </wsdl:message>
  <!-- PORT TYPES -->
  <wsdl:portType name="sensor_belt">
    <wsdl:operation name="IFSensor">
    </wsdl:operation>
  </wsdl:portType>
  <!-- BINDINGS -->
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="IFSensor">
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
  <!-- SERVICES -->
  <wsdl:service name="sensor">
    <wsdl:port name="Sensor" binding="tns:sensor_belt">
      <soap:address location="" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```
Chapter A: Designing CB automation systems in ControlBuild

A.4 WSDL file for belt component designed in ControlBuild

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Generator: ControlBuild -->
<wSDL:definitions name="belt" targetNamespace="http://www.sensor_belt.eu"
    xmlns:wSDL="http://schemas.xmlsoap.org/wsd1/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <!--
  TYPES
  -->
  <!--
  <wSDL:types>
    <xsd:schema targetNamespace="http://www.sensor_belt.eu"
        elementFormDefault="qualified" xmlns:tns="http://www.sensor_belt.eu">
      <!-- Dummyp Type -->
      <!-- Simple Types -->
      <!-- Composite Types -->
      <xsd:element name="IFBelt">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="IPBeltParam" type="xsd:short" />
          </xsd:sequence>
        </xsd:complexType>
        <xsd:schema>
        </xsd:schema>
      </xsd:element>
    </xsd:schema>
  </wSDL:types>
  <!--
  MESSAGES
  -->
  <!--
  <wSDL:messages name="IFBeltBeltReqMsg">
    <wSDL:input name="IFBeltBeltIn" element="tns:IFBelt" />
  </wSDL:message>
  <!--
  PORT TYPES
  -->
  <!--
  <wSDL:portType name="sensor_belt">
    <wSDL:operation name="IFBelt">
      <wSDL:input message="tns:IFBeltBeltReqMsg" wsa:Action="http://www.sensor_belt.eu/sensor_belt/IFBelt" />
    </wSDL:operation>
  </wSDL:portType>
  <!--
  BINDINGS
  -->
  <!--
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wSDL:operation name="IFBelt">
    <wSDL:input>
      <soap:body use="literal" />
    </wSDL:input>
    <wSDL:operation>
      <wSDL:binding>
      </wSDL:binding>
  </wSDL:operation>
  <!--
  SERVICES
  -->
  <!--
  <wSDL:service name="belt">
    <wSDL:port name="belt" binding="tns:sensor_belt"><soap:address location="/" />
    <wSDL:port>
  </wSDL:service>
  </wSDL:service>
</wSDL:definitions>
```
A.5 Client code of sensor component designed in ControlBuild

```xml
<xml version="1.0" encoding="utf-8" ?>
<dd:ServiceClass xmlns:dd="http://www.so4d.org/dpwscore/2007/08/dyndep1"
classId="http://www.schneider-electric.com/IEC61131Skeleton"
xmlns:app="http://www.schneider-electric.com/sensor_belt.eu/sensor_belt/sensor"
<dd:Interface type="app:Control" />
<dd:Reference name="IFBelt1" type="app:Control" mustSupply="true" />
<iec:Implementation IEC61131>
  <iec:Mapping refIsPrefix="true" refSeparator="/">
    <iec:Operation name="app:Control" role="service">
      <iec:Operation name="app:IFSensor" variable="IFSensor">
        <iec:Input tag="app:IFSensor" action="http://www.schneider-electric.com/IEC61131Skeleton"/>
        <iec:Parameter tag="app:IFSensorParam" type="INT"/>
      </iec:Parameter>
      <iec:Output tag="app:IFSensor" action="http://www.schneider-electric.com/IEC61131Skeleton"/>
    </iec:Operation>
    <iec:Operation name="app:IFBelt" variable="IFBelt">
      <iec:Input tag="app:IFBelt" action="http://www.schneider-electric.com/IEC61131Skeleton"/>
      <iec:Parameter tag="app:IFBeltParam" type="INT"/>
    </iec:Parameter>
    <iec:Output tag="app:IFBelt" action="http://www.schneider-electric.com/IEC61131Skeleton"/>
  </iec:Operation>
</iec:Implementation.IEC61131>
</dd:ServiceClass>
```
Chapter A: Designing CB automation systems in ControlBuild

A.6 Client code of belt component designed in ControlBuild

```xml
<xml version="1.0" encoding="utf-8" ?>
<dd:ServiceClass xmlns:dd="http://www.so4d.org/dpwscore/2007/08/dyndepl"
    classId="http://www.sensor_belt.eu/sensor_belt/belt" xmlns:app="http://www.sensor_belt.eu">
    <dd:interface type="app:Control" />
    <dd:reference name="ifSensor1" type="app:Control" mustSupply="true" />
    <dd:wsdlInfo location="http://www.dummy.com/dummy.wsdl"
        targetNamespace="http://www.dummy.com/none" />
    <ie:implementation IEC61131>
        <ie:Mapping refIsPrefix="true" refSeparator=".">
            <ie:PortType name="app:Control" role="service">
                <ie:Operation name="app:IFBelt" variable="IFBelt">
                    <ie:Input tag="app:IFBeltParam" type="INT" variable="IFBeltParam" />
                    <ie:Output tag="app:OFBelt" actions="http://www.sensor_belt.eu/sensor_belt/OFBelt" />
                </ie:Operation>
                <ie:PortType name="app:Control" role="ref">
                    <ie:Operation name="app:ISensor" variable="ISensor">
                        <ie:Input tag="app:ISensorParam" variable="ISensorParam" type="INT" variable="ISensorParam" />
                        <ie:Output tag="app:OFSensor" actions="http://www.sensor_belt.eu/sensor_belt/OFSensor" />
                    </ie:Operation>
                    <ie:Operation>
                        <ie:Mapping />
                    </ie:Operation>
                </ie:PortType>
            </ie:PortType>
        </ie:Mapping>
    </ie:implementation.IEC61131>
</dd:ServiceClass>
```

230
A.7 IO configuration of device 1 using ControlBuild

```plaintext
# IO configuration file
# Generated automatically by ioconf_generator
# Build date: Jul 23 2009 11:52:51
# Configuration created the 15/2/2011 at 22:45
# Project: Example1, device: STB1
# picopen file: C:\Example1\Applications\sensor_belt\hardwareArchitecture\new\STB1\STB1.picopen

# Island configuration:
# MUST BE CORRECTED MANUALLY !!
# Device type
DEV_TYPE STB
# Slots description
SLOT 0 D013725
SLOT 1 D013705

# Input list:
I 0.0 IOSensor

# Output list:
O 1.0 IO_Belt_Status
O 1.1 IO_Message_Reception
```
A.8 IO configuration of device 2 using ControlBuild

# IO configuration file
# Generated automatically by ioconf_generator
# Build date: Jul 23 2009 11:52:51
# Configuration created the 15/2/2011 at 22:45
# Project: Example1, device: STB2
# piconfig file: C:\Example1\Applications\sensor_belt\hardwareArchitecture\new1\STB2\STB2.plpconf

# Island configuration:
# MUST BE CORRECTED MANUALLY !!
# Device type
DEV_TYPE STB
# Slots description
SLOT 0 DDI3725
SLOT 1 DDI3705

# Input list:
I 0.0 IO_Belt

# Output list:
O 1.0 IO_Messagereception
O 1.1 IO_Belt_status
A.9 UUID parameters of sensor component in ControlBuild

<xml version="1.0" encoding="ISO-8859-1"/>
<!-- Generator: ControlBuild -->
<CONFIGURATION NAME="new1" UUID="24b5357d-e0d5-4b7a-8dc0-c0f7ca7f418d"
MAIN_APPLICATION="sensor_belt" PROJECT="Example1" TARGETS_NUMBER="2">
<TARGETS>
<TARGET NAME="STB1" UUID="be33ca7-cb39-4494-bc1d-fd0844bf1c4"
HOST_ID="127.0.0.1" DEVICES_NUMBER="1" PORT="1024" SHAREDMEMORY="1">
<DEVICE NAME="STB_01"
UUID="471d27b-b912-488d-abad-6e7fd873d839" TASKS_NUMBER="1">
<TASKS>
<TASK NAME="T1"
PATH="C:\Example1\Applications\sensor_belt\hardwareArchitecture\new1\STB1\STB_01\T1\sensor_belt\organic\STB_01\T1\exec" SIGNATURE="1721534302" CYCLE_TIME="50">
<INPUTS>
UUID="9dfe9d39-9d8d-4238-83f0-4948003d25ed" ADD="6" />
<UUID="d2bd813-3c11-f0f5-89b9-ec997f37351" ADD="422" />
<UUID="c4c5acc-c1e7-45cc-9baa-d99c3c819238" ADD="10" />
<UUID="7d45f0c2-50e5-45f9-a856-9414e8380115" ADD="422" />
<UUID="421ecff6-7bd0-44d8-a5b0-ed1083a3a0c" ADD="9" />
<UUID="1ba4eac-1491-4cb3-92ac-4816a08b212" ADD="423" />
<UUID="173d6d5-94d2-419b-85f4-b6df4aca6b71" ADD="6" />
<UUID="2d967649-f142-4256-880e-b19526762553" ADD="8" />
<UUID="f959d30-db36-4009-ae50-4c0a5baedc29" ADD="424" />
<UUID="e882d46-4a02-46a1-add1-3aeccf3e42" ADD="11" />
<UUID="6b8e51a-c319-4632-9152-fe1225a4fb35" ADD="425" />
<UUID="cf9a872e-3871-4597-b0ba-96e27e7253ba" ADD="12" />
<UUID="60268c3f-5cecc41b-afe9-29d39b6d05ea" ADD="7" />
</INPUTS>
</TASK>
<OUTPUTS>
UUID="7e659a79-6a13-453a-9dbd-062a2c060cc6" ADD="5" />
<UUID="92cf6d06-cfcb-4dd4-9b43-77a6e2116fcc" ADD="423" />
<UUID="39636196-d91d-49bc-a60f-738456456f8f" ADD="424" />
<UUID="6a447b8a-2c4b-4ac8-a253-7c666dc15ff7" ADD="5" />
<UUID="f182ba8c-35fe-45f0-85d4-a28cd1ffe308d" ADD="11" />
<UUID="7b4dbc5-f968-4895-856c-f78cdd31c8e7" ADD="425" />
<UUID="4541d8e1-04c7-474a-9040-860ff192b7b9" ADD="12" />
</OUTPUTS>
</TASK>
</TARGETS>
</CONFIGURATION>
Chapter A: Designing CB automation systems in ControlBuild

<UUID>
</DEVICE>
</TASKS>
</DEVICES>
</TARGETS>
</TARGET>
</CONFIGURATION>