Innovative approach to the design and realisation of a virtual prototyping environment for manufacturing systems engineering

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/2134/8018](https://dspace.lboro.ac.uk/2134/8018)

Publisher: © D.A. Vera

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

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
Innovative Approach to the Design and Realisation of a Virtual Prototyping Environment for Manufacturing Systems Engineering

By

Daniel Alexandre Vera

Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy

December 2004

© by D.A. Vera – December 2004
Acknowledgments:

I sincerely thank my supervisors, Dr A. A. West, Dr R. Harrison and Prof. R. H. Weston without whom none of this research could have been initiated or achieved. I am deeply grateful for their personal support, encouragement and their invaluable help during the years that have been required to complete this research. My gratitude goes beyond the words I could write on this page.

I would also like to express my sincere appreciation to all members of the MSI Research Institute, who also are my friends and colleagues, and without whom working in the MSI would not be such a pleasure.

I address a particular thank to Dr R. P. Monfared and Mr S. McLeod whose help and friendship have been particularly precious, and from whom I have learned a lot. Another particular thank goes to Mrs Margaret Carden, whose help and kindness have been immensely comforting and helpful in completing this research project.

A special thought goes to Murielle, my partner and friend. I thank here for having been so patient and understanding with me during those years of research.

Finally, I would like to show my appreciation to Loughborough University, the Wolfson School of Manufacturing Engineering, and the Manufacturing System Integration Research Institute in particular for providing me with the material support during the completion of this research.
Même si je sais qu'elle s'arrêtera sans doute à la lecture de cette dédicace, cette thèse est dédiée à ma Mère. C'est une façon de lui exprimer ma gratitude, sachant bien que je ne pourrais jamais la remercier assez pour tout ce qu'elle m'a apporté.
Abstract:

The highly dynamic context in which manufacturing related enterprises have to operate have had a direct effect on the organisation which are concerned with the design and commissioning of production lines, also referred to as manufacturing systems (MS). There is an increasing need for methods, tools, and technologies that allow the time frame for the design of such systems to be reduced in order to gain competitive edge in the market.

The research presented in this thesis is part of the COMPAG/COMPANION (COMponent-based Paradigm for AGile automation, and COmmon Model for PArtNers in automatION) projects conducted at Loughborough University UK. The COMPAG/COMPANION project aims at designing and implementing new tools to support the engineering lifecycle of MS. The present research focuses on the specification, design and implementation of a Virtual Prototyping Environment (VPE) that allows three-dimensional, computer-based and dynamic models of manufacturing systems to be implemented and used as virtual prototypes prior to the final design and MS commissioning phases.

This research proposes a new approach to the design and implementation of VPE tools, aimed at providing support for the engineering of flexible machine technologies (referred to as Reconfigurable Manufacturing Systems), which rely on the use of modular technologies and component-based distributed machine control systems. This research is focused on two aspects of VPE tools' development, which are i) ensuring the consistency between real and virtual systems architectures, design tools and design processes, and ii) maximising the potential of 3D computer-based virtual model as a basis for distributed engineering collaboration. A so-called component-based (CB) approach to VPE tools' design and implementation is proposed, which radically contrasts with approaches commonly adopted by both the commercial and academic VPE developers. The VPE developed in the context of this research should ultimately enable more effective management of RMS complex engineering lifecycle by engineering partners who are globally distributed.

Keywords: Virtual Prototyping, 3D modelling, Manufacturing System Engineering, System Architecture, Component and Component-based Design, Virtual Reality Modelling Language (VRML).
Table of Contents:

Chapter 1 Introduction..................................................................................... 1-13
  1-1 Research scope and focus ....................................................................... 1-13
    1-1.1 Manufacturing industry and context .................................................... 1-13
    1-1.2 Manufacturing System (MS) design ....................................................... 1-13
    1-1.3 Distributed engineering ........................................................................ 1-13
  1-2 State of the art solution: Three-dimensional modelling and Virtual Machine Prototyping ................................................................. 1-14
  1-3 Lack of Provision .................................................................................... 1-15
    1-3.1 Virtual prototyping and manufacturing design process support .......... 1-15
    1-3.2 Virtual prototyping and VE partners collaboration support .............. 1-15
  1-4 Research Objectives ................................................................................ 1-16
    1-4.1 Research approach ................................................................................ 1-17
    1-4.2 Thesis structure .................................................................................... 1-18

Chapter 2 Literature Review ........................................................................... 2-21
  2-1 Virtual Organisation in Automotive industry ........................................ 2-21
    2-1.1 VE and organisation partitioning ............................................................. 2-22
  2-2 Concept of Agility in the domain of manufacturing engineering .......... 2-23
    2-2.1 Definition .............................................................................................. 2-23
    2-2.2 Characterisation of Agility in manufacturing ........................................ 2-25
    2-2.3 Conceptual approach to agility performance characterisation .......... 2-26
  2-3 Manufacturing Systems Flexibility ....................................................... 2-28
    2-3.1 Review of manufacturing systems (MS) flexibility .............................. 2-28
    2-3.2 Production systems classification .......................................................... 2-30
  2-4 Enterprise Integration ............................................................................ 2-32
    2-4.1 Integration concept .............................................................................. 2-32
    2-4.2 Integration, Information Systems and Technologies ............................. 2-33
    2-4.3 Integration and Virtual Enterprises ....................................................... 2-35
  2-5 Use of three dimensional (3D) modelling technologies in the domain of manufacturing ................................................................. 2-36
    2-5.1 Virtual Reality ..................................................................................... 2-36
    2-5.2 Virtual prototyping ................................................................................ 2-38
    2-5.3 Virtual Manufacturing .......................................................................... 2-39
  2-6 Chapter Overview ................................................................................... 2-45

Chapter 3 A conceptual approach to Virtual Prototyping Environments (VPE) requirements specification ............................................................. 3-46
  3-1 Production system engineering lifecycle and context: Research case study 3-46
    3-1.1 Cross Hüller machine manufacture process description ..................... 3-46
    3-1.2 General approach to VPE requirements specification ......................... 3-50
  3-2 Engineering-related functionality of Virtual Prototyping Environment (VPE) 3-51
    3-2.1 Cross Hüller Manufacturing design process analysis .......................... 3-51
    3-2.2 Potential Use of 3D Virtual Prototyping Environments (VPE) in support of design of manufacturing systems ....................................... 3-53
    3-2.3 Characterisation of manufacturing systems (MS) and definition of modelling requirements ................................................................. 3-54
    3-2.4 Virtual Prototyping Environment (VPE) design / implementation .... 3-57
  3-3 Communication-related functionality of Virtual Prototyping Environment (VPE) ................................................................. 3-64
3-3.1 Collaboration between Virtual enterprise partners: generalisation of the machine design and build case study ................................................................. 3-66
3-3.2 Use of Virtual Prototyping Environment to support partners' collaboration... 3-66
3-4 Chapter overview ............................................................................. 3-71

Chapter 4 Component-based approach to Virtual Prototyping Environment (VPE) implementation ................................................................. 4-73

4-1 Chapter Introduction ....................................................................... 4-73
4-2 Overview of the concept of systems architecture ........................... 4-74

4-3 Flexible Manufacturing System architecture for Re-configurable Component-based Machine ................................................................. 4-77
4-3.1 Component-Based Paradigm for distributed machine control logic .... 4-77
4-3.2 Common hierarchical model ............................................................ 4-79
4-4 Three Dimensional (3D) models of machine systems ..................... 4-79
4-5 Component concept in Software engineering .................................. 4-81
4-5.1 Software Component definition ...................................................... 4-81
4-5.2 Software component functions and level functionality .................. 4-82

4-6 Review of various approaches to design and implementation of "modelling component" models and modelling software environment .... 4-85
4-6.1 Models of "modelling components" ............................................. 4-86
4-6.2 Analysis of "modelling component" models .................................. 4-87
4-6.3 Approach to modelling environment design and implementation .... 4-88
4-6.4 Analysis of VPE design and implementation approaches models .... 4-90

4-7 Portable Component-based Virtual Prototyping Environment (PoCo VPE) design ................................................................. 4-91
4-7.1 VPE generic description and requirements specification ............... 4-92
4-7.2 Overview of the PoCo component model and PoCo VPE design approach... 4-93
4-7.3 PoCo component infrastructure ...................................................... 4-96

4-8 Chapter Overview ........................................................................... 4-97

Chapter 5 PoCo modelling components design .................................... 5-99

5-1 Introduction .................................................................................... 5-99
5-1.1 PoCo modelling component’s functions overview ......................... 5-99
5-1.2 PoCo modelling component structure overview ............................. 5-100
5-1.3 PoCo Modelling element overview ................................................ 5-101

5-2 PoCo modelling elements description .............................................. 5-103
5-2.1 Modelling functions ..................................................................... 5-103
5-2.2 Structural functions ........................................................................ 5-109
5-2.3 Model utilisation and user interfaces ............................................. 5-114

5-3 PoCo Component model ................................................................. 5-117
5-3.1 Element class hierarchy ................................................................. 5-117
5-3.2 Initialisation sequences ................................................................. 5-118
5-3.3 PoCo distributed Logic engine’s elements interaction ..................... 5-120
5-3.4 PoCo Link Point (LP) Element and assembly sequence ................. 5-123
5-3.5 Example of PoCo model implementation ...................................... 5-124

5-4 Chapter overview ........................................................................... 5-128

Chapter 6 PoCo VRML object models ................................................. 6-130

6-1 VRML and Object Orientation ...................................................... 6-130
6-1.1 Review of Object Oriented paradigm .......................................... 6-130
6-1.2 VRML and object model ............................................................... 6-132
6-1.3 VRML Proto node ......................................................................... 6-133
6-1.4 VRML and inheritance ................................................................. 6-136
6-1.5 Approaches to VRML based inheritance ...................................... 6-137

6-2 PoCo VPE Objects' models ......................................................... 6-141
6-2.1 Element VRML file model ............................................................ 6-142

1-5
Chapter 7 PoCo VPE and real system engineering tools integration

7-1 Chapter introduction
7-2 Real/virtual system design data
7-3 Real/virtual system data integration

Chapter 8 Research Cases Studies

8-1 Chapter overview
8-2 Lamb Technicon Test Machine
8-3 Krause Test Rig
8-4 Ford Test Rig
8-5 Asda supermarket warehousing machine

Chapter 9 Conclusion

9-1 Overview of research objectives
9-2 Research Contribution
9-3 Future development for the present research
9-4 Suggestions for future direction of research and general conclusion

6-2.2 Component VRML file model
6-2.3 PoCo model VRML file model

6-3 Chapter overview
Chapter 7 PoCo VPE and real system engineering tools integration

7-1 Chapter introduction
7-2 Real/virtual system design data
7-3 Real/virtual system data integration

Chapter 8 Research Cases Studies

8-1 Chapter overview
8-2 Lamb Technicon Test Machine
8-3 Krause Test Rig
8-4 Ford Test Rig
8-5 Asda supermarket warehousing machine

Chapter 9 Conclusion

9-1 Overview of research objectives
9-2 Research Contribution
9-3 Future development for the present research
9-4 Suggestions for future direction of research and general conclusion
Table of Figures:

Figure 1-1: From 3D modelling to Virtual manufacturing, and Virtual enterprise partner’s collaboration. Adapted from Saadoun et al. [78] .......................................................... 1-14
Figure 1-2: Research objectives general overview ........................................................................................................ 1-16
Figure 1-3: General approach adopted in this research to specify and initiate the implementation of an innovative virtual Prototyping Environment to support the design of manufacturing systems ............................................................................................. 1-18
Figure 2-1: VE Partitioning considered at Organisation, Engineering, and system Level, and effect on manufacturing organisation IS/IT infrastructure ........................................................................ 2-23
Figure 2-2: Change capability and change capability rate as two metrics to evaluate systems’ agility .................................................................................................................................................... 2-27
Figure 2-3: Productions system classification. Adapted from Rampersad (1994) [63] ............................................. 2-30
Figure 2-4: Levels of enterprise integration and corresponding domain IT/IS integration. Adapted source F. Vernadat [26] .......................................................................................... 2-34
Figure 2-5: Virtual Enterprise and IT/IS infrastructure. Source M. Upton et al. [24] ........................................ 2-35
Figure 2-6: Virtual Reality (VR) system architecture. Source Burdea et al. [79] .......................................................... 2-37
Figure 2-7: Components of a virtual prototype. Source Wang [143] ............................................................................. 2-39
Figure 2-8: Virtual Manufacturing and main trends in manufacturing industry. Combined Source Bullinger [77] and Saadoun [78] ............................................................................................................. 2-40
Figure 2-9: Typical virtual environment architecture. Source Xu et al. [7] ............................................................................... 2-43
Figure 3-1: Cross Hüller / Ford Motor Companies Pre sale contract phase ........................................................................ 3-47
Figure 3-2: Cross Hüller Machine concept and detailed design phases ........................................................................... 3-48
Figure 3-3: Cross Hüller Pre Commissioning phase ........................................................................................................ 3-50
Figure 3-4: issues in achieving engineering domain design processes concurrency ........................................................................... 3-51
Figure 3-5: Design phase separation and data format and translation issues ........................................................................... 3-52
Figure 3-6: Use of Virtual Prototyping Environment and system prototypes for vertical and horizontal integration ........................................................................................................... 3-54
Figure 3-7: Invariant Design Basis of both Flexible and Re-configurable Manufacturing Systems types, and consequence on Change capability and Change Capability Rate ...................................................................................... 3-56
Figure 3-8: Generic description of production system virtual prototypes lifecycle phases .......................................................................................................................... 3-57
Figure 3-9: FMS virtual prototype lifecycle, and Example of Tecnomatix ......................................................................................... 3-60
Figure 3-10: RMS virtual prototypes lifecycle ........................................................................................................... 3-61
Figure 3-11: Current approach to the design of Virtual Prototyping Environment adapted to FMS system lifecycle ........................................................................................................... 3-62
Figure 3-12: Component-based approach to Virtual Prototyping Design and Implementation 3-63
Figure 3-13: Various media and their capability to represent tacit knowledge. Source Yap et al. [109].

Figure 3-14: Use of 3D machine prototypes as a basis for partners communication.

Figure 3-15: Use of VPE as communication media for partners' collaboration.

Figure 4-1: General approach to the design and realisation of a component-based VPE.

Figure 4-2: System architecture as central concept to system design.

Figure 4-3: Manufacturing System functional domains.

Figure 4-4: Manufacturing system views.

Figure 4-5: Bosses-above-Bosses, boxes-within-boxes, and function-related representation of systems hierarchical decomposition hierarchy model.

Figure 4-6: Component-based machine control architecture and Process Definition Environment (PDE tool) for control node configuration.

Figure 4-7: Component-based machine hierarchy and concept of module.

Figure 4-8: 3D modelling components as a common and intuitive view of the various system aspects.

Figure 4-9: Component Meta model. Source Van Baelen [121].

Figure 4-10: General overview of various approaches to software constructs and corresponding level of functionality.

Figure 4-11: Modelling component as defined by Salmela et al. [89].

Figure 4-12: Modelling component as defined by Adolfsson et al. [8].

Figure 4-13: Salmela et al. [89] Component-based modelling environment and process.

Figure 4-14: Example of integration between 3D modelling and machine logic editing environment using event interface and software integration infrastructure. Source Adolfsson et al. [8].

Figure 4-15: Inconsistency between Component and component-based systems models, and implementation.

Figure 4-16: Generic description and ideal requirements for a Virtual Prototyping Environment for component-based model implementation.

Figure 4-17: PoCo (Portable Component-based) approach to Virtual Prototyping Environment (VPE) implementation.

Figure 4-18: from monolithic to distributed modelling environment.

Figure 4-19: PoCo VPE implementation approach.

Figure 5-1: PoCo modelling elements functions.

Figure 5-2: PoCo modelling elements as functional objects, and PoCo modelling component as structural modelling objects.

Figure 5-3: General system descriptive aspects.
Figure 5-4: PoCo DYN modelling elements configuration parameters (attributes) functions (methods) and interface .......................................................... 5-105
Figure 5-5: Example of Pusher actuator modelled using PoCo DYN modelling elements. 5-106
Figure 5-6: Screen shot of DYN information display functions output ....................... 5-107
Figure 5-7: PoCo STA modelling element's parameters functions interfaces and sub classes 5-108
Figure 5-8: COND PoCo modelling element's parameters, functions and communication interface ........................................................................ 5-109
Figure 5-9: Component assembly sequence and associated 3D transformations .......... 5-111
Figure 5-10: Link Point modelling elements parameters, functions, interfaces, and sub classes .................................................................................. 5-112
Figure 5-11: Schematic representation of the approach to the automatic interlock event routing paths implementation supported by the INT modelling element's functions 5-113
Figure 5-12: VP modelling element attributes and interface ...................................... 5-115
Figure 5-13: AV PoCo modelling element attributes, functions, and interface .......... 5-116
Figure 5-14: PoCo modelling framework element Class hierarchy .............................. 5-117
Figure 5-15: Naming initialisation sequence .............................................................. 5-118
Figure 5-16: LP element initialisation sequence ......................................................... 5-119
Figure 5-17: Logic simulation engine and inter element communication .................... 5-121
Figure 5-18: Auto and manual mode configuration of DYN elements ....................... 5-122
Figure 5-19: PoCo LP elements and assembly sequence ........................................... 5-123
Figure 5-20: Asda machine Pusher Component editing process overview ................. 5-125
Figure 5-21: PoCo modelling component Class .......................................................... 5-126
Figure 5-22: Overview of Asda machine components and composition layout .......... 5-128
Figure 6-1: Class hierarchy and inheritances mechanism overview. Adapted from Albir [47] 6-132
Figure 6-2: Encapsulation, low level modelling language and VRML proto nodes ....... 6-134
Figure 6-3: Example of VRML code encapsulation using VRML proto node, and example of Proto node instantiation (both Proto and ExternProto) ...................... 6-135
Figure 6-4: Newtonians modelling objects Class Hierarchy (adapted from C. Beeson [45]).... 6-138
Figure 6-5: VRML++ and corresponding VRML code after processing (adapted from Diehl [88]) ................................................................. 6-140
Figure 6-6: PoCo Modelling element VRML file model ............................................ 6-143
Figure 6-7: PoCo element's nested PROTO communication scheme ....................... 6-145
Figure 6-8: Example of PoCo Link Point modelling element ................................. 6-146
Figure 6-9: PoCo modelling component file model, and duplication of elements fields ... 6-148
Figure 6-10: Example of VRML code describing a simple model composed of two components................................................................................................................. 6-149
Figure 7-1: Issue in implementing virtual machine prototypes and virtual prototyping environments ............................................................................................................................................ 7-152
Figure 7-2: Data model as a link between real and virtual system .......................................................................................................................................................................................................................... 7-154
Figure 7-3: Examples of sequential machine logic data formats .......................................................................................................................................................................................................................... 7-157
Figure 7-4: CAD data translation alternatives and issues .......................................................................................................................................................................................................................... 7-159
Figure 7-5: Type of data contained into CAD assembly models .......................................................................................................................................................................................................................... 7-160
Figure 7-6: Test and rating of CAD software VRML exporting capabilities .......................................................................................................................................................................................................................... 7-161
Figure 7-7: VRML CAD file output and post processing approach to model component implementation .......................................................................................................................................................................................................................... 7-165
Figure 7-8: Example of PoCo modelling element 3D geometry configuration .......................................................................................................................................................................................................................... 7-166
Figure 7-9: Integration between machine virtual prototyping and machine logic editing environment .......................................................................................................................................................................................................................... 7-168
Figure 7-10: Machine logic translation approach to the implementation of virtual machine prototype behavioural logic .......................................................................................................................................................................................................................... 7-169
Figure 7-11: Component-based approach to the implementation of the PoCo internal logic simulation engine .......................................................................................................................................................................................................................... 7-171
Figure 7-12: Issues in data formats translation / mapping, and structure of modelling and engineering data .......................................................................................................................................................................................................................... 7-173
Figure 7-13: PDE PoCo logic data Mapping Environment and process .......................................................................................................................................................................................................................... 7-174
Figure 7-14: Example of XML machine logic format generated by the COMPANION Process Definition Environment (PDE) tool, and COMPANION component-based machine hierarchy .......................................................................................................................................................................................................................... 7-175
Figure 7-15: PoCo model integration with external logic related event generating environment. Use of the COMPANION broadcaster integration infrastructure as case study .......................................................................................................................................................................................................................... 7-177
Figure 8-1: 3D CAD model (top left), Schematic top view (bottom left), and photograph of Lamb Technicon Test Machine (right) .......................................................................................................................................................................................................................... 8-182
Figure 8-2: Primary approach to the implementation of a virtual prototyping environment for manufacturing system .......................................................................................................................................................................................................................... 8-184
Figure 8-3: Screen Shot of the 3D and kinematics modelling software implemented as part of the COMPANION machine virtual prototyping environment .......................................................................................................................................................................................................................... 8-185
Figure 8-4: Screen shot of the PDE COMPAG component-based machine logic implementation and debugging environment .......................................................................................................................................................................................................................... 8-186
Figure 8-5: Screen shot of the machine 3D geometry/kinematics and logic data mapping software environment .......................................................................................................................................................................................................................... 8-187
Figure 8-6: Screen Shot of the PDE logic simulation environment with an embedded view of
lamb test machine virtual prototype ............................................................................. 8-191
Figure 8-7: Schematic and 3D view of Krauser Test Rig .................................................. 8-193
Figure 8-8: Approach adopted for the second development phases of a virtual prototyping
environment for manufacturing systems ....................................................................... 8-195
Figure 8-9: Screen shot of the component assembly interfaces, using link points feature. 8-196
Figure 8-10: Overview of the Ford Test Rig mechanical layout, and screen capture of the
corresponding 3D model ............................................................................................. 8-201
Figure 8-11: Third phase of the Virtual Prototyping Environment development. Concept of
modelling component as a key stone of the VPE environment design, and Virtual
Prototypes implementation process ............................................................................. 8-203
Figure 8-12: Sample of Asda test machine’s state based diagram logic control representation
.................................................................................................................................... 8-209
Figure 8-13: Schematic representation and virtual prototypes of Asda warehousing sorting sub
system ......................................................................................................................... 8-210
Figure 8-14: VRML Modelling component and internal elements structure. Use of
configuration and composition tools during the component-based model lifecycle. 8-211
Figure 8-15: PoCo component geometrical assembly user interface (link point selection /
assembly) .................................................................................................................... 8-214
Figure 8-16: Approach to the implementation PoCo VPE portability ............................... 8-214
Figure 9-1: An innovative approach to the specifications and design of manufacturing
systems’ Virtual Prototyping Environments .................................................................. 220
Figure 9-2: An innovative, approach to the implementation of 3D modelling components,
resulting from the investigations of various types of components .......................... 221
Figure 9-3: Modelling openness and implementation of additional modelling elements and
associated functions ................................................................................................. 224
Table of Tables:

Table 2-1: Various definition of the concept of Agility in the domain of manufacturing. Source: Gunasekaran [14] ................................................................. 2-24
Table 2-3: Flexibility types. Source Sethi et al. [62] ................................................................. 2-29
Table 2-4: Summary table of manufacturing systems comparison. Source M. G. Mehrabi [71] ........................................................................................................... 2-31
Table 8-1: Lamb Test Machine operation sequence description ........................................ 8-182
Table 8-2: Lamb Test Machine Transfer sub system, Transfer component, Raise/lower element hierarchy, Raise/Lower element’s state and interlock description ............. 8-183
Chapter 1  Introduction

1-1  Research scope and focus

1-1.1  Manufacturing industry and context

Today's market is characterised by a demand for products, with quality, diversity and cost have to match the consumer expectancies [1]. In particular, automotive industry is characterised by frequently changing and technologically advanced products. In addition to product related constraints, sociological, technological, and political factors also affect the dynamic and complex nature of the context in which manufacturing organisations have to operate [2]. Global markets and the easy access to knowledge and information are the main characteristics of today's highly dynamic and competitive industrial environment [1]. A customer driven [2] and competitive market is often characterised by a high level of uncertainty and continuous and unpredictable changes [3] [4]. This context has led modern manufacturing organisations to implement new manufacturing paradigm and adopt new tools to support rapid design and implementation of systems. Some of the most relevant in the context of this research are the concepts of Virtual Enterprises (VE), flexibility and flexible machine technologies, and virtual machine prototyping.

1-1.2  Manufacturing System (MS) design

The focus of this research is to investigate the potential of three 3D computer-based modelling, virtual prototyping, and simulation tools to support the activities of industrial partners involved in the design and implementation of production (or manufacturing) systems (MS). The design and change of MS for the automotive industry, requires effective collaboration between several partners (machine manufacturers, technology vendors, machine customers, sub contractors, etc.), who each possess the knowledge and expertise corresponding to a specific domain of design [4] [5]. The pressure to minimize the time-to-market of products leads to a compression of project timelines and necessitates a high degree of simultaneous engineering between product and MS design [6]. The communication and engineering tools used to support those processes must enable distributed partners to readily resolve problems and implement engineering changes that constantly occur as the product and manufacturing system design matures.

1-1.3  Distributed engineering

The case study providing a real life background to this research is representative of the way most modern car manufacturing partnerships operate. Ford (and its partners Mazda, Jaguar,
Alantifarturiri, 'lstrms Intrxrarinu Research Incriture, 14)I hbornrrgh Irnrirrusat), and Volvo) is aggressively pursuing strategies to rationalize the global engineering of vehicles. The latest collaborative engine programme (termed 14/15) between Ford and Mazda is to produce a new generation of four-valve cylinder in-line petrol engines. The 14/15 programme involves partners distributed around the world. There follows an implicit need to be able to develop methods and tools that co-ordinate and support the inter-working and decision-making of distributed engineering teams in such a way that multiple viewpoints of team members are considered.

1-2 State of the art solution: Three-dimensional modelling and Virtual Machine Prototyping

There is a great potential to apply virtual reality (VR) and 3D modelling technologies to aid MS design, system testing and process control and validation [7]. 3D computer-based modelling and simulation tools have proven to be highly effective for virtual prototyping, system visualisation and testing and analysis of “what if scenarios” [8]. 3D graphical representation of MS provides a “common model” [9] of the system being designed that facilitate collaboration between engineers from different domains [14]. Increasing hardware graphic’s performances, and new modelling formats and software allow highly realistic model to be implemented. Much more importantly, the capability to simulate real system behaviours consistently allows modelling and simulation capabilities to be merged [14], and therefore provide a solid basis upon which effective virtual prototyping tools can be realised.

As shown in Figure 1-1, 3D modelling and virtual prototyping can serve different purposes. Virtual Prototyping has widely been used for product and product parts prototyping [14] [14] [14], in order to assess the overall appearance or basic functionalities (e.g. kinematic simulation of simple mechanisms). In the domain of production systems design the use of 3D-based simulation is relatively new. The development of Computer Aided Design and
Manufacturing (CAD/CAM) software has allowed product design and product manufacturing to be linked by providing simulation capabilities for Computer Numerical Control (CNC) machines for instance. A number of research projects and commercial software development efforts have emerged from the need to achieve timely and cost effective design and change of MS. These efforts have focused on implementing so-called “digital manufacturing” environments used to support the design, configuration, maintenance and monitoring of MS lifecycles [109] [14] [14] [14].

1-3 Lack of Provision

1-3.1 Virtual prototyping and manufacturing design process support

As stated by currently available Virtual Prototyping environments (VPEs) are restricted to modelling “well established” types of MS [73] [8] [109] such as NC precision machining tools, multi-axis industrial robots and inspection machines. The need for agility has led MS engineers to seek improved MS re-configurability and re-usability [8]. This in turn has led to the investigation and use of new flexible machine technologies such as modular machine design and distributed control systems. Current VPEs cannot be used effectively to support the design lifecycle of Re-configurable Manufacturing Systems (RMS). There follows a need for new tools and methods adapted to the architectural and technological aspects of new flexible machine technologies.

1-3.2 Virtual prototyping and VE partners collaboration support

Currently available Commercial Virtual Prototyping Environments (VPE) are essentially “company specific” and are built using proprietary modelling technologies [7] (i.e. proprietary modelling formats, specific software integration infrastructure and services). This typically results in high deployment and maintenance costs, and involves logistical issues that force industrial partners who wish to share a common virtual prototyping software solution into inflexible and long-term partnerships [5]. Such an approach is not adapted to today’s approach to partners’ collaboration and integration (referred to as Virtual Enterprises (VE) or Virtual Organisation (VO)). VE approach to partners’ integration seeks to provide short time, adaptive and responsive collaboration, that allow enterprise to react effectively (i.e. in term of time and resources requirements) to changes.
1-4 Research Objectives

This research has been initiated from a general observation that, despite the potential advantages of using virtual machine prototypes to support the lifecycle of MSs, the virtual prototyping and 3D-based simulation is still considered as unproductive tools. The research presented in this thesis investigates a new approach to the specification and implementation of an innovative VPE that can enable virtual prototyping as the core activity in the distributed engineering of RMS.

This research has been conducted with respect to the lack of provision of VPEs currently available in commercial and public domains. First, the emergence of new approaches to flexible machine design and build processes imposes new requirement regarding the design and implementation of VPE tools. Secondly, the organisational context in which VPE tools are to be used imposes new constraints on practical realisation of VPE software environments. Consequently, part of the objectives of this research was to identify the limiting factors of existing VPEs, which are the result (cf. Figure 1-2) of the i) approach adopted specify and design VPE software applications, and of ii) the constraints and limitations imposed by currently used modelling technologies.

The performances requirements of an innovative VPE tool could be described as:

- **Model re-configurability**, namely the VPE functions and model design features that allow a given set of model and modelling parameters to be modified effectively in
order to broaden the range of machine configurations that can be modelled/tested in a given time frame

- **Model re-usability**, namely the implementation of VPE functions and modelling mechanisms that allow previously capitalised modelling efforts/knowledge to effectively be managed and re-used across different design projects

- **Modelling process manageability / simplicity**, to hide the complexity related to the specific activity of 3D computer modelling from the VPE users (e.g. system engineers), and therefore to maximise the value of VPEs tools as engineering tools

In addition:

- **Portability**, refers to i) the capability of the VPE modelling environment to be easily and effectively deployed among engineering partners who do not necessarily have the same level of Information technologies / Information Infrastructures (IT/IS) expertise or who do not have access to highly specialised software infrastructures, and ii) the capability to visualise and simulate machine models without the need for specific modelling knowledge or highly specialised software infrastructures

- **Functional openness** refers to the capability to extend the functions of a given VPE. The extended functions can be related to the modelling of various types of system (i.e. modelling functions), or to the functions which allow machine model to be exploited for engineering purposes (engineering related functions).

### 1-4.1 Research approach

As shown in Figure 1-3, the research presented in this thesis was initiated by the analysis of the current approach to VPE tool design and implementation. The purpose of this task was to highlight the lack of provision with respect to the engineering and communication capabilities of existing VPE tools. The relationship between different types of VPE and the engineering lifecycles of various types of MS was investigated in order to understand the industry requirements for MS prototyping. In the same way, the nature of modern manufacturing related enterprise and organisations was investigated in order to capture new requirements of such organisations in terms of general communication and engineering collaboration.
From this analysis phase, requirements regarding various characteristics that an innovative VPE should exhibit and various functions that it should provide were determined. From this set of requirements, a possible solution was outlined as a set of conceptual specifications for the design and development of an innovative VPE. An investigation of existing modelling technologies and tools, lead to the selection of particular modelling formats and programming languages, which would allow the requirements to be implemented. A conceptual design phase was mainly focused on the translation of the initial VPE functional specification into a software design and architecture. At this stage, the investigation of programming concepts was conducted in order to guide the VPE implementation phases, which consisted of the detailed implementation of the VPE functions and software environment.

1-4.2 Thesis structure

The thesis is organised as follow:

In Chapter 2, the literature relating to the current state of the art manufacturing paradigm is reviewed. It is believed, that the understanding of general manufacturing concepts is essential
in developing a 3D modelling environment that can be deployed effectively in a modern engineering context.

In Chapter 3, the conceptual approach adopted to determine the VPE requirements specification is described. The requirements of VPE tools as i) engineering tool which should provide effective support for system modelling and ii) as communication tool which should provide effective basis for distributed VE partners collaboration are highlighted.

In Chapter 4, the component concept (as adopted in the research conducted in the Manufacturing System Integration Research Institute (MSIRI)), is introduced and reviewed from the various perspectives which are required to understand the development of the VPE software tool conducted in this thesis.

Chapter 5 consists in a detailed review of the functional and architectural design of a Portable Component-based Virtual Prototyping Environment (PoCo VPE). The issues associated with the modelling of each aspect of RMS are highlighted and the corresponding PoCo VPE software constructs and functionalities are detailed.

A review and analysis of the modelling formats and languages used to implement the PoCo VPE software (i.e. the Virtual Reality Modelling Language (VRML)) is provided in Chapter 6. The issues associated with the use of a standard modelling language to implement highly specific engineering applications (i.e. virtual prototyping tools) are highlighted.

The focus of Chapter 7 is on providing an overview of the integration of PoCo VPE tool with real MS engineering tools. The Chapter focuses on i) the integration between PoCo VPE and CAD software and ii) on the integration between PoCo VPE and the Process Definition Environment (PDE) developed in the context of the COMPAG/COMPANION project as a machine control editing tool.

The real case studies used throughout the successive development phases of the PoCo VPE environment, are reviewed in Chapter 8. The case studies consist of two full scale demonstrators used at Lamb Technicon and Krauser machine builders sites, of a university test rig provided by Ford and finally, and of a conveying system used by Asda Warehousing. The development of the PoCo VPE concepts and architecture as a result of each case study is traced throughout the chapter.
A review of what is believed to be the contribution of this research in the domain of manufacturing system virtual prototyping and in the domain of modelling is provided in Chapter 9. Aspects of the present research that need to be further investigated and developed are highlighted and directions for future research efforts in the domain of VPE development are proposed.
Chapter 2   Literature Review

2-1 Virtual Organisation in Automotive Industry

Virtual Organisations (VO) and Virtual Enterprises (VE) have been described as the architecture of today’s and tomorrow’s manufacturing organisations [32]. Falkenberg [53] provides a general definition of organisations as “large and one-man companies, profit and non-profit-oriented organisations, clusters of companies interacting with each other, or even the community of all Internet users and similar communities”. Proper [49] defines organisations as a special kind of system, being normally active and open, and comprising the conception of how an organisation is composed and how it operates (i.e. performing specific actions in pursuit of organisational goals, guided by organizational rules and informed by internal and external communication). Systemic properties of organisation are their response to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment [49]. The definition of an organisation by Senge [93] is as follow: “a group of people who continually expand their capacity to create the results they truly desire, where new and expansive patterns of thinking are nurtured, where collective aspiration is set free, and where people are continually learning to see the whole together”. This definition places emphasis on organisation as “aware” or self-organising systems, whereas enterprise can be described more as tightly structured and possibly hierarchically controlled organisational sub sets. On the other hand, the term enterprise seems to be more specific in defining the way an organisation achieves its goals. For instance, Vernadat [38] defines an enterprise as a socio economic organisation created to produce products or to provide services to make profit, and more exactly as a system made of a large collection of concurrent business process executed by a set of functional entities (or resources) that contribute to business objectives.

Constantly changing and highly competitive manufacturing contexts require organisations, enterprises and facilities that are significantly more agile than existing one [8], and more effective regarding the final output (e.g. production time, product quality, cost) than existing one. The concept of Virtual Organisation potentially addresses to these needs, by defining a new architecture for manufacturing organisations that consists of the integration of individual, highly efficient functional entities, which each possess the knowledge and expertise required to achieve the VO’s overall objectives [33]. The final goal of VO might be very complex in nature, as for instance the introduction to the market of a new car product. Currently this involves the simultaneous activities of many companies belonging to different industry
sectors and with different competencies (e.g. car, production system design, but also product marketing). It should be noted that Henry et al [30] and Fujii et al, [28] state that the emergence of manufacturing VO has been made possible by the growing performances of information and communication technologies. The concept of VO has been extended at the enterprise level where several small size enterprises (e.g. sub contractors, technologies vendors) collaborate on a sub set of the VO goals [32] [34]

2-1.1 VE and organisation partitioning

2-1.1.1 Functions and processes partitioning
Enterprise specialisation emerges from the need to thrive in a highly competitive environment that forces enterprises to focus on their core competency [22] [32] [34]. Individual organisations often focus on simplifying and optimising their internal processes and infrastructure in the pursuit of leanness [23] [26]. This provides several advantages including easier enterprise function management and a more easily extendable task force since knowledge and skill required are limited to a specific domain of activity [5]. In the same way, engineering tools and internal process management can be made simpler and more effective. Although, functional decomposition allows enterprises to be more competitive and effective in their respective domains of activity, it also requires individual enterprises to be re-integrated in order to achieve effective collaboration at the VO / VE level. The specialisation of VO / VE partners often results in each partner having a very specific and possibly narrow perspective on the overall goal [5], which makes the overall project management more complex.

2-1.1.2 Engineering Infrastructure partitioning
As shown in Figure 2-1, a likely consequence of the functional partitioning described above is a partitioning of the manufacturing organisations’ Information Technologies and Information Systems (IT/IS) infrastructure and services. At IT level of the manufacturing infrastructure, collaboration infrastructure, and communication channels between globally distributed VE partners are broken down in a set of networks implemented upon heterogeneous IT technologies [72]. As an example, before the widespread adoption of standard Internet technologies, implementing Wide Area Networks to ensure effective communication between distributed partners involved large investment in dedicated lines (e.g. ISDN, T1) and high IT maintenance cost [5]. From an IT perspective, any VE therefore requires additional effort to ensure that geographically distributed partners can overcome communication barriers and collaborate effectively on their common goals.
From an Information System (IS) perspective, the implications of VEs regarding effective engineering collaboration are significant too. The various companies that compose any VE are likely to adopt software from different developers or even use “in house” software to implement the infrastructure that supports their engineering processes [4]. An obvious reason for software diversity is that each partner will typically operate in a different domain of engineering. This usually leads in what is commonly referred to as “islands of automation” [5] and simulation [89]. VE partners may also have different level of IT/IS expertise, which is the case when a large end user and Small and Medium Enterprises (SME) sub contracting partners for instance) form consortia. Typically, when consortium members have different levels of IT/IS expertise, IT/IS is only a limiting factor for collaboration effectiveness, but also a determinant criterion for partner selection [36].

2-2 Concept of Agility in the domain of manufacturing engineering

2-2.1 Definition

The application of the agility concept in manufacturing domains has been the subject of significant research effort in recent decades. This reflects increased uncertainty in the
environment in which manufacturing enterprises have to operate [24] [26] [9]. Increased competition and increased frequency and magnitude of political, economical and sociological, changes [34], contribute to the uncertain and dynamic nature of manufacturing [23] [24]. In addition, the products, processes and resource systems created and used by manufacturing organisations have become more complex. Products often embed complex technologies [24], which require increased types and levels of competencies, and infrastructures [28] in order to achieve product and production systems realisation, operation and change, timely and effectively [6].

<table>
<thead>
<tr>
<th>Authors</th>
<th>Agility Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeVor And Mills (1995)</td>
<td>Ability to thrive in a competitive environment of continuous and unanticipated change and to respond quickly to rapidly changing customer drive markets</td>
</tr>
<tr>
<td>Booth (1996)</td>
<td>More flexible and responsive</td>
</tr>
<tr>
<td>Gupta and Mittal (1996)</td>
<td>Agile stresses the importance of being highly responsive (...) while simultaneously striving to be lean.</td>
</tr>
<tr>
<td>Hong et al. (1996)</td>
<td>Flexibility and rapid response to markets constraints</td>
</tr>
<tr>
<td>Kusiak and He (1997)</td>
<td>Driven by the need to quickly respond to changing customers requirements</td>
</tr>
<tr>
<td>Gunasekaran (1997)</td>
<td>Capability to survive in a competitive environment of continuous and unpredictable changes</td>
</tr>
</tbody>
</table>

Table 2-1: Various definition of the concept of Agility in the domain of manufacturing. Source: Gunasekaran [26]

As a result agility has emerged as a fundamental concept in manufacturing domains where it is perceived that business value and competitiveness can increase if a manufacturing enterprise has capabilities to identify, rapidly react to, and effectively cope with unpredictable changes [24]-[26]. Table 2-1 summarises some of the “agility definitions” reported in the literature. Gunasekaran [26] also highlights the fact that manufacturing agility is perceived differently depending on the type of industry and on the domain of activity considered. To date, however, the literature on agility tends to characterise an organisation’s capabilities [23] [26], and does not directly aim at measuring performances which can directly be linked to specific manufacturing tools, technologies or methods.
2-2.2 Characterisation of Agility in manufacturing

A classification of the research on agility in Manufacturing Organisations was conducted by Sanchez [24]. The results of this review are summarised in Table 2-2. This classification shows that the largest number of citations was attributed to the domain of IT/IS, followed by supply chain, product and manufacturing systems (MS) design.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Sub Topic</th>
<th>No of citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product and manufacturing system design</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Process planning</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Production planning scheduling and control</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Facilities</td>
<td>Design locations</td>
<td>5</td>
</tr>
<tr>
<td>Material handling and storage</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Information systems</td>
<td>Integrated information systems</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Information system designed for supporting specific areas and activities</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Architecture requirement and implications</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Information exchange</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Evaluation of information models</td>
<td>1</td>
</tr>
<tr>
<td>Supply chain</td>
<td>Strategies</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Partner selections</td>
<td>9</td>
</tr>
<tr>
<td>Human factors</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Baseness practices and processes</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>


It is interesting to note that Sanchez [24] classifies product and MS design into the same category, and states that this is a very specific focus of concern. It should also be noted that the domain of IT/IS infrastructure design and management (designated as Information Systems) seems to be tightly coupled to implementing the agility concept within manufacturing enterprises. Sanchez classification of agility types does not provide a useful definition of key agility “drivers” (i.e. the needs for agility) and “providers” (i.e. tools which provide agility) as defined by Sharifi et al. [3] [24]. Rather, it does indicate manufacturing activities that are potentially concerned with agility. Gunasekaran [26] made similar observations about agility to those of Sanchez by identifying so called “agile manufacturing strategies and techniques” to achieve various manufacturing objectives (i.e. market focus, strategic objectives, flexibility). Gunasekaran [26] classes agility types into four main manufacturing domains, namely strategic planning, automation/IT/IS, Virtual Enterprise and product design.
This classifications of agility types appear vague because they place at a same level of importance i) strategies related to human resource management, strategic objective and decision making activities, legal issues, ii) activities involved in product, process and production system engineering, production activities various manufacturing tools (SAP/ERP, CAD/CAM, CAPP, E-Commerce) and iii) general manufacturing concepts (flexibility, virtual organisations, manufacturing integration). On the other hand, Gunasekaran [26] proposed a framework for the development of agile manufacturing based on the following major strategies and technologies for achieving this goal, namely:

- Partnership formation and supplier development
- IT in manufacturing
- Enterprise Integration and Management with the help of advanced IT/IS
- Virtual Reality tools and techniques
- Advanced manufacturing concepts such as CIM, Enterprise Integration CE, rapid prototyping
- Global manufacturing perspectives (physically distributed environment), E-Commerce, ERP, Internet, CAD/CAM

It can be argued that Gunasekaran’s [26] framework adds a useful dimension for agility analysis. However, it is interesting to note that most of the tools mentioned in this framework can be mapped onto views of manufacturing organisations described by Sharifi and al. [3] and Sanchez [24]. Another general conclusion is that many authors emphasise on i) the importance of IT/IS infrastructure and tools and ii) an need to focus on distributed partners integration as a basis for global manufacturing and iii) a more general need for flexible integration. It should be noted that “Virtual reality tools and techniques” have been identified as a new type of tool that can potential provide agility [26] [37] in a distributed engineering context.

2.2.3 Conceptual approach to agility performance characterisation

Weston [4] proposed a conceptual approach to characterising aspect of agility in manufacturing enterprises. Figure 2-2 illustrates various types of so-called “agents of changes” and environmental casual effects, which are similar in concept to what Sharifi et al. [3] characterises as “agility drivers”. Any manufacturing systems’ ability to cope with changes are characterised by Weston [4] with respects to their “change capability” and “change capability rate”. The change capability of a system can be measured in terms of
variety or reachable states. This change capability can be achieved in three distinctive ways (Weston [4]) namely via:

- Programmability which typically is used to realise predictable changes,
- Reactivity which is normally realised through changing the functionalities embedded into a system, and/or changing the way in which that functionality is structured so that late and possibly unanticipated change is accommodated,
- Proactivity which is achieved by i) forecasting/predicting needed changes, prior to their occurrence, and ii) planning to reprogram and/or react (via functional or structural changes) when the predicted requirement changes arise [4] [6].

Weston [70] defined change capability rate as the ability of a system to reach designated states at a given rate. Generally, this rate can be determined with respect to:

- the time taken for a system to accommodate a specified change with a defined unit of resource such as time for manufacturing system to be re-configured for a product change with a known and constrained amount of engineering resource which might be measured in terms of finance, available workforce and skill level, or
- The magnitude of resource required to accommodate a specified change within a given time frame [4].

![Figure 2-2: Change capability and change capability rate as two metrics to evaluate systems' agility](image-url)
Weston's agility definition [70] provides a basis for measuring and comparing the change capability and change capability rate of alternate systems and alternative methods of changing systems. Furthermore, Weston's notions of system change capability and change capability rate can be attributed to observed properties of i) system's building blocks and ii) structures and mechanisms used to compose (build) systems from building blocks, and iii) change processes and methodologies which can be deployed to change the system composition and behaviour. Thus, Weston's research [70] provides a conceptual basis to characterise system agility in a way that is independent of the specific nature of a system. More importantly, this definition of agility introduces a link between the ability of a system to accommodate change, and the intrinsic nature (i.e. architectural characteristics) of that system.

2-3 Manufacturing Systems Flexibility

Manufacturing context is characterised by frequent and unpredictable changes, which at production level will typically translate into change in production requirements. This may arise from variation in product type and families, product mix and production volume [24] [27]. To stay competitive, it is essential for manufacturing organisations to be able to adapt their production capability to changing production requirements constantly and effectively [71] [73]. New approaches to machine design and new technologies can provide manufacturing organisations with the change capability required to respond effectively to those continuous changes.

2-3.1 Review of manufacturing systems (MS) flexibility

Flexibility is a term commonly used in the literature to characterise the capabilities of MSs to cope with fluctuations in production requirements. Abdel-Malek et al. [63] and Stockton et al. [64] describe the flexibility of manufacturing facilities and products in a similar way. They both offer definitions of the term flexibility at an operational level and at a production management level, which are aligned to the classification of MS flexibility proposed by Sethi et al. [74] and summarised in Table 2-3.
According to Sethi et al. [74], flexibility mostly refers to two main capabilities of MS; i) the *range* of manufacturing operations they can support and ii) the *engineering effort* required in order to adapt the system to changing production requirements. Similarly, Slack et al. [65] define the *range flexibility* as the total envelop of operations that a system can achieve (referred to as short term flexibility), and the *response flexibility* which is defined as the ease (in terms of time and cost involved) with which change can be made within the capability envelop (long term flexibility). A parallel could be made with Weston's concepts of *change capability* and *change capability rate* [70] reviewed in more details in the previous paragraph. These classifications of flexibility emphasises the notion of *envelope or range* as a limit to production system flexibility. In addition, the concept of *response flexibility* or *change capability rate* emphasises on the effectiveness (regarding time and cost in this case) with which a production system can be "changed" in order to adapt its capabilities to new requirements.

More practical research projects have focused on the means to increase production *range* and *change capabilities*. For instance, Heilala et al. [71] research focused attention on material handling and on various shop floor operational workstation layouts e.g. series or "complete build" (parallel) task management and assess the flexibility potential of each of these in various contexts characterised by for example high product variants fluctuations, or mass...
customisation, machine breaks, rush orders. Stockton and al. [64] tested an approach to flexibility assessment by scrutinizing a MS composed of various elements (e.g. CNC milling machines, automated guided vehicle, robot, and vision inspection station). The assessment of various types of flexibility is made by evaluating the capability of the system to handle various part sizes, to manufacture various type of shapes, various type of raw materials and by testing the possible material routing paths. Abdel-Malek [63] proposed a methodology which focused on the need for flexibility types (product, machine, volume, process and routing), related changes the system is subject to, and the available resources (e.g. workers skills, investment capital), to provide guidelines that may be used to define a suitable level of flexibility for a particular system and context. A similar approach is adopted by Newman et al [68]. The case analysis focuses on an experimental work cell composed of robotic stations, material handling and part feeders, and machine vision systems. Interestingly, the authors differentiate between system flexibility and its agility and introduce as a boundary between both the concepts of system reusability and re-configurability.

2-3.2 Production systems classification

Figure 2-3 illustrates common relationships observed by Rampersad [75] between various types of MS and various production requirements High production volume requirements and low product variety are well supported by Dedicated Manufacturing Lines (DML). DML, or transfer lines are made of a collection of relatively simple machine tools using fixed automation [73]. DML typically result in a line of sequentially linked machines, which achieve repeatedly the same operations. DML provide a very high level of effectiveness. Material and part transfer can be highly automated and optimised since routing between stations is fixed. At the other end of the spectrum shown in Figure 2-3, flexible MSs (FMS) are designed to cope with changes in product variety or production volume.

![Figure 2-3: Productions system classification. Adapted from Rampersad (1994) [75]](image-url)
In the context of this research, two main strategies for creating flexible production systems were considered further: Cell-based Flexible MS (FMS) and Re-configurable MS (RMS). RMSs offer a relatively new approach to creating flexible MS [37] [73] [11]-[83], and are claimed to provide i) a higher ratio of productivity to investment than FMS, and ii) improved responsiveness relative to DML production requirement change is needed [73]. RMS were conceived to provide manufacturing facilities with i) flexible production capacity, ii) a suitable range (or envelop) of production functionality corresponding to a needed range of operation and iii) technological openness as the capability to integrate new machine (hardware of software) technologies as they appear [73] [11].

<table>
<thead>
<tr>
<th>System (machining/Manufacturing)</th>
<th>Definition/Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine System</td>
<td>One or more metal removal machine tools and tooling, and auxiliary equipment (e.g. material handling, control, communication), that operate in coordinated manner to produce parts at the required volumes and quality</td>
</tr>
<tr>
<td>Dedicated Machine Systems/Lines (DMS or DML)</td>
<td>A machining system designed for production of specific parts and which uses transfer line technology with fixed tooling and automation. The economic objective of DM is to cost-effectively produce specific part type at the high volumes and the required quality.</td>
</tr>
<tr>
<td>Flexible Manufacturing Systems (FMS)</td>
<td>A machining system configuration with fixed hardware and fixed but programmable software to handle changes in work orders, production schedules, part programs, and tooling for several types of parts. The economic objective of a FMS is to make possible the cost effective manufacture of several types of parts that can change overtime with shortened changeover time, on the same system. Note: A part family is defined as one or more part type with similar dimension, geometric features and tolerances such that they can be produced on the same or similar equipments.</td>
</tr>
<tr>
<td>Reconfigurable Manufacturing system (RMS)</td>
<td>A machining system which can be created by incorporating basic process modules (both hardware and software) that can be rearranged or replaced quickly and reliably. Reconfiguration will allow adding, removing, or modifying specific process capabilities, controls, software, or machine structure to adjust production capacity in response to changing market demand or technologies. This type of system will provide customised flexibility for particular part family and will be open-ended so that it can be improved, upgraded, and reconfigured, rather than replaced. The objective of RMS is to provide the functionality and capacity that is needed, when it is needed. Thus a given RMS configuration can be dedicated or flexible, or in between, and can change as needed. An RMS goes beyond the economic objectives of FMS by permitting i) reduction of lead time for launching new systems, and reconfiguring existing systems, and ii) the rapid manufacturing modification and quick integration of new technologies and or new functions in existing systems.</td>
</tr>
</tbody>
</table>

Table 2-4: Summary table of manufacturing systems comparison. Source M. G. Mehrabi [82]

To differentiate more explicitly between the FMS and RMS forms of flexible production systems, the classification of the system change types developed by Harrison et al [6] can be...
Harrison et al. [6] refined Weston’s [4] concepts of system change capability and change capability rate by characterising three classes of change capabilities:

- **Programmability** is the ability to readily program system behaviour and/or composition so that a system can reach a range of well-known states, thereby providing a means of coping with change of a predictable nature.
- **Reactivity** is the ability to react to change of an unpredictable nature by readily modifying system behaviour or composition.
- **Proactivity** is the ability to predict and anticipate change requirements in uncertain environments, and to prepare system changes (behaviour or composition) accordingly.

The means by which flexibility is achieved by FMS and RMS have marked differences. Adopting one or the other of these production strategies has major implications on the way in which the resultant MS can handle change. As shown in Table 2-4, FMS realise flexibility via the use of “fixed hardware and fixed but programmable software”. Conversely, RMS flexibility is achieved by “rearranging, and replacing basic process modules” i.e. by recomposing and restructuring the functional elements (both hardware and software) of configured systems to reach a configuration that meet specific requirements which may not have been predicted prior to the system conception. These observed characteristic differences in RMS and FMS approaches to production system flexibility are highly relevant in the context of this research. These differences impact significantly on the design and change processes of such systems, and therefore on:

- The modelling process required to implement 3D computer based virtual prototypes
- The functionalities (modelling and engineering) that a Virtual Prototyping Environment tools should provide
- The design and implementation approaches adopted to realise such tools.

### 2-4 Enterprise Integration

#### 2-4.1 Integration concept

Typical Virtual Organisations and Virtual Enterprises (VO and VE) are characterised by geographically distributed industrial partners and by heterogeneous engineering domains, processes, and infrastructures [28] [36]. Despite potential advantages in terms of functional effectiveness and partnership flexibility that this type of enterprise architecture provides,
partners have to collaborate on complex projects and in doing so need to deploy distributed engineering tools, knowledge management systems, heterogeneous IT/IS infrastructure. In this context, the process of achieving integration is generally presented in terms of realising a linkage of functional elements in order to obtain a system that exhibits specific functionalities [43]. Kosanke [1] introduces the notions of communication between system's elements, and defines integration as improving overall system efficiency by linking its elements by means of communication networks so that a high level of responsiveness and effectiveness can be obtained.

2-4.2 Integration, Information Systems and Technologies

Communication has been described as a fundamental aspect needed to achieve integration between functional entities within organisations [32] [34] targeted at developing agile MS [23] [3] [26] [29]. The concept of Computer Integrated Manufacturing paradigm (CIM) emphasises on the dependencies that exist between computer and network technologies and the need to achieve interworking of enterprise resource systems (and therefore the integrated interoperation of computers, machine, and people) in manufacturing environments. Information Technologies and Systems (IT/IS) are used to support communication within organisations.

As shown in Figure 2-4, the various levels of CIM integration defined by Vernadat [38] can be associated with various layers of the manufacturing organisation IT/IS infrastructure. A basic level of integration can be achieved by deploying a basic IT layer that enable general communication between partners (network communication protocols, Electronic Data Interchange mechanisms, electronic mail and general multimedia communication). The application and application integration layer focuses on the integration of Information Systems. IS integration is aimed at enhancing communication capabilities at an engineering level. This typically concerns the deployment of activity-specific software tools (e.g. central databases, and database management services, deployment of common software solution) which allow task specific software to be integrated. Finally, at the business integration level, industrial partners' activity might be tightly co-ordinated through the use of highly integrated environments allowing to share common business process models and decision support mechanisms (e.g. knowledge management tools, process modelling and simulation tools, data warehousing, case-based reasoning systems, manufacturing and resource planning (ERP / MRP software) or total shop floor management solution) [32].
Complex integration infrastructure and services can be deployed between partners in order to achieve integration at all IT/IS levels. However, the use of highly integrated IT/IS infrastructures has several implications in terms of required internal infrastructure and business process re-organisation, high deployment and maintenance cost and possibly constraints on future strategic decisions regarding collaboration with other partners [44]. Such solutions are usually the result of a common strategy adopted by companies that wish to merge into a long term and possibly permanent partnership.

Because of the financial, organisational and logistical constraints imposed by the deployment of such integration infrastructures, it is difficult for a given enterprise to get involved in more than one VE partnership. It follows that “ad-hoc integration” is often observed [6]. This type of integration results from long-term relationships during which partners can develop collaboration channels and processes. Ad-hoc integration could be qualified as adaptive [67] or emergent [62] since it is not the result of a particular collaboration model neither is it guided by specified control mechanisms, but simply emerges from the need maintain a level of collaboration that allow the VO goals to be reached. Ad hoc integration can prove to be very fragile, inefficient and resistant to change, because of its informal and reactive (rather than predictive) nature [5] [62]. As an alternative to focusing integration concerns and solution around IT/IS infrastructure integration, the design of a partnership organisation can centre on the capability of people to “self organise” in order to reach the best functional
performance. This "organic" approach to integration can be quite effective in the case of small organisations [5]. However as products and production systems become more and more complex, the understanding and organic management of complex partnerships prove to be beyond the capabilities of a small group of people [61].

2-4.3 Integration and Virtual Enterprises

For a number of decades, the Computer Integrated Manufacturing literature has been concerned with intra enterprise integration [38]. The specificity and cost of the computer communication and information infrastructure required to achieve integration was previously only manageable at the scale of isolated organisations. Research interest in intra enterprise integrations has been fuelled by the increasing performance and affordability of local and wide area network LAN/WAN. The standardisation of network communication protocols (i.e. TCP/IP) and the expansion of the World Wide Web infrastructure has provided an ideal basis to achieve effective and flexible (i.e. agile integration) integration of distributed partners at a global scale [31] [33].

Figure 2-5 is adapted from the research of Upton et al. [36] on virtual manufacturing and provides a simplified representation of various levels of integration, ranging from simple data transmission, data access to application sharing. This figure adds two dimensions to the information provided in Figure 2-4; the level of collaboration (metaphorically designated as married, engaged and dating), and the lowest common level of IT/IS expertise and infrastructure shared by VE partners. Public network technologies (i.e. World Wide Web, internet) have become a fundamental layer of manufacturing organisations' IT/IS integration infrastructure. Internet technologies are accessible to all modern enterprises and are readily usable at early stages of any partnership formation without requiring heavy software
infrastructure deployment. Therefore, Web compliant technologies provide an affordable basis to achieve effective and flexible integration of VE and VO partners [31] [33] [34]. In addition, the ongoing growth in demand for web-based technologies performances is being satisfied by constantly improved Web-compliant standards (i.e. data formats, software development technologies and languages), thus suggesting that there is potential for higher levels of collaboration (cf. vertical axis in Figure 2-5) to be achieved.

2-5 Use of three dimensional (3D) modelling technologies in the domain of manufacturing

This part of the literature review provides an overview of Virtual Reality (VR) technologies, and highlights important aspects of three-dimensional (3D) modelling in the domain of system engineering. Focus is on the 3D computer-based modelling and simulation of production systems, and on the design and realisation of so-called virtual MS prototypes and virtual prototyping software that allows such model to be realised and exploited. There is a considerable body of academic research focusing on the use of some form of 3D modelling to support the design of manufacturing systems. Much of that literature is specifically aimed at describing 3D computer models and modelling environments realisation and use. It has been necessary to constrain this review to research projects that were considered most relevant to the work described in this thesis.

2-5.1 Virtual Reality

VR is typically associated with applications that make use of 3D modelling as visual front end. Burdea et al. [136] define VR as a high-end human-computer interface allowing user interaction with simulated environments in real time and through multiple sensorial channels. VR, as defined by the Academic Press Dictionary of Science and Technology [143], is a computer simulation of a system, either real or metaphorical, that allows a user to perform operations on a simulated system, and that shows the effects of those operations in real time. The 3D-based simulation and modelling community commonly define VR as an artificial environment created with computer hardware and software, and presented to the user in such a way that it appears and feels like a real environment. VR is also described as computer-based simulation that uses 3D graphics and specific devices (e.g. data glove) to allow the user to interact with the simulation [136].

Despite the fact that immersive VE (i.e. providing a relatively high level of immersion) are commonly used by car manufacturers such as Daimler Benz, Ford, Volkswagen [139], a
survey conducted by Wilson et al. [142] has shown that the majority of industrial companies showed a clear preference for simpler VEs. Those are commonly referred to as desktop-based VE [11], which seems to be more adapted to the requirements imposed by time and cost critical engineering applications. For the purpose of machine virtual prototyping, the use of 3D models consists mainly in replacing a physical system with computer models (i.e. virtual prototypes) that simulate a system or product geometry and functionality [137]. This introduces the notion of virtual prototypes (VP) and virtual prototyping environments (VPE), which make use of 3D-based simulation to support the design lifecycle of mechanical systems or products.

As shown in Figure 2-6, Burdea et al. [136] describe a VR environment as a set of software and hardware components the most relevant of which is the VR engine often realised by a graphic workstation. The interactions between the user and the VR engine are made possible via Input/Output (I/O) devices. The virtual world is modelled using various software functions and modelling data, so that the user input can be interpreted, and user feedback generated.

![Figure 2-6: Virtual Reality (VR) system architecture. Source Burdea et al. [11].](image)

It has been suggested that virtual environments can be classified based on the degree of immersion they provide (i.e. the capability of a user to dissociate virtual reality from reality itself [140]). Ellis' [141] notion of VR is to maintain at least one sensory modality. Typically, VR is often associated with 3D computer graphical modelling since the visual channel is the most intuitive user interface and the easiest to implement using common computer hardware and software [115]. However, the degree of immersion can be augmented thought the use of specific devices such as 3D mouse (trackers), stereo Head Mounted Display (HMD), haptic interfaces and force feedback devices such as sensing gloves [11], audio/video space and
auditory system e.g. CAVE environment [138], and user position orientation tracking systems. However, it has been highlighted that such immersive environment are based on highly specific hardware and software components, which implies that their deployment and use is costly and required skilled resources [14].

2-5.2 Virtual prototyping

Kim et al. [92] defines Virtual Prototypes (VP) as an electronic definition of product assembly or component. These authors define virtual prototyping with 3D solid modelling as the "only unambiguous way to represent mechanical parts and assemblies". Kim et al. [92] also emphasise on the fact that virtual prototyping consists in modelling both physical and functional properties of a system. These properties are referred to by Xu et al. [7] as system semantic that allow a model to perform functionally as its physical counterpart. De Sa [139] et al. define virtual prototyping as the application of VR for prototyping physical mock-ups (PMUs) using product or process data. Similarly, Ressler et al. [111] use the term "physically based modelling" to designate the activity that consists of simulating all characteristics of a system, relevant to a particular context, as precisely and realistically as possible.

Haug et al. [137] state that the VR prototyping technology combines the VR approach with advanced modelling, simulation, and user interfacing techniques. A VP simulates product features such as visual appearance, functionalities, and user interfaces as closely as possible. Similarly to de Sa [139], Wang [144] proposes to use the concept of classic prototypes, defined as an early or original full scale model of a structure or piece of equipment used in evaluating form design, fit, and performances [143], which is refers to as "mock-ups". Wang's [144] definition is an attempt to generalise the concept of VP, and is given as follow: "Virtual prototype, or digital mock-up, is a computer simulation of a physical product that can be presented, analyzed, and tested from concerned product life-cycle aspects such as design/engineering, manufacturing, service, as if on a real physical model. The construction and testing of a virtual prototype is called virtual prototyping (VP)". In addition, of the description of a VR systems' architecture shown in Figure 2-6, Wang [144] provides a general description of the "components" that compose VP (cf. Figure 2-7). It should be noted that Wang does not differentiate VP (as 3D models) from the software environment used to implement and exploit virtual prototypes.
However, the component represented in Figure 2-7 place emphasis on the functions of the software environment which allows VP to be implemented and analysed rather than on the VP digital model itself. The lack of distinction between digital model and modelling environment is common. One reason for this is that, as mentioned by Burdea [11], due to the complexity of the systems being modelled, the use and simulation of virtual models and prototypes is tightly coupled to a specific software environment (which is often the same environment used to edit those models). This highlights the fact that the visualisation and simulation of complex VP typically depend on complex software environments providing the model simulation functions (e.g. real time control, user interfaces).

It is interesting to note that Wang et al. [144] present 3D computer models (CAD models in Figure 2-7) as the most "widely accepted" means to represent a system and to achieve the minimum level of model user interaction (or the minimal level of immersion for virtual prototyping environment). The "user interfaces" defined by Wang et al. (Figure 2-7), are defined as "the integration components that co-ordinates the behaviour of models and provide useful information to the system users". The user interfaces link the model to the "perspective test models" which are the (software) functions allowing digital models to be analysed according to the users’ and engineering requirement. The concept of "perspective test models" defined by Wang [144] implicitly leads to dissociate Virtual Prototyping from Virtual Reality [145], and links VP to the concept of Virtual Manufacturing (VM) i.e. the use and integration of 3D models and VPs within an engineering context.

### 2-5.3 Virtual Manufacturing

The concept of virtual manufacturing (VM) places emphasis on the potential of computer models and computer simulation to represent digital environments in which manufacturing and engineering related activities are simulated. Gunasekaran et al. [26] generally define VM as "an integrated synthetic manufacturing environment used to enhance all levels of decision
and control in a manufacturing enterprise”. Qiu [99] is more specific about the meaning of the term “virtual”, and states that VM employs computer simulation to model products and their fabrication processes, and aims at improving the decision-making process along the entire production cycle. As shown in Figure 2-8, the concept of VM places emphasis on providing an integrated computer-based environment that allows all aspects of the manufacturing activity, from product design to the design of manufacturing systems (MS) (i.e. the complete manufacturing lifecycle) to be supported [73] [14] [91]. Figure 2-8 illustrates the fact that computer-based 3D models can be used to model part of a product or a MS, or to model complete production plant and plant layout.

It should be noted that despite the fact that some research projects have focused on human presence and tele-immersion in a VM environments using VR devices (i.e. virtual theatres screens/environments (e.g. CAVE [138]), Head Mounted Goggles (HMD) goggles, haptic gloves) [108] [14] [136], the degree of immersion should be considered as a functionality (e.g. third axis in Figure 2-8) which is not necessarily required in an engineering context. This marks a difference between “human centric” virtual environments aimed at providing a high level or immersion (i.e. human user interfacing), and “context related” virtual environments which purpose is to provide support for the design of manufacturing system (i.e. requires interfacing and integration with system engineering tools). This research is focused on the modelling and virtual prototyping of MS and on how 3D based computer models can be used to support different phases of the lifecycle of such systems. Particular emphasis has been placed on the use of VP to enable the testing of various MS configurations in a virtual form.
2-5.3.1 Virtual Prototype implementation

The difficulties of implementing a 3D model of complex and large-scale MS, has been highlighted in the literature [14] [8] [90]. It is essential to minimise the amount of resources required to achieve the purely modelling tasks, in order to maximise the value of 3D model as engineering tool [110]. In order to simplify the modelling task several approaches to the implementation of 3D model of manufacturing systems have been suggested. Xu et al. [7] has catalogued two main approaches to the implementation of what he refers to as "virtual environments" (in fact VP). The first modelling class is described as "bottom up generative" construction which consists of sequentially implementing the various aspects of a system's model by conducting the graphical (3D) modelling, mapping the model behavioural functions and implementing the model / user interfaces. This process is sequential, tends to be time consuming and requires a good knowledge of modelling tools and processes. In addition, VP resulting from this type of implementation, although well structured, exhibit very limited reusability [7] and are often used as what Mersinger [14] defines as "exhibition models", generally implemented for specific purposes (i.e. single use, visual display purpose).

The second modelling class mentioned by Xu et al. [7] is described as a "building-block" approach that introduces the notion of "library" and "pre-built objects" from which larger models can be composed. It is stated that this approach potentially allows the modelling time to be reduced. In addition, the modelling skills required to implement a model can be reduced since the modelling of a system practically consists of composing rather than editing modelling objects [110]. Finally, the software environment supporting the modelling functions can be simplified since the modelling components' editing and composition functions are dissociated. This approach also has the advantages of providing a certain level of model re-configurability and re-usability since modelling components can be re-composed or re-used depending on the modelling requirements [100] [104]. Adolfsson et al. [8] highlight the importance of adopting a component-based approach to the implementation of VP what is referred to as "customised modular automation equipment". In the same way, Min et al. [73] has highlighted the importance of developing VPE adapted to the prototyping of so called "re-configurable" manufacturing systems (RMS), as opposed to what Zhao [109] refers to as "well established" types of system (typical examples of which are CNC machines and industrial robots). The importance of ensuring consistency between real and virtual systems is highlighted later in Chapter III and a more extensive review of the implementation of the concept of modelling components to the domain of 3D modelling is provided in Chapter IV.
2-5.3.2 Virtual Prototyping Environment (VPE) design and realisation

VPE are the software tools that provide the functionalities required to edit, view and simulate VP. VPE should implement modelling related and engineering related functionalities. The design and implementation of VPE therefore requires a good understanding of the specific characteristics of the system being modelled [14] [137] and of the aspects of the system should be modelled [7] [137] so that VPE that can effectively support system engineers in accomplishing their tasks [109]. Xu et al. [7] refer to a "task oriented approach which consist of designing and classifying VPE according to the type of systems they are adapted to. The task oriented approach requires a "domain analysis phase" that consists of defining i) the type of MS being modelled, ii) the aspects of the system and level of detail at which the system is modelled, and iii) the understanding the context in which models are used and the purpose they are intended for [7]. Zhao [109] similarly refers to the functions specifically designed to support the modelling of particular types of MS used in the industry as “application specific modelling environments”. In the same way, the concept of “perspective models” defined by Wang [144] focuses on the model analysis functions that a VPE should provide.

2-5.3.3 Virtual Prototyping Environment structure

VPE are typically designed as an integration platform for various software tools used during the design of real MS. For instance, Fujii et al. [28] state that “Virtual Manufacturing” is a means by which “islands of simulation” can be integrated in a single model to improve the design and management of factories. In their research, Min et al. [73] adopt a similar approach to the implementation and definition of a virtual manufacturing environment built on 3D modelling technologies and focus on the integration of real time computer-based simulation tools used for the design and utilisation of real MS (e.g. continuous NC simulation, discrete logic). In the same way the research conducted by Adolfsson et al. [8] is representative of the approach commonly adopted by the software industry in the development of so called “digital manufacturing environments” produced and commercialised by Tecnomatix and Dassault [118] [119]. Such software tools achieve integration of a CAD modeller with machine control editing, and other manufacturing system design tools in order to create an complete engineering environments that can be use to support the whole MS lifecycle from design to maintenance and monitoring.
Figure 2-9 is a representation of a general architecture that characterises VPE for MS prototyping. Typically, an interface layer (layer 1) provides interaction mechanisms, which essentially operate between models and users of models. This can be realised by simple screen display and user feedback through windows based Graphical User Interfaces (GUI) (or via more advanced peripheral devices for immersive environments). The interface layer also implements interfaces to the VPE context (layer 4) which can be realised by other engineering software environments such as a CAD modeller and machine control-editing environment. In such a scheme, layers 2 and 3 implement the knowledge handling and data management functions that represent the core of VPE [73] and provide the functions required to exploit virtual models for engineering purposes.

2.5.3.4 Virtual Environment utilisation

Unlike that for commercial VPE developers, there has been a significant effort among the research community to implement software that can be used in engineering environments where a wide variety of computing platforms and engineering tools are deployed [92]. Internet compliant technologies and standard formats are commons means by which such environments are realised [73] [92]. However, as mentioned by Cheng [10] such distributed and heterogeneous environments, can be an obstacle to the development of (computationally and) graphically intensive applications. Thus up to date programming methods used to implement distributed web-based virtual environments have failed to hide the software and hardware differences of distributed partners’ infrastructures [10] (i.e. Information Technology and Systems IT/IS).

For instance, Kim et al. [92] have developed a so-called 3D-Syn collaborative system that supports synchronous communication and manipulation of 3D part models on the World Wide Web (WWW). Their work investigates the effective use of Computer Supported
Collaborative Work (CSCW) in order to implement effective collaboration tools. As stressed by Kim et al. [92], in an environment where there may be a wide variety of computer platforms and engineering tools, the choice of modelling formats and programming language is crucial in providing collaboration capabilities among geographically dispersed work groups. The authors emphasises on the importance of using open, platform independent, and web compliant technologies in order to implement 3D modelling collaborative tools. In particular, vast potential is highlighted with respect to the potential use of the Virtual Reality Modelling Language (VRML) and platform independent technologies such as Java and HTML, which are now ubiquitous across almost all computing platforms to provide support to implement "virtual worlds networked via the World Wide Web (WWW)".

Kim et al. [92] research has highlighted the limitations of software built upon proprietary 3D modelling formats, which deployment requires high-end hardware (SGI station) and groupware software platforms (e.g. Joint Editing Service Platform). In the same way, the research of Suh et al. [104] has focused on the implementation of internet-based virtual machine tools (i.e. CNC machine). This research has highlighted that although commercial 3D based machine simulation environments (e.g. Delmia, VNC (Virtual NC), WorkSpace4, Flow Software, Tecnomatix RobCad) "offered networked version of their models, these are off-line software systems that run in a stand-alone fashion, not on the Internet". Suh et al. [104] use the term "stand-alone" to highlight the fact that model implemented using those application are not usable outside the (modelling and engineering) software environment used to edit them.

The use of 3D standard modelling technologies, (i.e. Virtual Reality Modelling Language (VRML)), is essential in implementing virtual environments which can effectively be used in distributed engineering environments [117] characterised by heterogeneous software and hardware infrastructure [92]. However, the export of engineering data required to implement MS virtual prototypes, outside the core-engineering environment, requires the translation of original system data such as CAD model, and logic control data into other formats. The issues involved with the format translation phase of the modelling process have been highlighted in many research projects. The projects conducted by Ressler et al. [111] as part of the SIMA (System Integration for Manufacturing Automation) program in the NIST (National Institute of Standard and Technology), has focused implementing translators that allow models produced using proprietary software such as Deneb's I Grip model, to be transformed into

1 Deneb Robotic inc., a leading provider of digital manufacturing solution, was acquired by Dassault Systems, the company developing the CATIA CAD solution, in year 1997.
portable VRML models. Despite an acceptable level of automation for this kind of task, manual intervention at the code level was still required, which required knowledge of the modelling language used (VRML in this case). Min et al. [73] highlights the fact that the integration of CAD data to implement the 3D geometry of VRML based virtual prototypes requires a tedious modelling editing process that consists of structuring, referencing and interfacing modelling data. Interestingly, Adolfsson et al. [8] mentions that, although data translation of machine control and CAD modelling data can be handled relatively easily for simple systems there remains the problem of complexity of such tasks, which increases exponentially with the complexity of the system being modelled.

### 2-6 Chapter Overview

In this chapter, an attempt has been made to review the essential concepts and paradigm that have been used as a conceptual basis to develop the research described in the following chapters. General manufacturing paradigms such as agility, have lead to analyse the context in which manufacturing organisations and enterprises operates. The agility concept have been narrowed down to a set of organisational practices (i.e. VO / VE), collaboration tools and services (i.e. IT/IS) and engineering tools (i.e. flexible MS technologies and design, VP and VPE) that are used to enhance the response of manufacturing related organisation to changing requirement, and to support more effectively the engineering lifecycle of MS in such context.

A particular emphasis has been placed on Virtual Prototypes (VP) and Virtual Prototyping Environment (VPE) requirements specifications and design. General approaches to the design of VP / VPE have been reviewed and a broad set of potential guidelines for the design and realisation of such tools have been browsed. In particular, different types of MS have been defined, and the consistency between real and virtual system have been highlighted as a fundamental requirement for the design of functionally effective VPE. In addition, the importance of realising VPE tools with respect to the needs of VE / VO in terms of (distributed) collaboration has been linked to suggestions on technological choices for the implementation of VPE tools.
Chapter 3  A conceptual approach to Virtual Prototyping Environments (VPE) requirements specification

3-1 Production system engineering lifecycle and context: Research case study

A list of conceptual requirements specifications that can be used as a basis to initiate the design and implementation of an innovative Virtual Prototyping software Environment (VPE) is detailed in this chapter. Two aspects of VPE are considered which are i) the potential of VPE to serve engineering purposes, and therefore to support engineers during the manufacturing systems (MS) design and change and ii) the potential of VPE to enhance communication and engineering collaboration between potentially distributed partners involved in the design of MS. This research was undertaken as part of the COMPAG / COMPANION (COMponent-based Paradigm for AGile automation, and COmmon Model for PArtNers in automatION) projects funded by EPSRC (grants GR/M43586 and GR/M53042) and conducted at Loughborough University UK. In the context of the COMPAG / COMPANION research, the MS design and build process used currently by a first tier machine manufacturer (Cross Huller), has been studied and was used as basis on which the conceptual specification of a VPE software tool might be grounded.

3-1.1 Cross Hüller machine manufacture process description

The engineering lifecycle of large and complex MS is typically conducted through several phases which are generally described within manufacturing companies as “requirements specification”, “concept and detail design”, “implementation, utilisation / maintenance / monitoring”, and finally “recycling or re-configuration/reuse” [43] [6]. The process is based on the working practices undertaken by Cross Hüller to design and build MS for the new I4/I5 Ford Motor Company engine programme. This process is presented from the machine manufacturer perspective and was decomposed into Business Processes (BP) and Enterprise Activities (EA) [38] [39]; a notation based upon use of CIMOSA terminology [41] [42] which have been extensively used to model processes of various industries in Loughborough. The Cross Hüller process decomposition was carried out jointly by the author and his colleagues as part of COMPAG/COMPANION project.
3-1.1.1 Customer requirements analysis

Customer requirements are determined within a Pre-Sales engineering Business Process (BP) where Cross Hüller (machine manufacturer) and Ford Motor Company (customer) collaborate on a Simultaneous Engineering (SE) activity. The Pre-Sales (cf. BP 211 Figure 3-1) consists of successive meetings involving group of representatives (mainly engineers) from all partners involved in the design and engineering of production lines. During these meetings, technical solutions are discussed in an attempt to foresee early design problems and solutions. This phase typically comprises a consultation where engineers product and production system engineers, seek to define a machine design solution that responds to Ford’s requirements in terms of for instance, production cycle time estimates, number of operations. As an example, from machine system customer’s (Ford) perspective, the ideal solution consists of a production line that achieves a maximum level of simultaneity between the machining operations of engine blocks so that the production cycle time can be reduced to a minimum. Conversely, the machine manufacturers and sub contractors’ role during the Pre-Sales phase is to inform the customer of the potential cost or reliability factors associated with certain design solutions, and to suggest alternatives design choices until a satisfactory solution is found.

The Pre-Sales phase is concluded by the acceptance of a contract (i.e. Sales Order) by the customer, which initiates an instance of the next Cross Hüller’s business process initial design, or concept design phase. At this stage of the collaboration, the customer provides draft drawings (mostly paper based or 2D CAD files) of the engine parts and a proposal for basic production line layout. These drawings and proposal are analysed and detailed until Cross Hüller and Ford agree on a final production line layout design, including all of the station

Figure 3-1: Cross Hüller / Ford Motor Companies Pre sale contract phase
mechanical layout, tooling and control sequences. Communication between the two partners consists exclusively of proposals and paper based documents (draft product part drawings from Ford and initial production line layout from Cross Hüller, Microsoft Excel documents) exchanged and analysed during meetings of senior engineers from both parties who have detailed knowledge of their own domain of activity and partial knowledge of each other's.

3-1.1.2 Design process

A detailed study of Cross Hüller's business processes revealed a structured but rather fragmented and sequential progression which starts with a conceptual design phase (BP-222, cf. Figure 3-2). At this stage, the customer's requirements are considered and senior designers undertake the station layout design. The conceptual design output takes the form of paper-based documents that describe the overall production line stations layout (EA2211 Initial design layout), the detailed unit cycle time and the tooling layout.

Figure 3-2: Cross Hüller Machine concept and detailed design phases

The senior designers' Initial Machine Layout (EA221) is decomposed into station layout processes (BP 222-1) during which a more detailed description of the group of machine /
assembly tasks allocated to each station is generated by project engineers, so that initial unit production cycle times (defined during the pre sale engineering phase) can be finalised.

When the first station layout has been completed, an advanced planning process (BP 222-2) starts which aims at estimating the total project's budget and time (for machine delivery), as well as generating a list of potential sub-contractors and standard machine parts requirements. The concept design is concluded via a design review process (BP 223) which regroups MS engineers and customers in order to validate the initial layout of the machine stations. At this stage, both partners agree on the estimated project time scales and costs. It is also at this time that any customer's requirements change is taken in account and re-translated into initial machine data until a final agreement, which marks the beginning of the detail design phase (BP224), is found.

The detailed design process consists of two concurrent sub processes, which generate MS function diagrams (BP 224-1), and the complete set of final assembly drawings (EA 2241). The MS function diagrams are a detailed description of sequences and timings of machine actuators and sensors that realise engine machining and assembly operations. The final unit cycle time has to match the one estimated during the initial station layout and validated by the customer during design review. Simultaneously, detailed drawings of the layout of the MS actuators are produced as part of the final machine documentation. Once the complete machine layout and behaviour have been described, the hard engineering phase (BP 224-2) starts. This consists of the engineering of the control hydraulic and electrical parts of the MS.

3-1.1.3 Pre-Commissioning phase

The pre-commissioning business process (BP227) described in Figure 3-3, is an important phase of the Cross Hüller machine design and build process. The concurrent design of both MS mechanical hardware and logic control software inevitably results in certain design inconsistencies, which, if not highlighted and resolved before the machine physical implementation is carried out, will most likely require later hardware re-design, and therefore a significant increase in the overall project cost and time.
Most of the design errors exposed during this phase are related to inconsistencies between the mechanical design of the machine and the control logic design. No tools are presently used to enable the overall MS design to be tested. The pre commissioning phase currently lasts for around six weeks (out of a 52 weeks total). The use MS prototyping and simulation tools could dramatically reduce the overall design process duration, by enabling engineers to detect errors earlier in the design lifecycle, and therefore to avoid costly and time consuming re-design loops.

3-1.2 General approach to VPE requirements specification

The analysis of Cross Hüller engineering process has lead to the definition of two main functional aspects of 3D Virtual machine Prototyping Environment (VPE) tools, which respond to the need of system engineers, namely:

- The potential of VPEs to serve as engineering tool, which places emphasis on the use of a VPE prototyping and simulation tool; Prototyping tools should enable system engineers to assess alternative design solutions in order reduce the time required to find the best or near optimal design solution. It is vital to make the modelling related tasks as transparent as possible to engineers who are non-modelling specialists, so that the benefits of virtual prototyping as an engineering activity can be maximised.

- The potential of VPEs to serve as a communication tool; VPEs should provides support for engineering collaboration between distributed partners who have designated roles and responsibilities in a Virtual Enterprise (VE). This aspect of a VPE places emphasis on the value of 3D models as intuitive and common representations of complex MS and the potential of using such models to serve communication purposes.
It should be noted that in this research a large emphasis is placed on the use of 3D-based virtual prototyping and simulation tools to support the design of MS. As highlighted later in this chapter the use of VPE tools can be extended to the production management, production planning and monitoring, if integrated within total shop floor and production management solution (Computer Integrated Manufacturing). However, because of the particular nature of the production systems considered in this research, it is believed that the value of 3D based prototyping and simulation tools is maximal when used to support the design phases.

3-2 Engineering-related functionality of Virtual Prototyping Environment (VPE)

3-2.1 Cross Hüller Manufacturing design process analysis

3-2.1.1 Design domains separation

The analysis of the Cross Hüller MS design and build process highlighted a strong separation between the system mechanical and control software design processes (in which the system customers are largely involved) which remain largely isolated from one another. Such a functional separation between these two domains of design often reflects a poor integration between engineering support software (sometime referred to as islands of automation [5] and simulation [89]). Control engineers and mechanical engineers make use of specialised tools that allow different aspects of the machine design of concern to them to be defined. However, currently, no common (or shared) environments provide support for monitoring the overall design process consistency. Ad hoc integration methods are deployed to co-ordinate the fragmented use of heterogeneous tools [62] [6] and there is no common representation or visualization of MS throughout the (design, analysis implement, test, maintain, reuse) life cycle, nor is there any overall computer-executable model capable of supporting "what if" analyse of resulting machine design and behaviour [6].
There is a crucial need for tools that enable engineers from different domains to communicate and interpret the overall system design in order to foresee design inconsistencies and errors earlier in the process, and therefore to avoid costly (in terms of time and engineering resources) re-design. As stated by Bullinger et al. [14], although design complexity is growing, the first sketches must be more detailed and the perception of the different basis characteristics should be guaranteed. There is a need to deploy prototyping and simulation tools that can help to decrease the time for data construction and data evaluation, particularly at early stage when core design decisions have a major impact regarding the future project cost and time scale.

3-2.1.2 Design phases separation

From the analysis of Cross Huller process, it is also apparent that the overall MS design is divided in two distinct phases (along the time axis), which are the “concept design” and “detail design”. During the concept design phase, the various system data are kept in formats that allow engineers to conduct design changes quickly and easily. Typically, the two main aspects of the machine considered at this stage (i.e. mechanical design and control data) are kept in paper based formats (e.g. “machine stations layout blue print” and “machine process timing diagram”) or in simple computer formats (e.g. Excel sheets or 2D CAD or Visio models of machine station layouts) [6]. This permits a progressive adding of details to physical and behavioural description of a machine by keeping data in a format that can readily be modified, an approach coined by Klein [86] as least commitment design.

![Figure 3-5: Design phase separation and data format and translation issues](image-url)
However, the data formats used during early design phases do not allow the integration of design information and data in a form that can enable engineers to assess accurately the design consistency with respect to initial requirements and the consistency between sub-designs produced within separate engineering domains. The experience of the engineers involved and the ad hoc communication channels between engineering domains are the only means by which effective process management can be achieved at this stage. This essentially relies on the capability of human resources to organise themselves to face design problems or changes [62].

Later in the “detailed design” phase, MS mechanical and control design data are described at higher levels of detail using specialised engineering tools (e.g. CAD, PLC editors). At this stage, design data are expressed in formats that are similar in nature to the real system configuration data (e.g. detailed CAD models, low level PLC programming code). The transition between data formats generally occurs when it has been agreed that the design has reach a state after which there will be no subsequent changes. The processes consisting of editing the data in a digital format is time consuming and requires skills and knowledge of the computer tools used for this purpose. If the system design has to be modified (i.e. need a new solution investigation / definition loop) engineers are more likely to go through a “paper, pencil and meeting” loop rather than working out a new solution directly using computer tools. The data dependencies and the format that they are expressed in implies that even minor changes result in time consuming and error prone data consistency checking re-editing tasks [110]. In addition, although the system design data are at this stage expressed in digital formats, the simulation capability might still be poor because of the low level of integration between the various computers tools used in various domain of engineering.

3-2.2 Potential Use of 3D Virtual Prototyping Environments (VPE) in support of design of manufacturing systems

Based on the abstractions of the Cross Hüller MS design and build processes previously described the potential use of a Virtual Prototyping Environment (VPE) to support machine engineering is conceptualised in Figure 3-6. The use can be viewed from two complementary perspectives, namely as providing vertical integration and horizontal integration capabilities. Vertical integration relates to the use of a 3D machine models to integrate activities carried out within various engineering domains. A common and intuitive 3D machine model can facilitate the co-ordination amongst various engineers and their activities. Vertical integration capability of a VPE and virtual prototype models therefore offers means of resolving integration issues associated with separated engineering domains and processes, which in the
case of Cross Hülle involves a marked separation between mechanical and control system design processes and tools.

Conversely, horizontal integration capabilities of the VPE tools can facilitate transitions between different phases through which a system design (data) is progressively and iteratively refined. The use of VPE tools can potentially provide manufacturing systems simulation capabilities in the early design phase and support the evaluation of rapidly changing design data.

### 3-2.3 Characterisation of manufacturing systems (MS) and definition of modelling requirements

The types of MS that generally compose production lines for the automotive industry can be described as being mechatronics systems. Mechatronics systems are materialised by a mechanical aspect that results from the structured composition of various mechanical parts. The kinematic layout determines the mechanical functionality of the system. The electronic and software part of the system provides the machine control capabilities used to define system behaviours (via control logic) and thereby the way the system will fulfil specific tasks. The study of Cross Hülle's MS engineering process has shown (cf. sub section 3-1.1) that the largest part of the design process was directly related to the description of those two aspects of production lines. Flexible machine technologies mainly focus on facilitating the design of mechanical hardware and control software [11] [82]. VPE modelling tools should therefore provide functions that allow both aspects to be transposed into an executable computer models to enable the consistency of both mechanical and control design to be assessed [109].

#### 3-2.3.1 Invariant design basis

In Chapter 2, key flexible machine technologies were categorised under two main groups, namely Flexible Manufacturing Systems (FMS) [76] [77] [78] and Re-configurable Manufacturing Systems (RMS) [37] [73] [11]. The conceptual approaches to realising system flexibility characterised by FMS and RMS system architectures were observed to be
distinctive and have different engineering lifecycle needs. Consequently, those two approaches to creating flexible systems have radically different implications with respect to functions that VPE tools should provide to support their engineering lifecycles.

In the context of this research, VPEs are perceived to be tools that enable engineers to exploit MS's invariant basis. FMS and RMS have been differentiated based on the characteristics of the re-usable and re-configurable constructs they provide. For instance, Adolfsson et al. [8] affirm that the adoption of modular design and standard system components can facilitate re-configurability, so that agility can be (partially) realised by re-using and re-configuring existing machine components [8]. Such statements link the architectural characteristics of MS and the particular characteristic of machine constructs (i.e. re-usability and re-configurability) to the type of process that characterise the engineering lifecycle of such systems. The concept of invariant basis defined in this research therefore refers to the part of machine design that is invariant throughout various design cases, and therefore, which provides design flexibility (i.e. which serves as a basis to initiate new design or to change on existing system configuration). In the following paragraphs, this concept is used alongside the concepts of change capability and change capability rate (Weston [70]) to differentiate two different approaches to flexible machines design.

3-2.3.1.1 FMS invariant design basis

The invariant basis that characterises FMS consists of a fixed hardware layout (i.e. mechanical) and fixed but programmable software (i.e. machine control), both integrated as highly re-programmable systems. FMSs are therefore characterised by a limited "change capability" (narrow triangle end in Figure 3-7) due to the permanent aspects of the machine hardware that only allows a finite range of operations and production requirement to be supported. Conversely FMS exhibit high "change capability rate" (large triangle basis) because of pre-defined system configuration processes, methods and tools, which allow such system to be re-configured with minimal engineering resources. As shown on the left part of Figure 3-7, FMS invariant basis (greyed area) is materialised by integrated hardware and software and pre-defined machine control configuration tools and processes (e.g. CAD/CAM software, off-line robot programming). This provides an ideal platform based on which product and MS engineers can collaborate, discuss requirements, foresee problems and examine possible solutions.
3-2.3.1.2 RMS invariant design basis

RMS are characterised re-usable and re-configurable software (machine control or other) and hardware components [6] designed independently from each other and defined at a lower level of granularity (i.e. possibly at individual actuator and sensor level). Typically, RMS invariant basis is materialised by independent mechanical, control and software components whose characteristics are defined according to the needs for re-use and re-configurability for each design aspect of RMS (e.g. MS mechanical modularity, software component functions re-usability). Customised RMSs, which answer specific requirements (e.g. product type, operations type, production volume) can therefore be built by configuring and composing various types of components providing needed functions and mechanical characteristics, according to a suitable machine configuration.

Despite the fact that RMS potentially provide a much larger change capability envelope than FMS (large triangle basis in Figure 3-7), achieving a high change capability rate for RMS re-design (i.e. RMS re-configuration) involves more complex and problematic issues than found with FMS. In industry, engineering environments associated with the design of RMSs are often more difficult to manage because the various design processes corresponding to each aspect of the system are separated by disciplinary and functional boundaries and supported by disparate engineering tools. Therefore, the RMS approach to manufacturing system design and change can potentially provide clear advantages in term of change capability so that
requirement can be matched more closely. However, there is a critical need for tools and methods that facilitate co-ordination across RMS engineering domains and processes in order to increase the change capability rate and therefore to maximise the potential of the RMS approach.

3-2.4 Virtual Prototyping Environment (VPE) design / implementation

3-2.4.1 3D modelling and virtual prototyping of mechatronic production systems

The implementation of a Virtual Prototype (VP) starts with a modelling task that consists of implementing a computer-based 3D model of the machine. This model can then be used to conduct simulation and analysis of the (initial) real system. The lifecycle of manufacturing systems' VPs is described based on the review of various approaches found in the literature [14] [7] [139]. Figure 3-8 represents this process as a succession of different phases.

![Figure 3-8: Generic description of production system virtual prototypes lifecycle phases.](image)

These phases are placed within the context of this research under the following sub headings:

3-2.4.1.1 Data set definition

Within this thesis, the primary purpose of virtual prototyping is to create a computer-based virtual model of a production system exhibiting 3D geometrical characteristics and dynamic behaviours of the real system. Many aspects of modelling tools, including the modelling process, the modelling functions, the integration between modelling and engineering software environments are causally dependant on the type of system that needs to be modelled and on the related purpose of the modelling exercise. The requirements definition of VPE modelling functions is essential and consists of i) defining a set of relevant aspects of the real system to be modelled and ii) defining a level of detail for which those aspects of the system need to be modelled.
3-2.4.1.2 Data gathering and Data translation

The second lifecycle phase consists of gathering the descriptive information and data about different aspects of the real system to be modelled. At this stage, issues are related to the availability of information and data in order to create system models. All needed engineering information / data from which modelling data can be derived might not be available at the same point in time due to the sequential nature of real MS engineering process. In addition, from the analysis of the Cross Hüller machine design and build process, it was observed that early design phases are characterised by the use of basic data formats (i.e. paper based or generic computer formats), which do not allow the existing data and information about the real system to be readily translated into executable models. Conversely, real system engineering data might be defined at a level of detail that is too high (i.e. containing too much detail) compared to the modelling requirements. In such case, the information/data translation process also requires a simplification phase in order to adapt the data to the specific modelling needs. It is therefore essential to analyse and understand the real system design process in order to design and implement VPE tools that can be used effectively during early phases of the machine engineering lifecycle (i.e. mainly design phases).

3-2.4.1.3 Data Integration

The set of data initially chosen for creating virtual models of manufacturing system needs to be integrated into a coherent model that exhibits overall system geometrical, dynamic and behavioural characteristics. In known practical situations, the integration of modelling data often consists of low level programming activities invoking the manipulation of modelling / programming codes and thus requires a specific, intimate knowledge of both the modelling domain and of the system being modelled [7]. It is possible to simplify and / or partially automate the data integration phase by providing engineers with software environments specifically dedicated to the modelling of a particular type of system. However, the use of system-specific tools (i.e. modelling functions and interfaces) limits the modelling flexibility i.e. the extent to which such tools can be used to model alternative types of systems [8].

3-2.4.1.4 Model configuration and simulation analysis

The model configuration phase implicitly suggests that the modelling environments used to implement machine prototypes provides sufficient model re-configurability (i.e. allow various modelling parameters to be quickly configured) so that various machine model configurations (and hence real machine configuration) can be tested. Depending on the LoD (Level of Detail) at which a system model has been modelled and on the configuration mechanisms provided by the VPE, the duration and complexity of the configuration phase may vary significantly. Finally, models have to be run and analysed and the results interpreted.
The lifecycle of virtual prototypes described above can therefore be summarised as follows:

- Defining the system, system aspects, and level of modelling detail
- Gathering the corresponding information and data
- Processing i.e. data simplification and format translation into the type of modelling data, which are used to implement the computer model
- Integrating modelling data
- Configuring the final model
- Interpreting and analysing the model

The four first phases of a virtual prototype lifecycle (cf. Figure 3-8) can be referred to as the *modelling cycle*. From an engineering perspective, creating a virtual prototype is not a productive activity and merely consists of building a system representation, or model of the system. Because of the specific tools, skills and time required to complete this task, the use of 3D graphic based computer models is still considered as by the engineering community as "exotic" and not necessarily a core engineering activity. In order to exploit fully the potential of 3D virtual prototyping for engineering purposes (i.e. to support engineering activities and provide support to non modelling specialists) it is essential to design and implement future VPEs that allow the time and resources required to complete the modelling phase to be minimised. The foregoing proposes to link the differences between different types of manufacturing systems architectures (namely RMS and FMS) to different requirements for FMS and RMS's modelling. The aim is to highlight the limitations of current in supporting effectively the creation of RMS models.

### 3-2.4.2 FMS and RMS Virtual Prototyping

#### 3-2.4.2.1 FMS Virtual Prototypes lifecycle

Because typical FMS are materialised by fixed hardware configurations and integrated hardware and control software, the overall system configuration is relatively invariant throughout its lifecycle. The actual FMS *modelling phase* (i.e. 3D geometry modelling, kinematic links modelling, and pre-defined parameterised model behaviour functions) therefore represents a small fraction of FMS VP lifecycle. As illustrated in Figure 3-9, FMS *modelling phase* is only conducted once at the very early stage of the real system lifecycle. The same 3D computer-based model (encapsulating the various type of modelling data, and the data structure) is used throughout the real FMS lifecycle. Only a small fraction of the modelling data is re-configured (i.e. machine control) and the VPE provides pre-defined model reconfiguration tools and procedure that facilitate the process. In the future, FMS VP
could even be delivered as part of pre-defined and integrated machine control software. Some industrial robot manufacturers such as ABB Group [114] already provide 3D models (static solid models only) of industrial robots in various CAD formats, which can be integrated into a CAD model of a shop-floor layout.

Because FMS VPs are subject to very little changes and are mostly re-configured rather than modified it is accepted and tolerated that the *modelling phase* of such prototypes requires time, skill and resource mobilisation. Most of the modelling phase is transparent to system engineers and is decoupled from the real FMS lifecycle. The principal activity required in order to exploit FMS VP for engineering purposes consists of re-configuring the data describing the virtual prototype behaviours in exactly the same way in which a real FMS system has change capability based on the programmability of its control software.

### 3-2.4.2.2 RMS Virtual Prototypes lifecycle

The Re-configurable approach to Manufacturing System design (RMS) does not rely on the use of fixed hardware configurations and preset control software to provide flexibility. Because RMS invariants basis consists of highly independent and different constructs (i.e. separated, low granularity mechanical modules and software components), it is necessary to re-design the machine mechanical, control and software layout partially or completely, for every new customer order.
Such machine lifecycle means that a large part of the modelling cycle of RMS VPs has to be re-conducted every time the real or virtual system needs to be re-configured. As shown in Figure 3-10, the complete RMS VP modelling phase (i.e. 3D geometry modelling, modelling component layout, kinematic layout, control/display function/3D data mapping) needs to be re-conducted for every new design or design changes. The engineering lifecycle of RMS places significant emphasis on the VPE tools and their capability to make the modelling task (which is time consuming and requires skilled resources) transparent to system engineers [8] [7].

3-2.4.3 Design and implementation of a RMS Virtual Prototyping Environment (VPE)

The modelling process which underpins the use of existing commercial [118] [119], or academic [14] VPEs is essentially sequential in nature and involves the modelling of invariant 3D geometry and kinematics layout followed by attaching a control model to the 3D geometry and kinematics models. This bottom up approach (according to Xu et al. [7]) can be difficult to manage and can be particularly time consuming in the case of large-scale systems. It also requires a detailed knowledge of the domain of computer modelling and of the use of modelling tools in order to anticipate and account for potential modelling inconsistencies and difficulties [110]. Conversely, RMS prototyping will typically require complex and iterative progression between conceptual and detailed design stages as new functional and behavioural requirements are identified and matched to changing manufacturing requirements. This in turn requires the superimposing a modelling process on top of the real system engineering process. It is therefore essential to make the modelling task as transparent as possible in order to minimise the amount of time and resource allocated to the modelling activity.

A number commercial software developers [113] [118] [119] and academic research project [73] [104] [117] have focused on implementing VPE specifically adapted to the prototyping of FMS systems (Figure 3-11). Examples of state of the art "digital manufacturing" or "virtual
manufacturing" solutions are provided by commercial software developers such as Delmia Dassault Systems (Quest/IGrip packages) [113] [118], and Tecnomatix (eM-Power packages) [119]. Both of the examples provide software modules that can be used to model specific types of FMS commonly deployed in the manufacturing industry, typically NC controlled machining systems (Delmia Real-NC, Tecnomatix eM-Machining), industrial multi-axis robots (e.g. eM-WorkSpace, Delmia UltraArc/Spot/Paint) Co-ordinate Measurement Machines (CMM) (e.g. Delmia Inspect), PLC controlled systems (eM-PLC). Academic projects have also investigated the development and use of simulation environment for NC machine programming [73] [104] and machining process simulation [117]. As shown in Figure 3-11, a strong emphasis has been placed on developing engineering environments that integrate VPE tools as part of larger "total shop floor and production management software" and in a larger extent as part of a CIM infrastructure [73]. Commercial virtual manufacturing environment, are typically built around central database and database management systems (e.g. Delmia DS PPR Hub integrated Product Process and Resource Database) that allow various aspects of MS engineering processes (i.e. resource and information management, manufacturing process planning, material flow, resources and process management, support Supervisory Control And Data Acquisition (SCADA), data and knowledge management systems (e.g. eM-document, eM-planner)).

Such approach is adapted to FMS lifecycle and more particularly to the lifecycle of so-called manufacturing cells [78] or holons [81] typically built from FMS machines. However, the RMS approach to flexible machine has radically different implications regarding VPEs design. The research community has raised concerns about the adequacy of existing VPEs to respond to the requirements of current manufacturing industry [91] [92] and current FMS prototyping needs [7] [109]. There is a lack of sufficiently capable VPE tools that can be adapted and deployed in order to support the modelling and simulation of customised modular automation equipment built from mechatronic components like sensors, actuators and motion

Figure 3-11: Current approach to the design of Virtual Prototyping Environment adapted to FMS system lifecycle
controllers [8] [7]. As shown in Figure 3-11, this research proposes to investigate a new approach to the design and realisation of VPE best suited to RMS lifecycle. This new approach consists of developing VPE which focus is on modelling related issues, which need to be addressed in order to support effectively the prototyping of RMS. In a greater extent, the potential of VPE to serve as a tool that can be used to better exploit the intrinsic re-usability and re-configurability characteristics of RMS constructs (i.e. invariant basis characteristics), and therefore to provide better support for RMS design and change, is investigated.

3.2.4.4 Proposed innovative VPE for RMS virtual prototyping

When specifying and developing an innovative VPE, a key research aim was to maintain consistency between the real and virtual system architectures. RMSs are characterised by reusable constructs including machine mechanical modules, re-configurable control-related and other software components. Furthermore, machine re-usable constructs might be defined at different levels of granularity to match the requirements in terms of machine part re-usability (e.g. at actuator, group of actuator or machine station level) or design flexibility requirements. It is necessary to provide system engineers with VPEs which modelling capabilities can be adequately used to match the invariant basis of a given RMS system. It is understood that the needed integration between real and virtual system elements (and therefore between the VPE and real system engineering tools) would need a suitable common architectural model to be defined and adopted in order to ensure consistency between real and virtual design environments.

![Component-based approach to Virtual Prototyping Design and Implementation](image)

It was decided therefore that a so-called component-based approach to VPE design would be adopted in this research, which would help reinforcing the consistency between VPE
modelling functions and processes, and real RMS systems engineering lifecycle and tools. As shown in Figure 3-12, the component-based approach to RMS prototyping and VPE design is essentially an extrapolation of the RMS invariant basis (mechanical module, software control components) into a set of reusable and re-configurable modelling components, which can be used as an invariant modelling basis. The approach envisaged would theoretically allow the modelling phase (i.e. 3D modelling, kinematics modelling and modelling data integration) to be realised in a distinctive "component editing" phase (cf. Figure 3-12). Once a library of reusable and re-configurable modelling components has been created, machine modelling is expected to be essentially a process of configuring and composing components into a complete and fully functional model.

Some commercial VPE environments [118] [119] and academic research projects [73] [8] [14] have already developed concepts of re-usable modelling constructs. However, it appears that in most cases, the reusable objects mainly conform to constrained data structures that can only be managed using specific VPE software functions. The advantage of using modelling objects therefore tightly depend on the use of large and complex VPE software providing the needed data editing and management functions. The concept of modelling component as adopted in the present research goes beyond the use of a simple data model. Modelling components are defined as autonomous, re-usable and re-configurable software objects, which provide the functions required to support a large part of the modelling phase (e.g. model editing, configuration and composition functions), as well as the functions required to support the simulation and analysis of the final machine virtual prototypes (e.g. machine logic simulation engine, user interface). The functions typically supported by VPE software are therefore distributed amongst the modelling components from which virtual prototypes are composed. Finally, in this research modelling component are designed and implemented to be portable (i.e. independent of any proprietary software services and infrastructure) so that virtual machine prototypes and machine prototyping software functions can be deployed amongst distributed partners having only access to limited IT/IS infrastructure (i.e. web-compliant prototyping environment).

3-3 Communication-related functionality of Virtual Prototyping Environment (VPE)

The design, realisation, and change of any complex and large-scale system, such as a production line will typically require engineering activities, which directly or indirectly involve a number of partners. This is often necessary for commercial reasons but also so that
sufficient knowledge, tools, and expertise is made available to support the complete system lifecycle [86]. In such a context, enabling communication and engineering collaboration has been considered to be fundamental in achieving integration between functional entities [32] [34] and contributing to an organisation agility [23] [3] [29]. As explained in Chapter 2, this new type of engineering organisation is referred to in the literature as Virtual Enterprises (VE) [30], or Virtual Organisation (VO) [35] [37].

Three dimensional (3D) computer-based graphics (3D graphics) have potential as a basis to support intuitive communication between partners who have different perspectives and roles in achieving manufacturing system (MS) design. As highlighted by Yap et al. [115], today’s 3D modelling technologies and environments have reach a level of maturity that allow highly realistic models to be created which exhibit real system behaviour and provide advanced user interaction (cf. Figure 3-13). In addition, because of advances in computer and computer graphic technologies, the activity of 3D modelling is no longer available only to elite institutions and tightly linked to specialised hardware and software systems. The level of skill required to implement 3D models can now be possessed by non-specialists [115]. In this research, the potential of 3D machine prototypes in enhancing collaboration between system engineers and industrial partners is perceived as one of the principal advantage of 3D technologies and virtual prototyping activities. Whilst the first part of this Chapter has
focused on the use of 3D machine prototypes and prototyping environment as engineering tool, the second part places emphasis on the importance of maximising the use of 3D prototypes and VPE as a tool enabling communication and engineering collaboration in a distributed context.

3-3.1 Collaboration between Virtual enterprise partners: generalisation of the machine design and build case study

The study of the Cross Haller machine design and build process described in section 3-1.1 highlighted key issues related to collaboration between industrial partners involved in the design of production systems. Explicit and formalised interaction between Cross Haller and Ford was observed to occur at two distinctive points in the MS engineering lifecycle. Despite the fact that informal collaboration can be maintained between customer and machine manufacturer throughout the concept design process (i.e. between the two engineering meetings), a design review meeting, whose total duration is half of the time required to achieve the concept design is still required. Moreover, requirement changes are mainly taken in account during this design review. Essentially, there is only a weak and informal coupling maintained between MS builder and MS customer processes, which is mainly due to the lack of tools that enable continuous and effective communication between product and MS engineers. Both product and MS design processes (i.e. machine customer and machine builder processes) are conducted as two parallel processes, but consistency between product and production system design is only checked at the design review stage. After this point, changes in customer requirements will likely result in a time and resource consuming new design loop. It has been estimated by industrial collaborators (i.e. Cross Huller and Ford Motor who are probably representative of most European machine builder collaborators), that changes made after the design review will incur around one third of the cost of an average design project and one quarter of the overall design time [6].

3-3.2 Use of Virtual Prototyping Environment to support partners' collaboration

VPE software and 3D machine virtual models should provide executable and easily reconfigurable machine models to enable product and production system engineers to collaborate effectively on the design of production lines. Three-dimensional (3D) Dynamic machine models can potentially enable the testing and assessment of various machine configurations (i.e. various machines mechanical and control layouts) so that customer requirements can be more easily captured, interpreted and discussed. It should be noted that
only engineering issues are considered here. Collaboration between VE partners is also concerned with organisational, financial, and strategic issues that the use virtual machine model is not related to. Two aspects of the potential of VPE in supporting engineering collaboration between VE partners involved in the design of production systems were considered in this research and are detailed below.

3.3.2.1 Improving “As is” collaboration process

3.3.2.1.1 Use of 3D models to improve customer requirements specification and interpretation

It was considered important to design and create a VPE that could support the “As is” Virtual Enterprise partner’s collaboration process, an example of which is provided by the engineering process described in paragraph 3.1.1. Two collaboration phases between Cross Hüller and Ford, namely the Pre Sales and Design Review meetings were taken as an example and analysed in detail. When a project is initiated, system requirements should be clearly expressed by system customers and understood by system engineers. In the Cross Hüller / Ford Motor company case study, the definition and interpretation of system requirements strongly relies on the background knowledge gathered during previous design cases, on the knowledge of engineers from both companies, and on the communication channels and processes built during previous projects. The intuitive representation of machine systems that 3D models provide and the can serve as a common representation that can underpin communication and discussions on design solutions.

As stated by Yap et al. [115], the richness of the media used to support communication is highly important in conveying “tacit knowledge”, which is defined as the information content conveyed by communication media. Video is placed above pictures in terms of media richness, but below dynamic 3D models, which offer navigation capabilities and possibly
user/model interactions. In the same way, Toussaint [112] defines several levels of "functionality" that a 3D machine model can have, ranging from a simple display sequence of 3D geometries (i.e. sort of 3D based video sequence) which allow users to navigate through the model but not to interact with it, to a fully interactive model that can be dynamically simulated. As schematically shown in Figure 3-14, it is reasonable to assume that the quality of the user feedback from a model (i.e. observed level of cognition) is related to the media richness associated with a model [115] (e.g. visual fidelity, use/model interactivity, real time control and system behaviour dynamic display).

3-3.2.1.2 Issues when using 3D models as a communication basis in a VPE

One of the major barriers to making effective use of 3D models in support of the production system requirements specification phase is a lack of capability provided by current VPEs to generate models and prototypes that can be viewed and simulated without the need for specialised software. The level of model portability is perceived as being related to the requirements in terms of software and IT/IS infrastructure services that need to be deployed in order provide distributed partners with access to MS models that exhibit a given level of media richness (e.g. real time behaviour, model / user interfacing). One goal of this research was to assess what level of media richness and model functionality could be provided using exclusively Web compliant technologies (i.e. Web 3D modelling formats and languages). Public network infrastructures, corresponding technologies, and data formats are considered, in the context of this research, as the lowest IT level commonly deployed amongst partners involved in the design and build of manufacturing system.

The limitations of current state of art VPE software environments are illustrated with an example of collaborative interactivity between Lamb Technicon (another competitor to Cross Hüller in the design and build of automotive production systems) and a car manufacturer (name not mentioned for confidentiality reasons) on the design of a production line for car engine transmission systems [113]. The focus of the project was to predict and assess the performance of a new type of CNC-based production line implemented from Mach 1 CNC modules. The model was implemented using a Delmia developer's Virtual NC software module that allows mechanical systems to be modelled at a significant level of detail (e.g. mechanical joints friction modelling, real time behavioural simulation and analysis tools). However, both model visualisation and simulation required the functions of highly specialised software and services (e.g. remote access to central database, real-time data access systems, distributed objects broker infrastructure (e.g. D-COM, CORBA), high-end graphic workstations), which could not be deployed outside the machine builder's core-engineering environment. The lack of portability of both the models and of the software environment...
required to view and simulate the model limited the media used during engineering meetings
to only videos sequence (.avi files) of the plant 3D model simulations. Although video output
proved beneficial and could show the information required by the customer (number of
machine stations, production cycle time, plant layout), the potential for increased awareness
of the system operation, “what if” analysis and interactive testing and modification was
severely limited. The capability to dissociate highly detailed, fully functional and interactive
models from native and complex modelling and engineering software environments (e.g.
model / user interfaces, machine logic run time engine) is referred to in the context of this
thesis as portability.

3-3.2.2 Developing “To Be” VE partners collaboration

The overall aim of this research with respect to developing the capabilities of VPEs as
communication tool, goes beyond achieving portability of machine virtual prototypes, but
also targets the portability of the functions, which are required to configure and compose
component-based virtual prototypes. A hypothesis made at this point is that the level of
collaboration between Virtual Enterprises’ (VE) partners can be further increased by
considering 3D machine models as a bilateral communication media that can be deployed to
enhance further MS engineering collaboration.

As shown in Figure 3-14, the use of 3D machine models as a communication basis currently
consists of a one-way communication through which MS builder proposes design solutions in
a form that the customer can directly relate to (e.g. video of simulation sequences). However,
customer feedback remains informal and needs to be interpreted because it is not expressed in
a form that can be directly translated into modelling or engineering data. In this research, the
use of 3D modelling as communication media is aimed at providing machine customers with
access to the 3D modelling and prototyping functionalities. VPEs are therefore perceived as a collaboration tool that can be used by system customer to express requirements in a format that can be interpreted directly by system (machine builder) engineers. By such a change in practice, it would therefore be possible to create an environment where both partners (customers and supplier) express and interpret design solutions and system requirements in a common format.

Similar results are generally sought in the deployment of integrated IT/IS environments which can be deployed to support partners' communication and processes integration. This approach is characteristic of commercial "digital (or virtual) manufacturing" software solutions. Whereas such tools are effective within the controlled environment of a single industrial partner, it has been highlighted that deploying and maintaining such software infrastructures within the heterogeneous engineering environments formed via distributed partners collaboration, requires large (typically unacceptably) investments. In addition, high levels of IT expertise [35] are required, as well as a long-term commitment from both partners [44] to adopting a common IT/IS solution (which may be soon become obsolete).

The potential of emergent web compliant modelling technologies and formats in increasing the portability of software and function, which have so far required the use of specialised technologies, is assessed in this research. The emphasis has been placed on the portability, accessibility (low deployment and maintenance cost requirements) and usability (low skill requirements in the domain of modelling) of both models and modelling environments in order to provide effective prototyping services that can be deployed using standard network technologies. The ability of such a VPE and 3D machine models to be used outside the core-engineering environment of each individual partner is designated as a measure of "portability". It was presumed therefore that developing a VPE environment that can be deployed using exclusively freely available, web compliant technologies and public network infrastructures would result in a portable VPE and an effective communication / collaboration tool. The deployment of such tools would require minimal investments, efforts, and skills. These requirements have been used in this research to guide the design and development decisions made about the VPE created during this study.

However, bearing in mind the significant effort already made by the academic community with respect to implementing web based prototyping environments, and the diversity of types of manufacturing systems and the specific modelling functions they require, it was understood that fully achieving the design and implementation of a portable VPE would be beyond the
scope of a single Ph D study. In order to focus on area of development in which effective contributions to knowledge and practice could be made, this research was focused on:

- Defining the type of manufacturing system which modelling and prototyping should be supported.
- Defining what level of media richness VPE models and prototypes should provide
- Defining which of the VPE functions should be made portable (i.e. accessible by both customer and machine builder)
- Defining the nature of the IT/IS infrastructure upon which VPE tools can be deployed and used

Regarding the first point mentioned above, the choice was placed on sequential logic driven RMS, for which there is a clear lack of support from both commercial software developer and academic research. VPE enabling effective RMS prototyping should allow a wide range of applications to be covered, and could potentially be extended to approximate (i.e. low level of detail) prototyping of FMS. In addition, an additional goal was to ensure VPE functional openness so that modelling and prototyping capabilities could be extended to alternative types of systems is required. Considering the level of collaboration that the VPE developed in this research should provide the level of virtual prototypes media richness targeted was high. Accurate machine geometry 3D modelling, real-time machine behaviour (i.e. machine control logic) simulation, model navigation and advanced model / user interaction capabilities were essential. The VPE functions that did not necessarily required to be made portable considering were advanced model editing functions (e.g. 3D modeller, logic editing). However, VPE functions enabling model configuration and modelling component composition needed to be made accessible to both system engineer and customer to enable strong (bilateral) engineering collaboration. Finally, considering the level of portability targeted in this research, the World Wide Web (i.e. public) network infrastructure and internet compliant technologies were considered as the only basis on which VPE tools which level of portability was target in this research, could be deployed.

3.4 Chapter overview

The focus of this Chapter has been placed on explaining the rationale behind the development of a conceptual approach to capturing the requirements specification for the design and implementation of an innovative Virtual Prototyping Environment. Two primes observations have been made, which relate to i) the lack of capabilities of existing VPEs in supporting the prototyping of manufacturing systems characterised by a fragmented *invariant basis* (such a
Re-configurable Manufacturing Systems), and ii) the lack of capability of existing VPE in providing tools that allow the advantages of 3D based computer models (in term of intuitive communication media and collaboration tools), to be exploited in a distributed engineering context.

Subsequently, the approach adopted in this research was to design and implement an innovative VPE, which development follows a component-based approach. This allow to maintain the consistency between the generic architecture of RMS manufacturing system and the structure of RMS virtual prototypes, hence transposing the intrinsic RMS flexibility (i.e. RMS re-usability and re-configurability) to the domain of RMS virtual prototyping. In addition, the development of a VPE conducted in this research focuses on the portability of both virtual prototype models and the modelling environment functions (i.e. model analysis, model / user interaction, model configuration functions) so as to allow virtual prototypes and virtual prototyping activities can be shared by distributed partners.
Chapter 4 Component-based approach to Virtual Prototyping Environment (VPE) implementation

4-1 Chapter Introduction

This research primarily focused on implementing an innovative virtual prototyping environment (VPE) software tool. The component concept has been used as a basis to achieve the conceptual design of this VPE, i) an architectural model for manufacturing systems (MS) virtual prototypes (VP), and ii) as an architectural model to design and realise the VPE software itself. It is shown in this chapter that both VP and VPE components types and their corresponding functions have been merged into the same constructs to provide what is referred to as modelling components.

As illustrated in Figure 4-1, the component concept is therefore reviewed from i) a modelling perspective which relates to the realisation of VP components that are the virtual equivalent of real RMS constructs, and ii) a software engineering perspective which relates to component as VPE software constructs. Subsequently the approach adopted in this research to implement modelling components that are the combination of modelling and software constructs is presented. In the following paragraphs, the component concept is first reviewed from a general architectural perspective that provides a conceptual background to the design and realisation of those modelling components.
4-2 Overview of the concept of systems architecture

The concept of system architecture is widely used in the domains of software engineering [46] [48] [51], consumer product design [43] [45] [52], MS design [54] [68] [71], MS control system design [43] [66], shop floor control [79] [80], IS/IT engineering [47] [49], and enterprise systems engineering [41] [42] [38]. In manufacturing and engineering domains, the concept of system architecture has emerged from a need to describe, understand, manage and design complex systems such as products and software [47] [84] [85], complex MS [73] [11] [82] and complex enterprises and enterprise integration infrastructures [40]. System architecting provides a conceptual basis that allows a better understanding of complexity as an intrinsic characteristic of a given system in order to optimise the design and management of such systems (cf. Figure 4-2).

The management of existing systems can be organised around architectural models, which highlight fundamental aspects of the system being considered [49]. Architectural models therefore provide a common view of a system that various parties involved in the design or redesign of this system can use to share their perceptions of problem and discuss possible solution [44].

4.2.1.1 System decomposition concept

It is interesting to note that system architecting is tightly linked to the concept of system decomposition. Breaking a system down is an empirical approach to managing system complexity [50]. Browning [45] decomposes systems engineering tasks into three main steps, namely I) decompose system into “element” (i.e. system decomposition), ii) understand and document those elements (i.e. system constructs design) and iii) analyse potential re-integration of those elements via clustering (system composition). As stated by Garlan et al. [46], a system’s architectural design is concerned with its decomposition into “elements” and their interactions. Zwegers [44] also defines systems’ architectures as the manner in which the “components” of a specific system are organised and integrated. In the domain of software
engineering, Soni et al. [48] state that software architecture is concerned with capturing the structures of a system and relationships among "elements". The performances that can be achieved through an implementation of an effective system architecture, depends on i) the nature and characteristics of the system's constructs [47] [49], and ii) on the methodologies used to compose or modify a system's composition and functions [70]. The following subsections are aimed at reviewing the various approaches that can be adopted to achieve decomposition of systems into "systems constructs".

4-2.1.2 Architectural Guidelines

4-2.1.2.1 System domains

Erens [52] associates the concept of domains with the various phases that characterise a system design lifecycle. According to Erens [52], the development of systems is characterised by several "activity domains", defined as functional, technological and physical domains, which contribute simultaneously to the creation of a system. This approach to the definition of domains focuses on the process that consists of linking the system functional description to the technological and physical system realisation. A similar approach is adopted by Bouti [55] who defines the "functional, behavioural and structural "views" of a system. It should be noted that the term "view" was used by Bouti [55] to designate what was defined by Erens [52] and Zwegers [44] as design domains. Interestingly, Zwegers [44] states that in the technological domain, "system modules" are defined relative to the functions defined in the functional domain, which suggests that the decomposition of a system is purely functional and that system modules are differentiated by the functions they support.

![Figure 4-3: Manufacturing System functional domains](image)

In this research, the perception of domains as different phases of a system design lifecycle was irrelevant because associated with the component editing phase rather that with the component utilisation phase (i.e. configuration, re-use). Although Vernadat [38] focused on manufacturing enterprises systems ([41] [42]), his definition of domains as the "functional areas" of a system is adopted in this research. Vernadat [38] describes domains as the result of the breakdown of the overall system's functionality into functional domains. In the same way, in this research, domains refer to the functional decomposition of the functions supported by modelling components.
4-2.1.2.2 System views

The second architectural concept reviewed in this section is the concept of systems' views, which are defined by Proper [49] as "the description of a system from the perspective of a related set of interests of a system viewer". Zwegers [44] defines system views in a similar way, as being "windows through which selective aspects of a system can be observed". Whilst some particular aspects or attributes are emphasised, other extraneous detail is suppressed to avoid obscuring the real issues at stake. It should be noted that the concept of views is different from that of functional domains as adopted in this research. Views can be used to emphasis on some aspects of a system, which are common to several domains. For instance, the CIMOSA reference architecture provides a pre-defined set of views, namely the organisation, resource, information and functions views, which can be used to consider different functional domains.

![System views](Figure 4-4: Manufacturing system views)

In this research, the concept of views is used to refer to essential aspects of MSs that are considered by system engineers during the design and change of MSs. It should be noted that in the domain of MS engineering, systems' views are materialised by representations [115] [14], which can be materialised by a variety of media (sometime computer-based) models (sometime executable computer-based models).

4-2.1.2.3 System decomposition hierarchy

As highlighted by Zwegers [44], the concept of hierarchy is often associated with hierarchical control structure; however, the boxes-within-boxes representation is more appropriated than the bosses-above-bosses representation to describe system hierarchy. In this research, the concept of hierarchy is considered as an empirical approach to the management of system complexity, which consists of realising a hierarchical decomposition of a system into smaller and simpler sub-systems [44]. Whereas views divide a system into "system aspects", hierarchy divides system into "sub-systems" [54]. Zachman [47] provide a definition of sub-
systems, which also refers to the concept of functional domains: “A sub-system of a system S is a subset of E (the set of elements of S) with all the attributes of the elements in question”. This is interpreted in the context of this research as a sub-system being a sub-set of the system S that contains part of the definition (i.e. attributes) of each functional domains of S.

![Diagram](image)

A complementary definition is given by Zachman [47]: “An aspect system A of a system S is the set E with only a subset of the original attributes” which can be interpreted as A being the result of views showing only the elements related to one functional aspect of a system.

4-3 Flexible Manufacturing System architecture for Reconfigurable Component-based Machine

The COMPAG / COMPANION project has focused on investigating new approaches to the design and implementation of a component-based (CB) machine control architecture. These two ESPRC funded projects have focused on i) the design and realisation of a distributed CB machine control [6], ii) the design and realisation of various software tools to enable component-based machine design and change [110] [14], and iii) on the business process analysis and tools required to evaluate the benefits of adopting a component-based machine architecture [69]. The research described in this thesis focuses on designing and implementing a Virtual Prototyping Environment (VPE) that can be used to support the design and change of CB manufacturing machines and therefore complements the CB machine-engineering environment developed in the context of the COMPAG project.

4-3.1 Component-Based Paradigm for distributed machine control logic

As illustrated in Figure 4-6, the distributed machine control architecture is implemented via control nodes which are realised by electronic devices that contain sequence and interlock
capabilities (namely control elements) corresponding to physical machine components (e.g. sensors, actuators, timers, counters) in a form that a process engineer can directly relate to. Complete machine systems control logic is defined via configuration and composition of the machine components rather than by writing specialised code in for example ladder logic, sequence charts or C programs. This approach enables highly autonomous, reusable and configurable machine control components to be "plugged onto" industrial networks and composed into complete and distributed machine control systems. The term "distributed" is used to denote that CB control systems conceived and tested during the COMPAG project do not require a central programmable logic controller (PLC) or personal computer (PC) based controller. The control data is generated via a software tool referred to as the Process Definition Editor (PDE), which is part of the engineering software environment implemented to support the lifecycle of the control system design and build process.

In the COMPAG project, the CB paradigm has principally been applied to the design and realisation of RMS control software. However, the concept of a control component should ideally be extended to all aspects of machine systems (i.e. mechanical, software) in order to realise completely autonomous, re-usable, and re-configurable mechatronic components that can be readily composed into various machine systems configurations depending on the end-user requirement and requirement changes. Realising such integrated mechatronic components effectively consists of isolating the different phases of component design i.e. the functional, technological and physical (design) domains (and the "zigzag" design patterns resulting from the need to investigate simultaneously various domains of design to avoid inconsistency [49]) from the actual machine design (i.e. component composition process). Therefore, the component model should ultimately be extended to all aspects of machine systems engineering and should be used as a common model across machine engineering domains.
4-3.2 Common hierarchical model

MSIRI's CB machine paradigm defines a machine hierarchy by formally defining the various levels of abstraction at which production machines can usefully be considered. As defined by the COMPANION CB terminology (i.e. from a machine control perspective), logic “elements” are used to represent the smallest (atomic level) reusable constructs. Elements describe the possible logical states in which a machine sensor or actuators can be and the possible transitions between states (illustrated in Figure 4-6 as state based diagrams). Several elements can be grouped as “components” which describe the possible states of various machine actuators and sensors that form a machine physical component. Components can be grouped into sub-systems and any complete machine is defined as a system.

![Diagram of component-based machine hierarchy](image)

The *component* level of the COMPAG machine hierarchy is used as a common hierarchical model across the whole range of CB machine engineering tools development. “Modules” are considered (in CB terms) as existing at the same level of granularity, or hierarchical level, as of a machine component. A machine module encapsulates control logic machine components, and any other components corresponding to other aspects of the system (e.g. mechanical module, 3D virtual component model, module lifecycle support services components).

4-4 Three Dimensional (3D) models of machine systems

The *component model* or CB approach to re-configurable machine design and change provides an architectural model that can be used to co-ordinate the design of re-usable machine components across various engineering domains. However, each aspect of the machine should be designed and tested with regard to other domains of design in order to ensure the consistency of the final design. As emphasised by West [14], engineers concerned
with the design of different aspects of the system typically use completely different mental models and make use of different tools to visualise, configure and compose machine components. For instance, mechanical engineers perceive production machine systems as a particular layout of mechanical parts and actuators, each of which possessing particular dimensional and kinematic characteristics (cf. Figure 4-8). Typically, 2D or 3D CAD models are used to provide representations of such aspects of mechatronic systems. On the other hand, control engineers focus on machine behaviour as a sequence of logical states and transition conditions, and the timing of machine operations. Specialised tools can be used to provide representations of machine logic, which in the case of sequential logic, typically consist in state/transition based diagrams such as Grafcet, or ladder logic diagrams and timing diagrams.

As illustrated in Figure 4-8, the concept of modelling component developed in this research is aimed at providing a common model of RMS components. Firstly, modelling components from which VP are composed result from the integration between several types of modelling data describing the real system mechanical, kinematic and control characteristics. As such, modelling components provide a highly intuitive machine representation, or view in architectural terms, that can be used to coordinate the design of various aspects of MS design and to share the corresponding tacit engineering knowledge [115]. Secondly, modelling components provide virtual constructs that can be used as a common hierarchical model based on which the design and realisation of various types of re-usable machine constructs (e.g. mechanical machine modules, control software components level of granularity) can be co-ordinated. By such practice, the overall machine design and the development and integration of various types of machine constructs can be achieved more effectively. It should be noted that modelling components stands themselves as on of the constructs from which complete real machine modules are built (cf. Figure 4-8) but provide a intuitive and synthetic view of the complete machine module.
4-5 Component concept in Software engineering

This paragraph is focused on the means by which modelling component can be realised practically in the form of software constructs that allow complete manufacturing VPs and VPEs to be readily and effectively composed. The focus is placed on the re-usability, re-configurability and composability of modelling components which is aimed at solving the problem associated with RMS 3D modelling (i.e. modelling time and skill required) to be avoided.

4-5.1 Software Component definition

The term “component” is widely used in the domain of software engineering but is rarely defined and is in danger of becoming “all things” and therefore meaningless [60]. A generic description of software components was advanced by Van Baelen [122] in the form of a “meta model” reproduced in Figure 4-9. This meta model of a component describes a component unit that participates in the composition of a larger system. Each component has various characteristics (identifier, behaviours, role, information, knowledge) and interfaces through which it can communicate with other components.

![Component Meta model. Source Van Baelen [122]](image)

As highlighted by Morton et al. [121], any (software) component is initially aimed at providing a “well-defined, pre-compiled building block, with standardized interfaces” that can be composed into complete and more complex (structurally and functionally) software systems. A similar and more practical definition is given by Collins-Cope [60] who defines a component as “a binary (non-source code) unit of deployment such as a .class file in Java, a .o file under Unix, or a .dll file under Windows”. It should be noted that both definitions introduce the notion of a software component as a well-defined and clearly distinct source or binary code object. Morton et al. [121] state that components are architectural constructs for larger software, and refer to libraries from which “systems can be assembled using appropriately configured software components”. This definition implicitly suggests that
if different functionalities provided by different components are reused together, the corresponding components should be merged into one component providing advanced functionalities

- the level of functionality provided by a component should be defined with respect to the need for re-usability. Low granularity components provide re-usability and system design flexibility, but requires additional composition efforts.

The following sub-sections are aimed at providing a review a various types of software constructs commonly utilised in the domains of software design. These various types of components are differentiated with respect to their functionality of the level of granularity at which they are defined.

4.5.2.1 Software components and objects

The concept of Object-Orientation (OO) has revolutionised the way that large software systems are designed, maintained and managed [56]. So-called OO concepts has provided software designers with means to better manage complexity associated with large systems, and to achieve code reusability [58]. Both software objects and software components stand as constructs from which larger software, or more complex software functions can be implemented. In the software engineering community, the difference between objects and components has been a subject of significant discussion [116].

The first concern is the level of functionality that differentiates objects and components. Miller [126] compares programming languages abstract data types (ADT), which are user defined data types that can be defined and instantiated, to objects. Miller [126] observes that if objects are "above ADT in terms of functionality" components are a "step beyond object orientation". Pidd et al. [125] points out the close link between object orientation and ideas underpinning component-based development. However, Pidd et al. [125] argue that software components need not be based upon models of inheritance, which is fundamental to object orientation paradigm. In the same way, Szyperski [116] states that classes (and therefore objects) are usually too tightly coupled with certain other classes to be useful components (i.e. to provide useful reusable constructs). It is thus useful to group a set of related classes and possibly further resources into a component [116]. An alternative approach was proposed by Stritzinger [100] who states that at run time components have similarities with objects, in that component also store a state in attributes which may be encapsulated and provide operations for testing and manipulating that state. Furthermore, components can be defined by one or more class thus object orientation is no pre-requisite for components. Component concepts
- if different functionalities provided by different components are reused together, the corresponding components should be merged into one component providing advanced functionalities
- the level of functionality provided by a component should be defined with respect to the need for re-usability. Low granularity components provide re-usability and system design flexibility, but requires additional composition efforts

The following sub-sections are aimed at providing a review a various types of software constructs commonly utilised in the domains of software design. These various types of components are differentiated with respect to their functionality of the level of granularity at which they are defined.

4-5.2.1 Software components and objects

The concept of Object-Orientation (OO) has revolutionised the way that large software systems are designed, maintained and managed [56]. So-called OO concepts has provided software designers with means to better manage complexity associated with large systems, and to achieve code reusability [58]. Both software objects and software components stand as constructs from which larger software, or more complex software functions can be implemented. In the software engineering community, the difference between objects and components has been a subject of significant discussion [116].

The first concern is the level of functionality that differentiates objects and components. Miller [126] compares programming languages abstract data types (ADT), which are user defined data types that can be defined and instantiated, to objects. Miller [126] observes that if objects are “above ADT in terms of functionality” components are a “step beyond object orientation”. Pidd et al. [125] points out the close link between object orientation and ideas underpinning component-based development. However, Pidd et al. [125] argue that software components need not be based upon models of inheritance, which is fundamental to object orientation paradigm. In the same way, Szyperski [116] states that classes (and therefore objects) are usually too tightly coupled with certain other classes to be useful components (i.e. to provide useful reusable constructs). It is thus useful to group a set of related classes and possibly further resources into a component [116]. An alternative approach was proposed by Stritzinger [100] who states that at run time components have similarities with objects, in that component also store a state in attributes which may be encapsulated and provide operations for testing and manipulating that state. Furthermore, components can be defined by one or more class thus object orientation is no pre-requisite for components. Component concepts
and OO concepts are therefore presented as being orthogonal, which means that object orientation and OO programming is not essential, but can be useful as means of designing and implementing better components.

Finally, Collins-Cope [60] places emphasis on the fact that "objects exist at run-time, whilst components are binaries that are deployed". Interestingly, Szyperski [116] links components and objects to the type of infrastructure required to deploy them (in particular the communication infrastructure between distributed object and components). Szyperski [116] provides a very detailed review of DCOM (Distributed Component Object Model) technologies and highlights differences between the ORB (Object Request Broker) and CORBA like architectures for object communication. Although details of those technologies are largely out of scope for this research it is interesting to note that the ORB architecture provides a more generic communication mechanisms than the DCOM architecture due to its use of a more standard and formal object communication model.

4-5.2.2 Components and software agents

The term "agent" is used designate software systems, which level of functionality is very high in comparison to that of software objects or components. Muller [130] describes agents as "autonomous software programs, which are capable of flexible action in complex and changing multi-agent environments". Maturana [127], states that agent technology is derived from distributed artificial intelligence. In the same way, Barthes [128] describes software agents as autonomous, cognitive entities with deductive, storage and communication capabilities. The term "autonomous" used in connection with an agent, means that an agent can function independently from any other agent. Shehory [129] states that agents are autonomous; where autonomy refers to components that do not depend on the properties or the states of other components for their functionality. However, a MAS (Multi Agent System) relies on interactions between agents. A MAS system is supported by specific communication protocols and languages such as the KQML agent communication language [131].

It is interesting to note that unlike objects or components, a basic set of functions that an agent should provide have been generally pre-defined. As stated by Muller [130], at the heart of an autonomous agent is its control architecture, i.e. description of the modules and of how they work together. Over the past few years, numerous architectures have been proposed in the literature, addressing different key features that agents should have. Shen et al. [128] state that a software agent should provide i) a communication interface, ii) a symbolic model of other agents, and associated methods to use them, iii) a model of its own expertise and iv) a model of the task to be performed. This definition of agents strongly emphasises "role",
"knowledge", and "goal" characteristics of software constructs generically described in a generic way in Figure 4-9 (cf. paragraph 4-5.1).

It appears therefore that agents can be positioned above (in terms of level of functionality) software constructs typically designated as object or components. The concept of agents seems to emerge from the need to design complex and autonomous entities which functions are built from a set of services. Those services can be provided by components, which in turn are built upon the basic functions that software objects provide (i.e. functional decomposition). It should be noted that a system might be composed from similar agents being configured to have different goals. In the same way, agents or components can be composed respectively from several similar components or objects instances with different configuration (structural decomposition).

4-6 Review of various approaches to design and implementation of “modelling component” models and modelling software environment

A number of research projects have focused on developing modelling software environments based to enable the editing of 3D virtual prototypes of mechatronic systems. A number of these projects have focused on the use of the component concept to enable a more effective modelling of functionally complex and / or large-scale systems (i.e. by seeking component re-usability, re-configurability). The following sub-sections are focused on reviewing two distinctive approaches to the design and implementation of i) modelling components as constructs of VPs and ii) as construct of VPEs software, which are representative of the different approaches adopted by the academic research body and by commercial VPE developers.
4-6.1 Models of “modelling components”

Salmela et al. [97] propose the use of so-called “Smart Virtual Prototypes” composed from “components as building blocks” [97] that aim at simulating the appearance and behaviour of electronic products (i.e. mobile phones). The Smart virtual prototypes have a ternary structure depicted in Figure 4-11, which consists of a UI User Interface object and a so-called “virtual component” (Java code describing the component behaviour). An “interactor component” (Java applet) is used links the VRML code and the “virtual component” code hence achieving a mapping between 3D modelling code (VRML) and behaviour description code (Java).

Salmela et al. [97] confusingly used the term “component” to designate both the complete model (“digital product component” in Figure 4-11), and its internal components (Key, Key Interactor and UI). It is interesting to note that each internal components has a specific function (e.g. 3D geometry, model behaviour, modelling / control functions mapping), which denotes that internal component have been designed according to a functional decomposition of the overall system functions (cf. Figure 4-11). Such approach can be effective for functionally simple and small-scale systems. However, if larger and behaviourally more complex systems have to be modelled, the complexity of each component will increase; both geometrical and behavioural modelling components will become less manageable (due to increased complexity), whereas the mapping between those components (i.e. Interactor component’s complexity) will become more difficult to manage and modify.
The approach to modelling component design proposed by Adolfsson et al. [8] aimed at supporting various phases of the engineering lifecycle of "re-configurable, modular and component-based machine systems". The component model proposed by Adolfsson et al. [8] is represented in Figure 4-12. This model of components exhibits a detailed internal architecture that achieves the integration of various objects providing various types of function (cf. Figure 4-12) required for the modelling of manufacturing systems. Adolfsson et al. [8] components' internal structure is realised by complex code objects and interfaces implemented using various programming technologies, and integrated using various architecture model (e.g. DCOM, CORBA, client-server).

![Component-based model](image)

**Figure 4-12: Modelling component as defined by Adolfsson et al. [8]**

In the same way that a component-based architecture provides better flexibility and change capability of component-based systems, the detailed internal structure of Adolfsson et al.'s component model provides better component management and change capabilities. As shown in Figure 4-12 (right part), both the type of objects (and hence the type of functions) and the number of objects and objects composition can be changed depending on the type of machine part that has to be modelled.

### 4.6.2 Analysis of "modelling component" models

The decomposition approach and the internal component structure of the first component model (Salmela et al. [97]) is clearly the result of the constraints imposed by the modelling technologies used (i.e. VRML, Java / Java Applet). Such model is focused on the modelling task management rather than on the management of the model complexity (very low in this case). The second component model (Adolfsson et al. [8]) has been entirely defined based on the nature of the system being modelled. Both internal component objects' functions (i.e.
portable, state transition, error monitoring) and component structure (i.e. objects' cardinality relationships, sequence of events) are defined according to the constructs that compose the real system (i.e. RMS manufacturing systems).

4-6.3 Approach to modelling environment design and implementation

This sub sections is aimed at analysing the type of modelling environment associated with the modelling component models previously reviewed. The modelling environments implemented by Adolfsson et al. [8] and Salmela et al. [97] were studied in details because they were viewed as being representative of the typical approaches adopted in both the academic and commercial domains for the design and realisation of VPEs. Several aspects were considered, including the choice of modelling technologies, the approach to model editing, management, and the approaches to VPE software design.

Salmela et al. [97] modelling environment is essentially built upon freely available or low cost standards programming languages and modelling technologies (i.e. VRML and Java, V-Realm Builder). The proposed VPE results from the integration several tools that enable the editing of modelling data and the deployment and use of the final VPs. As shown in Figure 4-13 the process of editing, (re-) configuring and using virtual prototypes involves the use of a CAD modeller, a CAD-to-VRML data translator, the parsing and re-editing of relevant modelling code sections (e.g. VRML nodes, editing of Java and JavaScript code), and the mapping implementation of various user interfacing functions implemented as part of the VPE’s interfaces.
This type of VPE design emerges from a sequential approach to the editing and integration of modelling data. Such VPE requires frequent code editing due to the lack of integration between the various software components, or the design of dedicated software functions that allow specific tasks to be accomplished (e.g. V-Realm Builder in the case of Salmela et al.'s research).

On the other hand, the modelling environment developed by Adolfsson et al. [8] is built upon highly complex and dedicated commercial applications (i.e. Delmia IGrip/Envision, Delmia Quest for discrete event simulation engine, ISaGraph for machine logic code editing) integrated using a specifically designed and proprietary integration infrastructure (i.e. CORBA, DCOM, central database and database management system). Figure 4-14 describes Adolfsson et al.'s [8] VPE enabling the design and "virtual engineering" (VIR-Eng) of RMS VPs. Adolfsson [8] characterises VIR-Eng as a suite of highly integrated software tools resulting from the integration of several "work packages" designated as MMDE (Modular Machine Design Environment), and built upon functionality provided by Delmia IGrip/Envision commercial discrete/continuous simulation software.

The MMDE provides the tools needed to support all the facets of the design process (i.e. mechanical, kinematics, and control logic data editing) using graphical interfaces. Once edited
those data are exported into the DCSE work package, which results from an integration of three "(software) elements" which are the CSDE (control system design environment), the CDE (component design environment) and the DRE (distributed run-time environment). The DCSE provides a run time environment in which data edited within the MMDE can be simulated, and translated into a real system data configuration. Finally, a so-called IIS (Infrastructure and Integration Services) provide the services required for the two work packages described above to exchange design models, information and data. Thereby, the IIS comprises the "component library, and the central data repository".

4-6.4 Analysis of VPE design and implementation approaches models

The fundamental difference between the two approaches to modelling component and VPE design and implementation reside in the choices of modelling technologies and formats used. Salmela et al. [97] research is typical of the approach adopted in the academic domain and rely standard modelling formats (e.g. VRML), modelling language (i.e. JavaScript / Java, HTML) and network infrastructure (i.e. Java server-side (model behaviour) and JavaScript / VRML client-side (model geometry and dynamic display)). Such approach has two main advantages. Firstly, there is no need to develop proprietary software infrastructure to enable the use and simulation of machine VPs. Secondly, model implemented upon standards web-compliant technologies are intrinsically portable since they can be deployed between partners who have access to standards and public domain web technologies. Conversely, the limitations of standards web-based modelling and programming technologies (initially aimed at developing lightweight and simple applications) can constrain the development of highly specific application (i.e. engineering applications) such as VPE for manufacturing systems.

The resulting VPE consists mainly of software modules (e.g. CAD-to-VRML translator, code generator, component interactor (Salmela et al. [97])) which functionalities are aimed at supporting the different model editing phases (3D geometry editing, model behaviour code editing, codes mapping). Therefore, the VPE architecture is mainly the result of the modelling process imposed by the technologies, which lower its value as engineering tool.

Adolfsson et al.'s [8] VPE design was focused on i) defining a component model consistent with the nature of the system being modelled (i.e. RMS), and ii) to make the modelling process transparent to the system engineers. To achieve this goal, Adolfsson et al. [8] have used highly specific commercial tools and software integration infrastructure (namely Delmia tools) to hide the editing and integration of various type of modelling data into (inherent to any modelling process) behind a process that consists in composing predefined model components very close in nature to the real RMS "modules".
Figure 4-15: Inconsistency between Component and component-based systems models, and implementation

Figure 4-15 illustrates how the component model effectively translates into a VPE software infrastructure. Although modelling components are defined as autonomous constructs, they practically consist of a set of distributed code and data objects, which functions are coordinated via real time integration infrastructure (IIS). This implementation of modelling component is in contradiction with how most of the software engineering community defines software components [116] [121] [122]. In particular, Collins-Cope [60] clearly differentiates objects from components because “objects only exist at run time (once loaded in computer memory), whereas components are pre built binary units of deployment” (i.e. a concrete and portable computer file (e.g. .dll, .class)). Adolfsson et al. [8] therefore make a distinction between VIR-Eng’s “components” and “simulation entities”. Simulation entities refer to concrete software objects (or modules), which can be “elements” in Quest Delmia environment, “devices” in IGrab/Envision, or other programming units (procedures and functions) executed by the “simulation engine”. On the other hand, simulation entities exist as run time functions that are coordinated via the IIS services. This approach to linking component model and VPE software architecture has a major disadvantage since the benefits of the component approach to virtual prototyping (i.e. virtual prototypes re-usability, re-configurability, modelling flexibility) are tightly linked to the underlying IIS services required to deploy and use virtual prototypes. It follows that the portability of virtual prototypes is seriously impaired since virtual prototypes can only be deployed amongst partners who have access to these proprietary and rather complex VPE IIS services.

4-7 Portable Component-based Virtual Prototyping Environment (PoCo VPE) design

In the previous sub section, two distinctive approaches to the design and realisation of modelling components and VPE software tools have been reviewed. The development of
public domain VPEs focus on the virtual prototypes portability by making use of standards web compliant technologies. However, a good knowledge in the domain of 3D modelling (i.e. modelling process, internal modelling data structure) is required in order to compensate for poorly integrated software infrastructure. Conversely, commercial VPEs development rely on complex software integration and data management services to make the modelling related tasks (i.e. modelling data editing, data integration, model deployment and use) transparent to the user, and therefore to maximise the value of VPEs as engineering tools.

The VPE developed in this research is aimed at investigating an alternative approach to the design and realisation of VPE. The PoCo VPE development is focused on:

- Design and realisation of *modelling component* adapted to the prototyping of RMS (e.g. similar to Adolfsson et al.'s [8]) i.e. which characteristics and internal structure is not the result of modelling related constraints (i.e. unlike Salmela et al.'s [97] component model)
- Preserve the portability of *modelling components* and hence the portability of complete virtual prototypes by making use of web-compliant modelling technologies exclusively (e.g. VRML, JavaScript) to realise modelling components.
- Simplifying the modelling related tasks (i.e. *modelling component* editing, configuration and composition, virtual prototype simulation and user / model interaction),
- Avoiding the use of complex software component and software integration services (e.g. large database and DBMS, distributed / real-time object broker architectures, complex user interfaces)

4-7.1 VPE generic description and requirements specification

The term Virtual Prototyping Environment (VPE) is used here to designate a software environment that provides the functions required to support the lifecycle of component-based virtual prototypes. This functionality includes:

- component editing
- component configuration
- component composition (into a complete machine model)
VPE typically provide data editing and data management systems corresponding to the specific functions needed to support each phases of a VPs lifecycle. A graphical user interface layer might be implemented to facilitate this process. As highlighted in Chapter 3 and illustrated in Figure 4-16, a major benefit of adopting a component-based approach to the design, implementation and change of VPs is the dissociation of the component-editing phase from the component utilisation phase (i.e. configuration and composition).

As highlighted in Chapter 3, portability of complete and fully functional virtual prototypes was a fundamental requirement in the development of the PoCo VPE. In addition, it was essential to enable distributed partners to access the VPE functions that allow individual component and component-based prototypes to be reconfigured (i.e. the functions required to re-configure and re-compose modelling components) so that PoCo VPE can be used as a bilateral collaborative tools (cf. Chapter 3). Finally, the user interfaces required for model / user interactions during the utilisation phases should also be portable so that virtual machine prototypes can be used and analysed by distributed partners.

4-7.2 Overview of the PoCo component model and PoCo VPE design approach

A characteristic common to all VPE studied in the context of this research is the low coupling between the VPE software architecture and the modelling components from which virtual prototypes are composed. Modelling components are usually conceived as data objects which editing and use is supported by VPE software modules providing the functions required to support specific phase of the component-based prototypes lifecycle (i.e. 3D geometry editing, control code, modelling data integration, component configuration and composition). In this research, the design and realisation of modelling component and VPE tools is based on the
original idea that *PoCo modelling components* should be portable, binary unit of deployment [55] encapsulating both modelling data all functions required to support the component-based prototypes lifecycle.

![Diagram of PoCo (Portable Component-based) approach to Virtual Prototyping Environment (VPE) implementation.](image)

This research has been focused on designing and implementing portable and re-configurable *modelling components* that encapsulate functions are related to the configuration, and composition of component-based VPs (Figure 4-17). By focusing on the portability of such PoCo components (i.e. exclusive use of web-compliant modelling format and technologies), it is possible to achieve portability of both component-based VPs and VPE software. As shown in Figure 4-17 a PoCo modelling component typically encapsulates the modelling data and functions that allow component-based models to be described (typically contained in VPE central database systems). These are referred to as “modelling data / functions”:

- **Various types of modelling data**: this principally encompasses 3D geometry and kinematic modelling data describing real mechanical machine components.

- **a description of the machine behaviour**: (sequential state based machine logic model in the case of this research), and the mapping information that allow these data to be linked into a fully functional VPs

- **dynamic display functions**: e.g. dynamic display of 3D geometry, cosmetic display (e.g. colour, alpha level change), position / orientation interpolation functions

- **a machine logic simulation engine**: this is equivalent to distributed control nodes that compose a RMS control

In addition, *PoCo modelling components* encapsulate the functions required to support the configuration and composition of components into complete model. Such functions are usually provided by software modules that enable the management (re-use, configuration) of modelling data. It can be considered that a certain degree of "knowledge" of the overall
system architecture is encapsulated in the *modelling components* themselves. A PoCo modelling component therefore encapsulates what is referred to here as "structural functions" which are:

- **logic related composition function**: enable individual component control logic to be composed into a complete RMS control sequence
- **geometric / kinematic composition functions**: enable the composition of modelling component geometry, and the management of parent-child relationship between kinematic parts
- **virtual prototypes analysis functions**: provide basic analysis functions that allow various machine configuration to be tested (e.g. component state visualisation, error reporting services, auto / manual simulation modes)
- **model / user interfacing functions**: provide interfacing functions enabling basic user model interactions e.g. user input (mouse click), visual feedback (cosmetic changes), spatial navigation (view point editing / selection), information display

As shown in Figure 4-18, the concept of merging *modelling components* and VPE software modules consists in designing and realising highly functional, and highly portable software constructs (i.e. PoCo modelling components) encapsulating all the functions required to support their own lifecycle (subsequent to the editing phase, i.e. (re-) configuration, composition). By such practice, it is possible to reduce dramatically the complexity of external functions not directly related to the modelling or simulation of virtual prototypes (e.g. modelling component library management, modelling component’s parameter configuration interfaces, extended analysis tools and functions). Similarly, the integration of PoCo virtual prototypes into a real system engineering environment (e.g. use of 3D machine models as HMI interface, as support for logic debugging, machine monitoring and
maintenance tasks) is simplified since all necessary functions and interfaces are encapsulated with the components from which VPs are composed.

As explained in chapter 6, realising functionally and structurally complex code objects using exclusively standards web compliant modelling formats and programming languages can be difficult. In particular, it can be difficult to ensure effective management of the code describing complex software objects, which would result in reduced re-configurability, re-usability and openness of PoCo modelling components. The use of more appropriate programming languages and modelling formats (i.e. object-oriented language, low level modelling formats) is a potential solution. However, such approach results trading VPs portability, and therefore was not adopted in this research. Instead, the potential of applying object-oriented paradigm to standards web-based Virtual Reality Modelling Language (VRML) modelling language has been investigated (cf. Chapter 7).

### 4-7.3 PoCo component infrastructure

Modelling technologies (i.e. modelling language and 3D modelling formats) can be classified as either proprietary or public domain. Proprietary formats are developed by commercial CAD and digital manufacturing environment developers (e.g. Dassault / Catia / Delmia) to suit specific needs. Conversely, new 3D modelling technologies have emerged from the growing popularity and availability of public networks. So-called Web3D technologies, such as the VRML standards (Virtual Reality Modelling Language) allow lightweight but highly functional models (i.e. high Level of Detail (LoD) modelling), advanced modelling capabilities (e.g. dynamic models, behaviour modelling, user interfacing, advanced cosmetics) to be realised. The VRML language marks a fundamental change in the perception of classic 3D modelling since it provides a very high level of modelling functionality and some programming capability in a highly portable and web compliant format. Because of its inherent flexibility, portability and open structure [94] [99], VRML has been ratified as an ISO standard [95] for modelling formats. The strength of VRML resides in the fact that:

- VMRL defines pre-defined modelling objects called nodes that provide basic modelling functionality, hence avoiding the need for low level programming
- VRML stands as a 3D geometry modelling interchange format [94].
- VRML is designed to fit into the existing infrastructure of the Internet and the WWW. VRML models can be visualised / run using simple web browser and the nodes' functionalities can be extended by embedding Java / JavaScript code
VRML therefore provides an ideal basis to test the feasibility of the concept of highly functional and fully portable PoCo modelling environments. As stated by Allen [14], VRML still lacks engineering specific functions and programming capabilities, which often result in VRML being used as a 3D modelling format [94]. However, it is believed that the potential of VRML is not fully exploited and part of this research therefore focused on assessing the extent to which the modelling and programming capabilities provided by VRML could support the realisation of a prototyping environment intended to support a highly specific engineering activity.

4-8 Chapter Overview

In this chapter, the background rationale for the research approach has been provided. In particular:

- **PoCo modelling components** were described as architectural constructs resulting from a specific approach to the decomposition of RMS virtual prototypes and VPE tools functions
- the types of generic functions that a PoCo modelling component should support were defined with respect to the architectural model defined above and to approach to component design developed in recent research and
- the choice of modelling and programming infrastructure needed to implement PoCo modelling components was justified with respect to the target PoCo VPE characteristic and performance in terms of portability

![Figure 4-19: PoCo VPE implementation approach](image_url)
The functions, and architectural characteristics of the software constructs from which the PoCo VPE tool and PoCo component-based virtual prototypes are composed therefore result from the combined perception of a component in i) the domain of system design and architecting (i.e. generic approach to component), ii) the domain of software engineering and architecting and iii) the specific domain of 3D modelling and VPE tool design. It is believed that the approach to modelling component design and realisation is unique since it is aimed at combining the characteristics of component as designed and realised by commercial VPE software developer (i.e. highly functional and highly specific modelling components), to the advantage of the approach to component design and realisation that characterise academic project (i.e. portability).
Chapter 5  PoCo modelling components design

5-1  Introduction

5-1.1  PoCo modelling component’s functions overview

The specification and conceptual design of a Portable Component-based Virtual Prototyping Environment (PoCo VPE) have been described in chapters 3 and 4. The prototyping environment is built upon a set of so-called PoCo modelling objects from which complete and fully functional Virtual Prototypes (VPs) can be composed. In Chapter 4, the architectural and functional characteristics of PoCo modelling components have been specified. PoCo modelling components are decomposed into PoCo modelling elements, which provide the basic modelling functions required to implement VPs. As shown in Figure 5-1, PoCo modelling components stands as highly functional software objects providing different types of functions, namely:

- **Modelling functions** related to the system virtual modelling. These functions are specific to the nature of the system being modelled. For mechatronics systems, modelling functions can generally be linked to three modelling domains, namely: 3D system geometry, the system kinematic layout, and the system logic or behaviour modelling.

- **Structural functions** related to the composition of modelling components into complete VPs. These functions are usually supported by the modelling software environment used to compose modelling components into complete VPs. In the case of the PoCo VPE, these functions are distributed amongst modelling components.

- **Interfacing functions** related to the implementation of interfaces and interactions mechanisms between i) the model and model user and ii) between the model and third party software environments (e.g. external events handling).
It should be noted that this Chapter is aimed at describing the functions of PoCo modelling elements, which have been implemented for the modelling mechatronic systems controlled via sequential logic. However, PoCo modelling components have been designed to provide modelling functions openness. This practically means that the generic component model developed in this research can be extended to support the modelling of different types of systems (e.g. modelling of continuous NC control instead of sequential machine logic). Such openness has been made possible by defining a PoCo modelling component internal structure, which is based on the composition of functional objects, referred to as PoCo modelling elements. The following sections aim at providing a detailed description of both PoCo modelling elements functions and PoCo modelling component structure.

5-1.2 PoCo modelling component structure overview

Modelling components and modelling elements represent different types of architectural constructs. Modelling elements (generically described in Figure 5-2 as DYN, COND, STA, VP, LP and AV) provide various types of functions from which fully functional modelling components are composed. Each element provides the functions corresponding to one functional domain, namely the modelling, structural and interfacing functions. Conversely, modelling components materialise structural constructs, which encapsulate and combine modelling elements' functions into reusable fully functional VP modelling constructs. Elements provide different types of functionalities whereas modelling components are constructs that allows those functions to be combined and encapsulated in a reusable modelling constructs, which level of granularity can be defined according to the re-usability needs.
As illustrated in Figure 5-2, PoCo modelling components are the result of what Albir [59] refers to as composition relationship between elements. The composition relationship implies that a component cannot exist independently of the elements it is composed from. Modelling components do not provide specific functions, but can be perceived as a shell that providing the structure to encapsulate elements. The aggregation relationship (Albir [59]) that describes the way elements are composed and the way component are composed implies that objects can exist independently of each other. Each modelling components may be different with respect to the number of elements it is composed from, and with respect to the configuration of those elements.

The decomposition of the overall VPs functionalities in modelling elements has resulted from the need to i) minimise the number of objects required to compose a model, which also minimises the amount of inter object communication (of importance because of the difficulties to orchestrating events within class hierarchies using the VRML event model). In addition, ii) this particular decomposition approach results from the grouping of modelling functions that are more likely to be reused together. Breaking down the modelling functions into more specific elements could have increased the modelling flexibility but would have resulted in a more tedious composition process. On the other hand integrating all of the functionality into one type of element would have simplified inter object communication and model composition, but at the expense of reduced modelling flexibility.

5-1.3 PoCo Modelling element overview

DYN (dynamic) elements
Components should be composed of at least one DYN element. DYN elements can be configured in three different ways: Dynamic DYN have both logical and kinematic behaviour and are used to model machine actuators. Static DYN element are used to model machine parts that have logic behaviour (i.e. logical state), but no kinematic parameters (e.g. sensors). Dynamic and Static DYN elements are associated with state elements (STA) and transition elements (COND). Neutral DYN elements are used to model structural machine parts (e.g. fixture plates, conveyor structures) and have no logic or kinematic behaviour (i.e. no STA or COND elements are associated and the DYN element kinematics attributes are left empty). It should be noted that components should contain at least one Neutral or one Static DYN element in order to provide at least one absolute element that can be used as reference for another DYN elements' transformation.

**STA (state) elements**
Component containing Dynamic or Static DYN elements should have two or more associated states (STA element). It should be noted that STA elements are effectively defined as sub classes of DYN elements, and provide complementary information such as the position associated to each state in the case of Dynamic DYN elements.

**AV (action / visualisation) elements**
A component can encapsulate one or more AV elements, which are effectively sub classes of the STA elements. Each STA element should be associated with at least one AV element, which provides the graphical interface enabling the user to visualise a trigger to a change of state.

**COND (condition) elements**
Components containing Dynamic or Static DYN elements should be associated with two or more COND elements. COND elements are sub-classes of DYN elements and provide the cinematic parameters such as the time taken to go from one position (corresponding to a logical state and represented by a STA element) to another. COND elements are the elements that trigger the DYN part-motion display mechanisms.

**VP (viewpoint) elements**
At least one LP element should be associated with each DYN or LP element (LP elements are defined as sub-class of DYN and LP element in the PoCo component model), so that model actuator and assembly link point can be located easily (either manually or automatically).

**LP (link point) elements**
LP elements provide essential functions that relate to the geometrical composition of two components. The functionality of VP element is complex and involves the management of multiple Reference Co-ordinate Systems (RCS) associated with the composition of DYN, and PoCo components, and the transformations enabling model geometry and parent child relationships to be managed. VPs are sub classes of DYN elements. DYN elements may or may not be associated with a LP; however, a component should encapsulate at least one LP element. Note that VP elements are defined as sub-classes of component components but act at the component level i.e. they participate in the composition of modelling components, which means that when composed a component is defined as a sub-class of VP elements.

5-2 PoCo modelling elements description

5-2.1 Modelling functions

Manufacturing Systems (MS) for the automotive industry generally take the form of large scale and complex mechatronics systems. The term mechatronics underlines the dual aspect of such systems which results from the combination of a mechanical layout (defining the physical states that the system can potentially reach), and of an electronic and software tier (defining the sequential logical behind the progression of the states that the system can reach) [11]

![Figure 5-3: General system descriptive aspects](image)

The generic description of mechatronics systems is schematically illustrated in Figure 5-3. A complete model of a mechatronics system can be implemented by focusing on i) the modelling the geometrical characteristics and spatial layout of the system, ii) the logic underlying the system behaviour, and iii) the time dependant, or cinematic system characteristics. In the following paragraphs the different types of PoCo modelling elements, which have been implemented to support the modelling of sequential logic driven mechatronics systems, are described.
5-2.1.1 Data types (modelling domains)

5-2.1.1.1 Geometric modelling
MSs are complex systems typically composed of a large number of individual actuators, sensors and various structural parts combined into a specific machine mechanical layout. The characteristics of individual mechanical parts, and the way those parts are assembled into a specific layout confers particular functional characteristics to the machine. The mechanical design of MSs is often used as a starting point to initiate the other domains of design (cf. Chapter 3 for details of Cross Hûller Machine design process). Three-dimensional (3D) modelling primarily aims at reproducing the geometrical and dimensional aspects of machine parts as well as their relative position in space, which define the overall machine layout.

5-2.1.1.2 Kinematics modelling
The functionality of mechanical systems relies on the characteristics and layout of kinematic joints that link machine parts. Examples of mechanical systems which functions typically depends on specific types of kinematic layout are industrial robots, (e.g. Cartesian, spherical, cylindrical, SCARA type), or multi-axis machining centres. Defining kinematic links between the mechanical parts that compose MSs consists in suppressing some degrees of freedom in order to define the potential motion a part can achieve relatively to another. Several types of basic kinematic joints (e.g. linear, spherical, punctual, etc.) can be defined depending on the number and nature of relative degrees of freedoms between two parts.

5-2.1.1.3 Logic modelling
The design of a MS logic control is, along the mechanical design, a fundamental part in the machine design lifecycle. The mechanical and behavioural design processes are tightly coupled and need to be conducted concurrently in order to avoid design inconsistencies. Therefore, machine logic modelling and simulation are essential functions that VPEs should provide. The following paragraphs are aimed at introducing the various types of PoCo modelling elements designed and implemented so far to enable the modelling of MS mechatronics systems.

5-2.1.2 DYN (DYNamic) modelling element
The functions of the PoCo DYN element are focused on the modelling of mechanical system kinematic joints. It also provides the interpolation and display functions, which allow a 3D geometry associated with a kinematics link to be dynamically displayed according to pre-defined cinematic parameters (e.g. translation of X mm in n second). The DYN element is therefore used to model single, or a combination of kinematic links describing a part of the
machine mechanical layout. The main types of parameters that used configure DYN elements are:

- A name user defined name
- A string parameter that specifies whether the DYN element is used to model a rotational or translational kinematic link. It should be noted that several DYN elements can be composed to describe complex kinematic joints
- An set of x,y,z co-ordinates defining an axis in 3D modelling space, and in the case of a rotation, the x,y,z, co-ordinate of the centre of rotation. These parameters are required to define a translation or rotation transformation between the DYN Reference Co-ordinate Systems (RCS) to which the transformation is applied, and an absolute (fixed) RCS that represents the absolute space (cf. Figure 5-5)
- A URL link pointing to a file containing the VRML code describing a 3D geometry, along with the position and orientation of this geometry with respect to the absolute RCS

![Figure 5-4: PoCo DYN modelling elements configuration parameters (attributes) functions (methods) and interface](image)

DYN elements therefore provide the modelling functions required to model transformations that can be applied to a RCS (to which a 3D geometry is attached) positioned in a virtual space. DYN elements can also be used to model static machine parts by leaving the kinematics configuration parameters void. The modelling of machine actuator or group of actuators can be achieved by composing (relative positioning and orientation in the absolute RCS) static and dynamic DYN elements, configured with various geometry and kinematics parameters values.
Figure 5-5 shows the PoCo model of a pusher actuator used to model a supermarket distribution (i.e. Asda) conveyor line. This modelling component is composed from two DYN elements, the pusher base and the pusher actuator. The Pusher base is configured as a fixed DYN (no kinematic parameters defined). The Pusher actuator's kinematic is defined as a translation along one the absolute RCS's axis. Motion in 3D virtual modelling, is absolute and not relative, which means that unlike a real actuator, the transformation has to be associated with a specific part of the model.

It should also be noted that the 3D geometrical modelling data is not directly part of the DYN elements code, but is defined as a URL location (string parameter) of a file containing the VRML code describing a 3D geometry (typically contains the definition of surface and vertex that realise the machine shape). This allows the 3D geometry modelling code and the code describing the DYN's kinematics and dynamic display functions, to be dissociated and therefore more re-configurable and reusable. For instance, the 3D shape of a drilling head can be changed (e.g. different tools), by simply re-configuring the DYN's element geometry parameter, so that the modelling elements code need not be modified. Finally, as shown in Figure 5-4, DYN elements can interact with external environment (user, another elements or components, external software inputs/outputs) via interfaces that define the nature and format of events that can be received and/or sent (e.g. Go_from_to, State_Update).

The main function of DYN elements is to support the dynamic display of 3D geometry according to their kinematic configuration parameters. A set of internal functions allow the
transformation to be applied dynamically in response to incoming events. These events can be of three basic types, namely: "go from current position to position X", "if in position X go in position Y", or "go to position X". These different DYN operating modes can be triggered by i) other elements that emulate machine logic (i.e. COND (conditions interlocks) and STA (logic state) elements described later in this Chapter), or ii) in response to external user or software inputs.

Figure 5-6: Screen shot of DYN information display functions output.

DYN elements can generate events during dynamic display sequence or when it is been completed. These mechanisms consist of emulating position sensors so that the position corresponding to a logical state can be monitored. Such events can be used in combination with other machine modelling elements to implement the overall component behaviour or to synchronise the PoCo model with an external machine simulation engine. Finally, as shown in Figure 5-6, monitoring and information display functions enable the user to gather information about the current DYN element's state and configuration parameters (e.g. array of state names, and corresponding position). In addition some 'sensing' mechanism have been implemented, which allow to gather the 3D position and normal vector co-ordinates of a point clicked on the DYN elements surface geometry. Such function is used to configure parameters of other modelling elements (i.e. for Link Points: LP pos, LP norm, VP pos, VP ori).

5-2.1.3 STA (STAtE) modelling element

The STA modelling element has been designed to support simultaneously the modelling of both the logical (i.e. sequential discrete logic control) and physical state in which a DYN element can be. From a logical perspective a machine part can be described by one of many states (e.g. waiting, in_position, On, OFF, etc.) whether the type of machine part being considered is an actuator, a sensor, or a purely logical component (i.e. not materialised by any physical representation, e.g. part counters, etc.). A logical state (as represented by a STA
element) is defined as any static state that machine actuator or sensor can reach. The term “static” places emphasis on the fact that transitional states are not considered as machine states. For instance, an actuator moving from position A to position B is defined as in a dynamic state.

As shown in Figure 5-7, the STA element main configuration parameters are a user-defined name, a “state_pos” parameter, which holds an integer value defining a physical position for the DYN element to which the STA is linked. This field can be blank if the STA represents a purely logical state (e.g. sensor ON/OFF). The “Is_active” field can be set to true whether the DYN element reaches or leaves the position defined by one of its STA elements. This is done via the STA interface which defines a “state_update” (state i) event in.

### 5-2.1.4 COND (Logic condition or state transition element)

The COND modelling element enables a user to define simultaneously both the logical conditions associated with a transition between static states and the cinematic parameters that characterise the type of transition between two positions (e.g. speed or time taken). From a machine logic perspective, state transitions are defined by a set of logical conditions, which need to be true for a machine actuator to change state (leave a state and start the transition to another one). It should be noted that defining transition as shown in Figure 5-8, also constrains the possible sequence of state. For instance, the system can go from “state a” to “state b” but not directly from “state a” to “state c”.

---

5-108
The COND modelling element is characterised by a user name, a "state_name" and a "From_to_trans" attribute. The "From_to_trans" field holds the names of two states as defined in STA elements' "State_name" attribute's field. The state transition logical expression is held by the "AND_cond_array" attribute that contains an array of states' names, and a value for the "Is_active" field of these state. The COND element therefore defines the conditions associated with the transition of a DYN element between two states (two STA elements), as well as the time taken by the state transition (i.e. "Trans_time" parameter).

The states contained in the "AND_cond_array" are linked by a AND relationship meaning that all states defined need to be true in order for the condition to be valid. An OR logical expression can be readily achieved by associating two or more COND elements with the same "From_to" attribute value to the same DYN element. COND elements communicate with other elements via an interface that defines the input and output events. For example an output event "Go_from_to" is used to signal to other elements that the conditions associated with a state transition have become true and an input event "State_updates" is used to inform COND elements of other DYN elements' change of state so that the COND element's internal "Check_cond" functions can be triggered.

5-2.2 Structural functions

Structural functions have been defined as the functions that support the configuration and composition of MS VPs from PoCo modelling components. The approach adopted in the design and implementation of the PoCo VPE has consisted of encapsulating the functions that enable component composition within the components themselves. The composition mechanisms required to implement a complete component-based machine model are of two main types:

---

1 Please refer to Chapter 4 for an introduction to the PoCo VPE modelling elements functions.
- The geometrical and kinematic composition of PoCo modelling components. From a modelling perspective, the geometrical and kinematic model assembly process consists in recreating a model tree structure describing the parent child relationships between modelling components geometry, so that the kinematic consistency of the overall model can be preserved. For instance, if a drilling head actuator is linked to an X-axis linear actuator, any X-motion from the linear actuator should be propagated to the drilling head. Whereas, in the case of a real machine implementation this can appear to be trivial, the fact that transformations defined in a virtual world rely on the manipulation of reference co-ordinate systems, obtaining a consistent model of complex kinematic system layout requires a careful management of the parent child relationships between model parts (i.e. model tree structure).

- Composition related to the model behaviour definition. The sequential logic describing the behaviour of a complete manufacturing system results from the definition of i) the states in which each machine actuator or sensor can be, and ii) of the conditions, also referred to as machine interlocks, that trigger machine actuators and sensors state transitions. Because a complete PoCo model is broken down into individual components (with individual states and interlock conditions), it is necessary to define and implement the event routing paths that allow modelling components that are linked by interlock conditions, to communicate.

The two types of composition (i.e. geometric/kinematic and logic) are supported by two different types of elements which are the Link Point elements, and the Interlock elements.

**5-2.2.1 LP (Link Point) modelling element**

Modular machines are characterised by standard mechanical interfaces that can be used to assemble quickly and easily machine components in various configurations quickly and easily. This is typically the case for the conveyor mechanism, which is part of the Krauser Test Rig used in this research as a case study\(^1\). LP elements have been designed as virtual equivalents of these standard modular machine interfaces. The LP elements' configuration parameters support the definition of an assembly position that can be used to assemble two modelling components. LP elements consist of the 3D co-ordinates of a point and of a normal vector to a surface at this point. The LP assembly point and vector co-ordinates of two LP elements can therefore be used to compute the transformation required to match the points

\(^1\) The cases studies, which have been used throughout this research, are presented in Chapter 8.
and the vectors directions. The Reference Co-ordinate Systems (RCS) management mechanism and transformation sequences that allow modelling components to be geometrically composed and the tree structure to be effectively managed, are briefly illustrated in Figure 5-9.

The cube shown in Figure 5-9 represents a DYN element (which is part of a component). The parameters of the cube element define i) a motion (greyed arrow), and ii) a LP element (characterised by a point and an associated 3D reference co-ordinate system). This LP element can be associated with the LP attached of another DYN element, by computing a 3D transformation resulting from the combination of a rotation and translation so that both LP positions and vectors are coincident. The composition mechanisms make use of the points and vectors’ co-ordinates expressed in either the absolute RCS or in the DYN RCS, in order to maintain the kinematic consistency of the final geometrical assembly.

The fundamental concept behind the LP modelling element is to implement the functions required to compute RCS transformations (necessary to enable composition of modelling components), directly as part of the element’s. It is therefore possible to reduce the process of composing modelling components to a simple task that consists in configuring LP parameters. Practically, this consists in defining a component instance as a parameters of an LP element’s sub class (cf. [COMP] sub class in Figure 5-10) and by setting the “Iam_child” parameter of one of the link point of this component to “true” (cf. Figure 5-10).
This approach enables the composition of PoCo modelling components to be dissociated from specific modelling software functions. Software providing a user interface to guide the user though the configuration of LP parameters (and therefore through the component composition process) can be implemented easily, since the core functions required to achieve geometrical composition are provided by LP elements. The functions provided by the PoCo components configuration software (interface) can therefore be simple and hence portable, and can potentially be designed and implemented according to the end users’ preferences in term of programming languages, interface layout, or constraints imposed on the modelling process (i.e. access rights, modelling procedure).

Each LP element is characterised by a user-defined name, a link point position and normal vector co-ordinates (expressed in the RCS attached to the DYN element), a value to fine tune the assembly angle in a plan perpendicular to the normal vectors (Adj_angle), and some display parameters used to set the size of the sphere and axis used to represent link points elements (see Figure 5-10). The assembly mechanisms relies on the intercommunication between the LP elements, the DYN element to which the LP are attached, and the PoCo modelling component, which DYN and LP elements are part of. The assembly sequence is described later in this Chapter.

5-2.2.2 INT (INTerlocking) modelling elements

The INT element type is the only PoCo modelling element that could not be implemented using exclusively VRML code and embedded JavaScript code. The INT element was initially specified to support the automatic implementation of event routing paths between COND and STA elements (encapsulated within different modelling components), which states and transition conditions were linked. This mechanism was implemented in order to avoid broadcasting component states changes to all of the components in the model, which could saturate the relatively limited VRML event model capabilities and result in poor real time
performances when conducting simulation of large models. The design of the INT element functionality relies on the fact that the expression of state transition condition contains the naming information of the components (and embedded COND and STA elements), which state appear in the interlock condition.

![Component Actuator STA element](image)

For example, a simplified representation of two PoCo modelling components is provided in Figure 5-11. One component represents a Pusher actuator and has two states (retracted and extended) modelled by two STA elements, and two set of conditions modelled by COND elements (Sensor/Sens_part/On, and Sensor/Sens_part/Off). The expressions for the logical condition which need to be validated for the actuator to go from a stated “retracted” to a state “extended”, contains the information required to isolate the modelling component and DYN elements whose state change needs to be monitored. The INT element was therefore designed to parse all condition of all COND elements within a modelling component, and to generate the event routing paths allowing “state_update” events from other components to be monitored. Therefore, it is possible to build or modify model logic and event routing paths dynamically by simply changing / configuring COND elements’ interlock condition parameters.

Java Script for VRML does not provide built in functions that allow deep hierarchy of object to be parsed, and it is therefore not possible to locate a VRML node by its name in a hierarchy of nested VRML proto nodes. Beeson [57] has proposed a VRML parsing mechanism based on a structurally flat repository containing copies of all nodes contained in the initial hierarchy. This approach allows parsing mechanisms to be implemented using simple VRML Scripting languages. However, such approach also requires a central repository of all VRML nodes that compose a model that every element can access in order to retrieve VRML nodes definition and parameters. The use of a central data repository was not consistent with the
Manufacturing Systems Integration Research Institute, Loughborough University

distributed approach to system design adopted in this research, which seek to embed every function within the modelling component themselves, and to minimise the amount of external services (not distributed) required for these components to operate.

A temporary solution was to implement the functions designed for the INT element as a Java class object that could extend the Java Script functions embedded within the INT element's VRML code. Another solution (which was chosen and implemented) consisted in implementing the INT element functions as part of an external software environment providing parsing services that component could access. Both approaches are in contradiction with the approach adopted in this research, namely to design and implement modelling components as highly portable and autonomous code objects. Exporting functions out of the component boundaries critically reduces the portability of PoCo models since external modelling software functions are required. Seeking an exclusively VRML based solution to the implementation of the INT element's functions, is at this time, part of the tasks planned for future PoCo VPE developments.

5-2.3 Model utilisation and user interfaces

PoCo modelling elements have been designed to encapsulate the interfaces enabling model/user interactions, so that there is no need for external software interface to control or interact with the model. The set of model/user interfacing functions that have been implemented within the PoCo elements relate to:

- The model navigation: Navigation within 3D space is known to be problematical due to the inadequacy of 2D controls (e.g. screen buttons) to navigate within 3D worlds. PoCo model interfaces are aimed to extend the navigation functions provided by VRML model viewing environment
- the implementation of interfaces allowing the user to visualise and interact with elements’s internal functions, so that the user can drive the model behaviour and therefore assess and debug alternative machine control configurations

The encapsulation of model/user interfacing functions within each modelling component is aimed at reducing the need for external software interface, and therefore is aimed at increasing the portability of the model visualisation/simulation environment.

1 Please refer to the Chapter 9 for more details on the future development of this research.
5.2.3.1 VP (view Point) modelling element

The virtual prototyping of MSs often results in large-scale 3D models, which are not fully appreciated due to the limited capabilities of current computer displays. This implies that mechanisms that allow the user to focus on a particular aspect of the model, or to view the model at a particular level of detail, need to be implemented (e.g. zoom in/out, navigation, viewpoint selection, etc.). Classic computer based 3D model display environments such as VRML web browser plug-in or CAD interfaces employ on screen control to provide the user with rotational, translational, or zooming capabilities. This navigation model, sometimes referred to as flying carpet model [107], assumes a camera positioned and oriented with respect to a base Reference Co-ordinate System. However, the navigation within a 3D scene using 2D screen controls (e.g. button, etc) can prove to be extremely difficult and disorientating especially in the case of large-scale models. Navigation is the most basic level of interaction between user and 3D model [106], but also represents one of the most challenging to implement [11].

The VP element’s functions are based on the viewpoint node that the VRML language provides in order to compensate for the limited navigational controls that characterise most VRML viewers. The VP element implemented in this research extends the VRML viewpoint feature’s functions. Typically, if part of the model is modified or moved the viewpoint parameters (position and orientation) need to be updated with respect to the change of position/orientation of the part they point at. In the PoCo modelling framework, VP elements are defined as sub class other elements (DYN, LP), the viewpoint parameters (orientation position) are expressed within the reference co-ordinate system (RCS) in which those elements are defined. Therefore, VP parameters do not need to be re-configured when component are reused or re-composed, hence making the implementation and management of navigation mechanisms for large models completely automated.

![Figure 5-12: VP modelling element attributes and interface.](image)

The attributes characterising the PoCo VP element’s parameters are a user name, a position and orientation co-ordinates (obtained by clicking on a LP of DYN 3D geometry (cf.
paragraph 5.2.1.2 Figure 5-6), and a description (e.g. component\_A\_View1\_top) (see figure 5.12). The VP communication interface allows VP elements to be activated by triggering the “activate” function, in which case the user viewpoint is directly set to the pre-defined position and orientation. The VP “activate” function can be used to locate quickly a DYN (or LP) element in the machine model. This function can be accessed through a web browser VRML plug in interface (e.g. Parallelgraphics Cortona, Cosmo, BlaxxSun Contact viewers), or it can be triggered by an external application to direct automatically the user to a faulty actuator for example. This function has been used in conjunction with the Human Machine Interface (HMI), and Process Definition Editor Tools, to provide effective error reporting mechanism and model analysis functionalities (e.g. fast location of faulty actuators, navigation through various machine part which logic are tested)

5.2.3.2 AV modelling element

The AV (Access/Visualisation) modelling element has been designed as a generic model/user interface that can be used as i) a visual representation (e.g. change of colour or shape associated with the AV element) for state change, or ii) as a clickable user interface allowing events / model functions (e.g. part motion, state change, information display) to be triggered. As shown in Figure 5-13, AV elements are characterised by a user defined name (AV\_name) and a 3D geometry defined by a parameter (Shape\_URL) containing the URL path pointing to a file that contains a VRML geometry of VRML 3D text. This geometry is positioned and oriented relatively to the STA element (and hence the DYN element) that contains the AV element. Colour and transparency (AV\_col and AV\_transp) parameters can be defined to indicate the change in the AV elements appearance.

The AV element is mainly used to represent logical state associated to a component (AV defined as a sub class of STA elements) so that the user can visualise or trigger a state change. However, the AV element’s generic functions can potentially be extended to other uses, one of which is in the display of the status of part detection sensors (a part is displayed when a sensor is on). The PoCo component model defines the AV control interface as a sub class of
STA elements, so that when a state based logic description and simulation engine is implemented (i.e. composition and configuration of COND and STA elements) the model logic/user interface is automatically generated.

5-3 PoCo Component model

The various PoCo modelling elements and elements functions are composed according to the PoCo component model, which provide a class hierarchy and events passing model that allow elements and elements’ functions to be composed into fully functional modelling components.

5-3.1 Element class hierarchy

The object-oriented (OO) concept of a class hierarchy and the corresponding UML representations have been used to structure the VMRL code describing PoCo modelling objects in order to ensure code re-usability, re-configurability and manageability.

Figure 5-14: PoCo modelling framework element Class hierarchy
Class hierarchies are used to describe the relationships between PoCo modelling elements and their composition into components. The aggregation and cardinality relationships between PoCo modelling elements from which modelling components are composed is illustrated in Figure 5-14. Depending on the type of system being modelling the number and configuration of elements that compose a component may vary. Nevertheless, the component model described as a hierarchy of elements remains the same. The main interfaces, which enable modelling elements to interact, are also illustrated in Figure 5-14.

5-3.2 Initialisation sequences

The first sequence of events occurs when the model is loaded into memory and consists of an initialisation sequence. The VP or DYN elements have more complex initialisation sequences than other elements, which consist of setting internal parameters.

5-3.2.1 Event routing paths generation

The first step of any PoCo element initialisation sequence is the automatic generation of event routing paths (using JavaScript CreateRoute functions). Event routing paths indicate the flow of events (and hence control) to and from all elements located lower in the (sub-classes). Each parent element therefore generates the event routing paths to and from their child elements.

5-3.2.2 Naming sequence

The naming sequence is initiated by the PoCo modelling components, which send an event to the first sub class in the hierarchy, i.e. the DYN element. The DYN element gathers the component’s name (string attribute) and attaches its own name to it (concatenation of string variable type). Each element that has completed its naming sequence sends an event to all its child elements, which in turn concatenate the received string with their own name.

![Diagram of event routing paths and naming sequence](image_url)
In this way, it is possible to define uniquely the name of each element in a model, which is essential to ensure VRML code integrity. Although two elements belonging to different components can have the same name, all components in a model need to be allocated different names.

5-3.2.3 LP initialisation

The DYN element implements a function, which returns to the user the 3D co-ordinates of a point clicked on the 3D geometry and the vector normal to the surface at this point. This mechanism is used to provide LP parameters’ editing functions and define interfaces that are integrated within each component. The LP position and normal vector co-ordinates that are returned to the user are expressed in the Reference co-ordinate System (RCS) associated with the DYN geometry. However, the geometrical composition of component requires the LP position and normal co-ordinate (of the child LP element) to be expressed in the absolute RCS (refer to Figure 5-9 paragraph 5-2.2.1 for more details on the assembly transformation algorithms). Part of the initialisation sequence therefore consists in requesting the position and orientation of the DYN and updating the LP position and normal co-ordinates, expressed in the absolute RCS.

![Diagram of LP element initialisation sequence]

**Figure 5-16: LP element initialisation sequence**

5-3.2.4 DYN element initialisation sequence

The DYN element initialisation sequence consists of setting the elements position and state in the initial state as defined by the DYN’s Init_state attribute (cf. Figure 5-4 paragraph 5-2.1.2).
A "State-update" event out is sent during the initialisation sequence that allow each element to set into its initial state.

5-3.3 PoCo distributed Logic engine's elements interaction

5-3.3.1 Automatic mode

The information and functions required to emulate machine behaviour are distributed amongst three PoCo elements namely the DYN, COND and STA elements. An example of the internal sequences of events between modelling elements is shown in Figure 5-17. Within this example, a DYN element is associated with three STA (1, 2 and 3) and three COND (1-2, 2-3, 3-1). The DYN element's parameters define an axis of translation between the 3D points 0,0,0 and 3,3,1 and the initial state is state 2. The top part of Figure 5-17 provides an example of the position associated with each state, and the transition times associated with each condition (COND element parameter). For the purpose of this description, one of the condition's logical expression has been defined as true, i.e. if A and B are true. A and B may be expression describing the state of other DYN elements being part of other components e.g. "CompNameC_DynNameI in state N".
Assuming that none of the logical expressions that compose the AND statement associated with COND 2-3 are initially true, any state change event from the ‘CompNameC_DynNameI’ modelling object is passed to the COND 2-3 element. In the COND 2-3 table, all the conditions related to the ‘CompNameC_DynNameI’ state are updated. The table is updated until all conditions are true simultaneously. A ‘Go-from-to’ event is sent from the COND element to the parent DYN element, along with a value defining the transition time (cf. paragraph 5-2.1.4). The DYN element’s functions check that the “from” state is its active state by interrogating the STA element which name match the “from” state name attribute. If this is the case, then the position associated with the “to” state is retrieved and used to extrapolate the display parameters. The DYN element’s functions then trigger the dynamic motion of its 3D geometry attributes and set the state corresponding to the position it is leaving to “not active”. The DYN element’s functions implement a position monitoring
mechanism that checks at every time interval dt (defined by the VRML clock) the relative position of the RCS with respect to the target position. During this time, the DYN element continuously (every dt) generates a "Position_update" event out, which can be used as input to others analysis or monitoring software applications. When the targeted position is reached, the DYN element outputs a "State_update" event to the external environment that can be used by other modelling objects to update their internal states.

### 5-3.3.2 Model internal logic override

PoCo modelling elements have been designed to provide a portable and distributed logic engine built upon various types of core modelling elements. This allows a PoCo model to be run, tested and re-configured independently of any other environments hence providing a high level of portability. However, it was essential to enable PoCo VPs to be used in conjunction with an external logic engine (i.e. logic editing environment simulation engine, or real machine state broadcasts). The DYN element therefore implements a "manual_drive" mode that enables the internal logic engine mechanisms and functions to be overridden. In this mode, events generated by the COND elements are ignored by DYN elements. However, "State_update" output events are still generated by DYN elements, so that other DYN elements set to automatic model can be run.

![DYN Element Configuration](image)

Because each DYN element composed into a machine model, can be set up in manual mode independently of each other, it is possible to implement a machine model that runs partially on an internal logic definition and partially on external events. This enables hybrid models representing a partially completed real system and a virtual system to be implemented. Such models can be used to support the re-configuration phases of a manufacturing lifecycle, where the logic of the "To Be" part of the system can be tested with the current logic of the "As Is"
This concept, termed "hybrid machine simulation", is shown schematically in Figure 5-18. Only a sub-set of DYN elements is set up in auto mode (black squares indicate the manual mode setting of DYN elements).

5-3.4 PoCo Link Point (LP) Element and assembly sequence

The LP modelling element provides the functions required to achieve the geometrical assembly of two modelling components. The sequence and nature of information exchanged between LP elements is illustrated in Figure 5-19; two LP elements (LP1 and LP2) are associated each with a DYN element materialised by a cube (Comp1) and cylinder (Comp2) geometries. In this example, Comp2 is defined as being a child object of the Link Point LP1. When the model is loaded, LP elements send their position and normal co-ordinates to the child components.

Upon receiving the position and normal co-ordinate broadcast, the "Iam_child" attribute of each LP is checked. Each LP for which the "Iam_child" attribute is set to "true", is used in the composition (to be matched with the parent LP). The transformation required to match the link points is computed by the LP element functions and applied with respect to the
Component’s Reference Co-ordinate System of the, which encapsulates the DYN element to which the LP is attached. The result is: i) two components with relative positioning and orientation such that the overall model geometry reflects the assembly of two machine parts linked at a fixture point and ii) the Comp2 RCS, is referenced as a child of the LP1 RCS parent in the tree structure which describes the model kinematics. Therefore, the Comp2 will automatically inherit all of the transformations applied to the DYN element to which the LP1 element is attached.

5.3.5 Example of PoCo model implementation

An overview of how a PoCo machine prototype can be composed from PoCo modelling objects is presented in this section. The example is generated using modelling components initially constructed for a prototype of a conveying machine at Asda (one of the case studies used to conduct this research¹). The Asda test machine was based upon a modular architecture that enabled to assess i) the reusability and re-configurability of modelling components and ii) the ease with which components could be composed into various model configurations.

5.3.5.1 PoCo element configuration

The component editing phase consists of implementing reusable and re-configurable modelling components which can be instantiated configured and composed into complete VPs. The first phase of component editing consists of configuring and composing a set of modelling elements. Figure 5-20 illustrates the composition of various types of elements into a Pusher component used to sort and redirect boxes towards various conveying belts. The Pusher actuator is composed of two DYN elements. One DYN element is used to model the Pusher base, which represents the fixed part of the actuator. The DYN kinematics parameters are therefore left empty and the only parameters that are defined are i) a URL link pointing to the file that contains the pusher base 3D geometry description, ii) the position and orientation of the geometry in the DYN reference co-ordinate and iii) a generic name for the DYN element.

The second DYN element is used to model the pusher’s linear actuation. The DYN kinematic parameters are configured by defining an axis of translation in this case ‘1, 0, 0’, which defines a translation along the x-axis. It should be noted that the most difficult and error prone part of the component editing phase is the positioning and orientation of DYN elements’ geometry. The VRML format used to reference orientation of reference coordinate (x,y,z axis co-ordinate and a rotation angle) is very awkward to use, so that DYN elements implement

¹ More details on the cases studies used in the different phases this research are provided in chapter 8.
internal functions that allow the orientation of the geometry to be defined using Euler's transformation, which is more intuitive.

The next step of the element configuration phase is aimed at configuring the model logic control. This is achieved by configuring a STA element for each state that the actuator can reach. In the Asda machine configuration, pushers could only be in two distinctive states (namely “extended” and “retracted”). In addition to defining machine states, the logical expressions of the conditions that allow the actuator to change state were defined. In this case, only two COND elements were defined, which where the conditions associated with the transition from “retracted” to “extended” and from “extended” to “retracted”. This involved the configuration of the “From_to_trans” fields with the STA elements’ state names (cf. Figure 5-20). It should be noted that STA and COND elements are not represented by any 3D geometry.

Each state is associated with an AV element that provides a visual interface allowing the user to interact with the internal model logic engine. In the case of the Asda machine pusher
modelling, the two AV elements corresponding to the two actuators' states were represented by arrows. For each AV element, cosmetic parameters were defined which allowed the geometry to be displayed differently depending on the element's state. Finally, a Link Point was attached to the DYN Pusher_base geometry. The LP element's Position and Normal vector co-ordinates and display parameters (e.g. Adj_angle, Displ_size) were defined. It should be noted that the LP parameters might be obtained by clicking on a DYN element's geometry. Without this function, the editing of LPs would be practically impossible. Having integrated this function directly as part of the elements that compose components means that external modelling application are not required and that PoCo models can be modified and re-configured very easily.

5-3.5.2 PoCo component configuration

The generation of PoCo components consists of composing the constituent elements according to the PoCo component model (i.e. class hierarchy) described in paragraph 5-3.1. This is very easily achieved since the VRML PoCo modelling object model, allows element instances to be defined simply as attributes of other elements. The hierarchy of nested elements described by the component model is defined by the type of sub-elements that each element can have as attributes (or sub classes). In addition, each element's routing function automatically generates the event routing paths to and from its child and parent elements, which further simplifies the editing process.

<table>
<thead>
<tr>
<th>Pusher Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component_name</td>
</tr>
<tr>
<td>Pusher_axis_Init_state</td>
</tr>
<tr>
<td>Manual_drive</td>
</tr>
<tr>
<td>Pusher_axis_extended_position</td>
</tr>
<tr>
<td>Pusher_axis_retracted_position</td>
</tr>
<tr>
<td>Retracted_extended_AND_cond</td>
</tr>
<tr>
<td>RetractedExtended_time</td>
</tr>
<tr>
<td>Extended_Retracted_AND_cond</td>
</tr>
<tr>
<td>Extended_Retracted_time</td>
</tr>
<tr>
<td>LP_Pusher_Adj_angle</td>
</tr>
</tbody>
</table>

Figure 5-21: PoCo modelling component Class.

Figure 5-21 shows the UML class diagram for a Pusher Component that has inherited some of the parameters of the elements from which it is composed. Some of the internal component parameters can be hidden from the user. This enables access to the internal component parameters to be controlled. Different component interfaces can be presented to engineers from different domains by suppressing the data that are not relevant. Control engineers could
for instance be provided with components, which only exhibit control, related model parameters (e.g. state transition conditions) hiding geometry and kinematic configuration parameters (Note: the result of the combined engineering data would still be visualised through the model behaviour).

It should be noted that one problem encountered during the implementation of Asda machine model using the process described above is that once edited some of the component class characteristics cannot be modified. This is the case for instance of the number of state of link points, which cannot be easily modified once the component, has been edited. Doing so requires the component to be partially re-edited. A solution to this problem has been found in systematically defining additional LP, STA and COND elements at component design time which might be used or not, depending on the machine configuration. Although this approach is functional, it is not elegant and a solution should be found by either modifying the component editing process in order to allow easier modification, or by re-designing COND and STA elements to allow greater flexibility in the machine state and condition definition.

5-3.5.3 PoCo modelling component composition

PoCo modelling component, once edited, can be instantiated and configured to compose complete machine models. It is at this stage that the potential of the PoCo modelling approach is appreciated. PoCo modelling components can be easily configured and composed due to the internal element's design and automated composition mechanisms. Model behaviour and model geometrical composition can be modified by simply re-configuring components parameters.
Figure 5-22 shows the hierarchy of machines, sub-systems, components and elements that have been implemented (and composed) for the modelling of the Asda test machine. The set of component have been edited, re-used, and re-configured to test various machine mechanical layouts and control configuration. The PoCo model architecture has enabled a very high level of manageability, re-configurability and re-usability for the modelling constructs to be achieved considering that all of the model functionality, including an internal sequential logic simulation engine, geometrical composition mechanisms and model/user interfaces were implemented as part of modelling objects. The PoCo modelling environment therefore enables highly functional and portable models to be implemented without the need for specific knowledge or skills in the domain of modelling and more importantly without the need for complex modelling software environment.

5-4 Chapter overview

This chapter was aimed at providing a detailed description of i) the PoCo VPE modelling constructs and ii) the PoCo VPE modelling architecture. Detailed descriptions for each type of modelling element and the functions they provide have been given. In addition, the component model providing the elements’ integration structure and interaction model have been detailed.
A particular emphasis has been placed on describing the unique element functions that enable the component composition mechanisms to be supported by the modelling component themselves, hence providing highly autonomous objects supporting both i) the modelling related functions (i.e. kinematic, 3D geometric, model behaviour) and ii) the model management functions (i.e. geometrical composition, user interfacing). Some of the limitations in implementing highly functional modelling constructs using exclusively web-based technologies have also been highlighted (i.e. INT element realisation). Finally, an example of machine prototyping (i.e. Asda machine) using PoCo modelling objects was proposed, in order to highlight the highly structured and manageable modelling process allowed by the use of the PoCo VPE modelling framework.
Chapter 6   PoCo VRML object models

6-1 VRML and Object Orientation

The Virtual Reality Modelling Language (VRML) has been used in the context of this research to implement PoCo modelling objects (namely PoCo modelling elements and components). VRML has not been designed as a fully featured object oriented programming language, which is partially due to the will of the Web3D consortium [146] to keep VRML as simple as possible in order to preserve its portability and its value as an interchange format [94]. However, this approach limits the potential of VRML to be used for the implementation of engineering tools. Some academic projects [57] [96] [98] have focused on potential ways to impart VRML with the capability of object oriented programming in order to combine the simplicity and portability of VRML language with the code reusability and manageability that Object Oriented (OO) programming languages provide. This Chapter is aimed at highlighting to what extent VRML programming capabilities can be extended to match the capabilities of OO languages. Various approaches to object-oriented VRML are reviewed and the VRML OO model implemented in the context of this research, in order to provide PoCo modelling elements and component code reusability and manageability, is presented.

6-1.1 Review of Object Oriented paradigm

Prior to the OO approach, typical software applications were viewed as monolithic systems offering logical procedures for taking input data, processing it, and producing output data. This approach often resulted in complex systems designed to achieve a specific task. The internal structure was focussed at the level of granularity defined by the individual functions of the programming language used and therefore the system complexity increased exponentially with its size. Such systems were ultimately very difficult to debug, modify and maintain and provided very little code and function reusability.

The concept of object-orientation first appeared in the early 1970's with the development of the Smalltalk OO Computer language [100]. The term “object-oriented" is generally used to describe systems in which the software is organized into a collection of objects that incorporate both data structure and behaviours [103]. The Object Oriented (OO) approach to system analysis and design has revolutionised the development lifecycle of large scale and complex software systems [56]. The capabilities of OO programming to provide code manageability and reusability, is essential in assuring the long-term value of software systems [57]. The OO approach gained popularity and has been applied to the design and development
of various types of systems, such as database architectures [5], manufacturing system design [101] and shop floor control architectures [102].

6.1.1 Encapsulation

Among all the concepts usually used in the literature to define the OO paradigm, the concept of encapsulation is of prime importance [59]. Encapsulation has been supported and implemented in various ways by programming languages and OO systems since it is necessary to decompose large systems into smaller subsystems that can be more easily developed, maintained and reused [56]. Nierstrasz [56] states that all OO concepts depend ultimately to the concept of encapsulation, a direct consequence of this being that any programming language providing support for encapsulation can be considered (at least to some degree) as OO. The concept of encapsulation is related to the concept of system decomposition into well-defined system building blocks with internal structures that are clearly separated from the external environment.

6.1.1.2 Classes, instances and class hierarchies

The OO paradigm defines two types of "objects", namely the classes and the instances of classes (commonly referred as "object"). A class is a generic definition of a particular object. The analogy often used to describe classes is to compare them to blue prints from which particular instances (i.e. objects) can be produced. The implementation of an object from its class is commonly referred to in the OO terminology as instantiation.

The concept of class hierarchy extends the type of relationships that link classes and objects. A class hierarchy is a classification of relationships in which each item except the top one (known as the root) is a specialised form of the item above it. Each item can have one or more items below it in the hierarchy. The OO paradigm exploits the concepts of class to enable designers to manage the functional complexity of the systems to be designed [98]. Classes are defined by regrouping functions and attributes common to several objects, hence achieving system abstraction through the definition of a set of classes that cover the complete range of system functions.
Inheritance between classes provides support for the implementation and management of class hierarchies [102]. It is achieved by defining methods and attributes common to several classes and by encapsulating them into what is referred to as super classes (cf. Figure 6-1). Class hierarchies, can therefore be created, and child classes, or sub classes, inherit from their super classes. The OO approach defines several inheritance mechanisms that allow the class hierarchies describing complex systems to be more effectively managed. The most common inheritance mechanisms include single and multiple inheritances (see Figure 6-1), where a class can inherit from one or more super classes' attributes and methods. However, other inheritance mechanisms allow classes to add variables and methods to the ones they inherit from the superclass, which is commonly referred to as class override or polymorphism (see Figure 6-1).

Improved maintainability is a consequence of the class and class instantiation mechanism, since modifying the class results in modifying all objects instances that are in use in the system [101]. It is therefore easier to modify or develop new functionalities since only the code corresponding to the class is modified. Class hierarchies and classes' inheritance mechanisms therefore allow the number of classes required to describe all possible state a system can reach to be reduced. The most generic objects are represented by the upper (super) classes whereas the classes that are lower in the hierarchy characterise the functionality and attributes of specific objects.

6-1.2 VRML and object model

Classes and class instantiation are perhaps the most basic OO mechanism [59]. However, all programming languages inherently provide some built-in data types which can be instantiated as needed. In the same way, VRML provides pre-defined functions, which are referred to as nodes. Whereas the node classes define types of general attributes, the node instance defines values for those attributes defined by specific attributes name and value.

In order to preserve its simplicity and portability, as well as to make it accessible to non-programmer, VRML has not been implemented as a fully-fledged programming language [94]. For this reason, the current capability of VRML to be used as a programming platform to implement complex 3D modelling applications (e.g. to support manufacturing system prototyping) is limited [100]. Nevertheless, VRML provides many advantages regarding model portability and usability, which are some of the main objectives of this research.
However, the scale and behavioural complexity of manufacturing systems automatically translates into modelling code complexity, and the lack of support for OO mechanism and code management can result in models being very difficult to maintain, modify, and a fortiori re-use. It was therefore essential to assess the potential of VRML in supporting OO mechanisms to guarantee the reusability and manageability of the VRML code describing the modelling components.

6-1.3 VRML Proto node

Pre-defined VRML nodes can be considered as “built-in” classes that can be instantiated to compose VRML scene. However, a requisite of OO languages is to enable the user to define “customised” classes or objects [56]. This is necessary in order to maintain the advantages introduced by the OO approach regardless of the complexity or level of abstraction at which the system is considered. Classes are defined by the concept of encapsulation [56]. Encapsulation aims at packing [56] or hiding part of a system [98] (i.e. attributes and methods) into an object (i.e. class) that can be reused and instantiated. For instance, a large VRML scene with complex behaviours might be materialised by hundreds of interacting VRML nodes. In such cases the level of granularity defined by VRML nodes does not help in managing the overall modelling code complexity. It is therefore necessary to define higher-level objects (i.e. at a higher level of abstraction) which encapsulate some of the system complexity.

VRML 2.0 has been implemented with the properties of “composability” and “reusability” in mind [94]. The VRML PROTO node type allows any VRML code to be encapsulated into a reusable user defined VRML class. The capabilities provided by this node have been defined by [94] as “a convenient mechanism that allows geometry and/or behaviour to be packaged in an easy-to-use way” or also as “a method of defining a library of reusable objects” [95]. As highlighted by other researchers [57] [96] [98], the VRML PROTO node is the unique feature which has allowed OO code management mechanisms to be emulated.
Note: the level of granularity mentioned above refers to the level at which the system (function or structure) is considered. To illustrate this concept VRML has been placed in Figure 6-2 along an axis that represents the level of granularity at which the modelling code is considered. VRML nodes encapsulate lower modelling objects (which belong to the Open Inventor library in the case of VRML). In the same way, the Open Inventor library’s objects encapsulate the complexity associated with the interfacing between software and hardware (often referred to as hardware acceleration supported by OpenGL code in the case of VRML). On the other end of the scale, VRML proto nodes represent modelling objects whose level of granularity is “higher” than the one provided by the VRML nodes.

6-1.3.1.1 Encapsulation and PROTO node structure

The VRML PROTO node consists of two parts. The first part is referred to as “PROTO definition” and consists of a classic VRML nodes and scene graph. The PROTO definition is no different from any other VRML scene description and can be of any size and contain any type of VRML nodes, including Script nodes. The second part of the PROTO node is referred to as “PROTO declaration” and consists in a PROTO keyword followed by the name of the PROTO and by a set of VRML fields.

6-1.3.1.2 PROTO Instantiation

User defined PROTO nodes can be instantiated. A PROTO node can either be instantiated in the VRML file that contains its declaration and definition. In the case of a local instantiation, the PROTO object code just consists in a name for the PROTO instance and the definitions of specific values for each fields present in the PROTO declaration. A PROTO can also be
instantiated in a remote VRML file by using the EXTERNPROTO instantiation mechanisms. This allows a VRML scene to be implemented from instances of PROTO classes whose definitions are distributed over the internet. This VRML feature is extremely valuable since it provides a highly effective basis to implement distributed modelling environments.

```vrml
#VRML V2.0 utf8
PROTO Pusher {
    field SFVec3f Position 0 0 0
    eventIn SFFrime UserTurn
    eventOut SFFrime ProcessStarted

    DEF All Transform {translation IS Position children [
        DEF shape Transform {children [
            Shape {geometry Box
                size 1 0.2 0.2
            ]}
        ]}
    }

    DEF pos_R PositionInterpolator {key [0 1] keyValue []}

    DEF timer TimeSensor {cycleTime cycleInterval 1
        startTime IS UserTurn
    }

    DEF script-code Script {
        eventIn SFFrime ProcessStart
        eventOut SFFrime ProcessStarted IS ProcessStarted

        url "javascript:
            function ProcessStart (val){
                ProcessStarted = 1"
    }

    ROUTE timer. fraction_changed TO pos_R. set_fraction
    ROUTE pos_R. value_changed TO All. translation
    ROUTE timer. startTime TO script-code. ProcessStart
}
```

<table>
<thead>
<tr>
<th>Proto Instantiation Code</th>
<th>ExternProto Instantiation Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF Pusher1 Pusher {</td>
<td>EXTERNPROTO Pusher {</td>
</tr>
<tr>
<td>Position 1 0 0</td>
<td>field SFVec3f Position</td>
</tr>
<tr>
<td></td>
<td>eventIn SFFrime UserTurn</td>
</tr>
<tr>
<td></td>
<td>eventOut SFFrime ProcessStarted</td>
</tr>
<tr>
<td></td>
<td>} &quot;file:///Fl/rhesis/Writing 9/Proto Pusher.wrl#Pusher&quot;</td>
</tr>
<tr>
<td></td>
<td>10 DEF Pusher2 Pusher {</td>
</tr>
<tr>
<td></td>
<td>Position 5 0 0</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Figure 6-3: Example of VRML code encapsulation using VRML proto node, and example of Proto node instantiation (both Proto and ExternProto)

The PROTO and EXTERNPROTO node encapsulation and node instantiation therefore provide the basic mechanisms that enable the user to encapsulate potentially large and complex scene into reusable code objects, or proto classes. VRML fields (such as event, or variable fields) or attributes of those classes can be made accessible to the user or other VRML objects through the PROTO declaration. The VRML PROTO node type can in some extent be compared to the class / class instantiation OO model, since it supports encapsulation and code reuse though object instantiation.
6.1.4 VRML and inheritance

While encapsulation is a very general concept, many concepts such as class, class code reusability, and OO inheritance mechanisms such as multiple inheritance and polymorphism, can be related to some extent to the encapsulation mechanisms that a language provide. Diehl [96] focused on assessing the ability of VRML to reproduce basic OO concepts such as classes, object, and inheritance mechanisms (e.g. multiple inheritance and polymorphism). Diehl [96] describes the relationships between VRML programming features and Object Oriented concepts as follows:

- VRML Prototypes (PROTO nodes) are classes without inheritance
- Nodes of the scene graph are Objects
- Events and Script nodes are Methods
- Node fields are Variables

Despite the encapsulation capabilities provided by the PROTO node, VRML imposes some constraints on the extent to which code can be managed in an OO fashion. The next paragraphs discuss these limitations through the analysis of previous research on OO VRML. Finally, the approach adopted in this research to implement the PoCo VPE VMRL object model is discussed.

6.1.4.1 Class inheritance and hierarchy of nested PROTO

The OO paradigm supports two basic types of inheritance namely: i) the class-object inheritance and ii) the inheritance within a class hierarchy. The class-object inheritance mechanism implies than objects inherit methods and attributes from the class they are instantiated from. Inheritance mechanisms within class hierarchies are aimed at defining mechanisms that allow classes to be composed from other classes. The results of this type of inheritance are classes and not instances of classes. The VRML PROTO node provides the basic mechanism to achieve class-object (i.e. PROTO – PROTO instance) instantiation and inheritance. However, the VRML PROTO features do not allow PROTO classes to be composed. Instead, the only way to compose PROTO classes is to create instance of those classes, composes those instances, and create a new classes by encapsulating the composed PROTO instances into a new PROTO class.

The result of this is that in comparison with an OO programming language, an extra step (which requires code management tasks not currently supported by the VRML language) is required in order to compose abstract objects (i.e. classes and not instance of classes). It is possible to compose PROTO instances by using what has been referred to as nested PROTO
Hierarchies [57] [96]. Nested PROTOs cannot be instantiated as a normal VRML PROTO node. However, a new PROTO class can be defined by encapsulating the nested PROTO instances in a new PROTO class. Despite the fundamental limitations of VRML, several academic research projects [57] [96] [98] have focused on implementing VRML based mechanisms which allow some of the inheritance mechanisms described by the OO paradigm to be implemented. Two of the main research projects to implement VRML based OO mechanisms are presented below.

6-1.5 Approaches to VRML based Inheritance

6-1.5.1 Beeson’s Approach to OO VRML

VRML enable VRML Script nodes to be implemented, which be used to i) implement JavaScript functions to extend VRML nodes', or ii) to implement interface between VRML code and an external Java object. This capability of VRML has been exploited by Beeson [57] in his research on implementing a VRML scene in which “Newtonian” VRML objects interact with each other in the same way objects submitted to laws of Physics would do. Beeson’s research is aims at investigating the possibility of implementing an OO VRML structure that would allow modelling objects to be reused and modelling objects' capabilities to be easily extended and managed. Beeson’s research focuses on the means to define behavioural classes that would allow various Newtonian behaviours to be managed in an OO like class hierarchy.
Beeson's research has highlighted that one restriction in generating VRML class hierarchies, is the inadequacy of the VRML event model to support event communication and data flow between nested prototypes. This limitation has also been experienced during the implementation of the PoCo VPE object model and quickly brings the event passing mechanisms required to emulate inheritance between nested PROTOs, to a level of complexity that make the object code and functions difficult to develop or maintain.

For these reasons, Beeson [57] has used two different programming languages to implement modelling objects functions; the dynamic display and 3D modelling functions were implemented using VRML objects, whereas the objects' behaviours were implemented using Java code (external to the VRML file code). The Java programming language provides better support for object orientation (i.e. class abstraction and classes inheritance mechanisms) so that, the physical laws driving the object behaviours could be more easily described. Beeson eventually used External Authoring Interface (EAI) tools so that Java code could directly access and interact with nodes within the VRML scene graph. Hence the "difficulties of orchestrating execution and data flow between nodes" (i.e. the limitation of the VRML event model could be overcome. Beeson [57] found in the use of Java language the capability to implement clean and simple inheritance mechanisms without the need for complex mechanisms required to manage events passing over several level of a nested VRML proto hierarchy.

6.1.5.2 Diehl's approach to OO VMRL

Research on combining OO concepts and the VRML language has also been carried out by Diehl [96]. Diehl's goal was to implement class inheritance mechanisms using exclusively the VRML and VRML JavaScript functions. The approach adopted by Diehl is to implement a so-called "VRML++" syntax that extends the VRML node syntax. The VRML++ syntax is VRML based but is very similar to the Java syntax in the way it describes classes. VRML++ syntax is based on the use of the two keywords EXTENDS and SELF, which allow programmer to define inheritance relationships between classes in more or less the same way OO programming languages do. The VRML++ file is then processed by a "pre-processor" tool that translates the VRML++ syntax into VRML file capable of implementing VRML-based mechanisms that simulate OO inheritance. The resulting VRML file exhibits a hierarchy of nested PROTO nodes and some event passing mechanisms that enable "forward and backward communication" [50] between PROTOs within a nested hierarchy. The VRML++ code and pre-processor allows simple inheritance mechanisms such as simple class
inheritance to be reproduced in a VRML form. Diehl’s research also has developed VRML++ syntax and post processing algorithms that allow simplified forms of multiple inheritance and polymorphism to be reproduced.

6-1.5.3 Qualitative analysis of research presented above

6-1.5.3.1 Use of external language and model portability

As determined by Beeson [57] and Diehl [96], nesting PROTO is the only means by which OO inheritance can be reproduces using the VRML syntax. Diehl’s [57] research has highlighted the fact that complex inheritance mechanisms such as polymorphism can be implemented using exclusively the functionalities and programming features provided by the VRML language. However, as emphasised by Beeson [96], the implementation of such OO class hierarchy inheritance mechanisms results in complex VRML files’ structures, which can be a limiting factor for modification or further development of the corresponding VRML code. The approach chosen by Beeson is to avoid those difficulties by exploiting the ability of VRML to interface with OO languages such as Java. Conversely, Diehl’s approach is focused on making use of a VRML++ language and code processing algorithms, which allows the VRML file structure complexity related to the implementation of inheritance mechanisms to be hidden from the user.

The use of Java code results in composite modelling objects, which functionalities are distributed amongst various types of code and file locations [57]. This distribution affects model infrastructure, model portability and manageability and is more difficult for an observer or a developer to visualise or choose how Java implemented methods and VRML based modelling objects are mapped [57]. Beeson’s approach is contrary to the primary vision of this research, which is to maximise the portability and usability of both the model and modelling environment and to preserve the integrity of VRML modelling objects by making exclusively use of VRML and embedded Script.

6-1.5.3.2 Syntaxic layers

The approach adopted by Diehl is more similar to the approach adopted in this research to implement “OO VRML”. However, a major disadvantage of Diehl’s approach is that it defined an additional syntaxic layer above VRML code, which means that some of the advantages associated with the use of a standard format are lost. Because of its hybrid nature, VRML++ cannot be used in conjunction with VRML compliant tools (editing, parsing, and import / export of VRML code to and from other environment). In addition, the lack of documentation and guidelines for the use of VRML++ reduced its suitability for commercial
modelling purposes. From a theoretical point of view and regardless of the performances level and reliability of Diehl's tools, the translation from one format to another often raises some concerns regarding the automation of the process. Generally, format translation is source of potential data distortion and error injection. However, it should be noted that the VRML++ approach to OO VRML primarily aims at providing some suggestions for the specification of the next generation of the VRML language.

The use of the VRML++ syntax and code processor also imposes technical constraints. For instance, the capabilities provided by the VRML EXTERNPROTO feature to instantiate VRML proto nodes in a remote file by referring to their URL location cannot be exploited using VRML++ syntax since all PROTO classes have to be contained in a same VRML file. The potential of EXTERNPROTO in providing distinct code objects located in separated files (i.e. distinct distributed components) was extensively used in this research in order to materialise the concept of portable modelling component at code level. The approach specified by Diehl [96] is therefore too constraining and would have required extensive development effort in order to be adapted effectively to the requirements of this research. As shown in Figure 6-5, the approach adopted in this research consisted in encapsulating the VMRL based inheritance mechanism within specific VRML file-object models. The VRML based inheritance mechanisms are very similar to the one used by Diehl, but instead using a particular syntax and code translation algorithm to generate OOVRLRML, the PoCo modelling
framework provide pre-defined *modelling object* which parameters and functions can be customised depending on the type of system being modelled, and on the type of modelling functions required. This approach to OO coding corresponds to what Lieberman [147] refers to as "prototypical objects", which are templates forms from which instances can be created and that encapsulate the inheritance mechanisms.

It should be noted that such approach constrain the modelling flexibility, since, the modelling objects and object composition model are pre-defined. However, modelling objects (i.e. PoCo modelling elements and components) are designed to be highly configurable. The PoCo modelling framework therefore provides the optimal balance between modelling flexibility (provide generic modelling object that can be used for the modelling of virtually any type of mechanical systems) and modelling effectiveness (minimise code editing and programming knowledge requirement). Furthermore, the overall PoCo modelling framework architecture (i.e. modelling objects hierarchy, and object integration model) remains open so that the modelling capabilities can be extended to new modelling requirements. It should be noted that some code editing is still required (e.g. modelling object parameters configuration, modelling object code composition). However, specific PoCo modelling objects functions (e.g. dynamic event routing generation, pre-defined event passing mechanisms) have allowed to reduce the code editing process to simple objects parameters configuration. This has allowed reducing the complexity of what is shown in Figure 6-5 as "composer tools" functions, to a minimum (i.e. mainly text parsing and editing functions).

### 6-2 PoCo VPE Objects’ models

The PoCo modelling environment defines two types of modelling objects, which are the PoCo *modelling elements* and the *modelling components*. Modelling elements provide basic modelling and model management functions. Elements are composed into components, which represent fully functional (as autonomous) structural constructs from which complete PoCo models can be generated. The PoCo VMRL objects’ models are pre-defined VRML file templates. The VRML code describing PoCo modelling objects is separated in two parts: i) One part, which functions and parameters can be customised in order to support specific functions (cf. chapter 5 for a detailed description of PoCo modelling elements’ functions, and ii) another part which define the invariant functions and event passing mechanism required to support inheritance (i.e. composition of PoCo elements into fully functional component, and composition of components into complete models). The following sub sections focus on describing the later aspect of PoCo modelling objects.
6-2.1 Element VRML file model

6-2.1.1 VMRL file structure

PoCo VPE modelling elements are associated with a single VRML file object in order to approach supports the concept of a software component being a "well defined, portable" binary unit [60] [121]. The VRML EXTERNPROTO feature was utilised in order to isolate PoCo modelling objects' code within separate files to provide a clean modelling code encapsulation. A PoCo modelling element's VRML file is built from i) one unique PROTO node i) one Script node. This configuration allows the event handling and element instantiation mechanisms to be simplified (see below). Furthermore internal element cohesion is the natural outcome when elements are composed into components by making a clear distinction between i) elements as defined by PROTO instances and ii) modelling components as being a composition of elements in a nested PROTO hierarchy.

The generic VRML file structure model is shown in Figure 6-6. The single PROTO node contains i) a VRML scene graph encapsulating all VRML nodes necessary to the implementation of the element's modelling functions, and ii) a single Script node encapsulating modelling related functions and functions necessary to manage internal and inter-elements events. As mentioned earlier, VRML PROTO nodes are composed of a PROTO declaration part and of a definition part. In the same way, Scripting nodes interface with the VRML code via a Script node declaration or interface. Each Script node declaration consists of fields and event declarations that are used to access or define functions implemented within the PROTO definition part. PoCo elements' PROTO definitions contain a VRML scene graph that regroups the VRML nodes required to describe the modelling functions specific to the type of modelling element considered. A common characteristic to all elements is that some nodes in the scene graph are unpopulated and defined as the element's attributes. These attributes are made available through the PROTO declaration (i.e. "Nested_node" field in Figure 6-6). The number and type of elements that are defined as attributes varies depending on the type of element considered.
PoCo elements’ PROTO declarations also define a set of eventIn/Out communication channels. In addition, each element can receive or send a set of User-fields, which provide the user or other modelling objects with access to the attributes of nodes within the scene graph. Finally, a set of Storage fields are defined which are used to hold values used by functions within the Script node. Values of variables defined within embedded JavaScript code are only stored for the duration of the function. To overcome this problem, Beeson [57] proposed the use of a “dumb” VRML node (within the PROTO declaration) whose attributes’ types corresponded to the type of value which needed to be stored (e.g. use of a PositionInterpolator field if an array of x,y,z co-ordinates need to be stored). Adopting this solution automatically result in the VMRL scene graph being overloaded with irrelevant nodes depending on the number of values. The solution adopted in this research, consists of using “Storage_fields” within the PROTO declaration. However, Storage fields (i.e. internal element information) are visible to other elements, which could be interpreted as a violation of the object encapsulation principle. However this approach was considered a worthwhile trade-off due to the dramatic reduction in the complexity of the element’s internal structure.

The JavaScript node embedded in each PoCo modelling element defines two types of functionality. Modelling functions (see Figure 6-6) might vary in number and nature, and are customised depending on the type of element and the functionality that need to be associated with this element. On the other hand, initialisation functions (see Figure 6-6) provide the
generic mechanisms that allow the event routing paths between nested PROTOs (and hence the mechanism required to achieve inheritance) to be automatically generated when model is loaded. Unlike Diehl's approach [96], the implementation of event routing paths between the PROTOs of a nested hierarchy is automatically supported by routine functions implemented as part of the file model. In this way, the internal element mechanisms are completely hidden from the user (i.e. no code editing required; only the configuration of element's attributes fields).

6-2.1.1.1 Event passing mechanisms and interface

The research on OO VRML has highlighted the difficulties of orchestrating events within a hierarchy of nested VRML prototypes [50]. This is because VRML is an event driven language, which complicates both communication within VRML and between VRML PROTO objects. Because of the lack of support for inheritance (i.e. VRML language "built-in" mechanisms), nested VRML nodes cannot directly access the attributes or functions of PROTO classes that are higher in the hierarchy of nested proto. Under the current situation, any function or attribute access mechanisms between nested PROTO must be implemented using the VRML event model, and problems occur when a large number of objects have to be synchronised.

Unlike approaches previously described, the PoCo element model predefines the event passing mechanisms as a part of the modelling element model. The PoCo approach to object event management supposes that the type of elements, elements functionality and nature of the information to be exchanged are known in advanced (i.e. the class hierarchy and types of classes that compose the hierarchy are known). This design choice has been made because the PoCo VPE environment is adapted to the modelling of a specific type of system (i.e. sequential logic driven mechatronics systems). Within this domain, the PoCo element model has been used to define six basic modelling elements that support modelling functionality specifically designed for the purpose of machine prototyping¹. Furthermore, the way elements are composed and interact is also pre-defined so that communication between those elements is part of the PoCo modelling component model.

¹ The detailed descriptions of the different PoCo modelling elements implemented to support the modelling of manufacturing systems are given in Chapter 5.
Figure 6-7 shows the event passing mechanism that characterises the PoCo element model, which aims at ensuring the communication within a hierarchy of nested VMRL PROTO nodes. The PoCo event model differentiates two different mechanisms to support “backward” (from lower to upper levels, path (b) in Figure 6-7) and “forward” (from upper to lower level, path (a) in Figure 6-7) communication with nested PROTO hierarchy [50]. The types and number of backward eventIn/eventOut are intrinsic characteristics of each modelling element, i.e. each element defines what type of event it needs to trigger its internal functions and what type of output will be sent to the external environment.

6-2.1.1.2 Example of PoCo modelling element

The example shown in Figure 6-8 illustrates how the PoCo element model materialises into VRML code. The PoCo element’s PROTO declaration is defined from line 3 to 23, and lists the eventIn (line 4-6) and eventOut (line 21, 22) which the element can receive or send. Lines 7 to 13 correspond to the user fields, which allow the element attributes to be configured. Line 15 and 16 correspond to storage fields which functions have been explained in the previous paragraph. Line 18 and 19 are user fields, which can be populated with the definition of other PoCo modelling element (nested sub classes).
The element PROTO declaration contains VRML nodes that implement the element's VMRL based modelling functions (e.g. TouchSensor node line 56). It should be noted that most of the proto declaration content has been hidden for clarity. Lines 59 to 75 illustrate the embedded JavaScript code used to implement element's functions that cannot be implemented using classic VRML nodes. As explained in the previous paragraph, the Script code can be separated into two sets of functions, which are the initialising and modelling functions. Lines 62 to 69 provide an example of initialising functions that parse all nested PROTO (other PoCo modelling elements) nodes and create event routing route to and from those PROTO. Lines 71 to 73 list JavaScript based modelling functions, which define behaviours specific to the modelling element under consideration (the Script code has been summarised in Figure 6-8, for clarity).

6-2.2 Component VRML file model

PoCo modelling components result from the composition of several modelling elements (characterised by PROTO VRML objects) in a hierarchy of nested PROTO nodes. Hierarchy
of nested PROTO is the means by which VRML PROTO objects functions and attributes can be encapsulated into a new object, hence providing the basic inheritance mechanisms. However, the basic inheritance mechanism provided by VRML can only be used to manipulate instances of PROTO classes, and hence hierarchy of nested PROTO instances cannot be instantiated and further reused. A process that consists in encapsulating instances of elements in new PROTO class node (which can be instantiated) is therefore required. The following paragraphs present the PoCo component VRML file model, which allows such process to be supported (note that a tool, referred to as "composer tool" in Figure 6-5 paragraph 6.1.5.3.2, is required to support this process).

6-2.2.1 PoCo Component object model

The model characterising VRML PoCo modelling components, is very similar to the element model. As shown in Figure 6-9, each component is defined by one PROTO node object whose definition encapsulates a hierarchy of nested PROTO element declarations and routine JavaScript functions. The component's PROTO declaration includes a sub-set of the attribute and event fields of the encapsulated elements. In addition, PoCo modelling components are characterised by a single attribute i.e. the component name. To ensure referencing consistency PoCo components include a script function that distributes this name to all elements directly below the component level in the hierarchy. Each element appends the component name to their own, which is then further distributed to elements below them in the hierarchy. This chain of events trigger the initialisation functions of all elements that compose the component and allows each element to build a unique name composed from their name plus all the names of their parent objects.
The encapsulation of elements within components consists of hiding the internal element hierarchy from the user by replicating the elements attributes and event fields as part of the component’s PROTO declaration (interface). The declaration of a component PROTO therefore varies depending on the number and types of node that compose the hierarchy. It should be noted that only a specific selection of element fields are included within the component interface (PROTO declaration) in order to prevent the user from modifying critical aspects of the component (e.g. actuators motion axis parameters, 3D geometry positioning). In this way, the component-editing phase (which consists of configuring the type and number of element instances that compose a component) can be differentiated from the component configuration and use phases during which only the intrinsic characteristics of the component (i.e. internal element configuration) should be changed. The implementation of PoCo modelling components that can be instantiated (i.e. component classes) therefore requires three steps that are:

- the instantiation of modelling elements from which the component is composed,
- the nesting of those elements according to the hierarchical model defined by the type of modelling elements and their nested PROTO attribute fields
- the implementation of new VRML PROTO objects that encapsulate the hierarchy of nested element PROTO instances into a reusable VMRL PROTO class
In the case of Diehl’s [96] research, the composition process consisted in VRML++ code editing, and was partially automated by using a code translator. In the case of this research, a so called “composer tool” is required in order to edit the VRML file materialising PoCo component. However, because of the particular PoCo element and component model (i.e. pre-defined file structure and Script-based composition functions), the functions that the composer tools needs to provide are reduced to simple parameters parsing and editing.

6-2.3 PoCo model VRML file model

The VRML file structure of a complete PoCo model is very simple and consists of a hierarchy of nested PROTO component declarations (instances). In the same way elements’ functions provide the basic functions to manage inheritance mechanism, the mechanisms allowing component to be composed are supported by specific types PoCo elements (i.e. LP and INT1), so that the editing and re-configuration of PoCo model is reduced to the level of simply re-configuring the model parameters.

As shown in Figure 6-10, the VRML code describing instances of PoCo components and PoCo models is very explicit and does not require any knowledge about the VRML language.

1 The functions of a particular type of PoCo modelling elements are dedicated to provide support for automatic composition of modelling component into a complete and fully functional model. Please refer to Chapter 5 for a description of those elements and the associated ‘structural functions’.
This implies modelling component and component-based models can be modified or re-configured without specific tools or knowledge of the VRML language. More importantly, it also implies that software implemented to either provide user a front end interface, or to integrate a PoCo models and modelling functions as part of a third party software (e.g. Human Machine Interface, complementary Logic Editing Environment machine view) are very simple, and only need to implement VMRL Proto attribute fields editing functions. Various aspects of the model can be changed by simply changing the field parameters. These aspects include: (i) the geometrical assembly that links the conveyor side sections and sensor (use of Link Point elements functions), (ii) the logic that links the behaviour of these two components (Note: the logic defined implies that the conveyor will start if the sensor is on and stop if the sensor is off) and (iii) cosmetic attributes (e.g. colour).

6-3 Chapter overview

The goal of this research was to implement modelling components from which complete manufacturing virtual prototypes could be easily generated and which would enable easy model re-configuration and re-use. PoCo modelling components have been designed as highly functional modelling objects which provide functions i) relating to various functions of manufacturing systems (e.g. kinematic layout, machine control logic modelling), and ii) related to the composition of components into fully functional and structurally complete models. Finally, because of the advantages provided by VRML in terms of modelling functionality and model portability the choice was made to make use of the VRML standard formats and the JavaScript language as an implementation platform.

Issues regarding the reuse and manageability of the VRML code describing the modelling component objects have surfaced due to the high level of functionality required of PoCo modelling components and the poor support from VRML for OO mechanisms such as inheritance. The few research projects that have focused on the development of OO capability within VRML were reviewed in order to determine how PoCo modelling component code management could be effectively improved. A solution has been developed to enable i) the decomposition of the overall component's functionality into PoCo modelling elements and ii) the implementation of an VRML based inheritance mechanism to facilitate the composition of elements into component. Both of these features enable component code reusability and manageability. The research and development outlined in this Chapter has resulted in the implementation of VRML file object models which provide i) re-usable and configurable templates which parameters and function can be customised to define various modelling elements, which functions are specific to the type of system being modelled, and ii) generic
code and event management mechanisms that allow VRML-based inheritance (and hence better modelling code management), to be implemented
Chapter 7  PoCo VPE and real system engineering tools integration

7-1 Chapter introduction

The implementation of manufacturing systems' virtual prototypes (VP) consists of integrating various types of data describing different aspects of the system into an executable 3D computer model. It is strongly believed that the use of virtual prototypes can have the largest impact in the early design phases of manufacturing system design\(^1\) where currently engineering data exist at low level of details and in simple, possibly non-digital formats. From this perspective, Virtual Prototyping Environments (VPE) should provide effective modelling and editing mechanisms at a level of detail appropriate for the implementation of early system prototypes. The translation of engineering data into modelling data is essential in ensuring that: i) a model can be quickly implemented and ii) that when a design solution has been validated in a virtual form it can effectively be used to implement the real system.

As illustrated in Figure 7-1, the issues of concern in this Chapter surround the integration of virtual prototyping, and real system engineering processes and tools. The goal is to automate both engineering / modelling data format translation and modelling data integration.

\(^1\) In Chapter 3, it was highlighted that in this research the approach adopted is to design VPE as an “advanced design drafting” tool, that can enable system engineers to collaborate in the early phases of the design, where uncertainty is high regarding the final solution, around a “common” and intuitive machine representation.
However, the amount and complexity of engineering data describing real manufacturing systems, makes the complete automation of the translation process difficult.

The difficulties of 3D modelling reside in the skills required to implement computer models of complex systems. The complexity that characterises manufacturing systems implies that the modelling of various aspects of the system (e.g. 3D geometry, kinematics, model behaviour) and the integration of various types of modelling data into a coherent model can result in a difficult and error prone task if not supported with effective tools and methods. This Chapter is focused on the mechanisms that allow the user to switch effectively (i.e. with a minimum time and resources mobilisation), between real and virtual data formats. The development of the Portable Component-based Virtual Prototyping Environment (PoCo VPE) has been focused on the integration mechanism between two main types of engineering tools whose functions are related to the mechanical and machine control design of manufacturing systems.

### 7.2 Real / virtual system design data

#### 7.2.1 Data model

Data model is generally defined in the domain of software engineering as a conceptual model in which the various data entities and their relationships are shown [57]. Nierstrasz [56] also describes the object-oriented (OO) paradigm as a data model for software engineering. In the context of this research, the term “data model” characterises a model used as a basis to understand and implement real/virtual data translation mechanisms. The data model implemented for this purpose is defined by:

- **Data type**: Each type of data describes specific aspects of the system being designed. For instance, different data types correspond to the design of Manufacturing Systems (MS) mechanical and control layout.

- **Data formats**: The formats in which design data are generated depend on the type tools used to edit those data or the phase at which the system design is considered. Study of Cross Hüller's machine design process has shown that different data formats were used during early and late design phases.
- **Data structure**: According to Zwegers' definition [57], the data structure is used here to describe the relationships between data types. For instance, relations exist between the set of data types describing the mechanical and control design of a machine, which allows the overall machine behaviour to be defined.

- **Level of detail (LoD)**: The LoD at which the data describe a system is defined as part of the data model. For instance, simple production line layout sketch and detailed 3D CAD models describe manufacturing system at different LoD, which are adapted to different usages.

### 7.2.2 Mechanical design data

#### 7.2.2.1 Mechanical data format

The design of mechanical systems has long been represented by paper-based 2D blueprints. Because of the poor sharing capabilities that this type of format provides, most enterprises have made use of CAD packages to digitalise existing, or edit new design data. It could be assumed that 3D CAD computer models would allow narrowing the gap between real system engineering and 3D virtual prototype data and therefore would provide a basis to implement seamless translation mechanisms. However, this is not necessarily correct since CAD software models are implemented using highly proprietary modelling formats, which make the translation between CAD modellers, and third party environments (such as Virtual Prototyping Environments (VPE)), very difficult. In addition, CAD models are often defined at a very high level of detail, which is not adapted for the virtual prototyping of complete production lines.

Commercial VPE solutions such as Delmia IGrip / Quest [118] have been tightly integrated with the CAD package from the same developer (i.e. in this case Dassault CATIA). Data generated by sub-contractors who do not use the same CAD solution cannot be easily used to generate machine virtual prototypes. Translation from one CAD format to another is highly problematic and is often source of time consuming and error prone tasks that requires specific knowledge and tools. Standards data formats, such as Standard for Transfer and Exchange of
Product data (STEP) [134], or the Initial Graphics Exchange Standard (IGES) [133] for product and CAD data allow engineering data to be more easily exchanged across engineering environments. However, those formats either are too specialised (e.g. STEP which is a process / product oriented format) or hold a lot less information than contained in the original CAD model. Therefore, the translation from native CAD to standard formats systematically results in a loss of semantics (Note: details on the issues associated with CAD format translation are discussed later in this Chapter).

7-2.2.2 Mechanical data LoD

CAD software is designed to support mechanical engineering requirements and therefore provide highly specialised modelling tools that allow a high LoD to be generated e.g. detailed geometry and assembly modelling. Therefore, the data required to implement a virtual prototype theoretically exist in a CAD model, but at a level of detail, which is by no means suitable for the prototyping of a complete production lines [14]. Implementing the virtual prototype of a complete production line from CAD models would result in significant modelling time and in a very high level of model complexity, which limits the use of virtual prototypes to departments and engineers in the enterprise who have access, highly specialised hardware, software and skilled personal (i.e. graphic stations and CAD environment).

The LoD of CAD models might be reduced in order to implement complete machine virtual prototypes that can be used in less specialised environments. However, the post processing of CAD models (if they exist) cannot easily be automated since the LoD requirement may vary depending on the machine, or the part of the machine, being prototyped. In addition, whereas the LoD simplification of 3D geometry (polygon simplification) can be achieved using specific algorithms, the simplification of more complex data (e.g. assembly kinematic) requires CAD models to be re-edited. Therefore, achieving tight integration between CAD and VPE software does not necessarily provide an effective solution to the issues related to the modelling of large-scale mechatronics systems.

7-2.3 Model kinematics data

7-2.3.1 Model kinematics data format

As for the machine part geometry and dimensions, the engineering data related to the machine kinematics is contained in the mechanical drawings (paper based or 2D CAD), or CAD assembly models. In the case of 2D drawings (even CAD-based 2D models), the model of a machine assembly is static in nature. The definitions of kinematic links between assembly parts exist but specific engineering knowledge is required to interpret the machine blue prints
and to retrieve the information describing machines’ kinematics. Three-dimensional CAD assembly models are more intuitive and contain a mathematical definition of the kinematic links between machine parts. These kinematic relationships are the result of two main assembly data types. The first type relate to the constraints between geometrical features of the model’s parts (sometimes called “assembly mating relationships”) e.g. coincidence, parallelism or other geometrical relationships between edges, plans, axes or other parts’ geometrical features. By creating mating relationships, the CAD engineer can define the possible motion characteristic of a part relatively to another (degree of freedom). The other type of data from which CAD models kinematic characteristics is defined as the parent-child relationships that exist between parts. Parent-child relationships ensure model kinematic consistency by automatically propagating any transformation applied to a given geometry, to all part of the CAD model defined as children of this geometry. Parent-child relationships are achieved through the manipulation of reference co-ordinates system (RCS) in which the parts’ geometries are defined, and in which mating constraints are mathematically expressed.

7-2.3.2 Model kinematics data LoD

CAD assembly models therefore contain the information required to implement virtual machine prototypes’ kinematic. However, as for the geometrical modelling, CAD assembly models are defined at very high LoD i.e. CAD assemblies are typically composed of large numbers of parts, describing the machine composition at a very high level of detail (i.e. individual parts definition). However, in a MS assembly, many parts are fully constrained, which means that they have no relative degree of freedom. Such fixed assembly do not need to be modelled for prototyping purposes (i.e. exhibit machine dynamic behaviours). The LoD simplification of CAD assembly models would consists in isolating the fixed assembly and simplifying those into a single 3D geometry (i.e. removing any data relating to the assembly definition). CAD software do not provides pre-defined assembly LoD simplification functions that would allow the transition between CAD models and 3D virtual prototypes to be automated.

7-2.4 Logic control data

7-2.4.1 Logic control data format

The behaviour of MS which prototyping is considered in this research is described by sequential state-based logic describing the various states in which machine actuators and/or sensors can be, and the conditions associated with the state transitions. Sequential machine control programming is most often associated with programmable logic controller (PLC). PLC programming environments (e.g. Siemens STEP7) supports the editing of machine logic
in various formats. In the domain of sequential logic programming, five forms of logic representation have been standardised by the IEC 1131-3 International Electrotechnical Commission's standard [135] (namely, the Sequential function chart (SFC), Ladder and Function blocks diagrams (LD and FBD), Structured Text (ST) and Instruction List (IL) languages).

Figure 7-3: Examples of sequential machine logic data formats

Logic programming environments typically provide logic simulation and debugging capabilities. Most commercial logic control editing packages provide one or several programming interfaces (largely language-based) for system engineers to develop, control and simulate machine logic configuration. Whereas these languages are generally powerful and flexible and provide simulation capabilities, users have to be proficient with these languages in order to implement or interpret machine logic description. In the case of the Cross Hüller design processes for instance, such tools are only used very late in the detailed design phase. During early design stage, paper-based timing diagrams are used which do not allowed the machine logic to be simulated. For prototyping purposes, it is essential to provide system engineers with tools that maximise the logic simulation capabilities in return of very simple data editing and model configuration processes, so that prototyping can be conducted during the early design phase.

7-2.4.2 Logic control data LoD

The Level of Detail (LoD) at which machine logic control data is expressed depends on the lifecycle stage of the machine design. As shown by the analysis of Cross Hüller design process (see Chapter 3), during the conceptual design stage, only drafts of the overall production line operation sequence and timing are produced. At this stage, the goal is to distribute the production throughput constraints imposed by the customer on to operational times between various machine stations and associated machining/assembly operations. The machine actuator interlocks are described at a very low level of detail, and the details of the control layout are mostly unknown. In the detailed design stage, the overall line operation is

1 A review of Cross Hüller production line manufacturer process has been proposed in Chapter 3.
broken down into individual "machine stations" timing and operations, and finally individual actuators timing and interlock are defined. As a tool used to support the design of manufacturing systems, the VPE should provide the capability to model control logic at various levels of details and at various stages of the design process in order to support the incremental and iterative approach that characterises the control logic design of MSs. It is necessary that prototyping tools allow model behaviour to be implemented and tested simply, before any detailed control data (e.g. ST and IL control code) is generated.

The following aims at describing the approach adopted in this research to implement the mechanisms that have allowed the Portable Component-based Virtual Prototyping Environment (PoCo VPE) to be integrated effectively with real MS engineering tools. The real / virtual design environment integration is mainly concerned i) the integration between CAD and PoCo VPE to minimise the modelling task by making use of existing 3D modelling data and ii) the integration between the engineering tools related to the editing of machine control data. The main conceptual performance targets were i) simplifying the data translation process and ii) maintaining simplicity and portability of the prototyping environment and the PoCo VPE virtual prototypes.

7-3 Real/virtual system data integration

7-3.1 PoCo - CAD data integration mechanism: a data format issue

CAD models encapsulate most of the data required to implement the geometrical and kinematic aspects of virtual prototypes (VPs). However, CAD models are expressed in highly proprietary formats, which limit the portability of such models. For the purpose of this research, it was essential to implement machine prototypes using standards and web compliant formats. The issues associated with modelling format translation are well known and efforts have been made to allow a more straightforward data exchange between CAD software using standards 3D modelling formats (e.g. IGES, STEP and VRML). CAD output results in "dumb" or "neutral" models, which only contain the geometrical information of the initial CAD part, or CAD assembly geometry in the form of solid or surfacic models. This approach results in an important loss of semantics (e.g. assembly constraints, feature history, cosmetics).
The proprietary nature of CAD formats creates a tight link between the formats in which the machine engineering data is expressed and the CAD software used to interpret these data. This is partially the result of CAD developers' strategy to keep the user base by preventing integration of third party software components. However, this makes the exchange of data between CAD, or between CAD and other modelling applications, problematical. The translation of large amounts of complex CAD nodes requires the services of specialised companies that possess the expertise and software tools necessary to ensure lossless ("Al) data translation. However, this approach raises many issues in terms of time and cost involved in the translation of CAD data but also regarding the release of confidential data source to a third party (Figure 7-4).

For the development of the PoCo VPE, the integration with CAD software has been a major concern. A substantial amount of companies' mechanical engineering data is already in a CAD format, and forcing industrialists to re-edit mechanical data in other format would significantly reduce the value of the VPE developed in this research. At the very beginning of this research, and for issues related to the virtual prototypes portability, it has been decided that PoCo 3D machine prototypes would by implemented using standard rather than proprietary modelling formats. This choice has been ported onto the Web3D VRML modelling language. The issue of CAD / VRML data translations was therefore relevant and needed to be carefully investigated in order to find an operational solution.

7-3.1.1 CAD – VRML translations issues

VRML was initially developed as an interchange format for 3D models. Despite the fact that the VRML format has widely been adopted by the academic community to implement fully functional but lightweight and web compliant 3D models, the use of VRML in industry is practically non-existent. Although VRML is supported as an input/output format by virtually
all of today’s CAD packages, the VRML import / export capabilities provided by CAD packages are very poor. The amount of information lost during the translation of a CAD assembly model to VRML varies depending on the CAD package used. One goal in investigating the CAD-to-VRML translation issues was to assess the amount of information that could easily be extracted from CAD models.

Figure 7-5: Type of data contained into CAD assembly models

As shown in Figure 7-5, CAD assembly models contain three types of information required to implement virtual prototypes, namely i) 3D model geometry and cosmetic information, ii) the assembly relationship and parent-child tree structure and iii) assembly kinematic definition. Two test CAD assemblies have been used to assess the potential of various CAD packages to translate the initial CAD assembly data into a VRML format. The two test assemblies characteristics’ (i.e. tree structure depth, type of assembly mating relationship and assembly kinematic) are relevant to the format translation capabilities that need to be tested. The assemblies were designed, implemented and translated into VRML format using the built-in CAD exporting modules.
The results are summarized in Figure 7-6. All of the CAD software packages generated an accurate VMRL definition of the assembly 3D geometry, with no loss of detail or data distortion. Absolute model scale and model proportion were also consistent with the initial CAD model. Conversely, the tree structure, or assembly structure defining the parent-child relationships between assembly parts, was partially or completely lost depending on the CAD software used (e.g. ProEngineer, Unigrafix, and SolidWorks). The reproduction of the initial CAD assembly tree structure was rated based on two criteria, which where the capability to output a separate VRML geometry node corresponding to each part in the assembly, and the capability to exploit the VRML scene graph and node grouping features to conserve the parent-child relationships between parts. In all cases, CAD-VRML output only consisted in one surface envelope of the overall CAD assembly geometry so both assembly structure and part definition were lost during translation. Dassault CATIA’s VRML output consisted in a set of VMRL files (rather than a single file) containing the 3D geometry definition of each part composing the assembly. An additional VRML file was created using the Inline VRML node to regroup all the parts geometry in a single VRML scene. However, the tree structure was also lost and all parts where placed at the same level in the VRML scene graph hierarchy. The Adams plug in for SolidWorks CAD software, output VRML in which each assembly part was defined as a distinct VRML geometry. The tree structure was partially reproduced too, however and surprisingly, only the two first levels of the CAD assembly tree structure (parts hierarchy) were translated. CAD assemblies with deeper hierarchy were systematically flattened into a two levels VRML scene graph hierarchy despite the numerous attempts to find an “in CAD” assembly creation procedure (via the use of sub assemblies and the testing of various assembly parameters).
The Solidworks Adams plug-in was also the only software which VRML file output contained information about the initial CAD assembly kinematic characteristics. The CAD assembly kinematics data were not explicitly expressed using VRML nodes (e.g. no definition of rotation or translation axes). However, the information was implicitly contained in a VRML interpolator node as a succession of positions and orientation values for each model's geometry that were computed based on the initial CAD assembly kinematics characteristics. This data could potentially be traced back but such task would require a (relatively complex) VRML file parsing and post-processing phase.

Other, less relevant format translation performances were also evaluated. It has been noted that all CAD VRML output resulted in large VRML file size (i.e. VRML file size significantly larger than the original CAD file) since parts' 3D geometry was described using a set of vertex and surface (IndexFaceSet VMRL node) rather than pre-define CAD modelling features (e.g. extrusion, fillet, holes templates). A test to determine the geometrical Level of Detail (LoD) reduction has been carried out using trial versions of third party software developed for this specific purpose (e.g. Rational Reducer, Vizup), and most CAD models (with usual geometrical characteristics) could be reduced by approximately 70% without obvious loss of visual details. Some CAD software were also better at translating cosmetic information such as colours, transparency and material characteristics (e.g. shininess).

In conclusion, it can be stated that CAD export modules from native CAD format to VRML are highly immature. Despite the advanced modelling functions provided by the VRML format that could easily be used to achieve lossless translation of the most important CAD assembly data (i.e. tree structure, kinematic information), none of CAD packages tested were able to output VRML files containing the initial assembly characteristics. It was clear that the limitations mentioned above were not the result of technical difficulties but were rather due to a lack of interest of CAD developers in the standard formats and especially in the VRML format. The Adams Mechanics plug-in for SolidWorks produced a VRML output model that exhibited the kinematic characteristics of a mechanical assembly. However, the resulting VRML model consisted of a "3D-based film sequence" displaying model parts at specific positions through time. This meant that the assembly kinematic data were not explicitly defined and additional post-processing was required to extract the original assembly information. It was therefore necessary to investigate the possibility of post processing CAD VRML file outputs as a mean to achieve integration between proprietary CAD software and the PoCo machine prototyping environment.
7-3.1.2 PoCo CAD integration

Even though the development of specific tool to ensure lossless translation between CAD and VRML formats was not the major focus of this research, it was essential to achieve a maximum level of integration between CAD and the PoCo modelling environment. CAD software provides modelling functions that are highly adapted and widely used in the domain of manufacturing engineering and therefore it was necessary to overcome the limitations of CAD export capabilities. Several approaches have been investigated and are described in the next sub-sections.

7-3.1.2.1 Primary approach to PoCo CAD integration

An approach investigated in the early version of the of the PoCo modelling environment\(^1\), was to post process CAD software VRML files output in order to generate re-configurable, reusable VRML modelling objects. Three main types of information were added to the VRML file as "meta tags" represented by VRML comments "#" placed in key locations in the VRML file:

- **Referencing information**: VRML nodes within CAD - VRML file output were referenced using generic names (e.g. "XXXX1") or were not referenced at all. The referencing information added during the post processing primarily aimed at adding some type of naming information that could be used to designate the various elements within the VRML file. This was required to enable further VRML code post-processing. Other user-defined naming information was added in order to facilitate the management and reuse of modelling objects.

- **VRML file structuring**: CAD VRML file output was in most cases characterised by a flat VRML scene graph hierarchy. This part of the post processing aimed at re-manipulating and complementing the internal VRML scene graph in order to provide the VRML file with the initial CAD parent-child parts hierarchy (required for maintaining the model kinematic consistency). This operation consisted of grouping VRML nodes and re-locating nodes in the VRML scene graph hierarchy. The structuring phase also consisted of encapsulating the VRML file code into reusable PROTO nodes, which could be instantiated and therefore provided modelling object reusability.

---

\(^1\) Please refer to Chapter 8 for an overview of the various phases that have characterised the development of a VPE, for more details on the various approaches adopted to integrate VPE and CAD software environment.
Interfacing: The objective of this post-processing phase was to generate modelling objects that could be composed with other similar objects to build a machine model. As such, these objects had to interact with each other. The interfacing information essentially aimed at adding JavaScript nodes and at implementing event-passing mechanisms in order to ensure effective intra and inter modelling object communication.

The post processing of CAD - VRML output raised several issues. An attempt to implement a post processor has been initiated as a case study, using Unigraphics V.17 and SolidWorks VRML file output. However, it quickly became obvious that this approach to the integration between PoCo and CAD environments, (whose main goal was to avoid repeating modelling tasks that had previously been conducted in CAD (i.e. 3D geometrical and assembly kinematics modelling)), had too many drawbacks:

- Complex post processing: The amount and complexity of information that had to be added to the VRML file became greater that the amount of information that could be extracted from the initial file. The post processing was therefore a re-editing process. The VRML file’s node referencing task, which was the most trivial (but essential for the interfacing and structuring phase), still required complex VRML file parsing. For CAD - VRML file output corresponding to small and simple assemblies (i.e. simple kinematics, limited number of shapes, no nested parent-child relationship hierarchies), the automation level of post processing was acceptable. However the post processing of VRML files corresponding to large CAD assembly required frequent and extensive manual input (file editing ) at the VRML code level

- File diversity: the overall file structure and the amount of initial CAD assembly information contained in VRML files differed significantly depending on the CAD package used. Trivial examples are cosmetic information, but more valuable information such as parent-child assembly relationships and kinematic information were either translated into different VRML scene graph structures, or were not translated at all. This meant that a post processor would have to be implemented for every CAD package. Considering the variety of CAD packages used in the industry, and sometimes within the same company, this approach would result in significant development time and resources. In addition, and considering the large difference in terms of output VRML files' structure and data content, it would not be possible to
guarantee similar levels of performance and functionality across post processors implemented for different CAD software.

![Diagram of VRML CAD file output and post processing approach to model component implementation](image)

It was clear that post processing the CAD-VRML file output could not be done effectively because of the CAD output diversity. In addition, among the software packages tested, only the Adams Mechanic plug in package for SolidWorks provided data that could somehow be exploited to trace back assembly kinematic information. The CAD – VRML file output post processing approach was therefore discarded due to the excessive amount of development required. An attempt to use the VRML meta tag "#" and to define a generic VRML file structure that could be used to aid the PoCo VPE development, (and which could potentially be used as a basis for collaboration with CAD developer on a "common VRML file ontology"), was made. However, it was clear that VRML file post processing was bound to limit significantly future development of the PoCo modelling environment.

7-3.1.2.2 Final approach to PoCo CAD integration

The final approach adopted to develop the latest version of the PoCo modelling environment and to achieve integration with CAD data, was radically different from the approaches previously described. This approach consisted in separating the information that could reliably be translated by any CAD software into a VRML format, and the information that could not be exported by CAD format translators (or whose form and composition was highly dependant of the CAD package used). All of the CAD packages tested were able to exploit VRML as a 3D geometry modelling interchange format. All types of geometry were effectively translated into a set of VRML IndexFaceSet nodes that described the overall surfacic skin of the initial CAD part or assembly. Conversely, assembly related data such as
kinematics links and parent-child relationships were in most cases lost or untraceable. The functions required to implement the assembly data were therefore implemented as part of the PoCo modelling environment functions. This has resulted in the implementation of particular modelling objects, namely i) the PoCo DYN modelling element\(^1\), which provides the functions required to model part transformation that can either be a translation, a rotation or a combination of both and ii) the LinkPoint (LP) object that allow part assembly to be achieved.

![Diagram](image)

**Figure 7-8: Example of PoCo modelling element 3D geometry configuration**

As shown in Figure 7-8, this approach has allowed model kinematics to be dissociated from model geometrical modelling, so that the 3D geometry of PoCo modelling component could be defined as a parameter rather than as an intrinsic part of the model. Because the CAD-VRML file output is only used to configure the model geometry, any CAD package providing VRML output (i.e. virtually all CAD packages) could be used to implement the geometry of PoCo models.

Compared to the CAD-VRML output post processing approach, this solution is highly advantageous since it fully exploit the capabilities offered by the VRML language. The only downside of this approach is that editing of the model kinematic is required. Nevertheless, this process is simplified by the particular nature of the PoCo modelling elements. This approach contrasts with virtually all other research projects [97] [14] [120] which have attempted to achieve integration of CAD and VRML based environments by focussing on the post processing of CAD VRML output files. These projects have highlighted the same issues

---

\(^1\) The description of the various PoCo modelling constructs, defined as modelling elements and component, which provide the basic modelling functionalities, can be found in Chapter 5.
as found in this research since the post processing tasks also consisted of referencing, grouping VRML nodes and of implementing interfaces to other software (e.g. Java code, Salmela et al. [97]). The best example is provided by the research of Ressler et al. [111] who have implemented a so-called “Deneb Translator” in order to generate structured VRML files. However, as mentioned by the authors, significant manual input was required in order to process large models and the translator could only be used to process Deneb models.

7-3.2 PoCo - machine logic editing data integration

The modelling of machine control logic is aimed at emulating the real manufacturing system behaviour. In the case of discrete and sequential logic, the control data describes the various states in which a system can be, and the logical conditions associated with the transition from one state to another. Two general approaches to the implementation of virtual machine model behaviour can be dissociated and are described here from a modelling perspective i.e. by emphasising on the implications of both approaches on the design and implementation of the machine prototyping environment.

7-3.2.1 Primary approach to model behaviour modelling

The editing of machine sequential logic is usually facilitated by specific engineering environments providing machine control editing and testing functionalities. Such environments allow engineers to detect and correct logic related design inconsistencies (e.g. dead locks, state inconsistency). These logic testing functions are implemented via a logic “simulation engine” which allows edited machine logic to be simulated in real time. Events generated by logic simulation engines can therefore be used to drive MS VPs so that both machine mechanical and logic designs can be conducted and tested simultaneously. As shown in Figure 7-9, a primary approach adopted in this research, has consisted in implementing a real time link between 3D machine model dynamic display mechanisms and an external logic simulation engine or any other machine logic input (e.g. real machine state broadcast events). From a VPE design and implementation perspective, this approach implies that only display mechanisms have to be modelled, so that the actual modelling of the system behaviour is not part of the functionality supported by the VPE.
This approach necessitates the implementation of an “integration interface” that allows external events, such as state changes, to be propagated to the VPE. The interface that links the two environments can be complex depending on the level of functionality required. Real time communication of events between two heterogeneous environments raises some issues. This is especially the case if VPE and logic simulation engine are distributed on remote machines. However, the major concern when designing the PoCo VPE was the implications that such an approach has on the overall system portability. Even by ensuring a maximum portability of the machine 3D models, the fact that an essential part of the prototype (i.e. behavioural logic) relied on the functions provided by external software seriously impaired the overall prototypes’ portability.

This approach to virtual prototype behaviour modelling was implemented in the first development phase of this research. The logic simulation engine was part of the Process Definition Environment (PDE), which is a machine logic control engineering software tool, implemented within the COMPAG/COMPANION projects. Both the PDE and virtual prototypes were linked via a so-called “broadcaster-server”, propagating the events generated by the PDE run time engine, to various other tools (e.g. HMI, 3D prototypes) developed as part of the Component-Based machine-engineering environment. However, this approach marked a separation between i) the environment in which the machine geometry and kinematics characteristics are described (e.g. a 3D machine prototype build upon the data described by a machine CAD model), and ii) the environment used to simulate the machine logic behaviour. This configuration requires defining how the information needed to implement a fully functional machine prototype, are distributed and co-ordinated between both environments; Ladder logic, Grafcet diagrams, or List statement do not contain any information about the machine dynamic settings, as for instance the position of actuators in
each states or the transition speed/time from one state to another. In the same way, the machine 3D model does not initially contain any information about the number of states for each actuators/sensors, the interlock condition associated with each transition. This implies that the modelling environment needed to integrates some logic related functions required to implement what is shown in Figure 7-9 as "mapping information", which allows machine logic description (state based description) to be mapped to the machine dynamic parameters (actuators position or part colours corresponding to each states for instance).

7-3.2.2 Real / virtual data translations

An alternative approach to implement virtual machine behaviour consisted of the translation of the machine logic control data generated using machine logic editing environment, in a format that could be used to configure a logic simulation engine implemented as a part of the modelling environment itself. This approach to the implementation of virtual machine prototype behaviour is radically different from the approach previously described. Regarding the objectives that were targeted in this research, this approach had many advantages. By decoupling the machine model and modelling environment from highly specific machine engineering tools, it was possible to increase vastly the portability of fully functional virtual prototypes. In addition, the implementation of the link between machine logic description and the actual 3D machine model was made simpler because developed as part of the same software environment. Real time event management was therefore easier and complex software integration infrastructure and data management services could be avoided.

Figure 7-10: Machine logic translation approach to the implementation of virtual machine prototype behavioural logic

However, this approach requires the control logic data to be translated from the initial format generated by the logic editing software to the format used to configure the model's logic simulation engine. As highlighted by Adolfsson [8] the sequential logic description data describing a simple machine actuator (e.g. isolated actuator) is very simple and only consists
in a set of state and state transitions definitions. Therefore, the translation of such data can be straightforward. However, the complexity of machine logic descriptions increases exponentially when hundreds of actuators' states and state interlocks are described. The task of translating control data from one format to another is therefore made more complex and difficult to automate and the risks of distorting data or injecting errors during the translation process increase. As explain later in this Chapter, the difficulties associated with the translation of large amount of data have been tackled by adopting a common component-based architecture for both real system and virtual prototypes control.

7.3.2.3 PoCo approach to virtual prototypes logic implementation and simulation

The approach adopted to implement the behaviour of models implemented using the PoCo VPE could be qualified as a “hybrid”. The target was to implement a logic simulation engine whose functions could be implemented as part of the virtual machine model so that the model could be decoupled from third party software that would impair the overall model portability. In addition, it was thought to be necessary to allow events generated by external software (e.g. machine control editing environment simulation engine) to be used to drive the model behaviour so that PoCo model could be tested using third party logic simulation engines. This hybrid approach allowed both model portability and integration of the PoCo modelling software within a machine-engineering environment, providing very flexible prototyping capabilities. The main issues that had to be faced during the implementation of the PoCo VPE logic simulation engine were:

- To implement a “distributed sequential logic simulation engine” whose functions could be encapsulated in the components from which machine prototypes are composed.
- To implement the logic simulation engine’s functions using exclusively VRML and Java Script for VRML, in order to preserve the portability of PoCo modelling component and PoCo machine prototypes.
- To automate, or at least to simplify the task that consisted of translating machine control logic from one format (e.g. List code) to the format used to configure the PoCo logic engine parameters (mainly text string parameters).
- To implement mechanisms allowing events from external logic simulation engines to be used concurrently with the model’s internal logic engine to drive the model behaviour (hybrid approach to virtual prototype behaviour modelling).
As shown in Figure 7-11, the decomposition of complex systems into smaller constructs has allowed the breaking down of the complexity associated with the implementation and management of functions relating to the modelling and simulation of machine logic. The COND (condition) and STA (state) PoCo modelling elements provide the basic functions from which sequential, state-based control functionalities can be implemented. The low level of granularity at which these elements are defined (i.e. one COND element represents one transition condition, one STA element represents one machine state) implies that the translation of logic data can be achieved at the component level, hence simplifying the translation functions and allowing exceptions to be handled as a separate process. In the same way, the mapping and integration mechanisms between components’ display functions and logic data (i.e. integration and/or composition of COND, STA and DYN elements functions) is pre-defined as part of the PoCo component model. The mapping process (i.e. modelling elements integration into a component) is therefore automated and made transparent to the user.

Each PoCo modelling component stands as a highly autonomous and fully functional machine model construct that encapsulates all of the functions (provided by different type of PoCo modelling elements) required to reproduce and co-ordinate both 3D machine mechanical and behavioural aspects. It should be noted that each component can be defined at any level of granularity (i.e. a component can be defined as a single moving machine part or as a group of machine actuators) but the elements are defined at a level at which data management and integration mechanisms are relatively easy to handle. This has allowed all PoCo modelling objects (i.e. components and elements) and object functions (i.e. dynamic 3D geometry display such as position interpolators, logic engine functions such as state transition condition testing and event management mechanisms) to be implemented using exclusively VRML and
JavaScript code (in order to ensure a maximum level of model portability). Only a web browser is required to view, run / test and interact with a PoCo machine prototype, once it has been implemented and configured.

7.3.2.4 Machine logic formats translation

In theory, the component-based approach applied to the VPE design also provides advantages with respect to the translation of machine control data (from one format to another). The decomposition of the data describing the behaviour of a machine makes the translation of such data simpler and more manageable because it is considered at the level of granularity of components. However, complete automation of the translation process can only be achieved if the structure of the control data is the same in both formats.

As shown in Figure 7-13, machine design tools (i.e. Human Machine Interface, Process Definition Environment (i.e. PDE machine logic editor, simulation engine) and PoCo VPE) which compose the COMPAG/COMPANION machine engineering environment, have been developed around the component-based paradigm. As such, all engineering tools make use of a common “data ontology” which allows engineering tools functions and integration infrastructure to be consistent. With respect to the integration between the real and virtual machine control environments, the common data structure defined by the component-based approach to machine engineering has allowed the translation of machine logic data generated within the Process Definition Environment into a format used to define PoCo modelling element parameters to be greatly simplified.
Machine control logic is edited in the PDE in the form of state-transition diagrams, which describe the control elements associated with machine actuators or sensors. Each state and transition objects used to implement the PDE machine logic description is associated with a type of PoCo modelling element (i.e. COND and STA elements) so that the mapping between the two descriptions of machine logic is straightforward and the translations between the two formats can easily be automated (see figure 7-12).

The first attempt to achieve integration between PoCo VPE and PDE logic editing environment resulted in the development and implementation of a “mapping” tool, which was part of the PoCo modelling environment. The mapping tool’s functions were designed to access a central data repository in which all machine design data generated using the COMPAG/COMPANION machine engineering environment were stored. As shown in Figure 7-13, the mapping tool’s user interface provided a detailed list of all machine logic elements stored in the PDE database (right panel), and a complete list of all modelling components available for configuration. Both machine logic elements and modelling components could be mapped, and a GUI allowed the user to define the cinematic parameters (e.g. position, transition time) (central panel in Figure 7-13).
However, this approach to the mapping tool design limited the exploitation of the PoCo VPE as a generic tool since it was tightly coupled to the PDE database and to the particular component-based data structure adopted within the COMPAG/COMPANION projects. Further development of the COMPANION machine engineering software environment has led to the use of the XML (eXtended Mark-up Language) format to achieve integration between PoCo VPE and PDE software. The XML format has been used to provide a more flexible way of integrating the PoCo VPE with other logic-editing environments. The XML language allows data structure and data content to be dissociated and can therefore be used to generate machine logic descriptions formatted according to any component-based machine architecture, regardless of the format in which the actual machine logic is expressed. This practically implies that the PoCo VPE modelling environment can be used to import machine logic descriptions from any environment able to generate XML file formats according to the component-based machine architecture.
Figure 7-14 shows a sample of the XML machine description generated by the PDE machine logic editing tools. The figure shows the XML data structure and its relationship with the COMPANION Component-based (CB) machine architecture. The VRML code corresponding to a PoCo modelling component used to model the Clamp_(JB6) machine component is shown in the bottom part of Figure 7-14. The XML code describing the real machine logic and the VRML code describing the same machine component’s logic control are consistent and share the common CB data model.

It should be noted that in Figure 7-14 the sub-system level is not defined in the PoCo machine model hierarchy, since the level of granularity at which PoCo component can be defined is...
flexible, and therefore can encompass what is defined as sub-systems in the COMPAG machine control terminology. This does not impair the automation of the translation between the two formats as long as the names designating the component and elements are consistent. Another difference between the two machine control models is the fact that the PDE machine logic description does not have to differentiate static from dynamic states. For instance, the Clamp\_JB6 actuator's state sequence is defined as the succession of four states, two of which describe dynamic states (Waiting\_for\_Clamped, and Waiting\_for\_Unclamped, cf. bottom of Figure 7-14). From a logic implementation perspective, there are no differences between static and dynamic states, since all machine states are purely logical. From a modelling perspective however, static and dynamic states have different implications regarding the model behaviour (motion or static display of 3D geometry). If component and element's naming is consistent between the two logic data formats (i.e. PoCo logic engine configuration parameters and PDE XML/database) the integration is straightforward. More details on the mapping environment functionality and interfaces shown in Figure 7-13 are given in Chapter 8.

7.3.2.5 External event driven PoCo model

PoCo machine prototypes encapsulate a logic simulation engine that increases model portability since all of the functions required to simulate a machine model are encapsulated within the model itself. However, the capability to "plug" the model into an external logic simulation engine is essential so that real machine logic can be directly tested instead of having to translate the real machine data into modelling parameters before testing it. The necessity of such functionality has been highlighted during the integration of the PoCo Virtual machine prototyping environment with the PDE environment. As part of the COMPANION environment, it was necessary to implement a link between PoCo machine models and the output events generated by the PDE logic simulation engine. A so-called "broadcaster" was implemented as part of the COMPANION software integration infrastructure and was used to broadcasts logic-related events (generated by the PDE logic simulation engine or a real shop floor network for instance) over LAN or internet networks.

The interface between PoCo models and the COMPANION integration infrastructure broadcaster are linked via a Java Applet embedded in a web page along with the PoCo VRML model. A list of all components from which a PoCo machine model is composed is retrieved by the Java Applet interface and a link to and from the PoCo modelling components (which implements an interface to receive/send logic related events as part of the DYN elements) is created. When the model detects an event from the broadcaster, the model display functions are triggered.
Modelling components that compose PoCo models can be configured to react to events generated by the internal PoCo logic engine (i.e. via PoCo STA and COND machine logic modelling elements) or to external events. A machine model composed from several PoCo components can therefore be configured: i) as a completely autonomous model which only relies on the internal logic engine, ii) as a model exclusively driven by external events (e.g. external logic simulation engine), or iii) as a hybrid partially open to external events and partially relying on the internal logic engine. Modelling components that are configured to respond to external events ignore events from their internal logic engine, update their states according to the external event received and still inform other components of their state change. Other components not linked to external events can therefore update their internal state, trigger logic engine functions and send events e.g. back to an external logic engine. It is therefore possible to implement “hybrid” prototypes, which partially runs on a virtual logic, and partially driven by, for instance the events propagated by a COMPANION broadcaster linked to a real machine. This capability is, as far as the author is aware, unique, and is made possible because of: i) the component-based nature of PoCo machine models and ii) the logic simulation functions have been distributed across the components from which machine models are composed. This type of model capability can be used to test a partially complete machine, or machines that are in a re-configuration phases.
PoCo models of Lamb Technicon, Krause and Asda real machine test rigs, have been tested in both internal (internal logic engine) and external (use of external event generated by the broadcaster) configurations. The hybrid simulation mechanisms have not been fully tested on a real industrial machine for safety reasons. However, the consistency of PoCo models configured as hybrid models has been validated by using manual user button implemented within the Applet interface, which allows events simulating external events to be manually generated. It should be noted that configuring PoCo modelling component in either automatic mode (internal logic event) or external (logic event) mode only consists of changing a Boolean value that is part of the component configuration parameters.

7-4 Chapter Overview

This chapter has focused on the integration between manufacturing system engineering and virtual prototyping (or modelling) tools. The approach adopted to initiate the integration of such environments was to define a data model that could be used to focus on the relevant aspects and functions of the environment that had to be integrated. This data model has defined four mains aspect (i.e. data types, formats, LoD and structure) of the three main data types (i.e. mechanical/geometrical, kinematics/dynamic, logic/behavioural) which need to be considered in order to implement effectively useful virtual prototypes.

An innovative approach to the translation of complex CAD assemblies models into VRML models has been proposed, which contrasts with the approaches adopted in previous research projects on the subject. The approach adopted and implemented in this research requires part of CAD models to be re-edited, but provides complete re-usability and re-configurability of resulting modelling components (down to the model geometry), effectively ensuring that modelling time (and hence costs) for future projects is minimised. Overall, it is believed that the approach adopted in this research is greatly beneficial in terms of reduced modelling time, increased process simplicity and environment genericity (compliant with any CAD software).

Finally, a radically new approach to the implementation of distributed logic simulation engine has been described. The component-based distributed sequential logic simulation engine provides a high level of portability for manufacturing systems virtual prototypes, and readily supports model re-configuration. In addition, mechanisms have been implemented that allow models to be remotely controlled and / or monitored, allowing them to either run in simulation or monitoring mode. Finally, and uniquely, PoCo models can also be configured as hybrid
models, partially relying on internal logic simulation functions, and partially on external input to provide full simulation capabilities.
Chapter 8  Research Cases Studies

8-1 Chapter overview

This Chapter is aimed at describing the overall research progress that has led to the final of the PoCo VPE development. The various development versions of the PoCo VPE modelling framework and software design are presented through the description of several cases studies, which consisted in the prototyping of various types of manufacturing systems and test rigs for both the automotive and the supermarket warehousing industries. At every stage, the applicability of the PoCo VPE was assessed and the overall development was evaluated with respect to the goals and conceptual performances initially targeted. The conceptual performances used to benchmark the VPE development are defined as follow:

8-1.1 Use of the component concept as common model

This aspect of the VPE development focused on concretising the concept of "component". This concept (or component-based paradigm) is used in the COMPAG/COMPANION project as a common model for the design and implementation of various tools that compose the machine engineering software toolset. The design of so-called modelling components focused on two aspects. Firstly, the consistency between real system components (e.g. mechanical machine modules, distributed control nodes, control software components) and the virtual modelling components in terms of functional and architectural characteristics was assessed. Secondly, the implementation of these modelling components was considered from a software engineering perspective, which aimed at assessing the consistency between PoCo modelling object and VPE software components.

8-1.2 Level of re-usability and re-configurability of PoCo models

The re-usability and re-configurability of PoCo modelling components and PoCo model was assessed. Re-configurability is considered as a necessary condition for re-usability. These two characteristics were essential in ensuring effective management of the modelling task mainly by ensuring that modelling efforts (i.e. modelling component editing) could be capitalised (i.e. component re-use) effectively in order to simplify the implementation of PoCo virtual prototypes (i.e. component and component based model re-configuration).

8-1.3 Level of portability of PoCo models and PoCo VPE software

The level of portability of PoCo virtual prototypes (VPs) and virtual prototyping environment (VPE) was assessed. Portability of VPs focused on the capability to deploy, view, simulate, and interact (i.e. analysis / control) with a PoCo model using VRML enabled (i.e. VRML
plug-in) web-browser software only. The portability of VPE software was assessed based on the capability to re-configure PoCo modelling components parameters, and to modify the composition of PoCo models without the need for specific modelling tools. As highlighted later in this chapter, portability of PoCo VPE tool was directly dependant on the integration of PoCo modelling components (i.e. encapsulating modelling related data and functions) and PoCo VPE software functions (i.e. providing model configuration and composition functions) into a same software object.

8-2 Lamb Technicon Test Machine

Lamb Technicon is one the COMPAG/COMPANION research project’s partner who have collaborated on the development the component-based machine engineering software environment (e.g. the PDE (Process Definition Environment), Human Machine Interface (HMI)) and the Portable Component-based Virtual Prototyping Environment (PoCo VPE) tools. Lamb Technicon has provided a full-scale demonstrator machine used internally at Lamb to evaluate new controls technologies prior to implementation. The same test machine has been used as a case study for the early development phase of the virtual machine prototyping environment implemented in this research.

8-2.1 Machine description

8-2.1.1 Mechanical layout

The Lamb Technicon test machine consists in a single station from a transfer line for cylinder head machining (I4/15 engine programme). The machine is composed from three sub-systems referred to as the Transport sub-system, the Wing-base sub-system and the Fixture sub-system. The machine sub systems’ mechanical layout is shown schematically in Figure 8-1. The Wing base sub-system consists of the machining unit and is mainly composed of an X-Z axis table on top of which a machining actuator is fixed. The Fixture sub-system is composed of a Z-axis clamping mechanism to secure the part on a fixed table. Four sensors allow the correct part positioning to be checked and the fixture sequence completion to be validated. The part transfer / feeding sub-system is composed of a “transfer bar” whose complex kinematics design allows motion along the Y and Z-axes that simultaneously achieves part transfer and positioning on the clamping table. The test machine used for the COMPAG/COMPANION project had only one “wing” fitted, however, depending on the type of engine block being machined and the type of machining operation that needs to be achieved, an additional wing base can be added along with a different the type of machining head.
8-2.1.2 Control layout

The sequential logic control of the Lamb Technicon test machine consists of relatively simple macro sequences, which are briefly described, in Table 8-1. The part transfer / feeding actuator raises a cylinder head, advances over the clamping mechanism and lowers to position the part. Once in position the part is clamped and the machining sequence “cycles” whilst the transfer mechanisms returns to its initial state. Once the machining sequence is finished, the machining Wingbase returns “Home”, the part is unclamped and another part is fed in the machine. It should be noted that the transfer mechanism is designed so that the part feeding and part unloading operations are conducted simultaneously.

<table>
<thead>
<tr>
<th></th>
<th>Raise</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer Raise/Raiser</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer Advance/Return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixture Clamp</td>
<td>Clamp</td>
<td></td>
</tr>
<tr>
<td>Wingbase Home/Cycle</td>
<td>cycle</td>
<td>Home</td>
</tr>
</tbody>
</table>

Table 8-1: Lamb Test Machine operation sequence description

The machine sequential logic control is based on the control architecture defined by the COMPANION component-based machine paradigm. The complete machine control (i.e. system) is broken down into sub-systems, components, elements, logical states and interlocks. Table 8-2 provides the complete description of the component-based machine control.
hierarchy for the transfer sub-system composed of only one component comprising two elements, one describing the actuation along the Y-axis (Advance/Retract) the other describing the actuation along the Z-axis (Raise/Lower). The raise lower element defines four possible states that are Lowered, Raising, Raised, Lowering, two of which are static states. The transitions between states are enabled either by a position feedback signal (i.e. end travel sensor signal) or by a logical interlock (i.e. typically defined as Boolean conditions (& statement) that are dependent upon other system sub systems/components/elements states.

<table>
<thead>
<tr>
<th>Subsystem 1: Transfer Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
</tr>
<tr>
<td>Transfer</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transfer Subsystem - Transfer component - Transfer Raise/Lower element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowered (\rightarrow) Raising</td>
</tr>
<tr>
<td>Transfer Advance/Retract (\rightarrow) Retracted &amp; Fixture Control (\rightarrow) Unclamped &amp; (all) Wingbase Home/Cycle (\rightarrow) Home &amp; (all) Wingbase Ready/Depth (\rightarrow) Depth</td>
</tr>
<tr>
<td>Raised (\rightarrow) Lowering</td>
</tr>
<tr>
<td>Transfer Advance/Retract (\rightarrow) Advanced &amp; Fixture Control (\rightarrow) Unclamped &amp; (all) Wingbase Home</td>
</tr>
</tbody>
</table>

Table 8-2: Lamb Test Machine Transfer sub system, Transfer component, Raise/lower element hierarchy, Raise/Lower element's state and interlock description.

The modelling of the Lamb Test Machine was conducted during the early stage of this research. At this point, the goal was to understand the approaches to machine modelling / prototyping adopted in other research projects and to gather a set of requirements from the industry that could be used to draw a list of specification for a new type of virtual machine prototyping environment.

### 8.2.2 Primary approach to VPE implementation

The early phase of this research was focused on the development of a set of software modules which functions were related to the implementation of a virtual machine prototype (i.e. 3D computer-based model). The functionality and interfaces with the user and external software (e.g. PDE, HMI) were designed to support the mapping between: i) a 3D geometrical and kinematic models of the machine mechanical layout, ii) a component-based description of the machine logic and iii) cinematic parameters such as actuators speed and position. The mapping between these three main types of modelling data was required to implement a fully functional and dynamic 3D machine model that could be used for machine prototyping. This type of software, designated here as "mapping environment" is characteristic of most research projects relating to the prototyping of manufacturing systems [8] [97] [7].
As shown in Figure 8-2 a “Mapping environment” allows modelling data (e.g. machine 3D geometry and kinematics modelling) to be mapped to control data (e.g. sequential machine logic) in order to edit a 3D-based computer model that exhibit real machine dynamic and behavioural characteristics (i.e. virtual machine prototype).

8-2.2.1 3D modelling environment

The original 3D machine geometry and kinematic modelling environment (i.e. 3D modeller) was developed as part of the COMPANION engineering tool suite (Qin [132]). The modelling tool was implemented using Sun Java Technologies, and models were based on the Java3D modelling formats. The 3D modeller output consisted of structured VRML files containing specific tags (meta tags) defined within the VRML code as comments (‘#’ line header in VRML). Note that the use of the Virtual Reality Modelling Language (VRML) was made since model portability and enabling the interaction between globally distributed partners was essential. The approach to tagging VRML files was similar to the concept behind XML (X-Mark-up Language) enabling the code (or document) content to be decoupled from the code (or document) structure. The ‘#’ tags placed in the VRML file were used to define a higher level VRML file structure above the VRML file structure (based on the VRML scene graph). Meta tags ‘#TagTypeName’ where placed in the VRML file so that the VRML file could be imported and parsed using the mapping environment. The parsing consisted in locating relevant nodes or group of nodes and in retrieving or setting the value of various parameters (e.g. naming, model kinematics and dynamic parameters) required to configure the model.

Figure 8-2: Primary approach to the implementation of a virtual prototyping environment for manufacturing system
It should be noted that modelling environment and VPE could have been directly integrated, in order to enable complete VRML files with appropriate structure (VRML scene graph) and referencing (VRML nodes naming) to be generated. However, such approach would have imposed the use of a specific modeller (namely the 3D modeller developed as part of the COMPAG/COMPANION tool set). Because it was foreseen that the integration with commercial CAD packages was essential, such approach was not considered. The use of meta tags '#' to provide structure above that of VRML was investigated in order to evaluate the possibility to post process CAD-to-VRML file output, or to implement a CAD/VRML plug-in that could directly generate “#” tagged VRML files.

8-2.2.2 Process Definition Environment (PDE) logic editing

The development of the Process Definition Environment (PDE) software was not part of this research. However, the output from this tool was used implement virtual machine prototypes logic related data. The PDE environment provides a suite of design tools that enable system engineers to specify, implement and test component-based machine sequential logic control. The various stages of the machine application lifecycle (namely application design / assembly, configuration, simulation, debug / analysis and installation on destination control nodes) is supported by a set of computer-based graphical visualisations that provide executable representations of machine logic that are easy to interpret and to interact with.
Figure 8-4 shows a screen shot of the PDE software’s logic editing environment, which provides a view of a complete machine (system) hierarchical decomposition into sub-systems, components, elements and a representation of each element as a state based diagram and associated transitions. The PDE software environment also provides a run time environment that allows complete system logic to be simulated, tested and debugged. Figure 8-4 shows both the PDE logic editing and simulation user interfaces. This selection of screen shots show the various machine logic representation as a subsystem/component/element hierarchy, state based logic diagrams (STD), and timing diagrams (TD) which describes the machine interlock and state progression sequence as a function of time.

The PDE software environment is complemented by a “broadcaster” that enables real time events to be exchanged between various software modules that compose the COMPANION machine-engineering environment, or between the design environment and the real system, (in this case the broadcaster provides an interface between shop floor and TCP/IP LAN networks). The broadcaster therefore real machine state to be used and visualised using either the PDE State Transition and Timing Diagrams, a Human Machine HMI, or a 3D model of
the machine. Similarly, the HMI or PDE can be used to drive the real system or a 3D model of the system.

8.2.2.3 Virtual Prototyping Environment

The focus of this research was to develop a VPE tool that could enable the mapping between i) the machine 3D geometry and kinematics data contained in a 3D model of the machine, ii) the machine control data describing the machine behaviour, and iii) machine cinematic data actuators timing and speed. The VPE was therefore designed as a software tool providing the functions required for the various types of data to be integrated into a virtual machine model that could exhibit the same dynamic and behavioural characteristics as the real system. This approach is characteristic of many academic and commercial VIT development projects intended to be used in an engineering context; such VPE’s do not directly provide the functions required to edit the modelling data. Instead, VPE development focuses on the translation, mapping and integration of real system data into a complete machine model.

Figure 8-5 shows a screen shot the mapping environment developed in the first phase of this research and modified in the following phases. The mapping environment supports the import of VRML files (left part of the GUI shown in Figure 8-5) generated from the 3D geometry/kinematics modelling software module described earlier (see Figure 8-3). The mapping environment’s functions were designed to:
- i) parse the ‘#’ meta tags placed in the VRML file in order
- ii) retrieve the model structure (i.e. the components that compose the complete machine model).
- iii) to detect the model parts defined as dynamic (i.e. those parts of the component to which logic state are associated), and
- iii) to retrieve default kinematic and cinematic parameters (e.g. translation/rotation axis, default initial position) associated with these parts.

Similarly, the mapping environment provides functions to access/retrieve/write the machine control data (right part of the GUI shown in Figure 8-5) stored in the PDE database.

The data mapping functions and interfaces (centre part of the GUI shown in Figure 8-5) enable the user to select a VRML component and one of the dynamic primitives (e.g. “clamp” in the example shown in Figure 8-5) that belong to this component, as well as one element that compose a logic component. Each state of the logic element (e.g. opened/closing/closed/opening in the example in Figure 8-5) can be mapped to a position (for static state) or time (for dynamic state) in order to complete the behavioural description of the machine component. The result of the mapping of 3D geometry / kinematics, logic and cinematic data can be visualised using the 3D component view. The mapping environment output consisted of a VRML file containing JavaScript code embedded in the VRML code, plus extra ‘#’ tags appended at the end of the file containing information relating to the 3D geometry/kinematics and logic mapping so that the same VRML file could be re-imported and re-configured afterwards. The JavaScript code provided an interface between the internal VRML dynamic display mechanisms and a Java applet interface to the PDE broadcaster events, so that the model behaviour could be driven by the PDE simulation engine or any other event source linked to the broadcaster (e.g. real machine system or HMI).

8-2.3 Critical analysis

This first phase of the implementation of a VPE was targeted at investigating the approach generally adopted in the academic domain and the modelling software industry to the implementation VPE software. The initial VPE environment development approach described above has allowed several important aspects of such software design to be highlighted:

8-2.3.1 Use of common model

The component-based model (i.e. the common model) was used in the COMPAG/COMPANION machine design software in two different ways. Firstly, the
integration of the 3D modelling and VPE (mapping) environments was achieved by using a set of ‘#’ Meta tags to structure the VRML file containing the machine 3D geometry and kinematic modelling data. The ‘#’ meta tags therefore defined a common data structure, which was used to achieve data exchange between two software environments (i.e. 3D modelling and VPE (mapping) environments). This approach allowed the disparities in modelling formats translation to be overcome. For instance, it was expected that any CAD software that could output tagged VRML file could be directly integrated with the COMPAG/COMPANION tools. However, i) VRML files structured with ‘#’ meta tags needed to be generated and ii) parsed and interpreted. One major downside of this approach (see Chapter 7) is that both software functions (and integration interfaces) need to be updated every time a change or update of the ‘#’ meta tags convention is conducted. This type of integration was described by Vernadat [38] as “loose integration” since, despite the fact a common ontology is used to support data exchange, the software (the modelling environment in this case) interpret this information according to their own data semantics.

Secondly, the integration between VPE and PDE logic editing environment adopted a different approach. Both VPE and PDE software were developed around a common SQL database holding the component-based machine logic description. It was found that this integration approach was much more straightforward since no third party data formats were involved and that database access mechanisms were standardised (i.e. via the use of OLE and the CORBA software integration infrastructure). It should be noted that the PDE and 3D modelling environments could have been integrated in the same way, therefore allowing the generation of a VRML file of a complete machine model (including sequential logic, 3D modelling data and data mapping) directly from the modelling environment. However, this approach would have effectively consisted of merging 3D modelling and a mapping environment, hence irremediably coupling the VPE with the use of an “in-house” proprietary modelling environment. As mentioned earlier, this approach was not developed further since constraining the VPE user to adopt a modelling environment whose functionality was far below the existing CAD modellers typically used in industry was not a viable solution.

8-2.3.2 Model reusability

Model reusability is defined here as the potential to modify a machine in response to a change in machine configuration or to use an existing model as a basis to develop a new machine model. Model reusability is considered in this research from two perspectives:
The capability to re-configure the model behaviour by either modifying the modelling data describing the behaviour, or by modifying the mapping between the sequential logic and 3D geometry / kinematic data

- The capability to change the model 3D geometrical and kinematic layout, or to reuse parts of the model to compose another

Re-configurability concepts often associated with the re-configurability of the control software used to describe the behaviour of a certain type of manufacturing system. The first approach to the VPE implementation provided a satisfying level of re-configurability. For instance, modelling components could be re-imported in the VPE mapping environment, so that the mapping parameters between VRML component's dynamic primitives and PDE logic elements (e.g. number of state of an actuator, dynamic parameters associated with each state or state transition) could be changed. Such mechanisms allowed the re-configuration of individual modelling components, and of complete machine's behaviour to be largely simplified. The whole model re-configuration process was hidden from the user, so that no code editing was required. At this stage, the re-configurability of machine prototypes control behaviour allowed effective machine prototyping and enabled various machine configurations to be tested / debugged quickly.

However, model reusability should also enable the re-use and re-configuration of model's 3D geometry and kinematic data. In this respect, the performances of this early version of VPE were poor. One cause of problem was the limitation imposed by the 3D modeller, which functionality and interfaces were designed for model editing, but modifying existing models was difficult and error prone. In addition, the mapping between modelling data (i.e. 3D and kinematic modelling data), dynamic data (e.g. timings, position) and control logic data (PDE control data) needed to be re-conducted every time the value of one of those parameter had to be changed. The goal for subsequent developments was therefore to improve the reusability and re-configurability of complete modelling components (i.e. resulting from mapping between modelling and control data) rather than achieving reusability and re-configurability of separated data objects (e.g. logic components (PDE), modelling components (3D modeller)).

1 The Flexible approach to Manufacturing Systems design (FMS) and the type Virtual Prototyping Environment supporting the lifecycle of those systems, is reviewed in details in chapter 3
8-2.3.3 Model portability

Model portability focuses on the ability to access and use a virtual machine prototype remotely, and to a lesser extent, outside the core engineering environment. Any virtual machine prototype edited using the VPE was therefore portable since the associated VRML file could be viewed on any machine with a web browser installed. However, the model behaviour (i.e. dynamic display sequences and sequential logic) was not implemented as part of the VRML model itself, and a link to the broadcaster was required (including the compiled Java code providing the interface with the VRML embedded JavaScript code) in order to obtain a fully dynamic model exhibiting the real machine behaviour. When the PDE simulation engine, Broadcaster and 3D VRML model were located on the same machine, the real-time performances were good (no delays between the PDE logic run time engine and the VRML model displays sequences). However, when PDE and VRML model-viewing environments were remotely connected, delays and inconsistencies the PDE simulated machine states and the VRML model actuators’ positions appeared because of the network latencies. Synchronisation mechanisms were implemented in order to ensure that both PDE machine state and VRML model state were consistent in time. However, this resulted in jerky 3D display sequences that significantly reduced the model usability for prototyping purposes.

Figure 8-6: Screen Shot of the POE logic simulation environment with an embedded view of lamb test machine virtual prototype

8-191
In addition, although the models were potentially viewable remotely, the interfaces effectively allowing the user to control the machine logic (e.g. step-by-step, automatic, manual simulation modes) were part of the PDE so that a user could only interact with the model if they had access to the PDE software. Otherwise, no “viewer / model” interactions (apart from model navigation control embedded within VRML enabled web browsers) were possible. Figure 8-6 provides a screen capture of the PDE run time environment with all machine views (i.e. timing diagrams, state based diagrams, and 3D machine view). At this stage, virtual machine prototypes were a complementary machine view that could be remotely viewed. However, because the full functionalities of the 3D machine prototypes relied on events generated by external PDE software, the overall model portability was poor.

8-3 Krause Test Rig

The Johann A. Krause GmbH assembly machine building company was another core partner in the COMPAG/COMPANION research projects. Krause provide another industrial based full-scale test rig that was used to evaluate the concept of distributed component-based machine control and the COMPANION machine engineering software tools from an assembly machine developer's perspective. The modelling of the Krause test rig has therefore provided an additional case study to investigate further the application of the component-based paradigm and to develop the corresponding VPE.

8-3.1 Machine Description

8-3.1.1 Machine mechanical layout

The Krause test machine consisted of two sub-systems, which were respectively labelled as assembly and transport sub-systems. The transport sub-system consisted of a motorised conveyor belt allowing pallets to be diverted to one of two possible paths. Stops and diverter actuators were placed along the conveyor belt to control and redirect the pallet flow. One portion of the assembly sub-system was designated as a “station” or feeding unit whose stop actuators’ layout (i.e. stop and pre-stop) allowed the passage of pallets to be controlled. Two radio frequency tag readers / writers were located along the conveyor belt to read or write pallet description data onto an ID device mounted underneath the pallets. Pallets could be stopped underneath the assembly sub-system. The assembly sub-system consisted of a “Pick and place” unit. A gantry actuator, composed of two servo drives for the Z and Y-axis motions, a pneumatic gripper and an ultrasonic sensor for checking the availability of parts allowed parts to be picked from a stock and parts to be placed on the pallets and vice versa.
8.3.1.2 Machine control layout

The overall machine mechanical and control layout was typical of assembly lines where several part routes and associated units are defined in order to achieve the assembly of several product variants. In the Lamb Test Rig, parts were transported by pallets that were conveyed to the feeding unit where a gate composed of pre-stop and stop actuators allowed part to be stopped above the RF tag reader/writer. The pallets were identified and loaded or unloaded accordingly with parts from or to the stock table, using the pick and place unit. Once loaded/unloaded, the pallet's ID was updated and the pallet released. The pallet was stopped again before the conveyor diverter actuator by a stop actuator. The palette ID was read, and the pallet was routed accordingly by the diverter actuator. More stop actuators (and associated sensors) were used to control the flow of pallets to avoid any collisions at the point where both conveyor paths converged.

The sequential control describing the pick and place unit's actuation sequence was not fully deterministic and relied on a memory buffer holding a description of the parts' stock content to drive the pick and place unit to the right slot without iteratively checking each slot for part
availability. From a control perspective, this implied that the machine state sequence was not known in advance and that dynamic states (e.g. going to position x) were not pre-defined. This type of case study has lead to the implementation of additional types of logic components (i.e. high level, or virtual components) allowing the user to define higher levels of sequential logic that are aggregations of several lower level component sequences. This in turn has lead to a reconsideration of the implementation of: i) the mechanisms that allowed model behaviour to be achieved and especially ii) how these mechanisms were distributed between the PDE runtime engine and the JavaScript code embedded within the VRML file. It was realised that for complex system behaviour, more complex functionality needed to be embedded within the VRML file, e.g. instead of defining pre-defined display sequence (all possible transitions between part’s stock slots), display sequences management mechanisms need to be implemented in order to dynamically define the gantry motion according to the machine logical state. Such requirements were further developed during the third development phase of the PoCo VPE.

8-3.2 Second VPE development phase

8-3.2.1 VMRL file as reusable modelling code

Unlike the Lamb test machine, which was mainly composed of unique machine components, the Krause test rig was composed from several similar parts (conveyors sections, RF tags, stop actuators, sensors placed along the conveyor). The modular conveyor system was particularly relevant to the Re-configurable approach to Manufacturing Systems (RMS) and highlighted the importance of having modelling components that could readily be re-used and re-configured to generate various machine configurations. As highlighted in the first development phase, ensuring re-usability of both 3D geometry/kinematics and sequential logic data separately could not provide effective re-usability of fully functional virtual models. Mechanisms allowing the sequential logic data to be stored and associated with a newly generated VRML file were complex and resulted in the redundant implementation of some of the VPE modelling/logic data mapping functions as part of the 3D modelling environment.
The second development phase was therefore focused around the concept of the modelling component as a clearly defined and portable VRML code object rather than as a data model used across various software tools. This was achieved by defining separate VRML files for each component that composed a complete machine model. In the first approach, components only existed as re-usable objects within the 3D modelling environment (e.g. reusable PDE database objects, VRML blocks of code defined by ‘#’ VRML files meta tag structure). In the second development phase, the 3D modelling environment was used to generate a library of VRML file “templates” which described the 3D geometry and kinematic layout of machine components. This approach had the advantage of producing individual modelling components (called templates at this stage) into a set of individual VRML files. This simplified the VRML ‘#’ meta tags structure (used only to indicate relevant VMRL scene graph nodes and nodes’ field parameters) and parsing requirements for future template’s configuration / re-configuration. However, in separating modelling components (i.e. instances of templates), the overall structure of a complete machine model (previously defined using the 3D modeller) has to be created later in the modelling process using so-called composer tools. This approach has therefore led to the implementation of a new VPE software module and to the embedding of additional information required to compose a model in the component templates. Relevant examples of such information were the “link points” position, and normal vector 3D co-
ordinates inserted in the VRML files as '#' meta tags values. Link point’s information allowed a machine model’s geometrical structure to be recomposed using the composer tool’s functions.

8-3.2.2 VRML component composer tools

The VPE software “composer tools” module to provide the functionality to:

- Parse the “link points” “#” meta tags and retrieve the associated information
- Compute the 3D geometrical transformation required to compose 3D model components,
- Output a VRML file of the complete machine model whose scene graph structure was such that parent-child relationships reproduce the complete machine kinematic layout.

As shown in Figure 8-9, the composer tool effectively consisted of window-based interfaces that guided the user through the process of assembling two modelling components by selecting two of their “link points”. Geometrical transformations were computed in order to
match the link point positions and to orient components reference co-ordinates so that the link points normal vectors’ scalar product was equal to -1 (1). Additional interfaces enabled the user to visualise component link points, and to fine-tune the final assembly. The assembly process results in a VRML file that consists of the concatenation of individual component’s VRML code, into a single VRML file and scene graph that reproduced the complete kinematic layout (i.e. parent-child hierarchy of components).

8-3.3 Critical analysis

8-3.3.1 Use of component model

The second development phase of this research has highlighted two aspects of the VPE development that needed to be carefully considered in order to ensure that the targets in terms of model portability and re-configurability (and reusability) could be reached. The most fundamental domain of development was i) the approach adopted to realise modelling components. Secondly, ii) it was necessary to focus on the design and implementation of the software infrastructure that allowed the component-based approach to machine modelling to be exploited optimally.

The two development domains clearly appeared to be linked. The second VPE development phase has shown that designing components as separated VRML code objects required the development of additional software modules (composer tools). However, it appeared that well-defined modelling objects (i.e. components as VRML code objects) also resulted in a simplification of the software associated with the configuration and composition of those objects. This has allowed to clearly dissociating the modelling components lifecycle (i.e. template editing, instantiation and configuration), from the component-based system lifecycle (i.e. component geometrical assembly). The general conclusions drawn from the second development phase, has lead to a radical change in the way modelling components and the associated VPE environment were implemented. The new approach to modelling component design consisted in encapsulating more functionality and information within well-defined code objects. Similarly, the software architecture required to support the components and component-based system lifecycle, migrated from monolithic software tools to simpler software modules which functions were specific to a particular phase of the components, and component-based systems lifecycle.

---

1 More details about the assembly mechanisms and associated mathematical transformations can be found in chapter 5
8-3.3.2 Model reusability

Model reusability has been defined as the capability to re-configure the model behaviour (re-configurability) and to change the 3D geometry and kinematics layout by re-composing existing modelling components. Compared to the initial approach to the VPE development, the model re-configurability was not significantly increased. The integration mechanisms with the PDE database were the same in both cases the first and second development phase, and the process allowing 3D modelling, logic and cinematic data to be mapped was similar. The noticeable differences were concerned with the VRML file parsing functions, which were implemented as part of the VPE. These were made significantly simpler in the second approach because the files only contained the VRML code and `#' Meta tags describing single machine components rather than a complete machine models.

The second development phase also marked a change in the use of virtual prototypes. In the first development phase the use of 3D machine model was generated to validate machine logic previously edited using the PDE environment. In the second development phase, the use of 3D machine model was made during the logic editing process. Under this process, 3D machine model were used to support the machine logic editing process as well as to validate machine logic configurations. At this stage, the benefits of having a simplified mapping environment (resulting from the better definition of modelling component as code objects) were evident (i.e. the mapping environment could be compiled as a simple ActiveX software component which could be activated by the PDE during key phases of the logic editing process).

The major change that characterised the second approach to the VPE implementation was to dissociate templates (i.e. modelling components without logic data), components and complete machine as well as the tools and process required to edit and configure those modelling objects. This separation has allowed limiting the use of the 3D modelling environment to the template editing only and therefore to simplify further the integration of the VPE with third party modeller. However, the VRML files describing complete machine model were still obtained by merging the code of all components into a single new VRML file. The modification of such file (i.e. re-configuration / change of a complete machine model) was therefore still prone to problem and required the parsing of very large files.

8-3.3.3 Model portability

In the domain of production system virtual prototyping, model behaviour is often supported by an external environment whose output events are used to drive the 3D model. In the context of this research, this was materialised by the link between the PDE logic editor and
the VRML model using the broadcaster interface. Such approach limits the portability of machine prototypes since without links to the (proprietary) broadcaster the virtual model is no more than a 3D display of the machine geometry.

Whereas for the Lamb cases study, all of the actuators’ states and state sequences were clearly pre-defined, for the Krause test rig’s Y-axis gantry actuator, neither the state sequence, nor the state to state transitions, were pre-defined due to the dependence of the actions on the output of the tag reader. From a modelling perspective, this meant that functions, allowing the part motion parameters to be dynamically computed, needed to be implemented as part of the model itself. Whereas the Lamb case study only required the implementation of pre-defined display sequences that could be triggered by broadcaster events, the Krause test rig prototyping required significantly more complex functions to be embedded within the modelling components, i.e. implemented within the VRML code. At this point in the research process, the potential of VRML in implementing more complex, autonomous, and to a certain extent “intelligent” modelling components (relatively to what is usually referred to as a component in the domain of 3D modelling), became the main focus of the research. The Krause case study has therefore led to the design of what would effectively be the equivalent of the PDE logic run-time engine, and which could be implemented as part of the modelling components themselves. This functions required to simulate machine logic (sequential logic engine) needed to be distributed across modelling components. If implemented using exclusively web compliant technologies such as VRML, highly functional and completely portable virtual prototypes that could run without the need for central and external control system (e.g. the PDE logic simulation engine and broadcaster, in the case of this research) could be implemented. The programming capabilities offered by VRML and the investigation of previous research on highly functional VRML code objects [96] [120] has reinforced this idea. However, the challenge was to maintain a good level of re-usability and configurability whilst at the same time implementing potentially complex modelling elements.

8-4 Ford Test Rig

The Ford Test Rig has been provided by Ford’s Automation Group to Loughborough University as a platform used to test new control paradigms. In the context the COMPAG/COMPANION research project, one of these test rigs has been used to implement and test the concept of distributed machine control logic, and has provided an additional case study to develop further the VPE and the concept of portable modelling components.
8-4.1 Machine Description

8-4.1.1 Mechanical layout
The Ford Test Rig (see Figure 8-10) is a small-scale test rig, composed of a part feeder and a part transfer unit consisting of a transfer arm equipped with a pneumatic gripper, a part lift and a linear conveyor. This part transfer sub-system is used to convey a part to an indexing table, which successively (by rotating the table) presents the parts to a drilling station, a part testing station to check the drilling operation and to a part unloading station. At unloading a pneumatic gripper mounted on an X-Y-Z axis gantry actuator can pick up parts and deposit them in either a "good" part stock or a "rejected" bin slot.

8-4.1.2 Machine control layout
The Ford Test Rig control layout has been decomposed into two sub-systems: the conveying (OP1) and the indexing (OP2). The part is first routed through the OP1 sub-system. The feeding unit is composed of a feeder and part presence sensor, which push the part being fed in the system to a transfer arm component. The transfer arm transfers the part to the part-conveying unit composed of a lift and linear conveyor. When the lift actuator's part presence sensor is set, the part is lifted and fed on to the conveyor. A part release mechanism, composed of two stop actuators (acting as a gate) and associated part presence sensor, control the feeding of part from the conveyor unit to the rotary indexing unit. The part is then indexed to the drilling station where it is clamped and drilled. The next indexed position consists of a probe checking the depth of the "drilled" hole. The next indexed table's position presents the part to the part-unloading unit, which, depending on the depth of the hole, either discards the part in a reject bin slot, or feeds it into a part stock slot. It should be noted that between the part's hole probing and the part unloading sequence (index table positions 3 and 4) a purely logical component (with no mechanical instantiation) determines whether the part is either a reject or a good item.
8-4.2 Third VPE development phase

The third development phase has marked a radical change to the design and implementation of VPE. This phase has focused on reviewing the way the concept of a modelling component could be practically implemented. The approach adopted was to consider modelling components as highly functional software objects rather than data objects, and to investigate:

- to what extent the functionality provided by the software that is required to implement a component-based machine 3D virtual prototype (i.e. 3D modelling and modelling/logic data mapping environment, modelling component configuration and composer tools, logic emulation engine, user interfaces) could be encapsulated within the modelling components themselves,

- to what extent the programming capabilities provided by VRML features (i.e. nodes, scene graphs, embedded JavaScript, and VRML code interfacing to Java programs), could effectively support the implementation of these potentially complex functions, and therefore to investigate the feasibility of implementing highly portable and autonomous yet highly functional modelling components.

| Figure 8-10: Overview of the Ford Test Rig mechanical layout, and screen capture of the corresponding 3D model |
In the early phases (i.e. phases 1 and 2) of the VPE development, the focus was placed on the way various software tools could be integrated in order to incrementally implement and manage data objects describing machines composed from component virtual models. Components' functions were at this stage relatively simple, since most of the functionality required to edit, configure and control the model were implemented as part of the PDE software tool (e.g. model logic emulation was provided by the PDE simulation engine). The third VPE development phase is characterised by a radical change in the way modelling components were conceived. The concept of component, initially materialised by "#" tags structure, and then by individual VRML files describing model geometry and simple display sequences, was to be replaced by pre-defined, highly functional, reusable and re-configurable modelling objects.

8-4.2.1 General approach to modelling component

The third development phase was therefore characterised by a different approach, which consisted in using pre-defined software components, referred to as component templates providing modelling functions whose parameters are configured depending on the system to be modelled using various software tools (i.e. 3D modeller, logic editing environments). Software tools involved in the implementation of a modelling component were therefore relegated from component editing to component configuration tools. The goal was to reduce the role of software tools required for the implementation of machine virtual prototypes to simple template parameter configuration, rather than complex VRML file editing, as it was the case in the first and second approaches to the VPE implementation.

Figure 8-11 illustrates the approach, which consisted of focusing on the implementation of pre-defined and highly functional modelling templates. One of the main tasks conducted in the third development phase was to investigate the extent to which VRML could be used as a programming language that could support implementation of functionality beyond the domain of 3D system modelling. The development of modelling templates was focused on implementing three type of pre-defined functions which were related to i) the modelling of kinematic links between parts, ii) the emulation of sequential logic driven system behaviour, and iii) the composition of modelling elements into a complete machine.
8.4.2.2 Pre-defined model kinematics layout

In the previous development of the VPE implemented in this research, the model 3D geometry and kinematics were generated using a 3D modelling environment specifically developed as part of the COMPAG/COMPANIION suite of tools. However, as a result of demonstrations and interviews conducted with collaborators it became obvious that many industrial partners were concerned about the need to adopt additional modelling software. The lack of integration between CAD and VPE was an important drawback for the original VPE from industrial perspectives. Extensive investigations on CAD modeller VRML file output has shown (see Chapter 7) that the post processing of VRML files in order to retrieve information on the initial model kinematics is difficult, time consuming and sometimes impossible due to the lack of support from CAD developers for the VRML standard. An alternative approach to achieve integration between the VPE and CAD modeller would have been to collaborate directly with CAD developers to determine a VRML file output structure. However, this approach was not practical within the timescale of this research due to the multitude and variety of CAD environments currently used by the industrial consortium (e.g. CATIA, ProEngineer, AutoCAD).

The approach adopted was therefore to implement the model kinematics as part of the data pre-defined by the modelling templates. This meant that each modelling template contained
the kinematic definition of the mechanical machine construct (e.g. machine mechanical module) it represented. This approach required part of the model being implemented during the template editing phase, but allowed the role of 3D modeller in the virtual machine model composition to be reduced to the editing of 3D geometry data parameters. This solution was clearly beneficial, considering the issues such as CAD assembly translation, CAD model / virtual prototypes modelling level of detail requirement\(^1\), and the fact that 3D geometry modelling is the part of virtual prototyping data editing that requires specific tools such as CAD. Model kinematic editing using VRML pre-defined nodes is relatively simple\(^2\) by comparison. Modelling component templates therefore provide pre-defined kinematic layouts between reference co-ordinates to which no geometry is associated. The geometry can be imported from a CAD VRML 3D geometry output. This approach therefore allowed, at the expense of an additional template editing phase (i.e. implementing model kinematics), integration with any modeller able to output 3D geometry in a VRML formats (i.e. with virtually any 3D modeller currently available) to be achieved.

\[8-4.2.3\] Pre-defined sequential logic simulation functions

The third development phase was focused mainly on the implementation of a "logic simulation engine" whose functions were distributed across modelling components. This was essential for decoupling both the VPE and virtual machine prototypes from the machine engineering software environment (i.e. the PDE), so that virtual prototypes could be used without the need for real-time links to the PDE logic simulation engine. The implementation of a logic simulation engine that did not require central control effectively consisted of reproducing the machine distributed control paradigm investigated in the COMPAG/COMPANION project and linking the machine logic emulation to the 3D model dynamic display functions.

The VRML files materialising pre-defined component templates typically consisted in three main parts, which were:

---

1 The level of modelling detail that characterise CAD models is not adapted to the implementation of virtual prototypes of complete machines. This practically means that CAD models cannot directly be used to implement machine virtual prototypes and a CAD model post processing phase is required. More details about this issue are given in chapter 7.

2 Virtually all kinematics joints types can be modelled as the composition of rotary and linear links. This makes the modelling of machine cinematic relatively simple. The real issues in implementing CAD and VPE resides in the modelling of 3D geometry which requires complex modelling functions and interfaces adapted to the domain of engineering, and specific to CAD software.
- i) the VMRL nodes, scene graph structure and JavaScript code required to achieve the dynamic display of 3D geometry according to the kinematics and cinematic configuration data
- ii) the JavaScript code implementing the sequential logic emulation functions
- iii) a set of interfaces allowing a) internal event routing and management functions so that i) and ii) could be co-ordinated, and b) external event communication to and from other components

Component templates were therefore designed as pre-defined VRML code objects containing all the functionality required to reproduce a sequential logic driven machine component behaviour. The same mapping environment interface used in the second phase of the VPE development, was modified and used. However, the mapping environment functions were modified since the component template configuration also included the logical expressions of the conditions associated with the component internal state transitions.

8-4.3 Critical analysis

8-4.3.1 Use of component model

The third development phase has focused on implementing the component as a highly functional modelling object, a software component¹ (whose functions and attributes were the virtual equivalent of a real machine mechatronic component), rather than a simple data structure or model. This has led to the investigation of the capabilities of VRML in supporting the implementation of potentially complex functions and its capability to provide code reusability that characterises object oriented languages². VMRL features such as advanced VRML scene graph management and PROTO nodes, Java Scripting capabilities and interfacing with external authoring interface (EAI) were investigated in more detail³.

This approach to modelling components has also allowed the three phases described in Figure 8-11 as component editing, component configuration, and component-based system (virtual prototypes) composition to be cleanly dissociated. This was essential:

¹ Please refer to chapter 4 for a review of the concept of component in the domain of software engineering.
² More details on the Object Oriented paradigm and the various mechanisms defined to ensure code reusability are reviewed in Chapter 5.
³ The specificities of the VMRL Proto node and the way it has been adapted (Object oriented VRML) and used to implement reusable modelling components, are introduced in Chapter 5.
to ensure consistency between the virtual system and the real system. It has been shown that the value of a VPE to support the design lifecycle of Re-configurable Manufacturing Systems (RMS) is strongly dependent on the capability to use VPE tools to implement modelling components with the same level of reusability, re-configurability and general architectural characteristics (e.g. level of granularity) as the real system components.

reduce the complexity of software functionality related to the modelling components' configuration. The new component design has allowed templates to be designed as pre-defined code objects. The most relevant example of this is the fact that 3D modelling environment functions related to the editing of model kinematics are no longer required. As a direct consequence, any 3D modeller could be used to edit the component geometry, which could be defined as a component parameter. In addition, functions such as link point component editing are directly implemented as part of the component functions, which means that once the 3D geometry is configured, no additional software is required to define link points.

8-4.3.2 Model portability

Because all modelling functions were implemented using exclusively VRML and embedded Java Script code, model portability was at this stage very good. Simulation of machine prototypes behaviour did not require any link to external logic simulation engine, so that machine models could be transported from computer to computer by simply copying VRML files, and only required a VRML-compliant web browser to be run. In addition, at this stage component could be re-configured by simply changing text strings parameters of very simple VRML files, which could be done manually, or using simple user interface. This practically meant that modelling components or complete machine models behaviour (e.g. logic interlock, state position, state transition speed and associated condition) could be modified using configuration tools which functions were very simple. This directly translated into portability of the model configuration tools themselves, which, because of their functional

---

1 The fundamental issue of ensuring consistency between real and virtual system component architectural characteristics and composition process, in order to ensure the effectiveness of VPE to support RMS prototyping, is highlighted in Chapter 3

2 This function allows the user to click on a 3D geometry and to directly gather the point’s position and normal vector coordinate, expressed in the reference coordinate required to achieve component assembly. Please refer to Chapter 5 for details about the link points feature and the component’s geometrical assembly functions.
simplicity, could be implemented using any languages and according to any user/developers preferences (e.g. user interface layout, modelling process constraints)

At this stage of the development, one of the major barriers to achieve full modelling environment portability was related to the fact that the functions allowing component geometrical composition was still supported by external modelling tools (rather than supported by modelling component internal functions). In the same way, the editing of event routing paths to support inter-component communication of logic related event was still supported by external software functions. Therefore, although modelling environment portability was good at this stage, it could further be improved by integrating components' geometrical and logical composition functions as part of the modelling components.

Another aspect of VPE that was considered was the implementation of portable model user interfaces. In previous versions of the VPE, 3D virtual machine models were principally used as a complementary view for machine engineering tools, which provided user interfaces to interact with the machine control (e.g. PDE logic run time, or HMI interfaces). However, user/model interfaces that could enable the use of stand-alone models were required. Note that the type of interactions mentioned here relates to model usage i.e. control of model's behaviour, possibly some additional navigation capabilities and access to machine component information through model interaction. However, the possibility to implement modelling component editing functions and interfaces as part of the component functions was also considered.

8-4.3.3 Model reusability

8-4.3.3.1 Reusability and re-configurability

Model re-configurability is defined as the ability to re-configure the data defining the model geometrical and behavioural characteristics. In this respect, modelling templates provided good model re-configurability since the possible transition sequence between states, as well as the condition associated to these transitions and the position and time parameters could easily be modified using the mapping environment. However, it was found that modifying the number of state required the VRML file functions and structure to be partially re-edited. This issue was avoided by defining a large number of pre-defined states, some of which were configured with void parameters if not required. Although such solution was perfectly practical, it was not particularly elegant. Modelling component reusability, which is defined as the capability to reuse existing modelling component to compose a new model, or to modify an existing model composition, was very good. The use of VRML Proto nodes has
resulted in much simpler model file structure, which made the parsing and modification functions implemented as part of the composer tools much simpler and usable. However, at this stage, the composer tools functions were still relatively complex since in addition of editing the final component-based model VRML file, composer tools were required to compute geometrical transformation based on link point parameters, and to create the event routing that allowed a given component's state change to be routed to components which transition were depending on.

8-4.3.3.2 Openness

Openness is defined here as the ability to extend the modelling functions of the PoCo modelling environment. During cases study, it was foreseen that the modelling of machine control other than state based machine logic (e.g. continuous control such as NC) would require completely re-designing and re-editing the entire set of modelling components. For instance, when component templates were used to implement the Krauser Test Machine, which gantry actuator's control logic was characterised by a non-deterministic sequence of states, modifications of an existing component template were difficult to manage because of all component functions where implemented in a single VRML object (materialised by a single VRML file). In the same way, for the implementation of a purely logical component, such as the one required by the Ford Test Rig machine, only part of a component VRML code would need to be reused, and yet a complete component template need to be re-design and re-edited. The reusability and changeability of individual component function was therefore limited, because, unlike component-based model, component template code was not structure properly, which ultimately, translated in poor openness.

It was realised at this point that the component template were designed as highly functional and complex modelling objects, which resulted in equally complex VRML code. Until this stage, the focus was placed on the re-configurability and reusability of the overall model (which has led to the decomposition into modelling components). However, with the implementation of increasingly functional and complex templates, the issue of the reusability re-configurability of individual components functions emerged as essential in ensuring the value of the component-based modelling approach as a more generic modelling tool.

8-5 Asda supermarket warehousing machine

The partnership between the MSI Research Institute and Asda Lutterworth and Blackmills distribution centres emerges from the will to extend the scope of application of the component-based approach to distributed machine control and modelling, outside the domain
of automotive industry. It is believed that the potential advantages of applying the component-based control paradigm to the automated warehousing systems may be even greater than in the case of machining or assembly systems. Automated warehousing systems are composed of a narrow range of relatively simple machine devices such as conveyor sections, sensors, pushers, which are composed in various ways depending on the requirements. From a modelling perspective, such systems are ideal to test the re-usability, re-configurability, and composability of modelling component and component-based models.

8-5.1 Machine description

The system used to investigate the potential of 3D virtual machine prototyping in the domain of warehousing industry consisted in a simple box sorting subsystem composed of one main conveyor section and two exit conveyor sections. The conveyor layout is schematically shown in Figure 8-13. The flow of boxes on the main conveyor was controlled by stop actuators and part presence sensors. Pusher actuators were used to direct box to either side of the main conveyor on the exit conveyor sections. Sensors were used to control the presence of boxes in various part of the system. A stop actuator (stop1) and associated sensors (1 and 2) allowed the passage of boxes under a bar code scanner to be controlled.

![State Based Diagram](image)

Figure 8-12: Sample of Asda test machine's state based diagram logic control representation

Depending on their bar code ID, the boxes were either stopped in front of the “accepted part” or “part rejected” pushers or directed to the exit conveyor sections. Once in exit position, the boxes were pushed and evacuated out of the conveying system. The interlocks are such that boxes were only released in the exit position when pushers were retraced and no other boxes were present.
Figure 8-13: Schematic representation and virtual prototypes of Asda warehousing sorting sub system

8-5.2 Current VPE development stage

8-5.2.1 Component model

The previous approaches to the PoCo VPE design have highlighted that implemented highly functional VRML objects resulted in poor reusability of components' functions and in difficulties relating to the management and changes of the VRML code describing those functions. PoCo VRML modelling components had reach a level of complexity that required
these component to be broken down into smaller constructs, which are referred to as modelling elements. 

Figure 8-14: VRML Modelling component and internal elements structure. Use of configuration and composition tools during the component-based model lifecycle.

Figure 8-14 illustrates the approach that consists in defining specific modelling elements corresponding to each type of functions that modelling component provide. Each modelling element consists in a specific VRML Proto object that encapsulates the VRML and JavaScript code describing its functions. Each type of modelling element can be instantiated, configured and composed as a specific modelling component. The way modelling elements functions are composed is pre-defined and characterised the component internal structure, which relies on modelling elements interfaces event passing scheme. The process of implementing PoCo (component-based) models therefore consists in three phases which are i) the design of PoCo modelling elements, ii) their configuration and composition into modelling PoCo modelling components, and iii) the composition of modelling component into complete PoCo machine virtual prototype.

8-5.2.1.1 Modelling element editing and configuration

The design and editing of PoCo modelling elements should be considered as a software-editing task. The actual PoCo VPE version provides six types of modelling elements which functions are related to the modelling of sequential machine logic, 3D geometry and

---

1 Please refer to chapter 5 for a complete description of PoCo modelling elements functions.
kinematics modelling, to the composition of modelling component into component-based model, and to the user/model interfacing. A PoCo modelling element VRML file template has been designed\(^1\) which allow extending the set of modelling elements to support the modelling of various types of systems (e.g. continuous instead of state based machine control). Once a library of modelling element (and associated functions) has been implemented, element can be instantiated, (re-)configured and re-used across various model configuration.

8-5.2.1.2 Component editing and configuration

The composition of modelling elements into modelling components is achieved according to the component file model defined as part of this research\(^2\). The way PoCo modelling elements are composed into modelling components is therefore pre-defined so that the composition mechanisms can be fully automated, using relatively simple user interfaces and modelling processes. In the last phase of this research, the attention has mostly focused on encapsulating the functions allowing element composition within PoCo element and component template so that external PoCo object composition can be simplified.

8-5.2.1.3 Component composition

PoCo modelling components have been design to encapsulate as much as it is possible, the functions required to support their composition into a complete model. These functions are provided by specific elements such as LP (Link Point) for instance, which provides the complete 3D transformation that allows components be to be geometrically composed by simply selecting two of their link points. The design of another element (the INTerlock) related to the composition of components’ logic (logic interlock) is being conducted as part of the future PoCo development phase.

8-5.2.2 Model reusability, re-configurability and openness

One of the major changes in the third development phase has been the decomposition of modelling components’ VRML code into reusable modelling elements. The Krause Test Rig has shown that the modelling elements supporting the emulation of machine logic needed to be tweaked in some cases to answer particular requirements. The internal structuring of modelling components has resulted in a very high level of element functions reusability, element re-configurability, and overall modelling function openness. The internal component structure has allowed specific logic modelling elements to be defined and integrated with

\(^1\) The VRML file object model, which has resulted of the use of Object oriented paradigm to structure the VRML code, is described in details in chapter 6.

\(^2\) The nested proto hierarchy VRML file structure, which characterise the VRML component model, defined in the context of this research are described in chapter 6.
other without major change in the component structure of other elements' code. The implementation of other modelling elements enabling parts to be modelled, part counter and are in development. Elements enabling NC control to be simulated would be the next step in order to extend the scope of manufacturing system types that PoCo VPE can be used to model. The reusability and re-configurability of modelling constructs has been achieved at all level of the component-based model hierarchy, from the functions that compose a fully functional and autonomous component, to the component, as well as the component-based models themselves.

8-5.2.3 Model and modelling functions portability

The better structuring of modelling component has allowed increasing the number and complexity of the functions supported by components (and provided by individual elements) objects. The internal component structure has allowed designing highly functional modelling objects. The last PoCo VPE development phase has therefore focused on encapsulating functions previously supported by external modelling software as part of the component functions, by designing specific elements ((e.g. composition of component into complete component-based models, model/user interfacing). For instance, the complex 3D transformation algorithms needed to geometrically compose components and manage assembly parent-child relationship is supported by the LP (link Point) element, whereas the INT (INTerlock) element allowing inter-component logic interlock to be managed is currently being designed.

Figure 8-15 shows the component geometrical composition software that provides a front-end interface to configure component assembly parameters and achieve component assembly. In previous version of the PoCo VPE, the software associated with the geometrical assembly of component was highly complex, consisted of several thousands of line of code and required interfacing with other modelling related software and database. Encapsulating component composition functions within specific modelling elements has allowed the functions supported by such software to be reduced to simple parameters configuration algorithm. The software currently used consists in an executable materialised by 50 line of code for a total size of 800 kb. Although Visual Basic language has been used, the software functions could easily be implemented using virtually any programming language.
Another issue highlighted in the third development phase was the need to have embedded model/user interface in order to dissociate PoCo model from external software interface. This was achieved successfully by defining specific modelling elements enabling the user to visualise, the model logical state, or to drive manually the model and force state. The model therefore implement all interface required for a user to configure, run and analyse the model i.e. to test a particular logic configuration. Overall, portability has been achieved for the PoCo models, which can be run, viewed and tested by engineers who have the various VRML files that compose complete models, simply using a Web Browser.

![Figure 8-15: PoCo component geometrical assembly user interface (link point selection / assembly)](image)

![Figure 8-16: Approach to the implementation PoCo VPE portability](image)
The PoCo VPE has proposed a new approach to the design and implementation of highly portable VPE adapted to the engineering of Re-configurable Manufacturing Systems. As shown in Figure 8-16, the current version of the PoCo VPE is built upon highly functional component, which result from the structured composition of modelling elements. All functions that enable model simulation, user interfacing, model composition and configuration are supported by the PoCo modelling elements functions and the specific VMRL object models they are built upon. The current version of the PoCo VPE has reached a level of reusability functionality and portability that far exceed VPE developed in other research projects.
Chapter 9 Conclusion

9-1 Overview of research objectives

The main objectives of this research were to:

- Conduct a critical analysis of the approaches currently adopted to the design and implementation of Virtual Prototyping Environment (VPE) intended to be used as engineering tool to support the design and change of manufacturing systems MS and in particular of Reconfigurable Manufacturing systems (RMS)

- Identify the aspects of VPE tools that could be improved by adopting an innovative approach to the design of VPE, and, given the time frame and resources available to conduct this research, initiate the implementation and testing of such innovative VPE tool

A review of general manufacturing paradigms (e.g. manufacturing system flexibility and agility concepts), systems and "real world" distributed engineering partner integration has been undertaken in order to have a better understanding of the context in which such tools are deployed and used. This has led to the definition of conceptual requirements that had to be taken in account when designing and implementing VPE tools. At this stage, the research was focused on improving two aspects of VPE tools, namely:

- The functionality of VPEs as engineering tools used to support the engineering lifecycle of manufacturing systems. The design and development of an innovative VPE environment was focused on the virtual prototyping of Reconfigurable Manufacturing Systems (RMS). RMS are built upon so-called flexible technologies (e.g. modular machines, distributed control systems) which implies that machine reconfiguration translates into extensive machine re-design and change. Current "state of the art" VPEs are not adapted to the modelling and virtual prototyping of this type of manufacturing systems and engineering/change lifecycle.

- The functionality of VPEs as communication tools that can be used to support distributed engineering of RMS systems. This aspect of the research was focused on exploiting the potential of the high level of tacit knowledge that intuitive 3D models
can convey. Evidence has shown that currently available VPEs and modelling tools do not take advantage of such potential, which translates into such tools being only deployable in highly specialised software environments and only usable by specialists.

This set of conceptual requirements has been formulated into a set of design specifications used for the implementation of an innovative VPE adapted to the distributed engineering of RMS systems. The implementation of the VPE software environment has been focused on essential characteristics of machine models (prototypes) modelling tools including:

- **Usability**: The primary purpose of the VPE tools is to support the engineering of MS by providing system engineers with virtual machine prototyping tools that can be used to assess systems design and support decision-making processes. Usability is the term used to emphasise the accessibility of VPE tools and virtual models and the simplicity of the modelling task. These features are essential to ensure the value of VPE as engineering tools. The difficulties related to the modelling task need to be hidden from the VPE and model end users.

- **Re-usability**: Re-usability is a general concept that can potentially allow the process of implementing VPs of behaviourally complex and large-scale MSs to be simplified. By capitalising on the modelling effort conducted through various machine prototyping cycles, in the form of re-usable modelling constructs, it is possible to reduce the modelling time and simplify the modelling task. The concept of model construct re-usability is practically achieved by reconfigurability of various parameters that characterise the modelling constructs.

- **Openness**: The concept of openness relates to ability to extend the range of functions that a given system can support without major changes, or to minimise the changes required for the functions of a given system to be extended. VPE functional openness relates to the type of system (and hence modelling functionality) that can be modelled. The VPE developed in this thesis was initially aimed at providing modelling functions for state-based sequential logic driven MSs. It was foreseen that VPE modelling functions would need to be open in order to support the modelling of other types of machine control, and therefore to increase the value of VPE tools as engineering tool.

- **Portability**: Whereas usability and re-usability concepts directly relate to the task of implementing VPs, the concept of portability refers to the deployment and utilisation
of VPE and VPs. Achieving portability aims at decoupling VPE tools and machine models from highly specialised software infrastructure and services. Practically, portability refers to the ability to deploy of VPE tools and use VPs amongst distributed engineering partners using communication infrastructure such as the public internet and web-based technologies.

The last phase of the research consisted of the implementation and testing of a new VPE tool. The design specifications have been translated into a set of design and technological solutions for the VPE implementation:

- **Component-Based approach to VPE software and model design**: the component-based (CB) paradigm for real manufacturing systems' control architectures has been investigated. The focus of the research described in this thesis was to investigate how the CB approach could be applied to the design of VPE in order to achieve the requirements in terms of VPE usability and model re-usability. This has led to the implementation of a CB modelling framework built upon the modelling and model editing/configuration functions provided by so-called *PoCo modelling components*.

- **Web3D based modelling environment**: Portability of both modelling functionality and 3D machine models were an essential requirement for an innovative VPE. The VPE developed in this research has been implemented using exclusively Web3D (i.e. WWW compliant) modelling technologies. One goal of this research was to assess the potential of modelling technologies used to develop general-purpose web-based models to support the implementation of highly specific (engineering) software tools. A large part of the development phase has consisted in transposing a software design paradigm (i.e. object-oriented design) to a Web3D modelling language (i.e. Virtual Reality Modelling Language VRML). By doing so, it was made possible to combine the benefits of effective code management (required to achieve re-usability and re-configurability of *modelling components*) and portability of modelling component and VPE tools.

The research approach presented above has resulted in a VPE labelled as Portable Component-Based (PoCo) VPE. The latest form of the PoCo VPE prototyping framework consists of a set of modelling objects (i.e. modelling elements and components) which can be configured and composed into complete machine prototypes. The PoCo model architecture has provided very high levels of (re-) configurability and (re-) usability of PoCo machine prototypes and ultimately a very high level of functional openness.
9-2 Research Contribution

9-2.1 Innovative approach to the design of virtual prototyping tools as manufacturing system engineering tools

The engineering lifecycle that characterises "well established" manufacturing systems, referred to as Flexible Manufacturing Systems (FMS) (i.e. fixed hardware and fixed but programmable software), is consistent with the typical 3D computer-based model editing process which consists of sequentially implementing of models' geometry (3D and kinematics), models' behaviour (programming code) and models' interfaces (model/user, modelling/engineering environment interfaces). However, such sequential modelling process is not ideally suited to the prototyping of RMS systems, whose engineering lifecycle is characterised by frequent machine hardware, machine control, and software re-design and re-configuration phases.

The approach adopted in this research to implement the PoCo VPE software is radically different from previous research and commercial VPE developments efforts. The emphasis has been placed on the consistency between real and virtual system architectures so that real / virtual system design tools and engineering processes can be integrated more naturally. The approach adopted in this research was to design highly re-configurable and re-usable modelling components as virtual equivalents of the mechanical modules, distributed control nodes and software components from which RMS systems are composed. The emphasis was placed on the design and on the composition process of modelling components as autonomous modelling code objects, rather than on the implementation of interfaces and software infrastructure services that allow a sequential modelling process to be hidden within the composition of reusable data. The approach adopted in this research is unique in the domain of VPE design. The design of re-configurable, re-usable and autonomous modelling objects has been researched elsewhere and in some cases partially implemented. However, the implementation of VPE for production systems engineering has never benefited from such developments. The final PoCo VPE modelling framework outlined in this thesis consists of highly functional modelling components, whose internal structure (i.e. elements) provides a high level of reconfigurability and openness. The benefits of the PoCo environment from a modelling perspective has led to the consideration that such a modelling framework could be used as a platform for the design / development and testing of real MSs' components and component-based structure, in addition to a modelling and prototyping environment used to validate previously developed such systems.
9-2.2 Innovative approach to the design of virtual prototyping tools as distributed engineering and communication tools

Three dimensional (3D) computer-based models used for the design and analysis (i.e., prototyping) of MS provide very intuitive representations of behaviourally complex and large-scale systems. Such potential is highly relevant in a context where system engineers with different background and domains of expertise have to discuss design solutions and exchange views in order to reach a near optimal design solution.

Current organisations operate as “Virtual Enterprises” (VE) and “Virtual Organisations” (VO). In this context the benefits of using 3D computer models is only relevant if such model can be shared between partners who may have very little knowledge of the domain of computer modelling. Existing commercial VPEs still rely on highly specialised and proprietary modelling technologies that restrain the use of 3D computer based model to partners who have knowledge in the domain of computer modelling, and who have access to specific software and hardware infrastructure required to view, edit and simulate computer models.

Typical approach to VPE design | P2Co approach to VPE design
---|---
Modelling of “well established” and FMS systems e.g. NC, industrial multi-axis Robots | Modelling of “component based” Or RMS systems e.g. modular machine, distributed control, etc.
Communication within control IT structures and integrated software environment | Communication in heterogeneous IT/IS context between Virtual enterprise distributed partners
VPE Software component defined by decomposition of software functions according to Modelling functions / modelling data types | VPE Software component defined by decomposition of software Engineering functions into elements types and integration of elements into fully functional modelling component

Figure 9-1: An innovative approach to the specifications and design of manufacturing systems' Virtual Prototyping Environments.

New modelling technologies (modelling language and formats) have recently emerged from the increasing performance and accessibility of Web based services. The so-called Web 3D technologies allow complex 3D computer based models to be deployed over common web based networks, which potentially allow 3D models to be shared by partners having access to a minimal network infrastructure (and therefore communication infrastructure). Because such technologies (e.g., VRML) are freely available, they have been used by a number of research projects aiming at promoting the use of 3D modelling in the domain of MSengineering. Whereas such projects generally focused on achieving portability of the model’s 3D
geometry, the research presented in this thesis aimed at achieving portability of the model geometry, the model logic engine, the model/user interfaces and to a certain extent, the model configuration and editing tools. Cases studies using the PoCo VPE have shown that a very high level of portability for both model and modelling environment has been reached. In addition, effective virtual prototyping can be conducted in a distributed engineering environment without the need for complex modelling and software integration infrastructure.

9-2.3 Innovative approach to the design and implementation of 3D modelling software

Typically, VPE software environments result from the integration of large software modules each providing the modelling functions, interfaces and data management functionality corresponding to one aspect of the model (e.g. 3D geometrical, model behaviour editing). This approach to modelling software design is partially the cause of the sequential approach to model implementation mentioned in chapter 3. In addition, such an approach results in the need for a complex integration infrastructure (e.g. software services, database and real time event management systems) which makes the deployment, maintenance and development of such software, resource and time consuming. The concept of component was used as a cornerstone in this research. The concept of PoCo modelling components that provide the basic infrastructure of the PoCo VPE, has resulted from merging the concept of “production system components” (e.g. machine physical modules, distributed control nodes) and “software components” typically used as architectural principle in software engineering. This has led to a unique approach to the design and implementation of so-called “modelling components”, which are highly structured (internal element structure), highly functional (encapsulate modelling, composition, interface functions) and highly portable software objects implemented exclusively on web compliant technology (namely VRML). PoCo-modelling components encapsulate both the data describing the different aspects of a machine component, and the functions required to view/run/edit complete machine virtual prototype.

![Diagram of PoCo Modelling Components](image.png)

Figure 9-2: An innovative, approach to the implementation of 3D modelling components, resulting from the investigations of various types of components.
This approach to the design and implementation to modelling software is unique since its aims at defining the VPE software architecture according to the real system architecture. The resulting modelling software consists of functions (i.e. model logic engine, user interfaces, model editing and configuration functions) which are distributed among highly functional, reusable, re-configurable, and most importantly portable software components. This approach contrasts with the typical approach to VPE design, which generally provides reusability of model parts and functions through the real time management of modelling data objects, implemented using different languages and software services. The PoCo approach to VPE design allows better modelling process management, which can benefit the modelling of both FMS and RMS manufacturing system types. In addition, the PoCo modelling object framework is “open”, so that the modelling objects functions and structure can be extended to suit the modelling of different systems.

9.2.4 An innovative approach to the choice of modelling component language implementation

It was decided at the beginning of this research, that PoCo modelling components would be implemented using exclusively the VRML language. This decision was made to i) preserve the portability of PoCo modelling component and therefore of both model and modelling functions, and to ii) evaluate to what extent the VRML programming capabilities (i.e. pre-defined VRML node functions and embedded JavaScript) could support the implementation of potentially complex functions whose purpose was beyond the domain of 3D modelling. The implementation of PoCo modelling components’ composition functions has shown that some functions could not be implemented using only VRML scripting capability, hence providing indications about the limitations of VRML in supporting certain types of functionalities.

The second point mentioned above ii) has lead to the investigation of the extent to which VRML could provide support for the object-oriented (OO) paradigm that provide guidelines and code management mechanisms to ensure software code reusability and manageability. This part of the research was significant and has extended the very short list of research projects that have focused on exploiting VRML as more than a simple 3D interchange format, or which have seek to provide guidelines and suggestions for future VRML development. The present research has highlighted the limitations of VRML in supporting true object oriented concepts (i.e. encapsulation, classes’ inheritance) and has provided alternatives (see Chapter 7) that could be used to overcome those limitations.
9-3 Future development for the present research

The current version of the PoCo VPE consists of a set of PoCo modelling elements providing the basic modelling and model usage functions and of a PoCo component and PoCo component-based model structure that define respectively the way modelling elements and modelling components are composed (integration and interaction) into respectively component and complete system models. The suggestions for the future developments of the PoCo modelling framework are:

- the improvement of existing PoCo modelling elements or the development of additional modelling elements and corresponding functions in order to extend the PoCo VPE modelling capabilities to new types of manufacturing systems (e.g. Numerical Control system, industrial robot control)
- to further development of the component and component-based model structure in order to improve the modelling component and component-based model reusability and re-configurability
- the development of the software interfaces that can help the user during the PoCo modelling elements, components and component-based system configuration phases.

9-3.1 Development of additional PoCo modelling elements

The set of PoCo modelling elements has initially been designed to provide the modelling functions required to implement 3D dynamic model of sequential logic driven MSs. This choice was made because the behaviour production lines such as conveying systems, assembly systems, which are characterised by a reconfigurable mechanical hardware is usually implemented using state-based sequential (PLC based most of the time) control logic. In addition, complex NC machining or robots operation can be modelled as a macro sequence of discrete states (e.g. ready, working) so that the modelling of complete production lines can still be achieved.
The PoCo modelling framework architectural characteristics allow additional elements to be integrated with existing elements (openness). This characteristic of the PoCo modelling framework has been exploited to implement modelling elements whose functions were specific to the requirements imposed by the particular type of logic (e.g. Krause test machine actuator\(^1\)). More intelligent logic elements such as part-counters have been implemented for the specific purpose of the ASDA case study. Other elements specific to the modelling of continuous machine control could be implemented, so that G code blocks for instance could be interpreted and translated into a set of cinematic parameters. By extending the modelling elements library, it would therefore be possible to extend i) the types of component that can be created and hence ii) the PoCo environment modelling capabilities to a variety of manufacturing systems. Future modelling elements implementation would focus in priority on typical manufacturing systems such as NC machines and industrial robots, which could be implemented as large, reusable and re-configurable PoCo modelling components. However, other elements could extend the way user interacts with the model. For instance, the implementation of augmented reality mechanisms would allow input from sensors placed on the real machine to be used to provide visual feedback for maintenance purposes (e.g. actuators vibration or torque analysis, tool weariness, measured actuator speed).

Additional elements could be implemented, whose functions are not directly related to the modelling of system behaviour or geometry. The possibility of implementing CLA elements (for CLAss elements) has been investigated. CLA elements could be used to support automatic model editing. For instance, a component modelling a clamping mechanism would be defined as belonging to the modelling component class *clamp*. The CLA element would hold a list of typical feature of this type of component e.g. “part in position” sensors, “clamping actuators state”. Another class of modelling component could be defined as *machining actuators*. The CLA elements would implement a knowledge base enabling elements to achieve automatic mapping of components’ state when two components belonging to two compatible classes are composed (e.g. automatic interlock definition between clamp components’ sensors and machining component actuator). From a user viewpoint, this would reduced the modelling task to simply drag and drop of a component belonging to the class “machining actuator” as a child (in the tree structure that describe the modelling component parent-child relationship) of another component belonging to the class “clamp”. This type of mechanism would raise the PoCo modelling objects from the status of

---

1 Please refer to chapter 8 for more details on the Krause Test Machine case study
software components, to the status of software agents, which encapsulate knowledge of other components' characteristics and knowledge of the system being model and its architecture.

The implementation of LP and INT\(^1\) PoCo modelling elements are the proof that more intelligent component can greatly simplify and automate the modelling process. However, the limitations of the VRML language and embedded Java Script for VRML have also shown that complex functions are more difficult to implement. At this stage, use of external programming language can be made. However, as shown by the analysis of previous research, this has a direct impact on model portability, which was defined as essential in this research. A careful assessment of the requirement in terms of model functionality should be made before trading model portability.

### 9.3.2 Enhancement of PoCo component structure

The PoCo component model (i.e. internal PoCo modelling component's structure) provides modelling openness and modelling reusability and manageability. The PoCo component model describes the way PoCo elements are integrated (i.e. elements' interfaces and event passing mechanisms between modelling elements). The decomposition of the overall model functionalities in modelling elements has resulted from the requirement to minimise the number of objects required to compose a model, which also minimises the amount of inter-object communication (of importance because of the difficulties to orchestrating events within class hierarchies using the VRML event model). In addition, this particular object decomposition results from the grouping of modelling functions that are more likely to be reused together. Breaking down the modelling functions into more specific elements could have increased the modelling flexibility but would have resulted in a more complicated and error-prone model composition process. On the other hand integrating all of the functionality into one type of element would have simplified inter-object communication and model composition, but at the expense of reduced modelling flexibility.

Object-oriented VRML has been implemented in order to decouple code structure from code functionality. However, the lack of support of the VRML language for object-oriented inheritance mechanisms makes the development of a clean modelling architecture difficult and imperfect. During the successive development phases of the PoCo VPE the fact that the PoCo component structure was partially coupled to the type of system being modelled became obvious. The link between components' structural and functional aspects implies that

---

\(^1\) Please refer to chapter 5 for a detail review of the modelling elements implemented so far, and their corresponding functions
the modification of component’s functions (e.g. inclusion of new modelling elements, modification of existing one), required a partial redesign of other component interfaces. Further development of the VRML based object-oriented mechanisms, and hence of the PoCo model architecture, could be conducted. The research conducted by Diehl [96]¹ has provided some interesting directions of research, which focus on the implementation of elements whose functions are dedicated exclusively to the management of communication between PoCo modelling elements. This would result in dissociating two type of elements, i) elements providing modelling functions, and ii) elements providing internal component management functions. The development of the second type of elements could ultimately lead to guidelines for future development of new types of VRML nodes, whose functions would be more oriented towards programming and modelling code management (as the VRML scene graph and grouping nodes are) rather than towards purely modelling functionality.

9-3.3 Development of additional software interfaces

The first phase of the design of the virtual prototyping environment was largely focused on the integration between manufacturing system engineering tools (e.g. PDE, 3D CAD modeller) and on the implementation of software modules to support the implementation of virtual machine prototypes. Conversely, the later phases of this research, characterised by the design and implementation of the final version of the PoCo VPE, have focused on the design and implementation of modelling elements, components and component-based model structure. The approach adopted for the design and implementation of PoCo VPE modelling objects allowed the configuration and composition of PoCo machine model via the editing of parameters in very simple VRML files. Little effort has been made to implement the software components providing graphical user interfaces since this was not essential to the PoCo VPE implementation and testing. Nevertheless, an example of a PoCo modelling component geometrical composition user interface has been implemented and presented in this research (see Chapter 8). However, in order to obtain a VPE that could be proposed as finished products to industrials partners, it would be necessary to wrap PoCo VPE functions into more elegant and usable interfaces. Note that this aspect of the future PoCo development is related to general software development and not to the PoCo VPE development.

¹ Please refer to chapter 6 for more details on Diehl’s research.
9-4 Suggestions for future direction of research and general conclusion

The previous paragraphs have focused on the future work that would potentially allow the PoCo VPE to be further improved, therefore providing a work plan for the author of this thesis. Comparatively, the following sub-sections are focused on the potential development that are out of the scope of this research, but which investigation could indirectly allow the present research to be improved.

9-4.1 On the use of virtual prototyping in the domain of manufacturing

The first suggestion concerns the way the activity of 3D virtual prototyping (VP) is perceived in the domain of manufacturing system (MS) design. The processes and tools associated with implementation and use of systems’ VPs, is conducted using existing methodologies and approaches to MS design. VPEs are therefore developed to complement existing design infrastructures rather than as a tools that can potentially enable new approaches to system design processes, design architectures and design organisation to be investigated. In some domains of manufacturing such as product design and development, the use of 3D modelling and VP technologies has revolutionised the way enterprises operate. Three dimensional modelling and simulation have not only changed the way product engineers work, but have also affected the way product engineers’ approach the design of new products [62]. In the same way, the potential of using 3D models in the manufacturing system design industry has been investigated in this research. The present research has highlighted two main aspects of MS design that could be significantly changed by adopting innovative VPE. Those two aspects are:

9-4.1.1 Focus on the type of manufacturing systems

The differences between Flexible Manufacturing Systems (FMS) and Reconfigurable Manufacturing Systems (RMS) design, and the means by which those two approaches provide flexibility of machine design, has been highlighted in this research1. RMS stands as a new machine paradigm whose potential can be fully exploited if supported with adapted design methodologies and software environments. Commercial VPE developing companies currently focus eighty percent of their effort in developing VPE modules whose functions and interfaces are adapted to well established FMS systems such as multi-axis machining centres, and industrial robots and coordinate measurement machines [8] [109]. From the author’s

1 Please refer to Chapters 2 and 3 for a review of FMS and RMS approaches to flexible machine design.
viewpoint, this results from the aim of VPE software developers to supply the demand of a relatively conservative manufacturing industry. However, the need for production flexibility and for more agile MSs has led machine builders to consider Reconfigurable Machine System (RMS) technologies more carefully. Statements from projects managers met during meeting at Ford have clearly shown that distributed control systems and modular machines are considered to be the future of automotive industry [9] [8]. For new technologies to be adopted, the consortium of manufacturing partners, including control system vendors and software developers (e.g. VPE developers) need to support such approaches. In particular, VPE developers should make use the high level of expertise they possess in both software development and machine design to take a leading position and to influence MS engineering industry to adopt what is referred to as the next generation of MSs.

The academic sector has been very active regarding the investigation, specification and implementation of new RMS machine technologies and of associated engineering software. However, the interest in using of 3D modelling technologies to implement VPE specifically adapted to the design and (re) configuration of RMS is relatively recent. Few research projects have paved the way towards a new approach to VPE for RMS design. However, manufacturing system 3D prototyping requires a high level of expertise and resources in the domain of manufacturing system design, 3D modelling, and software engineering. This often led to unbalanced academic projects, placing a strong emphasis on one of those aspects related to the development of VPE. The present research strongly suggests an innovative approach to 3D computer modelling of complex systems. The adoption of a component-based approach to 3D prototypes implementation and the use of emergent web-based modelling technologies propose a racially new way of approaching the development of VPE.

9-4.1.2 Focus on the type of organisation and on standards technologies

The concept of Virtual Enterprises and Virtual Organisations (VE and VO) has been reviewed in this research as emergent types of organisation resulting from the need of industrial partners to focus on core competencies. This approach to industrial partnership results in distributed engineering resources, separated processes and heterogeneous IT/IS infrastructure, which render the collaboration between partners difficult. An important aspect of virtual prototypes (VP) is the potential of 3D models to provide a very effective communication basis in a distributed engineering environment. Once more, from the author’s perspective, there is a crucial lack of consideration and support from commercial VPE developers for the potential

1 The concept of VE and its implications on manufacturing system design has been reviewed in Chapter 2.
of 3D virtual prototypes in supporting distributed engineering collaboration. It has been highlighted (cf. chapter 3) that VPs implemented using commercial VPE application offer no portability at all since they are implemented upon highly proprietary formats and exclusively rely on the editing software environment to provide model dynamic display and behaviour modelling capabilities.

From the development perspective adopted in this research, VPEs should be developed in order to offer highly effective support in a VE context. Because of the lack of model portability provided by currently available commercial VPE applications, the only option left to VE partners in order to benefit from the use of virtual machine prototyping, is to adopt a common VPE solution. Considering the costs, deployment efforts and maintenance issues associated with commercial VPE such as Delmia or Tecnomatix, such an option is often not achieved. Ad hoc communication and collaboration between partners that impairs the partnership efficiency and agility is often the result.

Throughout the development of this research project, it was clearly noted that there is a lack of support from modelling software development industry for emerging Web-based 3D modelling standards formats (e.g. VRML). Commercial VPE developers make use of proprietary formats to prevent integration of third party software. The investigations of CAD to VRML export capabilities\(^1\) has highlighted that the same approach is usually adopted in the CAD industry. This practically implies that modelling environments most adapted to the modelling of mechanical systems cannot effectively be exploited and integrated with VPEs. There is a critical need for the CAD and commercial VPE industry to provide more support for standard 3D modelling formats such as VRML and the future X3D format. Web based technologies such as Java, client-server architectures, HTML and XML, have shown the vast potential and advantages of using such technologies in implementing strong yet flexible communication, collaboration and software integration infrastructures. From the author’s perspective, it is crucial that the manufacturing industry realise the potential of emergent modelling standards in supporting the development of advanced engineering tools. The PoCo modelling environment developed in this research clearly shows that VPE software, which provides portability; usability and openness can be implemented with a minimum of resources.

\(^1\) Please refer to chapter 6 for additional details on the CAD to VRML formats translation issues.
9-4.2 General conclusion

The research presented in this thesis has proposed an alternative approach to the specification and implementation of an innovative Virtual Prototyping Environment intended to support the design of manufacturing systems. The portable component-based Virtual Prototyping Environment (PoCo VPE) has resulted from the review of general manufacturing concepts and from the analysis of current VPEs. As a result, the PoCo requirement specifications and implementation have focused on providing a new type of VPE whose functions and overall design approach are adapted to the design of Re-configurable Manufacturing Systems (RMS). In addition, the PoCo VPE has been specified and implemented to provide maximal support for communication and collaboration between distributed engineering resource and infrastructure, which is characteristics of today’s approach to manufacturing organisations.

Several development phases have lead to a progressive refinement of the PoCo VPE conceptual design and practical realisation. The final version consists in a VPE, whose functions and design are highly adapted to the prototyping of RMS system and provide a high level of model reusability and re-configurability. In addition, a very high level of portability has been achieved for both the virtual prototype models generated using the PoCo VPE and the modelling functions. This allows industrial partners who are distributed and who have different level of IT expertise and infrastructure to gain benefits from the virtual prototyping activity. Finally, future developments for the present research have been proposed. From the experience gained during the development of this research, relevant suggestions for future research in the domain of virtual prototyping of manufacturing systems and regarding the development of VPE technologies have been made.
References:


[67] J. L. Zhao. Intelligent agents for flexible workflow systems. Department of information and systems management, Hong Kong University of Science and Technologies


Manufacturing Systems Integration Research Institute, Loughborough University


[134] ISO 10303, Standard for The Exchange of Product Model Data, STEP.


Additional References:


