Springer Series in Advanced Manufacturing

Wasim Ahmed Khan · Abdul Raouf
Kai Cheng

Virtual Manufacturing

Springer

Springer Series in Advanced Manufacturing

For further volumes:
http://www.springer.com/series/7113

Wasim Ahmed Khan · Abdul Raouf ·
Kai Cheng

Virtual Manufacturing

Springer

Prof. Wasim Ahmed Khan
Faculty of Computer Science
Institute of Business Administration
City Campus, Garden Road
74400 Karachi
Pakistan
e-mail: Wasim_khans@hotmail.com

Prof. Kai Cheng
School of Engineering and Design
Brunel University
UB8 3PH Uxbridge, Middlesex
UK
e-mail: kai.cheng@brunel.ac.uk

Prof. Dr. Abdul Raouf
University of Management and Technology
Johar Town C-2
54600 Lahore
Pakistan
e-mail: abdulraouf@umt.edu.pk

Additional material to this book can be downloaded from http://extra.springer.com

ISSN 1860-5168

ISBN 978-0-85729-185-1 e-ISBN 978-0-85729-186-8

DOI 10.1007/978-0-85729-186-8

Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Cover design: eStudio Calamar, Berlin/Figueres

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Wasim A. Khan

Sadaf, Arqam, Sarah and Muhammad

Abdul Raouf

Dr. Razia Raouf

Kai Cheng

Lucy Lu

Preface

According to MSN Encarta the term 'virtual reality' is commonly used to express Simulated Reality, Computer Simulation, Simulation, Cyberspace, Computer Modeling or Computer Graphics. In today's scientific scenario, virtual reality is classified on a continuum from Real environment to its variations to virtual environment. These variations of virtual reality are from real environment to augmented reality to augmented virtuality to the virtual environment. All the intermediate representations are known as mixed reality.

Azuma et al. describes the Augmented Reality (AR) in their survey paper published in IEEE Computer Graphics and Applications [November/December 2001] as having the following properties:

1 AR combines real and virtual objects in a real environment;
2 AR runs interactively and in real time, and;
3 AR registers (aligns) real and virtual objects with each other.

The world of virtual reality still requires a specific definition of virtual reality considering the domains it is addressing. In such cases the particular research group provides a relevant definition. In this book, the augmented reality as defined by Azuma et al. is considered applicable.

A discrete manufacturing operation involves tangible activities such as machinery and its operation, use of tools and measurement gadgets, use of pick and place technology and use of storage and transportation equipment etc. On the other hand, the intangible part includes services such as process planning, scheduling, inventory, management information system and business accounting etc.

Establishment of discrete manufacturing facility for specified range of discrete products includes the factory and offices layout, machinery layout, operation of design office, operation of new and old machinery, production planning and control, scheduling, assembly, quality assurance, inventory, transportation, budgeting and accounting and financial activities. Monitoring of all these and other functions is required once the facility has been setup and is functional.

Modern virtual reality techniques using programming languages such as VRML (Virtual Reality Modeling Language), Open GL and object oriented tools C#, Java and Small Talk have allowed extension of concepts of real time simulation to real time simulation with user control in feed back environment. The simulation can be implemented to the extent that from an elaborate statement of corporate strategy to the smallest movement of part of the machine can be modeled and controlled. The supporting database allows maintenance of properties of metal in interaction with a moving tool, storage of different type of simulated machinery and other models and parameters. These models are based on mathematical or procedural methods facilitating functional characteristics of processes such as scheduling and process planning respectively.

The scope of this text is to describe development of virtual factory simulation software for discrete manufacturing based on Object Oriented Design (OOD) using Unified Modeling Language (UML). This book builds up from description of a micro level virtual reality construction of machine component to the virtual reality construction of discrete manufacturing organization. The executable version of virtual factory software for discrete manufacturing is available at the publisher's website (www.springer.com/). There is a scarcity in the market for a title, which has been written to introduce the students and professionals with virtual reality for Discrete Manufacturing as that of subject that can be practiced best through the study of relevant subject areas and that also addresses the relevant components of the technology. This book describes the concepts and technology associated with manufacturing equipment and their control at process and system level for product realization in modular form. It uses examples elaborating procedure to virtually describe processes and systems used in discrete and continuous manufacturing while experiencing flow of material, flow of information and flow of energy. The major emphasis is given to develop Augmented Reality (AR) for the following control gadgetry:

1. CNC based processes,
2. PLC based processes,
3. Industrial Manipulators,
4. Embedded systems based processes,
5. Mechatronics based processes and
6. SCADA based processes.

These micro level virtual realities are later amalgamated into virtual discrete manufacturing systems composed of procedural and mathematical models for intangible production functions.

The book has been divided into twelve chapters. The book can be consulted on the basis of individual chapters depending on the level of the reader. Chapter 1 sets the theme for the establishment of Augmented Reality based various levels of human computer interactions as the necessary requirement of the factory of the future. Chapter 2 explores the discrete and continuous manufacturing processes and examines the current technological trends. Chapter 3 surveys the current use of automation and control in manufacturing and comments on future directions it

may take. Chapter 4 examines the possibility of using sensors, transducers and actuators in a feed back virtual environment. Chapter 5 provides methodology for converting EIA 274 D based Computer Numerical Controlled (CNC) machinery into corresponding AR based machinery. Chapter 6 provides methodology for converting JIS SLIM (Standard Language for Industrial Manipulator) based manipulators into AR based robot controller. AR for gantry and conveyor is also discussed in this chapter. Chapter 7 details AR based process control using IEC 61131-3 based PLC programming languages. Chapter 8 examines conversion of embedded system based control to AR based processes. Chapter 9 details AR for SCADA based processes. Virtual reality for Mechatronics based applications are explained in Chap. 10. Methodology to simulate the intangible production functions is described in Chap. 11. Step by step construction of AR based discrete manufacturing facility based on either single or multiple CNC based processes, PLC based processes, embedded system based processed, SCADA based processes and/or Mechatronics based processes is described in Chap. 11. Chapter 12 provides the rationale for adopting AR strategy. The description of virtual discrete manufacturing organization uses both UML diagrams and software listing in part. Appendices at the end of the book provide basic information regarding software development process, Comprehensive bibliography is also provided at the end of each chapter to guide reader to the wealth of information available on the subject. This book is intended for manufacturing professionals with a background in mechanical engineering, industrial engineering, computer engineering and computer Science.

This work requires support from its user in order to improve the further editions. The authors welcome comments and suggestions. Authors may be contacted through authors20@yahoo.com

Contents

About the Authors

Professor Dr. Wasim Ahmed Khan

Professor Dr. Wasim A. Khan holds the first degree in Mechanical Engineering from NED University of Engineering and Technology, Karachi, Pakistan. He later obtained a PhD degree in the area of Operations Research from the Department of Mechanical Engineering, University of Sheffield, UK. He is a chartered engineer (CEng) of the Engineering Council, UK; and a Fellow of the Institution of Mechanical Engineers (FIMechE), UK. He has diverse work experience including working with manufacturing industry, software development for local and overseas clients and teaching production engineering and computer science students. Currently, Dr. Wasim holds a position of Professor at the Faculty of Computer Science, Institute of Business Administration, Karachi, Pakistan.

Professor Dr. Abdul Raouf

Professor Dr. Abdul Raouf is a distinguished scholar of international ranking, having a doctoral degree in Industrial Engineering and over 57 years experience in teaching, research and industry. He was at the University of Windsor, Ontario, Canada for more than two decades as Head of Industrial Engineering. He served King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia as Professor in Systems Engineering Department for ten years. He was Rector Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan for six years. He has been University Professor and Advisor of University of Management and Technology, Lahore since 2005. Dr. Raouf has been appointed Patron and Professor, Institute of Quality and Technology Management, University of Punjab.

Dr. Raouf has published extensively in the areas of Performance Evaluation which include Modeling and Optimization of Tasks involving Information Conservation, Information Reduction, Information Generation and Production System Optimization in the areas of Quality, Safety and Maintenance of Production

Systems. He has authored/co-authored ten books and contributed more than 150 research papers in refereed Journals and refereed conference proceedings.

Recognizing his scholarly pursuits, Dr. Abdul Raouf was bestowed upon the coveted title of 'Sitara-e-Imtiaz' by the Government of Pakistan. He was declared "Best Scientist" in the field of engineering by the Pakistan Academy of Sciences. The Higher Education Commission of Pakistan conferred upon him the title of Distinguished National Professor.

Besides being the Editor-in-Chief of International Journal of Quality in Maintenance Engineering Dr. Abdul Raouf is on editorial advisory boards of nine international research Journals. He is Chairman, National Quality Assurance Committee constituted by HEC. He has been appointed as a member of the Accreditation Committee, Education Department, and Government of Punjab. He is member of Governing bodies of number of public and private Universities.

Professor Dr. Kai Cheng

Professor Dr. Kai Cheng holds the chair professorship in Manufacturing Systems at Brunel University. His current research interests focus on micro manufacturing, design of precision machine tools, and global/sustainable manufacturing and systems. Professor Cheng has published over 160 papers in learned international journals and referred conferences, authored/edited five books and contributed six book chapters.

Professor Cheng is a fellow of the IET and IMechE. He is the head of the Advanced Manufacturing and Enterprise Engineering (AMEE) Department at Brunel University, which consists of ten academics and over 40 research assistants/fellows and PhD students. The department is currently working on a number of research projects funded by the EPSRC, EU 7th Framework Programs, Technology Strategy Board (TSB) high value manufacturing program, KTP Programs and the industry. The department is involved in 2008 RAE (research assessment exercise) Generation Engineering unit submission and ranked the 5th in the country. Professor Cheng is the European editor of the International Journal of Advanced manufacturing Technology and a member of the editorial board of International Journal of Machine Tools and Manufacture.

Amir Hussain

Software Engineer

Amir Hussain is a senior software engineer at the Faculty of Computer Science, Institute of Business Administration, Karachi, Pakistan. He holds a B.S degree in Computer Science and more than 10 years experience of software development.

Chapter 1
Augmented Reality for Manufacturing

1.1 Virtual Reality

The origin of the term 'Virtual Reality' can be traced back to the work of French playwright, actor and director Antonin Artaud [1938]. Artaud's drama 'Theatre and its double' was first of the sequel to be followed by several fiction and nonfiction novels, dramas, and movies generating enthusiasm in this novel area of work.

In current scenario, MSN Encarta defines the term virtual reality as commonly used to express simulated reality, computer simulation, simulation, cyberspace, computer modeling or computer graphics.

Michael Heim identifies seven different concepts of virtual reality as simulation, interaction, artificiality, immersion, telepresence, full-body immersion, and network communication.

The term virtual reality is also defined as a technology that allows a user to interact with a computer simulated environment whether that environment is a simulation of the real world or an imaginary world. The forms of virtual reality currently being practiced are visual experiences displayed on computer screen or through stereoscopic displays. Some simulations provide simple sensory information, while others make use of haptic sensors in a close loop feedback system. Some of the advance applications of the virtual reality utilize multimodal devices such as wired glove, the Polhemus boom arm, and omnidirectional treadmill.

In this book, it is proposed that the virtual manufacturing organization is an application of the augmented reality that apportions planning for a nonexistent manufacturing system, operation monitoring of the existent system, maintenance planning, fault diagnostic, rapid maintenance, practicing quality assurance, optimizing the local and global human–computer interfaces, optimizing the flow of information, acquiring rapid response from the system, training at the manufacturing systems, distance learning, practicing e-commerce and possibility of implementing different production philosophies.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_1, © Springer-Verlag London Limited 2011

1.2 Reality Virtuality Continuum

The reality virtuality continuum is a terminology used to describe a concept that there is continuous scale ranging between the completely virtual (virtual reality) and the completely real (reality). The reality virtuality continuum encompasses all possible variations and compositions of real and virtual objects. The reality virtuality continuum is depicted in Fig. 1.1.

This book relies on Azuma's description of augmented reality (AR). According to Azuma, AR is a state on reality virtuality continuum that exhibits following properties:

- AR combines real and augmented objects
- AR runs interactively and in real times, and
- AR registers (aligns) real and augmented objects with each other.

1.3 Augmented Reality: An Alternate Human–Computer Interaction

With enormous use of computers in almost all the walks of life, the user interface commonly known as human–computer interaction (HCI) has became the primary focus for research. The centuries-old lessons learned from the famous man–machine interface have been enhanced to meet the challenges of human–computer interaction. Over the years, the human–computer interaction itself has evolved to an extent whereby the user interfaces of 1960s have matured, and criticism on such human–computer interaction tool is commonly available in the contemporary literature. The design parameters for human–computer interaction include both hardware- and software-related parameters. On the other hand, the design theory for the human–computer interaction has now leaped from user-centered design to user involvement in the design. These components of HCI are utilized in designing an augmented reality for the discrete product.

These human–computer interaction parameters have progressed with the merits and demerits of batch processing with nonexistent user, command mode interaction using MS DOS or UNIX operating systems and graphics user interface (GUI) with MS WINDOWS. The current-generation human–computer interfaces are now addressing virtual environment using factors such as user's movement and voice.

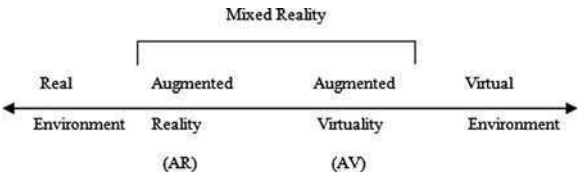


Fig. 1.1 Reality Virtuality Continuum

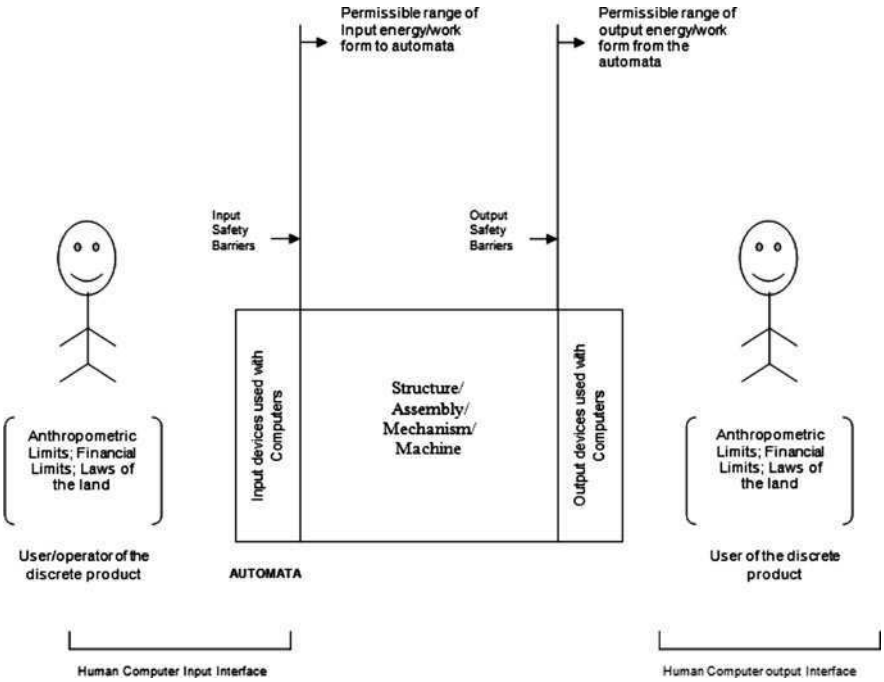


Fig. 1.2 Human–computer interface for discrete products. Extracted with permission from 'Standards for Engineering Design and Manufacturing' [28]

The use of AR to enhance human–computer interaction is demonstrated throughout this book.

The modern discrete products extensively utilize microprocessor-based control of the features. These characteristics require the design of a human–computer interface, allowing more automation in the operation of discrete products. A description of the scope of HCI in discrete products is presented in Fig. 1.2.

1.4 Virtual Manufacturing

In this book, virtual manufacturing (VM) is considered to be an augmented reality (AR) environment exercised to enhance all levels of control in a discrete or continuous manufacturing system. Virtual manufacturing objects include both tangible and intangible functions of production, including manufacturing, costing, process planning, scheduling, quality control, and management information system.

With enormous advancement in computer software and hardware design and development, augmented reality (AR) has taken a frontal role in the description, planning, and real-time operation monitoring, fault diagnostic and recovery, and training related to industrial products and processes. Ease in production planning,

process monitoring, fault diagnostics and recovery through fast response maintenance are identified as the major benefits of the use of augmented reality in state-of-the-art production methods.

The basic configuration of discrete and continuous manufacturing systems by Chryssolouris is considered. This comprises the following:

- Job shop
- Project shop
- Cellular system
- Flow line
- Flexible manufacturing system, and
- Continuous manufacturing.

These ideal layouts for discrete manufacturing system along with assembly methods constitute the core of the augmented reality template required for defining any virtual reality application in manufacturing. The processes required to build a manufacturing system are mainly adopted from Kalpakjian's description of current manufacturing processes comprising all or some of the following general areas:

- Metal forming processes
- Bulk deformation processes
- Sheet metal working processes
- Metal cutting processes
- Metal joining processes
- Thermal properties modification processes
- Surface properties modification processes
- Fabrication of micromechanical and microelectronics devices
- Nonconventional processes
- Continuous manufacturing processes.

The discrete manufacturing systems are composed of physical structure for the processes and the system, control characteristics of the processes and the system, and dynamics involved in the operation of the processes and the system. Augmented reality for discrete manufacturing systems involves defining steps required to build an augmented reality (AR) for a manufacturing process such as metal cutting process from component to mechanism to machine level and subsequently integrating the augmented reality (AR) of the process into the augmented reality (AR) of the manufacturing system such as job shop, project shop, cellular system, flow line, and continuous manufacturing system along with the dynamics of energy, material and information flow of a particular manufacturing system.

In a true augmented reality of discrete or continuous manufacturing system, the manufacturing processes should be defined with high level of virtuality encompassing all their functionality and should be capable of being configured on a chosen manufacturing system layout (e.g. on one of Chryssolouris' models of manufacturing system) along with comprehensive energy, information, and material flow capability.

The simulated processes have the capability to mimic real manufacturing processes and are configured on a simulated factory layout. It is possible to configure each of the tangible operations in variable quantity and size depicting a real factory. It is also possible to integrate tangible and intangible production functions in a virtual manufacturing system.

The manufacturing system and the manufacturing processes chosen in the virtual domain are reconfigurable, thus allowing infinite possibilities for the development of augmented manufacturing organization depending on the processing capability of the host computer. The processes are operated using variants of EIA 274D [8–10] and JIS SLIM (Standard Language for Industrial Manipulator), techniques used in programmable logic controllers (PLC) such as defined by IEC 61131-3, embedded system and Supervisory Control and Data Acquisition (SCADA) and other pertinent standards. The simulation interactively use inter departmental transportation methods using gantries and conveyors.

To provide an entirety to virtual discrete manufacturing system, the intangible functions of production are modeled mathematically. These models refer to the following activities:

- Costing
- Accounting and finance
- Sales
- Inventory management
- Procurement
- Process planning
- Quality assurance
- Scheduling
- Management information system.

Intangible functions are incorporated using mathematical models interfaced with factory simulation to maintain the record of presence of various physical objects and decision parameters, e.g. machine selection, machining parameters, and tools/fixtures are defined by the process-planning competency. All the simulations and mathematical models, virtual product in Direct X format, and other parameters are stored in databases.

Layout, machines, product, and their types are all parameters that have a presence. The augmented system detailed here is capable of modeling a discrete manufacturing facility and providing an insight into a nonexistent system. Alternatives may be generated by changing the layout of the manufacturing system and by increasing or decreasing the number and types of machines. The diversity of the product and its quantity can be changed to provide the feasibility of manufacturing system during the proposed period so that the system achieves the break even and profit margins may be implemented in a competitive e-market. For an existent system, the augmented factory contributes in operational planning and maintenance planning. The augmented manufacturing system may also be used for advance training and research in the operation of competencies, production philosophies, and the quality assurance procedures. The system shall prove an excellent training

resource and may also be used in distance learning. One novel software application of the virtual manufacturing organization is its use of e-business through semantic web employing web-enabled agents. The human–computer interface (HCI) at various levels of the organization can also be explored. The augmented discrete manufacturing facility is applicable to all type of discrete manufacturing facilities. This scheme models product, processes, and system in real time.

Communication in virtual manufacturing system is carried out through Internet, extranet and intranet. Due to the inherent flexibility of the augmented organization, the flow of information in an augmented organization can be reorganized conveniently. The development of software for augmented reality and procedural/mathematical models employ objected-oriented technique. Unified modeling language (UML) is used to model the software. Object-oriented high-level language C# is used to develop simulation and models with the support of dot Net framework. Graphic Modeler 3D Studio Max and rendering engine True Vision are used for graphic support. Other software tools that may be used to develop such AR system are listed in "Appendix 1".

1.4.1 Virtual Manufacturing Systems

Virtual manufacturing systems are synthetic environment designed to exhibit manufacturing systems operation on a reality virtuality continuum. The reality–virtuality state exhibited by the manufacturing system falls into the following categories.

- *Reality*: Real manufacturing operation
- *Augmented reality*: Manufacturing system control is augmented by the use of electronic hardware and computer software in order to facilitate managers with more micro- and macro-level parameters for accurate decision making leading to higher profitability.
- *Augmented virtuality*: Consist of higher level of virtuality than augmented reality. In augmented virtuality, a higher proportion of elements are synthetic in nature.
- *Virtuality*: Encompasses immersion in a completely synthetic environment.

1.4.2 Organization of Virtual Manufacturing Systems

Virtual manufacturing system is organized from component of machines to collection of machines in a cell to multiple cells on a shop floor and extending to all other processes on a factory floor.

In a real manufacturing system organization, the machines or processes are rarely moved from one place to another. On the other hand, the virtual

manufacturing system has the inherited characteristics of relocation during planning in order to explore optimal output under long-term schedule. For short-term schedules, while the machine remains at the designated position, the optimal part routing can be explored. This part routing maybe within factory premises or beyond, refer to Sects. 1.10 and 12.1.

The flow of energy in a manufacturing system is carried through electric cables, gas pipelines, and fossil fuel on conveyors. The raw material in a manufacturing system is carried through automatic feeding mechanisms, fork lift, gantry, and manually. On the other hand, the flow of information can be routed to any node supervised by a manager as per the administration access rights available to the manager.

1.4.3 Components of Virtual Manufacturing Systems

The virtual manufacturing systems components comprise both real and virtual objects. These objects have the property of aligning themselves in real time.

Discrete[1] manufacturing processes are developed as mechanical artifacts to perform production. These machines produce a variety of components, structures, assemblies, mechanisms, and machines. Each manufacturing process implementation into mechanical artifacts has three distinct features:

I. The input to the equipment, e.g. material type, form and feeding mechanism, final dimension of the product, energy source and other auxiliaries.
II. The process implementation allowing transformation of raw material into required size, shape, and surface finish using tools (e.g. tools as used in metal cutting, high-energy beams or various types of jets), measuring devices and manufacturing instructions. In process, supplies such as lubricating oil and coolants may also be used.
III. The output from the equipment comprising a component (the building block of structure, mechanisms or machines) and scrap. There may be several features present at the production machinery to make the task of manufacturing simpler, easy to control, and resulting in high production rate. The design of manufacturing equipment relies on system approach to design and considers few or all of the following special design requirements:

- Structural consideration for machine frame and assemblies
- Thermal effects on the equipment
- Noise emission from the equipment
- Vibration in the machinery
- Environmental effects on the equipment

[1] Extracted with permission from 'Standards for Engineering Design and Manufacturing' [27].

- Geometric and kinematics behavior of the manufacturing equipment
- Static and dynamic behavior of the equipment
- Availability of computer numerical control (CNC) or process control using programmable logic controllers (PLC)
- Electronic circuitry for implementing control features
- Foundation and installation requirement.

Like any other discrete product, the manufacturing equipment commonly utilizes standard mechanical components, machines elements, control elements, electrical and electronic components, and software components.

Special assemblies and other accessories utilized at the construction of manufacturing equipment may also include the following:

- Tool (referring to mechanical tool, light beams, water jets, etc.)
- Tool holding devices
- Work holding devices
- Lubricating oil pump assembly
- Coolant circulation pump assembly
- Material handling equipment, and
- Scrap handling equipment.

1.4.4 Control of Virtual Manufacturing Systems

A control system at the production equipment may be defined as one or more interconnected devices that work together to automatically maintain or alter the condition of machine tool, robot, or other manufacturing processes in a prescribed manner. Such system may be mechanical, electrical, electronic, hydraulic, or pneumatic. In practice, many control systems are a combination of these and termed hybrid system. If the output quantity has no effect on input quantity, the system is called an open loop system. In real life, control systems sensors and transducers provide the feedback that is used to adjust the input depending on the current state of the output.

Technologies used in discrete and continuous manufacturing for control and automation are:

- Programmable logic controller (PLC)
- Computer numerical control (CNC)
- Industrial manipulator
- Supervisory Control & Data Acquisition (SCADA)
- Embedded systems
- Mechatronics-based system

A programmable logic controller is an electronic apparatus that uses a programmable memory for the internal storage of instructions for implementing specific function such as logic, sequencing, timing, counting, and arithmetic to

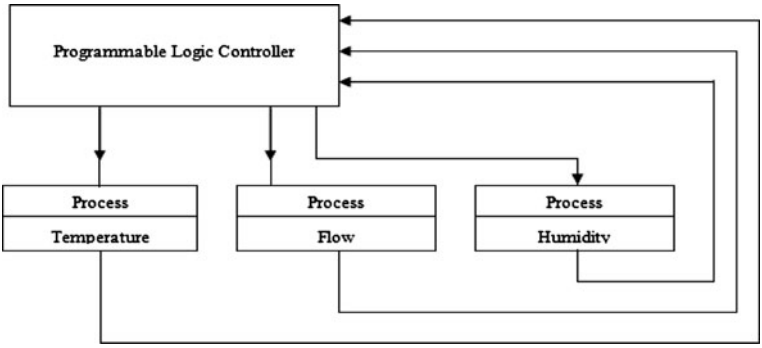


Fig. 1.3 Process parameter control using programmable logic controller (PLC). Extracted with permission from 'Standards for Engineering Design and Manufacturing' [27]

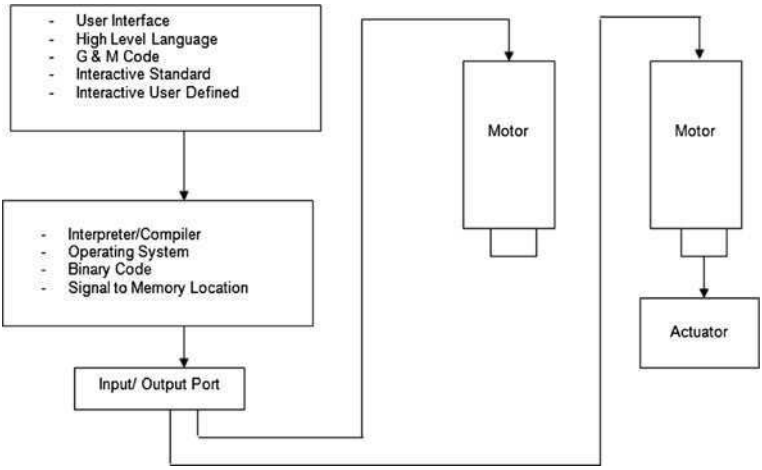


Fig. 1.4 Open loop control of CNC machines. Extracted with permission from 'Standards for Engineering Design and Manufacturing' [28]

control, through digital or analog input/output modules, various types of machines or processes. Programmable logic controller is widely used to automate discrete or continuous manufacturing processes as shown in Fig. 1.3.

Computer numerical control is the technique of giving instructions to a machine tool in the form of code (known as part program) according to a standard such as EIA 274 D that consists of numbers, letters of the alphabet, punctuation marks, and certain other symbols. The machine responds to this coded information in a precise and ordered manner to carry out various machining functions. Computer numerical control is widely implemented at the metal cutting processes. Block diagrams of open and close loop models of CNC controller are exhibited in Figs. 1.4 and 1.5.

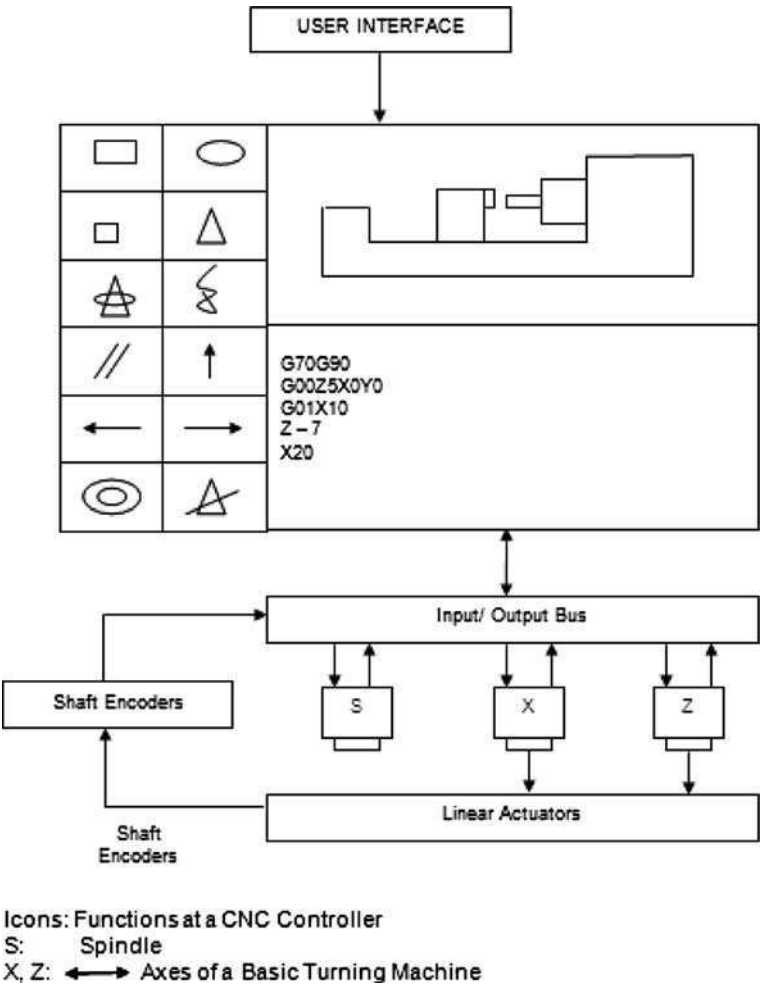


Fig. 1.5 Close loop control of CNC turning machines. Extracted with permission from 'Standards for Engineering Design and Manufacturing' [28]

Industrial manipulators are used to automate a wide range of equipment such as domestic gadgets to pick and place technology. The level of automation offered varies significantly at a manipulator used to automate a product to that in a robot used to help manufacture the product. Japanese Industrial Standard SLIM (Standard Language for Industrial Manipulator) is one of the leading standards used to control the robots. Figures 1.6 and 1.7 provide a schematic of SCADA and embedded systems, respectively. SCADA is normally used in continuous manufacturing where distances between input devices, output devices, and feedback gadgets are larger. Embedded systems are used at equipment requiring high speed of execution. Number of defined control methodologies also fall in the category of

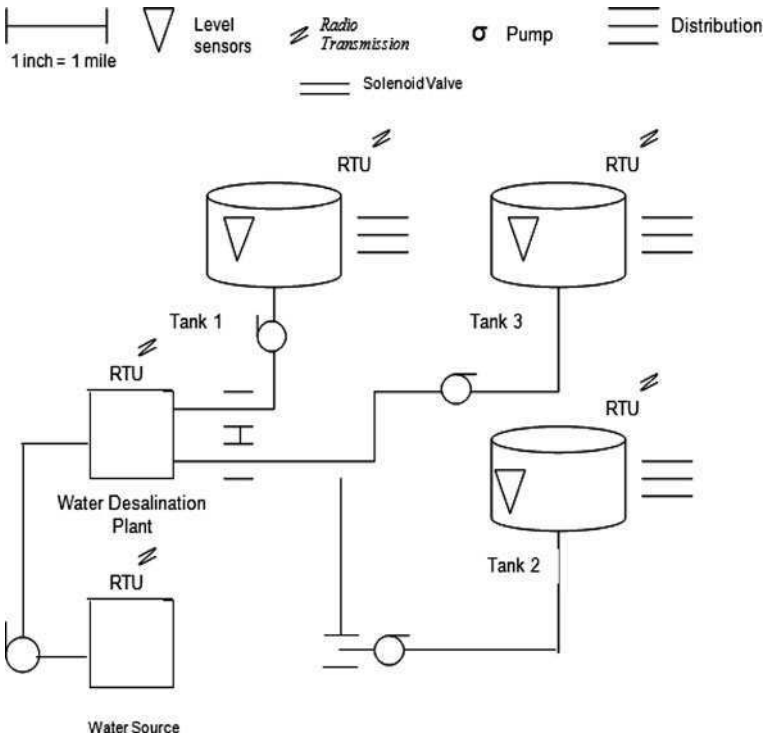


Fig. 1.6 Large-capacity water storage tanks controlled using SCADA

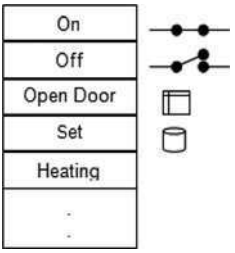


Fig. 1.7 A schematic of an embedded system

mechatronics-based system. These control systems are used for well-defined areas of products, while the mechatronics has a much wider scope with one exception that it must include control of mechanical parts.

An integrated manufacturing system is constituted when there are defined channels available for flow of information, flow of material, and flow of energy to operate the manufacturing system in an integrated manner. These channels may be based on established standard or may rely on company practices for the production of specified objects. There is a formation of levels of control in such systems that

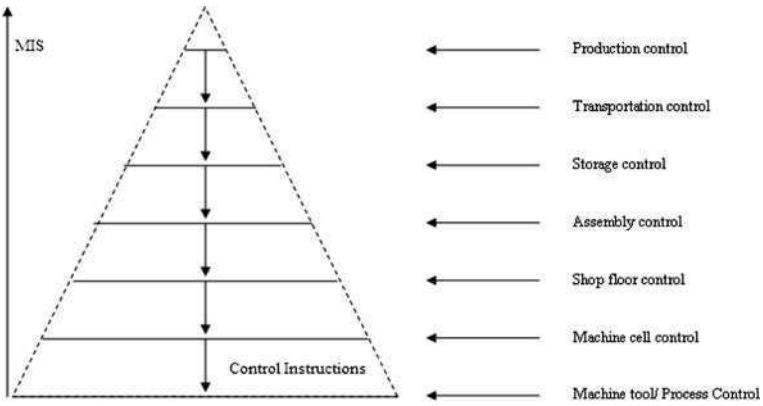


Fig. 1.8 Levels of controls in a single domain of integrated manufacturing system

are operated using a man–machine interface in the case of conventional machinery-based system or human–computer interaction in the case of automated machinery. These levels of control amalgamate in upward direction to provide control of manufacturing system, refer to Fig. 1.8

1.5 Development of Virtual Manufacturing System Using Augmented Reality

The development of virtual manufacturing system (VMS) using augmented reality is an exercise in object-oriented analysis and design from computer science perspective. The VMS requires definition of databases related to all the entities encapsulated in discrete manufacturing systems.

These entities, either at the factory floor or in the store, are an inventory item in real form as well as a graphical model or a data item in the virtual form in the database. Following database containing elements of discrete manufacturing is of importance for virtual manufacturing.

1.5.1 Machine Tool Database

The machine tool database comprises specification of all type of machine tools present in the real manufacturing system. The structure for this database considering both real and virtual objects is provided in Figs. 1.9 and 1.10. Some of the parameters for such a database may be center-to-center distance for turning operation or spindle speed ranges at a milling machine.

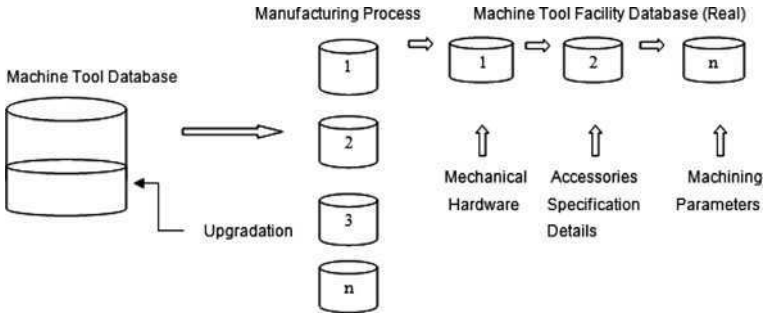


Fig. 1.9 Structure of real objects Database

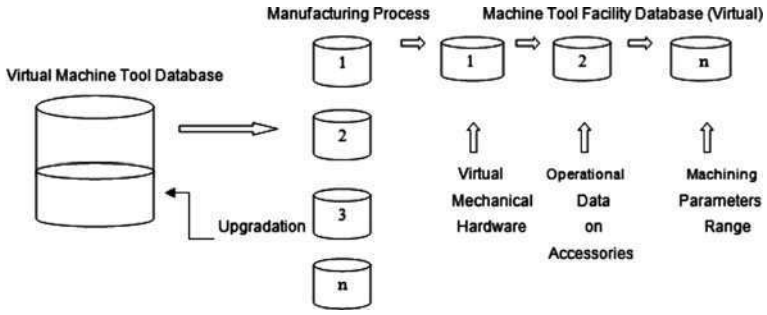
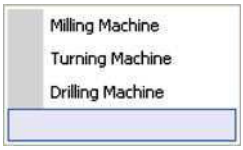


Fig. 1.10 Structure of virtual objects Database

Machines Database menu item has three subitems as shown in the following picture.



- Milling machines
  From Milling Machines Database Form, user can add new milling machine and view existing machines parameters. Refer to the form given as Fig. 1.11.
- Turning machines
  From Turning Machine Database Form, user can add new turning machine and view existing machines parameters. Refer to the form given as Fig. 1.12.
- Drilling machines
  From Drilling Machine Database Form, user can add new drilling machine and view existing machines parameters. Refer to the form given as Fig. 1.13.

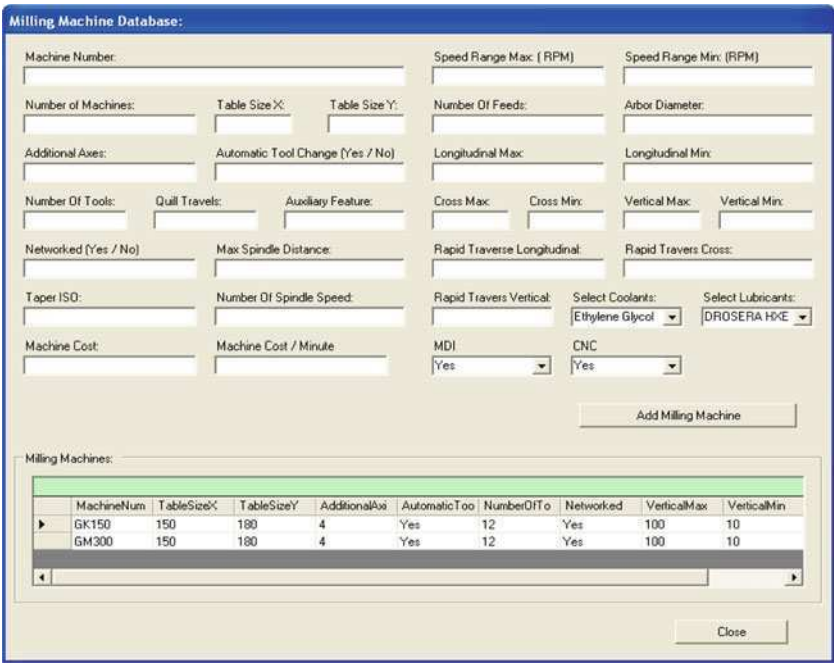


Fig. 1.11 Milling Database

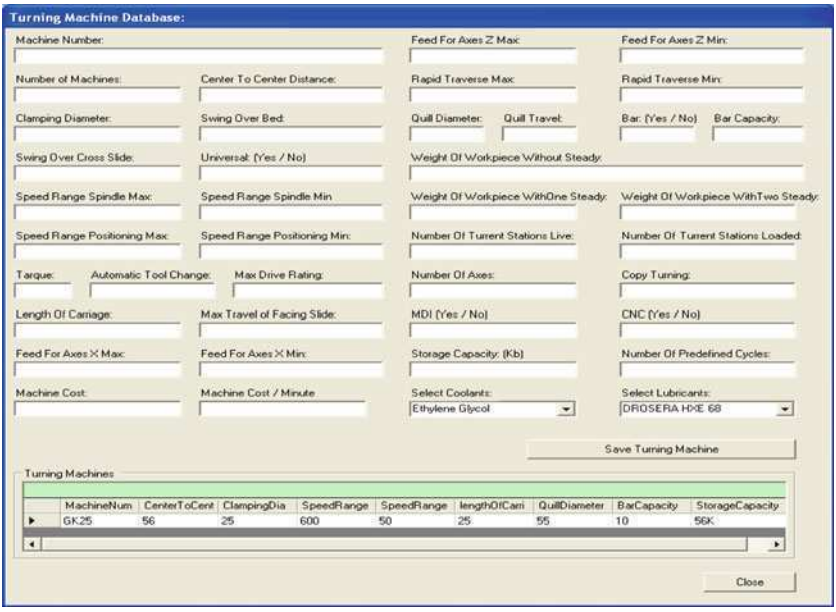


Fig. 1.12 Turning Database

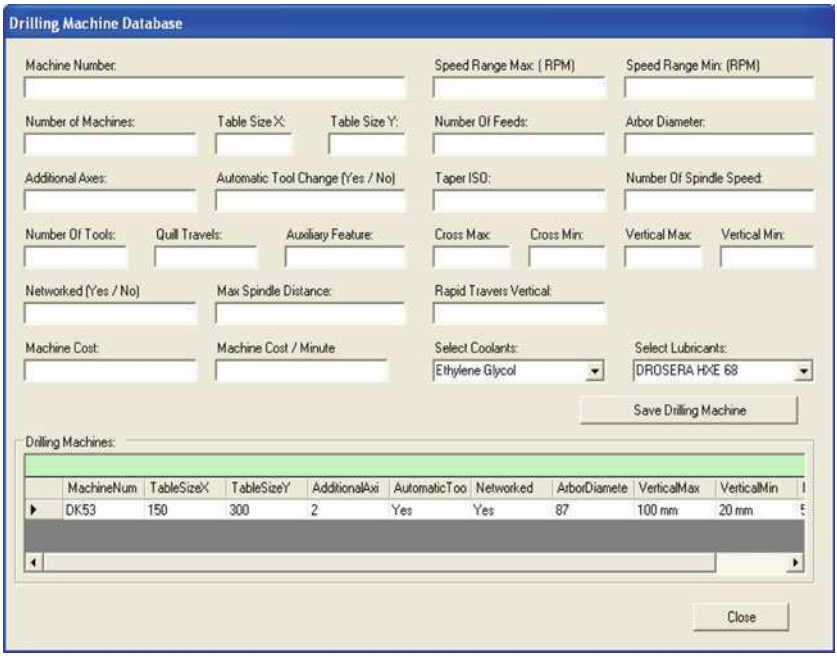


Fig. 1.13 Drilling Database

1.5.2 Tools Database

The cutting tools database refers to specification of mechanical tools, ultrasonic beams, high-intensity light beams and jets of various types. Figures 1.14 and 1.15 provide basic structure for the Tools Database.

1.5.3 Jigs and Fixture Database

All types of jigs and fixtures used in discrete manufacturing may be part of this database. This includes chucks, vises, and clamps . The objects are present in real and virtual form. Figures 1.16 and 1.17 show the basic structure of these databases.

1.5.4 Fluids

Lubricants, coolants, and fluid with, e.g., different electrical properties are generally used in discrete manufacturing. These fluids are lubricant and coolants of

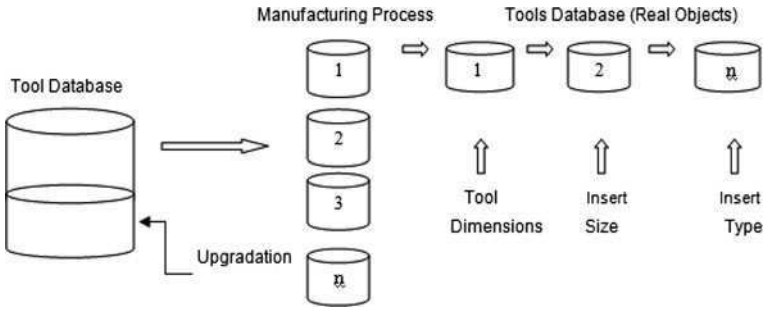


Fig. 1.14 Structure of Tools Database (real object)

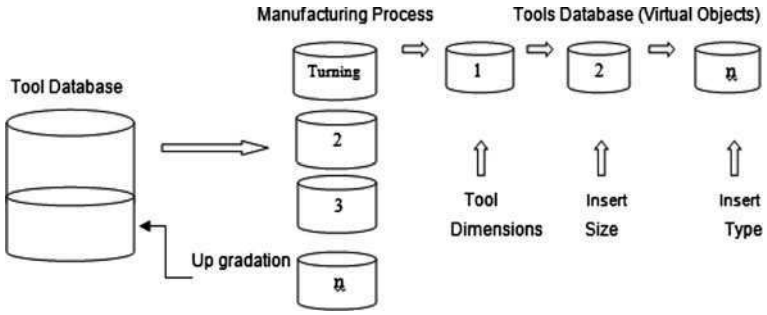


Fig. 1.15 Structure of Tools Database (virtual object) (tool referring to mechanical tools, energy beams, jets)

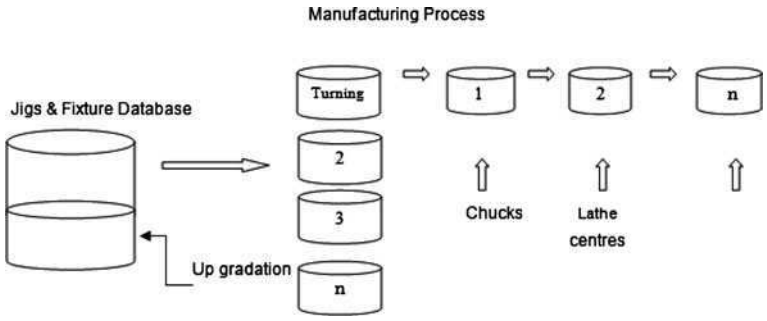


Fig. 1.16 Structure of jigs and fixture database (real object)

different types. Processes such as electro-discharge machining, electro-chemical machining, and others use chemicals of different kind. Availability of these fluids is recorded in the database of fluids of different types. Figure 1.18 shows general structure of these databases:

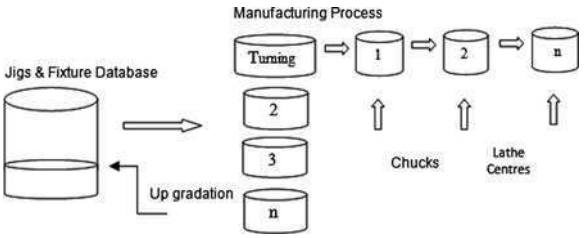
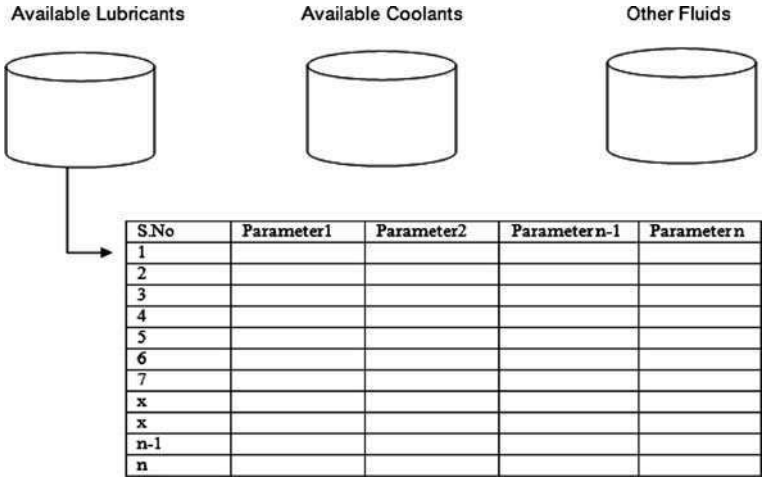


Fig. 1.17 Structure of jigs and fixture database (virtual object)



S.No	Parameter1	Parameter2	Parametern-1	Parametern
1				
2				
3				
4				
5				
6				
7				
x				
x				
n-1				
n				

Fig. 1.18 Liquids used in metal cutting (real objects)

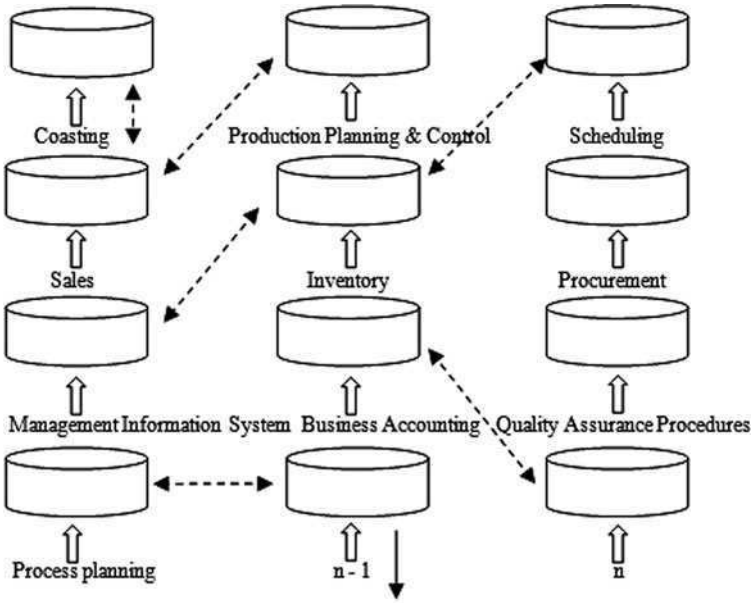
1.5.5 Parameters Related to Intangible Functions

Numbers of intangible functions associated with discrete manufacturing are considered. These intangible functions are listed in Sect. 1.4.

Each intangible function stores legacy data as well as the current value of the related parameters for the purpose of determining micro- or macro-level decision parameter pertaining to an intangible function. These databases with a tentative relation are shown in Fig. 1.19.

1.5.6 3D Graphic Models for Virtual Manufacturing

Several software tools are available to prepare the graphic models of structures, mechanisms, and machines for use in virtual manufacturing organization, refer to "Appendix 1". The development of 3D Graphics Models for virtual manufacturing



S.No	Parameter 1	Parameter 2	Parameter n-1	Parameter n
1				
2				
3				
4				
5				
6				
7				
x				
x				
n-1				
n				

Fig. 1.19 Database for intangible function

is described in "Appendix 2". This process involve the development of the graphic model using tools such as CATIA, Pro-Engineer, IDEAS, Unigraphics and 3D Studio Max, use of Microsoft DirectX technology and a rendering engine such as True Vision SDK.

1.5.7 VMS Graphical User Interface

The virtual manufacturing system is operated through VMS graphical user interface, web's graphical user interface and Vender's graphical user interface, refer to Figs. 1.20, 1.21, and 1.22. The logic used to operate the virtual model is shown in Figs. 1.23 and 1.24.

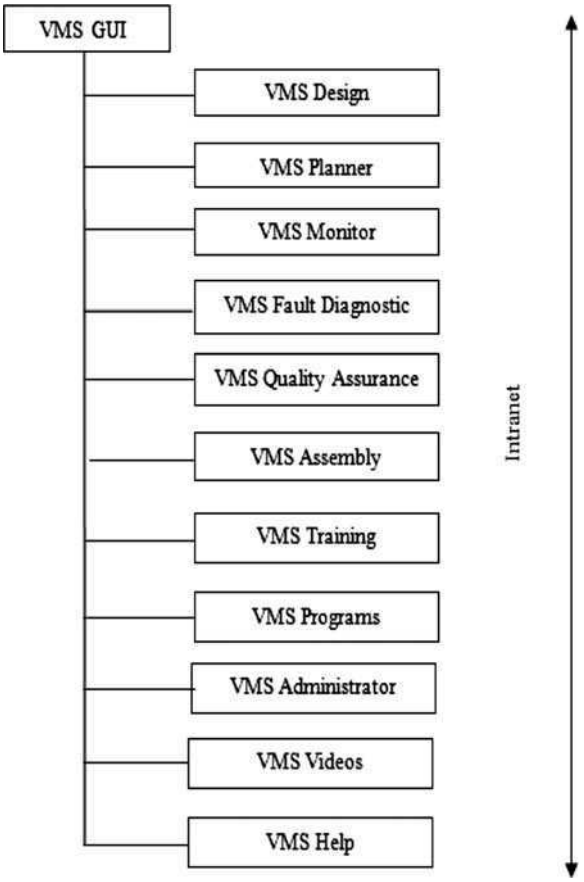


Fig. 1.20 Graphical user interface for virtual manufacturing

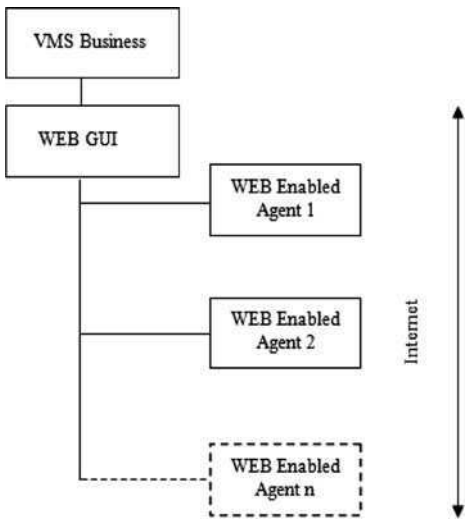


Fig. 1.21 Graphical user interface for VMS Business

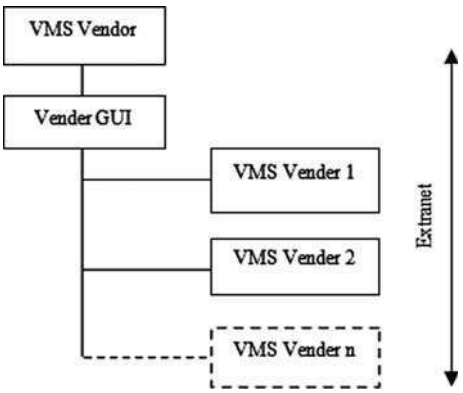


Fig. 1.22 Graphical user interface for VMS Vender

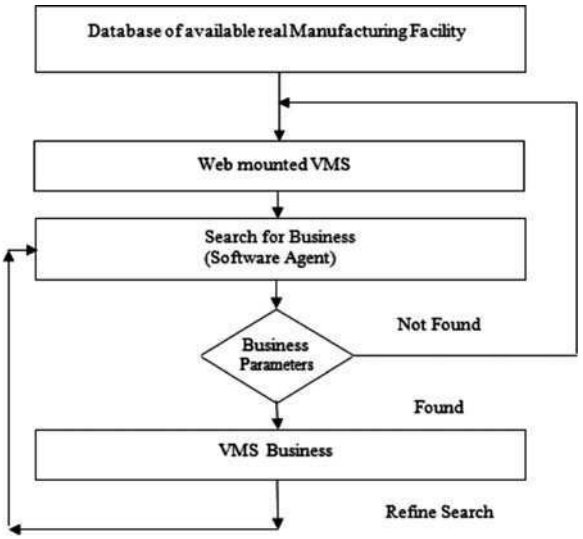


Fig. 1.23 Business logic (partial)

1.5.8 Inference Engines

The VMS graphical user interface provides access to inference engines implementing following VMS services:

- VMS Design
- VMS Planner
- VMS Monitor
- VMS Fault Diagnostic
- VMS Quality Assurance
- VMS Assembly
- VMS Training
- VMS Business

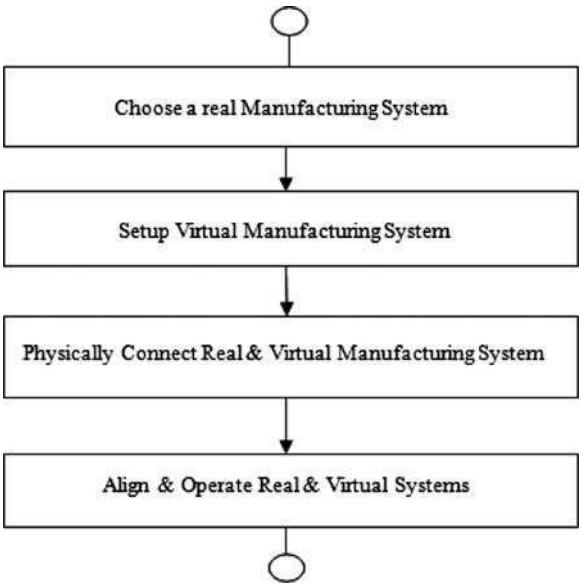


Fig. 1.24 Manufacturing logic (partial)

- VMS Vendor
- VMS Administrator
- VMS Programs
- VMS Videos
- VMS Help

The details of each inference engine are provided in Chap. 11.

1.5.9 AR for Discrete Manufacturing

Figures 1.25 and 1.26 provide schematic for discrete manufacturing in augmented reality environment. Real manufacturing environment is shown in manufacturing process perspective and in product perspective mode. These environments exhibit flow of information among various types of controllers as well as the controller operating the augmented reality.

1.6 Object-Oriented Analysis and Design

Object-oriented analysis and design (OOAD) is the process used by software engineer for analyzing and defining real-world problems and designing the solution in terms of objects. OOAD uses object-oriented perspective to model a system as group of interacting objects that work together to perform required functionality.

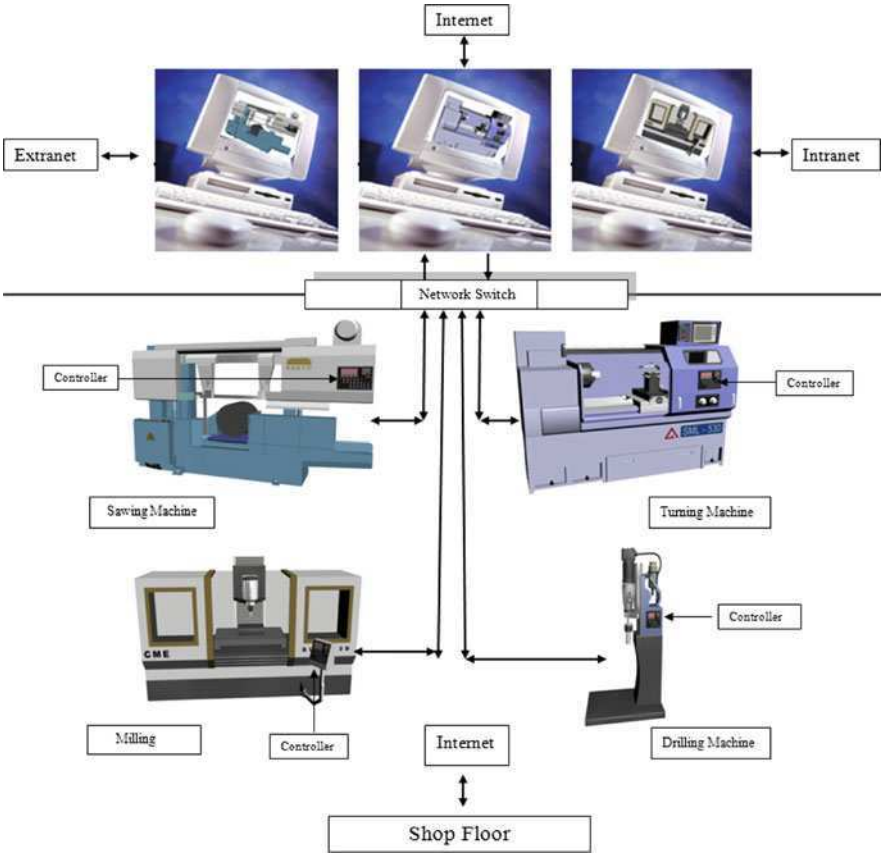


Fig. 1.25 Augmented reality for discrete manufacturing (manufacturing process perspective)

1.6.1 Object-Oriented Analysis

Object-oriented analysis (OOA) is the process in software engineering to identify/define the problem domain in terms of objects. These identified objects represent entities and possess relationships and methods that are necessary for the problem to be resolved.

1.6.2 Object-Oriented Design

Object-oriented design is (OOD) is the process in software engineering to define the solution of the problems and satisfies the requirements identified in OOA process. OOD defines the structure of classes, objects, interfaces, components, and attributes in order to solve the identified problems and requirements.

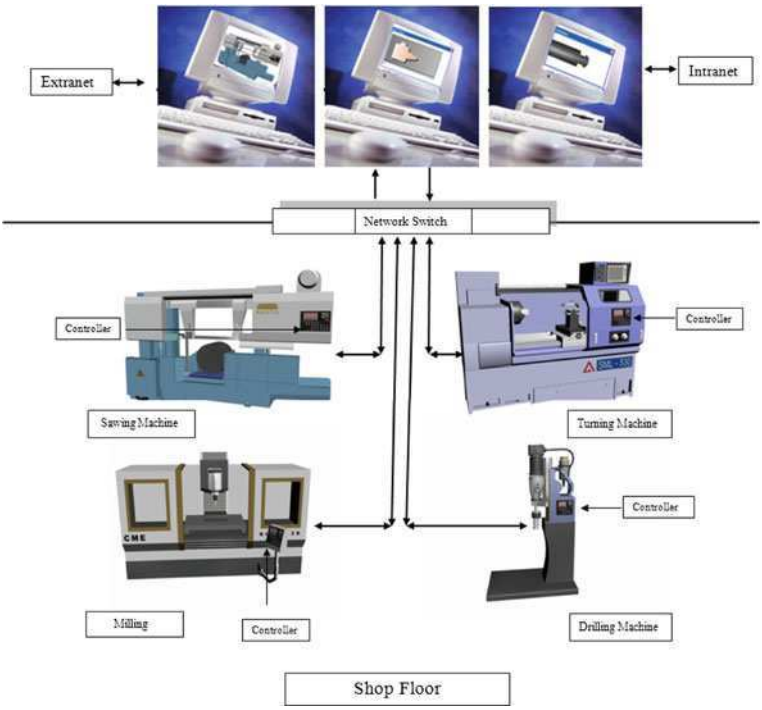


Fig. 1.26 Augmented reality for discrete manufacturing (product perspective)

In order to understand what the object is and what tool is used to model object-oriented software, a brief description of object-oriented programming and unified modeling language is given below.

1.6.3 Object-Oriented Programming

Most of the programming languages such as C++, Java, and C# support object-oriented programming and depend on the concept of Object. An Object is the instance of the class that contains all the program logic, functions, and attributes. These functions and attributes can be accessed by object of that class. Many similar objects can be created from a class but each has separate identity and attributes set by programmer. Object-oriented paradigm is a strong programming technique that gives so many benefits when compared with procedural programming inheritance such as reusability, reliability, robustness, extensibility, maintainability, and encapsulation.

The following are the key concepts of object-oriented programming:

(I) Class

Class is the collection of data, functions, and attribute. Functions and attribute can be accessed by an object to perform operation on data.

(II) Attribute

Attribute is the data value that changes the state or identity of an object. Two objects of same class can be differentiated by attribute. In normal programming term, attribute is called property.

(III) Operation

Operation is the function of an object that it can perform on data. In normal programming term, operation is called method or member function.

(IV) Object

Object is the instance of a class. Suppose that animal is the class and cat, dog, horse are the object of animal class. Objects provide attribute to change its state and function to perform operation on data.

(V) Inheritance

Inheritance provides the concept of parent and child. A child class inherited by its parent class has some common features and functionality as its parent class as well as its own features and functionality. For example, car and bus class can be child classes of vehicle class.

(VI) Component

Component is the collection of related classes. In normal programming term, component is called project.

1.6.4 Unified Modeling Language

The state of objects and classes, requirement, architecture, implementation, and deployment of object-oriented software can be modeled in different ways. The de facto standard to model object-oriented software is the UML, unified modeling language. UML is the standardized graphical language in software engineering for modeling, analyzing, and designing object-oriented software.

UML provide different diagrams to help software engineer to analyze and design object-oriented software. UML diagrams represent structure, behavior, and interaction of and between classes, objects, and components. The mostly used UML diagrams are listed below with a brief description.

(I) Use Case Diagram

Use case diagram is the behavioral diagram that shows the Actor Interaction with software, software behavior, and functionality and dependency with other use cases.

(II) Activity Diagram

Activity diagram shows the step-by-step process of each internal activity in software. Activity diagram is the elaboration of each use case. Activity diagram is the equivalent to flow charts in procedural programming.

(III) Sequence Diagram

Sequence diagram shows the interaction between objects and order in which interaction between objects take place.

(IV) Class Diagram

Class diagram or static diagram shows the structure of class including its members and relationship with other classes.

(V) State Chart Diagram

State chart diagram shows the life cycle of class and its state and event that causes the change of the state.

(VI) Component Diagram

Component diagram shows the structure of the software.

(VII) Collaboration Diagram

Collaboration diagram shows the relationship and interaction between objects.

(VII) Deployment Diagram

Deployment diagram shows the physical structure of the system including hardware and software. This book contains use case diagram, sequence diagram, and class diagram for different cases of augmented reality.

1.7 Computer-Aided Software Engineering Tools for Augmented Reality

Computer-aided software engineering (CASE) is the scientific application of a set of tools and methods to software system. which is meant to result in high-quality, defect–free, and maintainable software products. It also refers to methods for the development of information systems together with the automated tools that can be used in the software development processes. Some typical CASE tools are:

- Configuration management tools
- Data modeling tools
- Model transformation tools
- Program transformation tools
- Refactoring tools
- Source code generation tools
- Unified modeling language.

Many CASE tools not only output code but also generate other output typical of various systems analysis and design techniques such as:

- Data flow diagram
- Entity relationship diagram
- Logical schema
- Program specification
- User documentation.

1.8 Software Development Tools for Augmented Reality

A programming tool or software development tool is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object.

Software development tools can be roughly divided into the following categories:

- Performance analysis tools
- Debugging tools
- Static analysis and formal verification tools
- Correctness checking tools
- Memory usage tools
- Application build tools
- Integrated development environment.

The development of virtual model for describing the functionality of a product to demonstrate operation of manufacturing process, pick and place function or assembly function or to demonstrate the operation of a complete virtual organization requires simulation of all tangible production functions, while mathematical modeling for all related intangible functions. All these suites of programs should be modeled, developed, and run using state-of-the-art software modeling tools, software development technique, and software execution architecture.

Object-oriented software development offers a new and powerful model for writing computer software. This approach speeds the development of new programs and, if properly used, improves the maintenance, reusability, and modifiability of software.

There are almost two dozen major object-oriented programming languages in use today. But the leading commercial object-oriented languages are far fewer in number. These are:

- C#
- Smalltalk
- Java.

These object-oriented languages can be used in virtual reality design in conjunction with the following graphic support tools:

- VRML (virtual reality modeling language)
- OPEN GL (low-level graphics library)
- 3D Studio MAX and Other Graphic Design Tools
- True Vision 3D and other rendering engines.

Development of virtual manufacturing system uses an integrated development environment as listed in Sect. 1.9.7.

1.9 Software Requirement Specification For Discrete Manufacturing

This section is intended for the complete software requirements for the virtual manufacturing system (VMS). The purpose of the virtual manufacturing system is to facilitate the manufacturers to be able to design and model discrete manufacturing systems (e.g. job shop or cellular manufacturing systems), simulate it, and monitor (and control) the manufacturing system. The software supports the users by providing a friendlier user interface (virtual world) contrary to real-life operation or through cumbersome mathematical models. VMS is envisaged to support decision-making processing in the discrete manufacturing. This document is limited to the virtual manufacturing system (VMS) and its components. Throughout this document, all functional and some nonfunctional requirements are formalized defining the scope of the whole product. Figures 1.27, 1.28, 1.29, 1.30, and 1.31 provide pictures of the virtual reality for common constituents of job shop or the cellular manufacturing system. Figures 1.32 and 1.33 detail the design of a virtual job shop and a virtual cellular manufacturing system, respectively.

1.9.1 Purpose

The primary purpose of the virtual manufacturing system is to give the manufacturers a powerful tool that help them model new setups, modify existing setups, simulate the processes to find bottlenecks as well as irregularities and ultimately keep track of the performance of local or geographically distant discrete

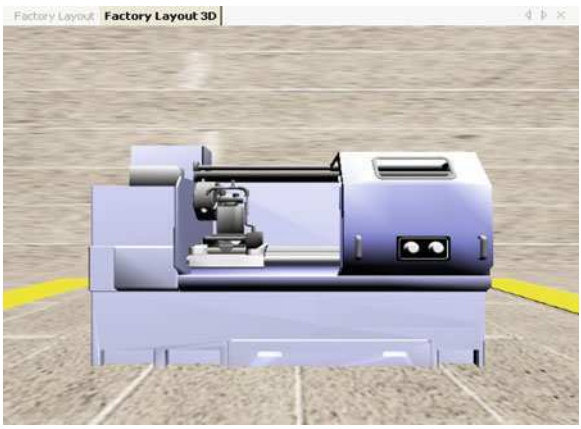


Fig. 1.27 Virtual model of a SML—530 milling machine

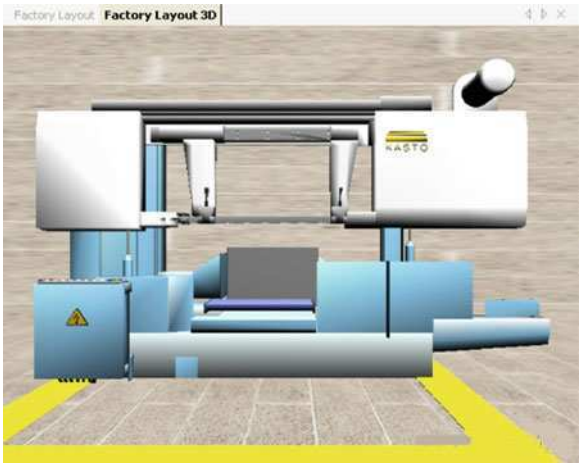


Fig. 1.28 Virtual model of KASTO band saw machine

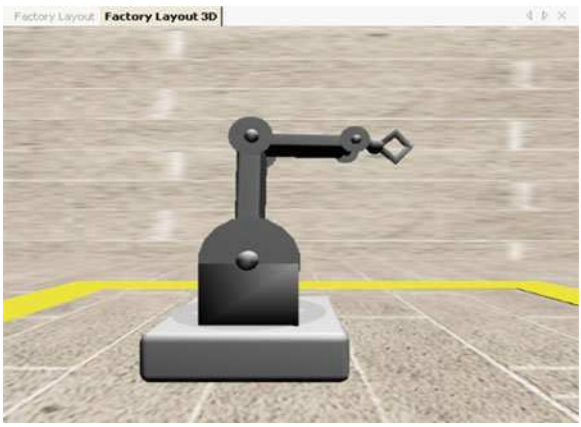


Fig. 1.29 Virtual model of a revolute robot

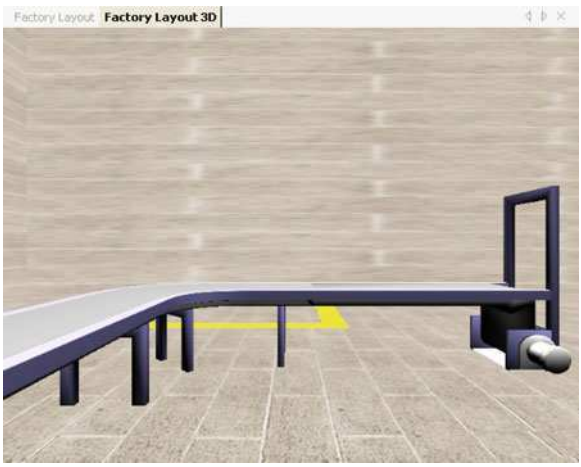


Fig. 1.30 Virtual model of conveyor system

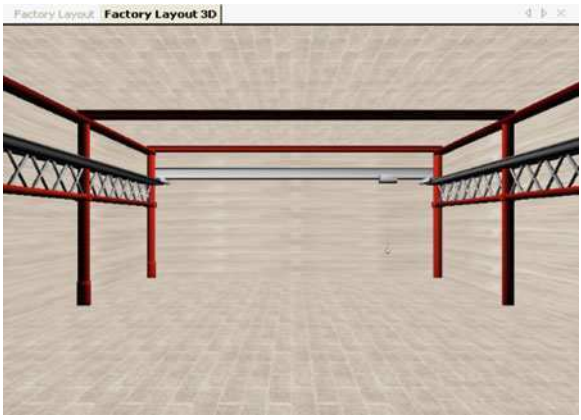


Fig. 1.31 Virtual model of a gantry crane

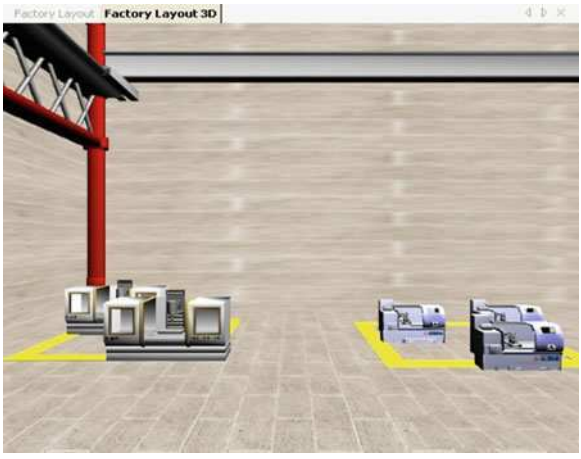


Fig. 1.32 Virtual representation of a job shop

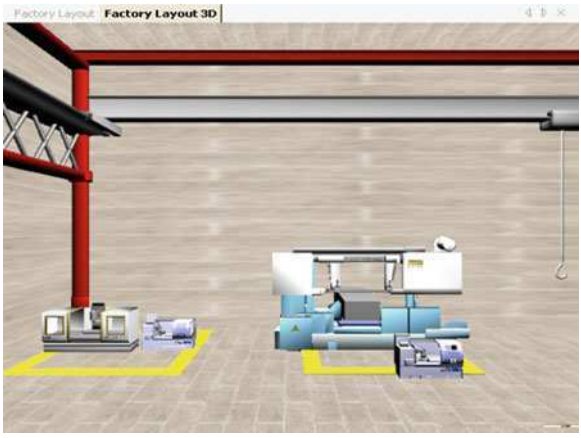


Fig. 1.33 Virtual model of a cellular manufacturing system

manufacturing setups. The VMS provides micro- and macro-level decision parameters to managers at various levels so as to help them adopt an optimal course of action.

1.9.2 The Concept

A discrete manufacturing operation involves tangible activities such as machinery and its operation, use of tools and measurement gadgets, use of pick and place technology, and use of storage and transportation equipment. On the other hand, the intangible part includes services such as process planning, scheduling, inventory, management information system, and business accounting.

The principle of operation of the virtual organization has a seven-tier strategy to meet its objectives. In the first tier, which is also the core tier, the basic decision-making foundation based on Enterprise Resource Planning (ERP) philosophy is laid down. In the second tier, a job shop or cellular manufacturing facility layout is defined. All the tangible competencies either have a mathematical model to provide the necessary input to the virtual manufacturing facility or are represented by a simulation showing actual manufacturing operation and the state of the product. The state of the product is initially defined in Direct X format. It may also be imported from CAD/CAM application packages in formats such as IGES, STEP or VDA-FS. In all intermediate stages of manufacturing, the product is represented by the Direct X neutral format in its current shape. Only the final shape conforms to originally designed Direct X and may be exported to CAD/CAM application package in the above-defined formats. In the third tier, distributed computing architecture, grid computing, acts as the inference engine as per the instruction of global management software for the defined production competencies. In the fourth tier, communication strategy is implemented through development of intranet between the production competencies. The fifth tier

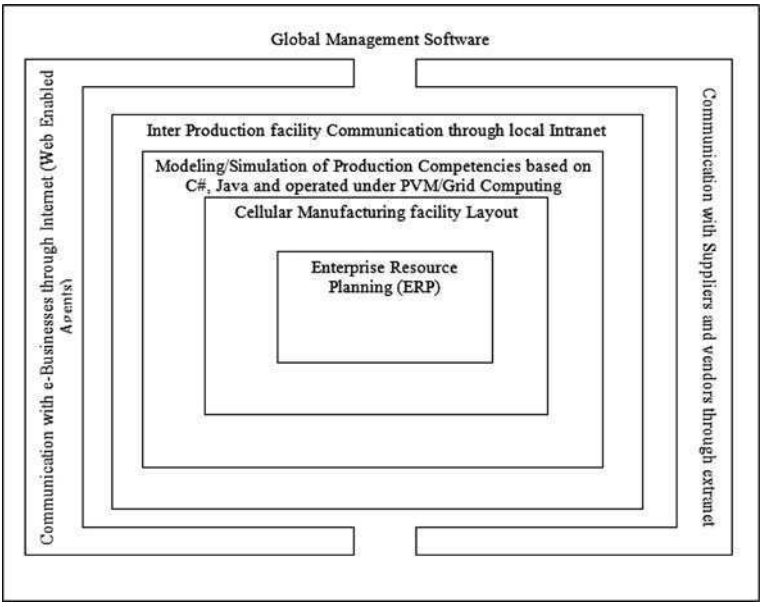


Fig. 1.34 An abstraction of VMS concept

enables communication between suppliers and vendors and the virtual factory using an extranet among the participating stakeholders. The sixth tier provides the communication channel to virtual organization with the external businesses using web-enabled software agents using Internet. The seventh tier comprises the management software that controls all competencies, database, and administrative parameters as and when required. The management software is the top layer of user interface to the virtual organization. An abstraction of above concept is provided in Fig. 1.34.

The discrete manufacturing system used is either a job shop or a cellular system. All the simulations and mathematical models, virtual product in Direct X format, and other parameters are stored in an object-oriented database. All the communication within the virtual facility is carried out through intranet. Figure 1.35 provides an abstraction of the modeling of some intangible competencies.

Figure 1.36 gives the general layout for the operation of virtual factory with competencies following the principle of a particular production philosophy.

The development of software for simulation and models employs object-oriented technique. Unified modeling language (UML) is used to model the software while the software development was carried out through Rational Unified Processes (RUP). Object-oriented tools C# is used to develop models and simulation with the support of dot Net framework. 3D Studio Max is used as the graphic modeler, while True Vision is used as a rendering engine. UML static structure and UML use case diagram for the augmented reality of CNC Milling is shown in Fig. 1.37.

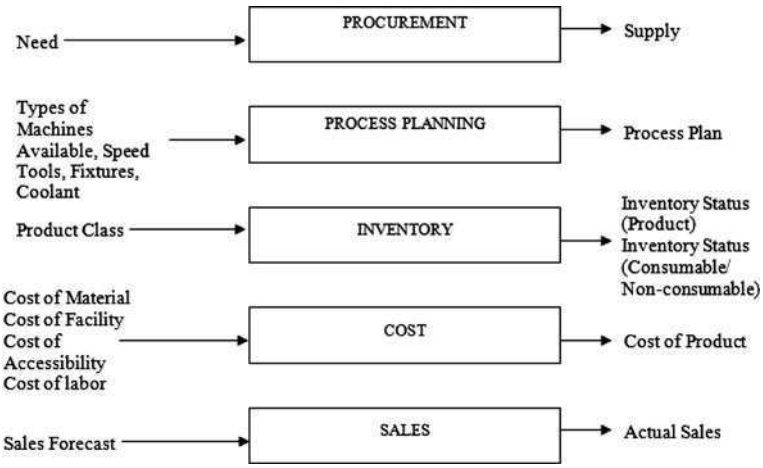


Fig. 1.35 General model of some intangible competencies

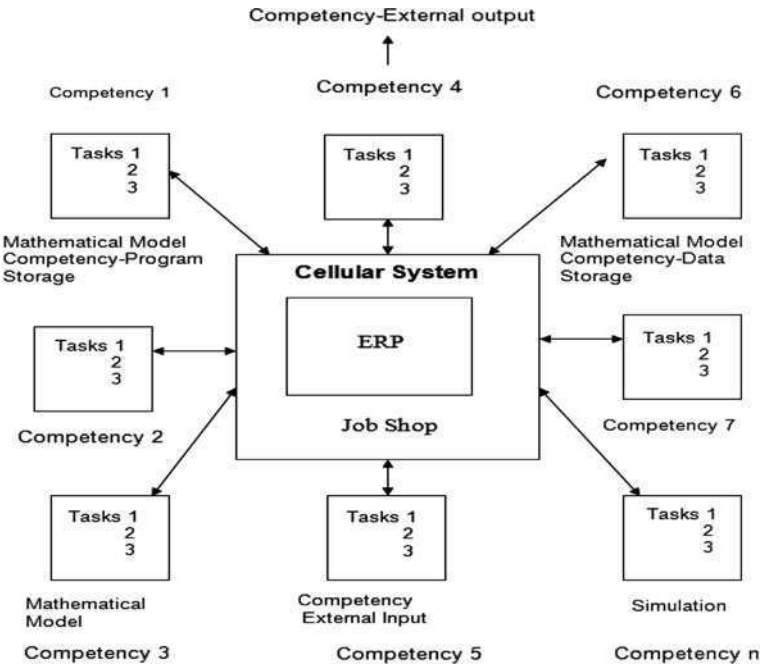


Fig. 1.36 An abstraction of proposed virtual manufacturing facility

Due to the inherent flexibility of the virtual organization, the flow of information in a virtual organization can be reorganized conveniently. Internal and external information flow in a virtual organization is shown in Fig. 1.38.

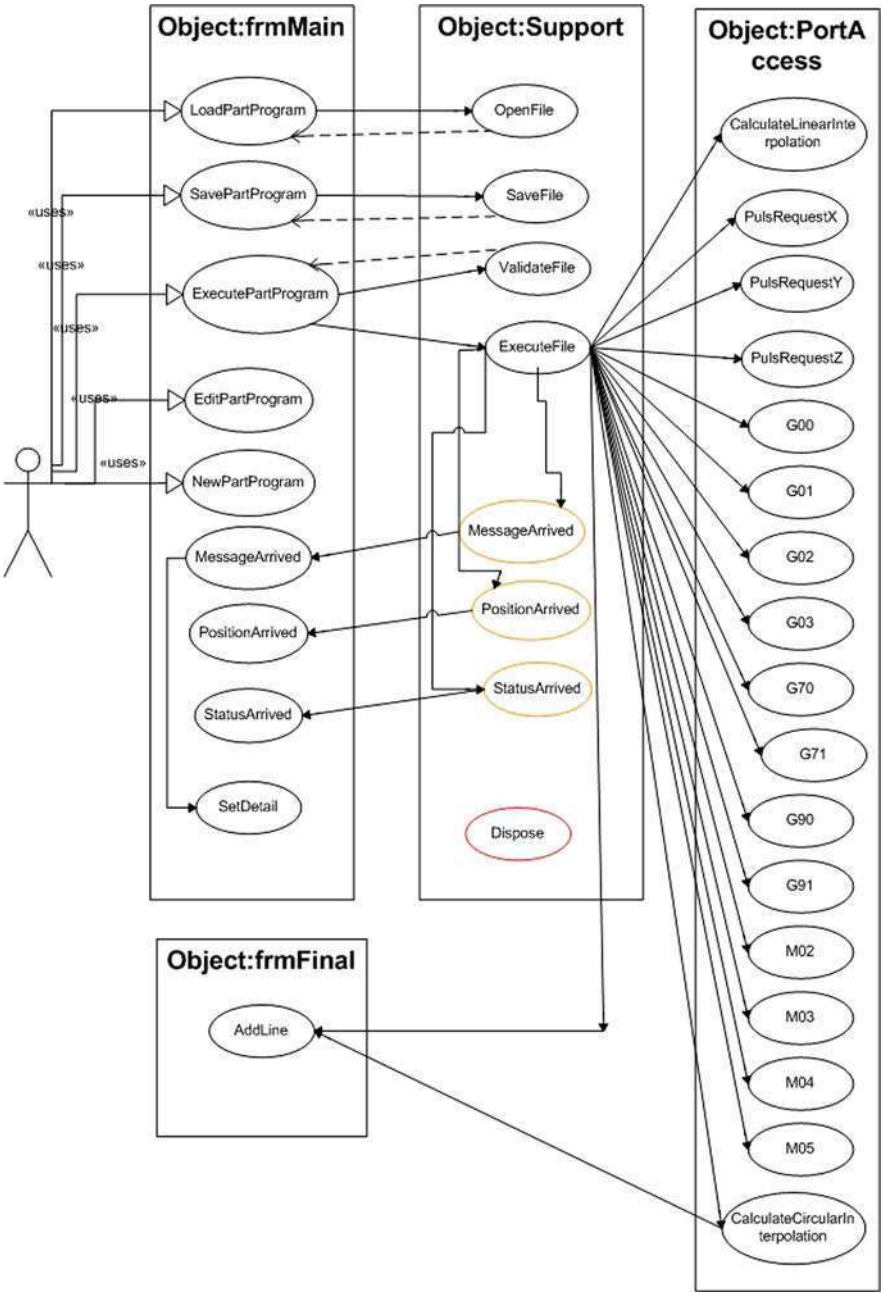


Fig. 1.37 CNC system UML static structure and UML use case diagram

1.9.3 Scope

The scope of the project is based on the implementation of the concept provided in the earlier sections to design and simulate complex tangible and intangible processes encountered in discrete manufacturing. The potential benefit of the augmented reality for discrete manufacturing is to promote increased productivity for the enterprise. The long-term benefits of the proposed system are to help improve the performance of the all manufacturing units especially companies with offshore manufacturing or third-party outsourcing.

1.9.4 System Overview

The service modules of the virtual manufacturing system (VMS) are:

- VMS Design
- VMS Planner
- VMS Monitor

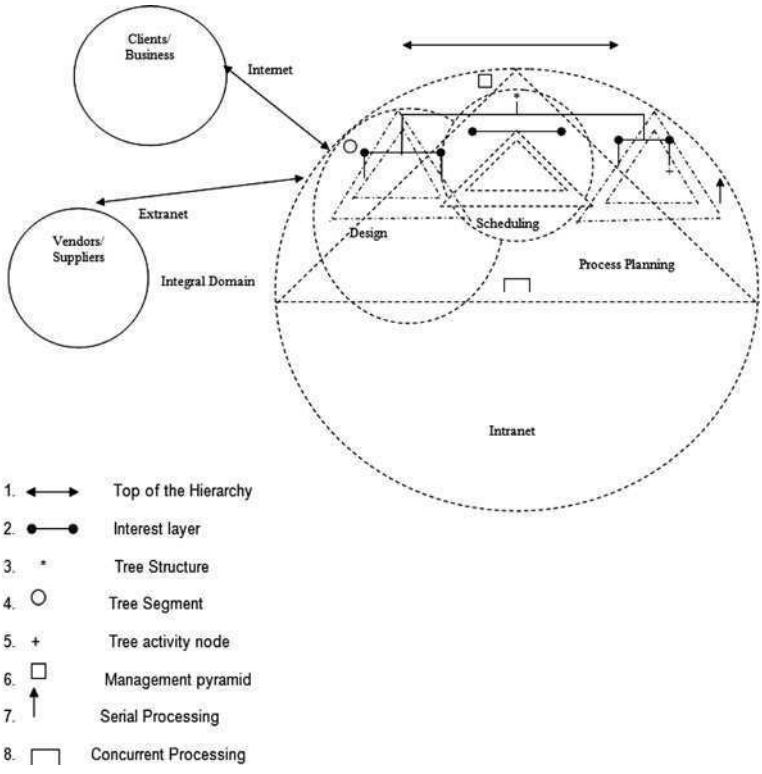


Fig. 1.38 Communication in virtual manufacturing system

- VMS Fault Diagnostic
- VMS Quality Control
- VMS Assembly
- VMS Training
- VMS Business
- VMS Venders
- VMS Administrators
- VMS Programs
- VMS Videos
- VMS Help

The functional requirements of VMS modules are:

1.9.4.1 VMS Design

This module allows design of virtual discrete manufacturing system (VDMS). The virtual organization layout can be defined, and position of required machine tools maybe explored for optimum output from the manufacturing system.

1.9.4.2 VMS Planner

In this module, a designed VMDS can be loaded with raw material to produce product/s. Information on job status at various virtual machine tools, jobs in queue, and job completion time may be obtained without performing operations at the real manufacturing facility.

1.9.4.3 VMS Monitor

VMS monitor shall ideally be used once the VMS design and VMS planner have achieved satisfactory results. This module acts as the human–computer interface to the real manufacturing facility and executes commands as optimally determined by VMS Planning for achieving maximum profit margins. VMS monitor operates on real-time hardware interface with the equipment in the factory. The hardware interface may be based on standard computer numerical control (CNC), programmable logic control (PLC), embedded system, industrial manipulator and/or Supervisory Control and Data Acquisition (SCADA).

1.9.4.4 VMS Fault Diagnostic

This module operates in conjunction with VMS monitor and is capable of identifying any operational breakdown in the manufacturing chain as designed using VMS design and planned using VMS planner. VMS monitor continuously scans

the control interface of the manufacturing equipment and reports corresponding fault if any.

1.9.4.5 VMS Quality Control

VMS quality control operates on the output of VMS monitor. It has a direct interface with quality control gadgets such as coordinate measuring machines.

1.9.4.6 VMS Assembly

VMS assembly adopts stationary workstation stationary operator or moving workstation stationary operator principles. These are the most common assembly operations in practice.

1.9.4.7 VMS Training

VMS training is an offline training facility for the manufacturing organization. It allows training of manpower to operate computerized manufacturing machinery, material handling equipment, and pick and place technology. True virtual models of all the equipment in the factory are available in the database and the user may explore various possibilities of using this equipment efficiently.

1.9.4.8 VMS Business

VMS business performs the e-commerce function. It is operated through semantic web with widely adopted manufacturing ontology. Software agents are the key components of this module.

1.9.4.9 VMS Venders

VMS venders allow communication specially between the venders and the VMS monitor.

1.9.4.10 VMS Administrator

VMS administrator allows definition of new user accounts and permission for a particular user.

1.9.4.11 VMS Programs

VMS programs provide programs for CNC-based machines, revolute robot, conveyor belts, and gantry cranes.

1.9.4.12 VMS Videos

VMS videos provide a collection of videos related to the virtual manufacturing organization.

1.9.4.13 VMS Help

Help is provided globally. VMS collects these pointers for help on various systems to facilitate the users of the VMS help.

The detail description of VMS modules is provided in Chap. 11.

1.9.5 Overall System Description

The virtual manufacturing system is to be used in an interactive environment where there is a continuous interaction between the virtual manufacturing system, real manufacturing system, vendors, and target businesses. The system can be defined as a whole by the following diagram (Fig. 1.39):

The service modules of virtual manufacturing organization are used to accomplish the interaction between virtual manufacturing systems, real manufacturing system, and the target business. Services modules have their own scope of operation and related interactions.

The description of the general operation of virtual manufacturing system to perform VMS planner service is provided through flowchart of Fig. 1.40. The description of flowchart symbols used frequently in this book is provided as "Appendix 3".

The microscopic details of the working of VMS planner are detailed in the sequence diagram given as Fig. 1.41.

1.9.6 Project Functions

The main function that the software performs is that it keeps a record of all the information pertaining to the machines and work in pieces in the form of variables, images, and neutral files representing the geometry in various databases. The records are provided to the VMS control events in the form of images and reports.

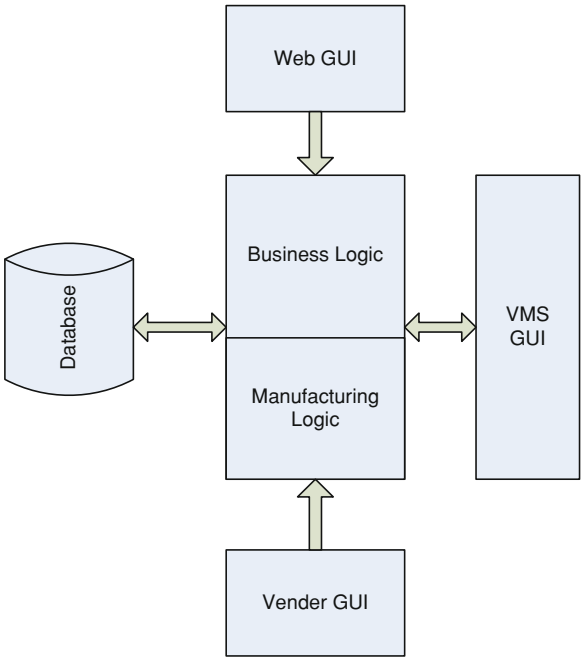


Fig. 1.39 Overall system description

The software also manages the inventory as well as the sales data for the client. In addition, some reports are generated that helps in having an outlook of the whole project.

1.9.7 System Interfaces

User-friendly and easy-to-use interfaces are provided to the user of the system at different levels. The software system receives requests from users and based on it, the simulation is run and reports are generated. These reports can be used to alter the parameters of machines and processes used, and simulation may be run again to achieve the desired efficiency. User interface provided for VMS planner are depicted in Figs. 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49, 1.50, and 1.51.

1.9.7.1 Hardware Interfaces

The VMS monitor and VMS fault diagnostic modules require real-time control of hardware interfaces in the factory. The required circuit diagrams are shown in respective chapters. Currently, system operates on a stand-alone personal computer with constrained I/O resources. The ideal hardware architecture is grid computing cluster using a communication network.

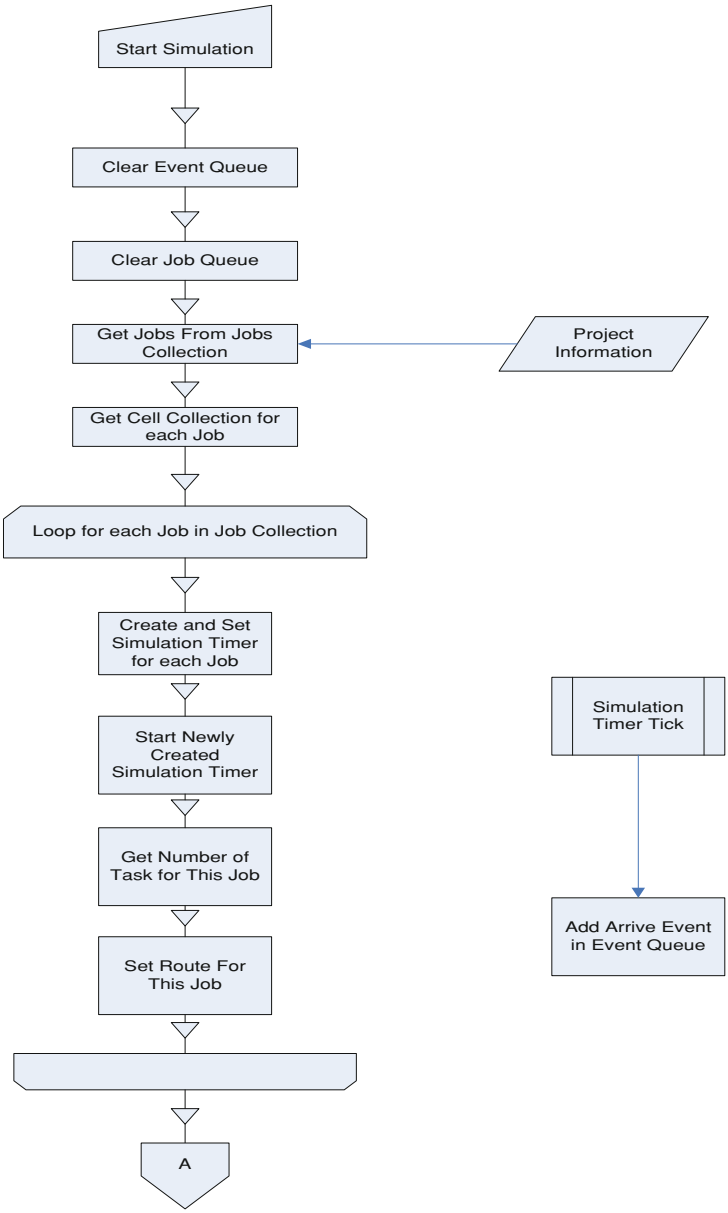


Fig. 1.40 VMS planner operation

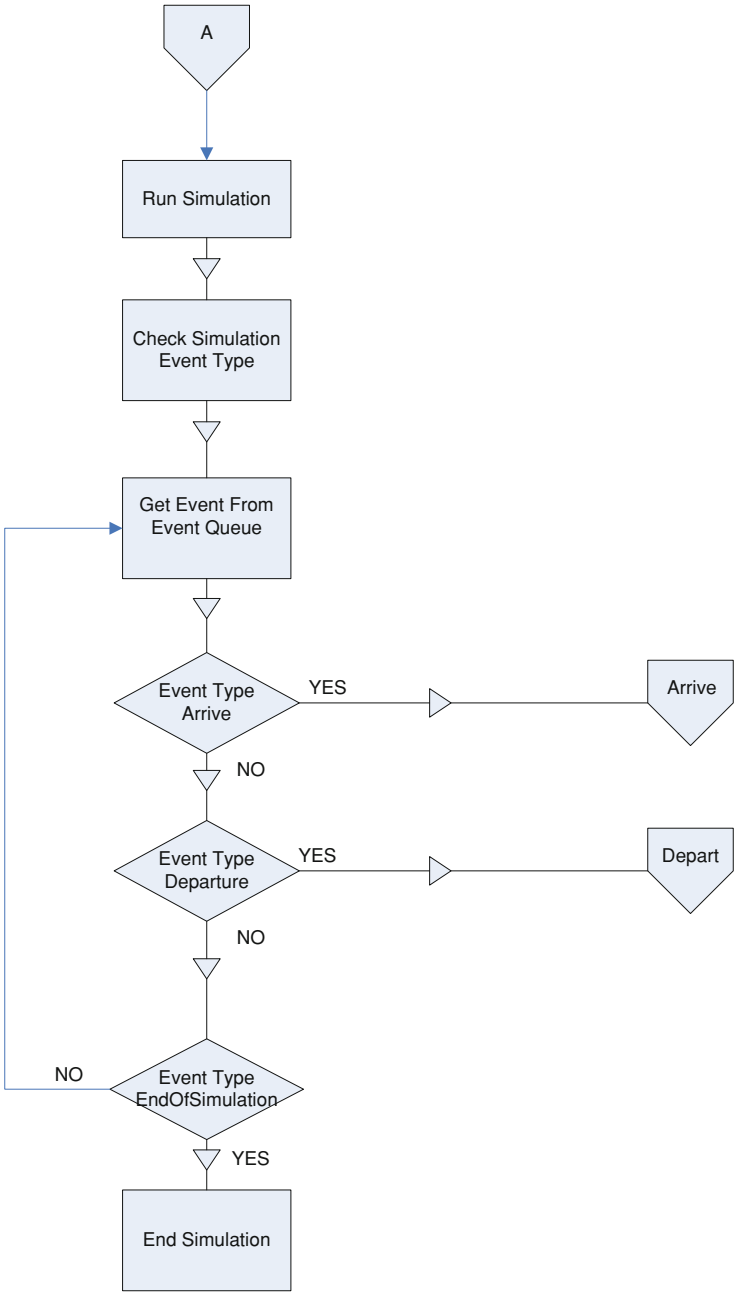


Fig. 1.40 Continued

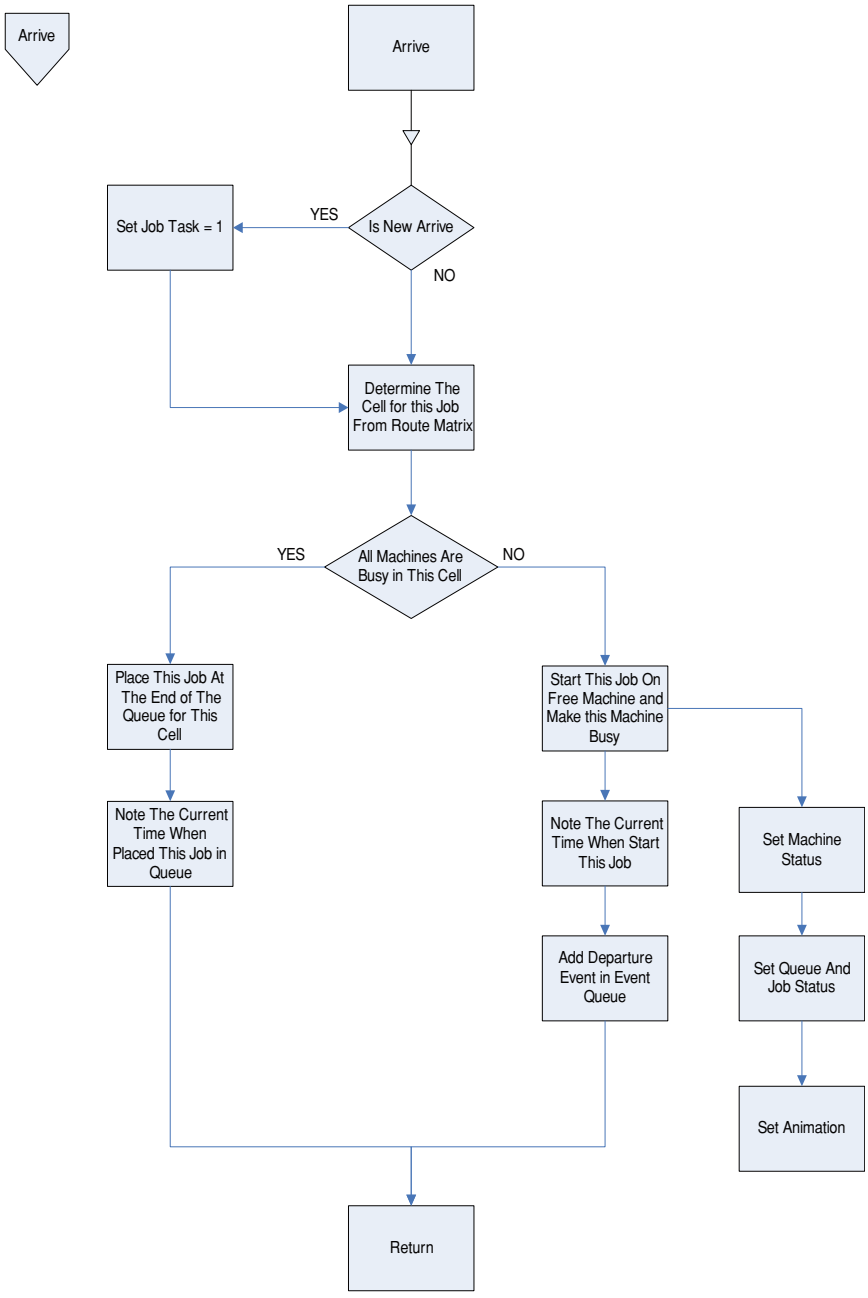


Fig. 1.40 (Continued)

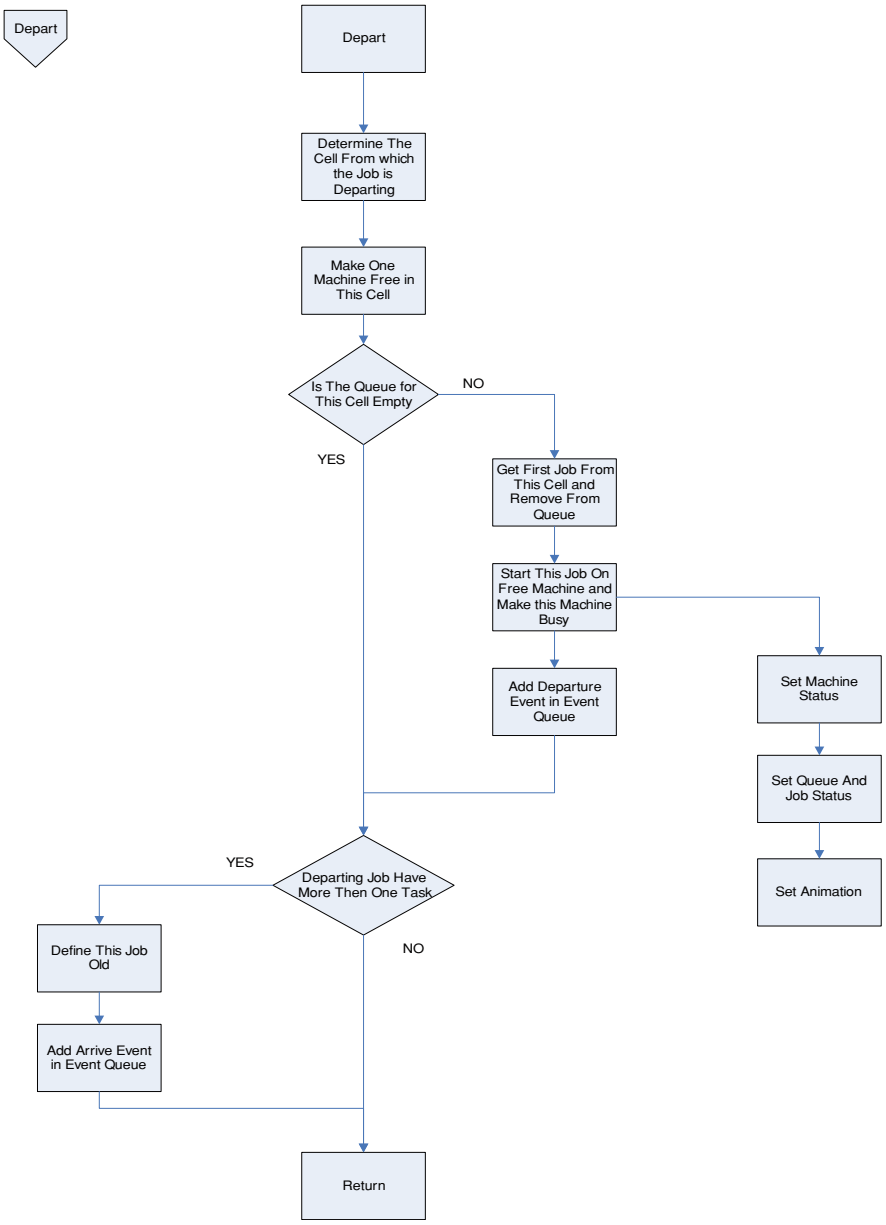


Fig. 1.40 (Continued)

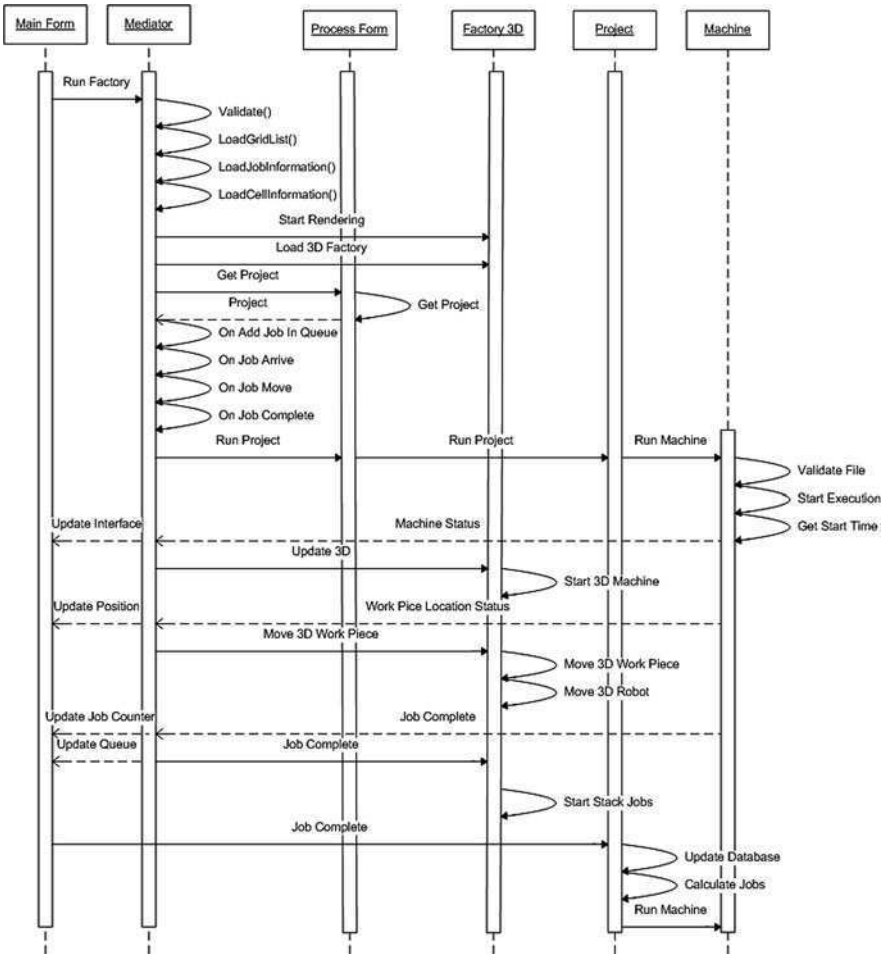


Fig. 1.41 VMS planner sequence diagram

1.9.7.2 Software Interfaces

Software interfaces for inference engines, graphic displays, and databases are developed using the integrated development environment generated through following software tools.

(I) Unified Modeling Language

Refer to Sect. 1.6

Fig. 1.42 Machine tool database window (drag & drop to factory layout)

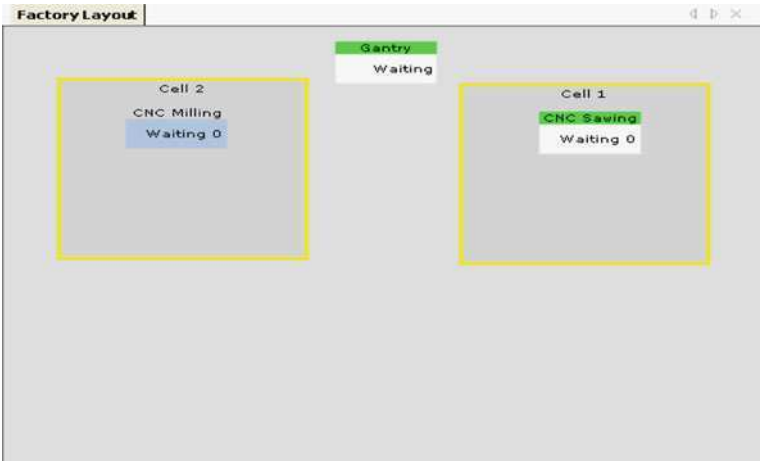


Fig. 1.43 Factory layout window (drag & drop from machine tool database window)

(II) Visual Studio

The visual studio development system is a comprehensive suite of tools designed to help software developers create innovative, next-generation applications. It is the perfect work environment for application developers.

(III) 3D Studio MAX

This is a full-featured 3D modeling, animation, and rendering solution. Enhanced toolsets enable create 3D environment. It manages complex scenes and takes advantage of improved software interoperability and pipeline-integration support.

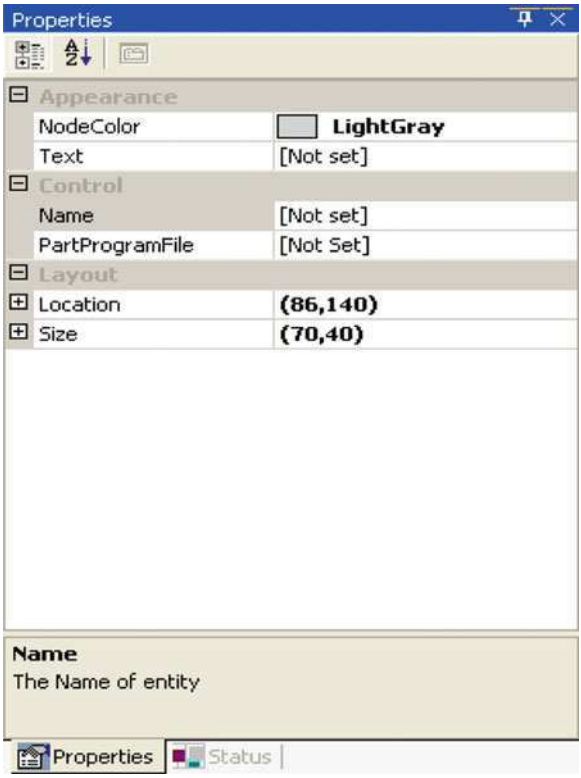


Fig. 1.44 Machine tool/ equipment properties window

(IV) True Vision 3D SDK

It is a top-quality 3D middleware for 3D application and game development. It allows integrating 3D content into existing applications with ease. It provides 3D Engine, Sound and Video Media Engine, or Networking Engine.

(V) Direct X

Microsoft DirectX is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.

(VI) C#

C# (pronounced "C Sharp") is a multi-paradigm programming language, encompassing imperative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within the dot NET initiative and later approved as a standard by ECMA (ECMA-334) and ISO (ISO/IEC 23270). C# is one of the programming languages designed for the common language infrastructure.

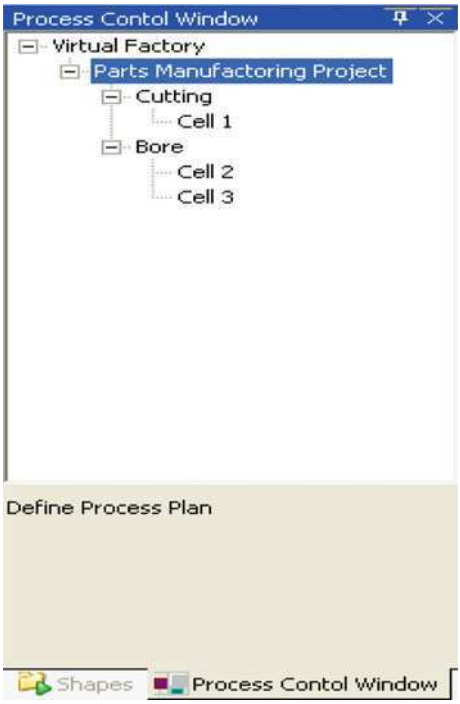


Fig. 1.45 Process control window (creation of projects and jobs)

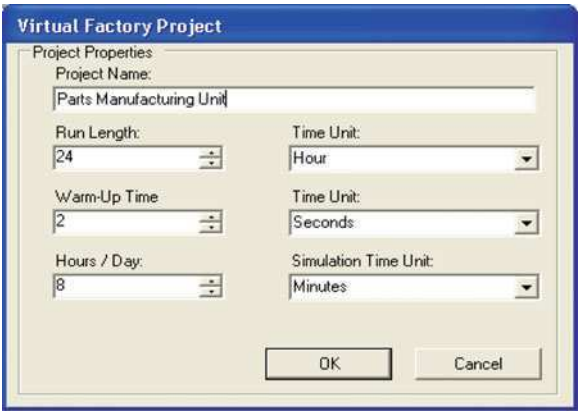


Fig. 1.46 Project setup window

(VII) MS ACCESS

Microsoft Office Access is a pseudo relational database management system from Microsoft. It combines the relational Microsoft Jet Database Engine with a graphical user interface and software development tools.

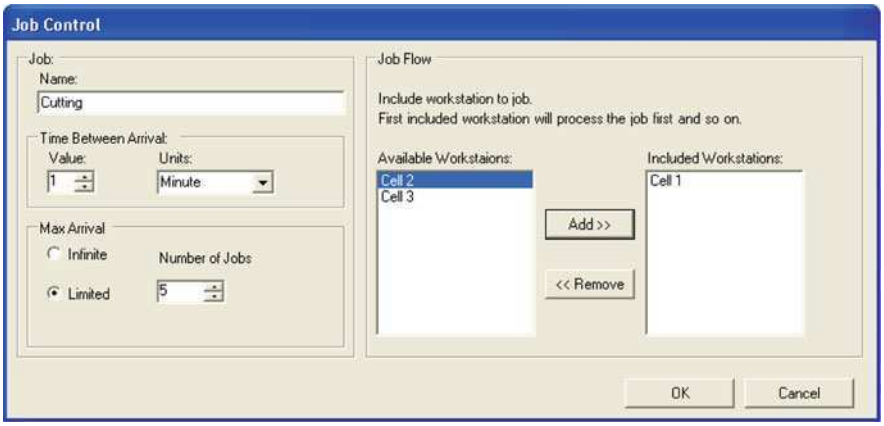


Fig. 1.47 Job control window (job assignment to a cell)

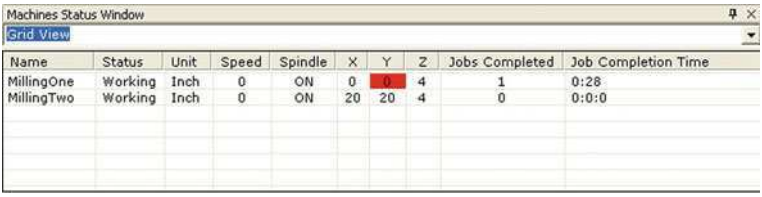


Fig. 1.48 Machine status window (report)

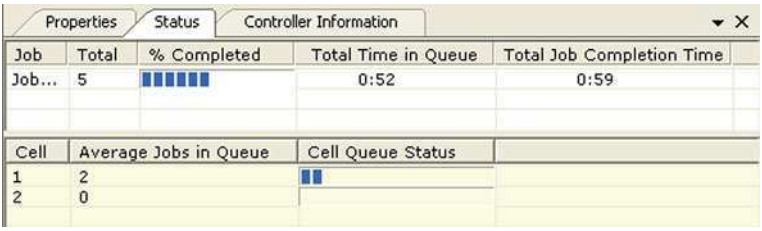


Fig. 1.49 Jobs in queue (report)

(VIII) J2SE

Java Platform, standard Edition, is a widely used platform for programming in the Java language. J2SE is used primarily for writing applets and other Java-based applications.

(IX) JENA Semantic Web API

Jena is a Java framework for building semantic web applications. Jena uses RDF (resource description framework) to represent all information, and future versions will adopt other semantic technologies as the specifications solidity.

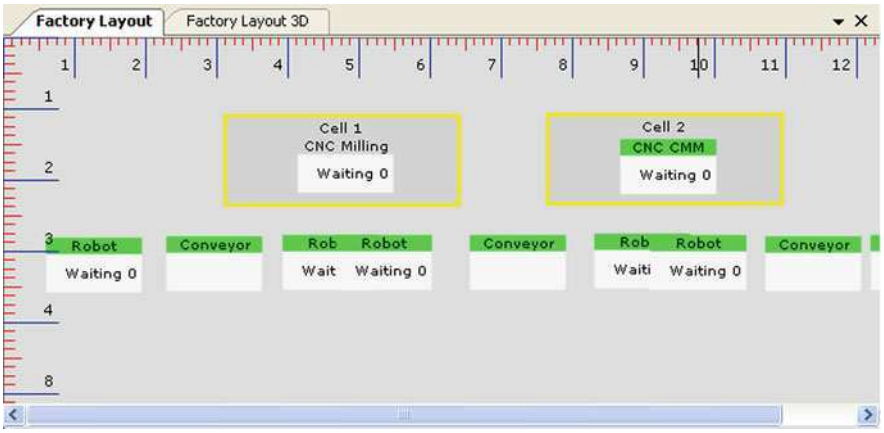


Fig. 1.50 2D view of VMS planner functionality

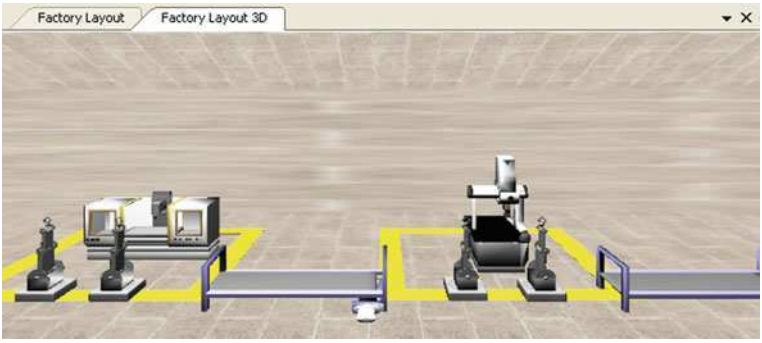


Fig. 1.51 3D View of VMS planner functionality

1.9.8 Requirements Specification

Functional and nonfunctional requirement specifications of VMS are given below:

1.9.8.1 Functional Requirements

No	Requirement	Description
1	VMS Design	Refer to Sect. 1.9.4.1
2	VMS Planner	Refer to Sect. 1.9.4.2
3	VMS Monitor	Refer to Sect. 1.9.4.3
4	VMS Fault Diagnostic	Refer to Sect. 1.9.4.4
5	VMS Quality Control	Refer to Sect. 1.9.4.5
6	VMS Assembly	Refer to Sect. 1.9.4.6

(continued)

(continued)

No	Requirement	Description
7	VMS Training	Refer to Sect. 1.9.4.7
8	VMS Business	Refer to Sect. 1.9.4.8
9	VMS Venders	Refer to Sect. 1.9.4.9
10	VMS Administration	Refer to Sect. 1.9.4.10
11	VMS Videos	Refer to Sect. 1.9.4.11
12	VMS Help	Refer to Sect. 1.9.4.12
13	VMS Programs	Refer to Sect. 1.9.4.13

1.9.8.2 NonFunctional Requirements

No	Requirement	Description
1.	Reliability	The system should be reliable as the data it provides should be accurate.
2.	Availability	The down times of the system can decrease customer interest in the system.
3.	Security	The system should be made secure as possible as no unauthorized user access is supported into the system.
4.	Maintainability	The design of the system (VMS design, VMS planner, VMS monitor, VMS fault diagnostic and VMS training modules) should be made in such a way that there is always room for improvement and the maintenance should be easier and faster.
5.	Portability	The portability of the system should be taken into consideration.
6.	Performance	Performance of the system should be of highest standards. The need for best performance is required due to the critical nature of the system.
7.	Supportability	The system should be able to support the newer versions of applications, protocols etc.
8.	Licensing Requirements	The software used in the systems should be licensed in a proper manner and licenses of copyrighted software should be obtained from the vendors.
9.	Legal, Copyright And Other Notices	The appropriate notices regarding the legal matters and claims on the project would be made public with the consultation of the Legal Advisor.
10.	Applicable Standards	The possible standards should be maintained so that the product can fulfill the interoperability requirements.

1.10 Operation of the VMS

The operation of VMS requires an augmented reality setup as shown in Figs. 1.25 and 1.26. The VMS software modules are accessible by the user through the top layer of the seven-tier software model described in detail in the

previous sections. These modules are also the entry point for accessing relevant modules of VMS through intranet and Internet. Web services shall be based on semantic web operated under widely used manufacturing ontology. Extranet services are used by the vendors to provide the supplies to the manufacturing organization.

Some of the operations performed by the various VMS modules are described using example cases. These use cases covering metal cutting and textile industry are detailed through Figs. 1.52 and 1.53. Figure 1.52 presents VMS planning for metal cutting job shop with the objective function being the optimized operations detail using parameters such as number of machines, the distance between the machines, the queue size, the buffer size and the operation time required by the server. Figure 1.53 provides the online operation detail through VMS monitor in a cellular manufacturing system. The VMS monitor is capable of providing current status of the manufacturing system. Figures 1.54 and 1.55 provides the physical model of the proposed virtual manufacturing organization. Figure 1.56 details system layout of the virtual manufacturing organization. Figure 1.57 gives the GUI for VMS. Figures 1.58 and 1.59 provides local and global operation of VMS.

1.11 Computer Hardware Configuration for Virtual Manufacturing

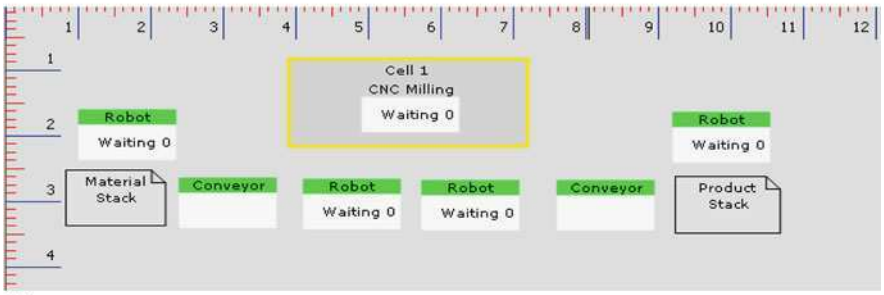
There is an enormous amount of data processing during the operation of VMS based on augmented reality for discrete manufacturing. Stand-alone computer cannot cope with this data flow. Parallel computing or distributed computing is the preferable solution. Use of distributed computing in grid computing format is more economical.

Grid computing links geographically dispersed machines across different administrative domains to create a virtual supercomputer. This virtual machine emerges as a single pool of computing resources and has the computational power to do jobs that stand-alone machine cannot.

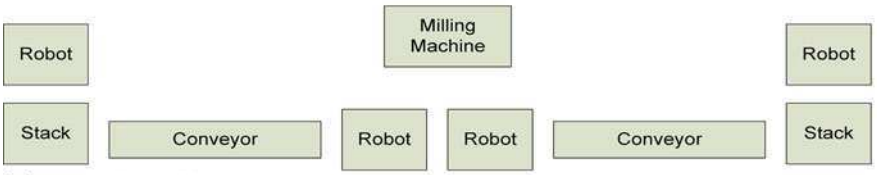
Grid deployments usually pursue either a research-oriented or enterprise-oriented track, reflecting the diverse needs and goals of the two main consumers of grid technology. Research-oriented grids are cobbled together by universities and laboratories, often employing open-source software and government subsidies.

Enterprise grids also create virtual supercomputers by uniting distributed resources to appear as one pool of computing power. However, while enterprise grids may be geographically divided, they are logically located within the same walls (for example, behind the same firewall). And unlike research grids, the extra capacity that is produced is only available within the organization.

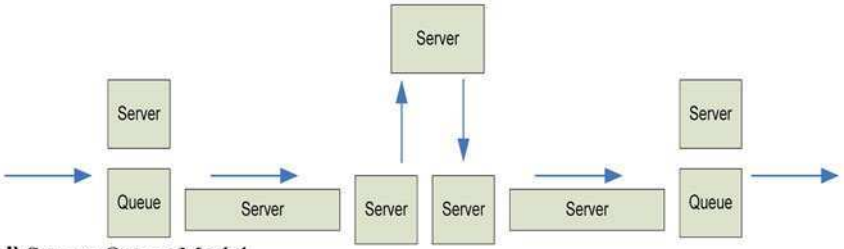
(a) 3-D Layout



(b) 2-D Setup

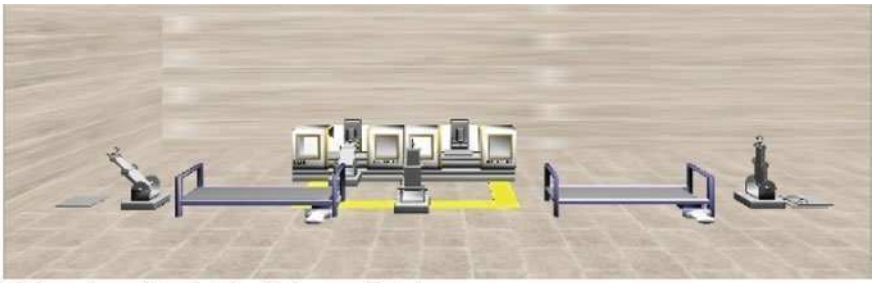


(c) General Detail

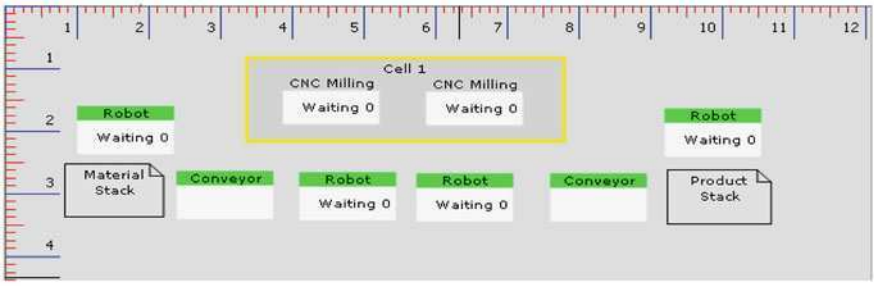


(d) Server-Queue Model

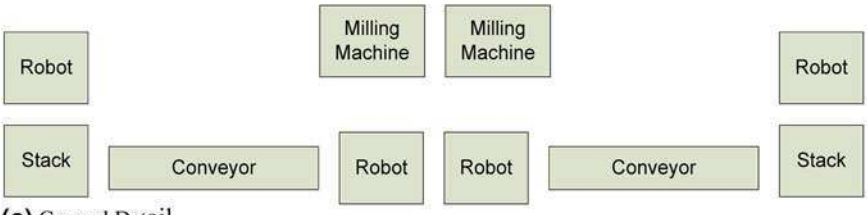
Fig. 1.52 Use case 1 (metal cutting in a job shop) **a** 3D layout, **b** 2D setup, **c** general detail, **d** server-queue model. Objective: VMS planning detail through intranet/Internet



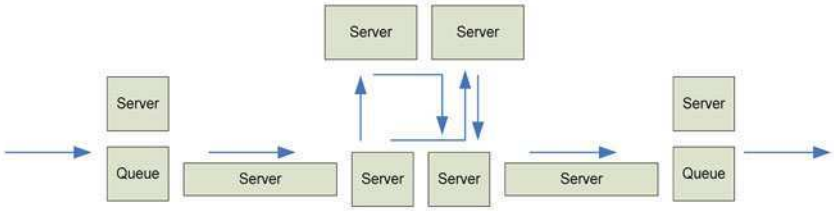
(a) Operation of the virtual cellular manufacturing



(b) 2-D Setup



(c) General Detail



(d) Multiple Server Multiple Queue Simulation Model

Fig. 1.53 Use case 2 (metal cutting in cellular manufacturing). **a** Operation of the virtual cellular manufacturing. **b** 2D setup, **c** general detail, **d** multiple server multiple queue simulation model. Objective: VMS operation detail through intranet/Internet

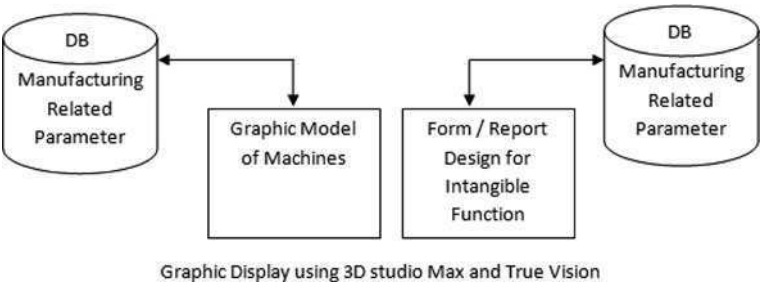


Fig. 1.54 Software components of virtual manufacturing system

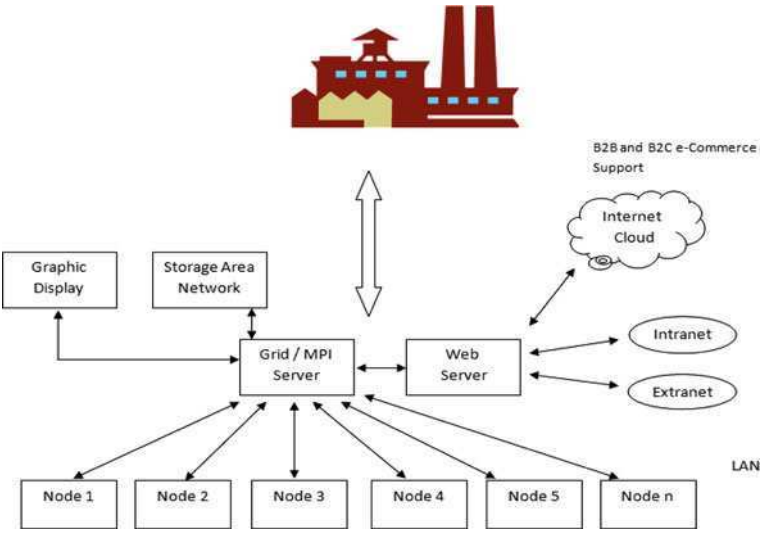
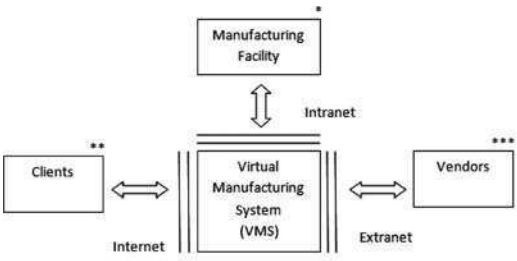


Fig. 1.55 Hardware components of virtual manufacturing system

Fig. 1.56 System layout of the virtual manufacturing system



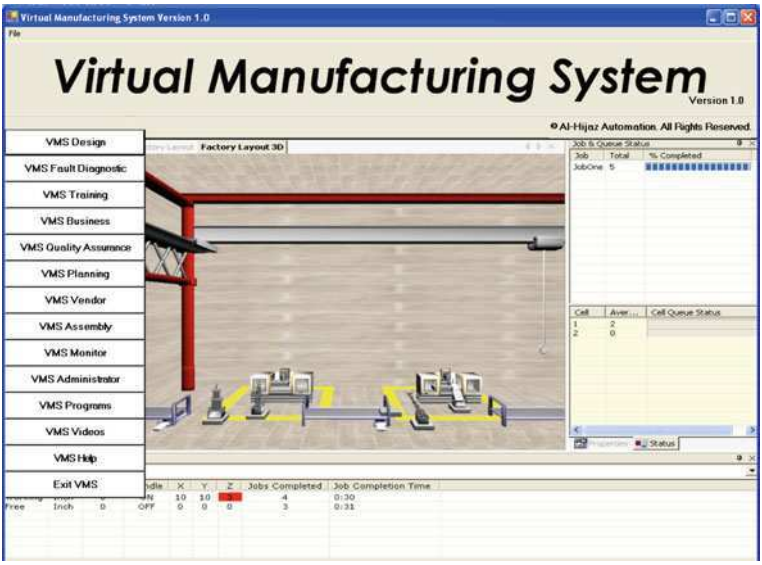


Fig. 1.57 User interface to VMS software

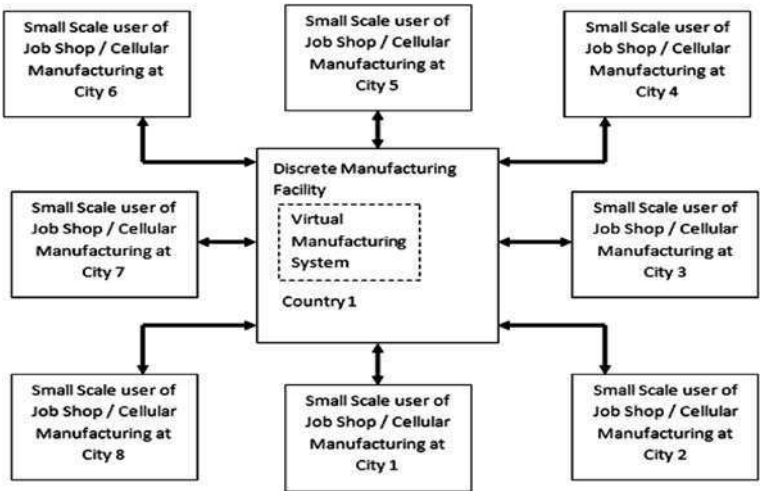


Fig. 1.58 Local operation of the virtual manufacturing system

1.12 Communication Methodology for Virtual Manufacturing

Telecommunication network links nodes (machines, processes, MDI consol) to the inference engine in an augmented reality for discrete manufacturing. Depending on the scope of operation, following types of networks may be employed:

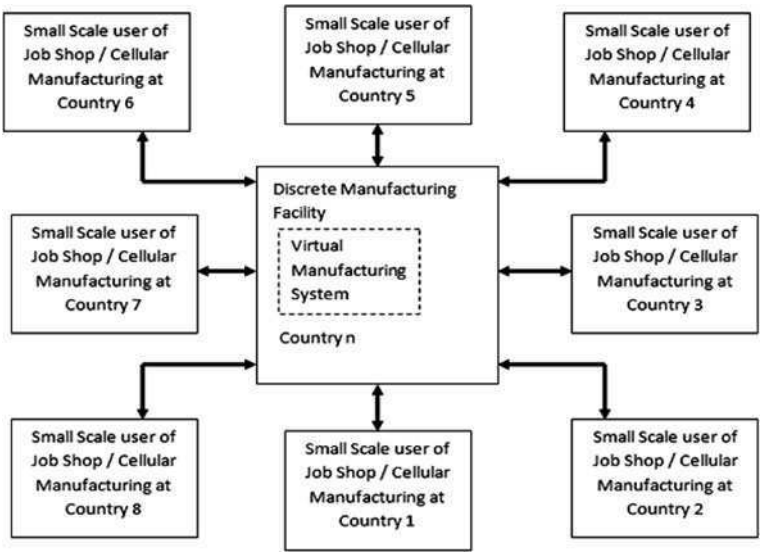


Fig. 1.59 Global operation of the virtual manufacturing system

- Local area network
- Metropolitan area network
- Wide area network.

The LAN size is limited to few kilometers covering from a home network to a company's office. LAN allows sharing resources such as hardware, software, and data. The current LAN data rates are normally 100–1000 Mbps. A metropolitan area network (MAN) is a network with a size between a LAN and a WAN. IT has high-speed connectivity and have endpoints spread over a city or part of city. A wide area network (WAN) is used for long-distance transmission of data, image, audio, and video over a large geographic area that may comprise a country, a continent or even the whole world. The common topologies used in networking are mesh, star, bus, and ring.

Bibliography

1. Liverani A et al (2004) A CAD-augmented reality integrated environment for assembly sequence check and interactive validation. Concurr Eng Res Appl 67–77
2. Alting L (1994) Manufacturing engineering processes. Marcel Dekker, New York
3. Askin RG, Standridge CR (1993) Modeling and analysis of manufacturing systems, 1st edn. Wiley, New York
4. Azuma R (2001) Recent advances in augmented reality. IEEE Comput Graph Appl 21(6):P34–P47
5. Berger AS (2005) Embedded system design: an introduction to processes, tools and techniques. CMP Books, New York

6. Booch G, Ranbaugh J, Jacobson I (2005) The unified modeling language user guide. Pearson Education, India
7. Booch G et al (2007) Object oriented analysis and design with applications. Addison Wesley, USA
8. Boyer SA (2004) SCADA: supervisory control and data acquisition, 3rd edn. ISA—The Instrumentation Systems and Automation Society, USA
9. Central Machine Tool Institute (1985) Machine tool design handbook. CMTI, Bangalore
10. Chawla R (2001) A virtual environment for simulating manufacturing operations in 3D. In: Proceedings of the 2001 winter simulation conference
11. Chryssolousis G (1992) Manufacturing systems: theory and practices. Springer, New York
12. Ghang T, Wysk RA (1985) An Introduction to the automated process planning system. Prentice Hall, New Jersey
13. Heim MR (1993) The metaphysics of virtual reality. Oxford University Press, USA
14. Holder R, Gregory B, Wilhelm W (2005) Augmented reality projects in the automotive and aerospace industries. IEEE Comput Graph Appl 48–56
15. http://en.wikipedia.org/wiki/case_tools#CASE_tools
16. http://en.wikipedia.org/wiki/Grid_Computing
17. http://en.wikipedia.org/wiki/programming_tools
18. http://en.wikipedia.org/wiki/virtuality_continuum
19. http://en.wikipedia.org/wiki/virtual-reality
20. http://encarta.msn.com/thesauras/thesaurus.html
21. http://www.eia.org
22. http://www.iec.ch/
23. http://www.jisc.go.jp/eng/index.html
24. Chong JWS et al (2009) Robot Programming using augmented reality: an interactive method for planning collision-free paths. Robot Comput Integr Manuf 689–701
25. Jazar RN (2007) Theory of applied robotics: kinematics, dynamics and control. Springer, New York
26. Jingzhou S et al (2009) Study on the perception mechanism and method of virtual and real objects in augmented reality assembly environment. In: IEEE conference on industrial electronics and applications, ICIEA 2009, pp 1452–1456
27. Kalpakjian S, Schmid SR (2003) Manufacturing process for engineering materials, 4th edn. Pearson Education Inc, New Jersey
28. Khan WA, Raouf A (2006) Standards for engineering design and manufacturing. CRC Press/ Taylor & Francis, USA
29. Magoules F et al (2009) Introduction to grid computing. Chapman and Hall/CRC, USA
30. Ong SK et al (2007) Augmented reality aided assembly design and planning. CIRP Ann Manuf Technol 56:49–52
31. Ong SK (2009) A mixed reality environment for collaborative product design and development. CIRP Ann Manuf Technol 139–142
32. Ong SK (2008) Augmented reality applications in manufacturing: a survey. Int J Prod Res 2707–2742
33. Smid P (2003) CNC programming handbook, 2nd edn. Industrial Press Inc, New York
34. Stefan N et al (2006) Augmented reality as a comparison tool in automotive industry. In: Proceedings ISMAR 2006, fifth IEEE and ACM international symposium on mixed and augmented reality, pp 249–250
35. Valentino JV, Godenberg J (2005) Introduction to computer numerical control. Pearson Education Inc, New Jersey
36. Webb JW, Reis RA (2003) Programming logic controllers: principles and application, 5th edn. New Jersey, USA
37. Wilhelm D et al (2005) Virtual and augmented reality support for discrete manufacturing system simulation. Comput Ind 56:371–382

Chapter 2
Manufacturing Processes and Systems

2.1 An Overview of Discrete Manufacturing Processes

A large number of manufacturing equipment exist for the implementation of basic discrete manufacturing processes into production machinery. Kalpakjian and Schmid (2003) provide a list of discrete manufacturing processes available under different categories of manufacturing, which are used in actual production. These major categories are

1. Metal forming processes
2. Bulk deformation processes
3. Sheet Metal forming processes
4. Metal removal processes
5. Metal joining processes
6. Processing of Polymers and reinforced plastics
7. Processing of Metal Powders, Ceramics, Glasses, Composites and Super Conductors
8. Thermal Treatment of materials
9. Surface Treatment of materials
10. Fabrication of Micro-mechanical and Microelectronic Devices
11. Nonconventional processes

The equipment used under each category of discrete manufacturing is further classified according to[1]:

1. Type of manufacturing equipments that are discrete products capable of manufacturing components, assemblies, structure, mechanism and machine (Kalpakjian and Schmid 2003)
2. Size

[1] Extracted with permission from Khan WA, Raouf A (2006) Standards for engineering design and manufacturing, Taylor & Francis/CRC, USA.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_2, © Springer-Verlag London Limited 2011

3. Accessories
4. Operation parameters, and
5. Type of control

Each manufacturing process for implementation into mechanical artifacts has three distinct features:

1. The inputs to the equipment: raw material type, form, and feeding mechanism; final dimension of the product; energy source; and other auxiliaries.
2. The process implementation allowing transformation of raw material into required size, shape, and surface finish using tools (e.g., tools as used in metal cutting, high-energy beams, or various types of jets); utilizing tool- and work-holding devices; measuring devices and manufacturing instructions. In process supplies such as lubricating oil and coolants may also be used.
3. The output from the equipment comprising a component (the building block of structure, mechanisms or machines); and scrap.

There may be several features present at the production machinery to make the task of manufacturing simpler, easy to control and result in high production rate.

The design of manufacturing equipment relies on a systematic approach to design and considers a few or all of the following special design requirements.

1. Structural consideration for machine frame and assemblies;
2. Thermal effects on the equipment;
3. Noise emission from the equipment;
4. Vibration in the machinery;
5. Environmental effects on the equipment;
6. Geometric and kinematics behaviors of the manufacturing equipment;
7. Static and dynamic behaviors of the equipment;
8. Availability of Computer Numerical Control (CNC) or process control using Programmable Logic Controllers (PLC);
9. Electronic circuitry for implementing control features;
10. Foundation and Installation requirements

Like any other discrete product, the manufacturing equipment commonly utilizes standard mechanical components, machine elements, control elements, electrical and electronic components, and software components.

Special assemblies and other accessories utilized at the construction of manufacturing equipment may also include

1. Tool—Referring to mechanical tool, beams, jets, etc.);
2. Tool–holding devices;
3. Work-holding devices;
4. Lubricating oil pump assembly;
5. Coolant circulation pump assembly;
6. Material handling equipment; and
7. Scrap handling equipment

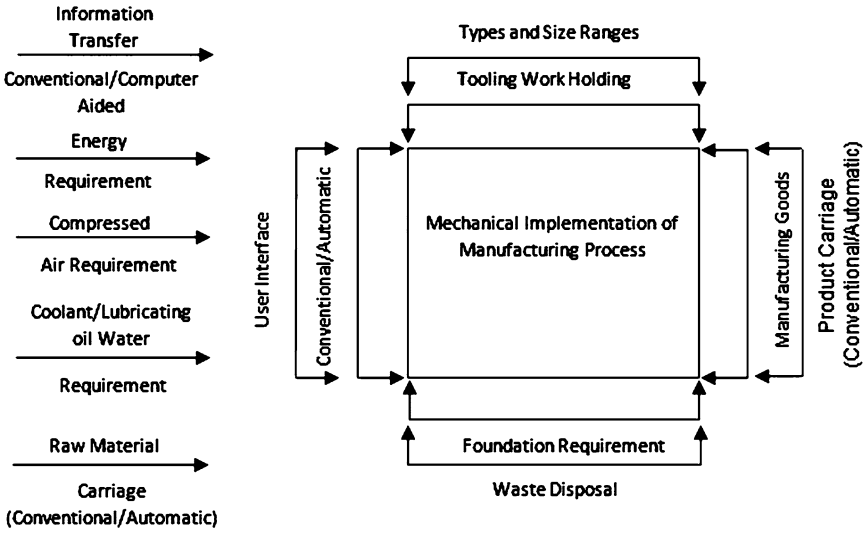


Fig. 2.1 Manufacturing process implementation into equipment

A schematic detail implementation of the manufacturing process into production equipment is presented in Fig. 2.1.[2]

At present, special consideration is given to the automation, as the use of Computer Numerical Controls and Programmable Logic Controllers have taken firm roots.

2.2 Discrete Manufacturing Systems[3]

A manufacturing system comprises manufacturing equipment arranged in certain fashion. Tangibly, these manufacturing systems have a physical layout while intangible production control operates on production philosophies. Other important elements of the manufacturing systems are methods of information, energy, and material transfer. The physical layout of the discrete manufacturing systems is normally divided into two areas:

1. Processing area;
2. Assembly area

---

[2] Extracted with permission from Khan WA, Raouf A (2006) Standards for engineering design and manufacturing, Taylor & Francis/CRC, USA.

[3] Extracted with kind permission of Springer Science + Business Media from 'Manufacturing systems—theory and practice' (1992), pp 223–230; and 'Types of manufacturing systems', Chryssolouris G, Figs. 5.2–5.9 and Tables 5.1 and 5.2, ©1992 Springer Verlag; New York, Inc.

The processing area is used for manufacturing the components, while the assembly area is meant for assembling the product.

2.2.1 Job Shop

In industrial practice, there are four basic approaches to structuring the processing area for discrete manufacturing [3]: the job shop, project shop, cellular system, and flow line.

In a job shop, Fig. 2.2, machines with the same or similar material processing capabilities are grouped together: the lathes form a turning work center, the milling machines form milling work center, and so forth.

The job shop is characterized by its high flexibility in the production of various types of products while the volume of production in this type of manufacturing system is low—refer to Fig. 2.5.

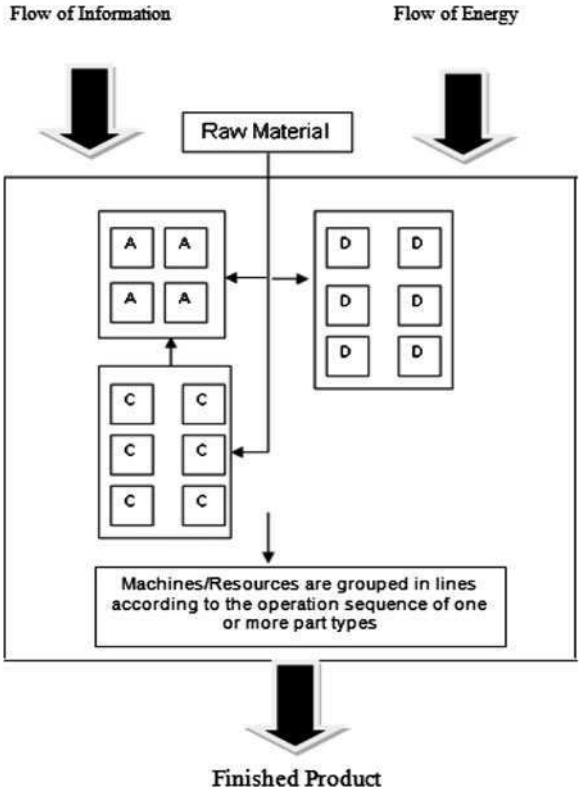


Fig. 2.2 Schematic of a job shop

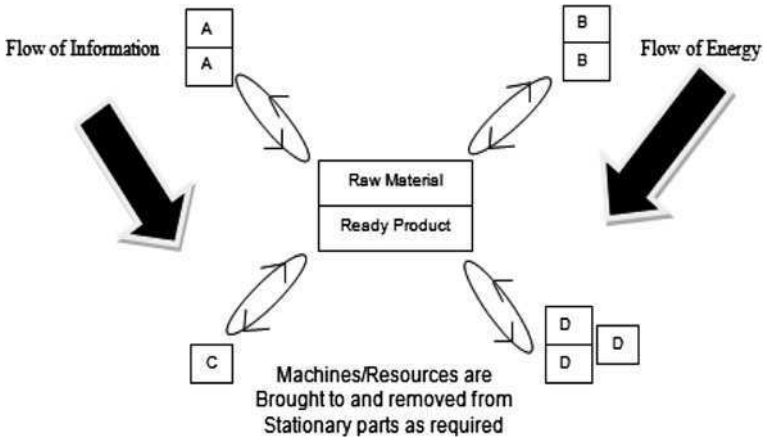


Fig. 2.3 Project shop

2.2.2 Project Shop

In a project shop, a product's position remains fixed during manufacturing because of its size and/or weight. Materials, people, and machines are brought to the product as needed. Facilities organized as project shops can be found in the aircraft and shipbuilding industries and in bridge and building construction. A schematic of project shop is presented in Fig. 2.3 project shop has the lowest lot size production among all type of manufacturing systems.

2.2.3 Cellular Manufacturing

In manufacturing systems organized according to the cellular plan, the equipment or machinery is grouped according to the process combinations that occur in families of parts. Each cell contains machines that can produce a certain family of parts. Figure 2.4 provides arrangement of the equipment in the cellular systems. The cellular manufacturing system is highly automated and flexible. It has low-to-medium lot size production capability.

2.2.4 Flow Line

In flow line, equipments are ordered according to the process sequences of the product to be manufactured. A transfer line consists of a sequence of machines, which are typically dedicated to one particular part or at the most a few very similar parts/products. Only one product is produced at a time (Fig. 2.5).

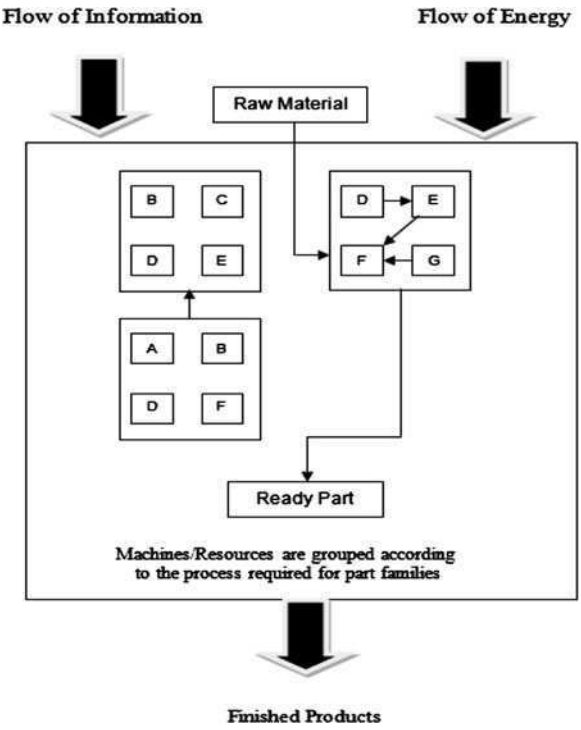


Fig. 2.4 Schematic of a cellular manufacturing system

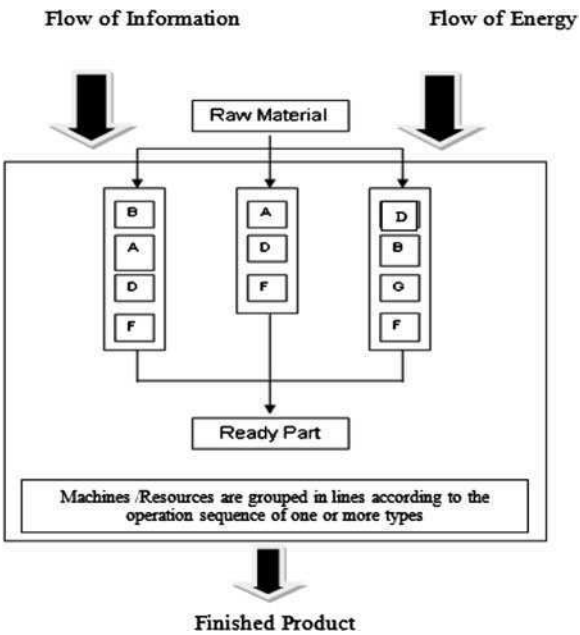


Fig. 2.5 Schematic of a flow line

2.2.5 Continuous Manufacturing System

Continuous manufacturing systems produce liquid, gases, or powders. As in flow line, processes are arrayed in the processing sequence of the products. The continuous system is the least flexible of the types of manufacturing systems (Fig. 2.6).

2.2.6 Flexible Manufacturing System

A flexible manufacturing system (FMS) is manufacturing system that comprises the properties of a job shop and a cellular manufacturing system. It has higher flexibility in manufacturing that can produce diverse product ranges as in the case of a job shop, and, can manufacture larger numbers of group of products as in the case in cellular manufacturing. The FMS arranges CNC machines tools through pick and place technology, and conveyors physically such that a group of products can be produced efficiently in small-to-medium batches (Fig. 2.7).

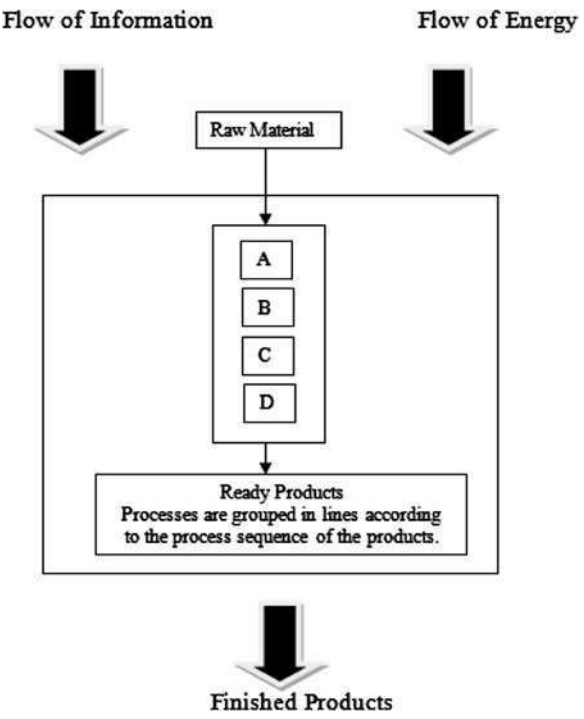


Fig. 2.6 Schematic of a continuous manufacturing system

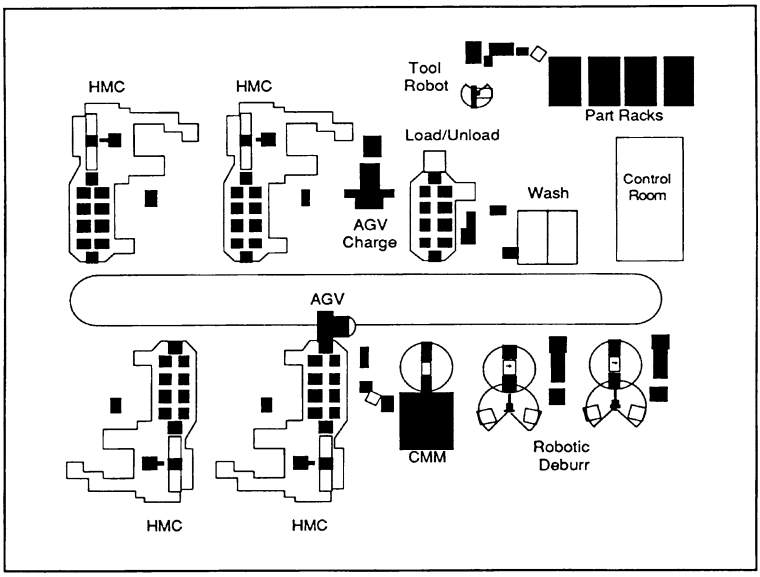


Fig. 2.7 Schematic of a flexible manufacturing system

2.2.7 Assembly System

Another substantial part of a manufacturing company's production facilities is the assembly system. Assembly system can be categorized according to the motion of parts and workplaces. Stationary part systems are usually employed for large assemblies, such as airplanes, which are difficult to be moved around. Moving part systems can be divided into stationary workplace systems, in which parts are brought to stationery workplaces, and moving workplace system, in which the workplaces move along with the parts. Assembly systems with stationery parts tend to have higher floor area requirements. They also tend to have more work at each workplace than moving part systems. Moving part systems are generally more expensive because complicated material handling equipment is required to move parts quickly from workplace to workplace.

In moving part-stationary workplace systems, the assembly operations at each workplace are usually short in duration and are highly repetitive. Moving part-moving workplace systems allow workers to work on each assembly for a longer period of time, making the assembly work less repetitive. The assembly systems are presented in Fig. 2.8. The characteristics of different assembly systems are shown in Table 2.1.

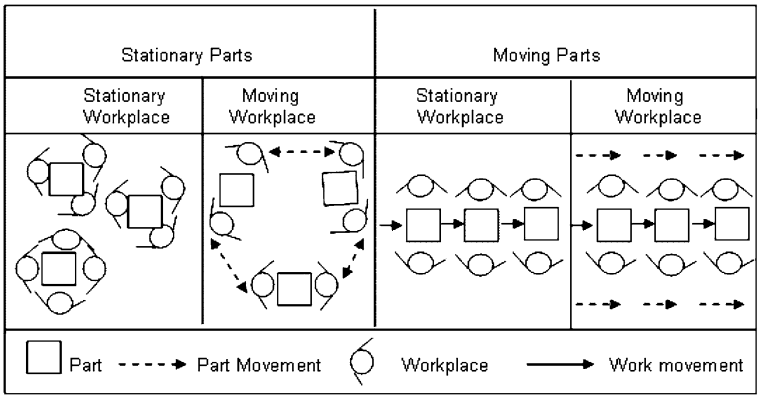


Fig. 2.8 Types of assembly system

Table 2.1 Characteristics of different assembly systems

	Stationary parts		Moving parts	
	Stationary workplace	Moving workplace	Stationary workplace	Moving workplace
Area requirement	High	High	Low	Medium
Work contents at each workplace	High	Medium	Low	Medium
Cost of system	Low	Medium	High	High

In the real-time manufacturing world, these standard manufacturing and assembly system structures often occur in combinations, or with slight changes. The choice of a manufacturing system depends on the design of the parts to be manufactured, the lot sizes of the parts, and market factors such as the required responsiveness. Suitable lot size for each manufacturing system described above is presented in Fig. 2.9.

2.3 Production Planning and Control

Methods and procedures employed in handling materials, parts assemblies, and subassemblies, from their raw or initial stage to the finished product stage in an organized and efficient manner are called production control. It also includes activities such as planning, scheduling, routing dispatching, storage, etc. Production control relies on the experience of the manager to worker level human resourses, and algorithms used for efficient control.

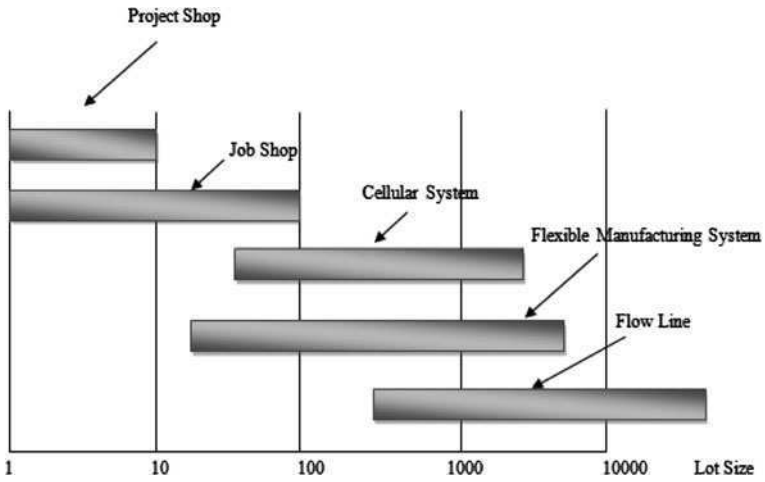


Fig. 2.9 Suitable manufacturing system types as a function of lot size

2.4 Virtual Reality for Manufacturing Systems and Processes

Discrete manufacturing utilizes diverse manufacturing processes and systems for producing wide range of products. It is rare to find similar manufacturing setup for producing same range of products.

Any manufacturing setup implementing Augmented Reality requires that all the production machinery is stored in the form of graphic models, all the decision parameters are stored in forms of variables, and the inference engine is running in a distributed computing environment. The process systems and the inference engine have the interconnectivity among themselves as well as with the vendors and businesses. There should be a mechanism for producing management information system reports so that the manager is capable of taking a simple to an intricate decision. Section 2.1 provides a scope of manufacturing process that may be used in a real Discrete manufacturing system. Section 2.2 details types of manufacturing system that a discrete manufacturing unit may adopt.

2.5 A Survey of the CNC Controller and Their Applications

Computer numerical control controller is used with number of discrete processes. A survey of these controllers is presented to demonstrate the scope of use of AR for CNC-based processes (Table 2.2).

Table 2.2 A survey of the CNC controllers

No.	CNC control manufacturer	Website, country	Control models	Application
1.	ABA Z&B Schleifmaschinen GmbH	http://www.abazandbusa.com/abamaticl.htm, Germany	(a) ABAMATIC 1	Grinding
2.	Actek, Inc.	http://www.actekinc.com, Seattle, WA, USA	FLEXcnc2-3 Axis PScnc-PMTI	Milling and boring machine Engine lathe conversion Abrasive water jet control table Definite purpose router Fabrication shop punch Profiling router floor mill retrofit Precision lather retrofit Fabrication shop punch retrofit Vertical turning center Filament winding machine Automatic overhead cranes Press and punch feeders Portable flam-cutting machine
3.	Acu-rite	http://www.acu-rite.com/view/millpwr-control-systems.aspx, Schaumburg	MILLPWR	Vertical knee mill
4.	Aerotech	http://www.aerotech.com/products/controllers/motion_controllers.html, USA	A3200 ESSEMBLE SOLOIST	Miscellaneous
5.	AjaxCNC	http://www.ajaxcnc.com/, Howard, PA	Mill kit Lathe kit	Mill/lathe simultaneous motion for up to 4-axis machines
6.	Allen-Bradley	http://www.ab.com/	9 SERIES CNC GRINDER	Grinding
7.	AMK	http://en.amk.antriebe.de, Germany	Modular CNC controllers; CNC 900 Compact CNC controllers; CNC 905 and 903,902	High-quality and individual controllers for multi-axle and multi-machine applications with convenient handling The economical and efficient solution for simpler applications for controlling up to four axles

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
8.	Anilam	http://www.anilam.com, North America	3000 COMMANDO	Boring
			3000 M	Milling
			4200 T	Turning
			5000 M	Milling
			6000I	Milling
9.	ArtSoft Mach3	http://www.machsupport.com, Fayette, ME, USA	MACH 3 CNC	Lathes
			Upto 6-AXIS CNC CONTROLLER	Mills Routers Lasers Plasma Engravers Gear cutting
10.	Asia Automation Pvt Ltd	http://www.asiaautomation.com/multi-axis-cnc.htm, India	SINGLE AXIS CNC CONTROLLER	Miscellaneous
			MULTI AXIS CNC CONTROLLER	Miscellaneous
11.	Bevel Cutting	http://www.bevelcutting.com	PC based CNC CONTROLLERS	
			USB CNC UPl-SERIES USB CNC UP2-SERIES USB CNC UP3-SERIES	plasma and oxyfuel metal sheet cutting
			CNC-P00 series	For machine's having Multi head cutting technologies installed like plasma, oxy-fuel, water-jet cutting, plasma marking, bevel cutting with triple torch or one torch bevel rotators
12.	Bosch Rexroth Corporation	http://www.boschrexroth.com/dcc/Vornavigation/Vornavi.cfm?Language=EN&VHist=g97568&PageID=g96072, Hoffman Estates, IL	4 SERVO-AXIS	Turning, milling, drilling
			8 SERVO-AXIS, CONTROLLER BASED 8 SERVO-AXIS, PC BASED 64 SERVO-AXIS	Turning, milling, grinding, drilling, punching nibbling, shape cutting, reforming

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
13.	Brothers	http://www.brother.com, Mizuho-ku, Nagoya	CNC-B00 CONTROL CNC-A00	Tapping centre Tapping centre
14.	Burny	http://www.burny.com/Cleveland, OH, USA	PC BASED CNC CONTROLS Burny XL Burny 10 LCD Plus Burny Phantom Burny Phantom ST NON-PC BASED CNC CONTROLS Burny 2.5 Plus Burny 2.8 Plus Burny 1250 Plus Burny 1400 Plus	Oxyfuel, plasma, laser, water jet, routers, punches, drills, knives, and markers
15.	Bystronic	http://www.bystronic.com, Hauppauge, NY	BY VISION CNC CONTROL	Laserjet cutting, water cutting, bending
16.	CamSoft Corporation	http://www.cnccontrols.com, Lake Elsinore, CA	MS-1500 CNC CONTROL CAMSOFT TOUCH SCREEN WITH SOFTWARE CNC-PROFESSIONAL OR CNC-LITE, CNC-PLUS	Pick and place, robots, welders, grinders, vision, camera systems, wire and sinker EDM, XYZ positioning tables, automatic height control, press brake, shear, bending, circuit board cutting, cut to length profilers, polishers, dispensing-glue, paint, epoxy, precious metals, cut of saws, stone saws stone cavers, drilling machines, assembly automation, CNC mill, CNC router, CNC lathe, CNC laser, CNC plasma, CNC grinder, CNC EDM and CNC punch presses
17.	Centroid	http://www.centroidcnc.com/, Howard, PA	M400, M-39 T400	Milling Lathe

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
18.	Chengdu Great Industrial Co. Ltd	http://www.great-cnc.com/index.html, China	GREAT-150iT/Ita GREAT-66Ti GREAT-150ITJ GREAT-150iM/iMA	Turning Lathe Lathe Milling
19.	Cincinnati Incorporated	http://www.e-ci.com/what_new.php#$, Harrison, OH	CB-II AUTOFORM-AF AUTOFORM-AS FORM MASTER-FM FORM MASTER-FM II	
20.	Cleveland Motion Controls (CMC)	http://www.cncontrols.com, Cleveland, OH, USA	Deals in BURNY MACHINE CONTROLS	See no. 13 above
21.	CNC Automation, Inc.	http://www.cncauto.com, NH, USA	System M3X/M4X CNC Control	Milling
22.	C-TEK Technologies Corp.	http://www.ctek.com.tw/EN/p06.htm, Taiwan	CNC WireCut EDM Controller (X, Y, Z, U, V, 5 axis) ZNC EDM Controller (50A, 75A, 100A)	EDM
23.	DACC	http://www.dacc.com/lynx.htm	DACC PICOPATH CNC CONTROL LYNX ProMotion 2000 ProMotion 2000 NT	For all cutting machines, includes oxy-fuel interface, plasma interface and motorized torch lifter control
24.	DMG	http://www.deckel.com/en,controls,dmg-ergoline-control, Germany	DMG ERGOline® Control with HEIDENHAIN With Fanuc With Siemens	Milling with iTNC 530 Milling with MillPlus iT V600 Turning with Heidenhain Plus iT Turning with Fanuc 3x0i Milling with Fanuc 3x0i Milling with 840D solutionline Turning with 840D solutionline

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
25.	DELECTRON	http://www.delectron.it, Italy	CNC Z32 Z—Star Series/Z32 CNC FZ Series	For milling machines, machining centres, lathes, laser cutting, plasma cutting, special machines
26.	Denford Education and training machines with control and programming software	http://www.denford.com/, UK	COMPACT 1000/COMPACT 1000 PRO ROUTER 2600/ROUTER 2600 PRO VERTICAL ROUTER	CNC routers
			PCB ENGRAVER	Engraving
			VMC 1300	Milling
			MICRO MILL	Milling
			TURN 270	Turning
			MICRO TURN	Turning
			CONTROL SOFTWARE VR CNC MILLING 5 VR CNC TURNING	
27.	Delta Tau Data Systems, Inc.	http://www.deltatau.com/common/products/cnc.asp?node=id600&connectionStr=release, Chatsworth, CA, USA	Advantage 900 Advantage 400	Miscellaneous
28.	Diaform	http//www.pgtechnology.co.uk/cnc_diaform.htm, UK	DIA FORM CNC	Profile dressing for CNC controlled grinding machines for both surface and cylindrical grinders
29.	Direct Motion	http://www.directmotion.com, Ontario, CA	Direct Motion CNC 5.0 (PC-based controller)	Milling/lathe
30.	DYNA Mechtronics	http://www.dynamechtronics.com/product.html, San Jose, CA	DYNA 64-Bit Pentium® CONTROL SYSTEM	2-5 axis PC based control for milling and turning
31.	DynaPath Systems, Inc.	http://www.dynapath.com/page_controls.htm, Livonia, MI	Delta 2000M	Milling
			Delta 2000T	Turning
			Delta 2000P	CNC punch
32.	Dynomotion	http://dynomotion.com/Calabasas, CA	K MOTION CNC (PC based)	2-5 axis CNC controller

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
33.	ECKELMANN	http://www2.eckelmann.de, Wiesbaden, Germany	CNC Classic: E·CNC 55	Electronics/productronics
			PC-based CNC: E-PNC55	
			Embedded CNC E-ENC55	Control of machines for pc board assembly and production
			Embedded Controller ExC55	
			Embedded Controller ExC66	Highly precise axis control during IC conditioning for electronics components
			Embedded Controller ExC66 compact	CNC controllers for laser systems for wafer processing and solar cell structuring
				CNC controllers for soldering systems (we will be pleased to provide you with further information on request)
				Machine tools/production system
				Drilling: drilling machine, pc board drilling machine, laser drilling machine
				Gas cutting: gas cutting machine (autogenous, plasma)
				Turning: lathe, automatic lathe, pipe and processing machine
				Eroding: eroding machine, wire erosion machine, cavity sinking machine
				Folding: folding machine, filter folding machine
				Milling: milling machine, pc board milling machine, wood milling machine, HSC machining
				Joining: joining machine
				Thread rolling: rolling machine, thread rolling machine
				Thread tapping: thread tapping machine

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
				Engraving: engraving machine, pantograph Laser removal: laser removal system Laser cusing: rapid prototyping Laser engraving: laser engraving machine Laser cutting: laser cutting machine Laser welding: laser welding machine Friction welding: friction welding machine Grinding: grinding machine, cylindrical grinding machine, out-of-round grinding machine, flat grinding machine, profile grinding machine Cutting: cutting machine, fabric cutting machine, leather cutting machine, polystyrene cutting machine, plastic cutting machine, foam cutting machine, contour cutting machine, water jet cutting, water jet cutting machine, water jet system Welding: welding machine, welding system, plastic welding machine, ultrasound welding machine Textiles Contour cutting machines CNC sewing machine production and processing of textiles Processing of glass, plastics and wood laser cutting process for flat glass wood and stone cutting

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
34.	ECS (Italian controls)	http://www.ecsitaly.it, Italy	SERIES 800	
			CNC 1801 & CNC 4801	Milling
			CNC 1802 & CNC 4802	Turning
			CNC 1805	Cutting
			SERIES 900	
			CNC 901	Milling
			CNC 902	Turning
			CNC 903	Turning milling
			CNC 904	Sheet bending
			CNC 905	Sheet cutting
			CNC 906	Pipe bending
			CNC 907	Pipe cutting
35.	ESAB	http://products.esabna.com/Cutting/EN/cutting/cutting_mechanized_cutting/mech_cutting_secondary_1180017524/q/display_id.id4367f4bblc3cb9.49585191/path.mechanized_cutting_computer_numerical_controls, USA	VISION PC VISION PC R VISION 51 VISION 52 VISION 55	CNC welding and cutting (inkjet, water jet, laser, plasma) Industrial PC based hardware and Windows based software
36.	Fagor Automation	http://www.fagorautomation.com/, Spain	8070 T CNC	Lathes To control turning centers, vertical lathes, slanted-bed lathes, parallel lathes, dual-turret (TT) lathes, lathes with several turrets and spindles and dual-purpose (mill-lathe) machines
			8055 T CNC	To control high-production turning centers, vertical lathes and parallel lathes
			8035 T CNC	For lathes and turning centers with 2 axes and 1 spindle
			8070 M CNC	Milling

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
			8055 M CNC	To control all types of milling machines and machining centers, both horizontal and vertical
			8035 M CNC	For milling machines and machining centers with 3 axes and 1 spindle
			8070 OL CNC	Other applications
			8055 GP CNC	Grinders, saws, press brakes, machines for wood, marble, stone and glass laser, plasma cutting machines, etc.
			8035 M CNC	For controlling all types of machines (engraving, cutting, etc.) with 2 or 3 axes and 1 spindle
37.	FANUC Ltd	http://www.fanuc.co.jp, Japan	0i Model D 0i Mate Model D	High reliable and high cost-performance CNCs
			30i /31i/32i-MODEL A	AI nano CNC of high speed and high accuracy machining
			16i/18i/21i-MODEL B 160i/180i/210i/MODEL B 160is/180is/210is-MODEL B 18i /180i /180is/-M MODEL B5	Ultra-compact, ultra-thin CNC integrated LCD and network features
			20i MODEL B Power Mate i-MODEL D/H	For manual milling machines and lathes FA network-capable positioning control CNC
			α-T21iFsa/T21iFsb α-T21iFa/T21iFb α- α-T14iFsa/T14iFsb α-T14iFa/T14iFb α-	ROBODRILL: Spindle taper size no. 30 for milling and boring as well as drilling and tapping

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
			S-2000i series 15B/30B/50B/50BP/100B/100BH/150B/250B/300B	ROBOSHOT: electric injection molding machines with advanced control with dedicated servo system
			ROBOCUT α-liD/ROBOCUT α-0iD	AI wire-cut electric discharge machine
			ROBONANO α-0iB	For super nano precise machining in optical electronics/semiconductors, medical and biometric fields
38.	FASTCUT	http://www.fastcutcnc.com/controller.php, Canada	G6 CONTROLLER	CNC plasma cutting oxy/fuel cutting
39.	FIDIA	http://www.fidia.it/, Italia	C CLASS CNC CONTROLLER	
			C0	For milling and machining centres
			C1	
			C10	
			C20	
			F CLASS CNC CONTROLLER	Milling machines, machining centres and boring machines
			F0	
40.	Flash Cut CNC	http://www.flashcutcnc.com/CNC-Controls.php, San Carlos, CA	FLASHCUT CNC CONTROL	Milling and lathes
41.	GSK CNC	http://gskcnc.com/cnc_controller.php, Thailand	218M	Milling/drilling
			983M	
			980 M	
			928 MA	
			980 TD	Turning
			928 TC	

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
42.	HAAS	http://www.haascnc.com/lang/, CA, USA	Haas CNC Lathe Control	Lathe
			Haas Mill (VMC/HMC/5AXIS) Control	Milling
43.	HEIDENHAIN	www.heidenhain.com, Schaumburg, IL	iTNC 530/TNC 620/TNC 320/TNC 124 MANUALplus 620	Milling turning
44.	Hurco	http://www.hurco.com, Indianapolis	Hurco Winmax® Lathe Control	Turning
			Hurco Winmax® Mill Control	Milling
45.	Hypertherm, Inc.	http://www.hyperthern.com/en/Products_and_Services/Automation/Computer_Numeric_Controls/index.jsp, Hanover, NH	EDGE PRO Micro EDGE EDGE Ti Mariner EDGE II VOYAGER III	Plasma cutting
46.	Integrated Industrial Technologies, Inc.	http://www.isquaredt.com/products/?g=1, Pittsburgh, PA, USA	PC CONTROL	Grinding, milling, drilling, boring, grinding, and turning
47.	ISEL	http://www.iselautomation.de/products/product.php?lang=en&ID=p66, Germany	CNC Controller C 142-4	For any 3-axes cutting machines
48.	Lakshmi Electro Controls & Automation	http://www.leca.in, Karnataka, India	Advantage 400	Universal motion controller for up to 5 axes machines
49.	LNC	http://www.lnc.com.tw, Taiwan	LNC-MS000 six independent path, and control axes can be up to 32 axes	Lathe/milling grinding/engraving machines PCB drilling machine, glass cutting machine, wafer cutting machine, laser cutting machine, printing machines, and another relevant industrial machines

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
50.	Larken Automation	http://www.larkencnc.com, Ottawa, Canada	3 axis micro-step stepper motor controllers	Custom xyz tables, plasma tables, milling machine retrofits
51.	Linatrol Profile & Shape Cutting Solutions	http://www.linatrol.com/products_cnc_picopath.asp, Burlington, Ontario, Canada	PICOPATH CNC Control LYNX CNC SYSTEM Profiler™ CNC Control Profiler Plus CNC Controller Infinity CNC	Plasma and oxy-fuel cutting process
52.	Lubi Electronics	http://www.lubielectronics.com/new_site/products/drives_&_motion/cnc_controller/cnc_controller.html, Taiwan		CNC lathe machine Milling machine Bending machine Grinding machine Engraving machine Water jet cutting machine Plasma arc cutting machine Induction hardening machine PCD cutting machine
53.	LUMONICS	http://www.lumonics.com/, Nepean, Ontario, Canada	GadgetMaster Motor Control	3 axes motion control retrofit kit
54.	Mach Motion	http://www.machmotion.com/, Newburg, MO, USA	MILLING MACHINE CONTROLS	
			X15-250	CNC mill control—standard
			X15-250-U	CNC mill control—ultimate
			X15-250-01	CNC mil stepper kit
			X15-250-06	Mill kit—TECO
			X15-250-5	CNC mill control analog
			X15-350-04	Mill kit—Mitsubishi
			X15-350-04ES	Mill kit – Mitsubishi with drive enclosure

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
			LATHE CONTROLS	
			X15-250L	CNC lathe control
			X15-250-5	CNC lathe control with analog drive controller
				Lathe kit—Mitsubishi
			X15-250-04L	
			Plasma and Oxy Fuel CNC Control	
			X15-250-06	CNC plasma or oxy fuel servo kit—TECO
				CNC plasma or oxy fuel servo kit—Mitsubishi
			X15-350-04	
			CNC Router Controls	
			X15-350R	CNC router control
			X15-250-01	CNC router stepper kit
			X15-350-06	CNC router kit—TECO
			X15-250-5	CNC router control with analog drive controller
			X15-350-04	CNC router kit—Mitsubishi
			CNC Grinder Controls	
			X15-350	CNC grinder control
			X15-250-5	CNC grinder control with analog drive controller
			X15-350-04G	CNC grinder kit
			CNC tube bender controls	
			X15-250-04T	CNC tube bender control/retrofit
			CNC Controls for Benchtop Machines	

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
			X15-250B	CNC control for Benchtop machines
			X15-250-01	CNC stepper kit
			X15-255	Mach3 CNC monitor enclosure
			X15-400s	Compact CNC control W/O switch panel
55.	Mazak corporation	http://www.mazak.eu, Japan	Mazatrol Matrix	Turning Milling Multi-tasking machines Laser processing machines
56.	MDSI	http://www.mdsi2.com, Dexter, MI	OpenCNC	Drill Grinder Hob Lathe Mill Ream Router Specialty machines
57.	Messer Cutting Systems, Inc.	http://www.mg-system-welding.com/Products/CNCcontrols/tabid/63/Default.aspx, Menomonee Falls, WI, USA	Global Control Plus System Metalmaster Plus MPC2000 TMC4500ST Titan II Global Control S Systems EdgeMax EdgeMate EdgeMaster	Plasma and oxyfuel

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
58.	Microkinetics Corporation	http://www.microkinetics.com/, Kennesaw, GA, USA	MN 400 MOTION CONTROL WITH DRIVE RACK With control software Mill-Master Pro	Milling
			MN 400 MOTION CONTROL WITH DRIVE RACK With control software Turn-Master Pro	Turning
59.	Mikini Mechatronics	http://www.mikinimech.com/products.htm, Watsonville, CA, USA	MIKINI V2.0 CNC CONTROLLER	Featuring 4 axis manual and CNC host operation. Four axis-DRO with feed rate calculation and spindle load monitoring
60.	Milltronics, Inc.	http://www.milltronics.net/index.aspx, Waconia, MN	Milltronics CNC control	Lathe/milling
			7200 SERIES 8000 Series	high speed milling
61.	MS-Tech Corporation	http://www.ms-tech.com/, Corona, CA, USA	MSTC-800	Machining centers, turning centers, grinders punches,
62.	MSHAK JSC	http://www.mshak.am/, Armenia	CNC MSH PC-104	Lathe, milling and other machines with control of up to 6 axes
			CNC MSH TURBO-U MSH TURBO -M	Complex machine tools with up to 16 controllable axes
			MSH GEO Brick	For multi-axis machine tools, machining centres and complex multi-machine tools units with controllable axe up to 32
				High-performance hard-ware control unit with integrated in control units digital drives for 4, 6, 8 axes

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
63.	MultiCam, Inc.	http://www.multicam.com/eng/, Dallas, TX, USA	EZ Control	CNC routers CNC plasma CNC water jet CNC tooling CNC laser
64.	MORISEIKI	http://www.moriseiki.com/english/products/app/index.html, Japan	MAPPS IV	
			MAPPS III for CNC Lathe	Turning
			MAPPS III for Integrated Mill Turn Centers	Milling turning
			MAPPS III for Machining Center	Milling
65.	Numatix CNC Controls, Inc.	http://www.numatix.com/, Farmington Hills, MI, USA	Numatix CNC Control	
			MAP-3	Specifically tailored for any 3-axis CNC machine application
			MAP-3+2	This retrofit package is specifically tailored to convert any 3 axis CNC machine to a fully capable 5-axis CNC machine by utilizing a 5-axis spindle attachment
			MAP-4	Specifically tailored to any 4-axis CNC machine application
			MAP-5	
			MAP-FADAL	Specifically tailored to any 5-axis CNC machine application This retrofit package is specifically tailored to Fadal CNC machines
66.	Num. Corporation	http:www.num-usa.com/, Battenhusstrasse, Teufen	CNC control with digital bus	Tool grinding
			Flexium CNC System Axium Power CNC System	Inline and rotary transfer and multi-spindle machines Any 5 or more axes machines

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
				Woodworking gear manufacturing machines
			CNC control with analog interface	
			Num Power 1020–1040–1060–1080: four ultra compact CN	Turning, milling, grinding, or wood, or glass, or stone or control processes such as palletising, handling or cutting
67.	NUTEC Components, Inc	http://www.nutecl.com/index_files/Page506.htm, Deer Park, NY	MICROMATIC-9 MICROMATIC-3	For virtually any type of motion control application, including Pick and Place applications, CNCs, PLC programing and even robotics with kinematics
68.	OmniTurn CNC	http://www.omniturn.com/bin/Controls.htm, Port Orford, OR	OmniTurn CNC Control	Turning
69.	Omtronix	http://www.omtronix.com/, Mississauga	TurboCut CNC	High speed milling Motion Control software and hardware for multiple axis control
70.	OKUMA	http://www.okuma.com/products/mechatronics/, Niwa-gun, Aichi, Japan	THINC PC BASED CNC CONTROLLER USING MICROSOFT WINDOWS PLATFORM	Different machine tools
71.	OSAI	http://www.osai.it/indexb.html, Italy	MC Models: 10/110 10/110 LIGHT OS-WIRE 10/510 Light 10/510i	Wood working Pantographs/work centres Polishing/painting Profiling Mortising Piercing/perforating/punching pantograph

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
			T (Lathes) Models	Plastic working
			3 Axes	Moulding
			3–6 Axes WinLink	Modelling
			GP MODELS	Glass working
			10/510 OpLink	
			10/510 WinLink	Cutting tables
			10/510 Control Unit	Bevelling machines
				Perforating
				Etching machines
				Grinders
				2 Axes Machines
			OS-Wire	Stone working
				Machining centres
				Grinding machines
				Sawing machines
				Lathes
				Laser scanning
				Bridge mills
				Metal working
				Lathes
				Mills
				Machining centres
				Special applications
				Turbine cutters
				Water jet
				Laser
				Transfer machines
				Spring bending
				Grinding

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
				Automation system Palletising Robotics Automated warehousing Wire and pipe bending
72.	Power Automation	http://www.powerautomation.com/, Pleidelsheim, Germany	PC BASED CNC CONTROLS	Routing
			PA 8000 e2 PA 8000 LW PA 8000 EL	Engraving Milling Turning Grinding Laser cutting Water jet cutting Plasma cutting Oxy-fuel cutting Multi- channel/multi-axes applications High speed applications
73.	PC Controls, LLC	http://pccontrols.net/, Arlington, TX	10 SERIES MC CONTROLS 10 SERIES LATHE CONTROLS	Machining centers, turning centers and lathes
74.	Siemens	http://www.sea.siemens.com/us/Industry_Solution/Machine-tools/Pages/Machine-Tools-Home.aspx, Munich, Germany	SINUMERIK 802C SINUMERIK 802S SINUMERIK 802D SINUMERIK 802D solution line	CNC for basic machines
			SINUMERIK 810D SINUMERIK 828D (NEW) ACRAMATIC CNC	Digital CNC for dynamic machining
			SINUMERIK 840D SINUMERIK 840D solution line	CNC for high-speed machining

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
75.	Slicer Controls, LLC	http://www.slicercontrols.com/, Minneapolis, MN	SLICER	Turret Punch Press retrofits Plasma retrofits, hydraulic clutch retrofits, and laser retrofits
76.	SOFT LOG	http://www.softlogcnc.com/machine-tool-automation-products.html, Pune, Maharashtra, India	Model—2T/2TG	2 axes CNC control—for lathe applications
			Model—3M/3MG	3 axes CNC control—for milling applications
77.	Taurin Group USA	http://www.tauringroupusa.com/CNC_controls.htm, Ontario, CA	CNC-C CNC-8 CNC-I	CNC bending
78.	UERRO	http://www.uerro.com, Taiwan	EPK6 SERIES PC Based CNC Controller 2 Axes EPCIO SERIES PC Based CNC Controller 3 Axes M2000 system PC Based CNC Controller 4 Axes	Tool machine control, semi-conductor position controls industrial lathe, milling, grinding machine and any 2/3/4 axes motion position control system
79.	UNIDEX	http://www.unidexmachinery.com/, IL, USA	NA	Spring and wire forming
80.	Vega CNC	http://www.isscnc.com/, Troy, MI	VEGA MX-2 CNC Control Vega SG CNC	CNC s for machine tool and factory automation CNC s for grinding applications
81	Velocitech CNC LLC	http://www.velocitechcnc.com/, Florence, KY, USA	Velocitechcnc	Retrofit of different machine tools
82.	VERTEX	http://www.vertex-monaco.com/vertexcnc, Monaco	COMPACT version VSC 1020–1030	Lathe
			UPTO 4 axes CNC Control	Milling

(continued)

Table 2.2 (continued)

No.	CNC control manufacturer	Website, country	Control models	Application
83.	Weihong CNC System	http://www.naiky.com/Product.asp, Shanghai, China	NK-300 CNC Integrated Machine System	Lathe
			NK-200 CNC Integrated Machine System	Milling
			NK-100 CNC Integrated Machine System	
			NC-100 CNC system	
84.	Yaskawa	http://www.yaskawa.com/site/Products.nsf/products/Multi-Axis%20Motion%20Controllers-MP2000iec_series.html, IL, WI, USA	B Series CNC Controller	Variety of machine tools
			G Series CNC Controller	
			J100 Series CNC Controller	
			X1 Series CNC Controller	
			X2 Series CNC Controller	
			X3 Series CNC Controller	

Bibliography

1. Al-Ahmari A et al (2009) Design of cellular manufacturing systems with labor and tools consideration 2009. Comput Ind Eng Conference (CIE 2009) 678–683
2. Anjard SR, Ronald P (1995) Computer integrated manufacturing: a dream becoming a reality. Ind Manage Data Syst 95(1):3–4
3. Calin C et al (2009) Formalisms for modeling the discrete manufacturing systems. In: IEEE 2009 international conference on mechatronics, ICM 2009
4. DeGarmo EP, Black JT, Kohser RA (2002) Materials and processes in manufacturing, 9th edn. Wiley, New York
5. http://www.abazandbusa.com/abamatic1.htm
6. http://www.actekinc.com, Seattle, Washington
7. http://www.acu-rite.com/view/millpwr-control-systems.aspx
8. http://www.aerotech.com/products/controllers/motion_controllers.html
9. http://www.ajaxcnc.com/, Howard
10. http://www.ab.com/
11. http://en.amk-antriebe.de
12. http://www.anilam.com
13. http://www.machsupport.com
14. http://www.asiaautomation.com/multi-axis-cnc.htm
15. http://www.bevelcutting.com
16. http://www.boschrexroth.com/dcc/Vornavigation/Vornavi.cfm?Language=EN&VHist=g97568&PageID=g96072
17. http://www.brother.com
18. http://www.burny.com
19. http://www.bystronic.com
20. http://www.cnccontrols.com
21. http://www.centroidcnc.com
22. http://www.great-cnc.com/index.html
23. http://www.e-ci.com/what_new.php#8
24. http://www.cmccontrols.com
25. http://www.cncauto.com
26. http://www.ctek.com.tw/EN/p06.htm
27. http://www.dacc.com/lynx.htm
28. http://www.deckel.com/en,controls,dmg-ergoline-control
29. http://www.delectron.it
30. http://www.denford.com
31. http://www.deltatau.com/common/products/cnc.asp?node=id600&connectionStr=release
32. http://www.pgtechnology.co.uk/cnc_diaform.htm
33. http://www.directmotion.com
34. http://www.dynamechtronics.com/product.html
35. http://www.dynapath.com/page_controls.htm
36. http://dynomotion.com
37. http://www2.eckelmann.de
38. http://www.ecsitaly.it
39. http://products.esabna.com/Cutting/EN/cutting/cutting_mechanized_cutting/mech_cutting_secondary_1180017524/q/display_id.id4367f4bb1c3cb9.49585191/path.mechanized_cutting_computer_numerical_controls
40. http://www.fagorautomation.com
41. http://www.fanuc.co.jp
42. http://www.fastcutcnc.com/controller.php
43. http://www.fidia.it

44. http://www.flashcutcnc.com/CNC-Controls.php
45. http://gskcnc.com/cnc_controller.php
46. http://www.haascnc.com/lang
47. http://www.heidenhain.com
48. http://www.hurco.com
49. http://www.hypertherm.com/en/Products_and_Services/Automation/Computer_Numeric_Controls/index.jsp
50. http://www.isquaredt.com/products/?g=1
51. http://www.iselautomation.de/products/product.php?lang=en&ID=p66
52. http://www.leca.in
53. http://www.lnc.com.tw
54. http://www.larkencnc.com
55. http://www.linatrol.com/products_cnc_picopath.asp
56. http://www.lubielectronics.com/new_site/products/drives_&_motion/cnc_controller/cnc_controller.html
57. http://www.lumonics.com
58. http://www.machmotion.com
59. http://www.mazak.eu
60. http://www.mdsi2.com
61. http://www.mg-systems-welding.com/Products/CNCControls/tabid/63/Default.aspx; http://www.microkinetics.com
62. http://www.mikinimech.com/products.htm
63. http://www.milltronics.net/index.aspx
64. http://www.ms-tech.com
65. http://www.mshak.am
66. http://www.multicam.com/eng/
67. http://www.moriseiki.com/english/products/app/index.html
68. http://www.numatix.com
69. http://www.num-usa.com
70. http://www.nutec1.com/index_files/Page506.htm
71. http://www.omniturn.com/bin/Controls.htm
72. http://www.omtronix.com
73. http://www.okuma.com/products/mechatronics/
74. http://www.osai.it/indexb.html
75. http://www.powerautomation.com/
76. http://pccontrols.net
77. http://www.sea.siemens.com/us/Industry_Solutions/Machine-tools/Pages/Machine-Tools-Home.aspx
78. http://www.slicercontrols.com
79. http://www.softlogcnc.com/machine-tool-automation-products.html
80. http://www.tauringroupusa.com/CNC_Controls.htm
81. http://www.uerro.com
82. http://www.unidexmachinery.com
83. http://www.isscnc.com
84. http://www.velocitechcnc.com/
85. http://www.vertex-monaco.com/
86. http://naiky.com/Product.asp
87. http://www.yaskawa.com/site/Products.nsf/products/Multi-Axis%20Motion%20Controllers ~ MP2000iec_Series.html
88. Luce F et al (2008) 3D graphic simulation of flexible manufacturing systems with day dream daemon and 3D create. In: IEEE international conference on industrial informatics (INDIN), pp 1401–1406

89. Manesh HF et al (2007) Applications of virtual reality in computer integrated manufacturing systems. Proceedings of the ASME international design engineering technical conferences and commuter and information in engineering conference, DETC2007, pp 403–410
90. Milell PG (2001) Fundamentals of modern manufacturing: materials, processes, and systems, 2nd edn. Wiley, New York
91. Wu B et al (2000) The design of business processes within manufacturing systems management. Int J Prod Res 38:4097–4111

Chapter 3
Automation and Control in Manufacturing

3.1 Modern Control Systems

Control systems and engineering have a history which is centuries old. The most up to date development in control systems from industrial perspective took place during and after industrial revolution that ranged from application of control theory to James Watt's steam engine to space shuttle to automated manufacturing systems. The basic classification of control system encompasses open and close loop systems. The implementation of the control systems may be based on sensors and transducers, actuators and data communication networks. These control system may be configured as a supervisory control system in all possible application. The other major property of the control systems is that they may either be based on digital electronics, analog circuits or hybrid digital analog circuits.

3.2 Mathematical Models for the Control System

Two common methods are used for solving differential equation based mathematical modeling of control systems. They are Laplace transforms and Z-transforms. Relation between the input and the output of linear time invariant system is logged through a transfer function. Apart from the representation of a control system using a mathematical equation, the function of the control system components can also be represented graphically through block diagrams.

3.3 Control Methodologies for Discrete Manufacturing Systems

Control methodologies implemented at discrete manufacturing processes span over varying technologies. It includes Computer Numerical Control (CNC),

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_3,

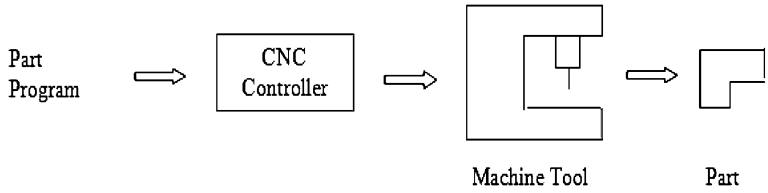


Fig. 3.1 Computer Numerical Control manufacturing sequence

Programmable industrial manipulators, Programmable Logic Controllers (PLC), Embedded Systems, Mechatronics based Controls and Supervisory Control and Data Acquisition (SCADA) systems.

3.3.1 Computer Numerical Control

CNC is widely used as a applied control system at discrete manufacturing artifacts. The general principle of operation uses a controller, essentially a digital computer designed to sustain industrial environment characterized by vibration, shock, temperature, and humidity. Refer to Fig. 3.1.

A part program, normally conforming to an international standard such as EIA RS274D, acts as input to the CNC machine tool. The instructions contained in part program are converted into digital signals and are fed to the actuators at the machine tool. Light machine tools are based on open loop control systems while the heavier systems commonly utilized complete closed loop system taking feedback from a group of sensors and transducers.

CNC machine tools are incorporated in to small and medium batch size discrete manufacturing system having diversity of product design. These systems have the provision of automated information transfer to information cloud at a manufacturing system thus allowing manipulation of micro level decision parameters for optimum operation of the shop floor. Application of CNC and Programmable logic Control is vast and well standardized. CNC has main applications in metal removal processes.

3.3.2 Programmed Control of Industrial Manipulators, Gantries and Conveyors

Discrete manufacturing operation commonly incorporates pick and place operation. This operation utilizes number of mechanisms to perform pick and place operation. Robots, Conveyors, and Cranes are the common mechanical artifact utilized for the purpose in a given manufacturing situation.

Pick and place technology operates in a single axis as in the case of conveyor to multiple axes as in the case of industrial robots. Accordingly the control system

used at the Pick and place technology vary from a pendent to programmable controller. Conveyor and Cranes have simpler operations and may be controlled using fewer commands. Robots use international standard to program multiple axes for performing pick and place operation as well as for measurement, welding and painting. In the case of robot, conveyor, and cranes an inference engine based on software compiler or software interpreter can be used.

3.3.3 Programmable Logic Controllers

PLC are commonly used in discrete and continues manufacturing systems. It is utilized in motion control, control of thermodynamical properties and control of properties related to fluid mechanics. Modern PLC are based on digital micro processors, and are enclosed in a box providing efficient operation during severe industrial conditions. Standards are commonly used to operate the PLC. The most common graphical user interface is used to operate PLC which is the ladder logic. IEC 61131-3 among other functions defines ladder logic in detail and provides programming methodology using this technique.

PLC are most suited to control operations in process industry. However, control of discrete manufacturing processes such as thermal treatment of material and melting furnaces commonly utilizes this technique.

3.3.4 Embedded Systems

Contrary to a personnel computer system, the embedded system are computer systems those are design to control one or few dedicated tasks in real time. The embedded system architecture may contain single or multiple processors and operate without peripherals except devices such as micro controller. The main element of an embedded system is a Real Time Operating Systems (RTOS). RTOS is multitasking operating system designed only to support tasks controlled by an embedded system.

Embedded system are commonly deployed at industrial manipulators, machine tool controls, automobile, aircraft, space crafts, domestic appliances, and research equipment. All these applications are time critical.

3.3.5 Mechatronics Based Application

Mechatronics is a subject area that makes use of mechanics, electronics computing, and control engineering. Mechatronics is applied widely in developing equipment requiring features those may not be standardized. CNC, PLC, industrial

manipulator, embedded system; and, SCADA system are all the established areas those normally implement standards and also fall in the category of mechatronics. However, many other applications from bioengineering, biomedical engineering, nuclear engineering, manufacturing, and other are some of the fields that uses non-standard application for a particular usage.

Applications for open or closed loop systems are used; however, application is not standardized or not commonly used. The Mechatronics provides a methodology to solve such a problem for one-off to small numbers of products.

3.3.6 Supervisory Control and Data Acquisition System

SCADA system are close loop control system used to detect, monitor and alter the state of an industrial and other applications. These industrial application may be control of events in a refinery, a potable water distribution system, set of traffic lights, a bank teller network, or condition monitoring of residential estate.

The interference engine at a SCADA systems are Remote Terminal Units (RTUs) that receives data from sensors, process the input according to set rules, and commands the actuators to achieve desired results. SCADA works in real time similar to an embedded system. The major difference between the two is speed and physical area in which they operate.

SCADA allows human intervention and this override function is provided at all critical junctions. SCADA comprise computer hardware, communication network, array of sensors and transducers, various kinds of actuators; and, computer software providing graphical user interface for the user and controller of the SCADA system.

Bibliography

1. Alsafi Y (2008) A deployment of an ontology-based reconfiguration agent for intelligent mechatronic systems. In: IEEE international symposium on industrial electronics, pp 1780–1785
2. Beaudoin J (1989) Simulation system for the control of manufacturing lines. In: Winter simulation conference proceedings, pp 579–583
3. Behalek M (2010) Usage of embedded process functional language as a modeling tool for embedded systems development. In: ISMS, 1st International conference on intelligent systems, modeling and simulation, pp 116–121
4. Bruccoleri M (2005) UML design and AWL programming for reconfigurable control software development of a robotic manipulator. In: IEEE symposium on emerging technologies and factory automation, pp 331–338
5. Cardenas C (2004) Supervisory control of a manufacturing cell; Image processing, biomedicine, multimedia, financial engineering and manufacturing In: Proceedings of the 6th biannual world automation congress, pp 449–454
6. Chung CA (2004) Design and development of an interactive CNC pre-operation training simulator. In: IIE annual conference and exhibition, pp 255–260

7. Crane P (1991) PLC brand standardization, the benefits. Process and Control Engineering, 44(1):22–23
8. Dykstra W (1987) System design specification for manufacturing process PLC controller application. 3rd IEEE Annual programmable control conference and exhibition, hamilton ontario, Canada
9. Fota A (2009) The method of optimization for control of flexible manufacturing systems. In: Proceedings of the 1st international conference on manufacturing engineering, quality and production systems, MEQAPS, pp 358–364
10. Garg DP (2002) Real-time open-platform-based control of cooperating industrial robotic manipulators In: Proceedings of the IEEE international symposium on intelligent control, pp 428–433
11. Ghunjie Z (2009) Development of a PLC virtual machine orienting IEC 61131-3 standard. International conference on measuring technology and mechatronics automation, ICMTMA, pp 374–379
12. Guerra-Zubiaga DA (2010) Mechatronics design methodology applied at manufacturing companies. Int J Manuf Technol Manag 19(3–4):191–210
13. Henry S (2004) Real time reconfiguration of manufacturing systems. Conference proceedings—IEEE international conference on systems, man and cybernetics, pp 4266–4271
14. Hibino Y, Maruno T (2001) Recent progress on large-scale PLC technologies with advanced functions. NTT Rev 13(5):4–9
15. Horvath L (2006) Virtual space for intelligent engineering in mechatronics. IEEE International Conference on Mechatronics, ICM, pp 248–253
16. Huang S (2009) Mechatronics and control of a long-range nanometer positioning servomechanism. Mechatronics 19(1):14–28
17. http://en.wikipedia.org/wiki/CNC
18. http://en.wikipedia.org/wiki/Programmable_logic_controller
19. http://en.wikipedia.org/wiki/Embedded_system
20. http://en.wikipedia.org/wiki/Mechatronics
21. http://en.wikipedia.org/wiki/SCADA
22. Johansson H (2004) A system for information management in simulation of manufacturing processes. Adv Eng Software 35(10–11):725–733
23. Kao YK (2005) Development of a virtual controller integrating virtual and physical CNC. Mater Sci Forum 505–507:631–636
24. Labib AW (2005) Intelligent real time control of disturbances in manufacturing systems. J Manuf Technol Manag 16(8):864–889
25. Lambert JM (2004) 3D path planning and real-time simulation for robot manipulators with applications to aerospace manufacturing. In: Proceedings of the ASME design engineering technical conference, pp 1139–1146
26. Landryova L (2006) Process knowledge generation and knowledge management to support product quality in the process industry by supervisory control and data acquisition (SCADA) open systems. Prod Plan Control 17(2):94–98
27. Lanoz M (2005) Progress in the PLC development during the years 2003-2004. In: IEEE 6th international symposium on electromagnetic compatibility and electromagnetic ecology, pp 4–9
28. Li B (2006) Research and design of embedded reconfigurable mechatronics system based on field bus. In: Proceedings of the 2nd IEEE/ASME international conference on mechatronic and embedded systems and application
29. Liu Z (2000) The study and realization of SCADA system in manufacturing enterprises. In: Proceedings of the world congress on intelligent control and automation (WCICA), pp 3688–3692
30. Luo Y (2009) CNC grid features modeling employing complex constraints characteristics. In: Proceedings of the international conference on computational intelligence and software engineering

31. Manjunath TC (2005) Design and implementation of a SCADA system using programmable logic controllers. In: Proceedings of the SICE annual conference, pp 1471–1476
32. Park SC (2008) A PLC programming environment based on a virtual plant. Int J Adv Manuf Technol 39(11–12):1262–1270
33. Park HT (2010) Plant model generation for PLC simulation. Int J Prod Res 48(5):1517–1529
34. Pereira CE (2007) Distributed real-time embedded systems: recent advances, future trends and their impact on manufacturing plant control. Annu Rev Controls 31(1):81–92
35. Perk CM (2006) Development of virtual simulator for visual validation of PLC program. CIMCA: International conference on computational Intelligence for modeling, control and automation
36. Rizwan SM (2007) Modelling and optimization of a single-unit PLC system. Int J Model Simul 27(4):361–368
37. Song Z (2010) A model integrated development of embedded software for manufacturing equipment control. Lecture notes in Electrical Engineering, pp 55–61
38. Thapa D (2008) Architecture for modeling, simulation, and execution of plc based manufacturing system. In: Proceedings—winter simulation conference, pp 1794–1801
39. Thramboulidis K (2005) Model-integrated mechatronics—toward a new paradigm in the development of manufacturing systems In: IEEE transactions on industrial informatics, pp 54–61
40. Unger K (2001) The new landscape of manufacturing applications. In: ISA TECH/EXPO technology update conference proceedings, pp 391–400
41. Wang L (2004) Distributed supervisory control and data acquisition (SCADA) software: Issues and state of the art. Autotestcon (Proceedings), pp 332–337
42. Wei R (1998) Modeling and reuse of object-oriented CNC software. Gaojishu Tongxin/High Technology Letters, pp 30–34
43. Whitman SL (2001) Establishing compliance of batch SCADA systems with FDA CGMP 21 CFR Part 11. Pharm Technol, 14–18 + 25
44. Xing B (2008) The application of mechatronic design approach in a reconfigurable manufacturing environment. In: 15th international conference on mechatronics and machine vision in practice, pp 138–143
45. Xu Y (2008) Design of reconfigurable CNC system based on embedded technology. In: Proceedings of the world congress on intelligent control and automation (WCICA), pp 6061–6064
46. Zhang X (2008) Ethernet-based real-time communication and time synchronization of CNC system. Jisuanji Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS, 14(6):1148–1154

Chapter 4
Augmented Reality for Sensors, Transducers and Actuators

4.1 Introduction

Sensors transducers and actuators are the essential elements of a close loop control system. The types of control equipment implementation such as CNC, PLC, Embedded System, Mechatronics, and SCADA Controller commonly utilize basic control theory. Following section provide details of sensors, transducers, and actuators.

4.2 Sensors and Transducers Types and Usage

Sensor is a device that when exposed to a physical phenomenon (temperature, displacement force, etc.) produces a proportional output signal (electrical, mechanical, magnetic, etc.). A transducer is a device that converts one form of energy to another form of energy. Table 4.1[1] represents various types of sensors that are classified by their measurement objectives.

4.3 Actuators Types and Usage

Actuators produce a change in the physical system by generating force, motion, heat, flow, etc. Normally, the actuators are used in conjunction with a power supply and a coupling mechanism. A list of various types of actuators is provided in Table 4.2.[2]

[1] Extracted with permission from 'The Mechatronics handbook'; Bishop R.H.; Instrumentation, Systems and Automation Society; CRC Press, USA.

[2] Extracted with permission from 'The Mechatronics handbook'; Bishop R.H.; Instrumentation, Systems and Automation Society; CRC Press, USA.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_4, © Springer-Verlag London Limited 2011

Table 4.1 Types of sensors for various measurement objectives

Sensor	Features
Linear/rotational sensor	
Liner/rotational variable differential transducer (LVDT/RVDT) Optical encoder	High resolution with wide range capability. Very stable in static and quasi-static applications Simple, reliable, and low-cost solution. Good for both absolute and incremental measurements
Electrical tachometer Hall effect sensor capacitive transducer	Resolution depends on type such as generator or magnetic pickups. High accuracy over a small to medium range. Very high resolution with high sensitivity. Low power requirements
Strain gauge elements	Very high accuracy in small ranges. Provides high resolution at low noise levels.
Interferometer	Laser systems provide extremely high resolution in large ranges. Very reliable and expensive. Output is sinusoidal
Magnetic pickup Gyroscope Inductosyn	Very high resolution over small ranges
Acceleration sensors	
Seismic and Piezoelectric accelerometers	Good for measuring frequencies up to 40% of its natural frequency. High sensitivity, compact, and rugged. Very high natural frequency (100 kHz typical)
Force, torque, and pressure sensor	
Strain gauge, dynamometers/load cells, piezoelectric load cells, tactile sensor, and ultrasonic stress sensor	Good for both static and dynamic measurements. They are also available as micro-and nanosensors. Good for high precision dynamic force measurements. Compact, has wide dynamic range. Good for small force measurements.
Flow sensors	
Pitot tube, orifice plate, flow nozzle, and venture tubes	Widely used as a flow rate sensor to determine speed in aircrafts. Least expensive with limited range. Accurate on wide range of flow. More complex and expensive
Rotameter	Good for upstream flow measurements. Used in conjunction with variable inductance sensor
Ultrasonic type	Good for very high flow rates. Can be used for upstream and downstream flow measurements
Turbine flow meter	Not suited for fluids containing abrasive particles. Relationship between flow rate and angular velocity is linear
Electromagnetic flow meter	Least intrusive as it is noncontact type. Can be used with fluids that are corrosive, contaminated, etc., the fluid has to be electrically conductive
Temperature sensors	
Thermocouples	This is the cheapest and the most versatile sensor. Applicable over wide temperature ranges (−200 to 1200°C typical)

(continued)

Table 4.1 (continued)

Sensor	Features
Thermostats	Very high sensitivity in medium ranges (up to 100 C typical). Compact but nonlinear in nature
Thermo diodes, and thermo transistors	Ideally suited for chip temperature measurements. Minimized self heating.
RTD-resistance temperature detector	More stable over a long period for time compared to thermocouple. Linear over a wide range
Infrared type and Infrared thermograph	Noncontact point sensor with resolution limited by wavelength. Measures whole-field temperature distribution
Proximity sensors	
Inductance, eddy current, hall effect, photoelectric, capacitance, etc.	Robust noncontact switching action. The digital outputs are often directly fed to the digital controller
Light sensors	
Photo resistors, photodiodes, photo transistors, photo conductors, etc. Charge-coupled diode.	Measure light intensity with high sensitivity. Inexpensive, reliable, and noncontact sensor. Compares digital image of a field of vision
Smart material sensors	
Optical fiber as strain sensor, as level sensor, as force sensor, and as temperature sensor	Alternate to strain pages with very high accuracy and bandwidth. Sensitive to the reflecting surface's orientation and status. Reliable and accurate. High resolution in wide ranges. High resolution and range (up to 2000°C)
Piezoelectric as strain sensor, as force sensor, and as accelerometer	Distributed sensing with high resolution and bandwidth. Most suitable for dynamic applications. Least hysteresis and good set point accuracy
Magnetostrictive as force sensors and as torque sensor	Compact force sensor with high resolution and bandwidth. Good for distributed and noncontact sensing applications. Accurate, high bandwidth, and noncontract sensor
Micro- and nano-sensors	
Micro CCD image sensor, fiberscope micro-ultrasonic sensor, micro-tactile sensor	Small size, full field image sensor, Small (0.2 mm diameter) field vision scope using SMA coil actuators. Detects flaws in small pipes. Detects proximity between the end of catheter and blood vessels

4.4 Augmented Reality for Sensors, Transducers, and Actuators

Sensors, transducers, and actuators implement close loop control in CNC based systems, PLC based systems, Mechatronics based systems, SCADA based system and systems relying on embedded systems. Sensors and transducers trigger on/off

Table 4.2 Types of actuators and their features

Actuator	Features
Diodes, bipolar transistor, triacs, diacs, power MOSFET, solid state relay, etc.	Electrical Electronic type. Very high frequency response. Low power consumption
	Electromechanical
DC Motor, wound field, and separately excited shunt series	Speed can be controlled either by the voltage across the armature winding or by varying the field current. Constant speed application. High starting torque, good speed regulation. Instability at heavy loads
Permanent magnet, conventional PM motor, moving-coil PM motor, and torque Motor	High efficiency, high peak power, and fast response. Higher efficiency and lower inductance than conventional DC motor. Designed to run for long periods in a stalled or a low rpm condition
Electronic commutation (brushless motor)	Fast response. High efficiency, often exceeding 75%. Long life, high reliability, no maintenance needed. Low radio frequency interference and noise production
AC Motor, AC induction motor, and AC synchronous motor	The most commonly used motor in industry. Simple rugged, and inexpensive. Rotor rotates at synchronous speed. Very high efficiency over a wide range of speeds and loads. Need an additional system to start
Universal motor	Can operate in DC or AC. Very high horsepower per pound ratio. Relatively short operating life
Stepper motor and hybrid variable reluctance	Change electrical pulses into mechanical movement. Provide accurate positioning without feedback. Low maintenance
Electromagnetic	
Solenoid type devices, electromagnets and relay	Large force, short duration. On/off control
Hydraulic and pneumatic	
Cylinder hydraulic motor	Suitable for liner movement
Gear type	Wide speed range
Piston type	High horsepower output
Air motor rotary type	High degree of reliability
Reciprocating	No electric shock hazard
Valves directional control valves, pressure control valves, and process control valves	Low maintenance
Smart material actuators	
Piezoelectric and Electrostrictive	High frequency with small motion. High voltage with low current excitation. High resolution.
Magnetostrictive	High frequency with small motion. Low voltage with high current excitation
Shape memory alloy	Low voltage with high current excitation. Low frequency with large motion

(continued)

Table 4.2 (continued)

Actuator	Features
Electrorheological fluids	Very high voltage excitation. Good resistance to mechanical shock and vibration. Low frequency with large force
Micro- and mono-actuators	
Micromotors microvalves	Suitable for micromechanical system. Can use available silicon processing technology, such as electrostatic motor.
Micropumps	Can use any smart material

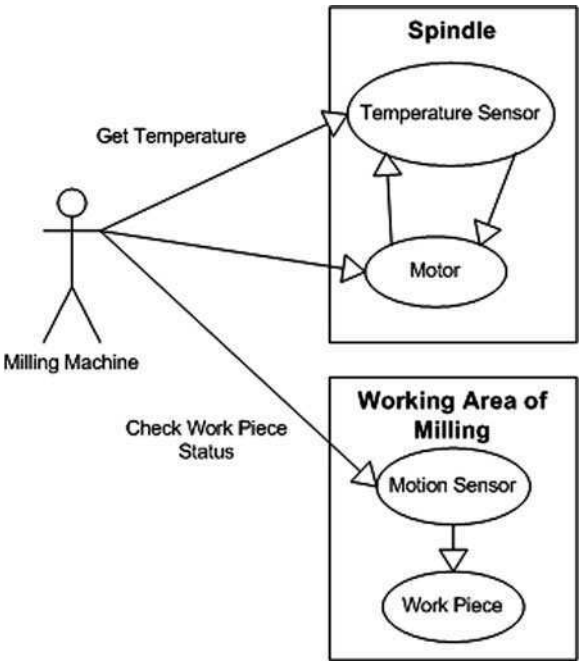


Fig. 4.1 Sensor processes use case diagram

action of the actuator based on switching mechanism in a close loop strategy. Figures 4.1, 4.2 and 4.3 show the use case diagram, the sequence diagram and process flow chart for sensor processes. Figures 4.4, 4.5, and 4.6 present the use case diagram, the sequence diagram and process flow chart for transducers processes. Figure 4.7, 4.8 and 4.9 describes the use case diagram, the sequence diagram and process flow chart for actuator processes.

The activities performed by the sensors, transducers and actuators are programmed through the interpreters and compilers used for operating the virtual or the real machinery.

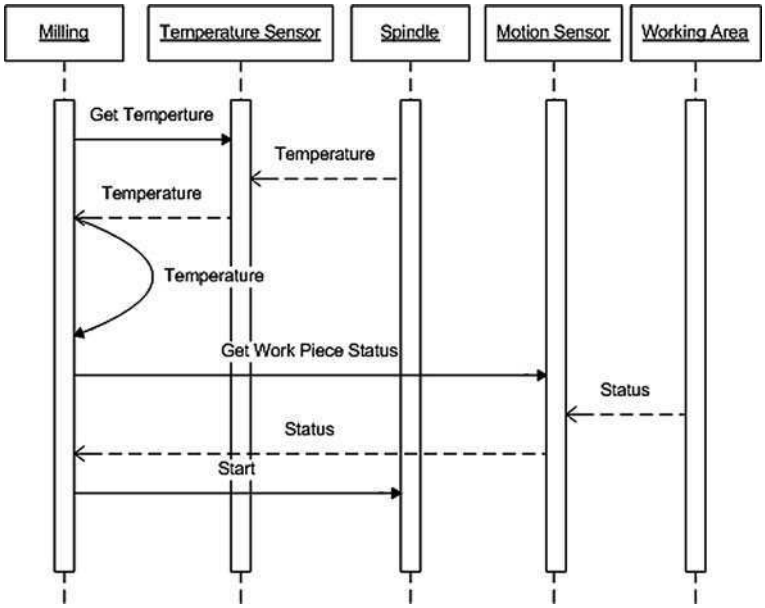


Fig. 4.2 Sensor processes sequence diagram

4.5 System Integration Methodology

VMS Monitor performs the function of monitoring the real factory and all the signals from every close loop circuit are copied to the virtual factory. This allows the other modules of the VMS software to act, as per programmed operation and any deviation may be dealt with accordingly, e.g., VMS Fault Diagnostic by virtue of above procedure is able to detect any malfunction.

Sensors, transducers, and actuators are the key components in the operation of discrete manufacturing facility. It is imperative to copy the signals emanating from these components to the virtual factory so as to have the total status of the system under consideration.

The control of sensor, transducer, and actuators are physically monitored through a controller to the control consoles operating the virtual factory via a local area network. Some examples of these electronic interfaces are shown in Figs. 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, and 4.17.

Appendix-IV provides the data sheets for all major integrated circuits. Power supply circuit for all the circuits used in this book is shown in Appendix-V.

A general view of a model virtual factory operating using above scheme, i.e., using sensors, transducers, and actuator in a manufacturing environment is provided in Chap. 11

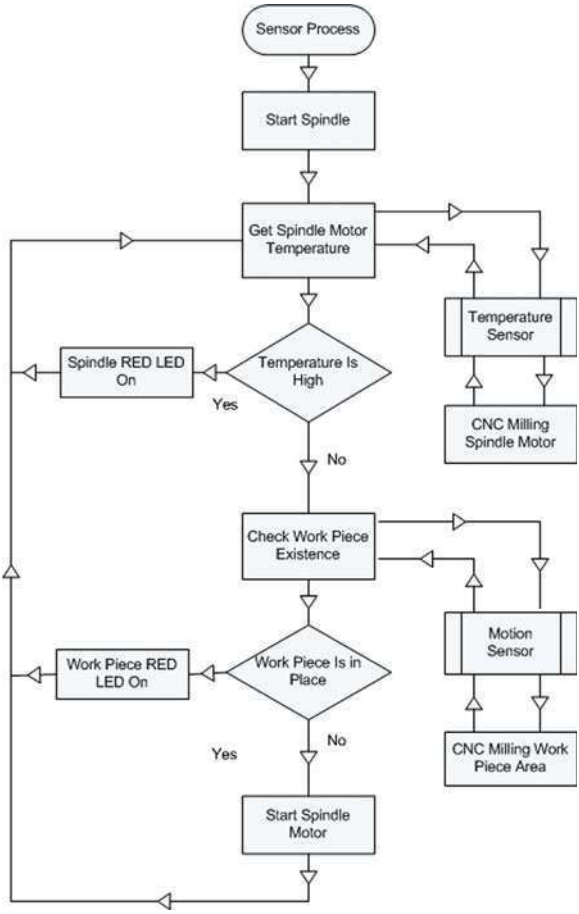


Fig. 4.3 Sensor processes flow chart

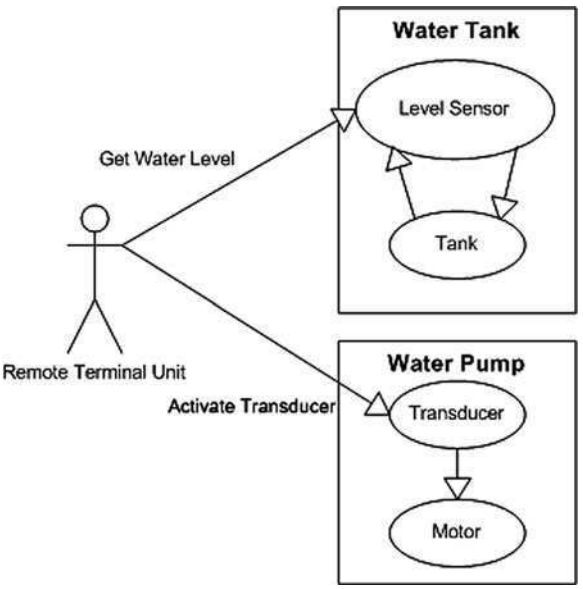


Fig. 4.4 Transducer processes use case diagram

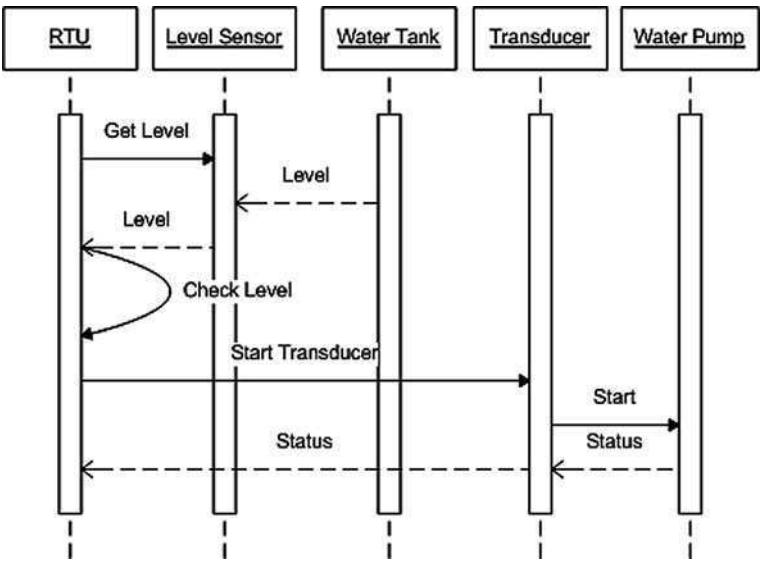


Fig. 4.5 Transducer processes sequence diagram

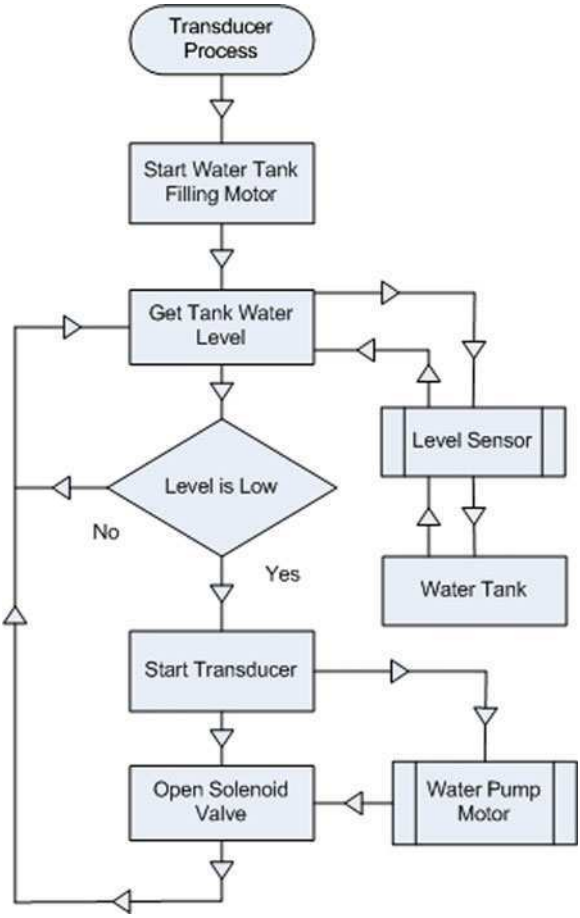


Fig. 4.6 Transducer processes flow chart

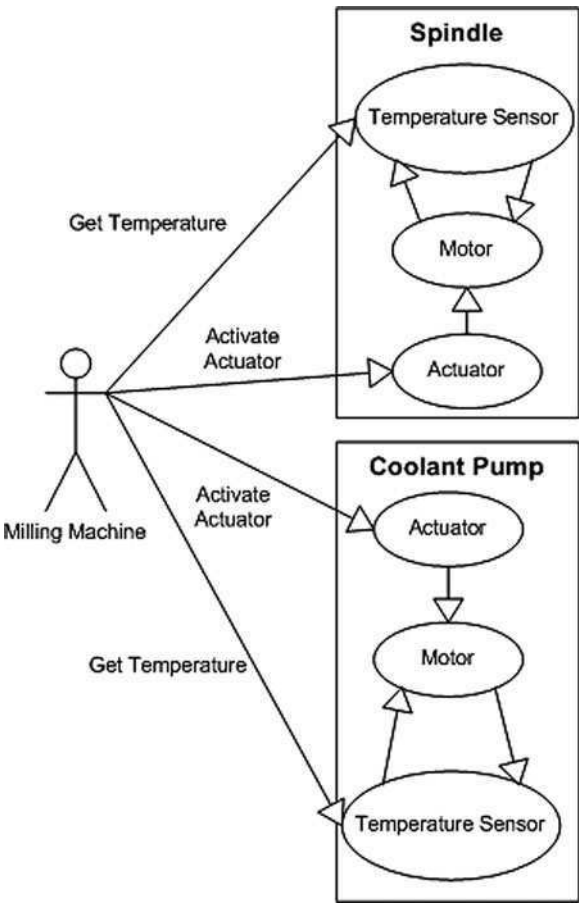


Fig. 4.7 Actuator process use case diagram

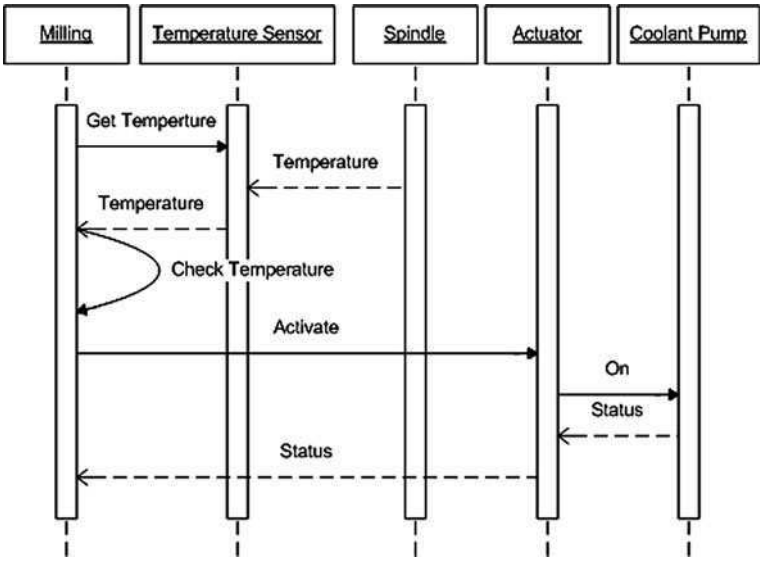


Fig. 4.8 Actuator processes sequence diagram

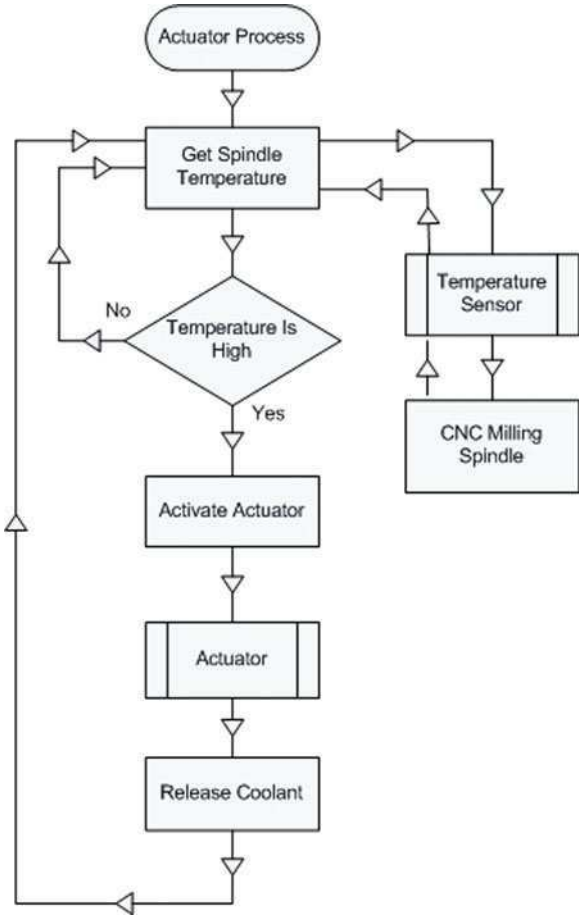


Fig. 4.9 Actuator processes flow chart

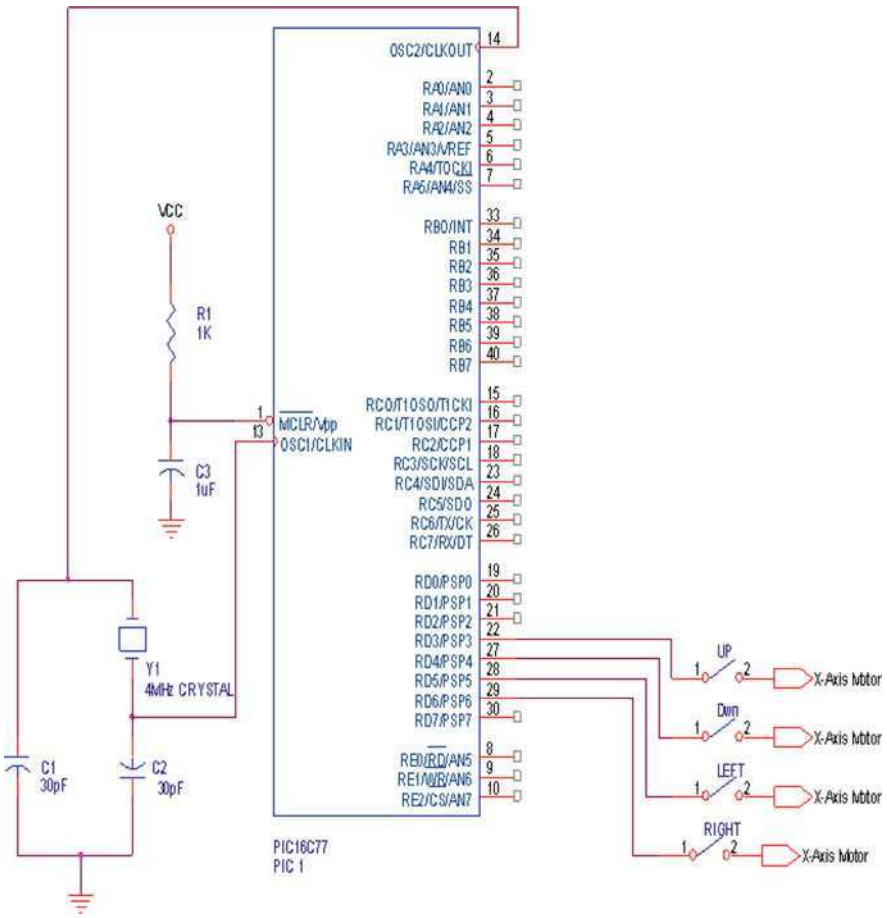


Fig. 4.10 Control of limit switches over local area network (continued)

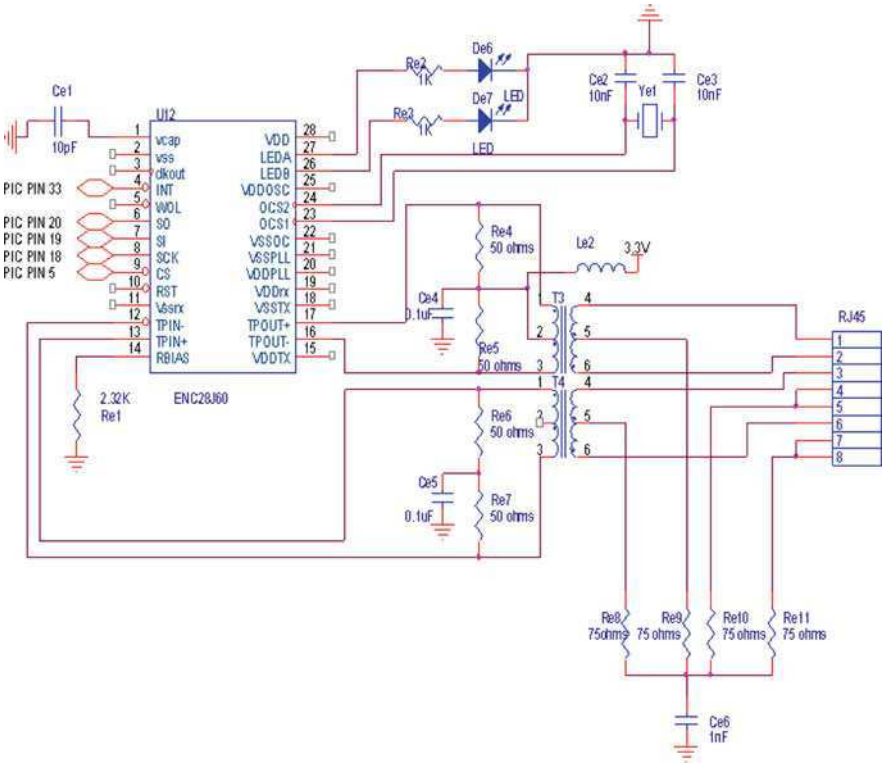


Fig. 4.10 (continued)

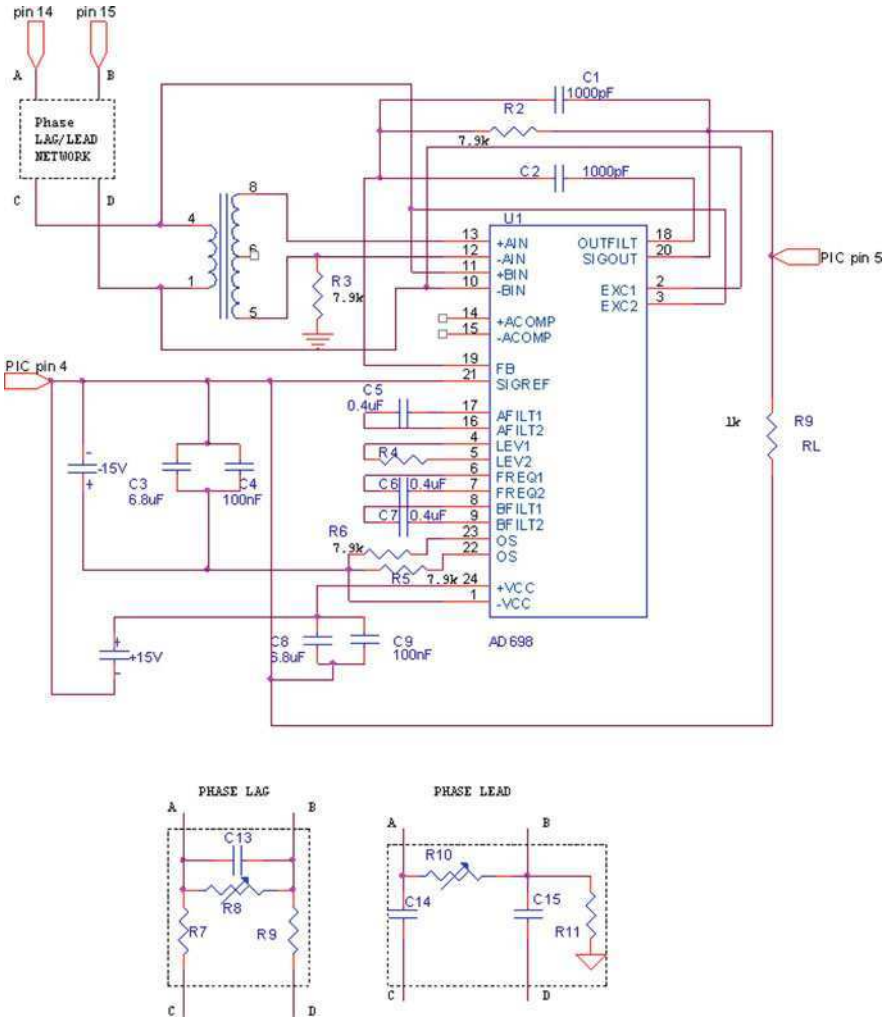


Fig. 4.11 Control of linear variable differential transformer (LVDT) over local area network (continued)

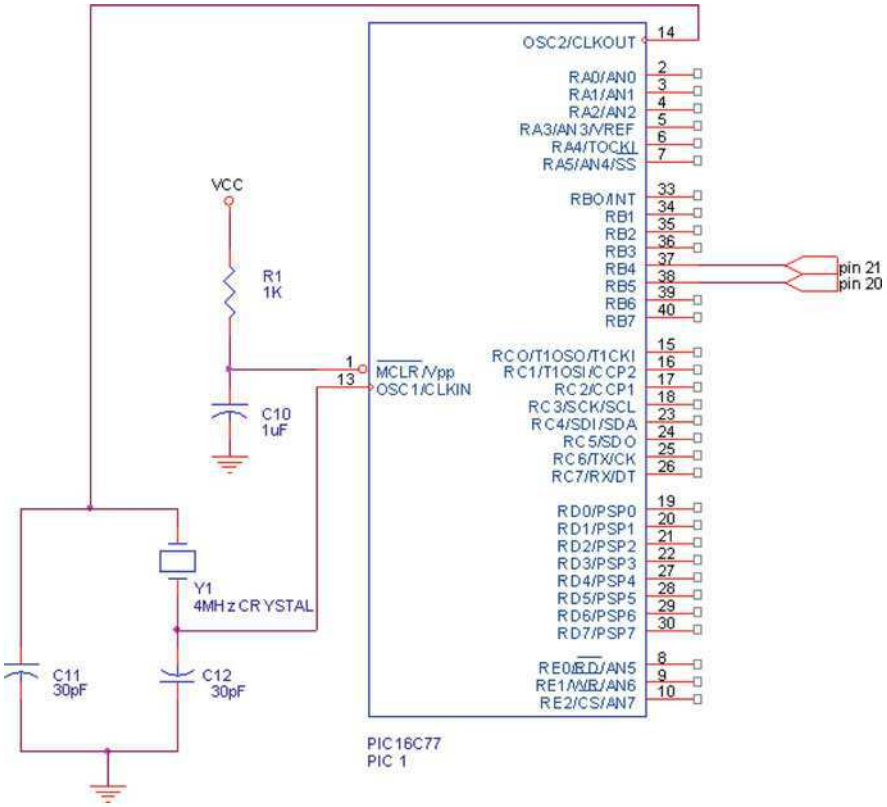


Fig. 4.11 (continued)

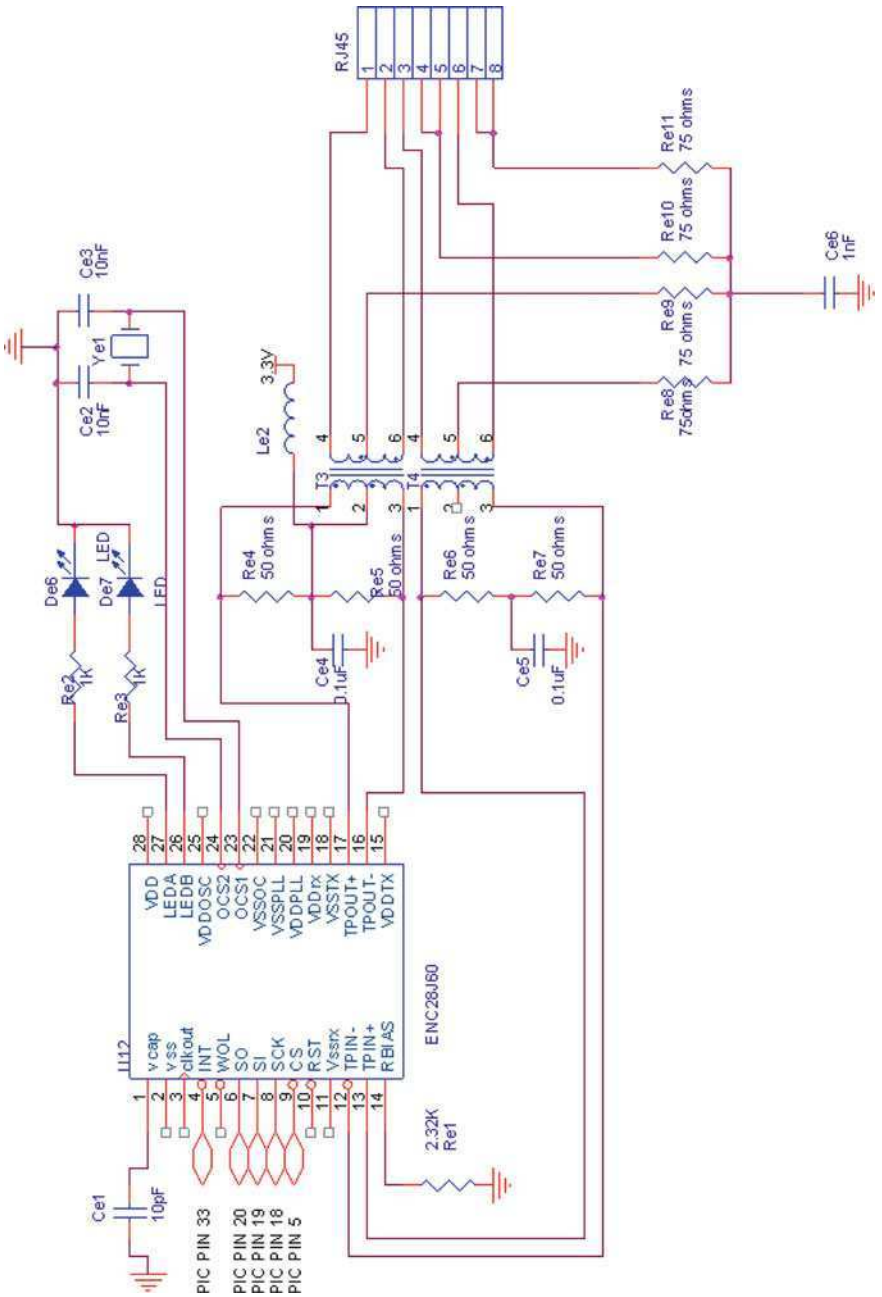


Fig. 4.11 (continued)

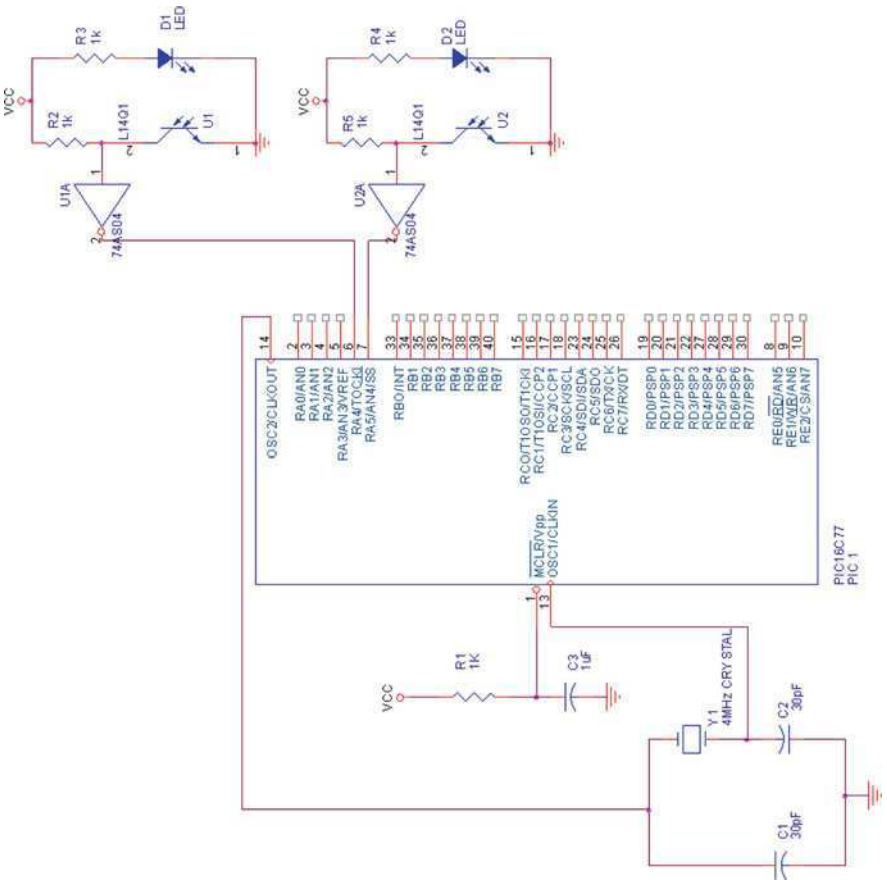


Fig. 4.12 Control of shaft encoder over local area network (continued)

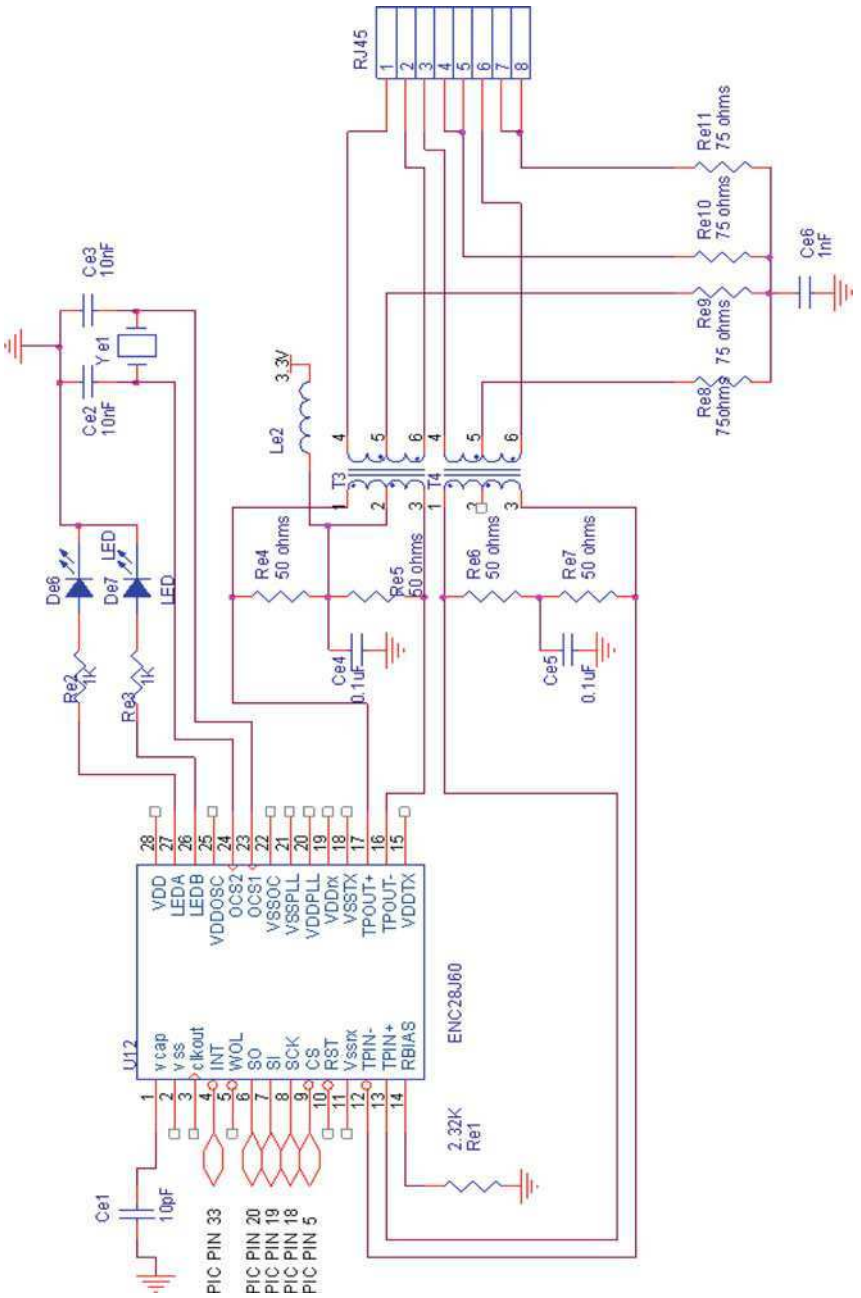


Fig. 4.12 (continued)

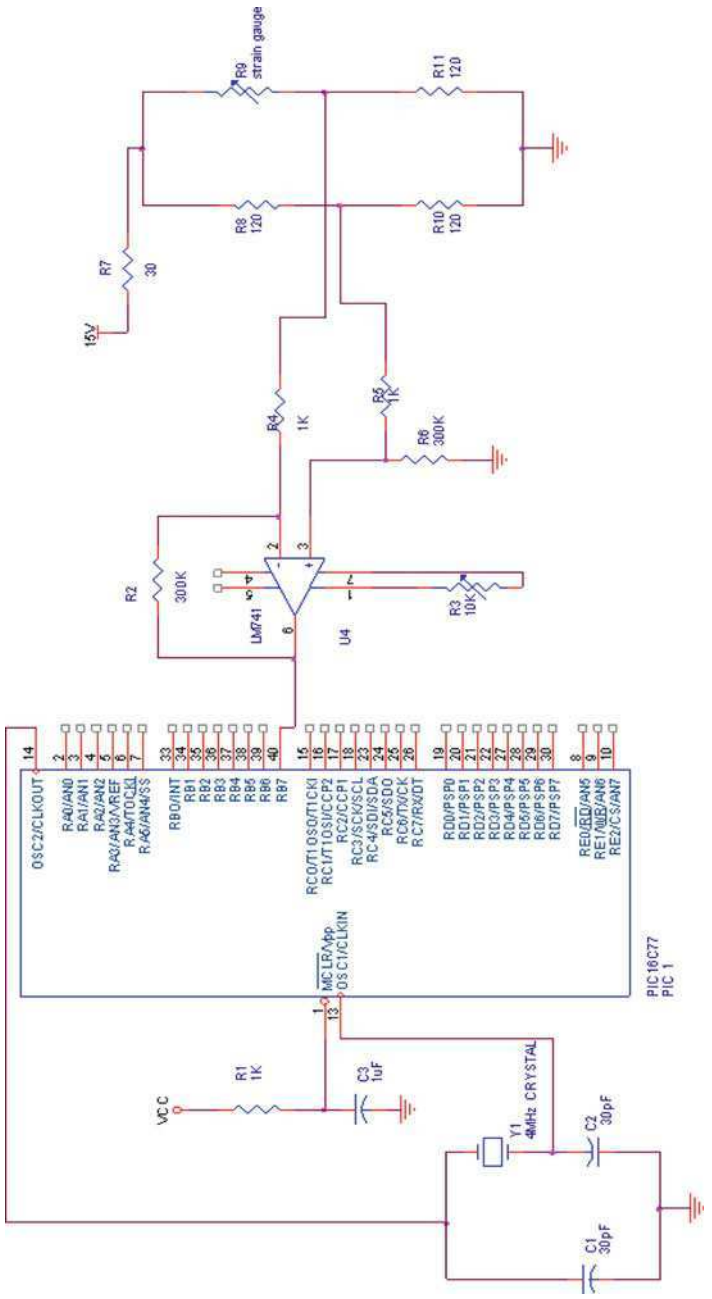


Fig. 4.13 Control of load cell over local area network (continued)

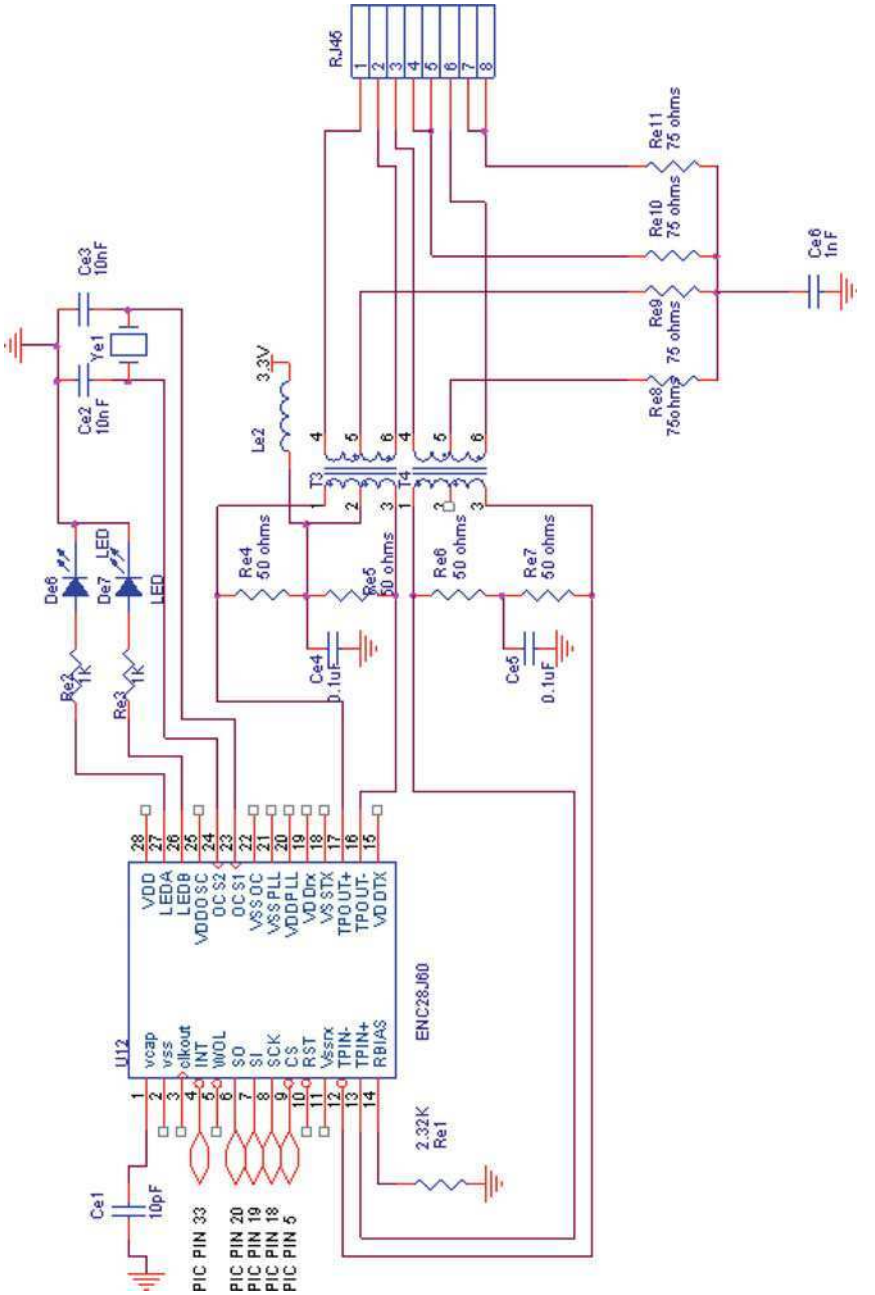


Fig. 4.13 (continued)

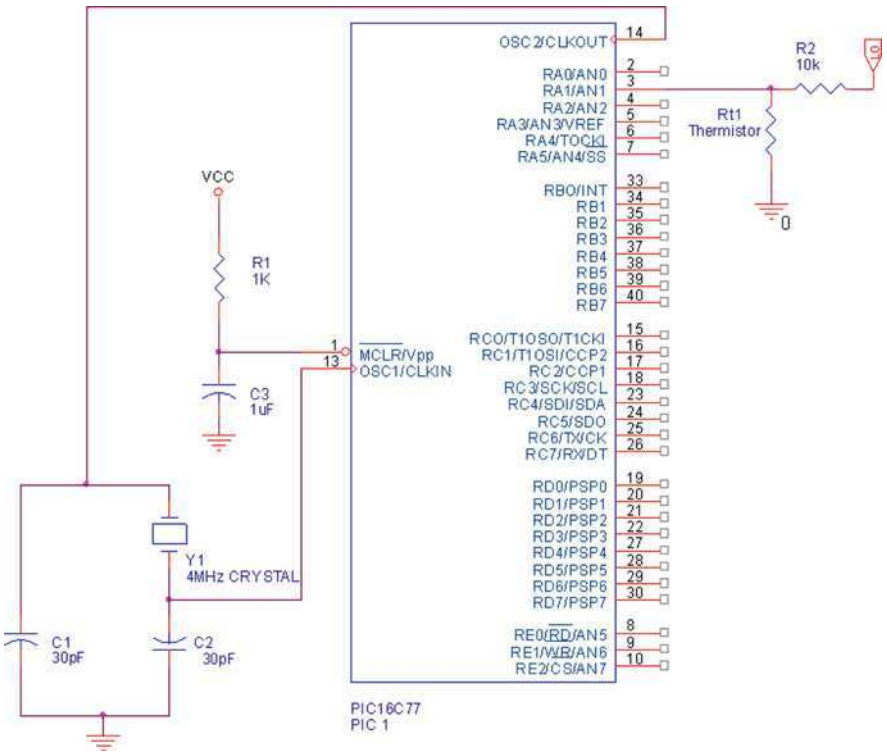


Fig. 4.14 Control of thermistor over local area network (continued)

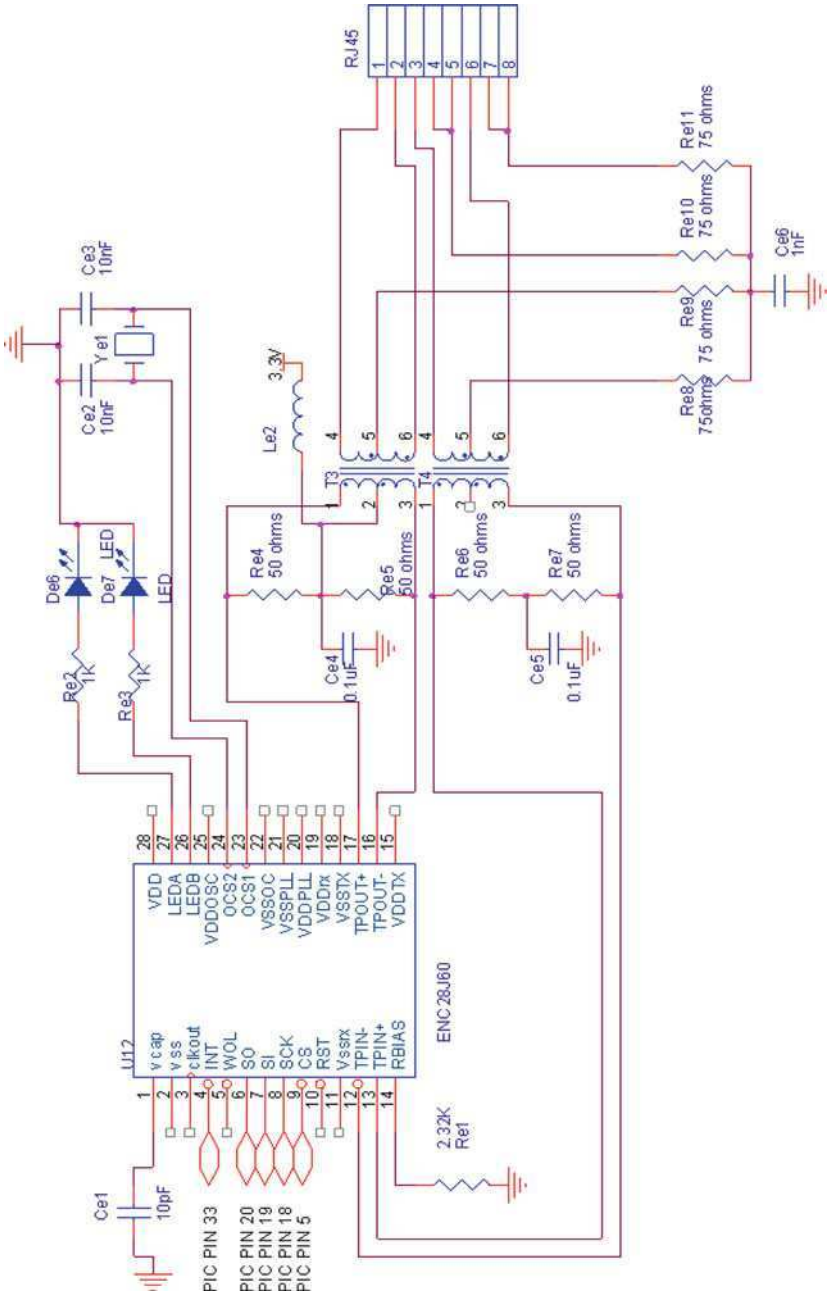


Fig. 4.14 (continued)

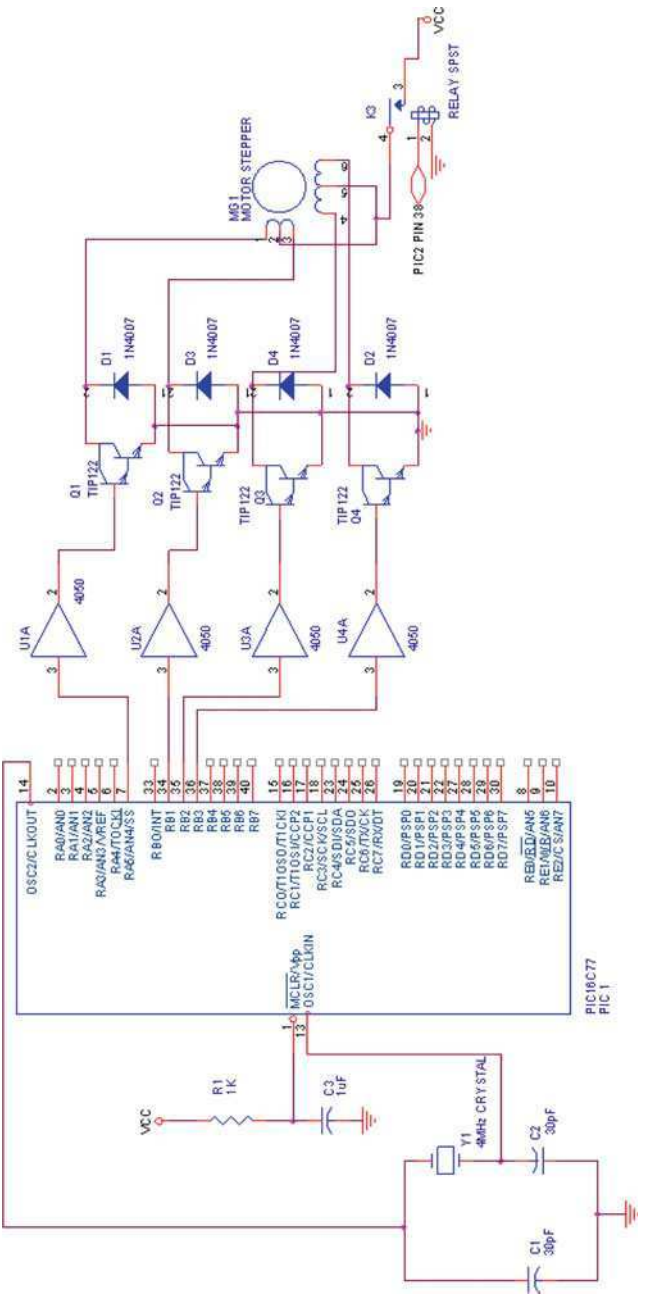


Fig. 4.15 Control of stepper motor over local area network (continued)

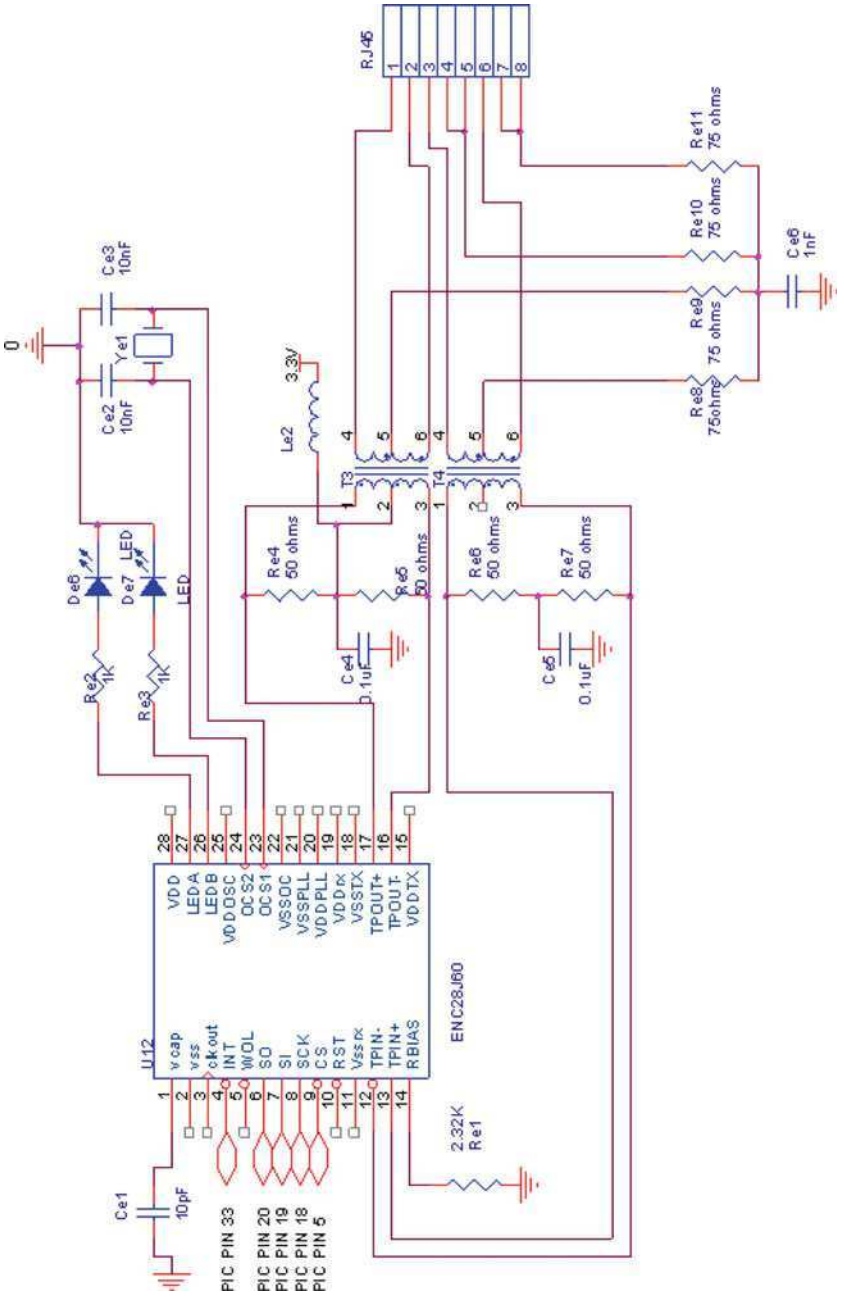


Fig. 4.15 (continued)

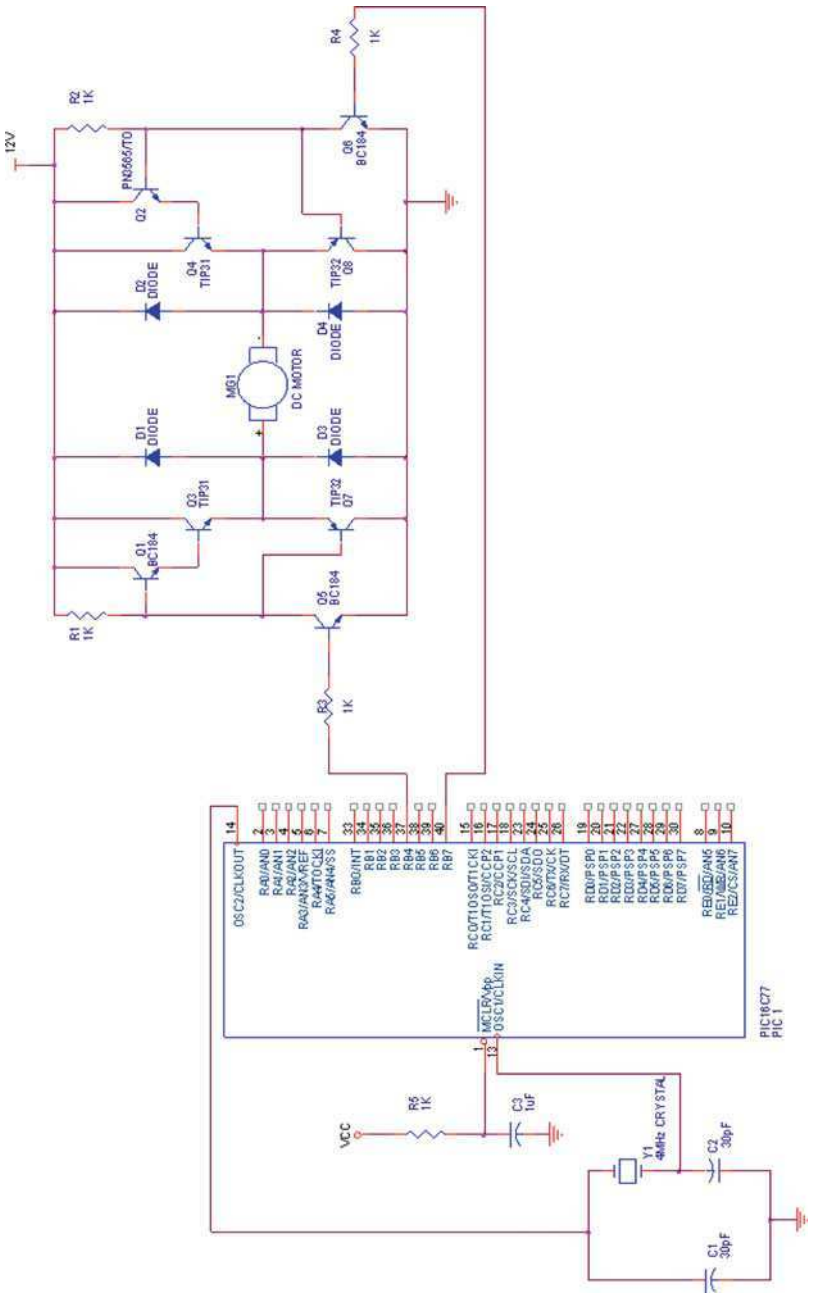


Fig. 4.16 Control of DC Motor over local area network (continued)

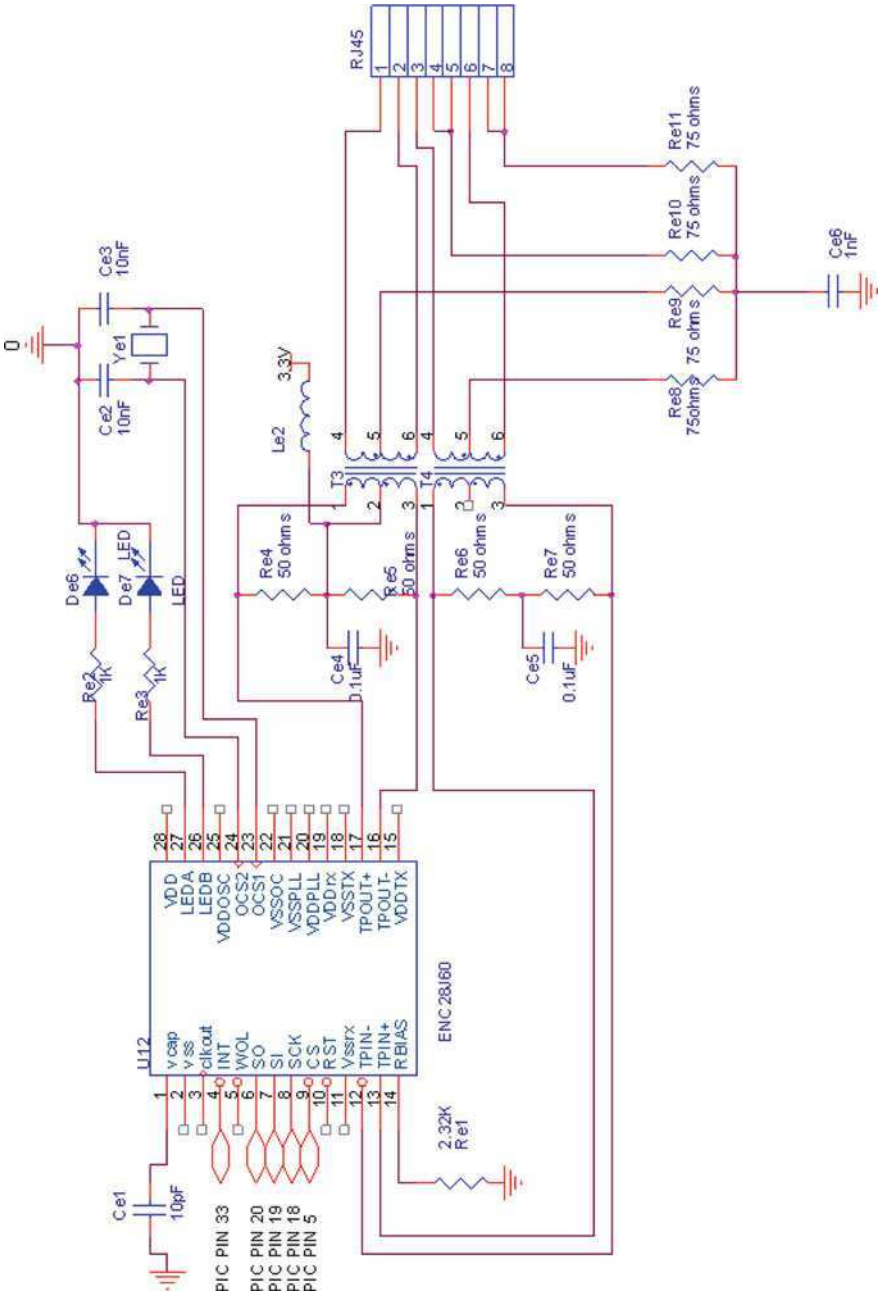


Fig. 4.16 (continued)

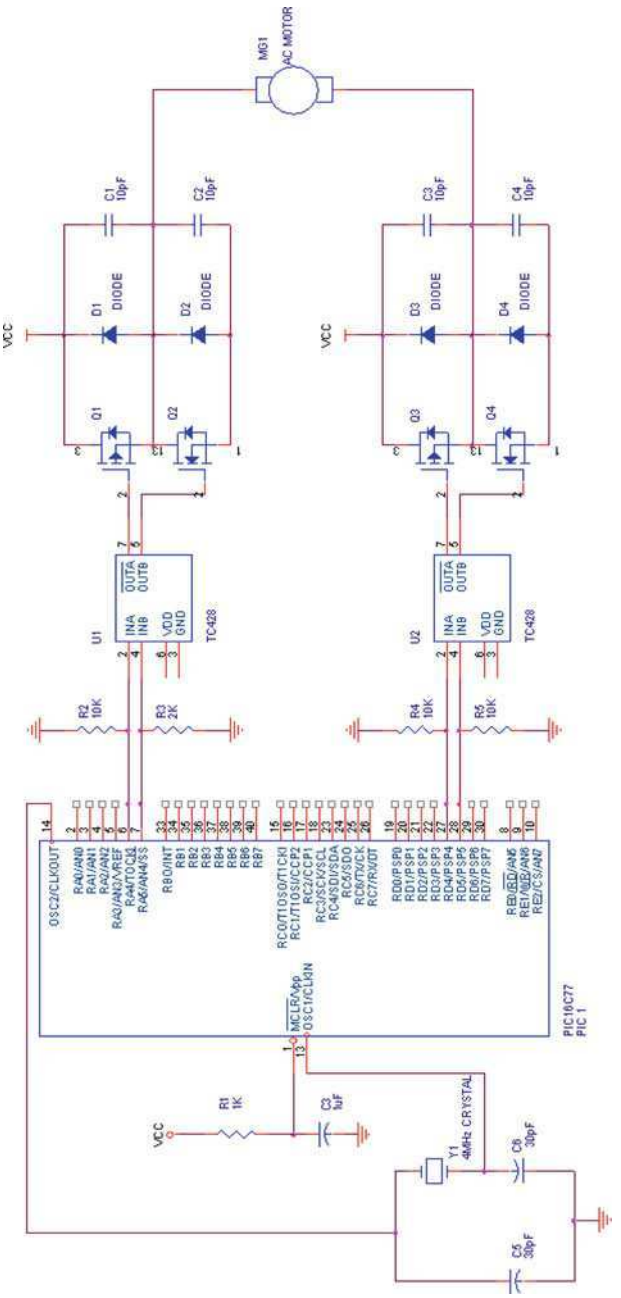


Fig. 4.17 Control of AC motor over local area network (continued)

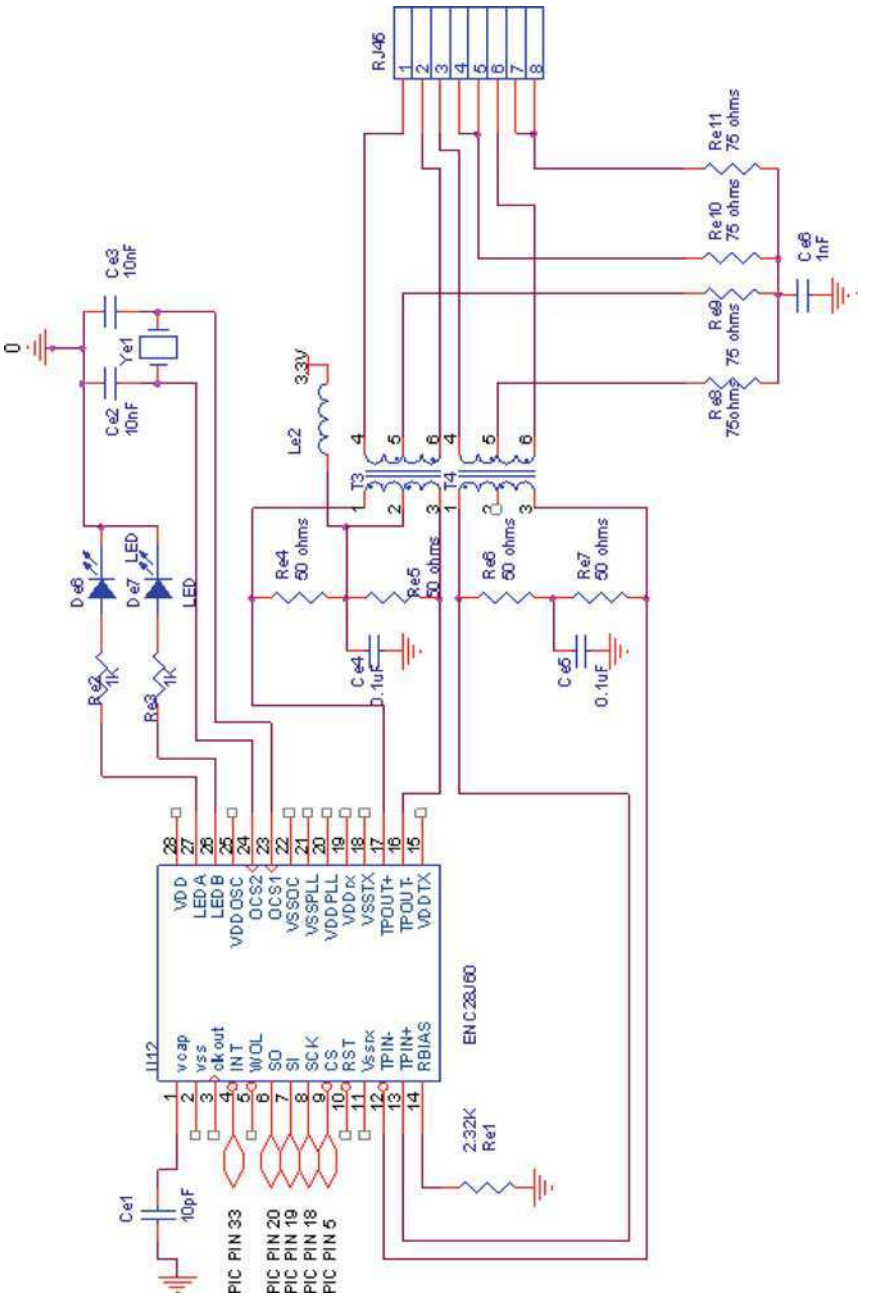


Fig. 4.17 (continued)

Bibliography

1. Blake J (2009) Haptic glove with MR brakes for virtual reality. IEEE/ASME Transac Mechatron 14(5):606–615
2. Bolzmacher C (2004) Polymer based actuators for virtual reality devices. In: Proceedings of SPIE—the international society for optical engineering 281–289
3. de Jong D, Paul C et al (2009) High-temperature electronic system for pressure-transducers. IEEE Transac Instrum Meas 49(2):365–370
4. Folgheraiter M (2008) A multi-model haptic interface for virtual reality and robotics. J Intell Robotics Syst Theory Appl 52(3–4):465–488
5. Gardner JW et al (2001) Micro sensors, MEMS and smart devices. Wiley, Chichester
6. Gulrez T (2007) Precision position tracking in virtual reality environments using sensor networks In: IEEE International Symposium on Industrial Electronics 1997–2003
7. Her MG et al (2002) Analysis and design of a haptic control system: virtual reality approach. Int J Adv Manuf Technol 19(10):743–751
8. Kon T (2006) A method for supporting robot's actions using virtual reality in a smart space. SICE-ICASE international joint conference 3519–3522
9. Koshal D (1993) Manufacturing engineer's reference book. Butterworth-Heinemann, Group, Oxford. Elsevier
10. Li Z (2001) Virtual reality modeling aiding in MEMS design. In: Proceedings of SPIE—the international soceity for optical engineering 153–162
11. Merlo AM et al (2006) Application of magneto-elastic sensors to the measurement of fluid flow rates in the automotive domain. In: Proceedings of 8th biennial ASME conference on engineering systems design and analysis, ESDA 2006
12. Norman P et al (2003) Opportunities, problems and solutions when instrumenting a machine tool for monitoring of cutting forces and vibrations. Laser Metrol Mach Perform VI, 557–567
13. Pallas-Areny R, Webster JG (2001) Sensors and signal conditionary, 2nd edn. Wiley, New York
14. Probst M (2007) Virtual reality for microassembly In: Proceedings of SPIE—the international society for optical engineering
15. Sheridan TB (2004) Musings on music making and listening: Supervisory control and virtual reality. In: Proceedings of the IEEE, pp 601–605
16. Tao B et al (2006) A novel peer-to-peer distributed sensor network framework based on IP sensor for telemonitoring. Assembl Autom 26(2):137–142
17. Tao B et al (2008) Role identification and reallocation of a distributed sensor network in manufacturing floor. Lect Notes Comp Sci, V5315 LNI (Part 2):869–878

Chapter 5
Augmented Reality for Computer Numerical Control-Based Applications

5.1 Introduction to CNC-Based Applications

The subject of manufacturing commenced when human altered the geometry of work pieces by rotating either a tool or a work piece to cut and to remove pieces of the material in the production of weapons as early as 700 B.C. It was not until the fifteenth century that the machining of metal begun.

After the eighteen-century industrialization, metal processing became a trade carried out by experienced craftsman, which passed from generation to generation. The introduction of the concept of interchangeability of parts by Eli Whitney (1765–1825) forced machined parts to be related to specified tolerances. The improved standards that emerged led to increased responsibility for the craftsman.

From this period until relatively recent times, the basic design of machine tools has remained unchanged, except that the source of power has changed from human to water, to steam, to electricity. The machine tools themselves have become lighter, stiffer, and faster but the machine operator interface has remained under the control of skilled humans.

The introduction of numerical control (NC), in the early 1950s, was a major step toward decreasing the human involvement in the manufacture. The repetitive production of component within very close tolerance in one-off to medium-sized batches became reality.

Currently, computer numerical control (CNC) is one of the most versatile human computer interface utilized in contemporary manufacturing. Figures 1.4 and 1.5 provide the basic description of open-loop and closed-loop computer numerical control manufacturing processes. The application of the concept associated with computer numerical control (CNC) is not limited to discrete manufacturing processes. The same concept of programmable user interface has been applied to number of other discrete products in the form as shown in Fig. 5.1.

Discrete manufacturing processes are developed as mechanical artifacts to perform production. These machines produce variety of components, structures,

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 127
DOI: 10.1007/978-0-85729-186-8_5, © Springer-Verlag London Limited 2011

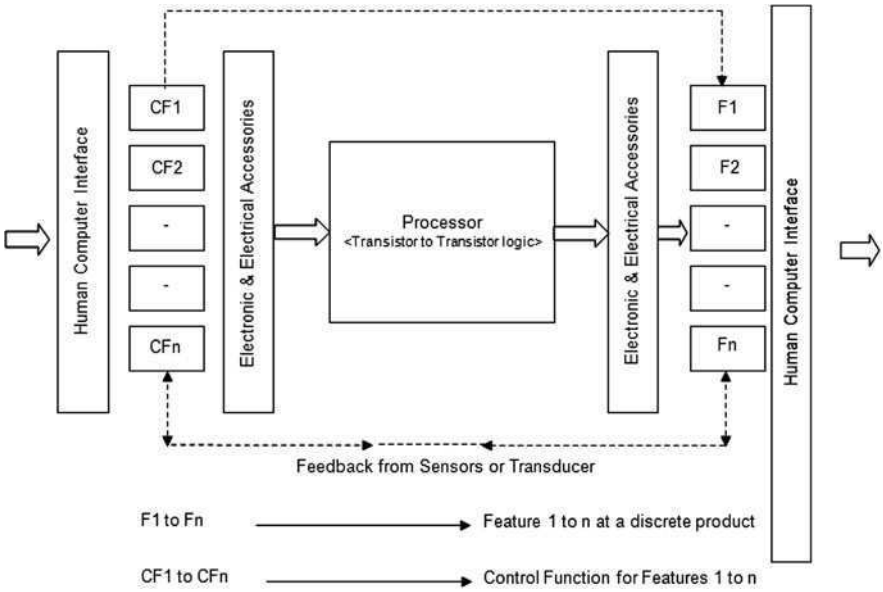


Fig. 5.1 Control of features at discrete products. Reproduced with permission from Khan and Raouf [59]

assemblies, mechanisms, and machines. Each manufacturing process implementation into mechanical artifacts has three distinct features:

1. The input to the equipment, e.g., raw material type, form and feeding mechanism; final dimension of the product, energy source and other auxiliaries.
2. The process implementation allowing transformation of raw material into required size, shape, and surface finish using tools (e.g., tools as used in metal cutting, high-energy beams or various types of jets); utilizing tool-holding devices and work-holding devices; measuring devices and manufacturing instructions. In process, supplies such as lubricating oil and coolants may also be used.
3. The output from the equipment comprising a component (the building block of structure, mechanisms or machines) and scrap.

There may be several other features present at the production machinery to make the task of manufacturing simpler, easy to control and resulting in high production rate.

Like any other discrete product, the manufacturing equipment commonly utilizes standard mechanical components, machine elements, control elements, electrical and electronics components, and software modules.

Special assemblies and other accessories utilized at the construction of manufacturing equipment may also include:

1. Tool (referring to mechanical tool, light beams, water jets, etc.)
2. Tool-holding devices
3. Work-holding devices
4. Lubricating oil pump assembly
5. Coolant circulation pump assembly
6. Material-handling equipment, and
7. Scrap-handling equipment

Computer numerical control, whether implemented in open-loop or closed-loop configuration, is the most common human computer interface at discrete manufacturing equipment. It is widely used to control features, assemblies, and accessories at the discrete manufacturing machinery. Computer numerical control is commonly implemented at the following discrete manufacturing equipment:

1. Mills and machining centers
2. Lathe and turning centers
3. Drilling machines
4. Boring and profilers
5. Electro discharge machines
6. Punch press and shears
7. Flame-cutting machines
8. Water jet and laser profilers
9. Cylindrical grinders
10. Welding machines
11. Benders, winding and spinning machines.

The above-listed processes along with processes listed in Table 2.2 cover the complete horizon of discrete manufacturing process classification as identified by Kalpakjian.

The programming languages used to exercise computer numerical control are classified as:

(i) Interactive (standard)
(ii) Interactive (user defined)
(iii) High-level computer programming languages
(iv) G and M codes (user defined); and
(v) G and M codes (standard)

G and M codes are the most common pneumonic-based language used to operate the human computer interface at the computer numerical machinery. Electronics Industries Association (EIA) RS 274 D interchangeable variable block data format for positioning, contouring and contouring/positioning numerically controlled machines is one of the most basic and most common standards used in the manufacturing industry for controlling CNC machinery.

A subset of this standard is used to demonstrate augmented reality design methodology for computer numerical control machinery.

Axes classification for numerically controlled machines is defined by another commonly adopted EIA standard: EIA RS 267 Axis and motion nomenclature for numerically controlled machines.

This standard is utilized for the axes classification of CNC machinery. Other series of standards in the area of computer numerical control are defined by International Standards Organization (ISO) British Standard Institute (BSI) and other organizations as listed in Sect. 5.3. Several CNC machine tool manufacturers have also defined company standards for various features of this technology.

It is partially because the international standards or standards from national or professional organizations may not cover features otherwise defined and practiced by the machine tool manufacturers for their products.

Defining the virtual reality design for computer numerical control (CNC)-based applications is a three-step process involving:

(a) Defining and rendering augmented reality representation of discrete manufacturing process,
(b) Interpreter design for the selected programming language used to implement the computer numerical control (CNC) at the discrete manufacturing process, and
(c) Electronic interface circuitry required to align virtual CNC process to real CNC process as depicted in Fig. 5.2.

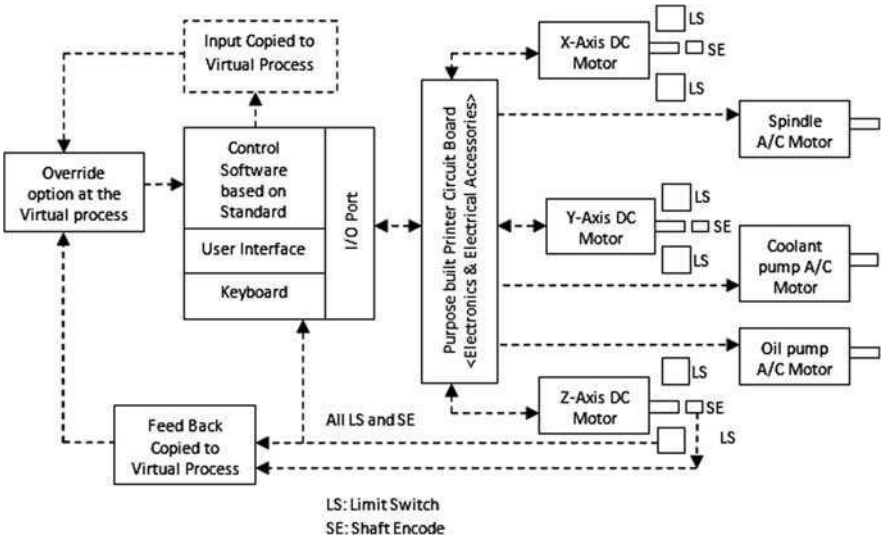


Fig. 5.2 General layout for milling machine CNC controller with proxy virtual process. Reproduced with permission from Khan and Raouf [59]

5.2 Components of Machine Tools for Augmented Reality Design

The augmented reality design for discrete manufacturing processes requires programmable control of following, all or selected functions/devices:

1. Machine tool axes
2. Tool[1] (tool changer)
3. Spindle
4. Pallet (pallet changer)
5. Work-holding device
6. Measurement devices
7. Coolant pump
8. Lubrication pump
9. Raw material feeder
10. Scrap-handling device
11. Finished product-handling device
12. Safety mechanism
13. Special features associated with non-conventional processes
14. Special features associated with equipment used for fabrication of MEMs devices
15. Electronics communications options.

Selection of above feature for AR implementation is based on the type of manufacturing equipment, required level of automation, and the available budgetary allocation.

A block diagram describing general layout for a milling machine controller with proxy virtual processes is given in Fig. 5.2.

5.3 Standards Pertaining to Augmented Reality for CNC-Based Machinery

Virtual reality design for CNC-based operations can be best practiced through the use of established standards. There are several corporate, national, and international standards organizations currently involved in the development of CNC-related standards. Some of these selected standards are listed below:

1. SME MS900251 01-06-1990
   Computer numerical control (CNC) spiral bevel gear grinding

[1] Tools as cutting tools, Water Jet, Water Abrasive Jet, Torches, Laser Beam, Plasma and Ultrasonic process.

2. FED A-A-59234 30-06-1998
Electrical discharge machine, manual and computer numerical control (CNC)
3. BS ISO 10303-203:1994 15-0l-1996
Industrial automation systems and integration, product data representation and exchange, application protocol, configuration-controlled design
4. FED A-A-59171 18-09-1998
Machining center, vertical, double-column, single-spindle, computer numerical control (CNC)
5. SME MS910546 01-06-1991
A methodology for computer integration of CNC machinery at minimal cost
6. BS EN ISO 2806:1997 15-02-1995
Industrial automation systems, numerical control of machines, vocabulary
7. ISO 841:2001 01-10-2001
Industrial automation systems and integration—Numerical control of machines—Coordinate system and motion nomenclature
8. SME MS98-258 01-10-1998
Development of an EDM CNC system for conjugate machining
9. BS ISO/IEC 10967-2:2001 24-08-2001
Information technology: Language independent arithmetic, elementary numerical functions
10. BS EN 12478:2001 15-01-2001
Safety of machine tools. Large numerically controlled turning machines and turning centers
11. A-A-59118 Revision: 97 Chg: Vn1 Date: 18-04-2003
Machining center, horizontal, single-spindle, computer numerical control (CNC)
12. A-A-59332 Revision: 98 Chg: Vn1 Date: 30-03-2005
Grinding machine, cylindrical, external, center-type, computer numerical control (CNC)
13. A-A-59415 Revision: 99 Chg: Vn1 Date: 25-10-2005
Grinding machine, universal, vertical-spindle, non-traversing rotary table, computer numerical control (CNC)
14. A-A-59459 Revision: 99 Chg: Vn1 Date: 07-04-2006
Grinding machine, tool and cutter, computer numerical control (CNC)
15. A-A-59171 Revision: 98 Chg: Cn1 Date: 30-06-2004
Machining center, vertical, double-column, single-spindle, computer numerical control (CNC)
16. Mil-L-80219 Revision: B Chg: Cn1 Date: 07-03-2000
Lathes, shaft turning, turret, vertical bed and slant bed, computer numerically controlled (CNC)
17. Mil-L-80245 Revision: A Chg: Cn1 Date: 29-03-2000
Lathe, multiple purpose, chucking, horizontal spindle, computer numerically controlled (CNC)

18. Mil-M-80212 Revision: B Chg: Cn1 Date: 17-04-2000
    Milling machines, vertical, single and multiple spindle, computer numerically controlled (CNC), and tracer-controlled profiler.

A vast majority of associated standards can also be found from catalogs of relevant standards organization. Information about standards from the website of international vendors of standards can be found with the benefit that these vendors represent larger number of standards developing organizations. The shelf life of standards is much higher than that off a research paper or a book. It is therefore advisable that search for a relevant standard should be carried out in detail and should span over number of years. The search should also take into account the current agreements between standards developing organizations and the committee work related to a new standard or the standard going through a revision.

5.4 Augmented Reality Design for CNC-Based Discrete Manufacturing Processes

In order to build an augmented reality for CNC-based discrete manufacturing processes, following steps are necessary:

1. Identification of functions/devices requiring programmable control at the machine tool
2. Computer-aided software engineering (CASE) modeling for the control of proposed function/devices considering the type of the machine tool and the selected method for programming the human–computer interface machine tool.
3. Development of an interpreter/compiler capable of:

(a) Translating selected CNC commands ideally based on an international standard
(b) Allowing user access to the manufacturing process through an augmented reality interface comprising same functions as that of CNC controller at the machine tool
(c) Sending binary signals to the actuators at the machine tool. Receiving signal from the sensors and transducers mounted on the machine tool to monitor the status of the machine tool and handling the function override.
(d) Providing the online graphics details of the operational process to the augmented reality user interface

For details on AR design, also refer to Appendix II. A large number of software development tools are currently available to implement above tasks in concurrence with appropriate hardware. Following paragraphs provide brief detail of tools used in the development of AR software for CNC-based processes.

5.4.1 EIA RS274 D Standard

EIA 274 D G and M code is one of the earliest presented and still the most commonly used programming methods for CNC machine tools. The CNC (or the numerically controlled) commands comprise a character from the alphabets followed by at the most two digits. The format of the command is classified through:

(a) Fixed sequential format
(b) Tab sequential format, and
(c) Word address format.

The first two out of three formats listed above have become obsolete. The word address format is practiced commonly. The EIA 274 D terminology describes the components of a part program as follows:

Letter X, Y, Z, I, J, K, L, G, M, S, T, F, S etc.
Word X20, G70, M02, etc.
Block N35 G70 G90 M03 etc.

The complete set of 'letters' used at EIA 274 D standard can be found in the original standard [10]. The 'word' or a 'command' is a combination of available letters and at the most two digits. Only combinations of letters and digits defined through standard are allowed for CNC operation; however, controller manufacturer may defy this rule. Combination of 'Words' is called a 'Block'. It is a complete set of commands representing a logical machining operation. Special characters are also assigned for performing programmable functions at the machine tool.

The EIA 274 D preparatory, and miscellaneous are described in the following sections [10].

5.4.2 Explanation of Functions

Explainatoin of preparatory and miscelleneous functions belonging to EIA 274D is listed below [10]:

G00	Point-to-point positioning	Point-to-point positioning at rapid or other traverse rate
G01	Linear interpolation	A mode of contouring control which uses the information contained in a block to produce a straight line in which the Victoria velocity is held constant
G02	Arc clockwise (2 dimensional)	An arc generated by the coordinated motion of two axes in which curvature of the path of the tool with respect to the workpiece is clockwise, when viewing the plane of motion in the negative direction of the perpendicular axis

(continued)

(continued)

G03	Arc counterclockwise (2 dimensional)	An arc generated by the coordinated motion of two axes in which curvature of the path of the tool with respect to the workpiece is counterclockwise, when viewing the plane of motion in the negative direction of the perpendicular axis
G02–G03	Circular interpolation (2 dimension)	A mode of contouring control which uses the information contained in a single block to produce an arc of a circle. The velocities of the axes used to generate this arc are varied by the control
G04	Dwell	A timed delay of programmed or established duration, not cyclic or sequential; i.e., not an interlock or hold
G06	Parabolic interpolation	A mode of interpolation used in contouring to produce a segment of a parabola. Velocities of the axes used to generate this curve are varied by the control
G08	Acceleration	A controlled velocity increase to programmed rate starting immediately
G09	Deceleration	A controlled velocity decrease to a fixed percent of the programmed rate starting immediately
G13–G16	Axis selection	Used to direct a control to the axis or axes, as specified by the format classification, as in a system which time-shared the controls
G17–G19	Plane selection	Used to identify the plane for such functions as circular interpolation, cutter compensation, and others as required
G33	Thread cutting, constant lead	Mode selection for machines equipped for thread cutting
G34	Thread cutting, increasing lead	Mode selection for machines equipped for thread cutting, where a constantly increasing lead is desired
G35	Thread cutting, decreasing lead	Mode selection for machines equipped for thread cutting, where a constantly decreasing lead is desired
G40	Cutter compensation/ offset cancel	Command which will discontinue any cutter compensation/ offset
G41	Cutter compensation—left	Cutter on left side of work surface looking from cutter in the direction of relative cutter motion with displacement normal to the cutter path to adjust for the difference between actual and programmed cutter radii of diagram
G42	Cutter compensation—right	Cutter on right side to work surface looking from cutter in the direction of relative cutter motion with displacement normal to the cutter path to adjust for the difference between actual and programmed cutter radii or diagram
G43	Cutter offset—inside corner	Displacement normal to cutter path to adjust for the difference between actual and programmed cutter radii or diameters. Cutter on inside corner
G44	Cutter offset—outside corner	Displacement normal to cutter path to adjust for the difference between actual and programmed cutter radii or diameters. Cutter on outside corner
G50–G59	Adaptive control	Reserved for adaptive control requirements
G70	Inch programming	Mode for programming in inch units. It is recommended that control turn on establish this mode of operation

(continued)

(continued)

G71	Metric programming	Mode for programming in metric units. This mode is canalled by G70, M02, and M30
G72	Arc clockwise (3 dimensional)	An arc generated by the coordinated motion of 3 axes in which the curvature of the tool path with respect to the work piece is counterclockwise
G73	Arc counterclockwise (3 dimensional)	An arc generated by the coordinated motion of 3 axes in which the curvature of the tool path with respect to the work piece is counterclockwise
G72–G73	Circular interpolation (3 dimensional)	A mode of contouring control which uses the information contained in a single block to produce an arc on a sphere. The velocities of the axed use to generate this arc are varied by the control
G75	Multi-quadrant circular	Mode selection if required for multi-quadrant circular, canceled by G74
G80	Cancel fixed cycle	Command that will discontinue any of the fixed cycles G81–G89
G81	Fixed cycle	A preset series of operation which direct machine axis movement and/or cause spindle operation to complete such action as boring, drilling, tapping, or combinations thereof

Fixed cycle operation

Fixed cycle			At bottom		Movement out to feed start	Typical usage
Number	Code	Movement in	Dwell	Spindle		
1	G81	Feed	–	–	Rapid	Drill, spot drill
2	G82	Feed	Yes	–	Rapid	Drill, counter bore
3	G83	Intermittent	–	–	Rapid	Deep hole
4	G84	Spindle forward feed	–	Rev.	Feed	Tap
5	G85	Feed	–	–	Feed	Bore
6	G86	Start spindle, feed	–	Stop	Rapid	Bore
7	G87	Start spindle, feed	–	Stop	Manual	Bore
8	G88	Start spindle, feed	Yes	Stop	Manual	Bore
9	G89	Feed	Yes	–	Feed	Bore

G90	Absolute input	A control mode in which the data input is in the form of absolute dimensions
G91	Incremental input	A control mode in which the data input is in the form incremental data
G92	Preload of registers	Used to preload register to desired values. No machine operation is initiated. Examples would include preload of axis position registers, spindle speed constraints, initial radius, etc. information within this block shall conform to the character assignments of Table 5.1

(continued)

(continued)

G93	Inverse time feed rate	The data following the federate address is equal to the reciprocal of the time in minutes to execute the blocks and is equivalent to the velocity of any axis divided by the corresponding programmed increment
G94	Inches (millimeters) per minute feed rate	The federate code units are inches per minute of millimeters per minute
G94	Inches (millimeters) per revolution	The federate code units are inches (millimeters) per revolution of the spindle
G96	Constant surface speed per minute	The spindle speed code units are surface feet (meters) per minute and specify the tangential surface speed of the tool relative to the work piece. The spindle speed is automatically controlled to maintain the programmed value
G97	Revolution per minute	The spindle speed is defined by the spindle speed word
M00	Program stop	A miscellaneous function command to cancel the spindle and coolant functions and terminate further program execution after completion of other commands in the block
M01	Optional (planned) stop	A miscellaneous function command similar to a program stop except that the control ignores the command unless the operator has previously validated the command
M02	End of program	A miscellaneous function indicating completion of work piece. Stops spindle, coolant, and feed after completion of all commands in the block. Used to reset control and/or machine. Resetting control may include rewind of tape to the end of record character of progressing a loop tape through the spicing leader
M03	Spindle CW	Start spindle rotation to advance a right-handed screw into the work piece
M04	Spindle CCW	Start spindle rotation to retract a right-handed screw from the work piece
M07–M08–M09	Coolant, on, off	Mist (No. 2), flood (No. 1), tapping coolant of dust collector
M10–M11	Clamp, unclamp	Can pertain to machine slides, work piece, fixtures, spindle, etc.
M12	Synchronization code	An inhibiting code used for synchronization of multiple sets of axes
M15–M16	Motion +, Motion −	Rapid traverse of feed direction selection, where required
M19	Oriented spindle stop	A miscellaneous function that causes the spindle to stop at a predetermined of programmed angular position
M30	End of data	A miscellaneous function that stops spindle, coolant, and feed after completion of all commands in the block. Used to reset control and/or machine. Resetting control will include rewind of tape to the end of record character, progressing loop tape through the slicing leader, or transferring to a second tape reader

(continued)

(continued)

M31	Interlock by-pass	A command to temporarily circumvent a normally provided interlock
M47	Return to program start	A miscellaneous function that continues program execution from the start of program, unless inhibited by an interlock signal
M49	Override bypass	A function that deactivates a manual spindle or feed override and returns the parameter to the programmed value. Canceled by M48
M59	CSS bypass updating	A function that holds the RPM constant at its value when M59 is initiated/canceled by M58
M90–M99	Reserved for user	Miscellaneous function outputs that are reserved exclusively for the machine user

Additional commands may be required on specific machines. Unassigned code numbers should be used for these and specified on the format classification sheet. On certain machines, the functions described may not be completely applicable; deviations and interpretations should be clarified in the format classification sheet

5.4.3 Other Functions

Other combination of numbers and letters are used to designate various machine operations otherwise not covered by the description of G and M commands given here. For complete set of EIA 274D letters, numbers and symbols, refer to the standard.

5.4.4 Selected G and M Code Command Set

The augmented reality for the CNC machinery shall mimic all the programmable functions of the real machine tool. Real CNC machinery generally implements part of the requisite features targeted to a section of industry for which the machine tool is designed and built.

In order to demonstrate operation of the augmented reality for machinery using EIA 274 D, following G and M command set is selected. These are most common types of command used at G and M code-based CNC machinery.

Following G, M, and other codes are used to design a model interpreter in the following sections. Various possible formats for listing a command are used.

“G0” Point-to-point movement—positioning
“G00” Point-to-point movement—positioning
“G01” Linear interpolation
“G1” Linear interpolation
“G02” Circular interpolation (clockwise—2 dimensional)
“G2” Circular interpolation (clockwise—2 dimensional)
“G03” Circular interpolation (counterclockwise—2 dimensional)
“G3” Circular interpolation (counterclockwise—2 dimensional)

"G04"	Dwell
"G6"	Parabolic interpolation
"G06"	Parabolic interpolation
"G40"	Cutter compensation—offset
"G41"	Cutter compensation—cancel
"G42"	Cutter compensation—right
"G43"	Cutter compensation—inside corner
"G44"	Cutter offset—outside corner
"G70"	Inch programming
"G71"	Metric programming
"G90"	Absolute dimension input
"G91"	Incremental dimension input
"M00"	Program stop
"M02"	Stop/end of program
"M2"	Stop/end of program
"M3"	Spindle on—clockwise
"M03"	Spindle on—clockwise
"M4"	Spindle on—counterclockwise
"M04"	Spindle on—counterclockwise
"M5"	Spindle off
"M05"	Spindle off
"M06"	Tool change
"M07"	Coolant on—mist
"M08"	Coolant on—flood
"M09"	Coolant off
"M10"	Clamp
"M11"	Unclamp
"M13"	Spindle CW and coolant on
"M14"	Spindle CCW and coolant on
"%"	Program start character
"+"	Plus sign
"−"	Minus sign
"*F*"	Feed
"*I*"	Polar coordinate (*x*-axis)
"*J*"	Polar coordinate (*y*-axis)
"*N*"	Sequence number
"*S*"	Spindle speed
"*T*"	Tool
"*X*"	Movement in (+ve) *x*-direction
"*Y*"	Movement in (+ve) *y*-direction
"*Z*"	Movement in (+ve) *z*-direction

"*X*+" Movement in (+ve) *x*-direction
"*Y*+" Movement in (+ve) *y*-direction
"*Z*+" Movement in (+ve) *z*-direction
"*X*−" Movement in (−ve) *x*-direction
"*Y*−" Movement in (−ve) *y*-direction
"*Z*−" Movement in (−ve) *z*-direction

5.4.5 American Standard Code for Information Interchange (ASCII)

American standard codes for information interchange (ASCII) is the oldest and one of the most common methods of representing alphabets, symbols, and other characters. The set of ASCII code listed below is used to explain the operation of interpreter for G and M codes.

5.4.6 Unicode

Wikipedia defines unicode as the computing industry standard allowing computers to consistently represent and manipulate text expressed in most of the world's writing system. Developed in tandem with the universal characters set standard latest version of unicode consists of a repertoire of more than 107,000 characters covering 90 scripts, a set of code charts for visual reference, an encoding methodology and set of standard character encoding, an enumeration of character properties such as upper and lower case, a set of reference data computer files, and a number of related items, such as character properties, rules for normalization, decomposition, collation, rendering, and bidirectional display order (for the correct display of text containing both right-to-left scripts, such as Arabic or Hebrew, and left-to-right script, complete details of unicode are available at www.unicode.org.

5.5 Interpreter Design for CNC Operation

In computer science, an interpreter is a computer program that reads source code written in a high-level programming language, transforms the code to machine code, and executes the machine code. Using an interpreter, a single-source file can produce equal results even in vastly different systems. Using a complier, a single-source file can produce equal results only if it is complied with distinct, system-specific executables.

In CNC technology, interpreter is a software that translates English-like commands (as in the case of interactive commands or high-level language commands

Table 5.1 American standard code for information interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	‘
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	“	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	’	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP from feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	−	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x

(continued)

Table 5.1 (continued)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
25	19	031	EM (end of trans)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	>	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	<	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

used for programming a CNC machine tool) or mnemonics (as in the case of G and M code according to company standards or EIA-274-D standard) into binary signals to be sent through the I/O card or micro controller to the CNC machine tool actuators. It also has the capability to receive the signals from sensors and transducers mounted at the CNC machine tools as the feed back through the I/O card or micro controller and issue instructions as binary signals for machine tool to adapt to current situation.

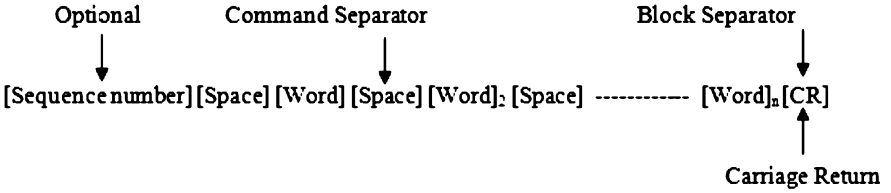
Interpreter translates a command and executes it contrary to a compiler that translates a whole sequence of instructions (program) and executes it afterward as an .exe file. Interpreters are more suitable for CNC implementation as these are comparatively simpler to design and build.

5.6 Interpreter Operation

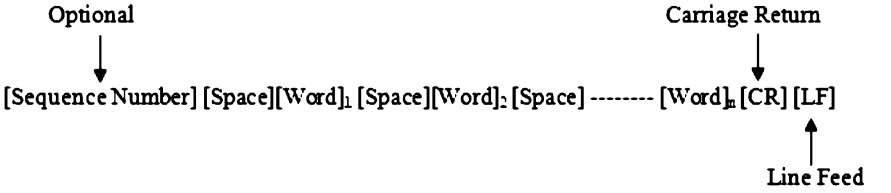
The EIA 274 D terminology describes the components of a part program as follows:

Letter X, Y, Z, I, J, K, L, G, M, S, T, F, S, etc.
Word X20, G70, M02
Block N35 G70 G90 M03

A block [such as: N10 G70 G90 M03 X4] is saved at the computer [controller] memory in ASCII format using one of the following methods:

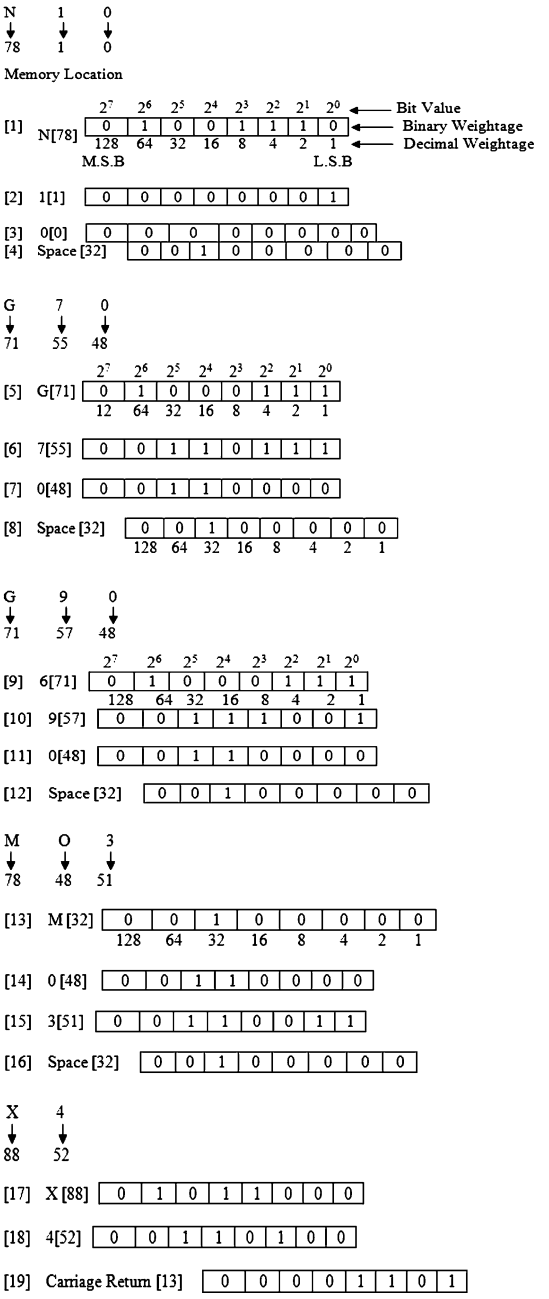


Or in the following alternate format:



Note that both [CR] and [LF] are non-printable character and used for identifying an end of block (EOB) condition. Each letter in the word is stored either as its binary equivalent or as its UNICODE equivalent. The following example from 3-axis milling machine part program takes ASCII designation for part program decoding into 8-bit binary code.

N10 G10 G90 M03 X4
Sequence Number—ASCII (Decimal)—Binary



A simple sequence of operation of an interpreter to execute above block shall be as given below:

```
        Counter=0
25      Counter=Counter + 1
        Read memory location [Counter]
Compare the content [character] with allowable characters
Found letter
Is it space?
        Yes
        Gosub Command Execution
Is it [CR]?
Yes
Goto 25
75      Compare the content [character] with allowable characters
Found letter
Yes
GoSub subroutine Command Formation
        Goto  25
Subroutine Command Formation
        Identify Letter
        Form command
        Previous character + New Character
      Is Expected number of Characters Complete?
        Yes
      Gosub Command Execution
        Return
Subroutine Command Execution
        Is It a G-Command?
        Yes
        Which G command is it – Compare with all allowable G-Command
        Set the requisite flag indicating effect of current G Command
        No
        Is it a M-Command?
        Yes
        Which M command is it – Compare with all allowable M-Command
        Execute requisite M – Command
        No
        Is it a F,S,T command?
        Execute respective function?
        No
        Is it an Axes motion command?
        Yes
        Which axis is it – X, Y or Z?
        Calculate Pulses for the Motor
        Send Pulses to the motor through I/O
        Return
```

The C# codes for some examples of axis motion at a 3-axis milling machines are listed in the following sections.

5.6.1 Rapid Movement

The rapid movement at a machine tool is used for positioning the tool at the work piece. Following software module describes the C# coding sample use to perform rapid movement of work piece in *x*-axis. The description of the code is given below:

Considering absolute coordinate systems, the letter *X* and its corresponding value are searched. This value is assigned to the variable Port. *X* value. The current position of the tool is compared with the numerical part of the Port. *X* value. If current value is greater than the assigned position of the tool in *X*-direction, the current position is subtracted from the assigned position. The result is passed on to port object's method Pulse Request *X* to generate number of pulse, for the stepper motor mounted in *X*-direction. Calling Port object's *X* plus method starts corresponding stepper motor and raises Position Arrive Method to control the 3D work piece position.

X Rapid

```
#region X in RapidMovement

if(lGMCode.StartsWith("X"))
{
/// GMCode first letter is X
if(Port.AbsoluteCoordinate)
{
/// take the value after X if there is X55 then takes only 55 and as-
sign in Port.Xvalue
Port.XValue = Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length -
1));
/// and now check the current position of x with new
/// value of x. this check will control the direction of
/// stepper motor in + or - direction. Suppose x current position is
/// 100 and new value of x is 150, its mean we move 50 mm in x plus
/// direction to reach 150 location.
if(Port.CurrentPositionX > Port.XValue)
{
///set the pulses of X stepper motor
///xcurrent position is 100 and new position will be
///150 we need only move 50 mm
///so minus the currentposition from new position
Port.PulsRequestX(Port.CurrentPositionX - Port.XValue);
/// send the pulses to stepper motor
Port.Xplus();
/// set the location of work piece of graphic of
/// milling machine
PositionArrived("X-",Port.CurrentPositionX - Port.XValue);
}
else
```

```
{
/// same as above but reverse direction
Port.PulsRequestX(Port.XValue - Port.CurrentPositionX);
Port.Xminus();
PositionArrived("X+",Port.XValue - Port.CurrentPositionX);
}
/// when we reach on 150 location of x
/// then the current position of x is 150
/// so we set the currentposition with new position of x
Port.CurrentPositionX = Port.XValue;
/// set the xValue of current position
StatusArrived(Current.X,Port.XValue.ToString());
}
else
{
/// Incremented Coordinate coding
}
}
#endregion
```

5.6.2 Linear Interpolation

EIA 274D allows direct movement between two points through linear interpolation. Following code details C# programming such as a move in the proxy virtual system. Code description is given below:

The following partial code is taken from CNC milling interpreter. In this code, it is first checked that the command start with *X* or *Y*. The values of *X* and *Y* are extracted and assigned to Port.*X*Value and Port.*Y*Value, respectively. These values are also assigned to Port.NextPosition*X* and Port.NextPosition*Y* representing specified find position in *x*- and *y*-axes. Afterward Port object's CalculateLinearInterpolation counts the pulses to reach at the required position and raises event AddLine and StatusArrive to cut interpolated line in work piece. Update, the method also with current position of *x*- and *y*-axes.

The values of *X* and *Y* with CurrentPosition*X* and CurrentPosition*Y* are evaluated to determine the forward or backward movement of work piece.

PositionArrived for both *x*- and *y*-axes to control the movement of work piece is called. Finally, The NextPosition*X* and NextPosition*Y* to Port's objects CurrentPosition*X* and CurrentPosition*Y* are assigned

Linear Interpolation

```
if(lGMCode.StartsWith("X"))
{/// GMCode first letter is X
if(Port.AbsoluteCoordinate)
{
        ///take the X Value
        Port.XValue = Con-
vert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
        /// we are now in Linear Interpolation so we need
        /// Y coordinate with X coordinate.
        /// check there is Y coordinates after X coordinate
        /// as the rules of EIA274D standard
        if(aList[i+1].ToString().StartsWith("Y"))
        { /// if we get Y coordinate after X coordinate
                i++;
                /// take the value of Y
                Port.YValue = Con-
vert.ToDouble(aList[i].ToString().Substring(1,aList[i].ToString().Len
gth - 1));
                Port.NextPositionX = Port.XValue;
                Port.NextPositionY = Port.YValue;

        Port.CalculateLinearInterpolation(Port.CurrentPositionX,Port.
CurrentPositionY,Port.NextPositionX,Port.NextPositionY);
                Fi-
nal.AddLine(Port.CurrentPositionX,Port.CurrentPositionY,Port.NextPosi
tionX,Port.NextPositionY);

                StatusArrived(Current.X,Port.XValue.ToString());
                StatusArrived(Current.Y,Port.YValue.ToString());
                if(Port.NextPositionX != Port.CurrentPositionX)
                {
                        if(Port.NextPositionX > Port.CurrentPositionX)
                                PositionArrived("X+",Port.NextPositionX
- Port.CurrentPositionX);
                        else
                                PositionArrived("X-
",Port.CurrentPositionX - Port.NextPositionX);
                }
                if(Port.NextPositionY != Port.CurrentPositionY)
                {
                        if(Port.NextPositionY > Port.CurrentPositionY)
                                PositionArrived("Y+",Port.NextPositionY
- Port.CurrentPositionY);
                        else
                                PositionArrived("Y-
",Port.CurrentPositionY  - Port.NextPositionY);
                }

                Port.CurrentPositionX = Port.NextPositionX;
                Port.CurrentPositionY = Port.NextPositionY;
        }

}
else
{
        /// Incremented Coordinate coding
}
```

5.6.3 Circular Interpolation

Clockwise or anticlockwise circular interpolation is facilitated in EIA 274 D. Following C# subroutine describes how to perform this function. The code description is given below:

The following partial code is taken from CNC milling interpreter. In this code, the interpreter first checks that commands start with *X* and *Y*, and corresponding values of *I* and *J* are given the values of *X*, *Y*, *I*, and *J* are extracted and assign to Port.*X*Value, Port.*Y*Value, Port.*I*, and Port.*J*, respectively, and *X* and *Y* values are assigned to Port.NextPosition*X* and Port.NextPosition*Y* to reach specified position in *x*- and *y*-axes and calculate the radius of the circular interpolation.

Afterward Port object's CaluculateCircularInterpolation to count the pulses to reach in required position is called. The event StatusArrive to update the user interface with current position of *x*- and *y*-axes is initiated.

The values of *X* and *Y* with CurrentPosition*X* and CurrentPosition*Y* are evaluated to determine the movement in forward or backward movement of work piece.

PositionArrived for both *x*- and *y*-axes to control the movement of work piece is called. Finally, assign the NextPosition*X* and NextPosition*Y* to Port's objects CurrentPosition*X* and CurrentPosition*Y* are assigned.

Circular Interpolation

```
if(lGMCode.StartsWith("X") && aList[i+1].ToString().StartsWith("Y"))
{
        if(aList[i+2].ToString().StartsWith("I") &&
aList[i+3].ToString().StartsWith("J"))
        {
                Port.XValue = Con-
vert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                Port.YValue = Con-
vert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToString()
.Length - 1));
                Port.I = Con-
vert.ToDouble(aList[i+2].ToString().Substring(1,aList[i+2].ToString()
.Length - 1));
                Port.J = Con-
vert.ToDouble(aList[i+3].ToString().Substring(1,aList[i+3].ToString()
.Length - 1));
                i = i + 3;

        }
        else
        {
                Port.XValue = Con-
vert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                Port.YValue = Con-
vert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToString()
.Length - 1));
                i = i + 1;
                if(Port.I > 0 || Port.J > 0)
                {
                        Port.I = Port.I;
                        Port.J = Port.J;
```

```
                }
                else
                {
                        Port.I = 0;
                        Port.J = 0;
                }
        }

        Port.NextPositionY = Port.YValue;
        Port.NextPositionX = Port.XValue;
        Port.Radius = Math.Sqrt(((Port.CurrentPositionX - Port.I) *
(Port.CurrentPositionX - Port.I)) + ((Port.CurrentPositionY - Port.J)
* (Port.CurrentPositionY - Port.J)));
        Port.CalculateCircularInterpolation(Final);
        StatusArrived(Current.X,Port.XValue.ToString());
        StatusArrived(Current.Y,Port.YValue.ToString());
        if(Port.NextPositionX != Port.CurrentPositionX)
        {
                if(Port.NextPositionX > Port.CurrentPositionX)
                        PositionArrived("X+",Port.NextPositionX -
Port.CurrentPositionX);
                else
                        PositionArrived("X-",Port.CurrentPositionX -
Port.NextPositionX);
        }
        if(Port.NextPositionY != Port.CurrentPositionY)
        {
                if(Port.NextPositionY > Port.CurrentPositionY)
                        PositionArrived("Y+",Port.NextPositionY -
Port.CurrentPositionY);
                else
                        PositionArrived("Y-",Port.CurrentPositionY -
Port.NextPositionY);
        }
        Port.CurrentPositionX = Port.NextPositionX;
        Port.CurrentPositionY = Port.NextPositionY;
}
```

5.6.4 Parabolic Interpolation

Parabolic interpolation is occasionally found at the CNC machine tools. It provides an added feature for profiling. A description of parabolic interpolation using C# code is given below.

The following partial code is taken from CNC milling interpreter. In this code, interpreter first checks that commands starting with *X* and *Y*, *I* and *J*; then extract the values of *X*, *Y*, *I*, and *J* and assign those to Port.*X*Value, Port.*Y*Value, Port.*I*, and Port.*J*, respectively, and also assign *X* and *Y* values to Port.NextPosition*X* and Port.NextPosition*Y* to reach specified position in *x*- and *y*-axes and calculate the parameters for the Parabolic Interpolation.

Port object's CaluculateParabolicInterpolation is called to count the pulses to reach in required position and initiate event StatusArrive to update the user interface for current position of *x*- and *y*-axes.

The values of X and Y with CurrentPosition$X$ and CurrentPosition$Y$ are evaluated to determine the movement in forward or backward direction of work piece.

Parabolic Interpolation

```
if(lGMCode.StartsWith("X") && aList[i+1].ToString().StartsWith("Y"))
{
        if(aList[i+2].ToString().StartsWith("I") &&
aList[i+3].ToString().StartsWith("J"))
        {
                Port.XValue = Con-
vert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                Port.YValue = Con-
vert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToString()
.Length - 1));
                Port.I = Con-
vert.ToDouble(aList[i+2].ToString().Substring(1,aList[i+2].ToString()
.Length - 1));
                Port.J = Con-
vert.ToDouble(aList[i+3].ToString().Substring(1,aList[i+3].ToString()
.Length - 1));
                i = i + 3;

        }
        else
        {
                Port.XValue = Con-
vert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                Port.YValue = Con-
vert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToString()
.Length - 1));
                i = i + 1;
                if(Port.I > 0 || Port.J > 0)
                {
                        Port.I = Port.I;
                        Port.J = Port.J;
                }
                else
                {
                        Port.I = 0;
                        Port.J = 0;
                }
        }

        Port.NextPositionY = Port.YValue;
        Port.NextPositionX = Port.XValue;
        Port.Radius = Math.Sqrt(((Port.CurrentPositionX - Port.I) *
(Port.CurrentPositionX - Port.I)) + ((Port.CurrentPositionY - Port.J)
* (Port.CurrentPositionY - Port.J)));
        Port.CalculateParabolicInterpolation(Final);
        StatusArrived(Current.X,Port.XValue.ToString());
        StatusArrived(Current.Y,Port.YValue.ToString());
        if(Port.NextPositionX != Port.CurrentPositionX)
        {
                if(Port.NextPositionX > Port.CurrentPositionX)
                        PositionArrived("X+",Port.NextPositionX -
Port.CurrentPositionX);
                else
                        PositionArrived("X-",Port.CurrentPositionX -
```

```
Port.NextPositionX);
        }
        if(Port.NextPositionY != Port.CurrentPositionY)
        {
                if(Port.NextPositionY > Port.CurrentPositionY)
                        PositionArrived("Y+",Port.NextPositionY -
Port.CurrentPositionY);
                else
                        PositionArrived("Y-",Port.CurrentPositionY -
Port.NextPositionY);
        }
        Port.CurrentPositionX = Port.NextPositionX;
        Port.CurrentPositionY = Port.NextPositionY;
}
```

PositionArrived for both *x*- and *y*-axes is monitored to control the movement of work piece, and finally the NextPosition*X* and NextPosition*Y* are assigned to Port's objects CurrentPosition*X* and CurrentPosition *Y*.

5.7 A Description of Development of AR for Metal-Cutting Machines

Use case diagrams, sequence diagram and UML static diagrams related to CNC milling are presented as Figs. 5.3, 5.4 and 5.5a detailed flow chart representing the operation of AR for CNC milling is listed as Fig. 5.6. The augmented reality for milling is shown in Fig. 5.7. Similar diagrams, Figs. 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18 and 5.19 provide details about CNC turning, CNC drilling and CNC sawing machines. Because of the similarity, the flowcharts for turning, drilling, and sawing are not provided. The AR for all the selected metal-cutting machines described above can be worked upon dynamically using VMS training, VMS Design, VMS Planner, VMS Monitor, and VMS Fault Diagnostic.

Sections 5.7.1–5.7.4 describes the development of AR for metal-cutting operations considered.

5.7.1 Developing AR for CNC Milling Operation

In order to build an augmented reality of a CNC milling machine, all features of the controller at the machine tool need to be simulated. For an example, refer to the block diagram depicted in Fig. 5.2.

The process starts with the development of use case diagram and sequence diagram. Figures 5.3 and 5.4 details use case diagram and sequence diagram for milling operation, respectively. Four classes namely frmMain, support, port access, and form final are defined. A small selection from EIA 274 D G and M code as listed in an earlier section is used. The next step comprises definition of classes using UML static diagram. Figure 5.5 lists all the classes used for the CNC milling operation. CNC milling software consists of three projects.

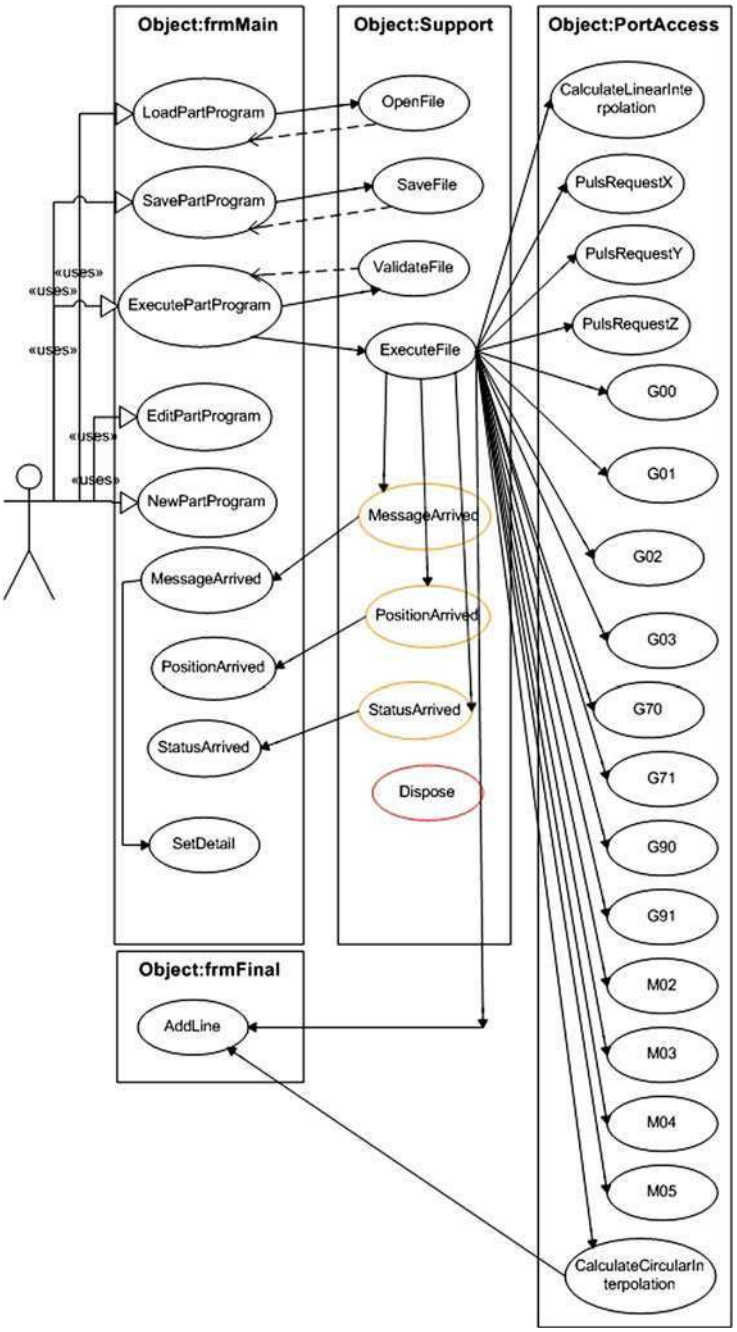


Fig. 5.3 CNC milling UML case diagram

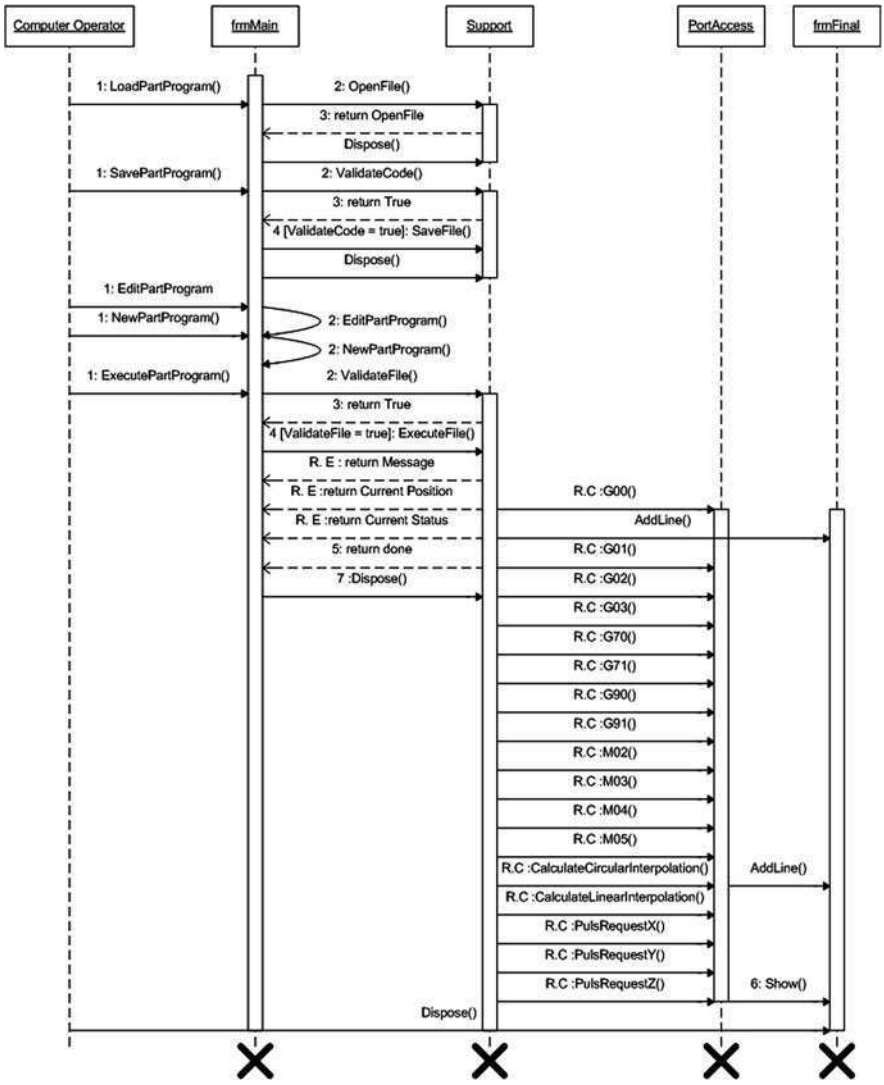


Fig. 5.4 CNC milling UML sequence diagram

1. Milling: Consists of classes used in CNC milling software such as port access and milling, two enumerations and five events. Milling library is a dynamic link library (DLL)-based project. The produced DLL of milling library will be in use in CNC milling application to provide separation between application and logic, and logic encapsulation.
   Class structure of milling: Milling is consists of two main classes milling class and MillingPortAccess class. Milling class has encapsulated all the main functionality required by CNC milling software such as Open File, Save File,

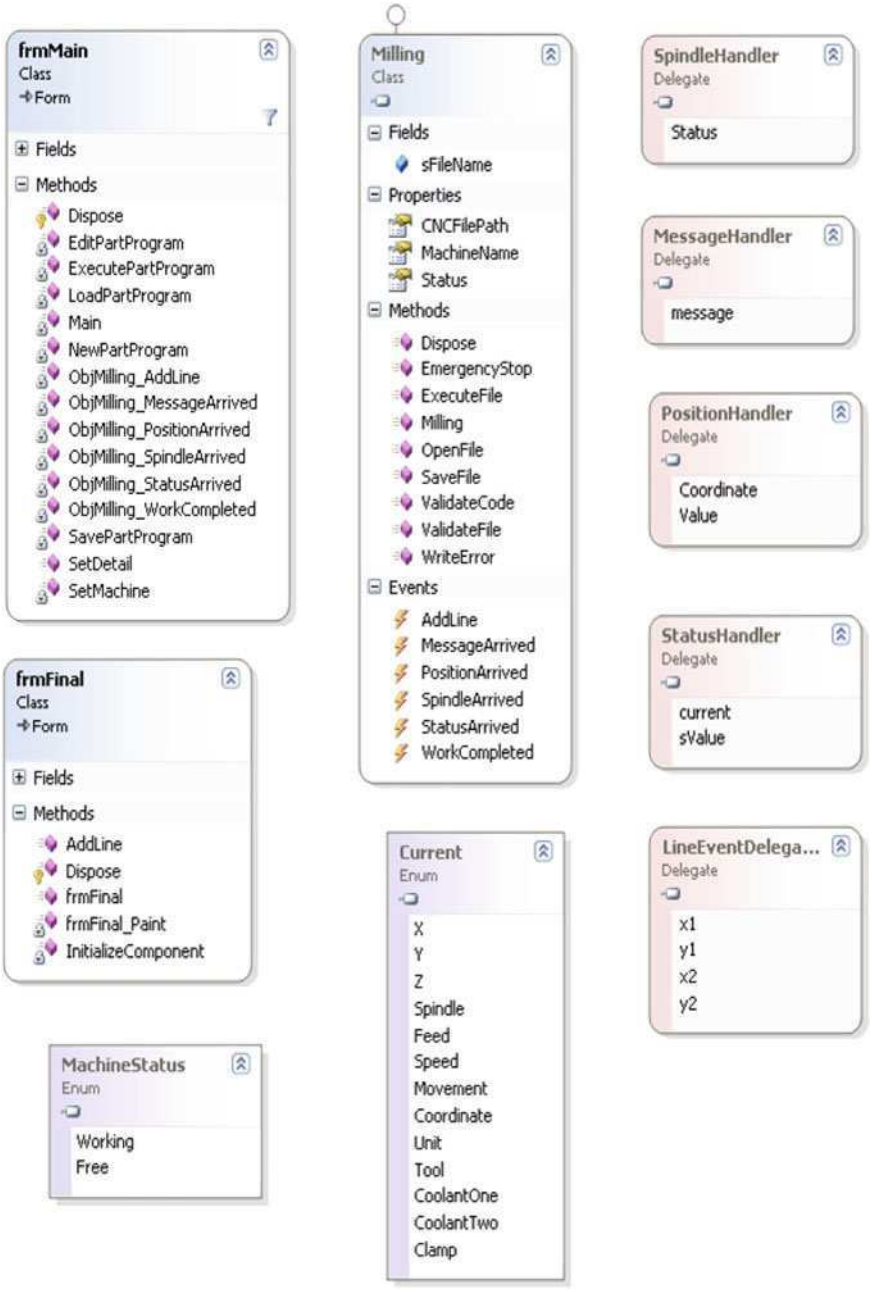


Fig. 5.5 CNC milling UML static diagram (continued)

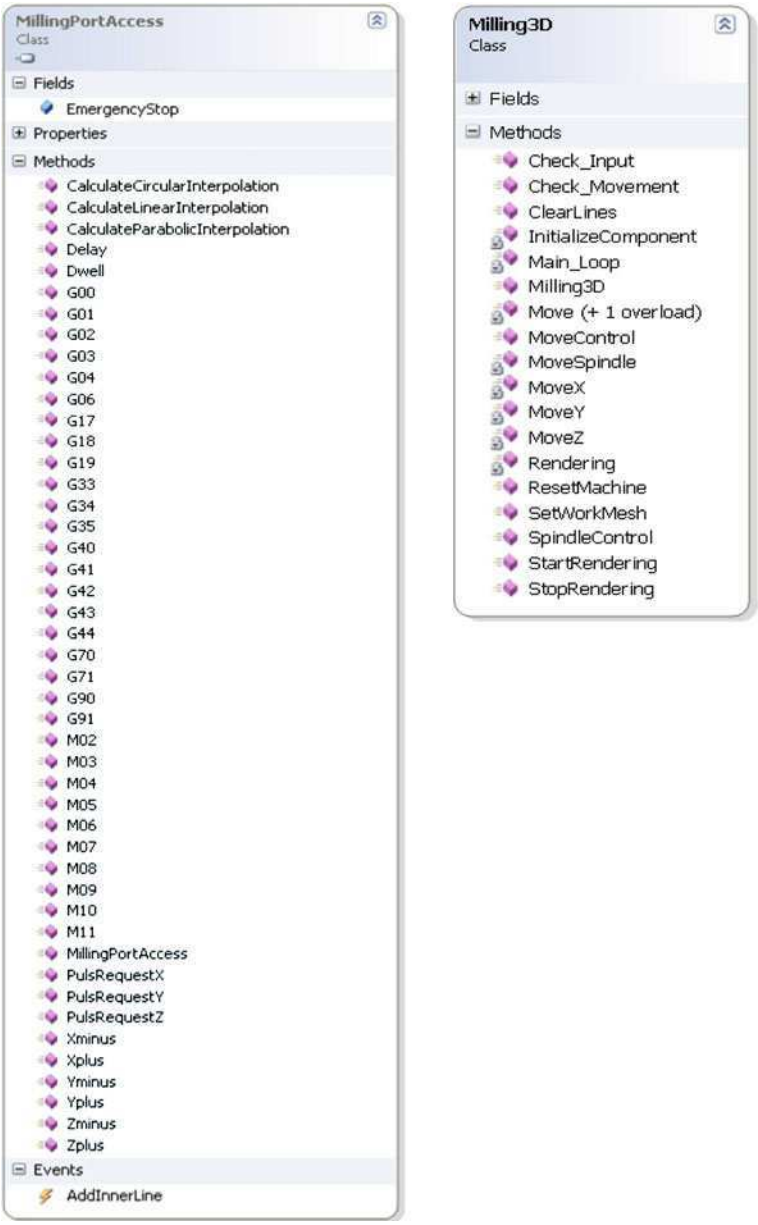


Fig. 5.5 (continued)

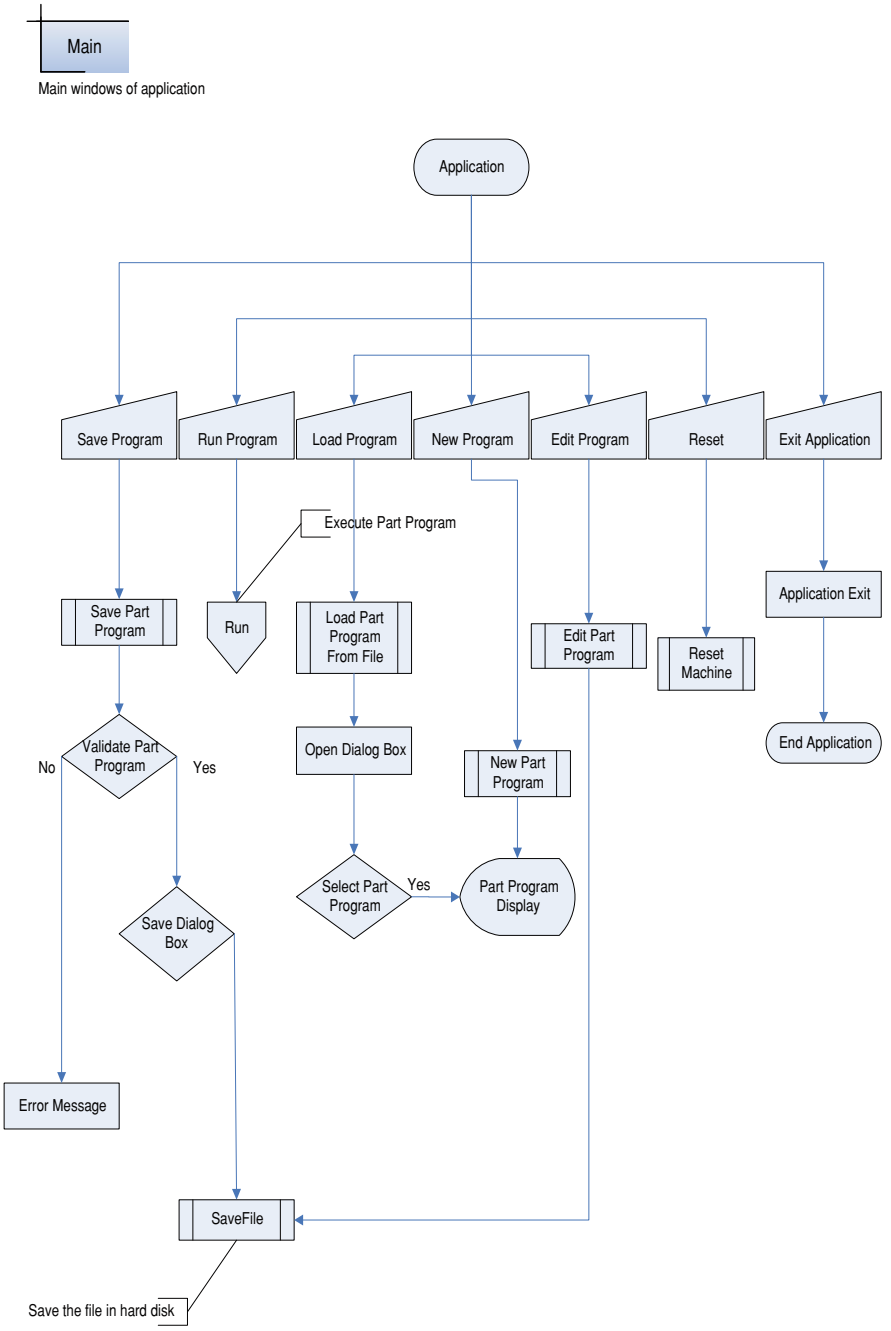


Fig. 5.6a Functions in CNC interpreter software (continued)

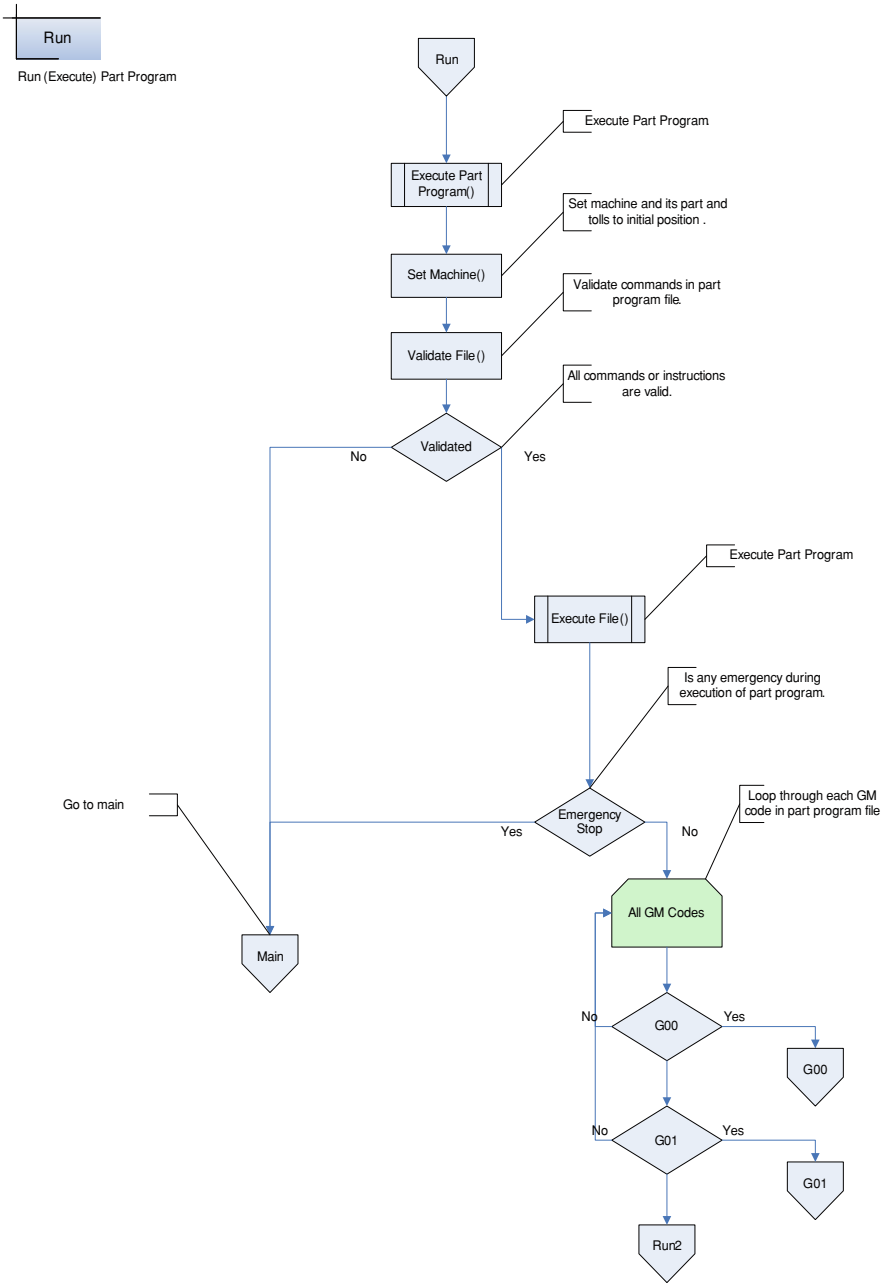


Fig. 5.6b (continued)

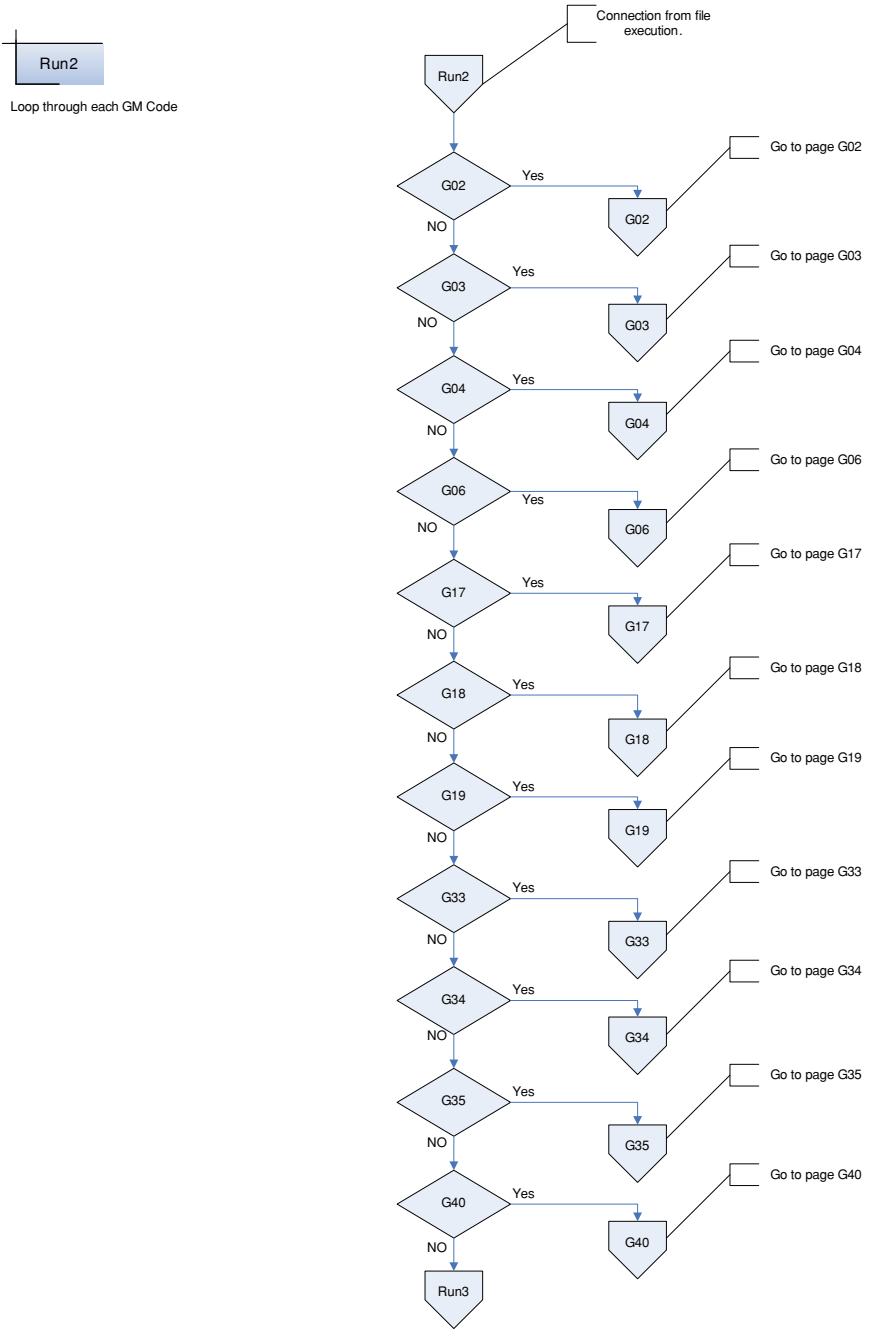


Fig. 5.6c (continued)

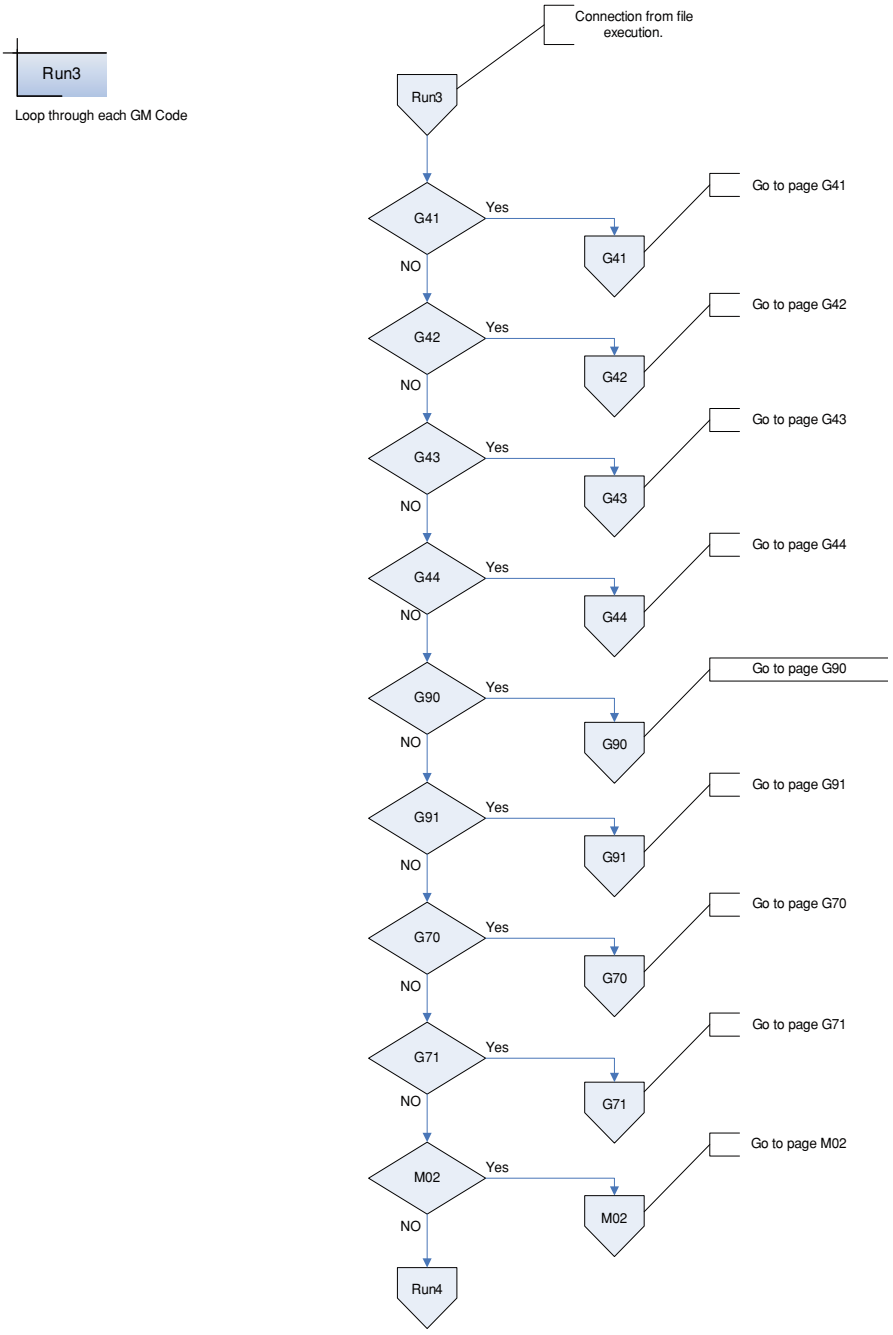


Fig. 5.6d (continued)

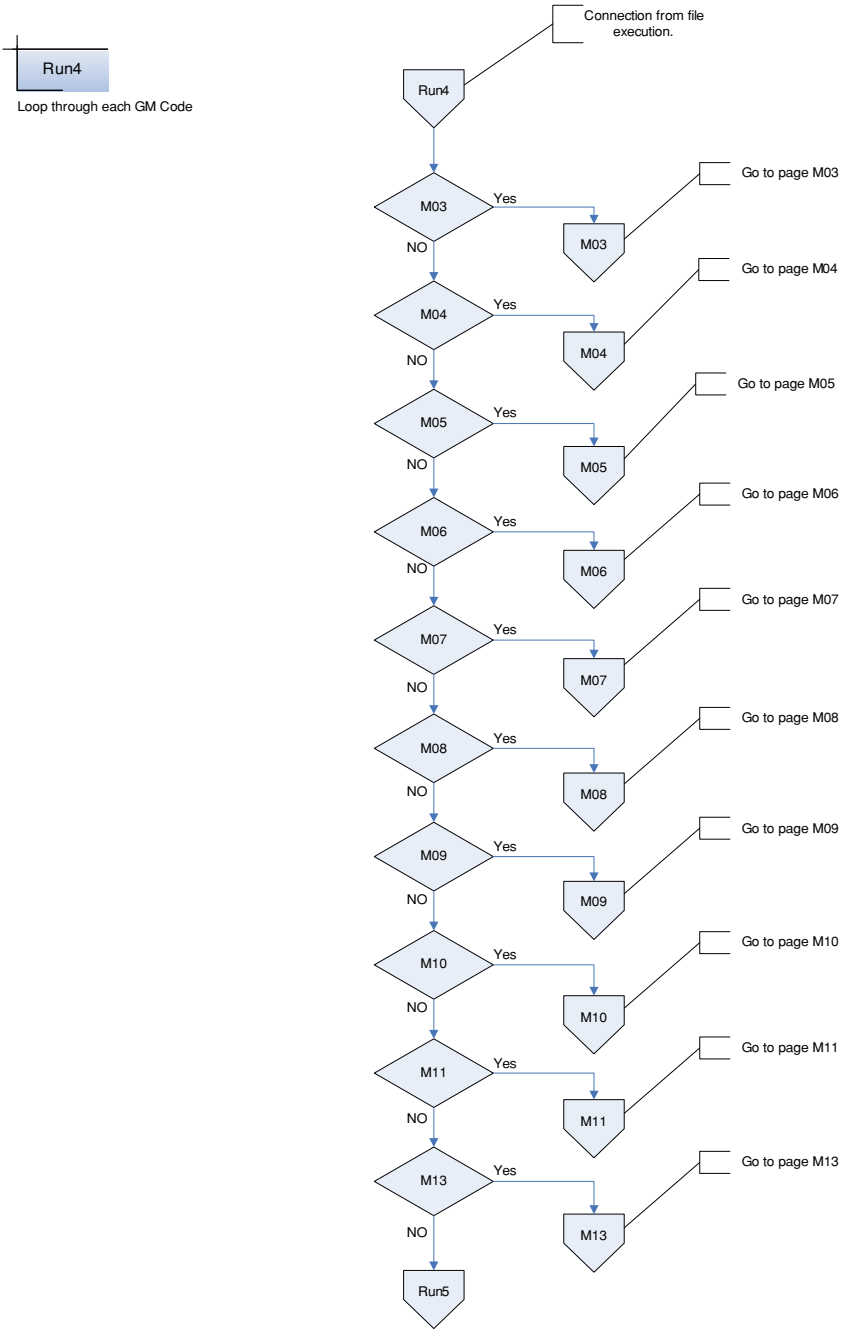


Fig. 5.6e (continued)

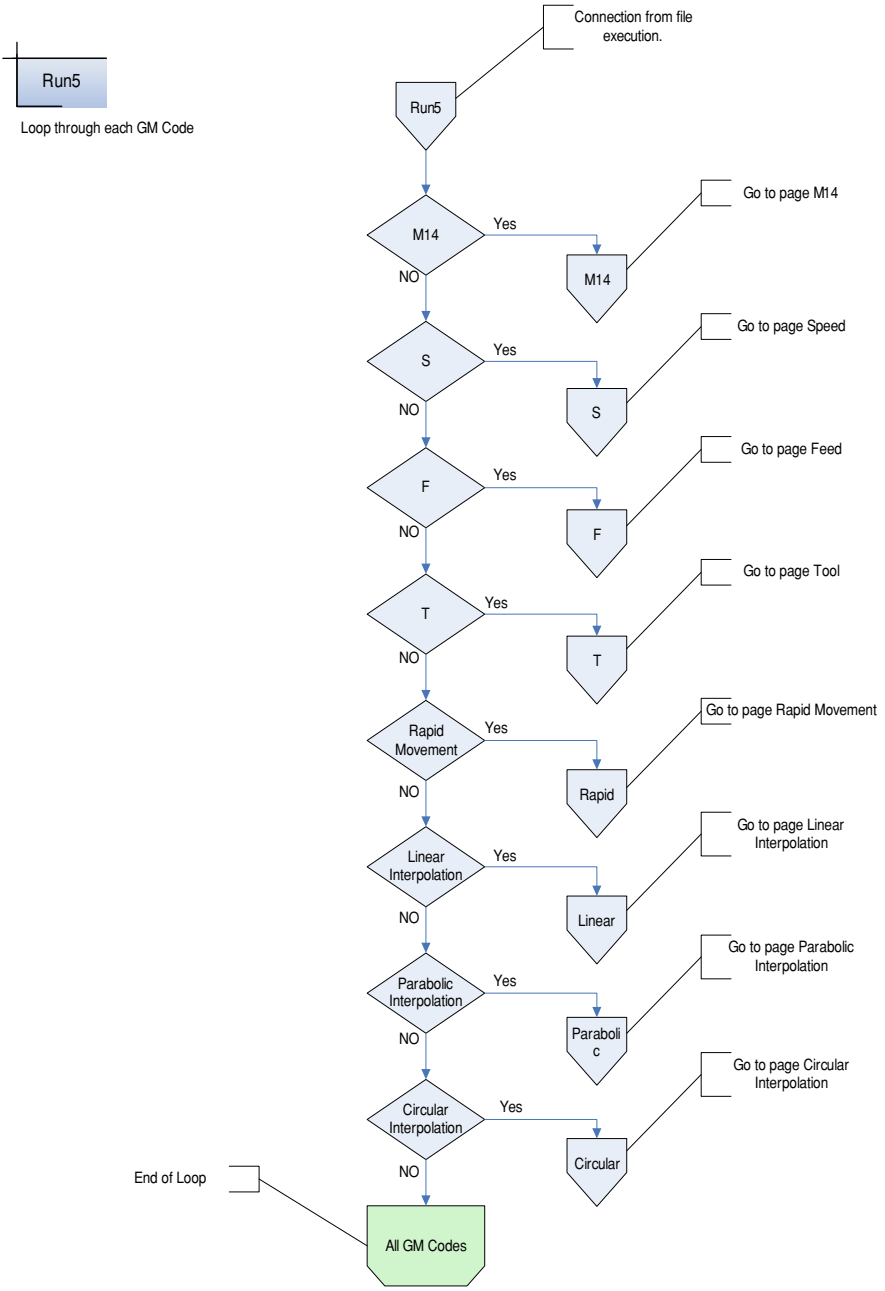


Fig. 5.6f (continued)

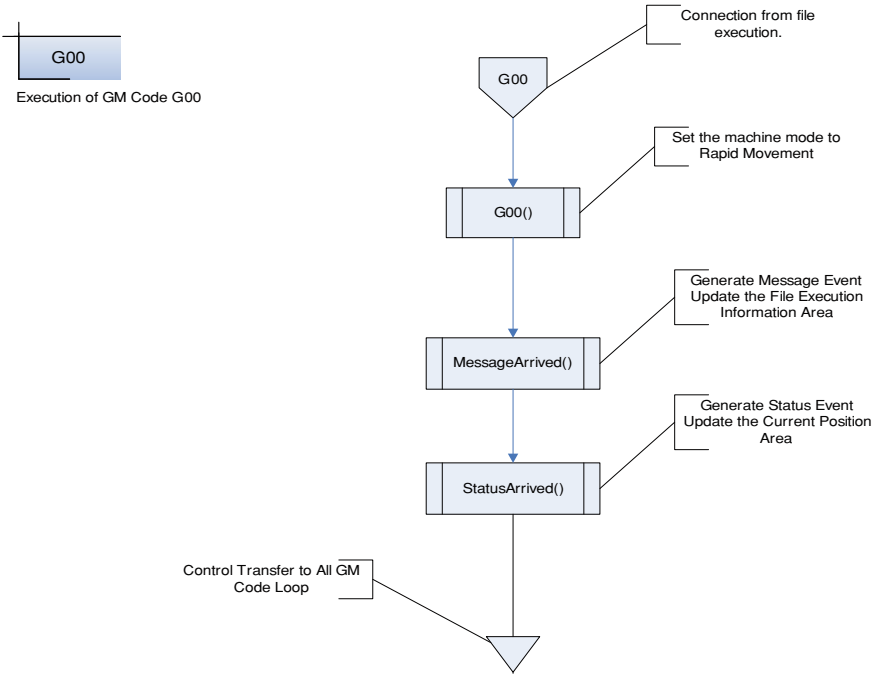


Fig. 5.6g Rapid traverse function (continued)

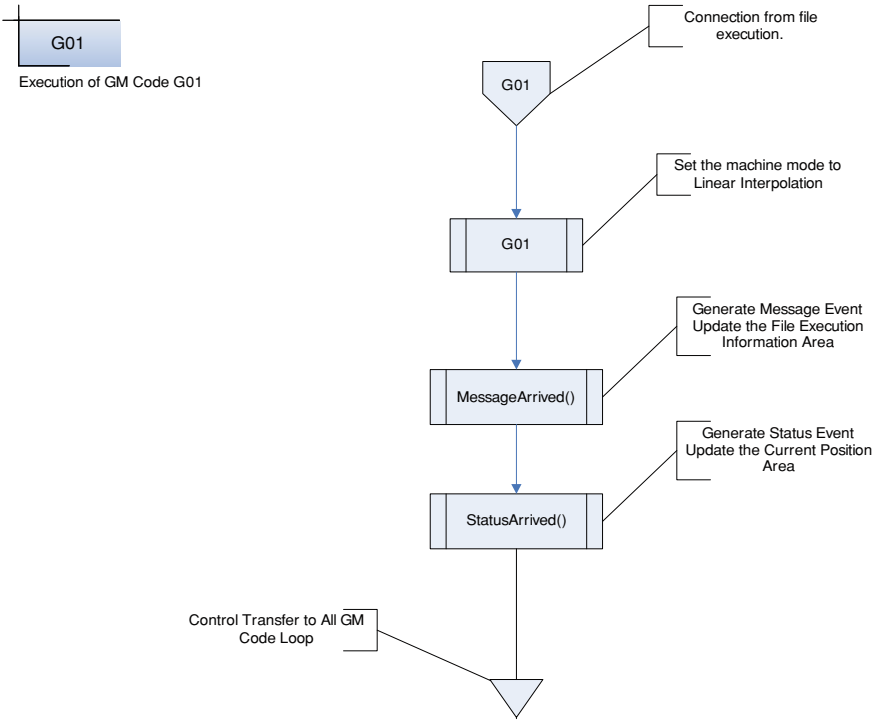


Fig. 5.6h Linear interpolation function (continued)

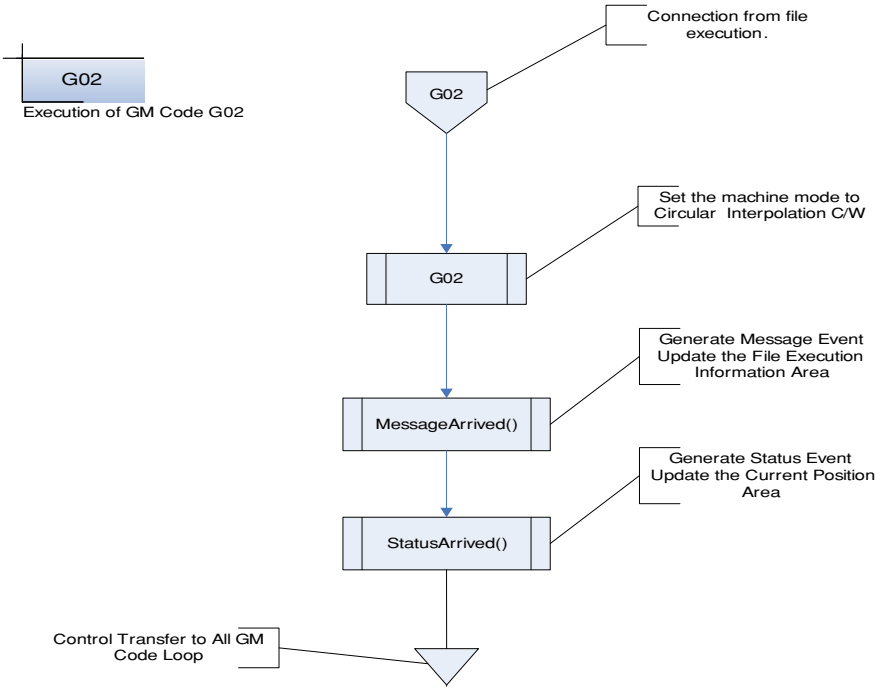


Fig. 5.6i Circular interpolation C/W function (continued)

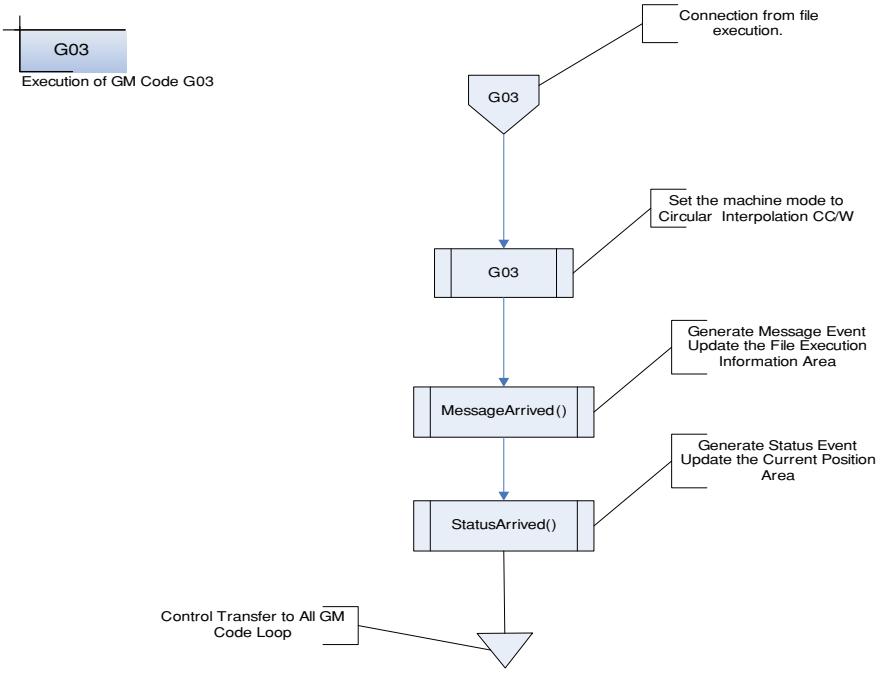


Fig. 5.6j Circular interpolation CC/W function (continued)

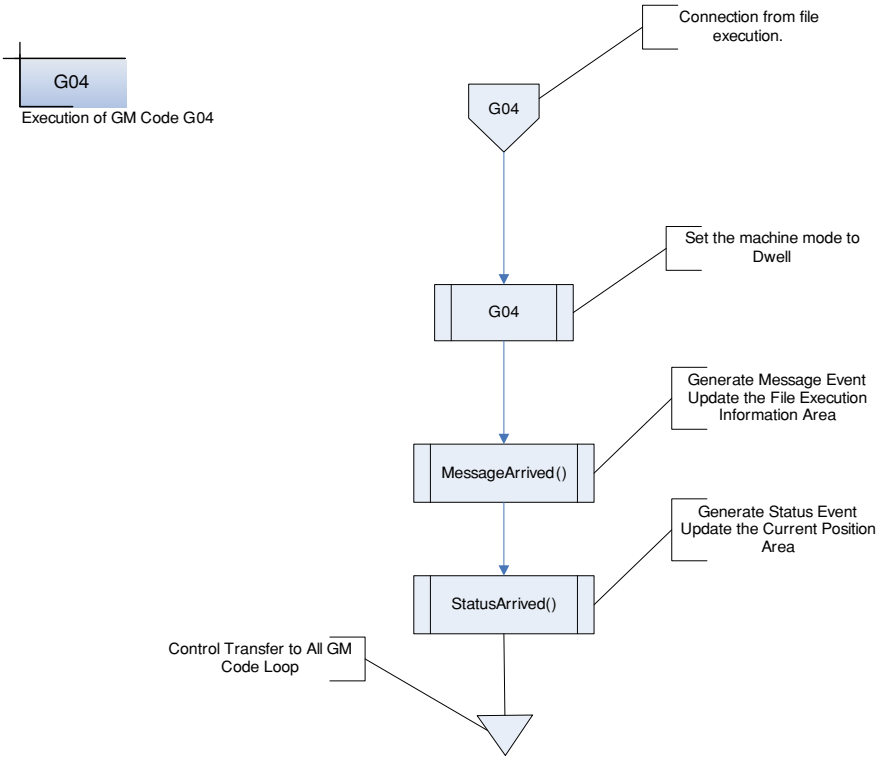


Fig. 5.6k Set mode dwell function (continued)

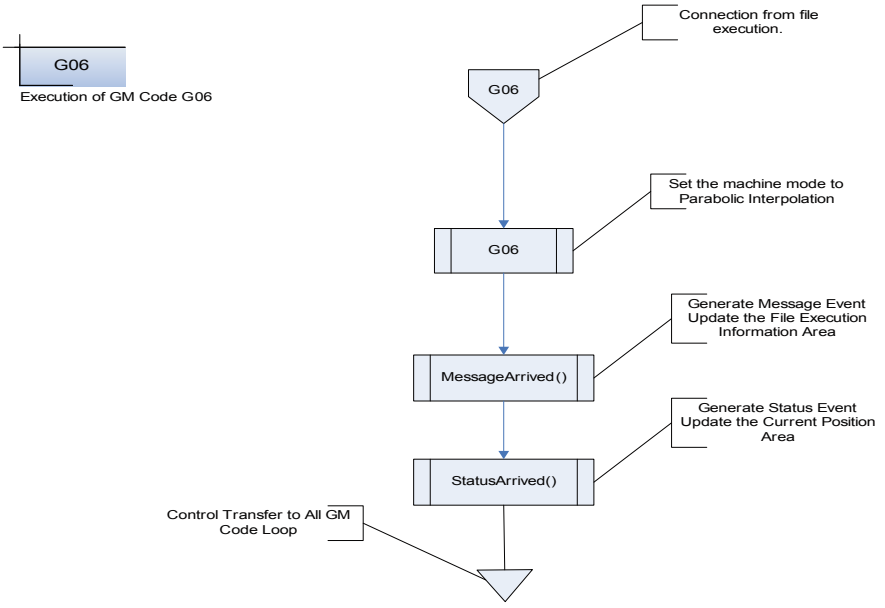


Fig. 5.6l Parabolic interpolation function (continued)

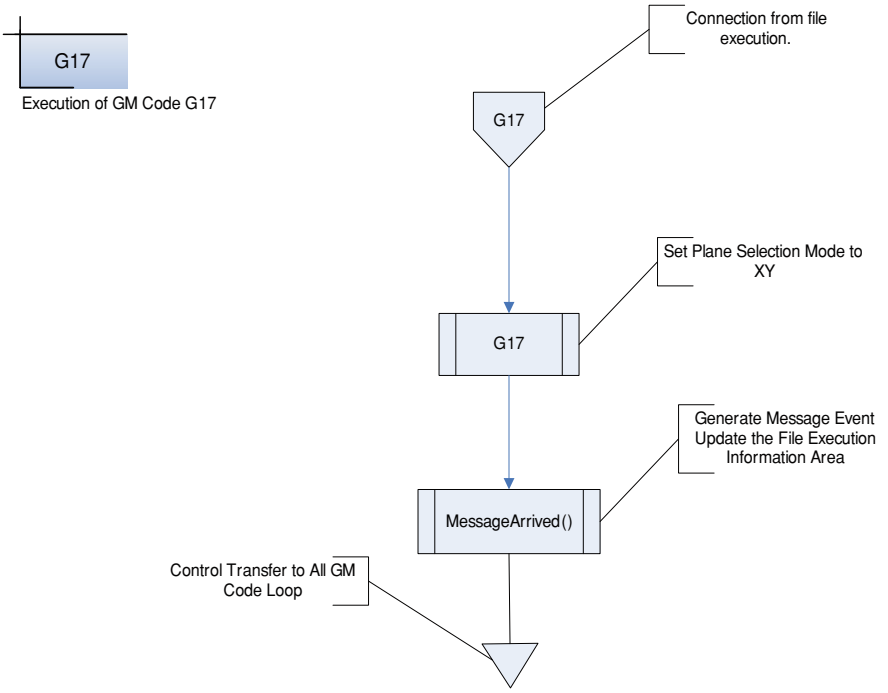


Fig. 5.6m Set plane selection mode *XY* function (continued)

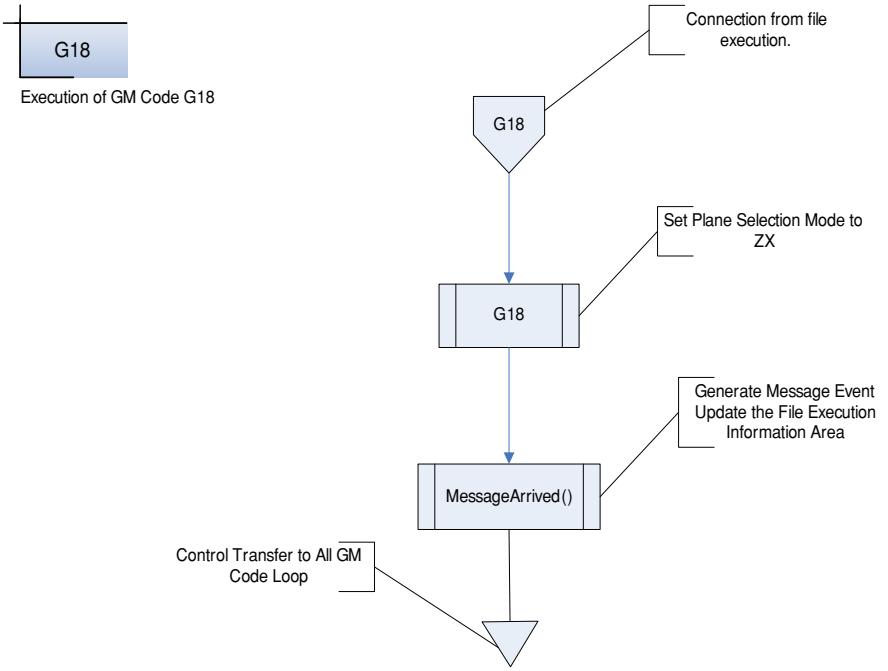


Fig. 5.6n Set plane selection mode *ZX* function (continued)

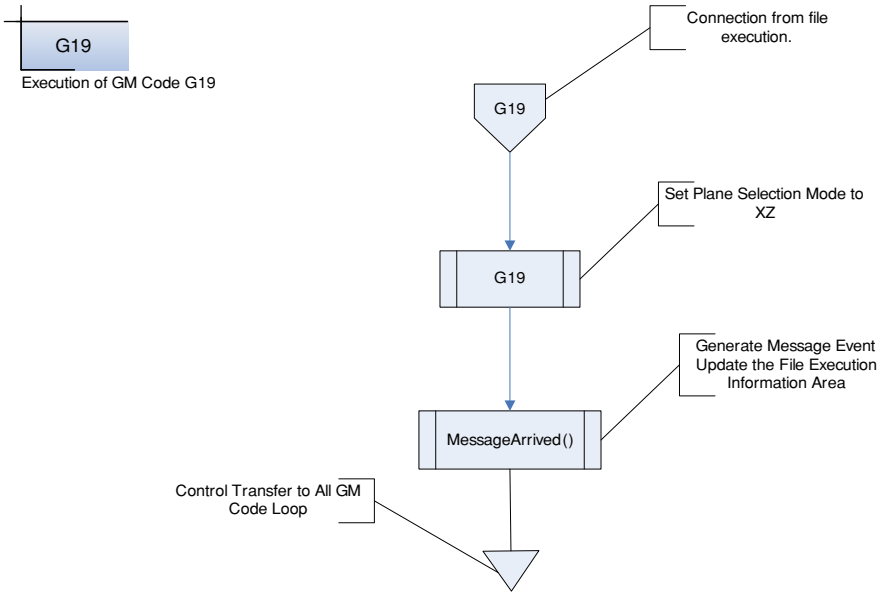


Fig. 5.6o Set plane selection mode *XZ* function

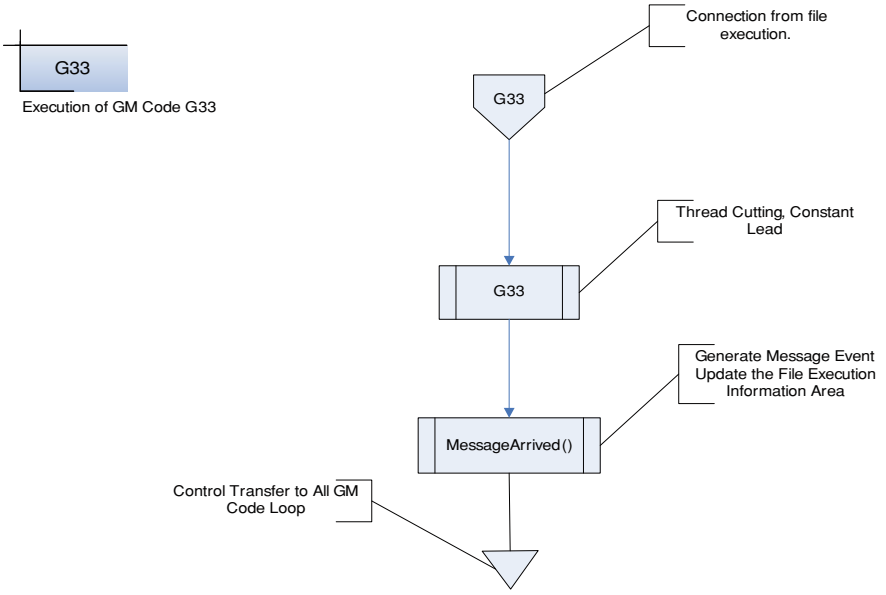


Fig. 5.6p Thread cutting, constant lead function

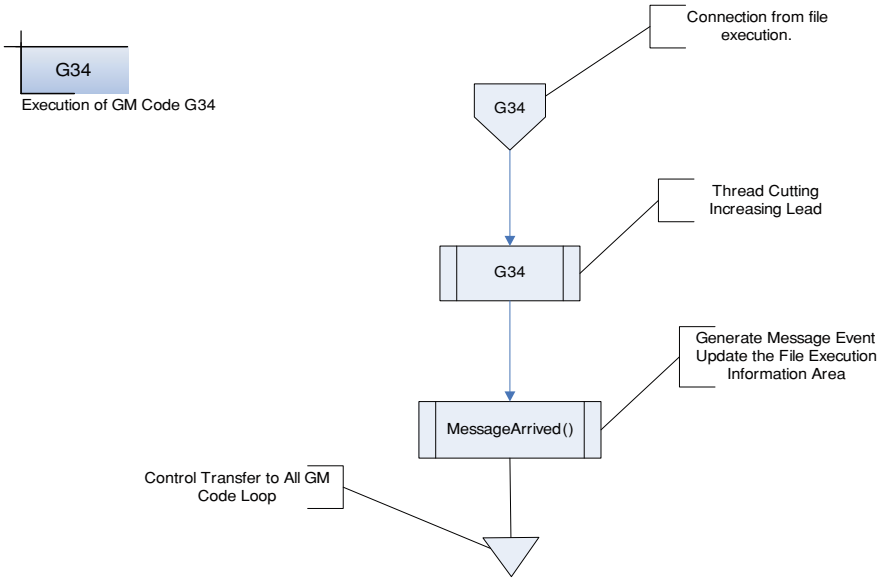


Fig. 5.6q Thread cutting, increasing lead function

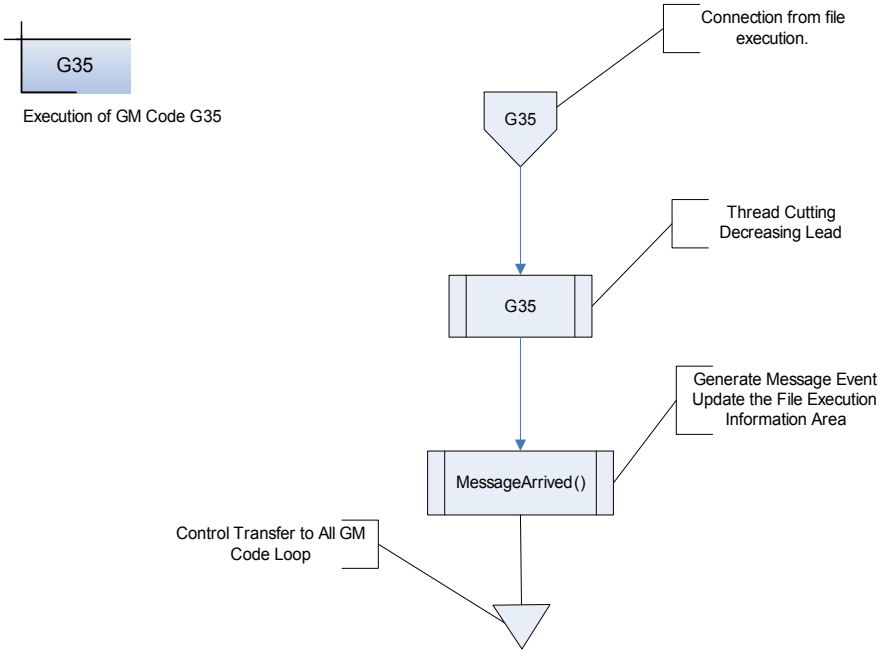


Fig. 5.6r Thread cutting, decreasing lead function

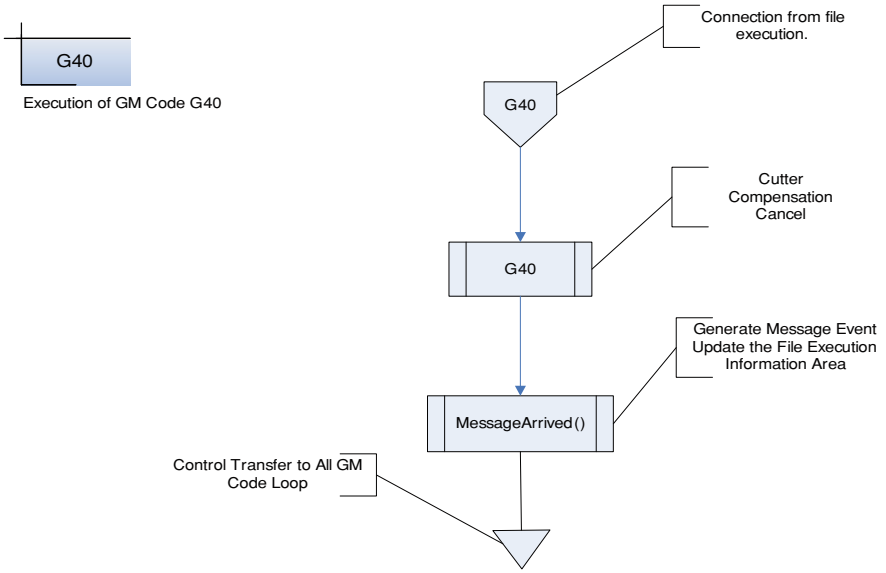


Fig. 5.6s Cutter compensation cancel function (continued)

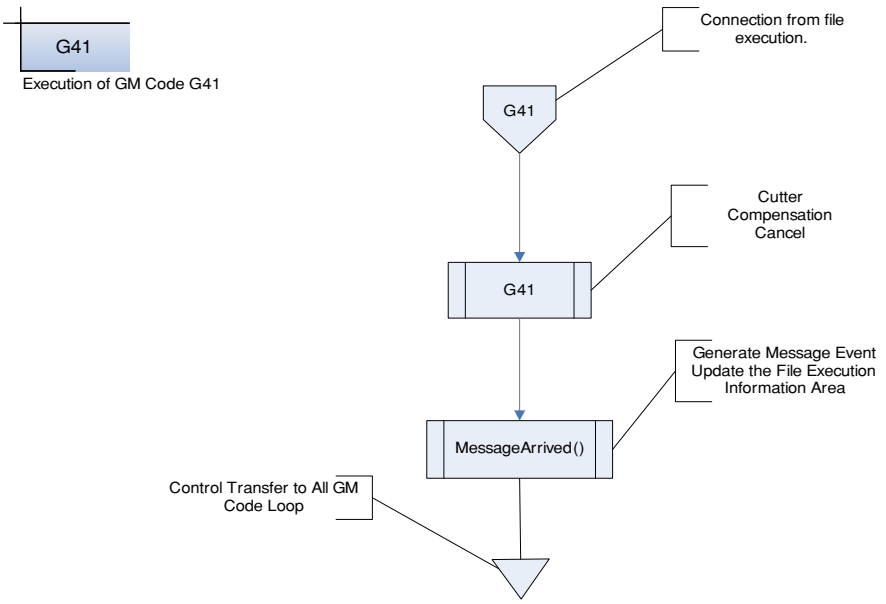


Fig. 5.6t Cutter compensation cancel function (continued)

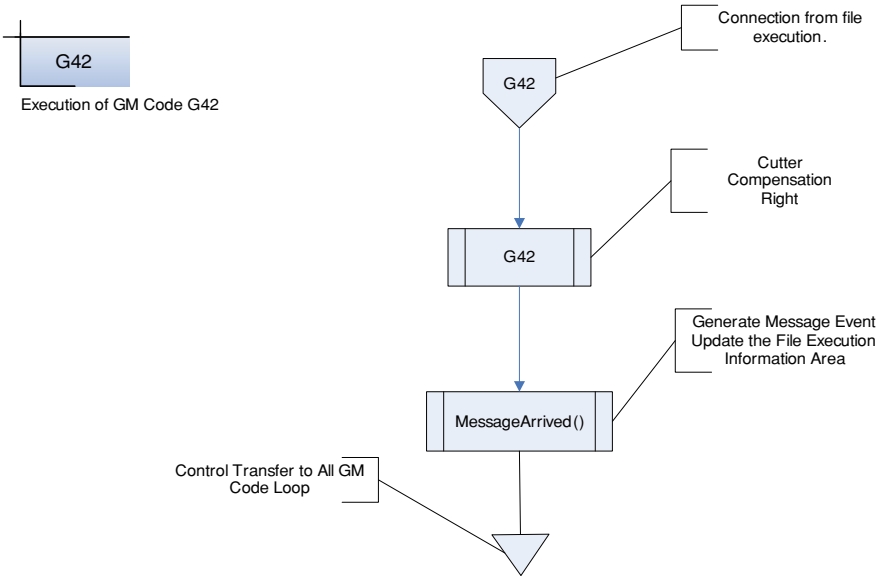


Fig. 5.6u Cutter compensation right function (continued)

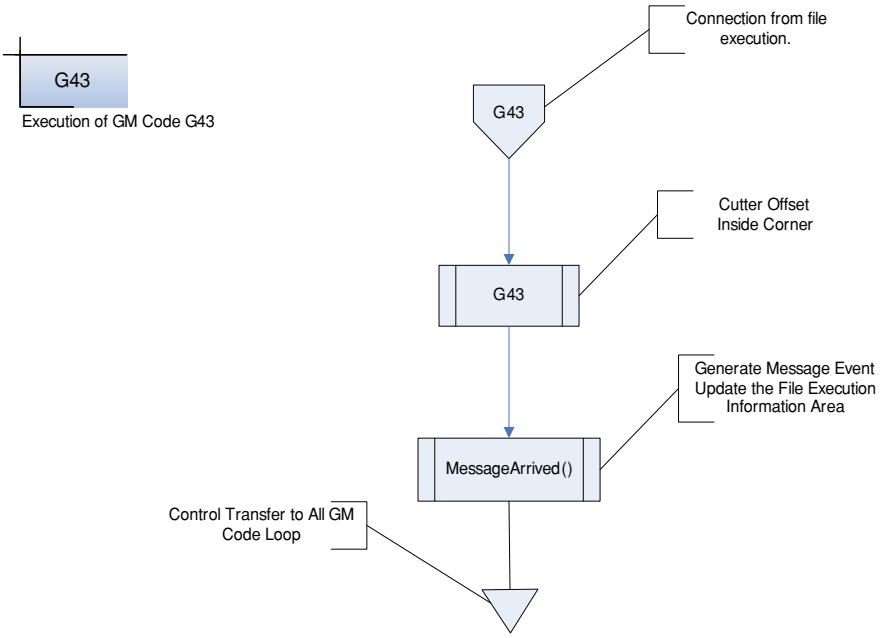


Fig. 5.6v Cutter offset inside corner function (continued)

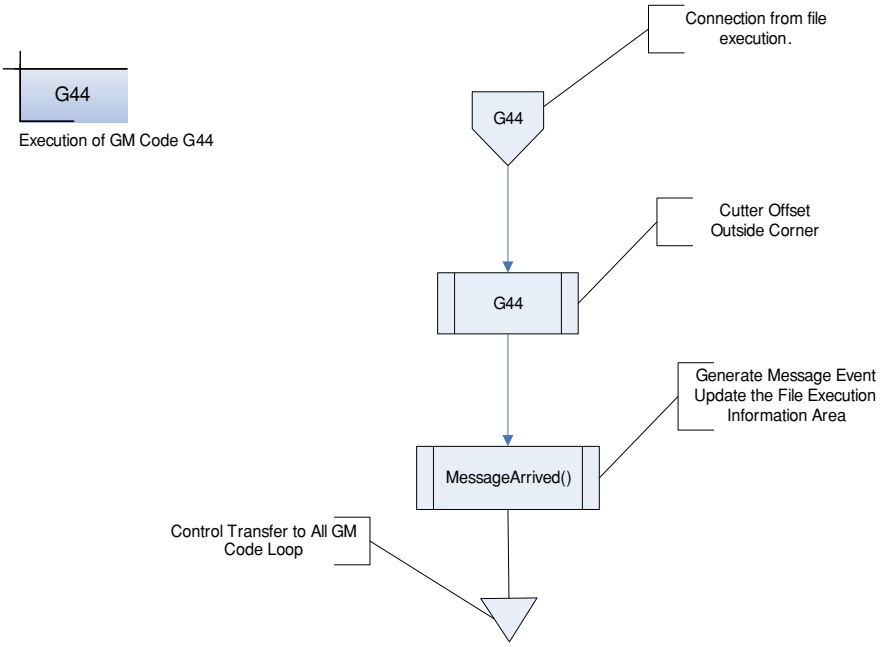


Fig. 5.6w Cutter offset outside corner function (continued)

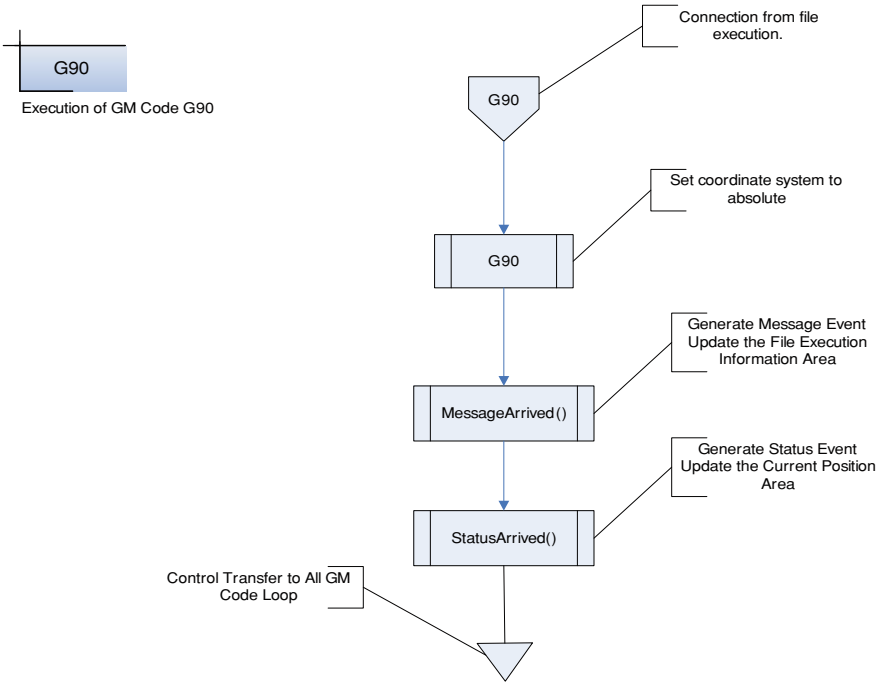


Fig. 5.6x Set coordinate system to absolute function (continued)

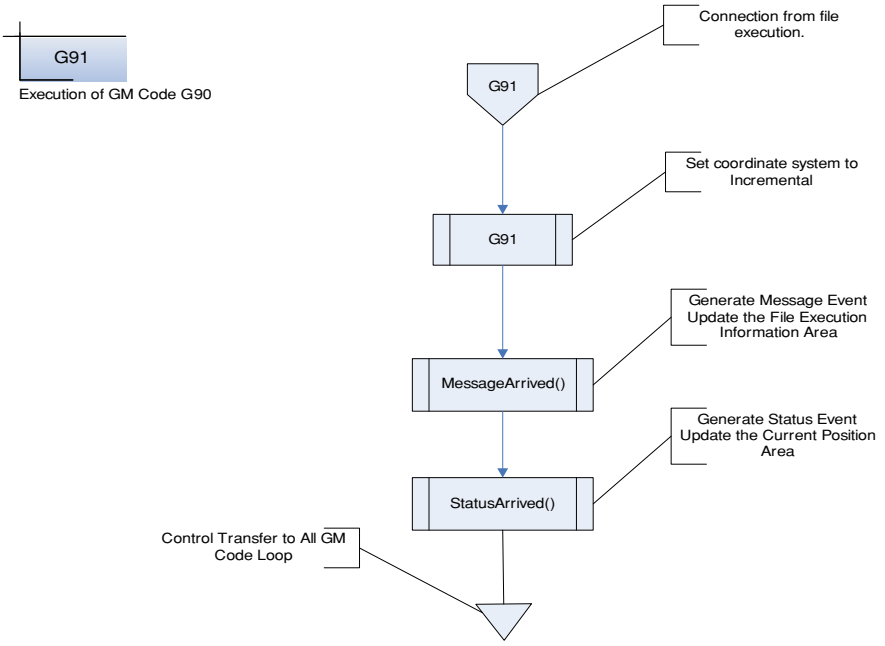


Fig. 5.6y Set coordinate system to incremental function (continued)

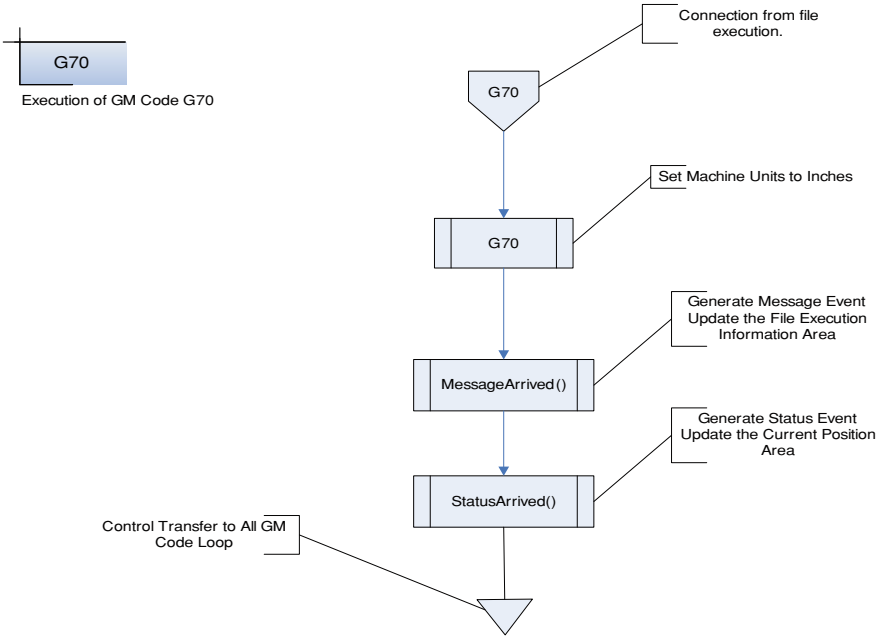


Fig. 5.6z Set units to inches function (continued)

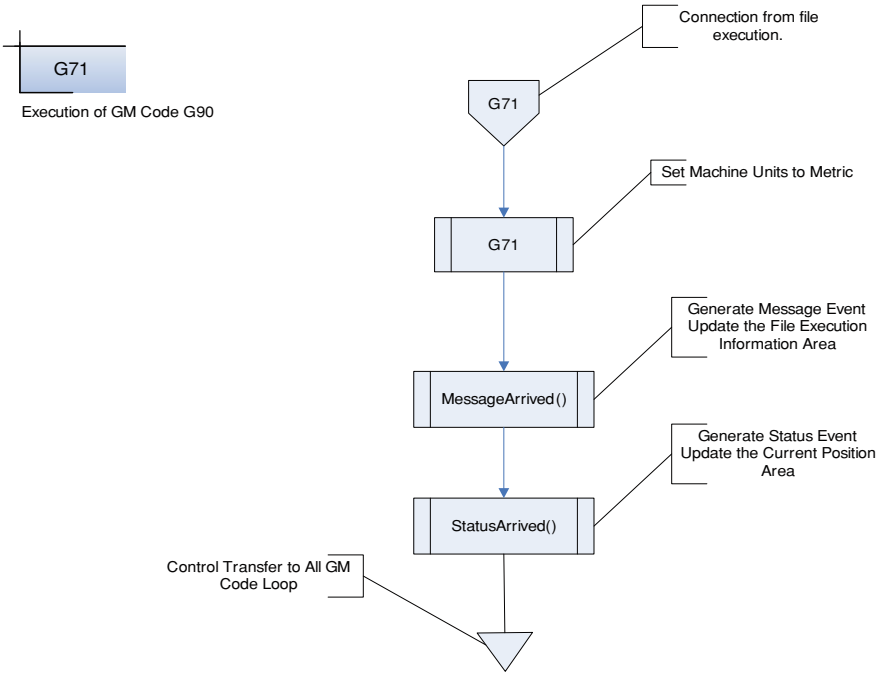


Fig. 5.6aa Set units to metric function (continued)

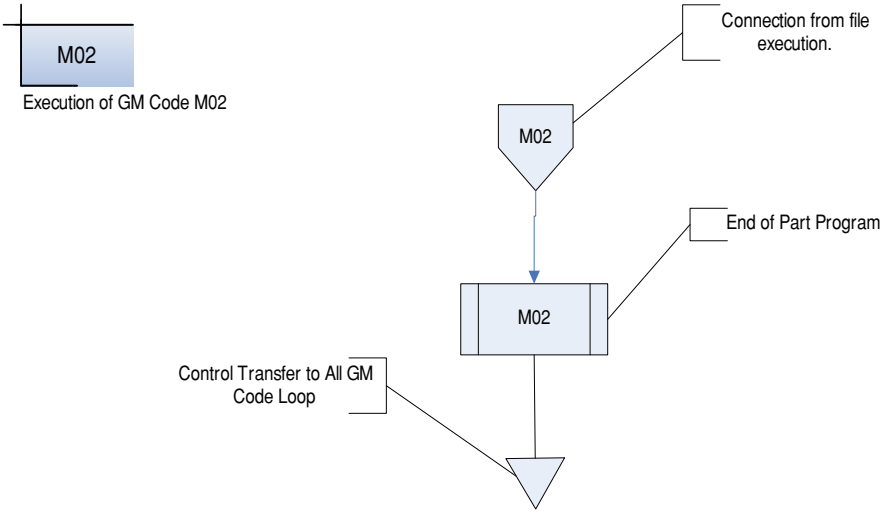


Fig. 5.6ab End of part program function (continued)

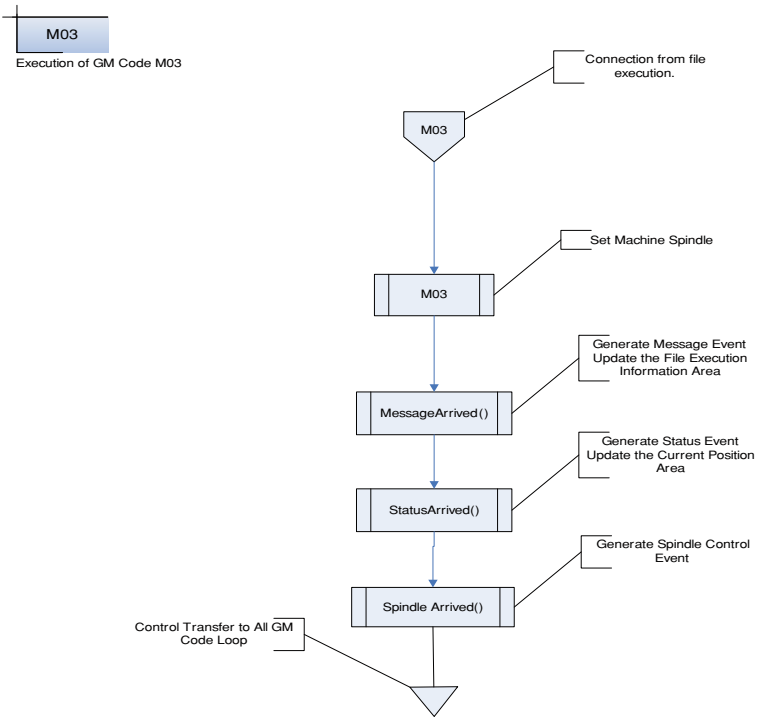


Fig. 5.6ac Set spindle function (continued)

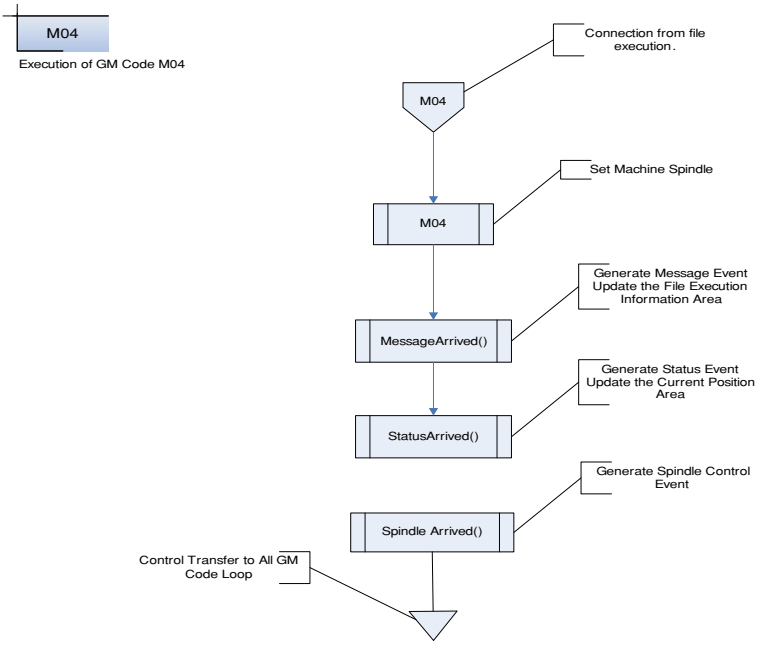


Fig. 5.6ad Set spindle function (continued)

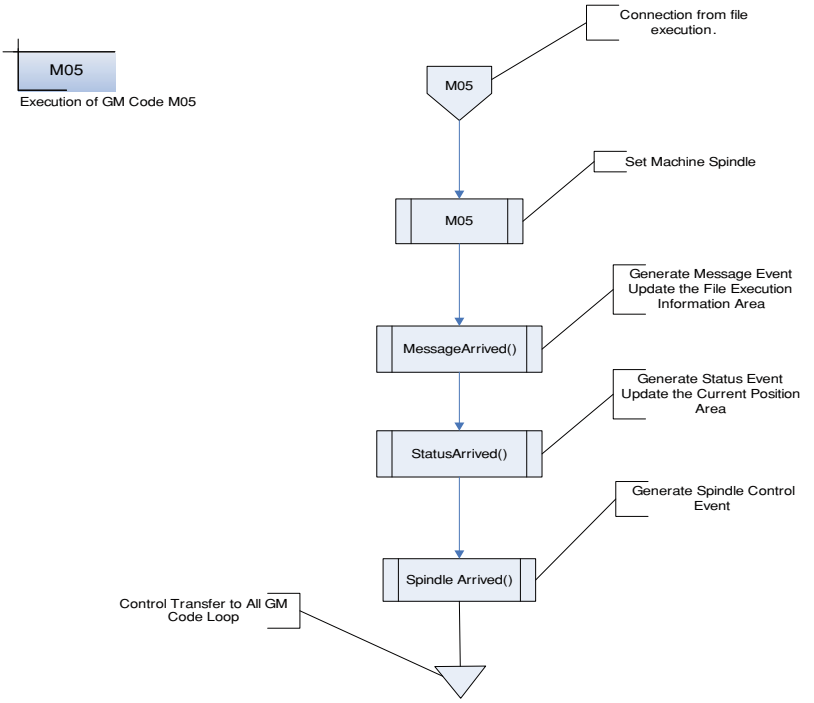


Fig. 5.6ae Set spindle function (continued)

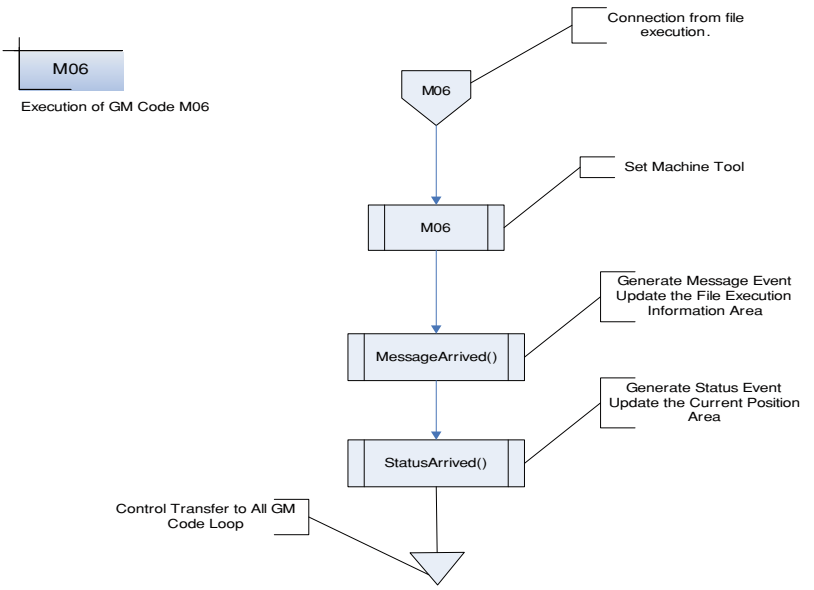


Fig. 5.6af Set tool function (continued)

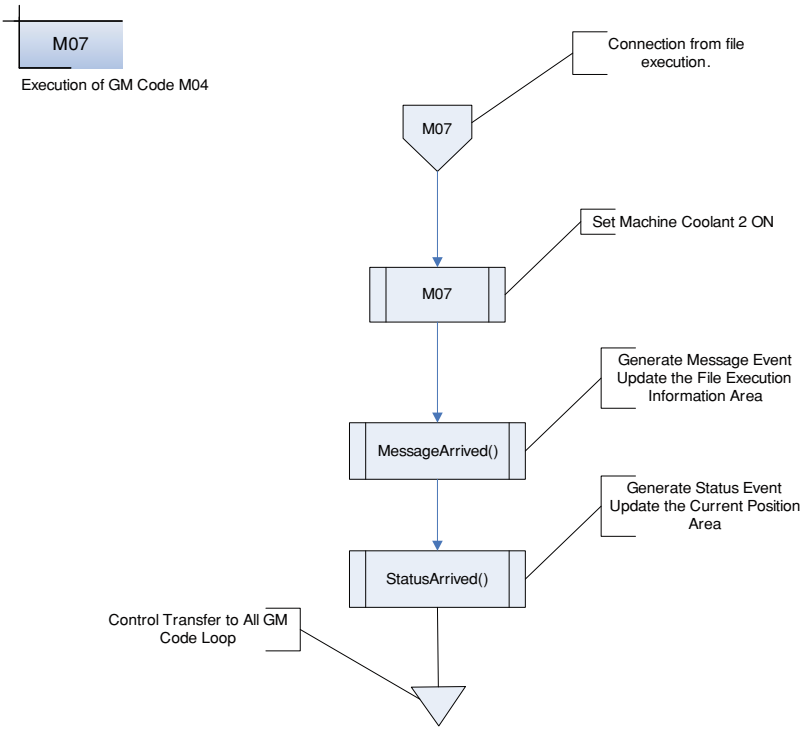


Fig. 5.6ag Set coolant function (continued)

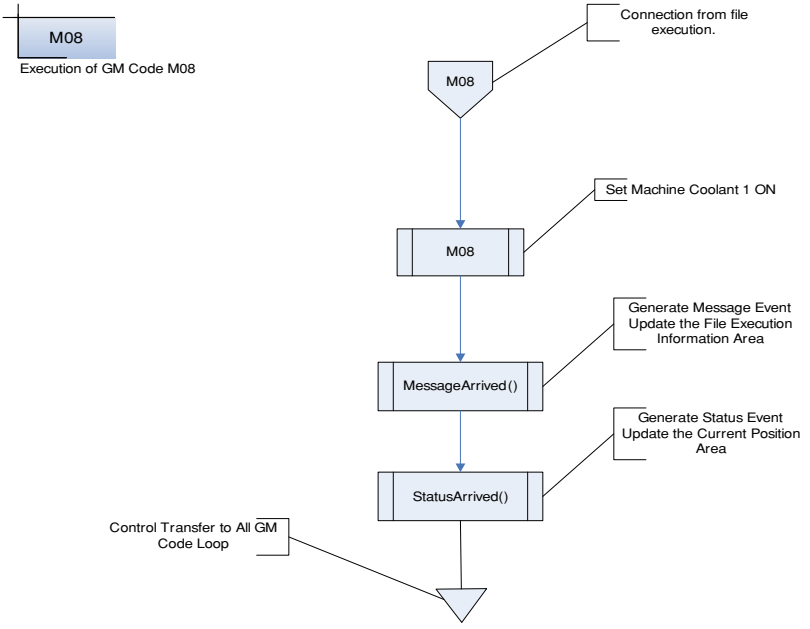


Fig. 5.6ah Set coolant function (continued)

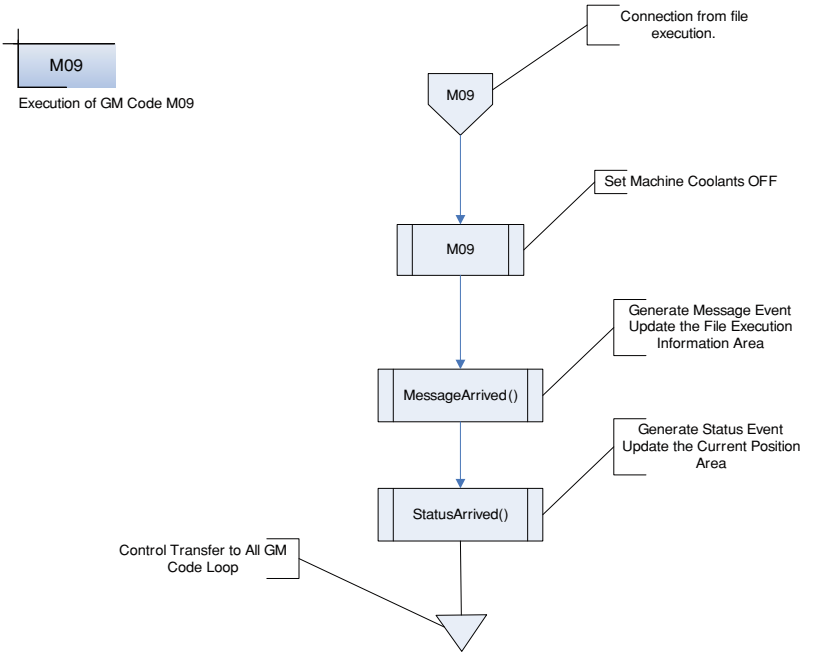


Fig. 5.6ai Set coolant function (continued)

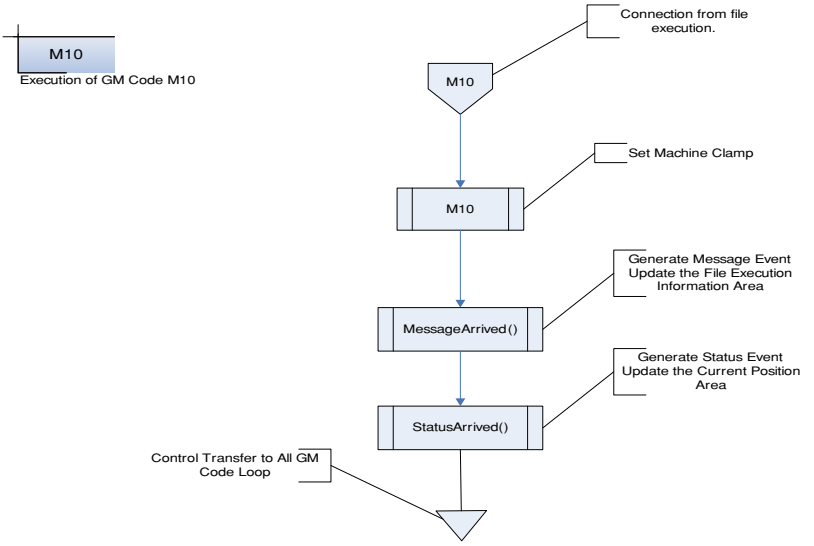


Fig. 5.6aj Set clamp function (continued)

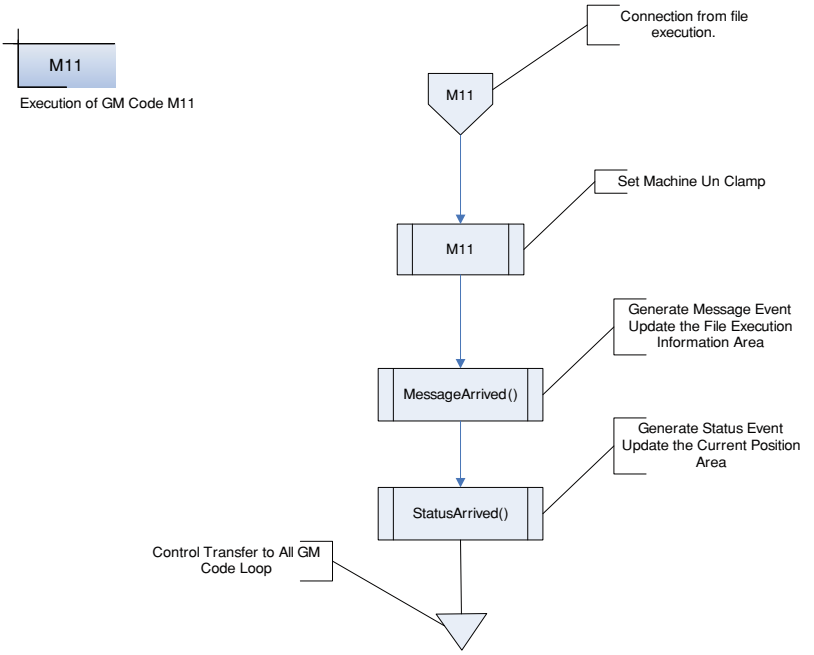


Fig. 5.6ak Set coolant function (continued)

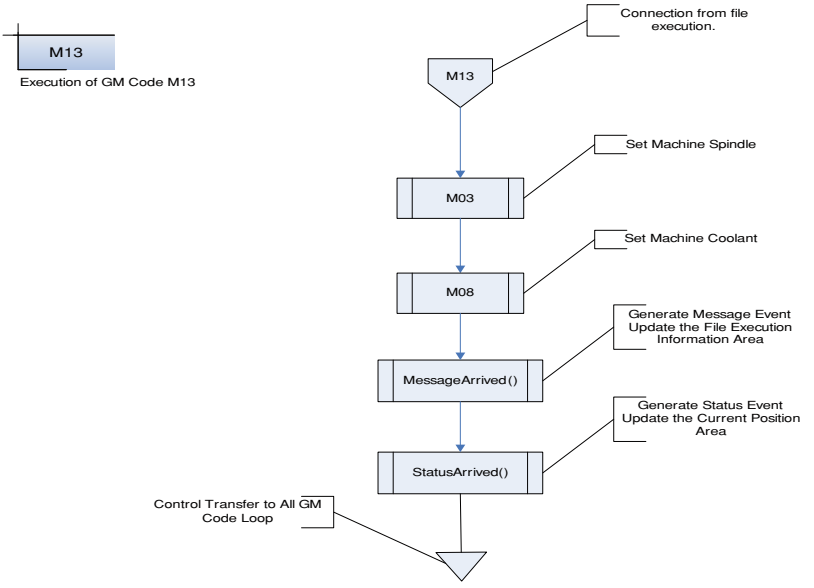


Fig. 5.6al Set spindle and coolant function (continued)

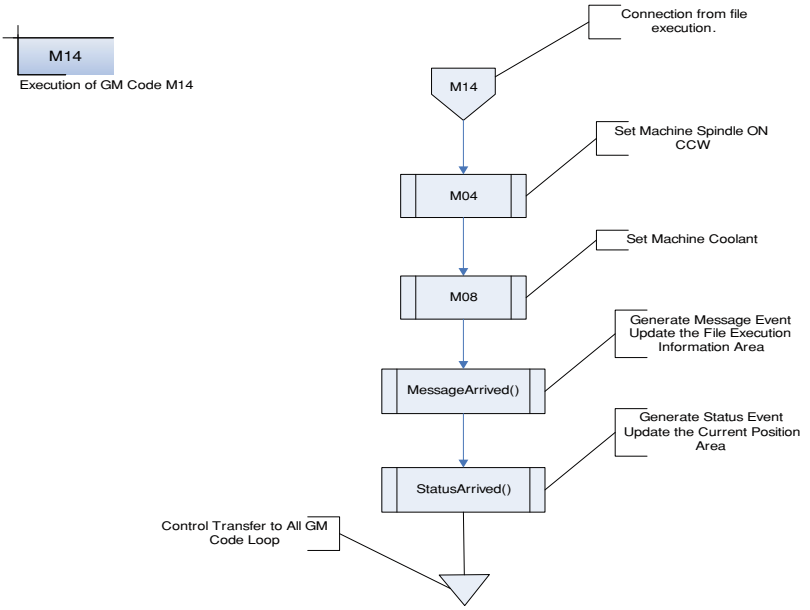


Fig. 5.6am Set spindle ON CCW function (continued)

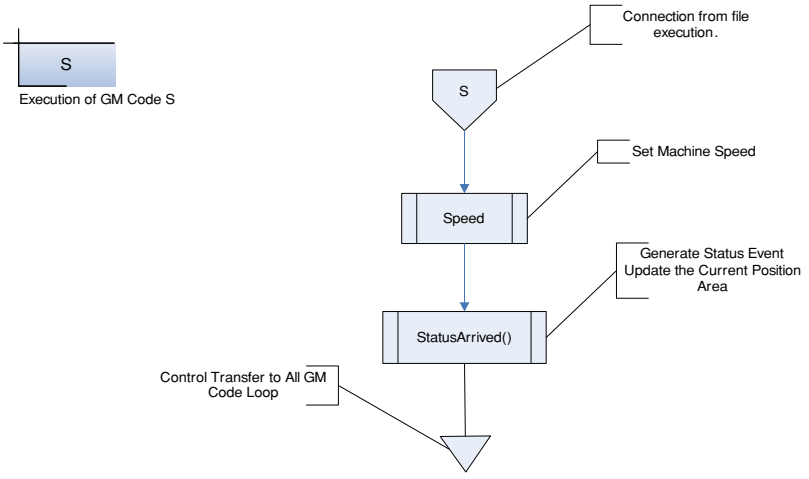


Fig. 5.6an Set speed function (continued)

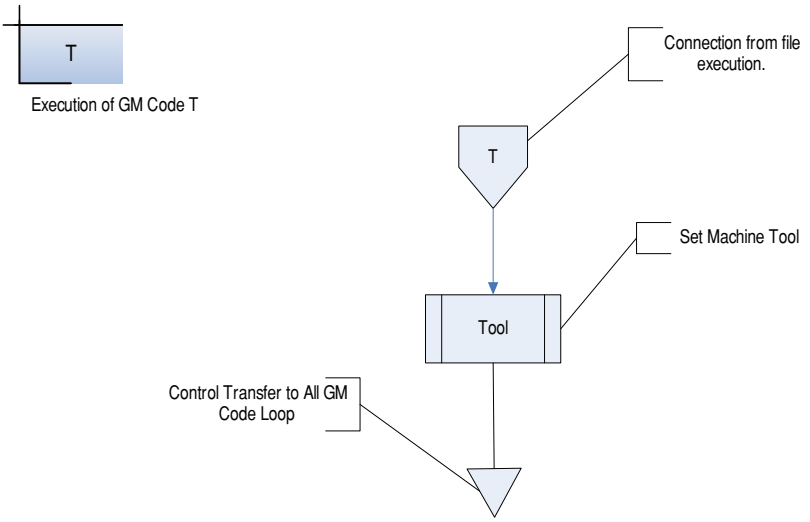


Fig. 5.6ao Set tool function (continued)

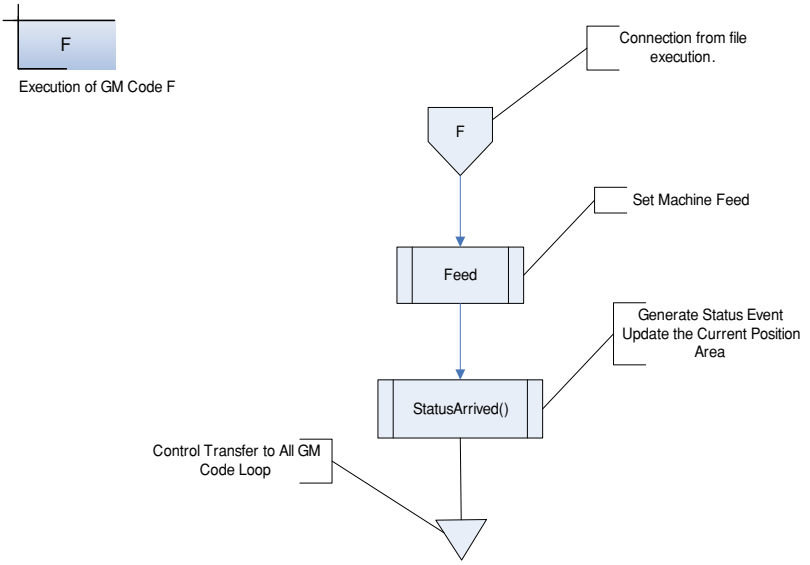


Fig. 5.6ap Set feed function (continued)

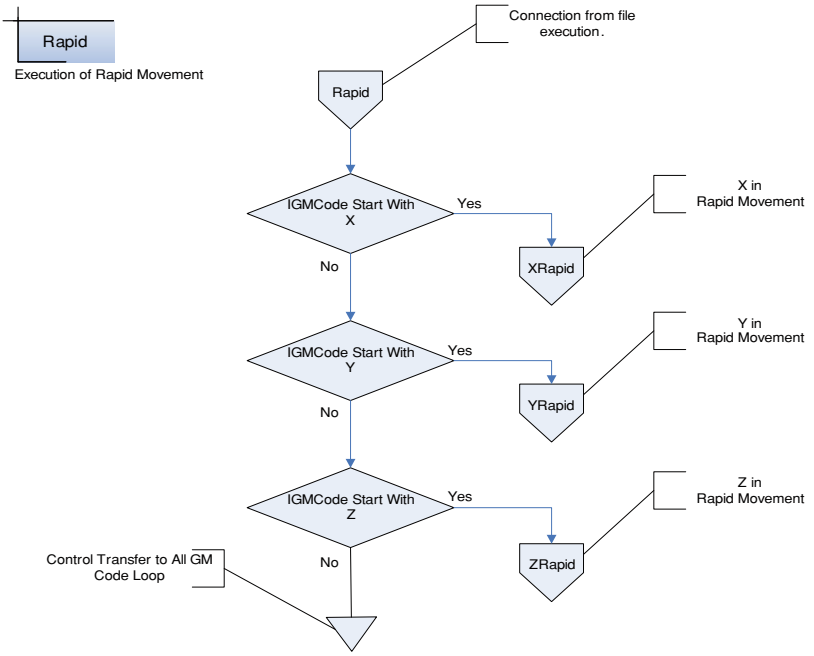


Fig. 5.6aq Rapid movement (continued)

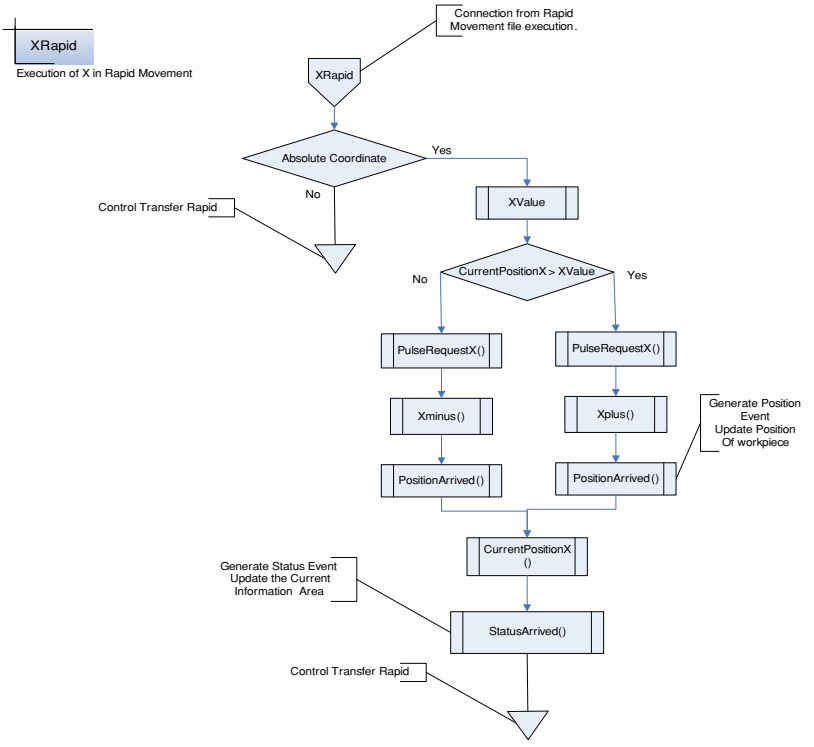


Fig. 5.6ar Rapid X movement (continued)

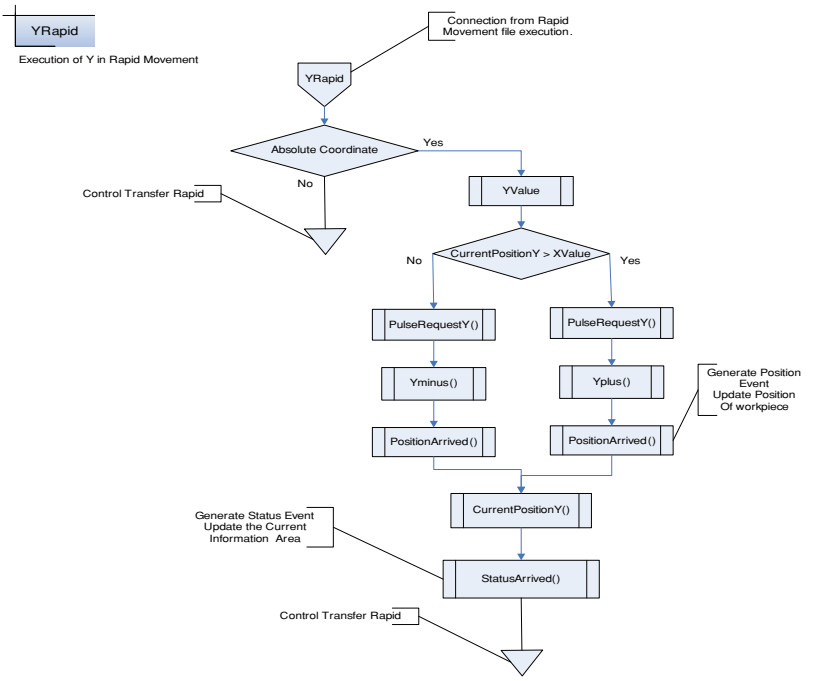


Fig. 5.6as Rapid Y movement (continued)

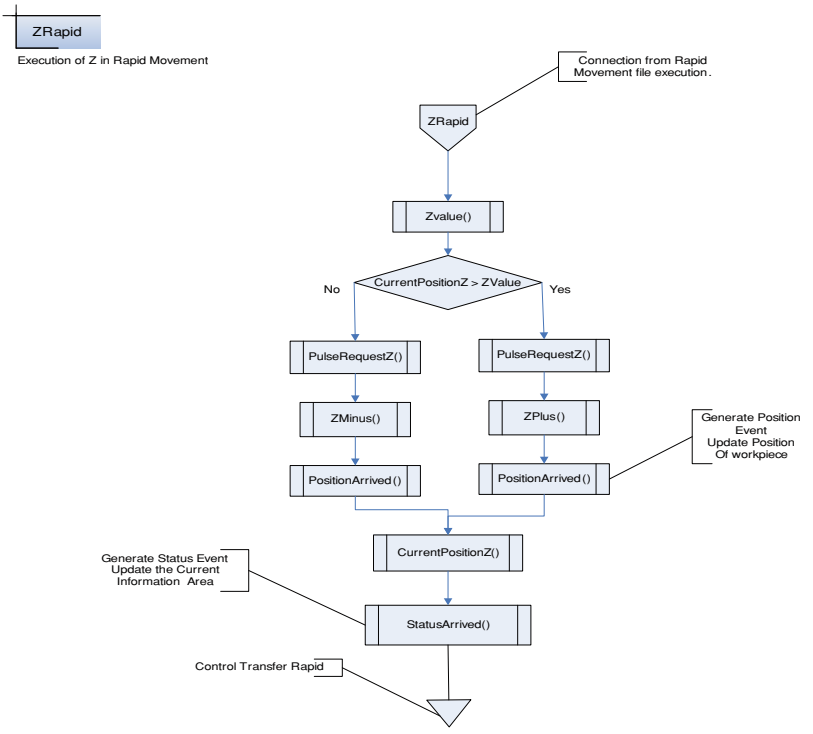


Fig. 5.6at Rapid Z movement (continued)

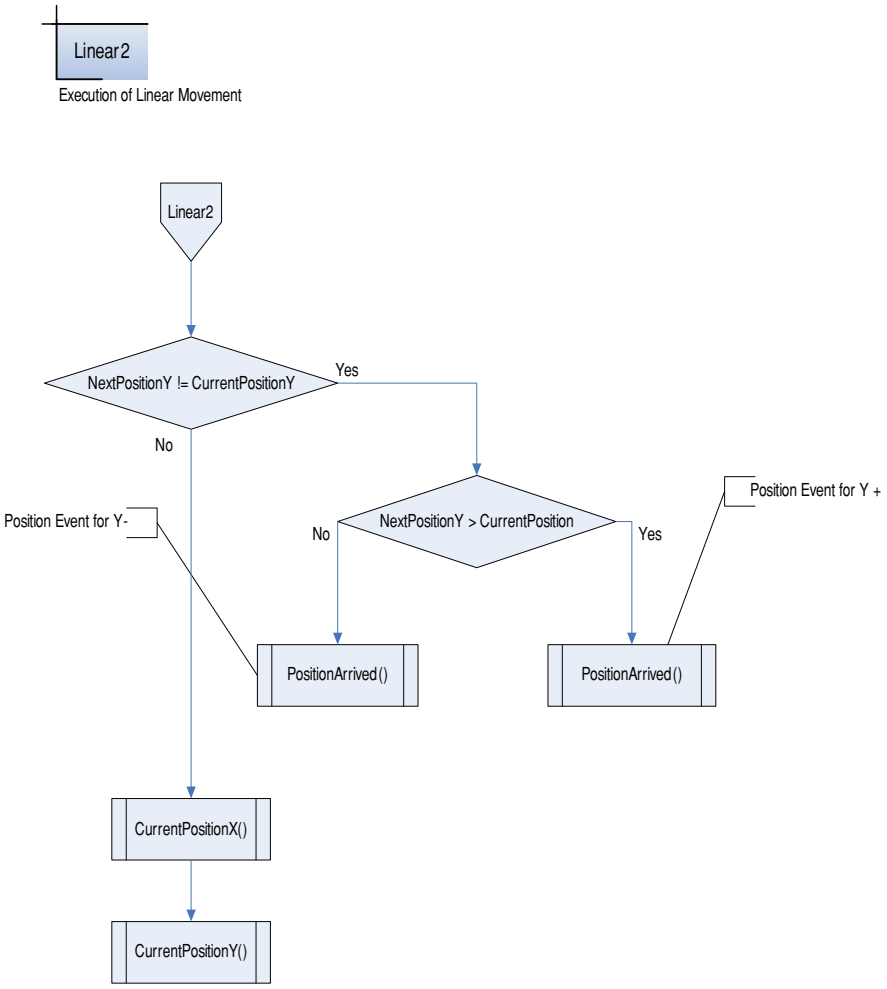


Fig. 5.6au Linear movement (continued)

Execute File, Validate File and Validate Code. A request from CNC milling software for each function is handled by milling class. The enumerations of milling are current and MachineStatus. Current enumeration identifies the current value of different parameter and MachineStatus identifies the machine running status.

To provide access to physical port on the system, milling has MillingPortAccess class. MillingPortAccess has encapsulated port access and interpolation functions. Only milling class has access to MillingPortClass and CNC milling software does not have this facility. Milling class calls appropriate interpolation method, which in turn calls private methods of port access.

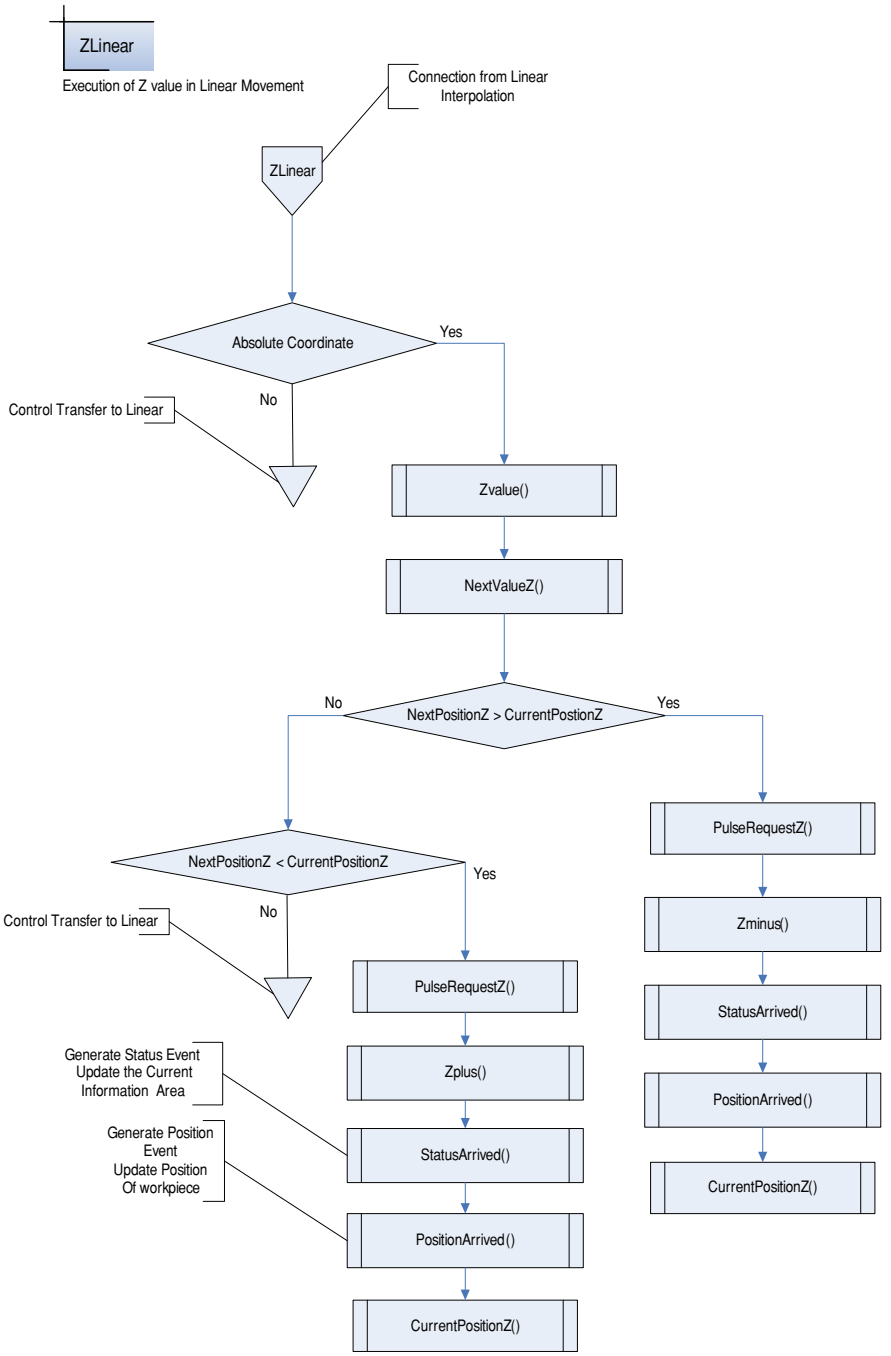


Fig. 5.6av Linear Z movement

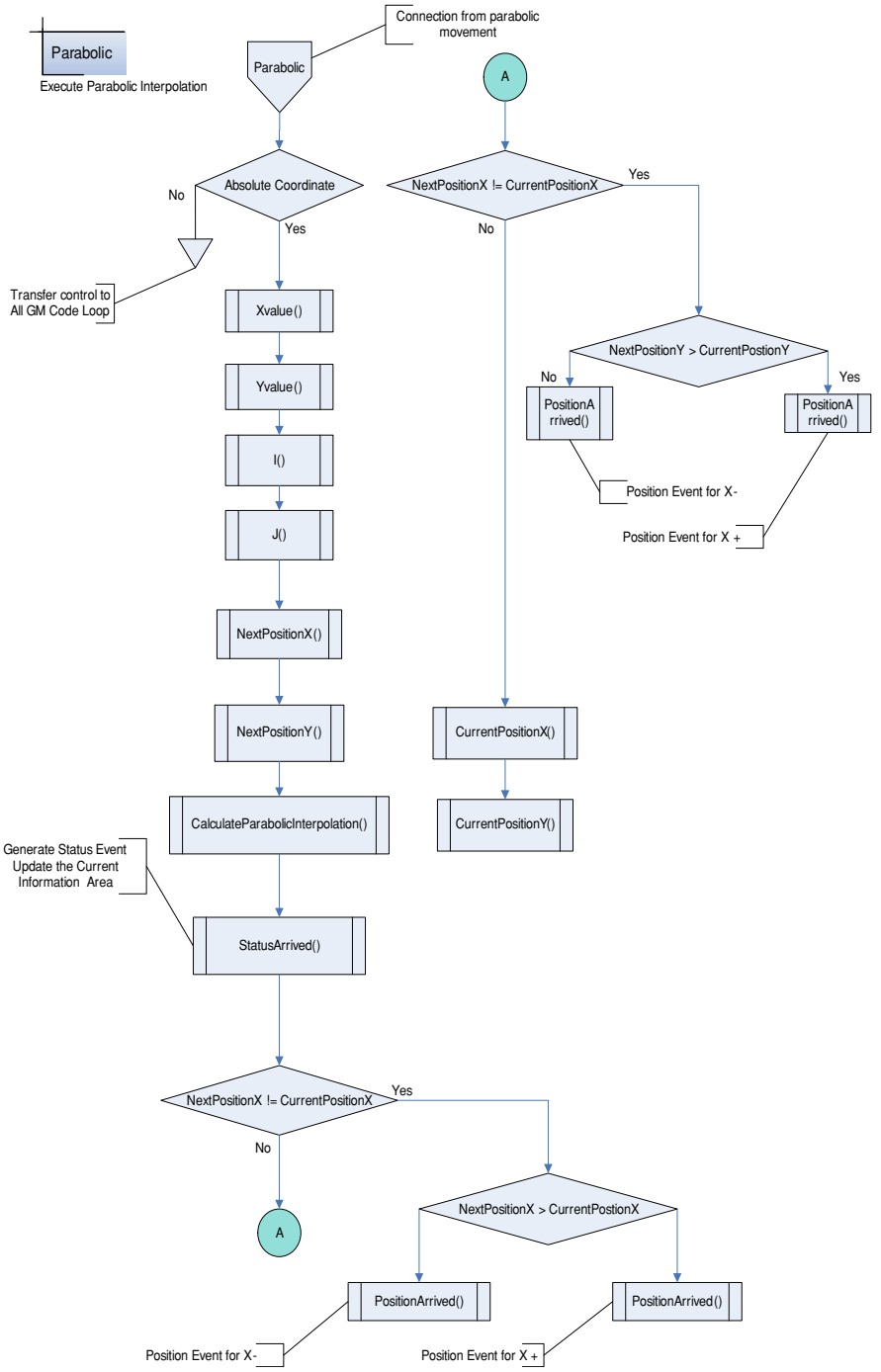


Fig. 5.6aw Parabolic movement

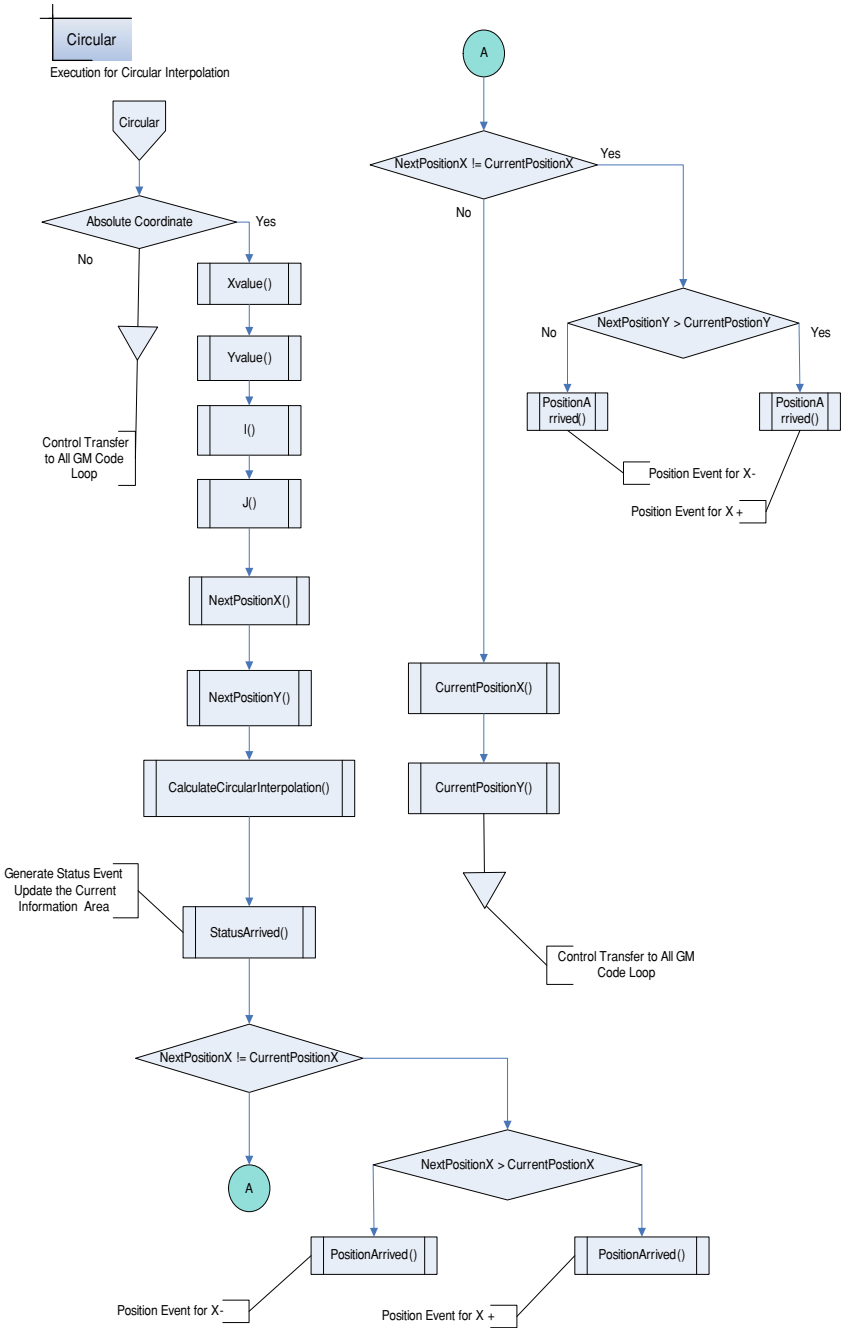


Fig. 5.6ax Circular movement

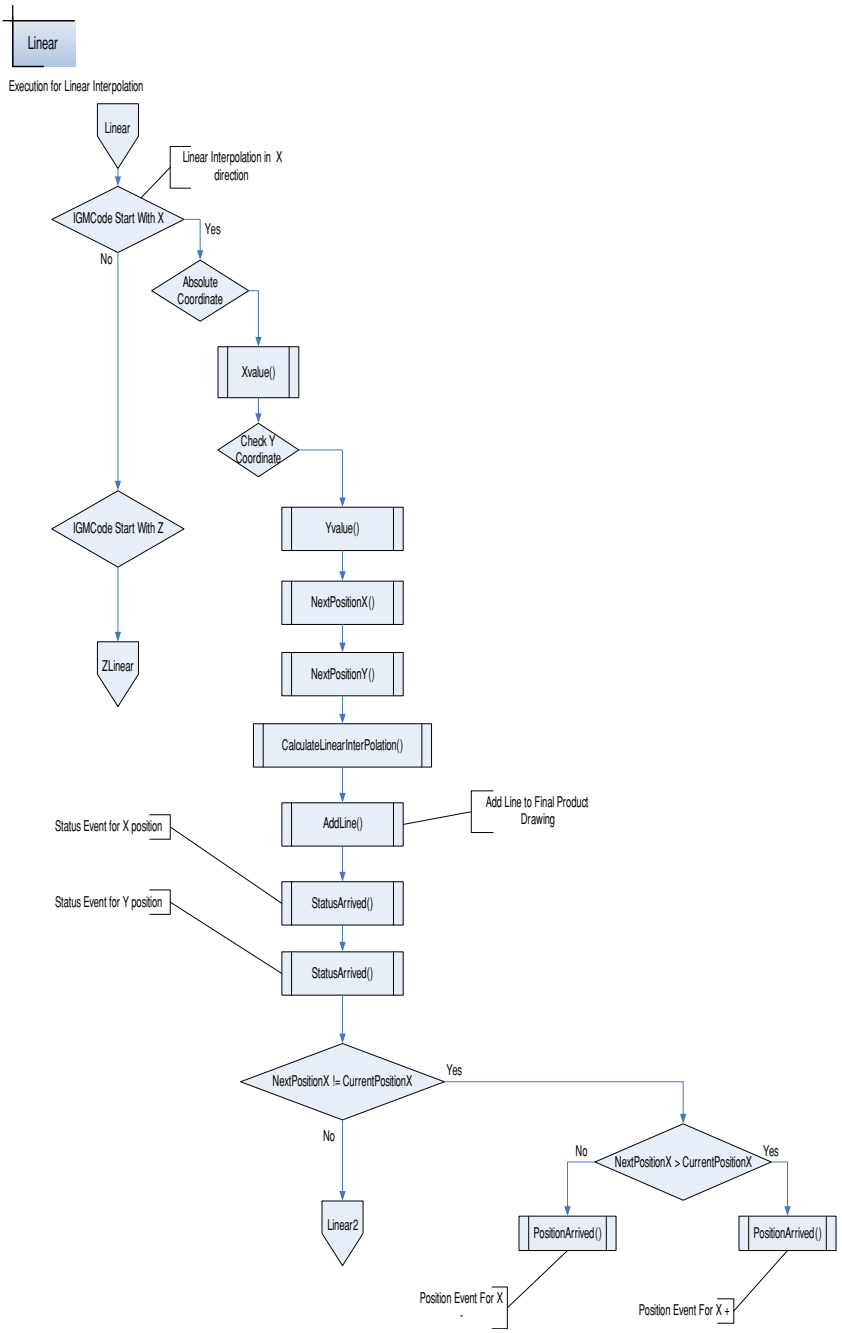


Fig. 5.6ay Linear movement

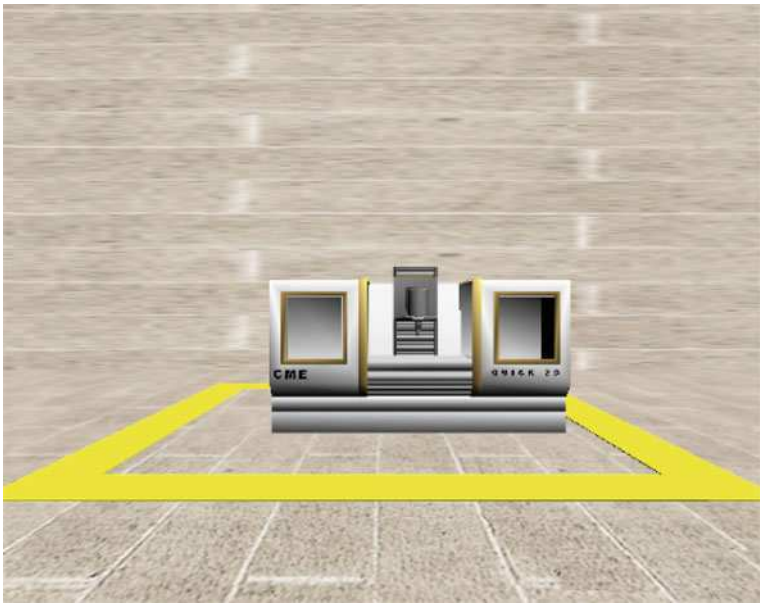


Fig. 5.7 AR for milling machine

In order to update the CNC milling software about the current state of execution, milling has five events: SpindleHandler, MessageHandler, PositionHandler, StatusHandler and LineEventHandler. SpindleHandler event is used to turn the spindle status ON or OFF. MessageHandler is used to show the messages on message information box. PositionHandler is used to show the current position of work piece in x, y, and z-axes. StatusHandler is used to show the current status of milling machine. LineEventHandler is used to draw line on work piece in order to show the cut. Information received from events by CNC milling software is used to update user interface and to control 3D graphics.

frmFinal class of milling is used to draw end product. frmFinal has AddLine method called by ExecuteFile method to pass cutting information on work piece or end product.

2. CNC milling: Consists of CNC milling software user interface that utilizes milling object to execute file and related functions. Current status of execution is also updated from CNC milling.
3. Milling machine animation: Consists of milling3D class and three enumerations. Milling3D class consists of 3D graphics rendering engine and provide methods to control movement of individual parts of machine. Enumerations of milling machine animation are direction, Spindle, and Axis.

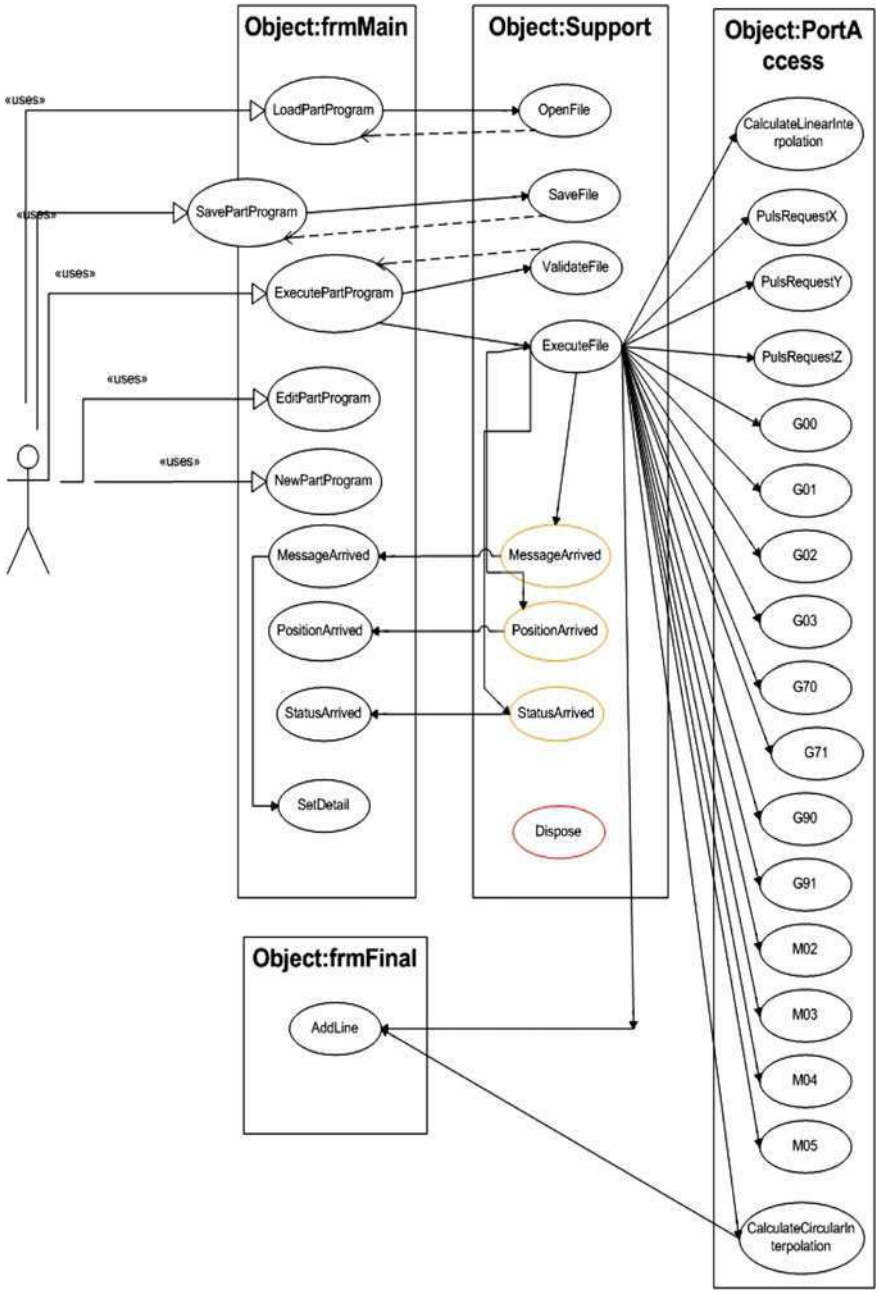


Fig. 5.8 CNC turning UML use case diagram

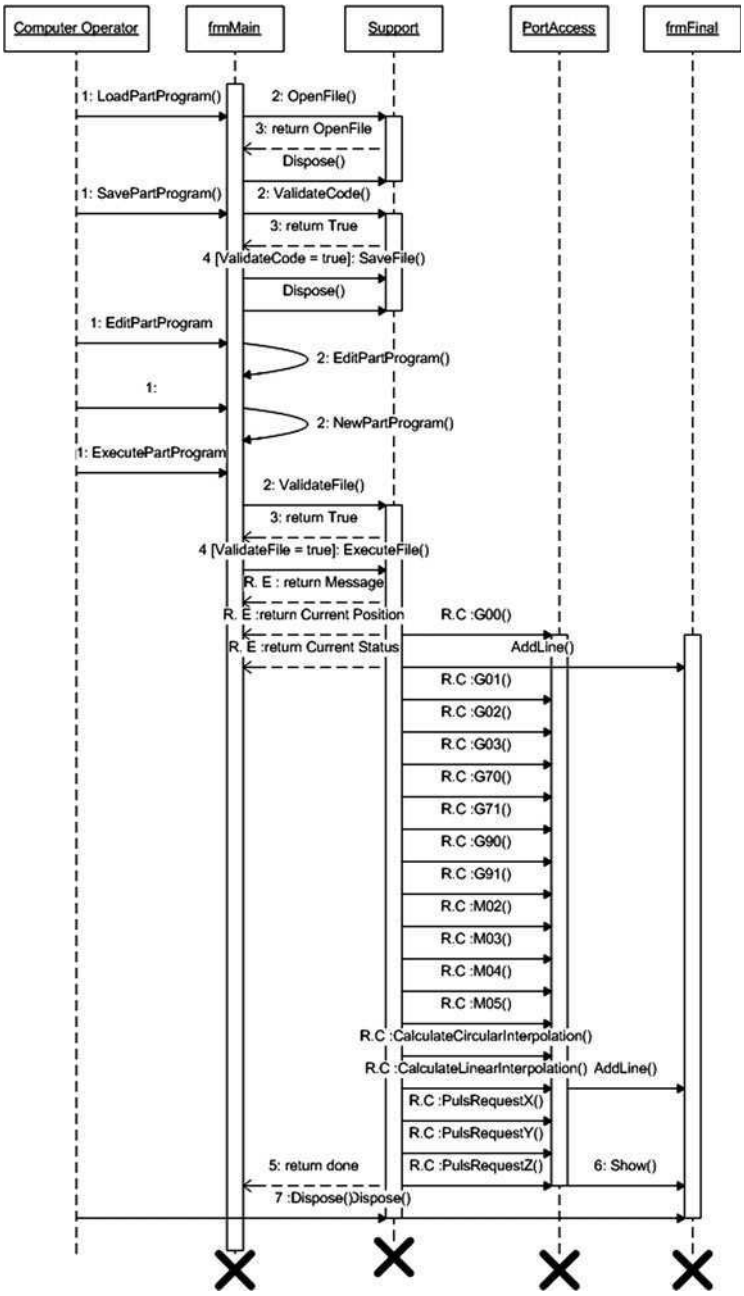


Fig. 5.9 CNC turning UML sequence diagram

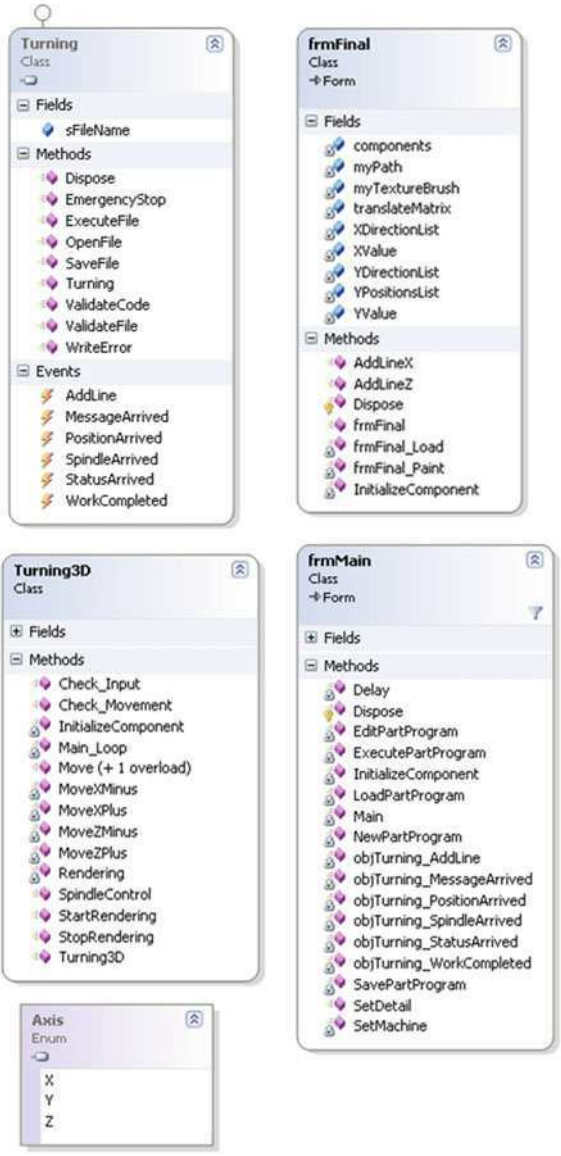


Fig. 5.10 CNC UML static diagrams for turning operation

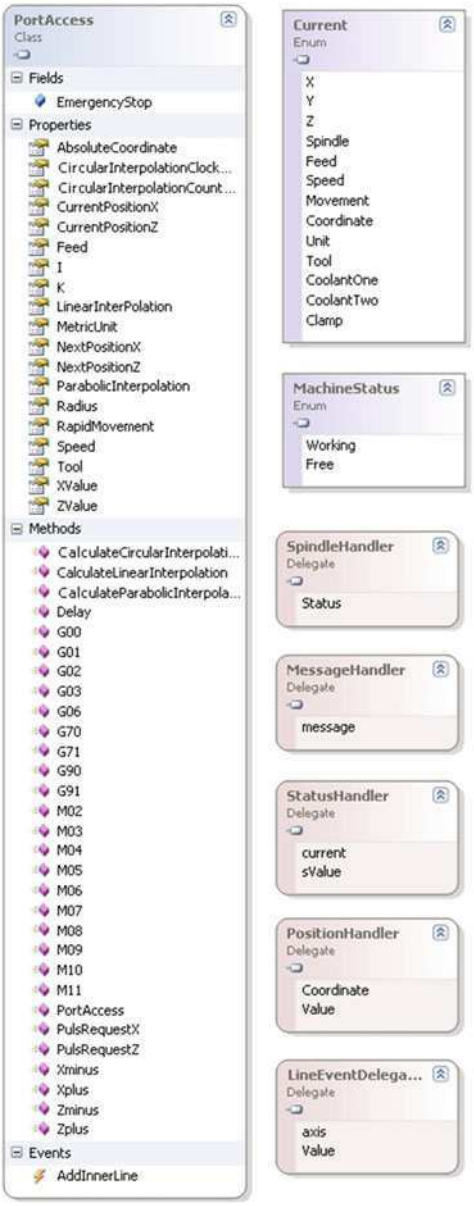


Fig. 5.10 (continued)

Table 5.2 shows the MillingPortAccess class members properties, methods, and variables. MillingPortAccess is the helper class of milling, and this class is used to pass pulses to CNC machine by using parallel port and calculate

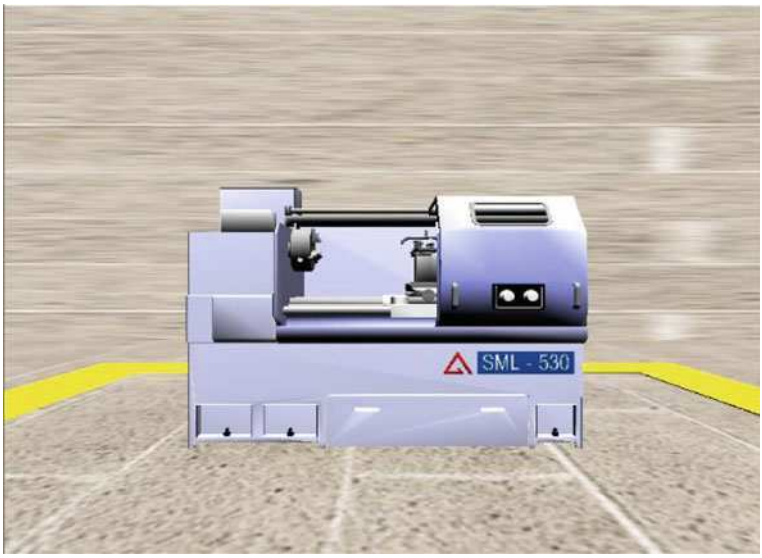


Fig. 5.11 AR for turning machine

the pulses, liner interpolation, circular interpolation, parabolic interpolation. MillingPortAccess also holds the current position of *x*, *y,* and *z*-axes and current state of the milling machine.

Table 5.3 describe the milling class members variables, properties, and methods. The milling class is used to read the part program file, validate it, open it, save it, and execute it. This class is also responsible for raising events to update user interface and 3D milling machine. Interpreter of CNC milling is also implemented in this class's ExecuteFile method.

Table 5.4 presents the Milling3D class member variables, properties, and methods. Milling3D class is used to render the graphics of milling machine in 3D format for display movement of machine and work piece. 3D rendering engine is also implemented in this class.

Table 5.5 outlines the summary of the CNC milling sequence diagram.

Table 5.6 depicts the summary of the CNC milling load part program use case.

Table 5.7 defines the summary of the CNC milling save part program use case.

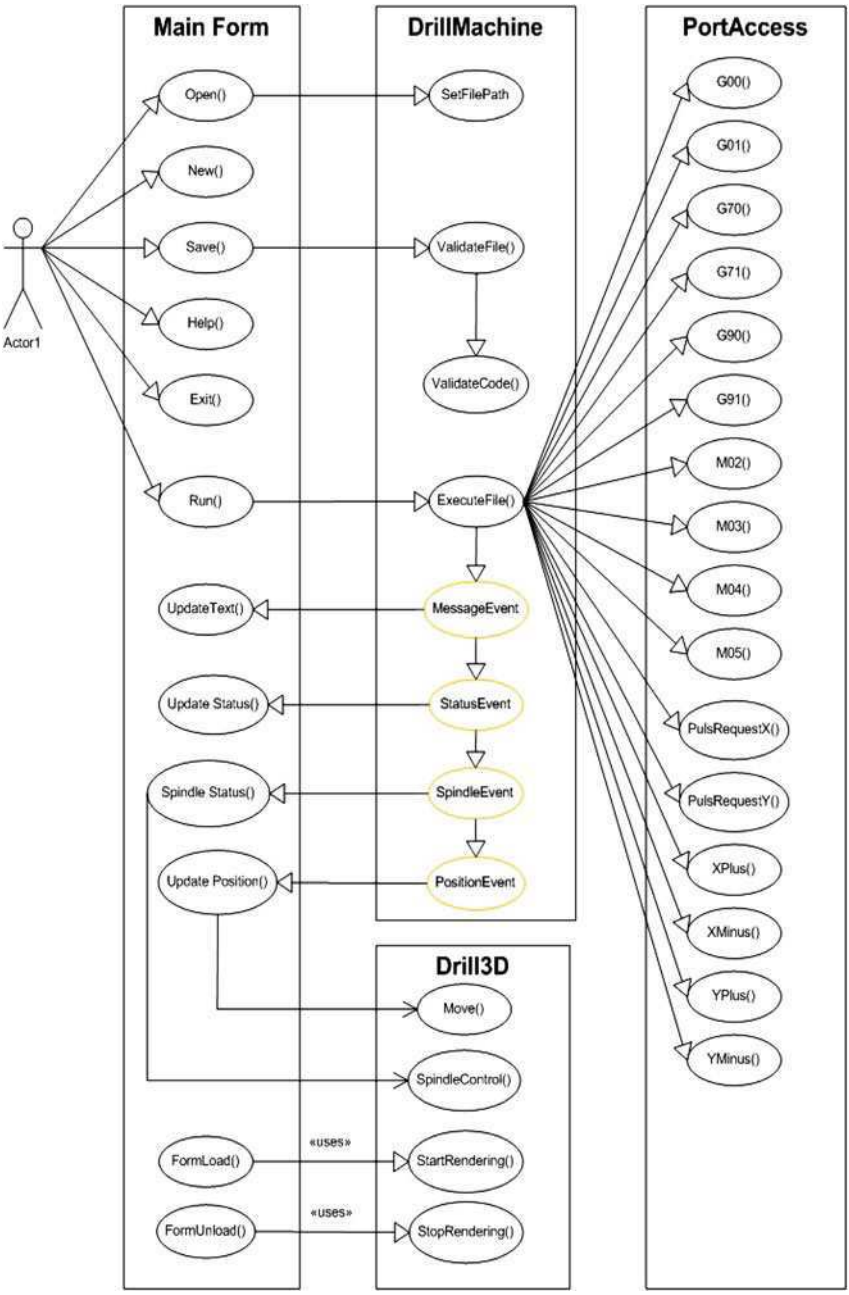


Fig. 5.12 Use case diagram of drill project

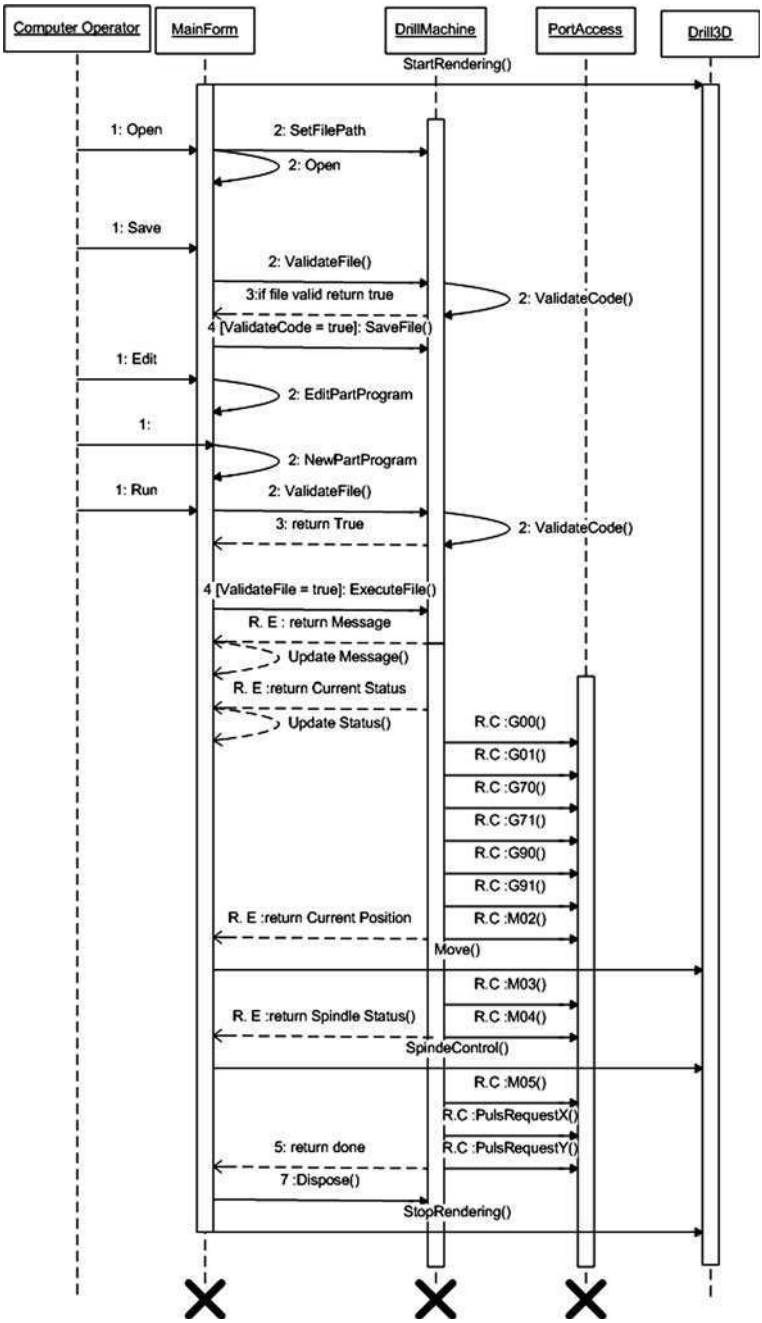


Fig. 5.13 CNC drill UML sequence diagram

Fig. 5.14 UML static diagram of drill project

Table 5.8 illustrates the summary of the CNC milling execute part program use case.

Table 5.9 displays the summary of the CNC milling edit part program use case.

Table 5.10 shows the summary of the CNC milling new part program use case.

Table 5.11 explains the summary of the CNC milling flow charts.

A section of C# code defining interpreter for EIA 274 D part program for milling operation is listed below:

The following C# code is taken from milling class's ExecuteFile member method. ExecuteFile method is the interpreter of CNC milling machine. In this

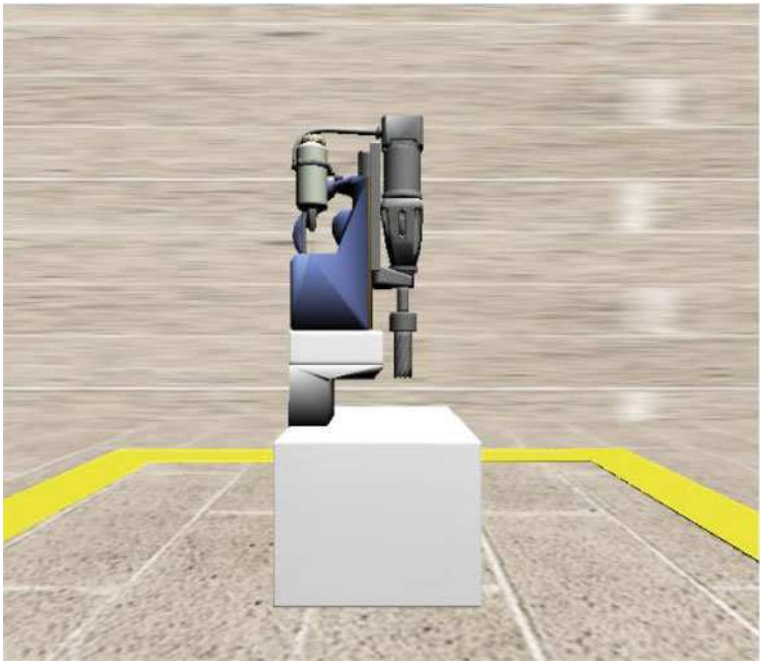


Fig. 5.15 AR of CNC drill

code, it reads part program command one by one from an array and instructs PortAccess class to calculate required number of pulses by using Linear Interpolation, Circular Interpolation or Parabolic Interpolation. It also updates user interface and 3D machine with the help of events. ValidateFile method is called before Execute method. ValidateFile method validates part program commands with ValideCode method and fills an array with validated command. In case of any invalid command, ExecuteFile method is not called and returns an error to inform user.

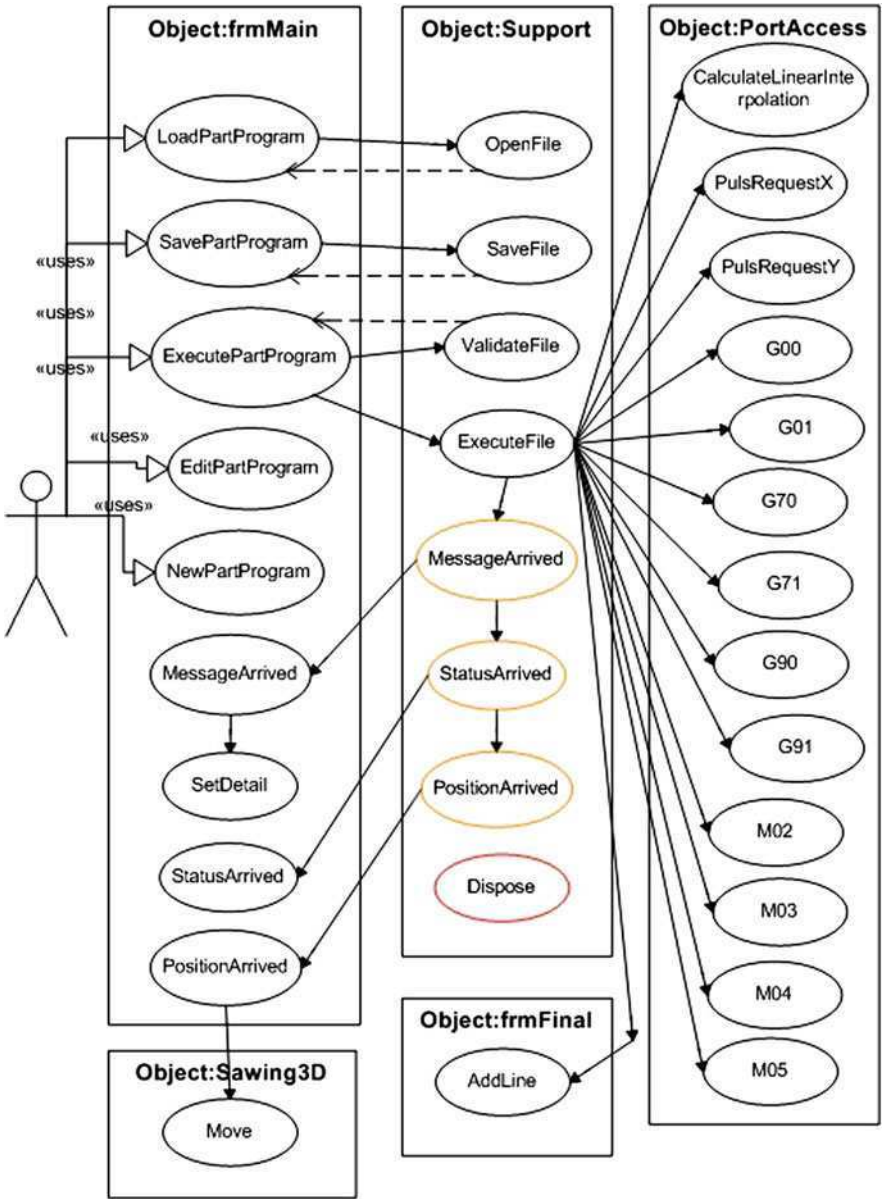


Fig. 5.16 CNC sawing UML use case diagram

Interpreter Source Code (Milling)

```
public void ExecuteFile()
              {

                      status = MachineStatus.Working;
                      /// lGMCode string variable for G M Code
                      string lGMCode;
                      /// aList is a ArrayList object and fill in
                      /// ValidateFile function.
                      for(int i =0; i < aList.Count; i++)
                      {

                              Application.DoEvents();
                              /// if Emergency stop then exit from
loop
                              if(MillingPortAccess.EmergencyStop)
                                      break;

                              #region Machine Preparation Handling
Code
                              /// <summary>
                              /// In this region We take one GMCode
at a time
                              /// from aList array in local string
variable lGMCode
                              /// and check Against all G M Codes
                              /// </summary>
                              lGMCode = aList[i].ToString();
                              if(lGMCode == "G0" || lGMCode == "G00")
                              {
                                      ///call Rapid Movement function
                                      Port.G00();
                                      ///Generate the evnet, this
event will
                                      ///handle by calling routin.
this event will
                                      ///inform the current status or
activity to
                                      ///object of this class.
                                      MessageArrived("G0 ok");

       StatusArrived(Current.Movement,"Rapid");

                              }
                              else if(lGMCode == "G01" || lGMCode ==
"G1")
                              {
                                      ///call Linear Interpolation
function

                                      Port.G01();
                                      ///Generate the evnet, this
event will
                                      ///handle by calling routin
                                      MessageArrived("G1 OK");
```

```
        StatusArrived(Current.Movement,"Linear");
                                }
                                else if(lGMCode == "G02" || lGMCode ==
"G2")
                                {
                                        ///call Circular Interpolation
c/w function
                                        Port.G02();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G2 OK");

        StatusArrived(Current.Movement,"Circular C/W");
                                }
                                else if(lGMCode == "G03" || lGMCode ==
"G3")
                                {
                                        /// call Circular Interpolation
cc/w function
                                        Port.G03();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G03 OK");

        StatusArrived(Current.Movement,"Circular CC/W");
                                }
                                else if(lGMCode == "G04" || lGMCode ==
"G4")
                                {
                                        /// Dwell
                                        Port.G04();
                                        MessageArrived("Dwell OK");
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin

                                }
                                else if(lGMCode == "G06" || lGMCode ==
"G6")
                                {
                                        /// call Parabolic Interpolation
function
                                        Port.G06();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G06 OK");

        StatusArrived(Current.Movement,"Parabolic");
                                }
                                else if(lGMCode == "G17")
                                {
                                        /// XY Plane Selection
                                        Port.G17();
                                        MessageArrived("G17 OK");
```

```
                        }
                        else if(lGMCode == "G18")
                        {
                                // ZX Plane Selection
                                Port.G18();
                                MessageArrived("G18 OK");

                        }
                        else if(lGMCode == "G19")
                        {
                                // YZ Plane Selection
                                Port.G19();
                                MessageArrived("G19 OK");

                        }
                        else if(lGMCode == "G33")
                        {
                                // Thread Cutting, Constant Lead
                                Port.G33();
                                MessageArrived("G33 OK");

                        }
                        else if(lGMCode == "G34")
                        {
                                // Thread Cutting, Increasing
Lead
                                Port.G34();
                                MessageArrived("G34 OK");

                        }
                        else if(lGMCode == "G35")
                        {
                                // Thread Cutting, Decreasing
Lead
                                Port.G35();
                                MessageArrived("G35 OK");

                        }
                        else if(lGMCode == "G40")
                        {
                                // Cutter Compensation Cancel
                                Port.G40();
                                MessageArrived("G40 OK");

                        }
                        else if(lGMCode == "G41")
                        {
                                // Cutter Compensation Left
                                Port.G41();
                                MessageArrived("G41 OK");

                        }
                        else if(lGMCode == "G42")
                        {
                                // Cutter Compensation Right
                                Port.G42();
```

```
                                        MessageArrived("G42 OK");

                                }
                                else if(lGMCode == "G43")
                                {
                                        // Cutter Offset - Inside Corner
                                        Port.G43();
                                        MessageArrived("G43 OK");

                                }
                                else if(lGMCode == "G44")
                                {
                                        // Cutter Offset - Outside
Corner
                                        Port.G44();
                                        MessageArrived("G44 OK");
                                }
                                else if(lGMCode == "G90")
                                {
                                        /// call Absolute Coordinate
function
                                        Port.G90();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G90 OK");

        StatusArrived(Current.Coordinate,"Absolute");
                                }
                                else if(lGMCode == "G91")
                                {
                                        /// call Incremental Coordinate
function
                                        Port.G91();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G91 OK");

        StatusArrived(Current.Coordinate,"Incremental");
                                }
                                else if(lGMCode == "G70")
                                {
                                        /// call Inch Units
                                        Port.G70();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("G70 OK");

        StatusArrived(Current.Unit,"Inch");
                                }
                                else if(lGMCode == "G71")
                                {
                                        /// call Metric Units function
                                        Port.G71();
                                        ///Generate the evnet, this
event will
```

```
                                        ///handle by calling routin
                                        MessageArrived("G71 OK");

        StatusArrived(Current.Unit,"MM");
                                }
                                else if(lGMCode == "M02" || lGMCode ==
"M2" || lGMCode == "M00")
                                {
                                        Port.M02();

                                }
                                else if(lGMCode == "M03" || lGMCode ==
"M3")
                                {
                                        Port.M03();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("M3 OK");

        StatusArrived(Current.Spindle,"ON C/W");
                                        SpindleArrived(true);
                                }
                                else if(lGMCode == "M04" || lGMCode ==
"M4")
                                {
                                        Port.M04();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("M4 OK");

        StatusArrived(Current.Spindle,"ON CC/W");
                                        SpindleArrived(true);
                                }
                                else if(lGMCode == "M05" || lGMCode ==
"M5")
                                {
                                        Port.M05();
                                        ///Generate the evnet, this
event will
                                        ///handle by calling routin
                                        MessageArrived("M5 OK");

        StatusArrived(Current.Spindle,"OFF");
                                        SpindleArrived(false);
                                }
                                else if(lGMCode == "M06")
                                {
                                        //Tool Change
                                        Port.M06();
                                        MessageArrived("M06 OK");

        StatusArrived(Current.Tool,Port.Tool.ToString());
                                }
                                else if(lGMCode == "M07")
                                {
                                        //Coolant 2 on
```

```
                                        Port.M07();
                                        MessageArrived("M07 OK");

        StatusArrived(Current.CoolantTwo,"On");
                                }
                                else if(lGMCode == "M08")
                                {
                                        //Coolant 1 on
                                        Port.M08();
                                        MessageArrived("M08 OK");

        StatusArrived(Current.CoolantOne,"On");
                                }
                                else if(lGMCode == "M09")
                                {
                                        //Coolant off
                                        Port.M09();
                                        MessageArrived("M09 OK");

        StatusArrived(Current.CoolantOne,"Off");

        StatusArrived(Current.CoolantTwo,"Off");
                                }
                                else if(lGMCode == "M10")
                                {
                                        //Clamp
                                        Port.M10();
                                        MessageArrived("M10 OK");

        StatusArrived(Current.Clamp,"Clamp");
                                }
                                else if(lGMCode == "M11")
                                {
                                        //Un Clamp
                                        Port.M11();
                                        MessageArrived("M11 OK");
                                        StatusArrived(Current.Clamp,"Un
Clamp");
                                }
                                else if(lGMCode == "M13")
                                {
                                        //Spindle c/w & Coolant on
                                        //Spindle On c/w
                                        Port.M03();
                                        //Coolant On
                                        Port.M08();
                                        MessageArrived("M13 OK");

        StatusArrived(Current.Spindle,"On");

        StatusArrived(Current.CoolantOne,"On");
                                }
                                else if(lGMCode == "M14")
                                {
                                        //Spindle cc/w & Coolant on
                                        //Spindle On cc/w
                                        Port.M04();
                                        //Coolant On
```

```
                                        Port.M08();
                                        MessageArrived("M14 OK");

        StatusArrived(Current.Spindle,"On");

        StatusArrived(Current.CoolantOne,"On");
                                }
                                else if(lGMCode.StartsWith("G04"))
                                {
                                        string DwellValue =
aList[i+1].ToString();
                                        if(DwellValue.StartsWith("F"))
                                        {
                                                int DwellTime =
Convert.ToInt32(DwellValue.Substring(1,DwellValue.Length - 1));

                                        }

                                }

                                else if(lGMCode.StartsWith("S"))
                                {
                                        Port.Speed =
Convert.ToInt16(lGMCode.Substring(1,lGMCode.Length - 1));

        StatusArrived(Current.Speed,Port.Speed.ToString());
                                }
                                else if(lGMCode.StartsWith("F"))
                                {
                                        Port.Feed =
Convert.ToInt16(lGMCode.Substring(1,lGMCode.Length - 1));

        StatusArrived(Current.Feed,Port.Feed.ToString());
                                }
                                else if(lGMCode.StartsWith("T"))
                                {
                                        Port.Tool =
Convert.ToInt16(lGMCode.Substring(1,lGMCode.Length - 1));


                                }
                                #endregion
                                #region Rapid Movement Handling Code
                                /// <summary>
                                /// In this region We Handle Rapid
Movement
                                /// And check X Y and Z Coordinate
Separately
                                /// </summary>
                                if(Port.RapidMovement)
                                {
                                        #region X in RapidMovement

                                        if(lGMCode.StartsWith("X"))
                                        {
                                                /// GMCode first letter
is X
```

```
        if(Port.AbsoluteCoordinate)
                                                {
                                                        /// take the
value after X
                                                        /// if there is
X55 then we
                                                        /// take only 55
and asign in
                                                        /// Port.Xvalue
                                                        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                                                        /// and now check
the current postion of x with new
                                                        /// value of x.
this check will control the direction of
                                                        /// stepper motor
in + or - direction. Suppose x current position is 100 and
                                                        /// new value of
x is 150, its mean we move 50 mm in x plus direction
                                                        /// to reach 150
location.

        if(Port.CurrentPositionX > Port.XValue)
                                                        {
                                                                ///set the
pulses of X stepper motor

        ///xcurrent positon is 100 and new position will be
                                                                ///150 we
need only move 50 mm
                                                                ///so
minus the currentposition from new position

        Port.PulsRequestX(Port.CurrentPositionX - Port.XValue);
                                                                /// send
the pulses to stepper motor

        Port.Xplus();
                                                                /// set
the location of work pice of graphic of
                                                                ///
milling machine

        PositionArrived("X-",Port.CurrentPositionX - Port.XValue);
                                                        }
                                                        else
                                                        {
                                                                /// same
as above but reverse direction

        Port.PulsRequestX(Port.XValue - Port.CurrentPositionX);

        Port.Xminus();

        PositionArrived("X+",Port.XValue - Port.CurrentPositionX);
                                                        }
```

```
                                                    /// when we reach
on 150 location of x
                                                    /// then the
current position of x is 150
                                                    /// so we set the
currentposition with new position of x

        Port.CurrentPositionX = Port.XValue;
                                                    /// set the
xValue of current position

        StatusArrived(Current.X,Port.XValue.ToString());
                                                }
                                                else
                                                {
                                                    /// Incremented
Coordinate coding
                                                }
                                            }
                                            #endregion
                                            #region Y in RapidMovement
                                            if(lGMCode.StartsWith("Y"))
                                            {

        if(Port.AbsoluteCoordinate)
                                                {
                                                    Port.YValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        if(Port.CurrentPositionY > Port.XValue)
                                                    {

        Port.PulsRequestY(Port.CurrentPositionY - Port.YValue);

        Port.Yplus();

        PositionArrived("Y-",Port.CurrentPositionY - Port.YValue);
                                                    }
                                                    else
                                                    {

        Port.PulsRequestY(Port.YValue - Port.CurrentPositionY);

        Port.Yminus();

        PositionArrived("Y+",Port.YValue - Port.CurrentPositionY);
                                                    }

        Port.CurrentPositionY = Port.YValue;

        StatusArrived(Current.Y,Port.YValue.ToString());
                                                }
                                                else
                                                {
                                                    /// Incremented
Coordinate coding
                                                }
                                            }
```

```
                                        #endregion
                                        #region Z in RapidMovement
                                        if(lGMCode.StartsWith("Z"))
                                        {

        if(Port.AbsoluteCoordinate)
                                                {
                                                        Port.ZValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        if(Port.CurrentPositionZ > Port.ZValue)
                                                        {

        Port.PulsRequestZ(Port.CurrentPositionZ - Port.ZValue);

        Port.Zplus();

        if(Port.ZValue == 0)

        PositionArrived("Z-",0);
                                                                        else

        PositionArrived("Z-",Port.CurrentPositionZ - Port.ZValue);
                                                        }
                                                        else
                                                        {


        Port.PulsRequestZ(Port.ZValue - Port.CurrentPositionZ);

        Port.Zminus();

        if(Port.ZValue == 0)

        PositionArrived("Z+",0);
                                                                        else

        PositionArrived("Z+",Port.ZValue - Port.CurrentPositionZ);
                                                        }


        Port.CurrentPositionZ = Port.ZValue;

        StatusArrived(Current.Z,Port.ZValue.ToString());
                                                }
                                                else
                                                {
                                                        /// Incremented
Coordinate coding
                                                }
                                        }
                                        #endregion
                                }/// End of if Port.RapidMovement
                                #endregion
                                #region LinearInterPolation Handling
Code
                                /// <summary>
```

```
                                /// In this region We Handle Linear
InterPolation
                                /// And check X and Y coordinate
togather and Z Coordinate Separately
                                /// </summary>
                                if(Port.LinearInterPolation)
                                {
                                        if(lGMCode.StartsWith("X"))
                                        {/// GMCode first letter is X

        if(Port.AbsoluteCoordinate)
                                                {
                                                        ///take the X
Value
                                                        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
                                                        /// we are now in
Linear Interpolation so we need
                                                        /// Y coordinate
with X coordinate.
                                                        /// check there
is Y coordinate after X coordinate
                                                        /// as the rules
of EIA274D standard

        if(aList[i+1].ToString().StartsWith("Y"))
                                                        { /// if we get Y
coordinate after X coordinate
                                                                i++;
                                                                /// take
the value of Y

        Port.YValue =
Convert.ToDouble(aList[i].ToString().Substring(1,aList[i].ToString().
Length - 1));

        Port.NextPositionX = Port.XValue;

        Port.NextPositionY = Port.YValue;

        Port.CalculateLinearInterpolation(Port.CurrentPositionX,Port.
CurrentPositionY,Port.NextPositionX,Port.NextPositionY);

        AddLine(Port.CurrentPositionX,Port.CurrentPositionY,Port.Next
PositionX,Port.NextPositionY);


        StatusArrived(Current.X,Port.XValue.ToString());

        StatusArrived(Current.Y,Port.YValue.ToString());

        if(Port.NextPositionX != Port.CurrentPositionX)
                                                                {

        if(Port.NextPositionX > Port.CurrentPositionX)

        PositionArrived("X+",Port.NextPositionX -
Port.CurrentPositionX);
```

```
        else

        PositionArrived("X-",Port.CurrentPositionX -
Port.NextPositionX);
                                                                }

        if(Port.NextPositionY != Port.CurrentPositionY)
                                                                {

        if(Port.NextPositionY > Port.CurrentPositionY)

        PositionArrived("Y+",Port.NextPositionY -
Port.CurrentPositionY);

        else

        PositionArrived("Y-",Port.CurrentPositionY  -
Port.NextPositionY);
                                                                }


        Port.CurrentPositionX = Port.NextPositionX;

        Port.CurrentPositionY = Port.NextPositionY;
                                                        }

                                                }
                                                else
                                                {
                                                        /// Incremented
Coordinate coding
                                                }
                                        }
                                        if(lGMCode.StartsWith("Z"))
                                        {

        if(Port.AbsoluteCoordinate)
                                                {
                                                        Port.ZValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        Port.NextPositionZ = Port.ZValue;

        if(Port.NextPositionZ > Port.CurrentPositionZ)
                                                        {

        Port.PulsRequestZ(Port.NextPositionZ -
Port.CurrentPositionZ);

        Port.Zminus();

        StatusArrived(Current.Z,Port.ZValue.ToString());

        PositionArrived("Z+",Port.NextPositionZ -
Port.CurrentPositionZ);

        Port.CurrentPositionZ = Port.NextPositionZ;
```

```
                                                        }
                                                        else

        if(Port.NextPositionZ < Port.CurrentPositionZ)
                                                        {

        Port.PulsRequestZ(Port.CurrentPositionZ -
Port.NextPositionZ);

        Port.Zplus();

        StatusArrived(Current.Z,Port.ZValue.ToString());

        PositionArrived("Z-",Port.CurrentPositionZ -
Port.NextPositionZ);

        Port.CurrentPositionZ = Port.NextPositionZ;
                                                        }

                                                }
                                        }
                                }/// end of if Port.LinearInterPolation
                                #endregion
                                #region Parabolic Interpolation
Handling Code
                                if(Port.ParabolicInterpolation)
                                {
                                        if(Port.AbsoluteCoordinate)
                                        {

        if(lGMCode.StartsWith("X") &&
aList[i+1].ToString().StartsWith("Y"))
                                                {

        if(aList[i+2].ToString().StartsWith("I") &&
aList[i+3].ToString().StartsWith("J"))
                                                        {

        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        Port.YValue =
Convert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToStrin
g().Length - 1));
                                                                Port.I =
Convert.ToDouble(aList[i+2].ToString().Substring(1,aList[i+2].ToStrin
g().Length - 1));
                                                                Port.J =
Convert.ToDouble(aList[i+3].ToString().Substring(1,aList[i+3].ToStrin
g().Length - 1));
                                                                i = i + 3;

                                                        }
                                                        else
                                                        {

        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));
```

```
        Port.YValue =
Convert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToStrin
g().Length - 1));
                                                                i = i + 1;
                                                                if(Port.I
> 0 || Port.J > 0)
                                                                {

        Port.I = Port.I;

        Port.J = Port.J;
                                                                }
                                                                else
                                                                {

        Port.I = 0;

        Port.J = 0;
                                                                }
                                                        }


        Port.NextPositionY = Port.YValue;

        Port.NextPositionX = Port.XValue;
                                                        //Port.Radius =
Math.Sqrt(((Port.CurrentPositionX - Port.I) * (Port.CurrentPositionX
- Port.I)) + ((Port.CurrentPositionY - Port.J) *
(Port.CurrentPositionY - Port.J)));

        Port.CalculateParabolicInterpolation();

        StatusArrived(Current.X,Port.XValue.ToString());

        StatusArrived(Current.Y,Port.YValue.ToString());

        if(Port.NextPositionX != Port.CurrentPositionX)
                                                        {

        if(Port.NextPositionX > Port.CurrentPositionX)

        PositionArrived("X+",Port.NextPositionX -
Port.CurrentPositionX);
                                                                else

        PositionArrived("X-",Port.CurrentPositionX -
Port.NextPositionX);
                                                        }

        if(Port.NextPositionY != Port.CurrentPositionY)
                                                        {

        if(Port.NextPositionY > Port.CurrentPositionY)

        PositionArrived("Y+",Port.NextPositionY -
Port.CurrentPositionY);
                                                                else
```

```
        PositionArrived("Y-",Port.CurrentPositionY -
Port.NextPositionY);
                                                    }

        Port.CurrentPositionX = Port.NextPositionX;

        Port.CurrentPositionY = Port.NextPositionY;
                                                }
                                        }
                                    }/// end of if
Port.CircularInterpolationCounterClockWise
                                    #endregion
                                    #region Circular Interpolation Handling
Code

        if(Port.CircularInterpolationCounterClockWise)
                                    {
                                            if(Port.AbsoluteCoordinate)
                                            {

        if(lGMCode.StartsWith("X") &&
aList[i+1].ToString().StartsWith("Y"))
                                                {

        if(aList[i+2].ToString().StartsWith("I") &&
aList[i+3].ToString().StartsWith("J"))
                                                    {

        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        Port.YValue =
Convert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToStrin
g().Length - 1));
                                                                Port.I =
Convert.ToDouble(aList[i+2].ToString().Substring(1,aList[i+2].ToStrin
g().Length - 1));
                                                                Port.J =
Convert.ToDouble(aList[i+3].ToString().Substring(1,aList[i+3].ToStrin
g().Length - 1));
                                                                i = i + 3;

                                                    }
                                                    else
                                                    {

        Port.XValue =
Convert.ToDouble(lGMCode.Substring(1,lGMCode.Length - 1));

        Port.YValue =
Convert.ToDouble(aList[i+1].ToString().Substring(1,aList[i+1].ToStrin
g().Length - 1));
                                                                i = i + 1;
                                                                if(Port.I
> 0 || Port.J > 0)
                                                                {
```

```
        Port.I = Port.I;

        Port.J = Port.J;
                                                                }
                                                                else
                                                                {

        Port.I = 0;

        Port.J = 0;
                                                                }
                                                        }


        Port.NextPositionY = Port.YValue;

        Port.NextPositionX = Port.XValue;
                                                        Port.Radius =
Math.Sqrt(((Port.CurrentPositionX - Port.I) * (Port.CurrentPositionX
- Port.I)) + ((Port.CurrentPositionY - Port.J) *
(Port.CurrentPositionY - Port.J)));

        Port.CalculateCircularInterpolation();

        StatusArrived(Current.X,Port.XValue.ToString());

        StatusArrived(Current.Y,Port.YValue.ToString());

        if(Port.NextPositionX != Port.CurrentPositionX)
                                                        {

        if(Port.NextPositionX > Port.CurrentPositionX)

        PositionArrived("X+",Port.NextPositionX -
Port.CurrentPositionX);
                                                                else

        PositionArrived("X-",Port.CurrentPositionX -
Port.NextPositionX);
                                                        }

        if(Port.NextPositionY != Port.CurrentPositionY)
                                                        {

        if(Port.NextPositionY > Port.CurrentPositionY)

        PositionArrived("Y+",Port.NextPositionY -
Port.CurrentPositionY);
                                                                else

        PositionArrived("Y-",Port.CurrentPositionY -
Port.NextPositionY);
                                                        }

        Port.CurrentPositionX = Port.NextPositionX;

        Port.CurrentPositionY = Port.NextPositionY;
```

```
                                        }
                                    }
                                }/// end of if
Port.CircularInterpolationCounterClockWise
                                #endregion

                        }
                        status = MachineStatus.Free;
                        if(WorkCompleted != null)
                                WorkCompleted();
                }
```

5.7.2 *Developing AR for Turning Operation*

CNC turning classes consist of three projects.

1. Turning: Consists of object used in CNC turning software such as PortAccess and Turning, two enumerations and five events. Turning is a dynamic link library (DLL)-based project. The product DLL of turning will be used in CNC turning application to provide separation between application and logic, and logic encapsulation.
   Class structure of turning: Turning consists of two main classes-turning class and PortAccess class. Turning class has encapsulated all the main functionality required by CNC turning software such as Open File, Save File, Execute File, Validate File and Validate Code. A request from CNC turning software for each function is handled by turning class. The enumerations of turning are Current and MachineStatus. Current enumeration identifies the current value of different parameter and MachineStatus identifies the machine running status.
   To provide access to physical port on the system, turning has PortAccess class. PortAccess class has encapsulated port access and interpolations function. Only turning class has access to PortAccess class and CNC turning software does not have this facility. Turning class calls appropriate interpolation methods of PortAccess which in turn calls private methods of PortAccess to send and receive physical port signal.
   In order to update the CNC turning software about the current state of execution, turning has five events: SpindleHandler, MessageHandler, PositionHandler, StatusHandler, LineEventHandler, and Completed. SpindleHandler event is used to show the spindle status as ON or OFF. MessageHandler is used to show the messages on information message box. PositionHandler is used to show the current position of work piece in x, y, and z-axes. StatusHandler is used to show the current status of turning machine. LineEventHandler is used to draw line on work piece in order to show the cut. Information received from events by CNC turning software is used to update user interface and to control 3D graphics.
2. CNC turning: Consists of CNC turning graphical user interface that utilizes turning object to execute file and related functions. Current status of execution is also updated from CNC turning.

Table 5.2 Class members of MillingPortAccess

Class summary			
Class ID: CM1			Namespace: MillingMachine
Class name: MillingPortAccess			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in *x*-axis
_CurrentPositionY	Double	Private	Holds the current position of work piece in *y*-axis
_CurrentPositionZ	Double	Private	Holds the current position of work piece in z-axis
_Radius	Double	Private	Holds the radius of the circular movement
_I	Double	Private	Holds the starting position in *x*-axis
_J	Double	Private	Holds the starting position in *y*-axis
_NextPositionX	Double	Private	Holds the next position of work piece in *x*-axis
_NextPositionY	Double	Private	Holds the next position of work piece in *y*-axis
_NextPositionZ	Double	Private	Holds the next position of work piece in *z*-axis
_*x*value	Double	Private	Holds the *X* value
_*y*value	Double	Private	Holds the *Y* value
_*z*value	Double	Private	Holds the *Z* value
_A	Double	Private	Holds the parabolic interpolation's intermediate points
Parabola	String	Private	Store the type of parabola
_RapidMovement	Bool	Private	Holds the boolean value to rapid movement at a machine tool used for positioning the tool
_LinearInterpolation	Bool	Private	Holds the boolean value to allow direct movement between two points through linear interpolation
_CircularInterpolation ClockWise	Bool	Private	Holds the boolean value for clockwise circular interpolation
_CircularInterpolation CounterClockWise	Bool	Private	Holds the boolean value for anticlockwise circular interpolation
_ParabolicInterpolation	Bool	Private	Holds the boolean value for parabolic interpolation
bMetricUnit	Bool	Private	Holds the boolean value for to check whether the unit is set to metric system

(continued)

Table 5.2 (continued)

Field name	Data type	Access modifier	Description
bAbsoluteCoordinate	Bool	Private	Holds the boolean value for to check whether the coordinate system is set to absolute coordinate
_CutterCompensation Cancel	Bool	Private	Holds the boolean value to set the cutter compensation cancel
_CutterCompensationLeft	Bool	Private	Holds the boolean value to set cutter compensation to left
_CutterCompensationRight	Bool	Private	Holds the boolean value to set cutter compensation to right
_CutterOffsetInsideCorner	Bool	Public	Holds the boolean value to set cutter offset inside corner
_CutterOffsetOutside Corner	Bool	Private	Holds the boolean value to set cutter offset inside corner
_Speed	Short	Private	Holds the value to control spindle speed
_Feed	Short	Private	Holds the feed of the spindle
_Tool	Short	Private	Holds the tool of the spindle
xPulses	Int	Private	Sets the pulse of the spindle in *x*-axis
yPulses	Int	Private	Sets the pulse of the spindle in *y*-axis
zPulses	Int	Private	Sets the pulse of the spindle in z-axis
_signflg	Int	Private	Holds the sign flag
_signflgy	Int	Private	Holds the sign flag in *y*-axis
_signflgx	Int	Private	Holds the sign flag in *x*-axis
_chksign	String	Private	Holds the chksign string to display
_chksignx	String	Private	Holds the chksign in *x*-axis string to display
_chksigny	String	Private	Holds the chksign in *y*-axis string to display
_xyzs	String	Private	Holds the XYZ coordinate string to display
_yxzs	String	Private	Holds the YXZ coordinate string to display
EmergencyStop	Static bool	Private	Holds the boolean value to control the machine running status

Method name	Return type	Access modifier	Description
CalculateLinearInterpolation (double X1, double Y1, double X2, double Y2)	Void	Public	Function to calculate linear interpolation
CalculateCircularInterpolation()	Void	Public	Function to calculate circular interpolation
CalculateParabolicInterpolation()	Void	Public	Function to calculate parabolic interpolation
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
PulsRequestZ(double Z)	Void	Public	Function to set the zPulses

(continued)

Table 5.2 (continued)

Method name	Return type	Access modifier	Description
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to generate pulses for physical port to control movement in Y plus axis
Yminus()	Void	Public	Function to decrease the yPulses
Zplus()	Void	Public	Function to increase the zPulses
Zminus()	Void	Public	Function to decrease the zPulses
G00()	Void	Public	Function to generate point-to-point positioning
G01()	Void	Public	Function to generate linear interpolation
G02()	Void	Public	Function to generate arc clockwise (2D)
G03()	Void	Public	Function to generate arc counterclockwise(2D)
G04()	Void	Public	Function to generate dwell
G06()	Void	Public	Function to generate parabolic interpolation
G17()	Void	Public	Function to generate plane selection
G18()	Void	Public	Function to generate plane selection
G19()	Void	Public	Function to generate plane selection
G33()	Void	Public	Function to generate thread cutting, constant lead
G34()	Void	Public	Function to generate thread cutting, increasing lead
G35()	Void	Public	Function to generate thread cutting, decreasing lead
G40()	Void	Public	Function to generate cutter compensation/offset cancel
G41()	Void	Public	Sets machine mode to ON for cutter compensation left
G42()	Void	Public	Sets machine mode to ON for cutter compensation—right
G43()	Void	Public	Sets machine mode to ON for cutter offset-inside corner
G44()	Void	Public	Sets machine mode to ON for cutter offset-outside corner
G70()	Void	Public	Sets machine mode for inch programming
G71()	Void	Public	Sets machine mode for metric programming
G90()	Void	Public	Sets machine mode for absolute input
G91()	Void	Public	Sets machine mode for incremental input
M00()	Void	Public	Sets machine mode for program stop

Table 5.2 (continued)

Method name	Return type	Access modifier	Description
M03()	Void	Public	Sets machine mode for spindle CW
M04()	Void	Public	Sets machine mode for spindle CCW
M05()	Void	Public	Sets machine mode for spindle off
M06()	Void	Public	Sets machine mode for tool change
M07()	Void	Public	Sets machine mode for coolant on—mist
M08()	Void	Public	Sets machine mode for coolant on—flood
M09()	Void	Public	Sets machine mode for coolant off
M10()	Void	Public	Sets machine mode for clamp
M11()	Void	Public	Sets machine mode for unclamp
Dwell()	Void	Public	Sets machine mode for timed delay of established duration

Event name	Delegate	Access modifier	Description
AddInnerLine	LineEventDelegate	Public	Event to add inner line to the work piece

Table 5.3 Milling class members

Class summary			
Class ID: CM2			Namespace: MillingMachine
Class name: Milling			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with:None

Field name	Data type	Access modifier	Description
sFileName	Static string	Public	Holds the file name of .cnc file
aList	Static ArrayList	Private	Holds the GM codes extracted from .cnc file
GMCodes	Static ArrayList	Private	Holds all GM codes
Component	Component	Private	Holds other managed resource this class uses
Disposed	Bool	Private	Tracks whether dispose has been called
Port	MillingPort Access	Private	Used to call the port functions for sending GM codes
Status	MachineStatus	Private	Holds and sets the machine status
sMachineName	String	Private	Holds the machine name

Method name	Return type	Access modifier	Description
OpenFile(TextBox textBox)	Void	Public	Function used for open .cnc file and display in textbox object based in this function
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	Static void	Public	Function to write errors in error log

(continued)

Table 5.3 (continued)

Method name	Return type	Access modifier	Description
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute
ValidateCode(string sGMCode,ref bool bIsGMCodeValid)	Void	Public	This function will validate the GMCode
ValidateFile()	Bool	Public	This function will validate the .cnc part programe file. It will take file name from static variable set by calling routine. If .cnc part programme file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This function will save the part program file. It will take file name from static variable set by calling routine

Event name	Delegate	Access modifier	Description
MessageArrived	MessageHandler	Public	Generates the event of message
StatusArrived	StatusHandler	Public	Generates the event of status
PositionArrived	PositionHandler	Public	Generates the event of current position
SpindleArrived	SpindleHandler	Public	Generates the event of spindle on or off
AddLine	LineEventDelegate	Public	Generates the event to add line
WorkCompleted	Completed	Public	Generates the event of work completed

Table 5.4 Milling3D class members

Class summary			
Class ID: CM3			Namespace: MillingAnimation
Class name: Milling3D			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
TV	TVEngine	Public	Declares the TrueVision engine object
Inp	TVInputEngine	Private	Declares the input engine variable
TVLightEngine	TVLightEngine	Private	Declares the light engine variable
MatFactory	TVMaterialFactory	Private	Declares the material factory variable
TVScene	TVScene	Private	Declares the TVScene object
Global	TVGlobals	Private	Declares the global variable
FactoryFloor	TVMeshClass	Private	Declares the mesh class for the factory floor
Light	D3DLIGHT8	Private	Declares the light variable
Lines	ArrayList	Private	Holds the list of lines
MWorkPiece	MillingWorkPiece	Private	Holds the object for work piece
sngPositionX	Float	Private	Holds object position in *x*-axis for keyboard input
sngPositionY	Float	Private	Holds object position in *y*-axis for keyboard input
sngPositionZ	Float	Private	Holds object position in *z*-axis for keyboard input
snglookatX	Float	Private	Holds camera position in *x*-axis for keyboard input
snglookatY	Float	Private	Holds camera position in *y*-axis for keyboard input
snglookatZ	Float	Private	Holds camera position in *z*-axis for keyboard input
tmpMouseX	Int	Private	Holds mouse input position in *x*-axis
tmpMouseY	Int	Private	Holds mouse input position in *y*-axis
tmpMouseB1	Short	Private	Holds mouse input button1
tmpMouseB2	Short	Private	Holds mouse input button2
tmpMouseB3	Short	Private	Holds mouse input button3
tmpMouseScrollOld	Int	Private	Holds mouse input scroll old value
tmpMouseScrollNew	Int	Private	Holds mouse input scroll new value
sngAngleX	Float	Private	Holds mouse input angle in *x*-axis
sngAngleY	Float	Private	Holds mouse input angle in *y*-axis
sngWalk	Float	Private	For smooth movement while camera is walking
sngStrafe	Float	Private	For smooth movement while camera is strafing
DoLoop	Bool	Private	Holds the boolean value true until the scene is rendering

(continued)

Table 5.4 (continued)

Field name	Data type	Access modifier	Description
floor_x	Float	Private	Holds initial position for factory floor in *x*-axis
floor_y	Float	Private	Holds initial position for factory floor in *y*-axis
floor_z	Float	Private	Holds Initial position for factory floor in *z*-axis
ModelPath	String	Private	Variable for storing model paths
UpperX	Float	Private	Holds initial position for the machine in *x*-axis
UpperY	Float	Private	Holds initial position for the machine in *y*-axis
UpperZ	Float	Private	Holds initial position for the machine in *z*-axis
WorkX	Float	Private	Holds initial position for the work piece in *x*-axis
WorkY	Float	Private	Holds initial position for the work piece in *y*-axis
WorkZ	Float	Private	Holds initial position for the work piece in *z*-axis
MeshMachine	TVMesh	Private	Holds the mesh for machine
MeshWork	TVMesh	Private	Holds the mesh for work piece
MeshWorkBase	TVMesh	Private	Holds the mesh for work piece base
MeshTemp	TVMesh	Private	Holds the mesh for temporary working piece
AnimationClass	TVKeyFrameAnim	Private	For animating the 3D mesh
WorkPosition	D3DVECTOR	Private	Holds current work piece position
gWorkPosition	D3DVECTOR	Private	Holds initial work piece position
WorkBasePosition	D3DVECTOR	Private	Holds current work base position
gWorkBasePosition	D3DVECTOR	Private	Holds initial work base position
MachineX	Float	Private	Holds initial position for the machine in *x*-axis
MachineY	Float	Private	Holds initial position for the machine in *y*-axis
MachineZ	Float	Private	Holds initial position for the machine in *z*-axis

Method Name	Return type	Access modifier	Description
ClearLines()	Void	Public	Function to clear lines on work piece
Move(Direction D)	String	Private	Function to define movement limits
MoveControl(double x1, double y1, double x2, double y2)	Void	Public	Function to add the line on the work piece
Move(string coordinate, int value)	Void	Public	Function to move the spindle in *x*, *y,* or *z*-axis

(continued)

Table 5.4 (continued)

Method Name	Return type	Access modifier	Description
SetWorkMesh()	Void	Public	Function to set the temporary working piece mesh position and attach it to the machine mesh
MoveY(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *y*-axis
ResetMachine()	Void	Public	Function to reset the work piece and base to initial position
MoveX(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *x*-axis
MoveZ(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *z*-axis
StartRendering()	Void	Public	Function to start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to set the DoLoop variable to false
Check_Movement()	Void	Public	Function to check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen

3. Turning machine animation: Consists of Turning3D class and three enumerations. Turning3D class consists of 3D graphic rendering engine and methods to control movement of individual parts of machine. Enumerations of turning machine animation are direction, spindle, and axis.

Table 5.12 shows the PortAccess class member properties, methods, and variables. PortAccess is the helper class of turning, and this class is used to pass pulses to CNC turning by using parallel port and calculate the pulses, liner interpolation, circular interpolation, parabolic interpolation. PortAccess also holds the current position of *x*, *y,* and *z*-axes and current state of the turning machine.

Table 5.13 explains the turning class members variables, properties, and methods. The turning class is used to read the part program file, validate it, open it, save it, and execute it. This class is also responsible for raising events to update user interface and 3D turning machine. Interpreter of CNC turning is also implemented in this class's ExecuteFile method.

Table 5.14 defines the Turning3D class member variables, properties, and methods. Turning3D class is used to render the graphics of turning machine in 3D format for display movement of machine and work piece. 3D rendering engine is also implemented in this class.

Table 5.5 Summary of CNC milling sequence diagram

Sequence diagram summary				
Sequence ID: SD-M1 Sequence name: CNC milling sequence diagram				
Method name	Type	Description	Source	Destination
LoadPartProgram()	Message	Function to load part program	User	FrmMain
OpenFile()	Return message	Function to open file on disk	FrmMain	Support
SavePartProgram()	Message	Function to save part program	User	FrmMain
ValidateCode()	Return message	Function to validate GM codes	FrmMain	Support
SaveFile()	Message	Function to save file on disk	FrmMain	Support
EditPartProgram()	Self-message	Function to edit part program	User	FrmMain
NewPartProgram()	Self-message	Function to new part program	User	FrmMain
ExecutePartProgram()	Message	Function to execute part program	User	FrmMain
ValidateFile()	Return message	Function to validate program file	FrmMain	Support
ExecuteFile()	Return message	Function to execute program file	FrmMain	Support
Dispose()	Message	Function to close rendering and exit	User	FrmFinal
CalculateLinearInterpolation()	Message	Function to calculate linear interpolation	Support	PortAccess
CalculateCircularInterpolation()	Message	Function to calculate circular interpolation	Support	PortAccess
CalculateParabolicInterpolation()	Message	Function to calculate parabolic interpolation	Support	PortAccess
PulsRequestX()	Message	Function to set the xPulses	Support	PortAccess
PulsRequestY()	Message	Function to set the yPulses	Support	PortAccess
PulsRequestZ()	Message	Function to set the zPulses	Support	PortAccess
Xplus()	Message	Function to increase the xPulses	Support	PortAccess
Xminus()	Message	Function to decrease the xPulses	Support	PortAccess
Yplus()	Message	Function to generate pulses for physical port to control movement in *Y* plus axis.	Support	PortAccess
Yminus()	Message	Function to decrease the yPulses	Support	PortAccess
Zplus()	Message	Function to increase the zPulses	Support	PortAccess

(continued)

Table 5.5 (continued)

Method name	Type	Description	Source	Destination
Zminus()	Message	Function to decrease the zPulses	Support	PortAccess
G00()	Message	Function to generate point-to-point positioning	Support	PortAccess
G01()	Message	Function to generate linear interpolation	Support	PortAccess
G02()	Message	Function to generate arc clockwise (2D)	Support	PortAccess
G03()	Message	Function to generate arc counterclockwise(2D)	Support	PortAccess
G04()	Message	Function to generate dwell	Support	PortAccess
G06()	Message	Function to generate parabolic interpolation	Support	PortAccess
G17()	Message	Function to generate plane selection	Support	PortAccess
G18()	Message	Function to generate plane selection	Support	PortAccess
G19()	Message	Function to generate plane selection	Support	PortAccess
G33()	Message	Function to generate thread cutting, constant lead	Support	PortAccess
G34()	Message	Function to generate thread cutting, increasing lead	Support	PortAccess
G35()	Message	Function to generate thread cutting, decreasing lead	Support	PortAccess
G40()	Message	Function to generate cutter compensation/offset cancel	Support	PortAccess
G41()	Message	Sets machine mode to ON for cutter compensation left	Support	PortAccess
G42()	Message	Sets machine mode to ON for cutter compensation—right	Support	PortAccess
G43()	Message	Sets machine mode to ON for cutter offset-inside corner	Support	PortAccess
G44()	Message	Sets machine mode to ON for cutter offset-outside corner	Support	PortAccess
G70()	Message	Sets machine mode for inch programming	Support	PortAccess
G71()	Message	Sets machine mode for metric programming	Support	PortAccess
G90()	Message	Sets machine mode for absolute input	Support	PortAccess
G91()	Message	Sets machine mode for incremental input	Support	PortAccess
M00()	Message	Sets machine mode for program stop	Support	PortAccess
M03()	Message	Sets machine mode for spindle CW	Support	PortAccess
M04()	Message	Sets machine mode for spindle CCW	Support	PortAccess
M05()	Message	Sets machine mode for spindle off	Support	PortAccess

(continued)

Table 5.5 (continued)

Method name	Type	Description	Source	Destination
M06()	Message	Sets machine mode for tool change	Support	PortAccess
M07()	Message	Sets machine mode for coolant on—mist	Support	PortAccess
M08()	Message	Sets machine mode for coolant on—flood	Support	PortAccess
M09()	Message	Sets machine mode for coolant off	Support	PortAccess
M10()	Message	Sets machine mode for clamp	Support	PortAccess
M11()	Message	Sets machine mode for unclamp	Support	PortAccess
Dwell()	Message	Sets machine mode for timed delay of established duration	Support	PortAccess
AddLine()	Message	Function to add line to the final output	PortAccess	frmFinal
Show()	Message	Function to show display	PortAccess	frmFinal

Table 5.6 Use case summary load part program in CNC milling

Use case summary	
Use case ID:	UC-M1
Use case name:	Load part program
Actors:	User
Description:	The user selects and loads part program through a file with "cnc" extension
Preconditions:	Part program file is present on the system
Post conditions:	Part program is loaded on the screen
System parameters	Part program file
Success scenario	Part program is loaded successfully
Alternative scenario	Error in loading part program
Includes:	Open file
Priority:	High

Table 5.7 Summary of use case save part program in CNC milling

Use case summary	
Use case ID:	UC-M2
Use case name:	Save part program
Actors:	User
Description:	The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions:	None
Post conditions:	Part program is saved on the file
System parameters	Part program with GM codes
Success scenario	Part program is validated and saved successfully
Alternative scenario	Error in saving part program
Includes:	Validate file, save file
Priority:	High

Table 5.8 Summary of use case execute part program in CNC milling

Use case summary	
Use case ID:	UC-M3
Use case name:	Execute part program
Actors:	User
Description:	The user executes the part program after loading it on the screen
Preconditions:	Part program is loaded and validated
Post conditions:	Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in *x*, *y,* and *z*-axes
System parameters	Part program with GM codes
Success scenario	Part program is executed successfully
Alternative scenario	Error in executing part program
Includes:	Validate file, execute file
Priority:	High

Table 5.9 Summary of use case Edit part program in CNC milling

Use case summary	
Use case ID:	UC-M4
Use case name:	Edit part program
Actors:	User
Description:	The user edits the part program which is already saved
Preconditions:	Part program is selected and loaded on screen
Post conditions:	Part program is modified and saved after validation
System parameters	Part program file
Success scenario	Part program is edited successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.10 Summary of use case new part program in CNC milling

Use case summary	
Use case ID:	UC-M5
Use case name:	New part program
Actors:	User
Description:	The user inputs a new part program on the screen
Preconditions:	None
Post conditions:	Part program is validated and saved
System parameters	Part program with GM codes
Success scenario	New part program file is created successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.11 Summary of CNC milling flow charts

Flow chart summary	
Flow chart ID:	FC-M1
Flow chart name:	CNC milling process flow chart
No. of inputs:	7
No. of processes:	15
No. of outputs:	3
No. of control decisions:	49
Input name	Description
Save program	Save button on screen
Run program	Run button on screen
Load program	Load button on screen
New program	New button on screen

(continued)

Table 5.11 (continued)

Flow chart summary	
Edit program	Edit button on screen
Reset	Reset button on screen
Exit application	Exit button on screen
Process name	Description
Load part program from file	Function to load part program
Save part program	Function to save part program
Open dialog box	Function to open file on disk
New part program	Function to new part program
Edit part program	Function to edit part program
Reset machine	Function to reset the machine to initial position
End application	Function to close rendering and exit
Save file	Function to save file on disk
Execute part program	Function to execute program file
Set machine	Function to set machine position
Validate file	Function to validate program file
Execute file	Function to execute part program
All GM codes Loop	Loop to check the GM code
Message arrived	Event to check a message
Status arrived	Event to check a status
Output name	Description
Part program display	Part program loaded on screen
Final output display	Work piece display on screen
Machine position display	Machine position values on screen
Control decision name	Description
Validate part program	Check to validate GM codes
Select part program	Select the program file
Save dialog	Save dialog box
Validated	Check whether file is validated
Emergency stop	Stop rendering and reset
CalculateLinearInterpolation	Check to calculate linear interpolation
CalculateCircularInterpolation	Check to calculate circular interpolation
CalculateParabolicInterpolation	Check to calculate parabolic interpolation
PulsRequestX	Check to set the xPulses
PulsRequestY	Check to set the yPulses
PulsRequestZ	Check to set the zPulses
Xplus	Check to increase the xPulses
Xminus	Check to decrease the xPulses
Yplus	Check to increase the yPulses
Yminus	Check to decrease the yPulses
Zplus	Check to increase the zPulses
Zminus	Check to decrease the zPulses
G00	Point-to-point positioning
G01	Linear interpolation
G02	Arc clockwise (2D)
G03	Arc counterclockwise(2D)

(continued)

Table 5.11 (continued)

Flow chart summary	
G04	Dwell
G06	Parabolic interpolation
G17	Plane selection
G18	Plane selection
G19	Plane selection
G33	Thread cutting, constant lead
G34	Thread cutting, increasing lead
G35	Thread cutting, decreasing lead
G40	Cutter compensation/offset cancel
G41	Cutter compensation—left
G42	Cutter compensation—right
G43	Cutter offset-inside corner
G44	Cutter offset-outside corner
G70	Inch programming
G71	Metric programming
G90	Absolute input
G91	Incremental input
M00	Program stop
M03	Spindle CW
M04	Spindle CCW
M05	Spindle off
M06	Tool change
M07	Coolant on—mist
M08	Coolant on—flood
M09	Coolant off
M10	Clamp
M11	Unclamp
Dwell	A timed delay of established duration

Table 5.15 displays the summary of the CNC turning sequence diagram.

Table 5.16 presents the summary of the CNC turning load part program use case.

Table 5.17 describes the summary of the CNC turning save part program use case.

Table 5.18 explains the summary of the CNC turning execute part program use case.

Table 5.19 depicts the summary of the CNC turning edit part program use case.

Table 5.20 defines the summary of the CNC turning new part program use case.

Table 5.21 illustrates the summary of the CNC turning flow charts.

Table 5.12 Summary of turning PortAccess class

Class summary			
Class ID: CT1		Namespace: TurningMachine	
Class name: PortAccess		Class type: Concrete	
Is base class? No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in *x*-axis
_CurrentPositionZ	Double	Private	Holds the current position of work piece in *z*-axis
_Radius	Double	Private	Holds the radius of the circular movement
_I	Double	Private	Holds the starting position in *x*-axis
_K	Double	Private	Holds the starting position in *z*-axis
_NextPositionX	Double	Private	Holds the next position of work piece in *x*-axis
_NextPositionZ	Double	Private	Holds the next position of work piece in *z*-axis
_xvalue	Double	Private	Holds the *X* value
_zvalue	Double	Private	Holds the *Z* value
Parabola	String	Private	Stores the type of parabola
_RapidMovement	Bool	Private	Holds the boolean value to rapid movement at a machine tool used for positioning the tool
_LinearInterpolation	Bool	Private	Holds the boolean value to allow direct movement between two points through linear interpolation
_CircularInterpolation ClockWise	Bool	Private	Holds the boolean value for clockwise circular interpolation
_CircularInterpolation CounterClockWise	Bool	Private	Holds the boolean value for anticlockwise circular interpolation
_ParabolicInterpolation	Bool	Private	Holds the boolean value for parabolic interpolation
bMetricUnit	Bool	Private	Holds the boolean value for to check whether the unit is set to metric system
bAbsoluteCoordinate	Bool	Private	Holds the boolean value for to check whether the coordinate system is set to absolute coordinate
_Speed	Short	Private	Holds the value to control spindle speed
_Feed	Short	Private	Holds the feed of the spindle

(continued)

Table 5.12 (continued)

Field name	Data type	Access modifier	Description
_Tool	Short	Private	Holds the tool of the spindle
xPulses	Int	Private	Sets the pulse of the spindle in *x*-axis
zPulses	Int	Private	Sets the pulse of the spindle in *z*-axis
EmergencyStop	Static bool	Private	Holds the boolean value to control the machine running status

Method name	Return type	Access modifier	Description
CalculateLinearInterpolation (double X1, double Y1, double X2, double Y2)	Void	Public	Function to calculate linear interpolation
CalculateCircularInterpolation()	Void	Public	Function to calculate circular interpolation
CalculateParabolicInterpolation()	Void	Public	Function to calculate parabolic interpolation
Delay()	Void	Public	Function to cause a delay of defined no. of cycles
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestZ(double Z)	Void	Public	Function to set the zPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Zplus()	Void	Public	Function to increase the zPulses
Zminus()	Void	Public	Function to decrease the zPulses
G00()	Void	Public	Function to generate point-to-point positioning
G01()	Void	Public	Function to generate linear interpolation
G02()	Void	Public	Function to generate arc clockwise (2D)
G03()	Void	Public	Function to generate arc counterclockwise(2D)
G06()	Void	Public	Function to generate parabolic interpolation
G70()	Void	Public	Sets machine mode for inch programming
G71()	Void	Public	Sets machine mode for metric programming
G90()	Void	Public	Sets machine mode for absolute input
G91()	Void	Public	Sets machine mode for incremental input
M02()	Void	Public	Sets machine mode for program stop
M03()	Void	Public	Sets machine mode for spindle CW
M04()	Void	Public	Sets machine mode for spindle CCW
M05()	Void	Public	Sets machine mode for spindle off
M06()	Void	Public	Sets machine mode for tool change
M07()	Void	Public	Sets machine mode for coolant on—mist
M08()	Void	Public	Sets machine mode for coolant on—flood
M09()	Void	Public	Sets machine mode for coolant off
M10()	Void	Public	Sets machine mode for clamp
M11()	Void	Public	Sets machine mode for unclamp

Event name	Delegate	Access modifier	Description
AddInnerLine	LineEventDelegate	Public	Event to add inner line to the work piece

Table 5.13 Summary of turning class

Class summary			
Class ID: CT2			Namespace: TurningMachine
Class name: Turning			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with: None
Field name	Data type	Access modifier	Description
sFileName	Static string	Public	Holds the file name of .cnc file
aList	static ArrayList	Private	Holds the GM codes extracted from .cnc file
GMCodes	static ArrayList	Private	Holds all GM codes
Component	Component	Private	Holds other managed resource this class uses
Disposed	Bool	Private	Tracks whether dispose has been called
Port	PortAccess	Private	Used to call the port functions for sending GM codes
Status	MachineStatus	Private	Holds and sets the machine status
sMachineName	String	Private	Holds the machine name
Method Name	Return type	Access modifier	Description
Dispose()	Void	Private	Function to dispose the rendering
EmergencyStop()	Void	Public	Function to stop the rendering if emergency stop is pressed
OpenFile(TextBox textBox)	Void	Public	Function used for open .cnc file and display in textbox object based on this function
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	Static void	Public	Function to write errors in error log
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute

(continued)

Table 5.13 (continued)

Field name	Data type	Access modifier	Description
Turning()	Void	Public	This function activates the machine
ValidateCode(string sGMCode,ref bool bIsGMCodeValid)	Void	Public	This function will validate the GMCode
ValidateFile()	Bool	Public	This function will validate the cnc part programe file. It will take file name from static variable set by calling routine. if cnc part programe file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This function will save the part program file. It will take file name from static variable set by calling routine

Event name	Delegate	Access modifier	Description
MessageArrived	MessageHandler	Public	Generates the event of message
StatusArrived	StatusHandler	Public	Generates the event of status
PositionArrived	PositionHandler	Public	Generate the event of current position
SpindleArrived	SpindleHandler	Public	Generates the event of spindle on or off
AddLine	LineEventDelegate	Public	Generates the event to add line
WorkCompleted	Completed	Public	Generates the event of work completed

Table 5.14 Summary of Turning3D class

Class summary			
Class ID: CT3			Namespace: TurningMachineAnimation
Class name: Turning3D			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with:None
Field name	Data type	Access modifier	Description
TV	TVEngine	Public	Declares the TrueVision engine object
Inp	TVInputEngine	Private	Declares the input engine variable
TVLightEngine	TVLightEngine	Private	Declares the light engine variable
MatFactory	TVMaterialFactory	Private	Declares the material factory variable
TVScene	TVScene	Private	Declares the TVScene object
Global	TVGlobals	Private	Declares the global variable
FactoryFloor	TVMeshClass	Private	Declares the mesh class for the factory floor
Light	D3DLIGHT8	Private	Declares the light variable
Lines	ArrayList	Private	Holds the list of lines
MWorkPiece	MillingWorkPiece	Private	Holds the object for work piece
sngPositionX	Float	Private	Holds object position in *x*-axis for keyboard input
sngPositionY	Float	Private	Holds object position in *y*-axis for keyboard input
sngPositionZ	Float	Private	Holds object position in *z*-axis for keyboard input
snglookatX	Float	Private	Holds camera position in *x*-axis for keyboard input
snglookatY	Float	Private	Holds camera position in *y*-axis for keyboard input
snglookatZ	Float	Private	Holds camera position in *z*-axis for keyboard input
tmpMouseX	Int	Private	Holds mouse input position in *x*-axis
tmpMouseY	Int	Private	Holds mouse input position in *y*-axis
tmpMouseB1	Short	Private	Holds mouse input button1
tmpMouseB2	Short	Private	Holds mouse input button2

(continued)

Table 5.14 (continued)

Field name	Data type	Access modifier	Description
tmpMouseB3	Short	Private	Holds mouse input button3
tmpMouseScrollOld	Int	Private	Holds mouse input scroll old value
tmpMouseScrollNew	Int	Private	Holds mouse input scroll new value
sngAngleX	Float	Private	Holds mouse input angle in *x*-axis
sngAngleY	Float	Private	Holds mouse input angle in *y*-axis
sngWalk	Float	Private	For smooth movement while camera is walking
sngStrafe	Float	Private	For smooth movement while camera is strafing
DoLoop	Bool	Private	Holds the boolean value true until the scene is rendering
floor_x	Float	Private	Holds initial position for factory floor in *x*-axis
floor_y	Float	Private	Holds initial position for factory floor in *y*-axis
floor_z	Float	Private	Holds initial position for factory floor in *z*-axis
ModelPath	String	Private	Variable for storing model paths
UpperX	Float	Private	Holds initial position for the machine in *x*-axis
UpperY	Float	Private	Holds initial position for the machine in *y*-axis
UpperZ	Float	Private	Holds initial position for the machine in *z*-axis
WorkX	Float	Private	Holds initial position for the work piece in *x*-axis
WorkY	Float	Private	Holds initial position for the work piece in *y*-axis
WorkZ	Float	Private	Holds initial position for the work piece in *z*-axis
MeshMachine	TVMesh	Private	Holds the mesh for machine
MeshWork	TVMesh	Private	Holds the mesh for work piece
MeshWorkBase	TVMesh	Private	Holds the mesh for work piece base
MeshTemp	TVMesh	Private	Holds the mesh for temporary working piece
AnimationClass	TVKeyFrameAnim	Private	For animating the 3D mesh
WorkPosition	D3DVECTOR	Private	Holds current work piece position
gWorkPosition	D3DVECTOR	Private	Holds initial work piece position

(continued)

Table 5.14 (continued)

Method name	Return type	Access modifier	Description
ClearLines()	Void	Public	Function to clear lines on work piece
Move(Direction D)	String	Private	Function to define movement limits
MoveControl(double x1, double y1, double x2, double y2)	Void	Public	Function to add the line on the work piece
Move(string coordinate, int value)	Void	Public	Function to move the spindle in *x*, *y*, or z-axis
SetWorkMesh()	Void	Public	Function to set the temporary working piece mesh position and attach it to the machine mesh
MoveY(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *y*-axis
ResetMachine()	Void	Public	Function to reset the work piece and base to initial position
MoveX(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *x*-axis
MoveZ(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *z*-axis
StartRendering()	Void	Public	Function to start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to set the DoLoop variable to false
Check_Movement()	Void	Public	Function to check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen

Table 5.15 Summary of CNC turning

Sequence diagram summary				
Sequence ID: SD-T1 Sequence name: CNC turning sequence diagram				
Method name	Type	Description	Source	Destination
LoadPartProgram()	Message	Function to load part program	User	FrmMain
OpenFile()	Return message	Function to open file on disk	FrmMain	Support
SavePartProgram()	Message	Function to save part program	User	FrmMain
ValidateCode()	Return message	Function to validate GM codes	FrmMain	Support
SaveFile()	Message	Function to save file on disk	FrmMain	Support
EditPartProgram()	Self-message	Function to edit part program	User	FrmMain
NewPartProgram()	Self-message	Function to new part program	User	FrmMain
ExecutePartProgram()	Message	Function to execute part program	User	FrmMain
ValidateFile()	Return message	Function to validate program file	FrmMain	Support
ExecuteFile()	Return message	Function to execute program file	FrmMain	Support
Dispose()	Message	Function to close rendering and exit	User	FrmFinal
CalculateLinearInterpolation()	Message	Function to calculate linear interpolation	Support	PortAccess
CalculateCircularInterpolation()	Message	Function to calculate circular interpolation	Support	PortAccess
CalculateParabolicInterpolation()	Message	Function to calculate parabolic interpolation	Support	PortAccess
Delay()	Message	Function to cause a delay of defined no. of cycles	Support	PortAccess
PulsRequestX()	Message	Function to set the xPulses	Support	PortAccess
PulsRequestZ()	Message	Function to set the zPulses	Support	PortAccess
Xplus()	Message	Function to increase the xPulses	Support	PortAccess
Xminus()	Message	Function to decrease the xPulses	Support	PortAccess
Zplus()	Message	Function to increase the zPulses	Support	PortAccess
Zminus()	Message	Function to decrease the zPulses	Support	PortAccess
G00()	Message	Function to generate point-to-point positioning	Support	PortAccess
G01()	Message	Function to generate linear interpolation	Support	PortAccess

(continued)

Table 5.15 (continued)

Method name	Type	Description	Source	Destination
G02()	Message	Function to generate arc clockwise (2D)	Support	PortAccess
G03()	Message	Function to generate arc counterclockwise (2D)	Support	PortAccess
G06()	Message	Function to generate parabolic interpolation	Support	PortAccess
G70()	Message	Sets machine mode for inch programming	Support	PortAccess
G71()	Message	Sets machine mode for metric programming	Support	PortAccess
G90()	Message	Sets machine mode for absolute input	Support	PortAccess
G91()	Message	Sets machine mode for incremental input	Support	PortAccess
M02()	Message	Sets machine mode for program stop	Support	PortAccess
M03()	Message	Sets machine mode for spindle CW	Support	PortAccess
M04()	Message	Sets machine mode for spindle CCW	Support	PortAccess
M05()	Message	Sets machine mode for spindle off	Support	PortAccess
M06()	Message	Sets machine mode for tool change	Support	PortAccess
M07()	Message	Sets machine mode for coolant on—mist	Support	PortAccess
M08()	Message	Sets machine mode for coolant on—flood	Support	PortAccess
M09()	Message	Sets machine mode for coolant off	Support	PortAccess
M10()	Message	Sets machine mode for clamp	Support	PortAccess
M11()	Message	Sets machine mode for unclamp	Support	PortAccess
AddLine()	Message	Function to add line to the final output	PortAccess	frmFinal
Show()	Message	Function to show display	PortAccess	frmFinal

Table 5.16 Summary of load part program use case

Use case summary	
Use case ID:	UC-T1
Use case name:	Load part program
Actors:	User
Description:	The user selects and loads part program through a file with "cnc" extension
Preconditions:	Part program file is present on the system
Post conditions:	Part program is loaded on the screen
System parameters	Part program file
Success scenario	Part program is loaded successfully
Alternative scenario	Error in loading part program
Includes:	Open file
Priority:	High

Table 5.17 Summary of save part program use case

Use case summary	
Use case ID:	UC-T2
Use case name:	Save part program
Actors:	User
Description:	The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions:	None
Post conditions:	Part program is saved on the file
System parameters	Part program with GM codes
Success scenario	Part program is validated and saved successfully
Alternative scenario	Error in saving part program
Includes:	Validate file, save file
Priority:	High

Table 5.18 Summary of execute part program use case

Use case summary	
Use case ID:	UC-T3
Use case name:	Execute part program
Actors:	User
Description:	The user executes the part program after loading it on the screen
Preconditions:	Part program is loaded and validated
Post conditions:	Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in *x*, *y,* and *z*-axes
System parameters	Part program with GM codes
Success scenario	Part program is executed successfully
Alternative scenario	Error in executing part program
Includes:	Validate file, execute file
Priority:	High

Table 5.19 Summary of edit part program use case

Use case summary	
Use case ID:	UC-T4
Use case name:	Edit part program
Actors:	User
Description:	The user edits the part program which is already saved
Preconditions:	Part program is selected and loaded on screen
Post conditions:	Part program is modified and saved after validation
System parameters	Part program file
Success scenario	Part program is edited successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.20 Summary of new part program use case

Use case summary	
Use case ID:	UC-T5
Use case name:	New part program
Actors:	User
Description:	The user inputs a new part program on the screen
Preconditions:	None
Post conditions:	Part program is validated and saved
System parameters	Part program with GM codes
Success scenario	New part program file is created successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.21 Summary of CNC turning flow chart

Flow chart summary	
Flow chart ID:	FC-T1
Flow chart name:	CNC turning process flow chart
No. of inputs:	7
No. of processes:	15
No. of outputs:	3
No. of control decisions:	34
Input name	Description
Save program	Save button on screen
Run program	Run button on screen
Load program	Load button on screen

(continued)

Table 5.21 (continued)

Flow chart summary	
New program	New button on screen
Edit program	Edit button on screen
Reset	Reset button on screen
Exit application	Exit button on screen
Process name	Description
Load part program from file	Function to load part program
Save part program	Function to save part program
Open dialog box	Function to open file on disk
New part program	Function to new part program
Edit part program	Function to edit part program
Reset machine	Function to reset the machine to initial position
End application	Function to close rendering and exit
Save file	Function to save file on disk
Execute part program	Function to execute program file
Set machine	Function to set machine position
Validate file	Function to validate program file
Execute file	Function to execute part program
All GM codes loop	Loop to check the GM code
Message arrived	Event to check a message
Status arrived	Event to check a status
Output name	Description
Part program display	Part program loaded on screen
Final output display	Work piece display on screen
Machine position display	Machine position values on screen
Control decision name	Description
Validate part program	Check to validate GM codes
Select part program	Select the program file
Save dialog	Save dialog box
Validated	Check whether file is validated
Emergency stop	Stop rendering and reset
CalculateLinearInterpolation	Check to calculate linear interpolation
CalculateCircularInterpolation	Check to calculate circular interpolation
CalculateParabolicInterpolation	Check to calculate parabolic interpolation
Delay	Check to cause the delay
PulsRequestX	Check to set the xPulses
PulsRequestZ	Check to set the zPulses
Xplus	Check to increase the xPulses
Xminus	Check to decrease the xPulses
Zplus	Check to increase the zPulses
Zminus	Check to decrease the zPulses
G00	Point-to-point positioning
G01	Linear interpolation
G02	Arc clockwise (2D)
G03	Arc counterclockwise(2D)
G06	Parabolic interpolation

(continued)

Table 5.21 (continued)

Flow chart summary	
G70	Inch programming
G71	Metric programming
G90	Absolute input
G91	Incremental input
M02	Program stop
M03	Spindle CW
M04	Spindle CCW
M05	Spindle off
M06	Tool change
M07	Coolant on—mist
M08	Coolant on—flood
M09	Coolant off
M10	Clamp
M11	Unclamp

5.7.3 Developing AR for Drilling Operation

CNC drilling consists of three projects.

1. Drill: Consists of classes used in CNC drilling software such as PortAccess and DrillMachine class, four enumerations and five events. Drill is a dynamic link library (DLL)-based project. The produced DLL of drill is used in CNC drilling software to provide separation between application and logic, and logic encapsulation.
   Class structure of drill: Drill consists of two classes, DrillMachine class and PortAccess class. DrillMachine class has encapsulated all the main functionality required by CNC drilling software such as Open File, Save File, Execute File, Validate File and Validate Code. A request from CNC drilling software for each function is handled by DrillMachine class. The enumerations of DrillMachine are Current, MachineStatus, Spindle, and Axis. Current enumeration identifies the current value of different parameter and MachineStatus identifies the machine running status. Spindle and axis enumerations are used to pass current spindle status and axis current position, respectively.
   To provide access to physical port on the system, drill has PortAccess class. PortAccess class has encapsulated port access and interpolations function. Only drill class has access to PortAccess class, and CNC drilling software does not have this facility. Drill class calls appropriate interpolation methods of PortAccess, which in turn calls private methods of PortAccess to send and receive physical port signal.
   In order to update the CNC drill software about the current state of execution, DrillMachine class has five events: SpindleHandler, MessageHandler, PositionHandler, StatusHandler, and WorkCompleted. SpindleHandler event is used to show the spindle status machine ON or OFF. MessageHandler is used to show the messages on information message box. PositionHandler is used to

show the current position of work piece in *x*, *y*, and *z*-axes and controls 3D movement of machine. StatusHandler is used to show the current status of drill machine. Information received from events by CNC drill software is used to update user interface and control 3D graphics.
2. CNC drilling: Consists of CNC drilling graphical user interface that utilizes drill object to execute file and related functions. Current status of execution and controlling of 3D animation is also updated from CNC drilling.
3. CoreAnimation: Consists of Drill3D class and three enumerations. Drill3D class consists of 3D graphic rendering engine and methods to control movement of individual parts of machine. Enumerations of CoreAnimation are direction, spindle, and axis. Drill3D displays the drill machine in 3D format.

Table 5.22 shows the PortAccess class member properties, methods, and variables. PortAccess is the helper class of DrillMachine class. PortAccess class is used to pass pulses to CNC drill by using parallel port and calculate the pulses, linear interpolation. PortAccess also holds the current position of *z*-axis and current state of the drill machine.

Table 5.23 describes the DrillMachine class member variables, properties, and methods. The DrillMachine class is used to read the part program file, validate it, open it, save it, and execute it. This class is also responsible for raising events to update user interface and 3D drill machine. Interpreter of CNC drill is also implemented in this class' ExecuteFile method.

Table 5.24 shows the Drill3D class member variables, properties, and methods. Drill3D class is used to render the graphics of drill machine in 3D format for display movement of machine. 3D rendering engine is also implemented in this class.

Table 5.25 defines the summary of the CNC drill sequence diagram.

Table 5.26 depicts the summary of CNC drill load part program use case.

Table 5.27 presents the summary of CNC drill save part program use case.

Table 5.28 explains the summary of CNC drill execute part program use case.

Table 5.29 illustrates the CNC drill edit part program use case.

Table 5.30 displays the CNC drill new part program use case.

Table 5.31 explains the CNC drill processes flow charts.

5.7.4 Developing AR for Sawing Operation

CNC sawing consists of three projects:

1. Sawing: Consists of classes used in CNC sawing software such as PortAccess and sawing, two enumerations and five events. Sawing is a dynamic link library (DLL)-based project. The produced DLL of sawing will be used in CNC

Table 5.22 Summary of CNC drill PortAccess class

Class summary			
Class ID: CD1			Namespace: Drill
Class name: PortAccess			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with:None
Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in *x*-axis
_CurrentPositionY	Double	Private	Holds the current position of work piece in *y*-axis
_NextPositionX	Double	Private	Holds the next position of work piece in *x*-axis
_NextPositionY	Double	Private	Holds the next position of work piece in *y*-axis
_xvalue	Double	Private	Holds the *X* value
_yvalue	Double	Private	Holds the *Y* value
_RapidMovement	Bool	Private	Holds the boolean value to rapid movement at a machine tool used for positioning the tool
_LinearInterpolation	Bool	Private	Holds the boolean value to allow direct movement between two points through linear interpolation
bMetricUnit	Bool	Private	Holds the boolean value for to check whether the unit is set to metric system
bAbsoluteCoordinate	Bool	Private	Holds the boolean value for to check whether the coordinate system is set to absolute coordinate
_Speed	Short	Private	Holds the value to control spindle speed
_Feed	Short	Private	Holds the feed of the spindle
_Tool	Short	Private	Holds the tool of the spindle
xPulses	Int	Private	Sets the pulse of the spindle in *x*-axis
yPulses	Int	Private	Sets the pulse of the spindle in *y*-axis
EmergencyStop	Static bool	Private	Holds the boolean value to control the machine's running status
Method name	Return type	Access modifier	Description
Delay()	Void	Public	Function to cause a delay of defined no. of cycles
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to increase the yPulses
Yminus()	Void	Public	Function to decrease the yPulses
G00()	Void	Public	Function to generate point-to-point positioning
G01()	Void	Public	Function to generate linear interpolation
G70()	Void	Public	Sets machine mode for inch programming
G71()	Void	Public	Sets machine mode for metric programming
G90()	Void	Public	Sets machine mode for absolute input
G91()	Void	Public	Sets machine mode for incremental input
M02()	Void	Public	Sets machine mode for program stop
M03()	Void	Public	Sets machine mode for spindle CW
M04()	Void	Public	Sets machine mode for spindle CCW
M05()	Void	Public	Sets machine mode for spindle off

Table 5.23 Summary of DrillMachine class

Class summary			
Class ID: CD2			Namespace: Drill
Class name: DrillMachine			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with: None
Field name	Data type	Access modifier	Description
sFileName	Static string	Public	Holds the file name of .cnc file
aList	Static ArrayList	Private	Holds the GM codes extracted from .cnc file
GMCodes	Static ArrayList	Private	Holds all GM codes
Component	Component	Private	Holds other managed resource this class uses
Disposed	Bool	Private	Track whether dispose has been called
Port	PortAccess	Private	Used to call the port functions for sending GM codes
Status	MachineStatus	Private	Holds and sets the machine status
sMachineName	String	Private	Holds the machine name
Method name	Return type	Access modifier	Description
Dispose()	Void	Private	Function to dispose the rendering
EmergencyStop()	Void	Public	Function to stop the rendering if emergency stop is pressed
OpenFile(TextBox textBox)	Void	Public	Function used for open .cnc file and display in textbox object based on this function
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	Static void	Public	Function to write errors in error log
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute

(continued)

Table 5.23 (continued)

Field name	Data type	Access modifier	Description
ValidateCode(string sGMCode,ref bool bIsGMCodeValid)	Void	Public	This function will validate the GMCode
ValidateFile()	Bool	Public	This function will validate the cnc part programe file. It will take file name from static variable set by calling routine. if cnc part programe file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This function will save the part program file. It will take file name from static variable set by calling routine

Event name	Delegate	Access modifier	Description
MessageArrived	MessageHandler	Public	Generates the event of message
StatusArrived	StatusHandler	Public	Generates the event of status
PositionArrived	PositionHandler	Public	Generates the event of current position
SpindleArrived	SpindleHandler	Public	Generates the event of spindle on or off
AddLine	LineEventDelegate	Public	Generates the event to add line
WorkCompleted	Completed	Public	Generates the event of work completed

Table 5.24 Summary of Drill3D class

Class summary			
Class ID: CD3			Namespace: CoreAnimation
Class name: Drill3D			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
TV	TVEngine	Public	Declares the TrueVision engine object
Inp	TVInputEngine	Private	Declares the input engine variable
TVLightEngine	TVLightEngine	Private	Declares the light engine variable
MatFactory	TVMaterialFactory	Private	Declares the material factory variable
TVScene	TVScene	Private	Declares the TVScene object
Global	TVGlobals	Private	Declares the global variable
FactoryFloor	TVMeshClass	Private	Declares the mesh class for the factory floor
Light	D3DLIGHT8	Private	Declares the light variable
Lines	ArrayList	Private	Holds the list of lines
MWorkPiece	MillingWorkPiece	Private	Holds the object for work piece
sngPositionX	Float	Private	Holds object position in *x*-axis for keyboard input
sngPositionY	Float	Private	Holds object position in *y*-axis for keyboard input
sngPositionZ	Float	Private	Holds object position in *z*-axis for keyboard input
snglookatX	Float	Private	Holds camera position in *x*-axis for keyboard input
snglookatY	Float	Private	Holds camera position in *y*-axis for keyboard input
snglookatZ	Float	Private	Holds camera position in *z*-axis for keyboard input
tmpMouseX	Int	Private	Holds mouse input position in *x*-axis
tmpMouseY	Int	Private	Holds mouse input position in *y*-axis
tmpMouseB1	Short	Private	Holds mouse input button1
tmpMouseB2	Short	Private	Holds mouse input button2

(continued)

Table 5.24 (continued)

Field name	Data type	Access modifier	Description
tmpMouseB3	Short	Private	Holds mouse input button3
tmpMouseScrollOld	Int	Private	Holds mouse input scroll old value
tmpMouseScrollNew	Int	Private	Holds mouse input scroll new value
sngAngleX	Float	Private	Holds mouse input angle in *x*-axis
sngAngleY	Float	Private	Holds mouse input angle in *y*-axis
sngWalk	Float	Private	For smooth movement while camera is walking
sngStrafe	Float	Private	For smooth movement while camera is strafing
DoLoop	Bool	Private	Holds the boolean value true until the scene is rendering
floor_x	Float	Private	Holds initial position for factory floor in *x*-axis
floor_y	Float	Private	Holds initial position for factory floor in *y*-axis
floor_z	Float	Private	Holds initial position for factory floor in *z*-axis
ModelPath	String	Private	Variable for storing model paths
UpperX	Float	Private	Holds initial position for the machine in *x*-axis
UpperY	Float	Private	Holds initial position for the machine in *y*-axis
UpperZ	Float	Private	Holds initial position for the machine in *z*-axis
WorkX	Float	Private	Holds initial position for the work piece in *x*-axis
WorkY	Float	Private	Holds initial position for the work piece in *y*-axis
WorkZ	Float	Private	Holds initial position for the work piece in *z*-axis
MeshMachine	TVMesh	Private	Holds the mesh for machine
MeshWork	TVMesh	Private	Holds the mesh for work piece
MeshWorkBase	TVMesh	Private	Holds the mesh for work piece base
MeshTemp	TVMesh	Private	Holds the mesh for temporary working piece
AnimationClass	TVKeyFrameAnim	Private	For animating the 3D mesh
WorkPosition	D3DVECTOR	Private	Holds current work piece position
gWorkPosition	D3DVECTOR	Private	Holds initial work piece position
WorkBasePosition	D3DVECTOR	Private	Holds current work base position

(continued)

Table 5.24 (continued)

Field name	Data type	Access modifier	Description
gWorkBasePosition	D3DVECTOR	Private	Holds initial work base position
MachineX	Float	Private	Holds initial position for the machine in *x*-axis
MachineY	Float	Private	Holds initial position for the machine in *y*-axis
MachineZ	Float	Private	Holds initial position for the machine in *z*-axis

Method name	Return type	Access modifier	Description
ClearLines()	Void	Public	Function to clear lines on work piece
Move(Direction D)	String	Private	Function to define movement limits
MoveControl(double x1, double y1, double x2, double y2)	Void	Public	Function to add the line on the work piece
Move(string coordinate, int value)	Void	Public	Function to move the spindle in *x*, *y,* or *z*-axis
SetWorkMesh()	Void	Public	Function to set the temporary working piece mesh position and attach it to the machine mesh
MoveY(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *y*-axis
ResetMachine()	Void	Public	Function to reset the work piece and base to initial position
MoveX(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *x*-axis
MoveZ(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *z*-axis
StartRendering()	Void	Public	Function to start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to set the DoLoop variable to false
Check_Movement()	Void	Public	Function to check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen

Table 5.25 Summary of CNC drill sequence diagram

Sequence diagram summary				
Sequence ID: SD-D1 Sequence name: CNC drilling sequence diagram				
Method name	Type	Description	Source	Destination
LoadPartProgram()	Message	Function to load part program	User	FrmMain
OpenFile()	Return message	Function to open file on disk	FrmMain	Support
SavePartProgram()	Message	Function to save part program	User	FrmMain
ValidateCode()	Return message	Function to validate GM codes	FrmMain	Support
SaveFile()	Message	Function to save file on disk	FrmMain	Support
EditPartProgram()	Self-message	Function to edit part program	User	FrmMain
NewPartProgram()	Self-message	Function to new part program	User	FrmMain
ExecutePartProgram()	Message	Function to execute part program	User	FrmMain
ValidateFile()	Return Message	Function to validate program file	FrmMain	Support
ExecuteFile()	Return Message	Function to execute program file	FrmMain	Support
Dispose()	Message	Function to close rendering and exit	User	FrmFinal
Delay()	Message	Function to cause a delay of defined no. of cycles	Support	PortAccess
PulsRequestX()	Message	Function to set the xPulses	Support	PortAccess
PulsRequestY()	Message	Function to set the yPulses	Support	PortAccess
Xplus()	Message	Function to increase the xPulses	Support	PortAccess
Xminus()	Message	Function to decrease the xPulses	Support	PortAccess
Yplus()	Message	Function to increase the yPulses	Support	PortAccess
Yminus()	Message	Function to decrease the yPulses	Support	PortAccess
G00()	Message	Function to generate point-to-point positioning	Support	PortAccess
G01()	Message	Function to generate linear interpolation	Support	PortAccess
G70()	Message	Sets machine mode for inch programming	Support	PortAccess
G71()	Message	Sets machine mode for metric programming	Support	PortAccess
G90()	Message	Sets machine mode for absolute input	Support	PortAccess

(continued)

Table 5.25 (continued)

Method name	Type	Description	Source	Destination
G91()	Message	Sets machine mode for incremental input	Support	PortAccess
M02()	Message	Sets machine mode for program stop	Support	PortAccess
M03()	Message	Sets machine mode for spindle CW	Support	PortAccess
M04()	Message	Sets machine mode for spindle CCW	Support	PortAccess
M05()	Message	Sets machine mode for spindle off	Support	PortAccess
AddLine()	Message	Function to add line to the final output	PortAccess	frmFinal
Show()	Message	Function to show display	PortAccess	frmFinal

Table 5.26 Summary of CNC drill load part program use case

Use case summary	
Use case ID:	UC-D1
Use case name:	Load part program
Actors:	User
Description:	The user selects and loads part program through a file with "cnc" extension
Preconditions:	Part program file is present on the system
Post conditions:	Part program is loaded on the screen
System parameters	Part program file
Success scenario	Part program is loaded successfully
Alternative scenario	Error in loading part program
Includes:	Open file
Priority:	High

Table 5.27 Summary of CNC drill save part program

Use case summary	
Use case ID:	UC-D2
Use case name:	Save part program
Actors:	User
Description:	The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions:	None
Post conditions:	Part program is saved on the file
System parameters	Part program with GM codes
Success scenario	Part program is validated and saved successfully
Alternative scenario	Error in saving part program
Includes:	Validate file, save file
Priority:	High

Table 5.28 Summary of CNC drill execute part program

Use case summary	
Use case ID:	UC-D3
Use case name:	Execute part program
Actors:	User
Description:	The user executes the part program after loading it on the screen
Preconditions:	Part program is loaded and validated
Post conditions:	Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in *x*, *y,* and *z*-axes
System parameters	Part program with GM codes
Success scenario	Part program is executed successfully
Alternative scenario	Error in executing part program
Includes:	Validate file, execute file
Priority:	High

Table 5.29 Summary of CNC drill edit part program

Use case summary	
Use case ID:	UC-D4
Use case name:	Edit part program
Actors:	User
Description:	The user edits the part program which is already saved
Preconditions:	Part program is selected and loaded on screen
Post conditions:	Part program is modified and saved after validation
System parameters	Part program file
Success scenario	Part program is edited successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.30 Summary of CNC drill new part program

Use case summary	
Use case ID:	UC-D5
Use case name:	New part program
Actors:	User
Description:	The user inputs a new part program on the screen
Preconditions:	None
Post conditions:	Part program is validated and saved
System parameters	Part program with GM codes
Success scenario	New part program file is created successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.31 Summary of CNC drill processes flow char

Flow chart summary	
Flow chart ID: FC-D1	
Flow chart name: CNC drilling process flow chart	
No. of inputs: 7	
No. of processes: 15	
No. of outputs: 3	
No. of control decisions: 22	
Input name	Description
Save program	Save button on screen
Run program	Run button on screen

(continued)

Table 5.31 (continued)

Input name	Description
Load program	Load button on screen
New program	New button on screen
Edit program	Edit button on screen
Reset	Reset button on screen
Exit application	Exit button on screen
Process name	**Description**
Load part program from file	Function to load part program
Save part program	Function to save part program
Open dialog box	Function to open file on disk
New part program	Function to new part program
Edit part program	Function to edit part program
Reset machine	Function to reset the machine to initial position
End application	Function to close rendering and exit
Save file	Function to save file on disk
Execute part program	Function to execute program file
Set machine	Function to set machine position
Validate file	Function to validate program file
Execute file	Function to execute part program
All GM codes loop	Loop to check the GM code
Message arrived	Event to check a message
Status arrived	Event to check a status
Output name	Description
Part program display	Part program loaded on screen
Final output display	Work piece display on screen
Machine position display	Machine position values on screen
Control decision name	Description
Validate part program	Check to validate GM codes
Select part program	Select the program file
Save dialog	Save dialog box
Validated	Check whether file is validated
Emergency stop	Stop rendering and reset
Delay	Check to cause the delay
PulsRequestX	Check to set the xPulses
PulsRequestY	Check to set the yPulses
Xplus	Check to increase the xPulses
Xminus	Check to decrease the xPulses
Yplus	Check to increase the yPulses
Yminus	Check to decrease the yPulses
G00	Point-to-point positioning
G01	Linear interpolation
G70	Inch programming
G71	Metric Programming
G90	Absolute input

(continued)

Table 5.31 (continued)

Process name	Description
G91	Incremental input
M02	Program stop
M03	Spindle CW
M04	Spindle CCW
M05	Spindle off

sawing software to provide separation between application and logic, and logic encapsulation.

Class structure of sawing: Sawing consists of two classes, sawing class and PortAccess class. Sawing class has encapsulated all the main functionality required by CNC sawing software such as Open File, Save File, Edit File, Execute File, Validate File and Validate Code. A request from CNC sawing software for each function is handling by sawing class. The enumerations of sawing are Current and MachineStatus. Current enumeration identifies the current value of different parameter and MachineStatus identifies the machine running status.

To provide access to physical port on the system, sawing has PortAccess class. PortAccess class has encapsulated port access and interpolations function. Only sawing class has access to PortAccess class and CNC sawing software does not have this facility. Sawing class calls appropriate interpolation methods of PortAccess, which in turn calls private methods of PortAccess to send and receive physical port signal.

In order to update the user interface of CNC sawing software about the current state of execution, sawing class has five events: SpindleHandler, MessageHandler, PositionHandler, StatusHandler, and Completed. SpindleHandler event is used to show the spindle status for machine ON or OFF. MessageHandler is used to show the messages on information message box. PositionHandler is used to show the current position of work piece and machine in *x*, *y,* and *z*-axes and control 3D movement of machine. StatusHandler is used to show the current status of sawing machine. Information received from events by CNC sawing software is used to update user interface and control 3D graphics.

2. CNC sawing: Consists of CNC sawing graphical user interface that utilize sawing object to execute file and related functions. Current status of execution and controlling of 3D animation is also updated from CNC sawing.
3. SawingMachineAnimation: Consists of sawing3D class and three enumerations. Sawing3D class consists of 3D graphic rendering engine and methods to control movement of individual parts of machine. Enumerations of sawingMachineAnimation are direction, spindle, and axis. Direction enumeration is for direction of movement, spindle enumeration is to control the running status of spindle, Axis enumeration is for controlling the movement of work piece and machine.

Table 5.32 shows the PortAccess class member properties, methods, and variables. PortAccess is the helper class of sawing class. PortAccess class is used to pass pulses to CNC sawing by using parallel port and calculate the pulses and leaner interpolation. PortAccess also holds the current position of *x*, *y,* and *z*-axes and current state of the sawing machine (Figs. 5.17, 5.18, 5.19).

Table 5.33 describes the sawing class member variables, properties, and methods. The sawing class is used to read the part program file, validate it, open it, save it, and execute it. This class is also responsible for raising events to update user interface and 3D sawing machine. Interpreter of CNC sawing is also implemented in this class's ExecuteFile method.

Table 5.34 illustrates the Sawing3D class member variables, properties, and methods. Sawing3D class is used to render the graphics of sawing machine in 3D format for display movement of machine. 3D rendering engine is also implemented in this class.

Table 5.35 defines the summary of CNC sawing sequence diagram.

Table 5.36 details the summary of CNC sawing load part program use case.

Table 5.37 explains the summary of CNC sawing save part program use case.

Table 5.38 outlines the summary of CNC sawing execute part program use case.

Table 5.39 produces the summary of CNC sawing edit part program use case.

Table 5.40 outlines the summary of CNC sawing new part program use case.

Table 5.41 displays the summary of CNC sawing processes flow charts

5.8 Developing AR for CNC CMM

CNC coordinate measuring machine (CMM) is part of VMS quality assurance. The construction of the CNC CMM is akin to that of a CNC milling machine. Its treatment from software prospective for developing AR is also similar to that of metal-cutting machinery.

CNC CMM software consists of three projects.

1. CoordinateMeasuringMachine: Consists of classes used in CNC CMM software such as CMMPortAccess and Machine classes, two enumerations, and five events. CoordinateMeasuringMachine library is a dynamic link library (DLL)-based project. The produced DLL of CoordinateMeasuringMachine library used in CNC CMM software to provide separation between application, and logic, and logic encapsulation.
   Class structure of CoordinateMeasuringMachine: Machine class has encapsulated all the main functionality required by CNC CMM software such as Open File, Save File, Execute File, Validate File, and Validate Code. A request from

Table 5.32 Summary of sawing PortAccess class

Class summary			
Class ID: CS1			Namespace: Sawing Machine
Class name: PortAccess			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in *x*-axis
_CurrentPositionY	Double	Private	Holds the current position of work piece in *y*-axis
_NextPositionX	Double	Private	Holds the next position of work piece in *x*-axis
_NextPositionY	Double	Private	Holds the next position of work piece in *y*-axis
_xvalue	Double	Private	Holds the *X* value
_yvalue	Double	Private	Holds the *Y* value
_RapidMovement	Bool	Private	Holds the boolean value to rapid movement at a machine tool used for positioning the tool
_LinearInterpolation	Bool	Private	Holds the boolean value to allow direct movement between two points through linear interpolation
bMetricUnit	Bool	Private	Holds the boolean value for to check whether the unit is set to metric system
bAbsoluteCoordinate	Bool	Private	Holds the boolean value for to check whether the coordinate system is set to absolute coordinate
_Speed	Short	Private	Holds the value to control spindle speed
_Feed	Short	Private	Holds the feed of the spindle
_Tool	Short	Private	Holds the tool of the spindle

(continued)

Table 5.32 (continued)

Field name	Data type	Access modifier	Description
xPulses	Int	Private	Sets the pulse of the spindle in *x*-axis
yPulses	Int	Private	Sets the pulse of the spindle in *y*-axis
EmergencyStop	Static bool	Private	Holds the boolean value to control the machine running status
Method name	**Return type**	**Access modifier**	**Description**
Delay()	Void	Public	Function to cause a delay of defined no. of cycles
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to increase the yPulses
Yminus()	Void	Public	Function to decrease the yPulses
G00()	Void	Public	Function to generate point-to-point positioning
G01()	Void	Public	Function to generate linear interpolation
G70()	Void	Public	Sets machine mode for inch programming
G71()	Void	Public	Sets machine mode for metric programming
G90()	Void	Public	Sets machine mode for absolute input
G91()	Void	Public	Sets machine mode for incremental input
M02()	Void	Public	Sets machine mode for program stop
M03()	Void	Public	Sets machine mode for spindle CW
M04()	Void	Public	Sets machine mode for spindle CCW
M05()	Void	Public	Sets machine mode for spindle off

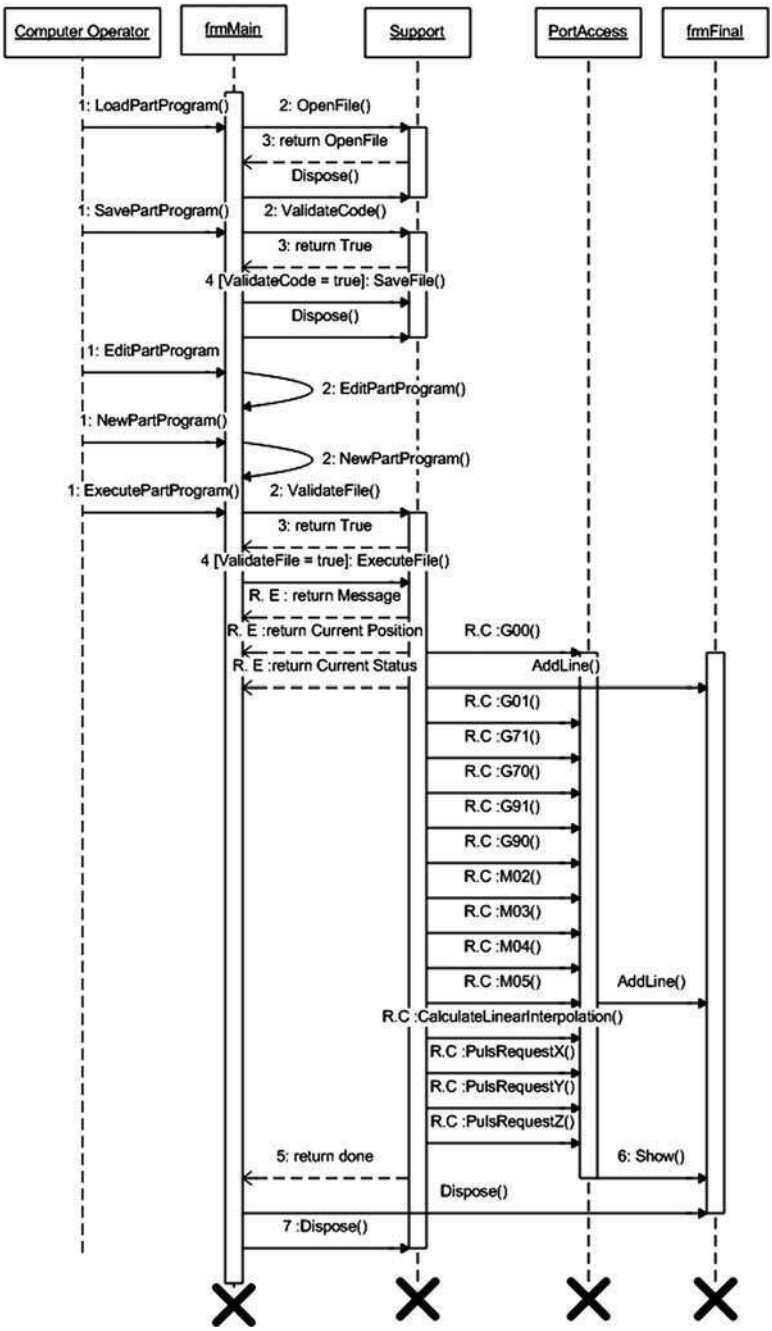


Fig. 5.17 CNC sawing UML sequence diagram

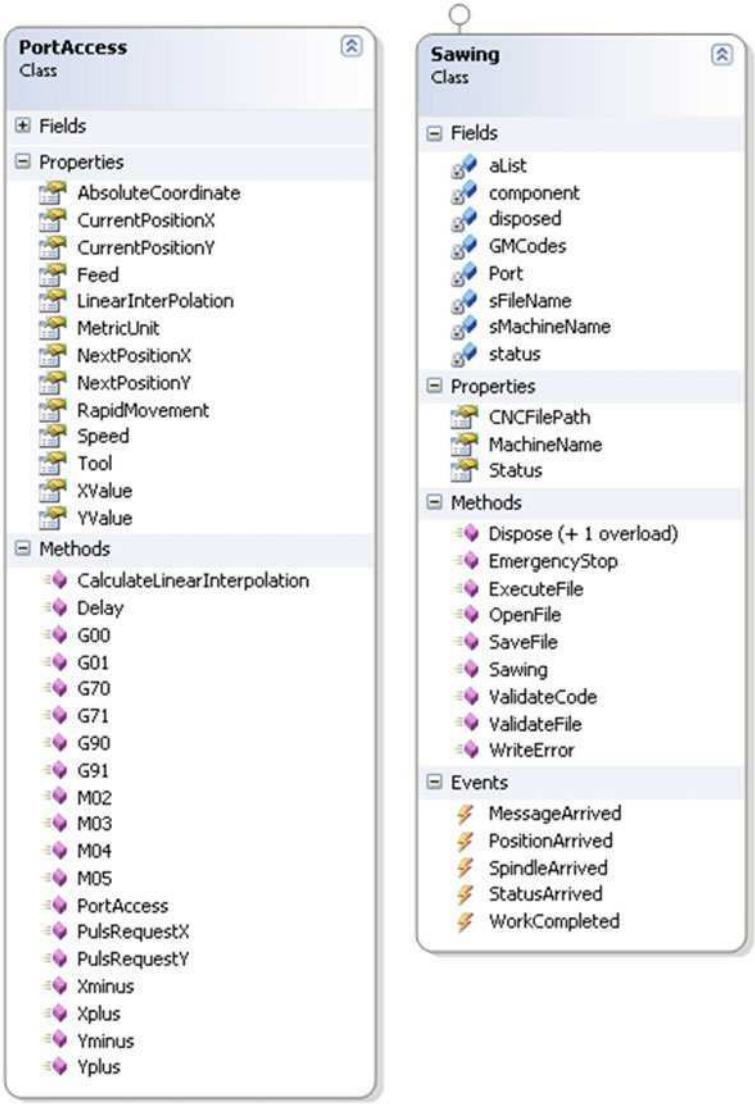


Fig. 5.18 UML static diagram for CNC sawing machine (continued)

CNC CMM software for each function is handling by Machine class. The enumerations of Machine are Current and MachineStatus. Current enumeration identifies the current value of different parameter and MachineStatus identifies the machine running status.

To provide access to physical port on the system, CoordinateMeasuring-Machine has CMMPortAccess class. CMMPortAccess has encapsulated port access and interpolation functions. Only Machine class has access to

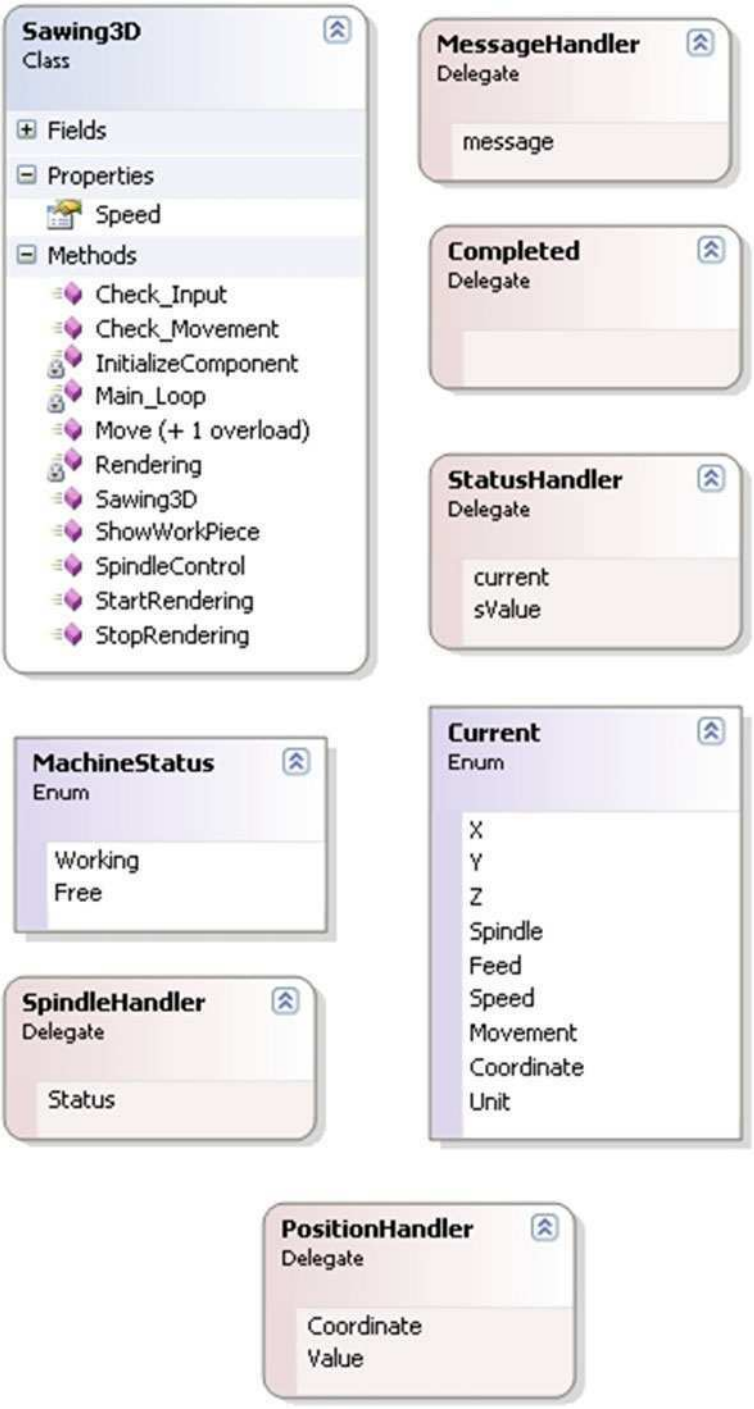


Fig. 5.18 (continued)

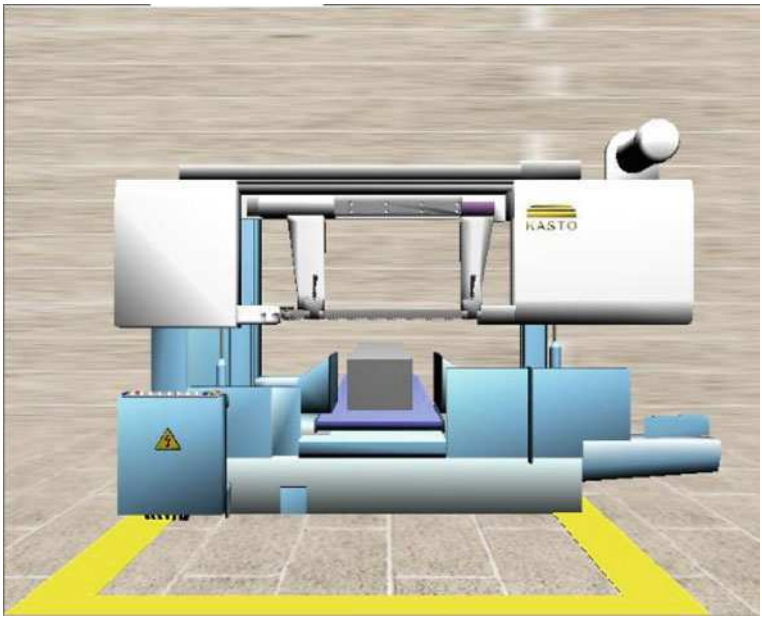


Fig. 5.19 AR for CNC sawing

CMMPortClass and CNC CMM software does not have this facility. Machine class calls appropriate interpolation method, which in turn calls private methods of port access.

In order to update the CNC CMM software about the current state of execution, machine has five events: SpindleHandler, MessageHandler, PositionHandler, and StatusHandler. SpindleHandler event is used to show the spindle status ON or OFF. MessageHandler is used to show the messages on information message box. PositionHandler is used to show the current position of work piece in *x*, *y,* and *z*-axes. StatusHandler is used to show the current status of milling machine. Information received from events by CNC CMM software is used to update user interface and control 3D graphics.

2. CNCCMM: Consists of CNC CMM software user interface that utilize machine object to execute file and related functions. Current status of execution is also updated from CNC machine.
3. CoordinateMeasuringMachine3D: Consists of Machine3D class and three enumerations and a event. Machine3D class consists of 3D graphics rendering engine and provide methods to control movement of individual parts of machine. Enumerations of CoordinateMeasuringMachine3D are direction, spindle, and axis. Event of Machine3D CollisionOccur is used to update the CNC CMM user interface when stylus touches the work piece (Figs. 5.21, 5.22, 5.23).

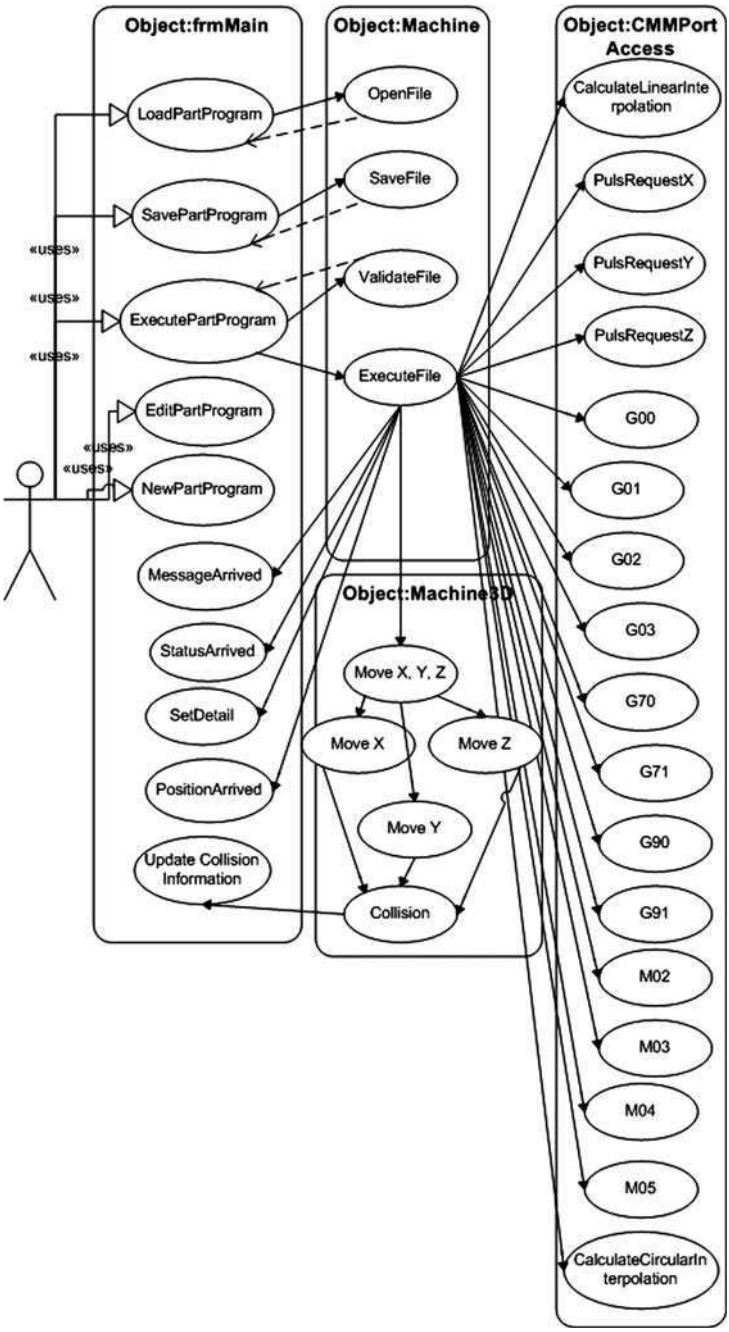


Fig. 5.20 CNC CMM UML use case diagram

Table 5.33 Summary of sawing class

Class summary			
Class ID: CS2			Namespace: Sawing Machine
Class name: Sawing			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with: None
Field name	Data type	Access modifier	Description
sFileName	Static string	Public	Holds the file name of .cnc file
aList	Static ArrayList	Private	Holds the GM codes extracted from .cnc file
GMCodes	Static ArrayList	Private	Holds all GM codes
Component	Component	Private	Holds other managed resource this class uses
Disposed	Bool	Private	Track whether dispose has been called
Port	PortAccess	Private	Used to call the port functions for sending GM codes
Status	MachineStatus	Private	Holds and sets the machine status
sMachineName	String	Private	Holds the machine name

Method name	Return type	Access modifier	Description
Dispose()	Void	Private	Function to dispose the rendering
EmergencyStop()	Void	Public	Function to stop the rendering if emergency stop is pressed
OpenFile(TextBox textBox)	Void	Public	Function used for open cnc file and display in textbox object based on this function
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	Static void	Public	Function to write errors in error log
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute

(continued)

Table 5.33 (continued)

Method name	Return type	Access modifier	Description
ValidateCode(string sGMCode,ref bool bIsGMCodeValid)	Void	Public	This function will validate the GMCode
ValidateFile()	Bool	Public	This function will validate the cnc part programe file. It will take file name from static variable set by calling routine. If cnc part programe file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This function will save the part program file. It will take file name from static variable set by calling routine

Event name	Delegate	Access modifier	Description
MessageArrived	MessageHandler	Public	To generate the event of message
StatusArrived	StatusHandler	Public	To generate the event of status
PositionArrived	PositionHandler	Public	To generate the event of current position
SpindleArrived	SpindleHandler	Public	To generate the event of spindle on or off
AddLine	LineEventDelegate	Public	To generate the event to add line
WorkCompleted	Completed	Public	To generate the event of work completed

Table 5.34 Summary of sawing3D class

Class summary			
Class ID: CS3			Namespace: Sawing Animation
Class name: Sawing3D			Class type: Concrete
Is base class? No			No. of Inherited Classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
TV	TVEngine	Public	Declares the TrueVision engine object
Inp	TVInputEngine	Private	Declares the input engine variable
TVLightEngine	TVLightEngine	Private	Declares the light engine variable
MatFactory	TVMaterialFactory	Private	Declares the material factory variable
TVScene	TVScene	Private	Declares the TVScene object
Global	TVGlobals	Private	Declares the global variable
FactoryFloor	TVMeshClass	Private	Declares the mesh class for the factory floor
Light	D3DLIGHT8	Private	Declares the light variable
Lines	ArrayList	Private	Holds the list of lines
MWorkPiece	MillingWorkPiece	Private	Holds the object for work piece
sngPositionX	Float	Private	Holds object position in *x*-axis for keyboard input
sngPositionY	Float	Private	Holds object position in *y*-axis for keyboard input
sngPositionZ	Float	Private	Holds object position in *z*-axis for keyboard input
snglookatX	Float	Private	Holds camera position in *x*-axis for keyboard input
snglookatY	Float	Private	Holds camera position in *y*-axis for keyboard input
snglookatZ	Float	Private	Holds camera position in *z*-axis for keyboard input
tmpMouseX	int	Private	Holds mouse input position in *x*-axis
tmpMouseY	int	Private	Holds mouse input position in *y*-axis
tmpMouseB1	Short	Private	Holds mouse input button1
tmpMouseB2	Short	Private	Holds mouse input button2

(continued)

Table 5.34 (continued)

Field name	Data type	Access modifier	Description
tmpMouseB3	Short	Private	Holds mouse input button3
tmpMouseScrollOld	int	Private	Holds mouse input scroll old value
tmpMouseScrollNew	int	Private	Holds mouse input scroll new value
sngAngleX	Float	Private	Holds mouse input angle in *x*-axis
sngAngleY	Float	Private	Holds mouse input angle in *y*-axis
sngWalk	Float	Private	For smooth movement while camera is walking
sngStrafe	Float	Private	For smooth movement while camera is strafing
DoLoop	Bool	Private	Holds the boolean value true until the scene is rendering
floor_x	Float	Private	Holds initial position for factory floor in *x*-axis
floor_y	Float	Private	Holds initial position for factory floor in *y*-axis
floor_z	Float	Private	Holds initial position for factory floor in *z*-axis
ModelPath	String	Private	Variable for storing model paths
UpperX	Float	Private	Holds initial position for the machine in *x*-axis
UpperY	Float	Private	Holds initial position for the machine in *y*-axis
UpperZ	Float	Private	Holds initial position for the machine in *z*-axis
WorkX	Float	Private	Holds initial position for the work piece in *x*-axis
WorkY	Float	Private	Holds initial position for the work piece in *y*-axis
WorkZ	Float	Private	Holds initial position for the work piece in *z*-axis
MeshMachine	TVMesh	Private	Holds the mesh for machine
MeshWork	TVMesh	Private	Holds the mesh for work piece
MeshWorkBase	TVMesh	Private	Holds the mesh for work piece base
MeshTemp	TVMesh	Private	Holds the mesh for temporary working piece
AnimationClass	TVKeyFrameAnim	Private	For animating the 3D mesh
WorkPosition	D3DVECTOR	Private	Holds current work piece position
gWorkPosition	D3DVECTOR	Private	Holds initial work piece position
WorkBasePosition	D3DVECTOR	Private	Holds current work base position

(continued)

Table 5.34 (continued)

Field name	Data type	Access modifier	Description
gWorkBasePosition	D3DVECTOR	Private	Holds initial work base position
MachineX	Float	Private	Holds initial position for the machine in *x*-axis
MachineY	Float	Private	Holds initial position for the machine in *y*-axis
MachineZ	Float	Private	Holds initial position for the machine in *z*-axis

Method name	Return type	Access modifier	Description
ClearLines()	Void	Public	Function to clear lines on work piece
Move(direction D)	String	Private	Function to define movement limits
MoveControl(double x1, double y1, double x2, double y2)	Void	Public	Function to add the line on the work piece
Move(string coordinate, int value)	Void	Public	Function to move the spindle in *x*, *y*, or *z*-axes
SetWorkMesh()	Void	Public	Function to set the temporary working piece mesh position and attach it to the machine mesh
MoveY(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *y*-axis
ResetMachine()	Void	Public	Function to reset the work piece and base to initial position
MoveX(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *x*-axis
MoveZ(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *z*-axis
StartRendering()	Void	Public	Function to start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to set the DoLoop variable to false
Check_Movement()	Void	Public	Function to check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen

Table 5.35 Summary of CNC sawing sequence diagram

Sequence diagram summary				
Sequence ID: SD-S1 Sequence name: CNC Sawing sequence diagram				
Method name	Type	Description	Source	Destination
LoadPartProgram()	Message	Function to load part program	User	FrmMain
OpenFile()	Return message	Function to open file on disk	FrmMain	Support
SavePartProgram()	Message	Function to save part program	User	FrmMain
ValidateCode()	Return message	Function to validate GM codes	FrmMain	Support
SaveFile()	Message	Function to save file on disk	FrmMain	Support
EditPartProgram()	Self-message	Function to edit part program	User	FrmMain
NewPartProgram()	Self-message	Function to new part program	User	FrmMain
ExecutePartProgram()	Message	Function to execute part program	User	FrmMain
ValidateFile()	Return message	Function to validate program file	FrmMain	Support
ExecuteFile()	Return message	Function to execute program file	FrmMain	Support
Dispose()	Message	Function to close rendering and exit	User	FrmFinal
Delay()	Message	Function to cause a delay of defined no. of cycles	Support	PortAccess
PulsRequestX()	Message	Function to set the xPulses	Support	PortAccess
PulsRequestY()	Message	Function to set the yPulses	Support	PortAccess
Xplus()	Message	Function to increase the xPulses	Support	PortAccess
Xminus()	Message	Function to decrease the xPulses	Support	PortAccess
Yplus()	Message	Function to increase the yPulses	Support	PortAccess
Yminus()	Message	Function to decrease the yPulses	Support	PortAccess
G00()	Message	Function to generate point-to-point positioning	Support	PortAccess
G01()	Message	Function to generate linear interpolation	Support	PortAccess
G70()	Message	Sets machine mode for inch programming	Support	PortAccess
G71()	Message	Sets machine mode for metric programming	Support	PortAccess
G90()	Message	Sets machine mode for absolute input	Support	PortAccess

(continued)

Table 5.35 (continued)

Method name	Type	Description	Source	Destination
G91()	Message	Sets machine mode for incremental input	Support	PortAccess
M02()	Message	Sets machine mode for program stop	Support	PortAccess
M03()	Message	Sets machine mode for spindle CW	Support	PortAccess
M04()	Message	Sets machine mode for spindle CCW	Support	PortAccess
M05()	Message	Sets machine mode for spindle off	Support	PortAccess
AddLine()	Message	Function to add line to the final output	PortAccess	frmFinal
Show()	Message	Function to show display	PortAccess	frmFinal

Table 5.36 Summary of CNC sawing load part program use case

Use case summary	
Use case ID:	UC-S1
Use case name:	Load part program
Actors:	User
Description:	The user selects and loads part program through a file with "cnc" extension
Preconditions:	Part program file is present on the system
Post conditions:	Part program is loaded on the screen
System parameters	Part program file
Success scenario	Part program is loaded successfully
Alternative scenario	Error in loading part program
Includes:	Open file
Priority:	High

Table 5.37 Summary of CNC sawing save part program use case

Use case summary	
Use case ID:	UC-S2
Use case name:	Save part program
Actors:	User
Description:	The user inputs the part program on the screen and saves it to a file with "cnc"extension
Preconditions:	None
Post conditions:	Part program is saved on the file
System parameters	Part program with GM codes
Success scenario	Part program is validated and saved successfully
Alternative scenario	Error in saving part program
Includes:	Validate file, save file
Priority:	High

Table 5.38 Summary of CNC sawing execute part program use case

Use case summary	
Use case ID:	UC-S3
Use case name:	Execute part program
Actors:	User
Description:	The user executes the part program after loading it on the screen
Preconditions:	Part program is loaded and validated
Post conditions:	Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in *x*, *y,* and *z*-axes
System parameters	Part program with GM codes
Success scenario	Part program is executed successfully
Alternative scenario	Error in executing part program
Includes:	Validate file, execute file
Priority:	High

Table 5.39 Summary of CNC sawing edit part program use case

Use case summary	
Use case ID:	UC-S4
Use case name:	Edit part program
Actors:	User
Description:	The user edits the part program which is already saved
Preconditions:	Part program is selected and loaded on screen
Post conditions:	Part program is modified and saved after validation
System parameters	Part program file
Success scenario	Part program is edited successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.40 Summary of CNC sawing new part program use case

Use case summary	
Use case ID:	UC-S5
Use case name:	New part program
Actors:	User
Description:	The user inputs a new part program on the screen
Preconditions:	None
Post conditions:	Part program is validated and saved
System parameters	Part program with GM codes
Success scenario	New part program file is created successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.41 Summary of CNC sawing processes flow charts

Flow chart summary	
Flow chart ID:	FC-S1
Flow chart name:	CNC Sawing process flow chart
No. of inputs:	7
No. of processes:	15
No. of outputs:	3
No. of control decisions:	22
Input name	Description
Save program	Save button on screen
Run program	Run button on screen
Load program	Load button on screen
New program	New button on screen

(continued)

Table 5.41 (continued)

Flow chart summary	
Edit program	Edit button on screen
Reset	Reset button on screen
Exit application	Exit button on screen
Process name	Description
Load part program from file	Function to load part program
Save part program	Function to save part program
Open dialog box	Function to open file on disk
New part program	Function to new part program
Edit part program	Function to edit part program
Reset machine	Function to reset the machine to initial position
End application	Function to close rendering and exit
Save file	Function to save file on disk
Execute part program	Function to execute program file
Set machine	Function to set machine position
Validate file	Function to validate program file
Execute file	Function to execute part program
All GM codes loop	Loop to check the GM code
Message arrived	Event to check a message
Status arrived	Event to check a status
Output name	Description
Part program display	Part program loaded on screen
Final output display	Work piece display on screen
Machine position display	Machine position values on screen
Control decision name	Description
Validate part program	Check to validate GM codes
Select part program	Select the program file
Save dialog	Save dialog box
Validated	Check whether file is validated
Emergency stop	Stop rendering and reset
Delay	Check to cause the delay
PulsRequestX	Check to set the xPulses
PulsRequestY	Check to set the yPulses
Xplus	Check to increase the xPulses
Xminus	Check to decrease the xPulses
Yplus	Check to increase the yPulses
Yminus	Check to decrease the yPulses
G00	Point-to-point positioning
G01	Linear interpolation
G70	Inch programming
G71	Metric programming
G90	Absolute input
G91	Incremental input
M02	Program stop
M03	Spindle CW
M04	Spindle CCW
M05	Spindle off

Table 5.42 shows the CMMPortAccess class members properties, methods, and variables. CMMPortAccess is the helper class of machine, and this class is used to pass pulses to CNC CMM by using parallel port and calculate the pulses, liner interpolation, circular interpolation, parabolic interpolation. CMMPortAccess also holds the current position of *x*, *y,* and *z*-axes and current position of *x*, *y,* and *z*-axes and current state of the CMM machine.

Table 5.43 depicts the Machine class members variables, properties, and methods. The Machine class is used to read the part program file, validate it, open it, save it, and execute it. This class is also responsible for raising events to update user interface and 3D CMM machine. Interpreter of CNC CMM is also implemented in this class's ExecuteFile method.

Table 5.44 defines the Machine3D class member variables, properties, and methods. Machine3D class is used to render the graphics of CMM machine in 3D format for display movement of machine and work piece. 3D rendering engine is also implemented in this class.

Table 5.45 illustrates the CNC CMM load part program use case.

Table 5.46 presents the CNC CMM save part program use case.

Table 5.47 explains the CNC CMM execute part program use case.

Table 5.48 outlines the CNC CMM edit part program use case.

Table 5.49 produces the CNC CMM new part program use case.

Table 5.50 displays the CNC CMM processes sequence.

Table 5.51 shows the CNC CMM processes flow charts.

5.9 Interface Design for system Integration

The electronic circuit presented as Fig. 5.24 is designed to fulfill the objectives of development of proxy interface. This circuit is designed for a 3-axis milling machine controlled using EIA-247-D G and M code. The electronic circuit fully supports operation of the milling process through an AR controller. The data sheets for the microcontroller and all other important integrated circuits are provided as Appendix-IV. Power supply circuit for all the electronic circuits implementation used in this book is given in Appendix V.

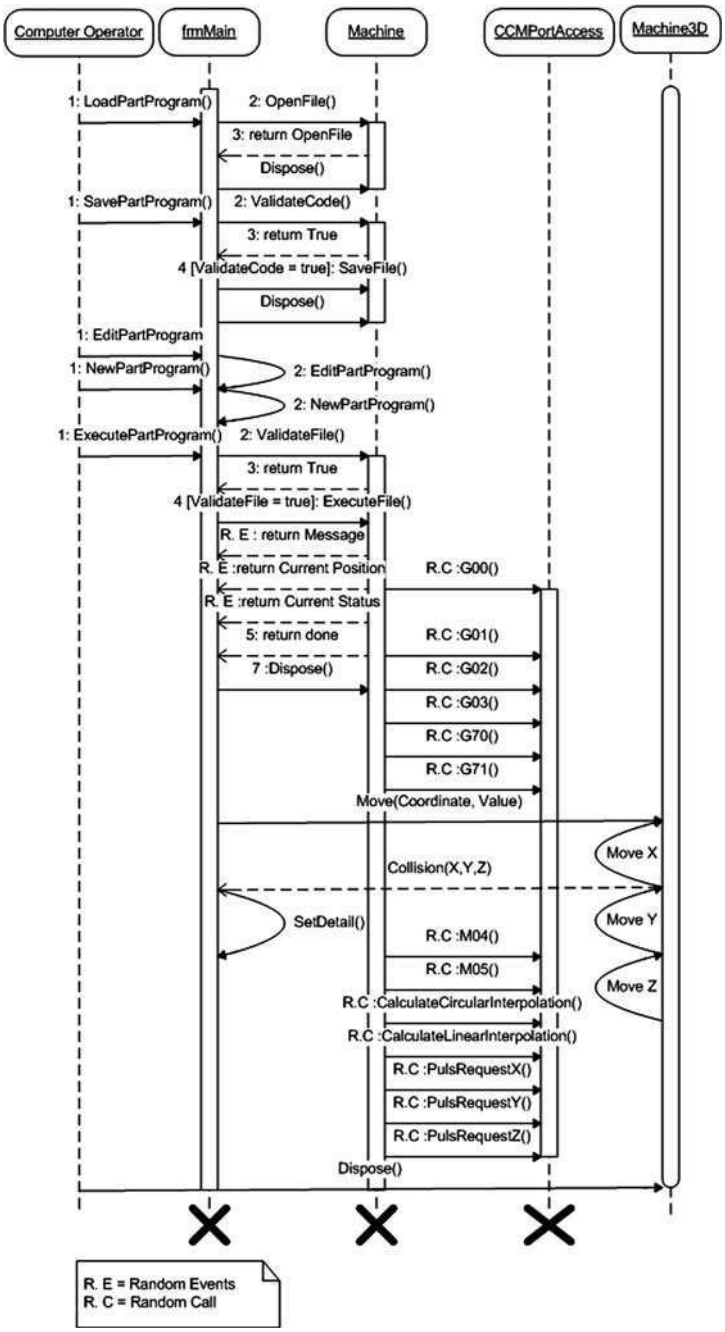


Fig. 5.21 CNC CMM UML sequence diagram

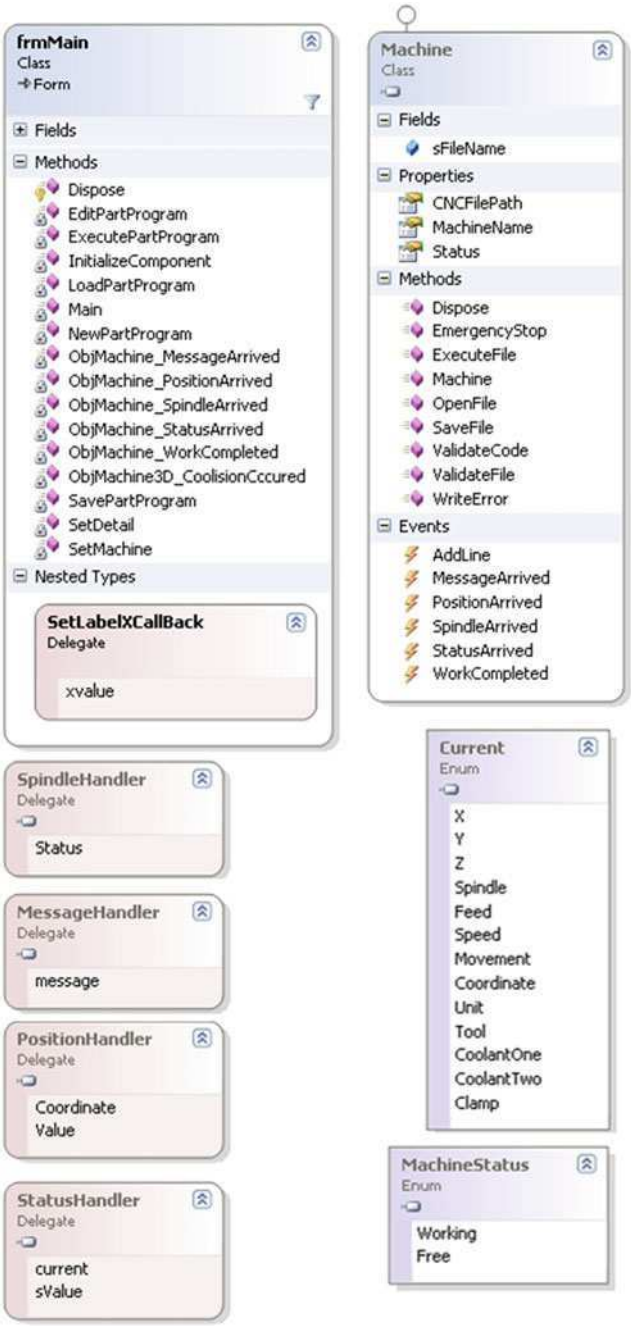


Fig. 5.22 CNC CMM UML classes static diagram

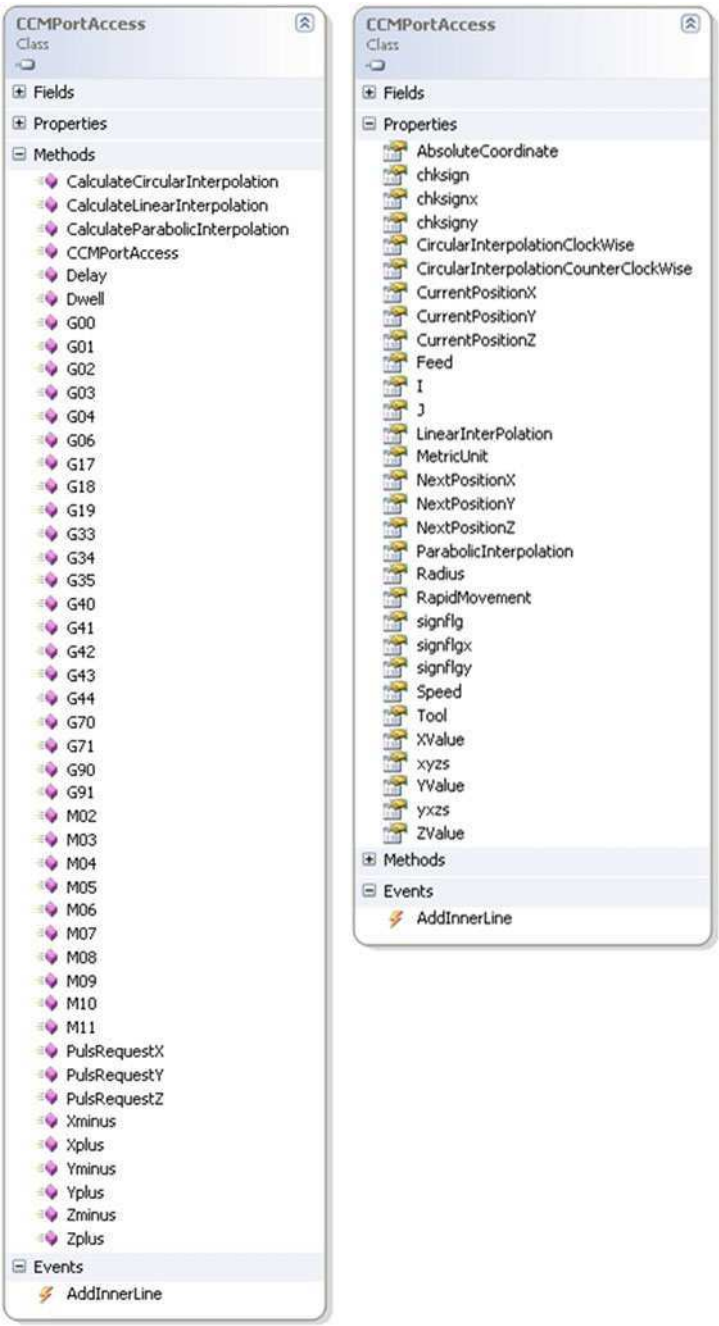


Fig. 5.22 (continued)

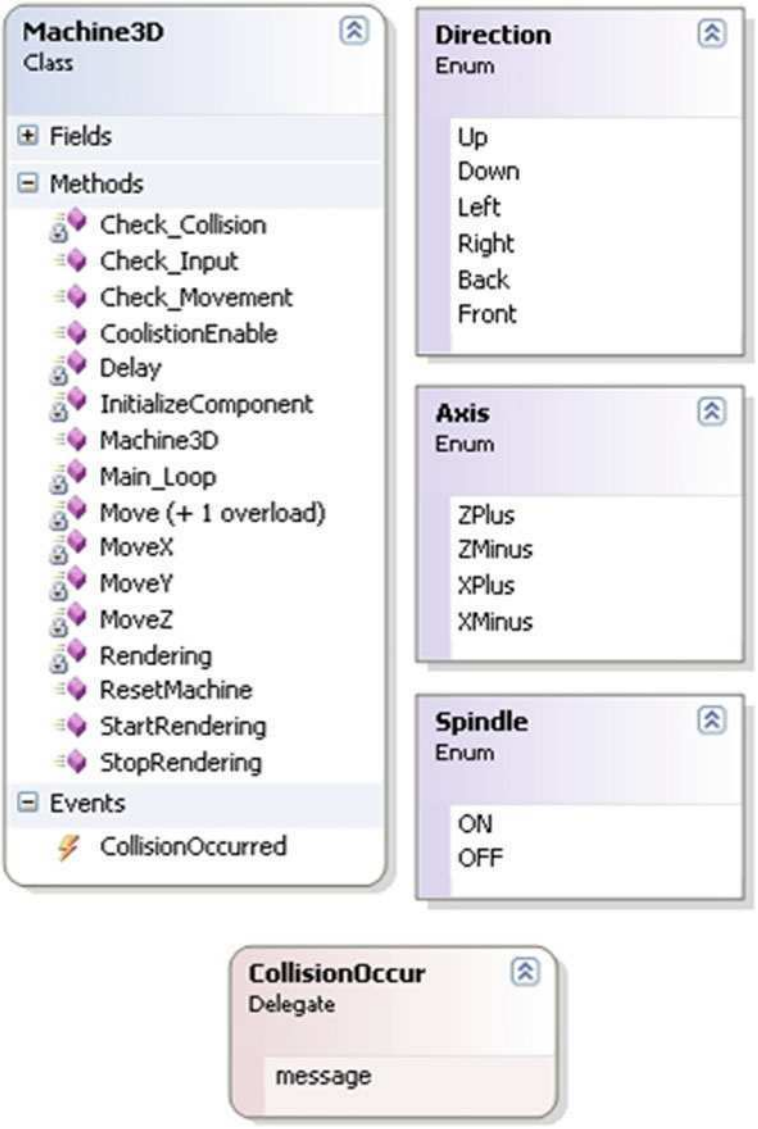


Fig. 5.22 (continued)

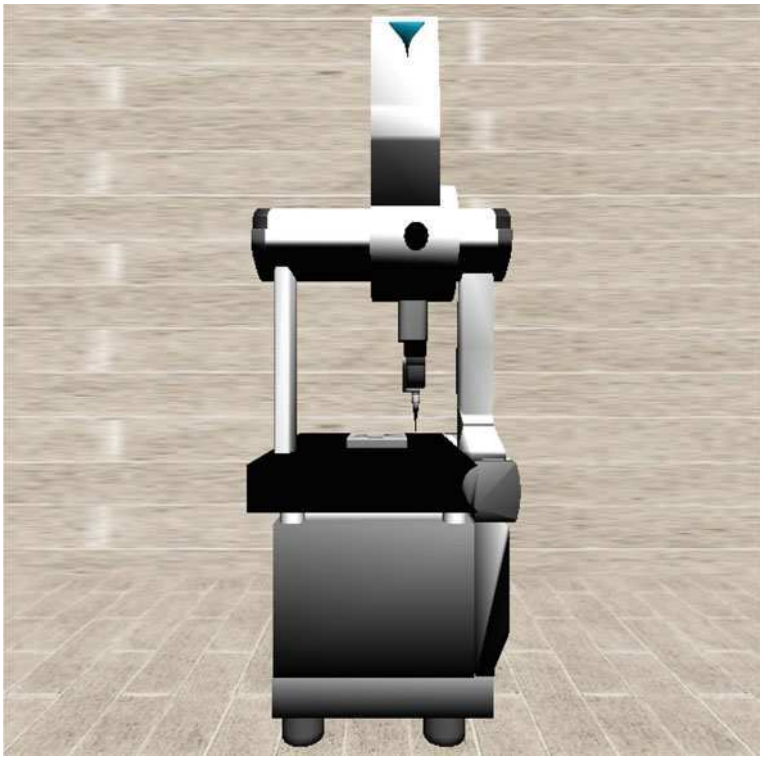


Fig. 5.23 AR for CNC CMM

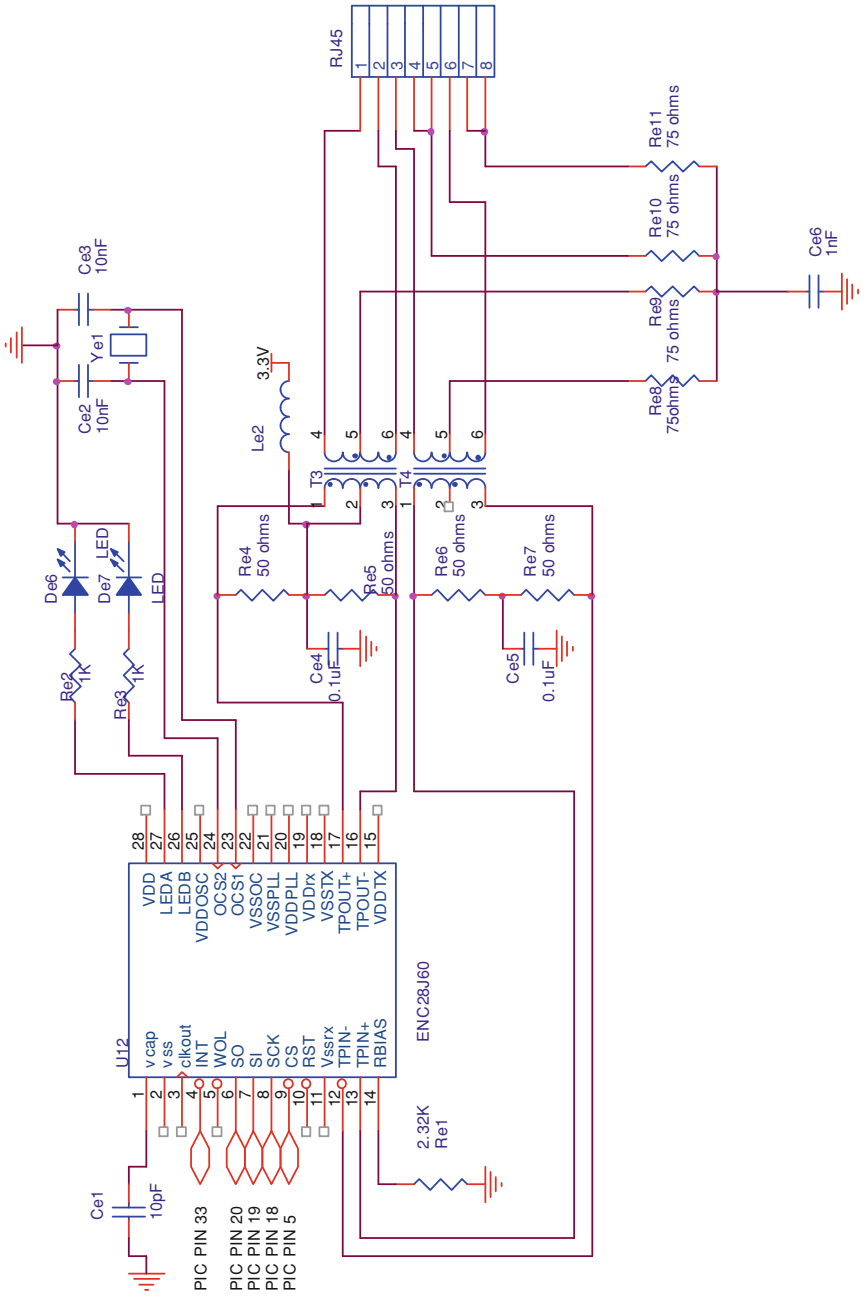


Fig. 5.24 Control of CNC milling through local area network (continued)

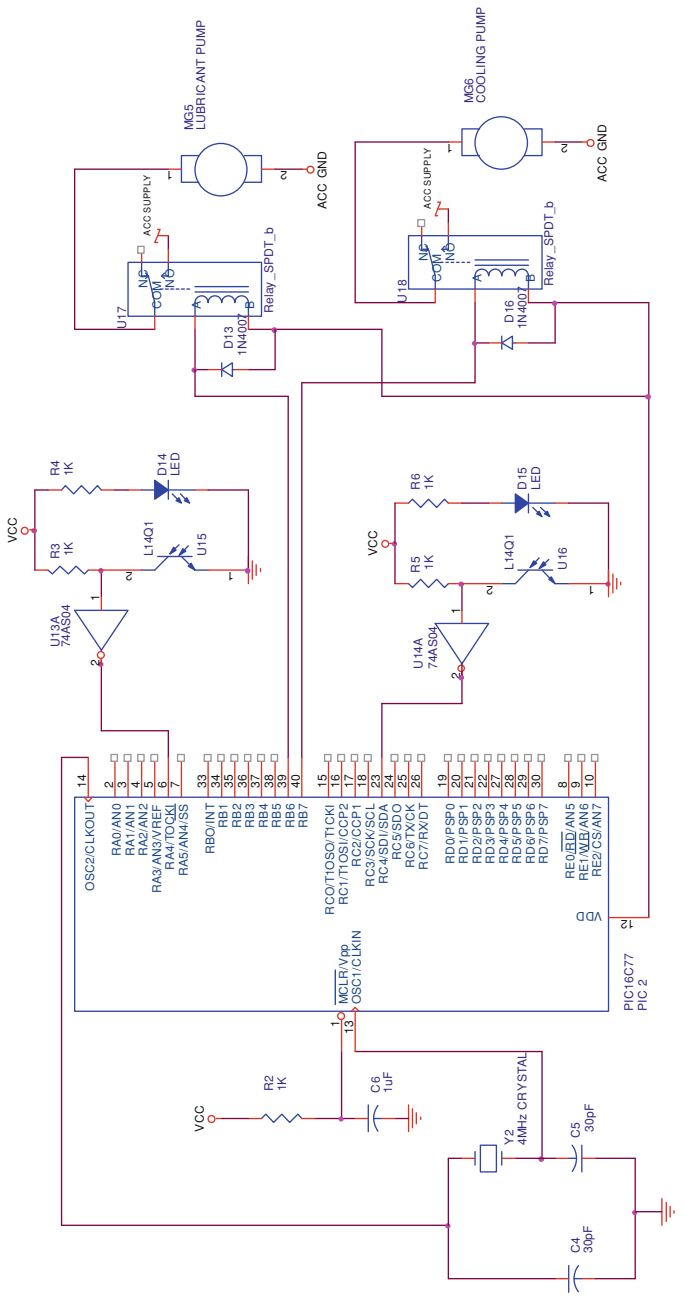


Fig. 5.24 (continued)

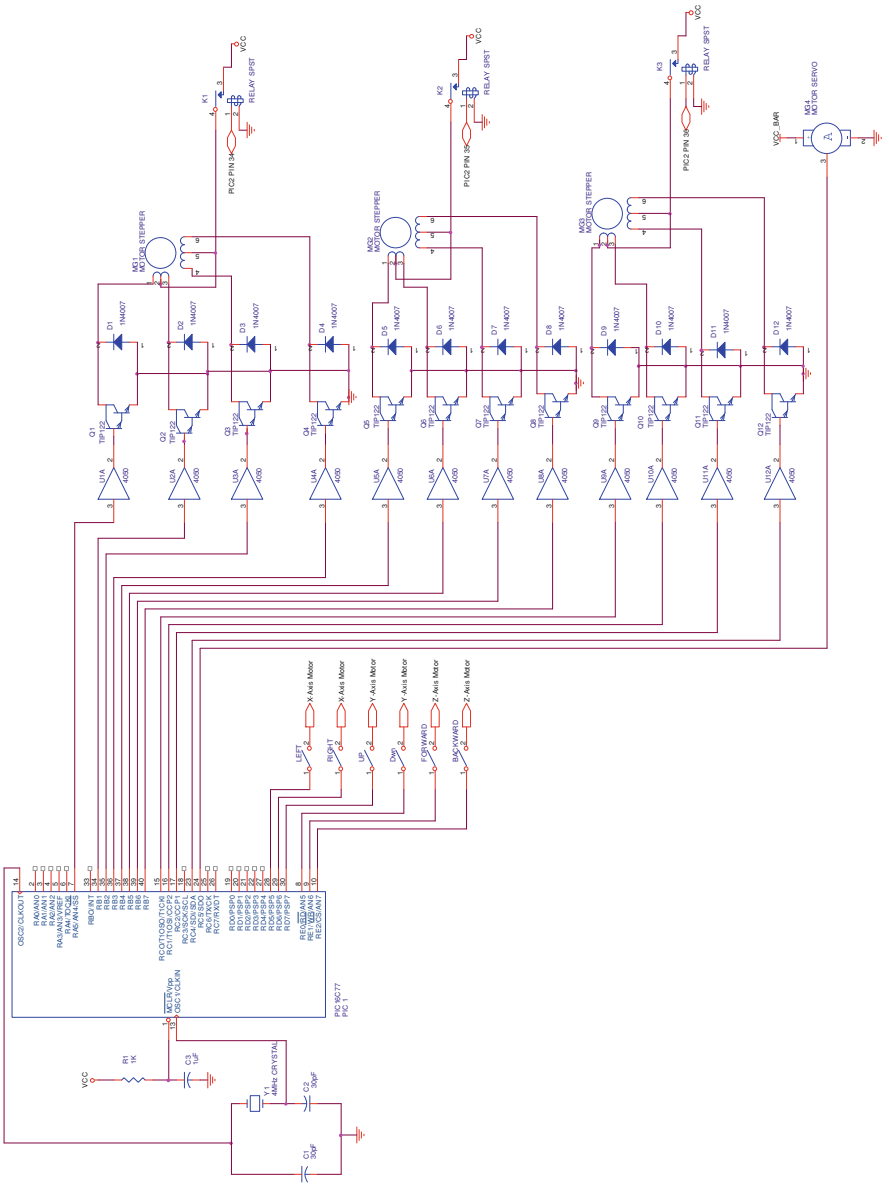


Fig. 5.24 (continued)

Table 5.42 Summary of CNC CMM CMMPortAccess class

Class summary			
Class ID: CC1			Namespace: CoordinateMeasuringMachine
Class name: CCMPortAccess			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in *x*-axis
_CurrentPositionY	Double	Private	Holds the current position of work piece in *y*-axis
_CurrentPositionZ	Double	Private	Holds the current position of work piece in *z*-axis
_Radius	Double	Private	Holds the radius of the circular movement
_I	Double	Private	Holds the starting position in *x*-axis
_J	Double	Private	Holds the starting position in *y*-axis
_NextPositionX	Double	Private	Holds the next position of work piece in *x*-axis
_NextPositionY	Double	Private	Holds the next position of work piece in *y*-axis
_NextPositionZ	Double	Private	Holds the next position of work piece in *z*-axis
_xvalue	Double	Private	Holds the *X* value
_yvalue	Double	Private	Holds the *Y* value
_zvalue	Double	Private	Holds the *Z* value
_A	Double	Private	Holds the parabolic interpolation's intermediate points
Parabola	String	Private	Store the type of parabola
_RapidMovement	Bool	Private	Holds the boolean value to rapid movement at a machine tool used for positioning the tool
_LinearInterpolation	Bool	Private	Holds the boolean value to allow direct movement between two points through linear interpolation
_CircularInterpolationClockWise	Bool	Private	Holds the boolean value for clockwise circular interpolation

(continued)

Table 5.42 (continued)

Field name	Data type	Access modifier	Description
_CircularInterpolationCounterClockWise	Bool	Private	Holds the boolean value for anticlockwise circular interpolation
_ParabolicInterpolation	Bool	Private	Holds the boolean value for parabolic interpolation
bMetricUnit	Bool	Private	Holds the boolean value for to check whether the unit is set to metric system
bAbsoluteCoordinate	Bool	Private	Holds the boolean value for to check whether the coordinate system is set to absolute coordinate
_CutterCompensationCancel	Bool	Private	Holds the boolean value to set the cutter compensation cancel
_CutterCompensationLeft	Bool	Private	Holds the boolean value to set cutter compensation to left
_CutterCompensationRight	Bool	Private	Holds the boolean value to set cutter compensation to right
_CutterOffsetInsideCorner	Bool	Public	Holds the boolean value to set cutter offset inside corner
_CutterOffsetOutsideCorner	Bool	Private	Holds the boolean value to set cutter offset inside corner
_Speed	Short	Private	Holds the value to control spindle speed
_Feed	Short	Private	Holds the feed of the spindle
_Tool	Short	Private	Holds the tool of the spindle
xPulses	Int	Private	Sets the pulse of the spindle in *x*-axis
yPulses	Int	Private	Sets the pulse of the spindle in *y*-axis
zPulses	Int	Private	Sets the pulse of the spindle in *z*-axis
_signflg	Int	Private	Holds the sign flag
_signflgy	Int	Private	Holds the sign flag in *y*-axis
_signflgx	Int	Private	Holds the sign flag in *x*-axis
_chksign	String	Private	Holds the chksign string to display
_chksignx	String	Private	Holds the chksign in *x*-axis string to display
_chksigny	String	Private	Holds the chksign in *y*-axis string to display

(continued)

Table 5.42 (continued)

Field name	Data type	Access modifier	Description
_xyzs	String	Private	Holds the XYZ coordinate string to display
_yxzs	String	Private	Holds the YXZ coordinate string to display
EmergencyStop	Static bool	Private	Holds the Boolean value to control the machine running status
Method name	**Return type**	**Access modifier**	**Description**
CalculateLinearInterpolation(double X1, double Y1, double X2, double Y2)	Void	Public	Function to calculate linear interpolation
CalculateCircularInterpolation()	Void	Public	Function to calculate circular interpolation
CalculateParabolicInterpolation()	Void	Public	Function to calculate parabolic interpolation
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
PulsRequestZ(double Z)	Void	Public	Function to set the zPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to generate pulses for physical port to control movement in *Y* plus axis.
Yminus()	Void	Public	Function to decrease the yPulses
Zplus()	Void	Public	Function to increase the zPulses
Zminus()	Void	Public	Function to decrease the zPulses
G00()	Void	Public	Function to generate point-to-point positioning
G01()	Void	Public	Function to generate linear interpolation
G02()	Void	Public	Function to generate arc clockwise (2D)
G03()	Void	Public	Function to generate arc counterclockwise(2D)
G04()	Void	Public	Function to generate dwell

(continued)

Table 5.42 (continued)

Method name	Return type	Access modifier	Description
G06()	Void	Public	Function to generate parabolic interpolation
G17()	Void	Public	Function to generate plane selection
G18()	Void	Public	Function to generate plane selection
G19()	Void	Public	Function to generate plane selection
G33()	Void	Public	Function to generate thread cutting, constant lead
G34()	Void	Public	Function to generate thread cutting, increasing lead
G35()	Void	Public	Function to generate thread cutting, decreasing lead
G40()	Void	Public	Function to generate cutter compensation/offset cancel
G41()	Void	Public	Sets machine mode to ON for cutter compensation left
G42()	Void	Public	Sets machine mode to ON for cutter compensation—right
G43()	Void	Public	Sets machine mode to ON for cutter offset-inside corner
G44()	Void	Public	Sets machine mode to ON for cutter offset-outside corner
G70()	Void	Public	Sets machine mode for inch programming
G71()	Void	Public	Sets machine mode for metric programming
G90()	Void	Public	Sets machine mode for absolute input
G91()	Void	Public	Sets machine mode for incremental input
M00()	Void	Public	Sets machine mode for program stop
M03()	Void	Public	Sets machine mode for spindle CW
M04()	Void	Public	Sets machine mode for spindle CCW
M05()	Void	Public	Sets machine mode for spindle off
M06()	Void	Public	Sets machine mode for tool change
M07()	Void	Public	Sets machine mode for coolant on—mist
M08()	Void	Public	Sets machine mode for coolant on—flood
M09()	Void	Public	Sets machine mode for coolant off
M10()	Void	Public	Sets machine mode for clamp
M11()	Void	Public	Sets machine mode for unclamp
Dwell()	Void	Public	Sets machine mode for timed delay of established duration

Table 5.43 Summary of CNC CMM Machine class

Class summary			
Class ID: CC2			Namespace: CoordinateMeasuringMachine
Class name: Machine			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with: None
Field name	Data type	Access modifier	Description
sFileName	Static string	Public	Holds the file name of .cnc file
aList	Static ArrayList	Private	Holds the GM Codes extrated from .cnc file
GMCodes	Static ArrayList	Private	Holds all GM codes
Component	Component	Private	Holds other managed resource this class uses
Disposed	Bool	Private	Track whether dispose has been called
Port	MillingPortAccess	Private	Used to call the port functions for sending GM codes
Status	MachineStatus	Private	Holds and sets the machine status
sMachineName	String	Private	Holds the machine name
Method Name	Return Type	Access modifier	Description
OpenFile(TextBox textBox)	Void	Public	Function used for open .cnc file and display in textbox object based on this function
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	Static void	Public	Function to write errors in error log
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute

(continued)

Table 5.43 (continued)

Field name	Data type	Access modifier	Description
ValidateCode (string sGMCode, ref bool bIsGMCode Valid)	Void	Public	This function will validate the GMCode
ValidateFile()	Bool	Public	This function will validate the cnc part programe file. It will take file name from static variable set by calling routine. if cnc part programe file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This function will save the part program file. It will take file name from static variable set by calling routine

Event name	Delegate	Access modifier	Description
MessageArrived	MessageHandler	Public	To generate the event of message
StatusArrived	StatusHandler	Public	To generate the event of status
PositionArrived	PositionHandler	Public	To generate the event of current position
SpindleArrived	SpindleHandler	Public	To generate the event of spindle on or off
AddLine	LineEventDelegate	Public	To generate the event to add line
WorkCompleted	Completed	Public	To generate the event of work completed

Table 5.44 Summary of CNC CMM machine3d class

Class summary			
Class ID: CC3		Namespace: CoordinateMeasuringMachine3D	
Class name: Machine3D		Class type: Concrete	
Is base class? No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Data type	Access modifier	Description
TV	TVEngine	Public	Declares the TrueVision engine object
Inp	TVInputEngine	Private	Declares the input engine variable
TVLightEngine	TVLightEngine	Private	Declares the light engine variable
MatFactory	TVMaterialFactory	Private	Declares the material factory variable
TVScene	TVScene	Private	Declares the TVScene object
Global	TVGlobals	Private	Declares the global variable
FactoryFloor	TVMeshClass	Private	Declares the mesh class for the factory floor
Light	D3DLIGHT8	Private	Declares the light variable
Lines	ArrayList	Private	Holds the list of lines
MWorkPiece	MillingWorkPiece	Private	Holds the object for work piece
sngPositionX	Float	Private	Holds object position in *x*-axis for keyboard input
sngPositionY	Float	Private	Holds object position in *y*-axis for keyboard input
sngPositionZ	Float	Private	Holds object position in z-axis for keyboard input
snglookatX	Float	Private	Holds camera position in *x*-axis for keyboard input
snglookatY	Float	Private	Holds camera position in *y*-axis for keyboard input
snglookatZ	Float	Private	Holds camera position in *z*-axis for keyboard input
tmpMouseX	Int	Private	Holds mouse input position in *x*-axis
tmpMouseY	Int	Private	Holds mouse input position in *y*-axis
tmpMouseB1	Short	Private	Holds mouse input button 1
tmpMouseB2	Short	Private	Holds mouse input button 2
tmpMouseB3	Short	Private	Holds mouse input button 3

(continued)

Table 5.44 (continued)

Field name	Data type	Access modifier	Description
tmpMouseScrollOld	Int	Private	Holds mouse input scroll old value
tmpMouseScrollNew	Int	Private	Holds mouse input scroll new value
sngAngleX	Float	Private	Holds mouse input angle in *x*-axis
sngAngleY	Float	Private	Holds mouse input angle in *y*-axis
sngWalk	Float	Private	For smooth movement while camera is walking
sngStrafe	Float	Private	For smooth movement while camera is strafing
DoLoop	Bool	Private	Holds the boolean value true until the scene is rendering
floor_x	Float	Private	Holds Initial position for factory floor in *x*-axis
floor_y	Float	Private	Holds Initial position for factory floor in *y*-axis
floor_z	Float	Private	Holds Initial position for factory floor in *z*-axis
ModelPath	String	Private	Variable for storing model paths
UpperX	Float	Private	Holds initial position for the machine in *x*-axis
UpperY	Float	Private	Holds initial position for the machine in *y*-axis
UpperZ	Float	Private	Holds initial position for the machine in *z*-axis
WorkX	Float	Private	Holds initial position for the work piece in *x*-axis
WorkY	Float	Private	Holds initial position for the work piece in *y*-axis
WorkZ	Float	Private	Holds initial position for the work piece in *z*-axis
MeshMachine	TVMesh	Private	Holds the mesh for machine
MeshWork	TVMesh	Private	Holds the mesh for work piece
MeshWorkBase	TVMesh	Private	Holds the mesh for work piece base
MeshTemp	TVMesh	Private	Holds the mesh for temporary working piece
AnimationClass	TVKeyFrameAnim	Private	For animating the 3D mesh
WorkPosition	D3DVECTOR	Private	Holds current work piece position
gWorkPosition	D3DVECTOR	Private	Holds initial work piece position
WorkBasePosition	D3DVECTOR	Private	Holds current work base position
gWorkBasePosition	D3DVECTOR	Private	Holds initial work base position

(continued)

Table 5.44 (continued)

Field name	Data type	Access modifier	Description
MachineX	Float	Private	Holds initial position for the machine in *x*-axis
MachineY	Float	Private	Holds initial position for the machine in *y*-axis
MachineZ	Float	Private	Holds initial position for the machine in *z*-axis

Method name	Return type	Access modifier	Description
ClearLines()	Void	Public	Function to clear lines on work piece
Move(Direction D)	String	Private	Function to define movement limits
MoveControl(double x1, double y1, double x2, double y2)	Void	Public	Function to add the line on the work piece
Move(string coordinate, int value)	Void	Public	Function to move the spindle in *x*, *y,* or *z*-axis
SetWorkMesh()	Void	Public	Function to set the temporary working piece mesh position and attach it to the machine mesh
MoveY(string coordinate, int value)	Void	Private	Function to move the spindle in +ve and −ve *y*-axis
ResetMachine()	Void	Public	Function to reset the work piece and base to initial position
MoveX(string Coordinate, int Value)	Void	Private	Function to move the spindle in +ve and −ve *x*-axis
MoveZ(string Coordinate, int Value)	Void	Private	Function to move the spindle in +ve and −ve *z*-axis
StartRendering()	Void	Public	Function to start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to set the DoLoop variable to false
Check_Movement()	Void	Public	Function to check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen
CollisionEnable(bool value)	Void	Public	Function to enable and disable the collision detaction
Check_Collision()	Void	Public	Function to check and control collision between CMM prob and work piece
Check_Movement()	Void	Public	Function to check and control camera movement
Check_Input()	Void	Public	Function to check keyboard input

Table 5.45 Summary of CNC CMM load part program use case

Use case summary	
Use case ID:	UC-C1
Use case name:	Load part program
Actors:	User
Description:	The user selects and loads part program through a file with "cnc" extension
Preconditions:	Part program file is present on the system
Post conditions:	Part program is loaded on the screen
System parameters	Part program file
Success scenario	Part program is loaded successfully
Alternative scenario	Error in loading part program
Includes:	Open file
Priority:	High

Table 5.46 Summary of CNC CMM save part program

Use case summary	
Use case ID:	UC-C2
Use case name:	Save part program
Actors:	User
Description:	The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions:	None
Post conditions:	Part program is saved on the file
System parameters	Part program with GM codes
Success scenario	Part program is validated and saved successfully
Alternative scenario	Error in saving part program
Includes:	Validate file, save file
Priority:	High

Table 5.47 Summary of CNC CMM execute part program use case

Use case summary	
Use case ID:	UC-C3
Use case name:	Execute part program
Actors:	User
Description:	The user executes the part program after loading it on the screen
Preconditions:	Part program is loaded and validated
Post conditions:	Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in *x*, *y,* and *z*-axes
System parameters	Part program with GM codes
Success scenario	Part program is executed successfully
Alternative scenario	Error in executing part program
Includes:	Validate file, execute file
Priority:	High

Table 5.48 Summary of CNC CMM edit part program use case

Use case summary	
Use case ID:	UC-C4
Use case name:	Edit part program
Actors:	User
Description:	The user edits the part program which is already saved
Preconditions:	Part program is selected and loaded on screen
Post conditions:	Part program is modified and saved after validation
System parameters	Part program file
Success scenario	Part program is edited successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.49 Summary of CNC CMM new part program use case

Use case summary	
Use case ID:	UC-C5
Use case name:	New part program
Actors:	User
Description:	The user inputs a new part program on the screen
Preconditions:	None
Post conditions:	Part program is validated and saved
System parameters	Part program with GM codes
Success scenario	New part program file is created successfully
Alternative scenario	Error in validating part program or in saving file
Includes:	Validate file, save file
Priority:	High

Table 5.50 Summary of CNC CMM processes sequence

Sequence Diagram Summary				
Sequence ID: SD-C1 Sequence name: CNC coordinate measuring machine sequence diagram				
Method name	Type	Description	Source	Destination
LoadPartProgram()	Message	Function to load part program	User	FrmMain
OpenFile()	Return message	Function to open file on disk	FrmMain	Support
SavePartProgram()	Message	Function to save part program	User	FrmMain
ValidateCode()	Return message	Function to validate GM codes	FrmMain	Support
SaveFile()	Message	Function to save file on disk	FrmMain	Support
EditPartProgram()	Self-message	Function to edit part program	User	FrmMain
NewPartProgram()	Self-message	Function to new part program	User	FrmMain
ExecutePartProgram()	Message	Function to execute part program	User	FrmMain
ValidateFile()	Return message	Function to validate program file	FrmMain	Support
ExecuteFile()	Return message	Function to execute program file	FrmMain	Support
Dispose()	Message	Function to close rendering and exit	User	FrmFinal
CalculateLinearInterpolation()	Message	Function to calculate linear interpolation	Support	CMMPortAccess
CalculateCircularInterpolation()	Message	Function to calculate circular interpolation	Support	CMMPortAccess
CalculateParabolicInterpolation()	Message	Function to calculate parabolic interpolation	Support	CMMPortAccess
PulsRequestX()	Message	Function to set the xPulses	Support	CMMPortAccess
PulsRequestY()	Message	Function to set the yPulses	Support	CMMPortAccess
PulsRequestZ()	Message	Function to set the zPulses	Support	CMMPortAccess
Xplus()	Message	Function to increase the xPulses	Support	CMMPortAccess
Xminus()	Message	Function to decrease the xPulses	Support	CMMPortAccess
Yplus()	Message	Function to generate pulses for physical port to control movement in *Y* plus axis.	Support	CMMPortAccess
Yminus()	Message	Function to decrease the yPulses	Support	CMMPortAccess
Zplus()	Message	Function to increase the zPulses	Support	CMMPortAccess

(continued)

Table 5.50 (continued)

Method name	Type	Description	Source	Destination
Zminus()	Message	Function to decrease the zPulses	Support	CMMPortAccess
G00()	Message	Function to generate point-to-point positioning	Support	CMMPortAccess
G01()	Message	Function to generate linear interpolation	Support	CMMPortAccess
G02()	Message	Function to generate arc clockwise (2D)	Support	CMMPortAccess
G03()	Message	Function to generate arc counterclockwise(2D)	Support	CMMPortAccess
G04()	Message	Function to generate dwell	Support	CMMPortAccess
G06()	Message	Function to generate parabolic interpolation	Support	CMMPortAccess
G17()	Message	Function to generate plane selection	Support	CMMPortAccess
G18()	Message	Function to generate plane selection	Support	CMMPortAccess
G19()	Message	Function to generate plane selection	Support	CMMPortAccess
G33()	Message	Function to generate thread cutting, constant lead	Support	CMMPortAccess
G34()	Message	Function to generate thread cutting, increasing lead	Support	CMMPortAccess
G35()	Message	Function to generate thread cutting, decreasing lead	Support	CMMPortAccess
G40()	Message	Function to generate cutter compensation/offset cancel	Support	CMMPortAccess
G41()	Message	Sets machine mode to ON for cutter compensation left	Support	CMMPortAccess
G42()	Message	Sets machine mode to ON for cutter compensation—right	Support	CMMPortAccess
G43()	Message	Sets machine mode to ON for cutter offset-inside corner	Support	CMMPortAccess
G44()	Message	Sets machine mode to ON for cutter offset-outside corner	Support	CMMPortAccess
G70()	Message	Sets machine mode for inch programming	Support	CMMPortAccess
G71()	Message	Sets machine mode for metric programming	Support	CMMPortAccess
G90()	Message	Sets machine mode for absolute input	Support	CMMPortAccess
G91()	Message	Sets machine mode for incremental input	Support	CMMPortAccess
M00()	Message	Sets machine mode for program stop	Support	CMMPortAccess
M03()	Message	Sets machine mode for spindle CW	Support	CMMPortAccess
M04()	Message	Sets machine mode for spindle CCW	Support	CMMPortAccess
M05()	Message	Sets machine mode for spindle off	Support	CMMPortAccess

(continued)

Table 5.50 (continued)

Method name	Type	Description	Source	Destination
M06()	Message	Sets machine mode for tool change	Support	CMMPortAccess
M07()	Message	Sets machine mode for coolant on—mist	Support	CMMPortAccess
M08()	Message	Sets machine mode for coolant on—flood	Support	CMMPortAccess
M09()	Message	Sets machine mode for coolant off	Support	CMMPortAccess
M10()	Message	Sets machine mode for clamp	Support	CMMPortAccess
M11()	Message	Sets machine mode for unclamp	Support	CMMPortAccess
Dwell()	Message	Sets machine mode for timed delay of established duration	Support	CMMPortAccess
AddLine()	Message	Function to add line to the final output	CMMPortAccess	frmFinal
Show()	Message	Function to show display	CMMPortAccess	frmFinal

Table 5.51 Summary of CNC CMM processes flow charts

Flow chart summary	
Flow chart ID:	FC-C1
Flow chart name:	CNC coordinate measuring machine process flow chart
No. of inputs:	7
No. of processes:	15
No. of outputs:	3
No. of control decisions:	49
Input name	Description
Input name	Description
Save program	Save button on screen
Run program	Run button on screen
Load program	Load button on screen
New program	New button on screen
Edit program	Edit button on screen
Reset	Reset button on screen
Exit application	Exit button on screen
Process name	Description
Process name	Description
Load part program from file	Function to load part program
Save part program	Function to save part program
Open dialog box	Function to open file on disk
New part program	Function to new part program
Edit part program	Function to edit part program
Reset machine	Function to reset the machine to initial position
End application	Function to close rendering and exit
Save file	Function to save file on disk
Execute part program	Function to execute program file
Set machine	Function to set machine position
Validate file	Function to validate program file
Execute file	Function to execute part program
All GM codes loop	Loop to check the GM code
Message arrived	Event to check a message
Status arrived	Event to check a status
Output name	Description
Output name	Description
Part program display	Part program loaded on screen
Final output display	Work piece display on screen
Machine position display	Machine position values on screen
Control decision name	Description
Validate part program	Check to validate GM codes

(continued)

Table 5.51 (continued)

Output name	Description
Select part program	Select the program file
Save dialog	Save dialog box
Validated	Check whether file is validated
Emergency stop	Stop rendering and reset
CalculateLinearInterpolation	Check to calculate linear interpolation
CalculateCircularInterpolation	Check to calculate circular interpolation
CalculateParabolicInterpolation	Check to calculate parabolic interpolation
PulsRequestX	Check to set the xPulses
PulsRequestY	Check to set the yPulses
PulsRequestZ	Check to set the zPulses
Xplus	Check to increase the xPulses
Xminus	Check to decrease the xPulses
Yplus	Check to increase the yPulses
Yminus	Check to decrease the yPulses
Zplus	Check to increase the zPulses
Zminus	Check to decrease the zPulses
G00	Point-to-point positioning
G01	Linear interpolation
G02	Arc clockwise (2D)
G03	Arc counterclockwise(2D)
G04	Dwell
G06	Parabolic interpolation
G17	Plane selection
G18	Plane selection
G19	Plane selection
G33	Thread cutting, constant lead
G34	Thread cutting, increasing lead
G35	Thread cutting, decreasing lead
G40	Cutter compensation/offset cancel
G41	Cutter compensation-left
G42	Cutter compensation—right
G43	Cutter offset-inside corner
G44	Cutter offset-outside corner
G70	Inch programming
G71	Metric programming
G90	Absolute input
G91	Incremental input
M00	Program stop
M03	Spindle CW
M04	Spindle CCW
M05	Spindle off
M06	Tool change
M07	Coolant on—mist
M08	Coolant on—flood
M09	Coolant off

(continued)

Table 5.51 (continued)

Output name	Description
M10	Clamp
M11	Unclamp
Dwell	A timed delay of established duration

Bibliography

1. Simulated benefits (2006) Machinery, pp 24–26
2. Advances in Artificial Reality and Tele-Existence (2006) 16th International conference on artificial reality and telexistence, ICAT
3. Alting L (1994) Manufacturing engineering processes, 2nd edn. Marcel Dekker, New York
4. Asvahem T (2004) Graphical user interface product simulator for motion control of machine path. ASEE annual conference proceedings, pp 6245–6255
5. Azuma R, Bailout Y, Behringer R, Feiner S, Julier S, MacIntyre B (2001) Recent advances in augmented reality. IEEE computer graphics and applications, November/December
6. Bedworth DD et al (1991) Computer integrated design and manufacturing. McGraw Hill, New York
7. Chang CF (2001) Haptic and aural rendering of a virtual milling process. Proceedings of the ASME design engineering technical conference, pp 105–113
8. Chen J (2006) A survey of 3D graphics software tools. IEEE computer society, CS ready notes
9. Chryssolouris G (1992) Manufacturing systems—theory and practice. Springer, New York
10. Electronics Industries Association Recommended Standard EIA RS 274 D, 1979
11. Erkorkmaz K (2006) Virtual computer numerical control system. CIRP annals—manufacturing technology, pp 399–403
12. Gong Y (2002) The application of virtual reality modeling language in NC simulation. Proceedings of the 5th international conference on frontiers of design and manufacturing, pp 132–135
13. He H (2007) Web-based virtual CNC machine modeling and operation. Chin J Mech Eng 20(6):109–113
14. Hill FS Jr (2000) Computer graphics using open GL. Prentice Hall, New Jersey
15. http://en.wikipedia.org./wiki/ASCII#ASCII_control_charactors
16. http://en.wikipedia.org/wiki/unicode
17. http://www.eia.org
18. http://www.unicode.org
19. http://en.wikipedia.org/wiki/CNC
20. Jamsa K et al (1997) VRML programmers library. Jamsa Press, Las Vegas
21. Johnson B et al (2003) Inside microsoft visual studio. NET 2003. Microsoft Press, Redmond
22. Jonsson A (2005) A virtual machine concept for real time simulation of machine tool dynamics. Int J Mach Tool Manuf 45(7–8):795–801
23. Jou M (2005) Development of an interactive e-learning system to improve manufacturing technology education. Proceedings—5th IEEE international conference on advanced learning technology, pp 359–360
24. Hitomi K (1975) Manufacturing system engineering. Taylor and Francis, London
25. Kalpakjian S, Schmid SR (2000) Manufacturing engineering and technology, 4th edn. Prentice Hall, New Jersey
26. Kao YC (2005) Development of a virtual controller integrating virtual and physical CNC. Mater Sci Forum 505–507(part 1)631–636

27. Khan WA (1990) Investigation of tool path sequencing problem in Hierarchical CIM environment. Phd thesis, University of Sheffields UK
28. Khan RA (2005) Standards for engineering design and manufacturing. Taylor and Francis, UK
29. Kief HB (1986) Flexible automation—the international CNC reference book. Becker Publishing, Havant, England
30. Kong SH (2002) The internet based virtual machining system using CORBA. Integr Manuf Syst 13(5):340–344
31. Leatham-Jones B (1986) Introduction to computer numerical control. Pitman, New York
32. Lenoir J (2006) Rapid, traditional, and virtual: prototypes in the undergraduate curriculum. Proceedings of ASME international mechanical engineering congress and exposition
33. Li G (2006) Research on digitized configuration and automated management of cutting tools for CNC machine. IET conference publications, pp 1262–1267
34. Lin F (2002) Developing virtual environments for industrial training. Inf Sci 140(1–2):153–170
35. Liu C (2000) Smalltalk, objects and design. iuniverse.com
36. Luna F (2006) Introduction to 3D game programming wit direct X 9.0 c: a shader approach. Wordware Publishing Inc, Texas
37. Luo YB et al (2002) An internet-based image and model-based virtual machine system. Int J Prod Res 40(10):2269–2288
38. Murdock KL (2000) 3D studio max: R3 Bible. Wiley, New York
39. Okafor A (2009) Development of web-based virtual CNC milling machine and machining process simulation and learning. Proceedings of the 5th IASTED European conference on internet and multimedia systems and applications, pp 36–40
40. Olwal A (2008) Spatial augmented reality on industrial CNC machines. Proceedings of SPIE
41. Ong SK (2002) An internet-based virtual CNC milling system. Int J Adv Manuf Technol 20(1):20–30
42. Quatrni T (1999) Visual modeling with rational rose 2000 and UML. Addison Wesley, Reading
43. Rogers D (2008) Setting the standard. Engineering 244(9):12–14
44. Rolt LTC (1961) A short history of machine tools. MIT Press, USA
45. Shao X (2009) A market approach to decentralized control of a manufacturing cell. Chaos Soliton Fract 39(5):2303–2310
46. Sheu JJ et al (2006) Development of an intelligent virtual reality for high speed machining. Mater Sci Forum 505–507(part 1):625–630
47. Smid P (2003) CNC programming handbook, 2nd edn. Industrial Press, USA
48. Suh SH et al (2003) Modeling ad implementation of internet-based virtual machine tool. Int J Adv Manuf 21(7):516–522
49. Susanu M (2004) Advanced axis control implementation within a virtual machine-tool environment. Proceedings of the IEEE international symposium on computer aided control system design, pp 7–12
50. Valerio NA (2002) Virtual reality for machine tool prototyping. Proc Am Soc Mech Eng cong and expo15–22
51. Wang WJ et al (2008) Research on material removal algorithm model in virtual milling process based on adaptive dynamic quadrees algorithm. Appl Mech Mater 10–12:822–827
52. Wasfy TM (2005) Virtual training environment for a 3-axis milling machine. Proceedings of the ASME international design engineering technical conference, pp 1111–1120
53. Xiaoling W (2004) Development an interactive VR training for CNC machining. Proceedings VRCAI, pp 131–133
54. Ye HW (2008) Research on visual collaborative design platform for CNC machine tools. Proceedings of the IEEE international conference on automation and logistics, pp 3010–3013
55. Yeoman RW et al (1985) CIM in the small firm—in design rule for CIM systems. North Holland

56. Zhang J (2006) A volumetric model-based CNC simulation and monitoring system in augmented environments. International conference on cyberworlds, pp 33–40
57. Zhang J et al (2010) Development of AR system achieving in situ machining simulation on a 3-axis CNC machine. Comput Animat Virtual Worlds 21(2):103–115
58. Zhu X (2006) A 3-D simulation system for milling machine based on STEP-NC. Proceedings of the world congress on intelligent control and automation (WCICA), pp 6137–6141
59. Khan WA, Raouf A (2006) Standards for engineering design and manufacturing. CRC/Taylor and Francis, USA

Chapter 6
Augmented Reality for Industrial Manipulators, Gantries and Conveyors

6.1 Introduction to Industrial Manipulators, Gantries and Conveyors

With the advent of industrial revolution the centuries old concept of 'artificial' helper became more desirable. In contemporary literature there is evidence of existence of such machines as early as first century A.D. Industrial manipulators are the current form of artificial helpers those exist since after industrial revolution.

An industrial manipulator is a device working under human control to handle materials with or without direct human contact. Discrete manufacturing systems such as job shop, project shop, cellular manufacturing system, flow lines, and flexible manufacturing systems use industrial manipulators to handle workpiece, workpiece in process, and finished products. All these states of the product are normally metallic in nature. In continuous manufacturing processes, liquids, gases, powder, and grains, which may be hazardous or non-hazardous, are supplied to and removed from the continuous manufacturing process by single or combination of industrial manipulators.

Most of the industrial manipulators are designed using control system having feed back control and degree of freedom defined through Cartesian coordinate system, polar coordinate system, or spherical coordinate system. A large variety of material handling equipment such as Robots, Cranes, and Conveyors fall under this category. This chapter covers procedure for development of augmented reality for revolute robots, gantry cranes, and conveyors.

6.2 Components of Industrial Manipulators, Gantries and Conveyors for Augmented Reality

Components of industrial manipulators for developing augmented reality comprise:

I. Mechanical Hardware

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 303
DOI: 10.1007/978-0-85729-186-8_6, © Springer-Verlag London Limited 2011

II. Electronics Hardware
III. Computer Software

The industrial manipulators are built using mechanical hardware comprising many machine elements depending on its functionality. The industrial robots are characterized by column, arm, joints, and gripper. The gantry crane is constructed using guideways, rack, rope, and a hook. The conveyor is fabricated using belt or chain, sensors, transducers, and actuators are also the most common part of the industrial manipulators. Augmented reality for these machines requires building their graphics model, rendering it on the computer screen and physically connecting it to the controller of individual machine. This operation is performed according to the basic definition of the augmented reality, i.e., augmented reality combines real and augmented objects and it runs interactively and in real times. Also the real and augmented objects align with each other.

To meet the above objectives of augmented reality for industrial manipulators special electronic hardware is required that physically connects the mechanical hardware with the controller of the system and the computer displaying the augmented reality of the system. The key design aspect of such electronic hardware is that it maps the address, data, and control lines to system controller and augmented reality controller. A rugged digital electronic systems as depicted in Fig. 6.1 is required to work in industrial environment which is characterized by protection from noise, vibration, and humidity.

The third component of developing augmented reality for industrial manipulators is the computer software. The computer software is required to capture the electronics signals from the electronics hardware, depict the graphics model of the

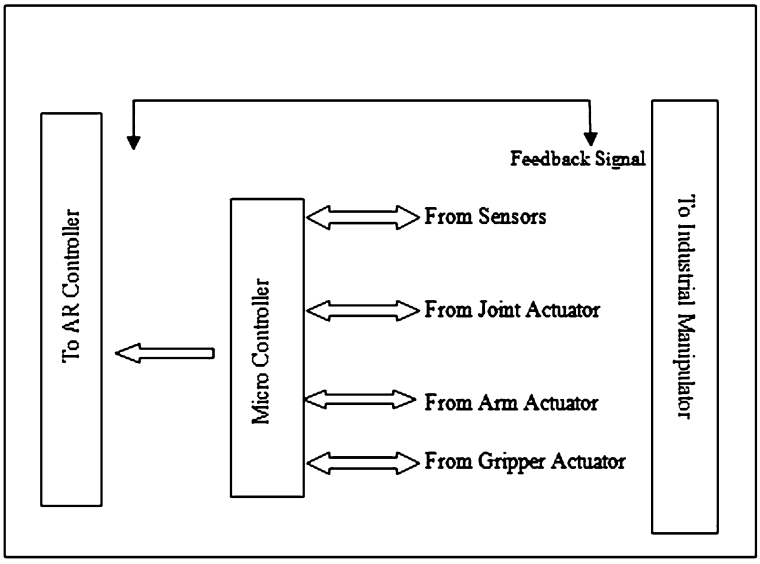


Fig. 6.1 Block Diagram for AR Interface for industrial manipulator

industrial manipulator in current state, and evaluate parameters those assist in decision making at micro- and macro-level in the virtual organization.

Complex system such as robot requires application software operating through special high level language because of the number and type of distinct operation they perform. Japanese Industrial Standard (JIS) recommends Standard Language for Industrial Manipulator (SLIM) for such application. An appropriate software compiler for such a language needs to be developed. For simpler operations such as the operation of gantry and conveyors, software interpreters as used in the case of CNC may be used.

6.3 Standards Pertaining to Augmented Reality for Industrial Manipulator, Gantry and Cranes

In order to define augmented reality for industrial manipulators, Gantry and Conveyor, it is imperative to use standard specification. Some of the relevant standards from various standard organizations are listed below:

I. BS EN ISO 11593
MANIPULATING INDUSTRIAL ROBOTS—AUTOMATIC END EFFECTOR SYSTEMS—VOCABULARY AND PRESENTATION OF CHARACTERISTICS
II. BS EN ISO 15187
MANIPULATING INDUSTRIAL ROBOTS—GRAPHICAL USER INTERFACES FOR PROGRAMMING AND OPERATION OF ROBOTS (GUI-R)
III. BS EN ISO 8373
MANIPULATING INDUSTRIAL ROBOTS—VOCABULARY
IV. BS EN ISO 9787
MANIPULATING INDUSTRIAL ROBOTS—COORDINATE SYSTEMS AND MOTION NOMENCLATURES
V. ISO 10218-1
ROBOTS FOR INDUSTRIAL ENVIRONMENTS—SAFETY REQUIREMENTS—PART 1: ROBOT
VI. JIS B 8439
INDUSTRIAL ROBOTS—PROGRAMMING LANGUAGE SLIM
VII. ASTM F 1034
CLASSIFYING INDUSTRIAL ROBOTS
VIII. JIS B 8435
GENERAL CODE OF DESIGN FOR MODULARIZATION OF INDUSTRIAL ROBOTS
IX. ISO 17874-2
REMOTE HANDLING DEVICES FOR RADIOACTIVE MATERIALS—PART 2: MECHANICAL MASTER–SLAVE MANIPULATORS

X. ASME B30.17
OVERHEAD AND GANTRY CRANES (TOP RUNNING BRIDGE, SINGLE GIRDER, UNDERHUNG HOIST)
XI. ASME NOG 1
RULES FOR CONSTRUCTION OF OVERHEAD GANTRY CRANES (TOP RUNNING BRIDGE, MULTIPLE GIRDER)
XII. BS ISO 4306-5
CRANES—VOCABULARY—PART 5: BRIDGE AND GANTRY CRANES
XIII. CMAA 70
SPECIFICATIONS FOR TOP RUNNING BRIDGE & GANTRY TYPE MULTIPLE GIRDER ELECTRIC OVERHEAD TRAVELING CRANES
XIV. ISO 10972-5
CRANES—REQUIREMENTS FOR MECHANISMS—PART 5: BRIDGE AND GANTRY CRANES
XV. ANSI B30.17
OVERHEAD & GANTRY CRANES
XVI. ASME B20.1
SAFETY STANDARD FOR CONVEYORS AND RELATED EQUIPMENT
XVII. BS 2890
SPECIFICATION FOR TROUGHED BELT CONVEYORS
XVIII. BS 4531
SPECIFICATION FOR PORTABLE AND MOBILE TROUGHED BELT CONVEYORS
XIX. BS 8438
TROUGHED BELT CONVEYORS—SPECIFICATION
XX. FORD CX9
FLAT BELT CONVEYORS
XXI. MIL-C-11218
CONVEYORS, ROLLER AND WHEEL, GRAVITY AND CONVEYOR SUPPORTS
XXII. MIL-C-12584
CONVEYORS,DRAG: SELF-PROPELLED, ELEC,GAS, ENG. DRIVE

6.4 Augmented Reality Design for Industrial Manipulator

The augmented reality for industrial manipulator is constructed in a similar manner as described in Sect. 5.4 for the computer numerical control-based applications. The functionality of the software developed to control the augmented reality for industrial manipulator and to integrate the proxy to real process differ in

the case of robots, gantry, crane, and conveyor. A compiler based on Japanese Industrial Standard (JIS) which recommends Standard Language for Industrial Manipulator (SLIM) is described, while due to the simple operation of the Gantry crane and Conveyor an interpreter using indigenous command is developed. Section 6.4.1 describes the set of commands chosen for developing the SLIM-Compiler. Sections 6.5 and 6.6 deal with the development of interpreters for Gantries and Conveyor.

6.4.1 SLIM Command Set for Industrial Manipulator

JIS SLIM is a commonly used standard for developing user interface for the robots. A selection of command used here is listed below:

I. grasp

The grasp statement is the statement which designates to close the end-effector.

II. release

The release statement is the statement which designates to open the end-effector.

III. drive

The drive statement is the statement which controls each axis directly. The syntax of the drive command is as follows:
drive (axis number, angle of rotation)
SLIMCompiler recognizes this command in two formats:

1. drive (axis number, angle of rotation)
2. drive (axis number, angle of rotation) (axis number, angle of rotation)

example: drive (3,45)
example: drive (3,45)(2,90)

IV. rotate

The rotate statement is the statement which allows a robot to rotate by a designated angle in a designated rotational plane.
rotate Plane Specifier, Angle, Center of Rotation
example: rotate xy, 45, 3

V. speed

The speed statement is the statement which designates the transfer speed of the fingers of a robot.
example: speed 4

VI. delay

The delay statement is the statement with which the execution of a program waits for a designated time in ms.
example: delay 5000

VII. out

The out statement shall be the statement which outputs data to the port address indicated by I/O variable names.
out IOVariable, Data to be sent
example:
integer ioaddress = 888
integer iodata = 1101
out ioaddress,iodata

VIII. in

The in statement is the statement which inputs data from the port address indicated by I/O variable names in I/O data variable. The syntax of the command is:
in iodatavariable = ioaddress
example:
integer ioaddress = 888
in iodata = ioaddress
A program statement for this command is given below:
if iodata == 1101 then
do some thing
else
do some thing
end

IX. pose

The pose statement is the statement that declares position variable. SLIM-Compiler supports two type of pose statements:

1. pose variable declaration

example: pose mypose
NOTE : In the above statement declared pose variable name is “mypose” but at this time this variable have default x, y, z coordinates to 0.

2. pose variable declaration with initialization

pose mypose (x coordinate, y coordinate, z coordinate)
example:
pose mypose (45,80,90)

X. letx

The letx command sets the pose variable's x coordinate.
example
letx mypose = 45

XI. lety

The lety command sets the pose variable's y coordinate.
example
lety mypose = 45

XII. letx

The letz command sets the pose variable's z coordinate.
example
letz mypose = 45

XIII. home

The home statement is the statement which defines a home pose. The home statement shall usually be used being combined with a gotohome statement. The home pose which has been defined once shall be valid until redefined by the home statement.
example:
home mypose
NOTE: When the home pose has not been defined in the program, it shall not be possible to move to the home pose using the gotohome statement.

XIV. gotohome

The gotohome statement is the statement which moves the robot to its home location.
example:
gotohome

XV. move

The move statement is the statement which allows the robot to transfer from the present pose to the pose indicated by the pose variables. The example in the case where to transfer from the present pose to posetwo through poseone.
example:
move L, poseone, posetwo
The second statement of move command is use for move from current location indicated by '*' sign to new location without using pose variable
example:
move L, * (45,90,60)

XVI. SLIM Language Keywords:

The SLIM keywords are

- integer
- string
- if
- then
- else
- end
- while
- do

XVII. SLIM Arithmetic Operators:
Following arithmetic operators are used:

- + (Plus)

– (Minus)
* (Multiplication)
/ (Division)

XVIII. SLIM Relational Operators:
The relational operators used are

- < (Less than)
- <= (Less than or equal to)
- > (greater than)
- > = (Greater than or equal to)
- <> (Not Equal to)
- == (Equal to)

XIX. SLIM Assignment Operators:
The SLIM assignment operator is:

- =

XX. SLIM Comments:
The SLIM comment statements are

- Comment Line//
- Comment Start/*
- Comment End */

6.4.2 Software Compiler Design Based on JIS SLIM

A Compiler is a suite of computer programs that performs lexical analysis, pre-processing, parsing, semantic analysis and converts code into binary form known as object code. The objective of this exercise is to obtain an executable program.

Contrary to the interpreter's operation, the compiler operates on the complete source code once and no executable code is generated unless all the bugs are removed. Afterward the executable program can be run repeatedly without any translation stage. In the case of an interpreter the translation and execution of command is after each other and no compilation phase takes place. Figures 6.2, 6.3, 6.4, and 6.5, presented in the following pages depict use Case Diagram, Sequence Diagram, UML Static Diagrams, and Flow charts use to develop SLIMCompiler. SLIM Robot is consisting of three projects:

1. SLIM: Consists of six classes; Parser class, TParser class, SLIMParser class, Position class, PositionArray class, GenerateCode class, and two events. Object of SLIMParser class is used in SLIM Robot software to provide basic functionality such as compile and execute SLIM program. SLIMParser is inherited by TParser class which is also inherited by Parser class. SLIM is a Dynamic Link Library (DLL)-based project. The produced DLL of SLIM is used in SLIMCompiler project to provide separation between application and logic and provide logic encapsulation.

   Class structure of SLIM: SLIM has encapsulated all the main functionality required by SLIMCompiler Application such as compile SLIM File and, execute SLIM file. A request from SLIMCompiler Application for each function is handled by SLIMCompiler class. Two events of SLIM are MessageHandler and SLIMCommandHandler. MessageHandler event is used to send any compilation-related messages including error message to SLIMCompiler Application to notify user. SLIMCommandHandler event will be used to send current SLIM command or Token which is currently executed by Parser to SLIMCompiler Application to control the 3D robot movement.

2. SLIMCompiler: Consists of a class Form1 which is inherited by System.Windows.Forms.Form to provide SLIMCompiler Application graphical user interface that utilizes SLIM object to compile and execute SLIM file. Current status of execution and controlling of 3D animation of Robot is also updated from SLIMCompiler. Enumerations PlaneSpecifier and Interpolation are also implemented in SLIMCompiler project. PlaneSpecifier specifies the plane of movement and Interpolation specifies the movement interpolation of 3D robot.
3. Robot: Consists of an Animation class. Animation class has 3D graphics rendering engine and methods to control movement of individual parts of Robot. Animation class object is used by SLIMCompiler Application to control animation of robot.

Table 6.1 shows the SLIMParser class members properties, methods, and variables. SLIMParser is the helper class of SLIMCompiler software and it is used to parse the tokens of the program file and create grammar for the parsed statements and execute each statement. During the execution of statements it generates SLIMCommandHandler event to update SLIMCompiler software. It inherits from TParser class which contains the basic functions for syntax parsing.

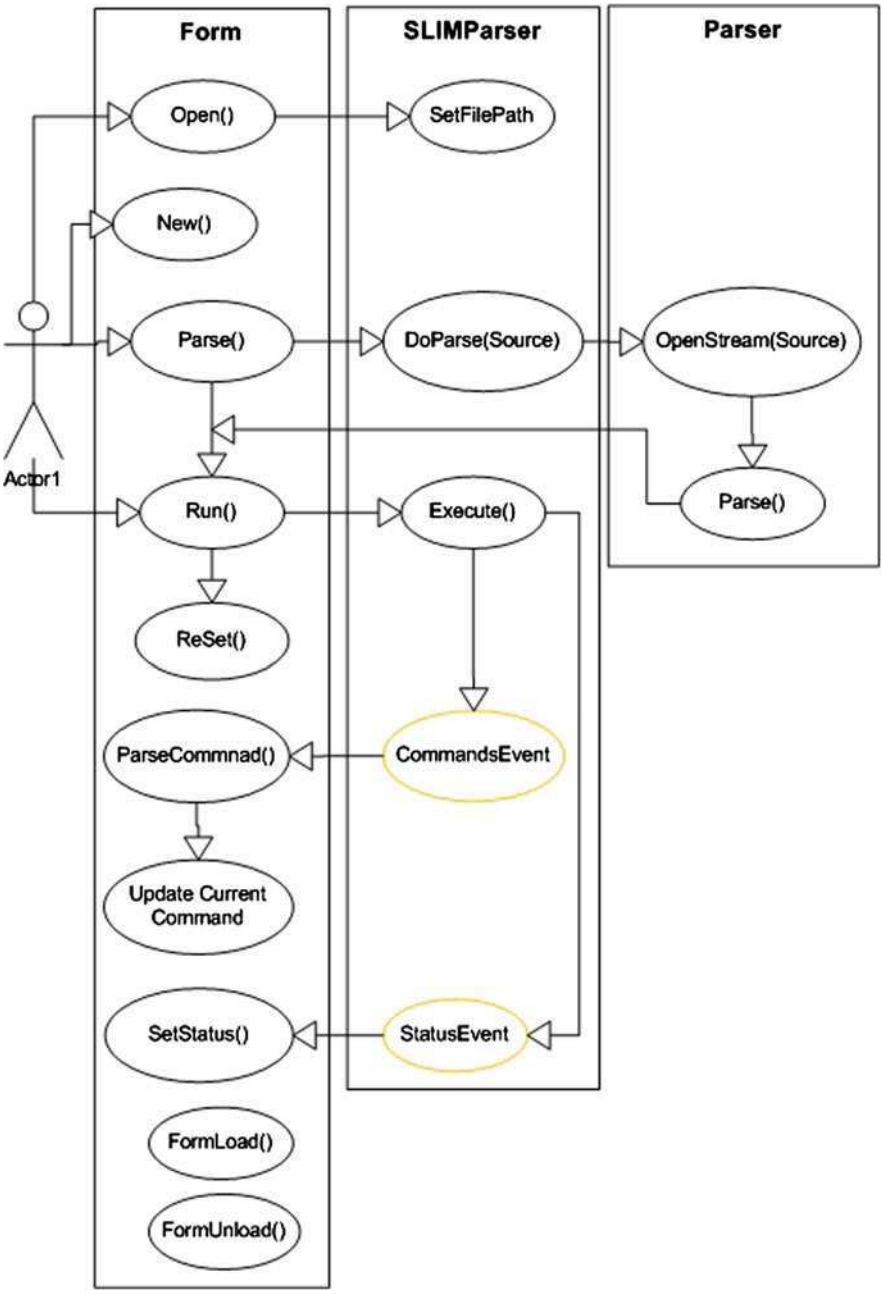


Fig. 6.2 Use Case Diagram of SLIMCompiler

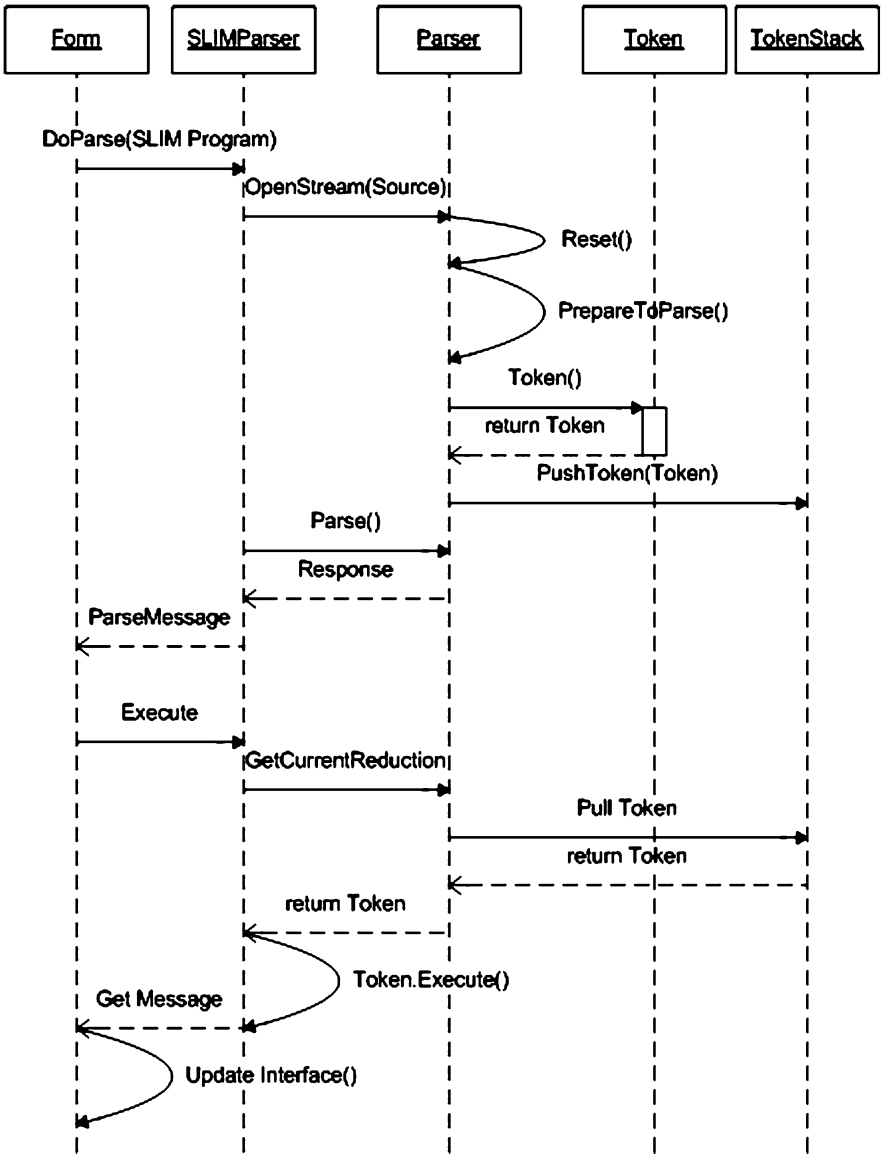


Fig. 6.3 SLIMCompiler UML Sequence Diagram

Table 6.2 presents the TParser member variables and methods. TParser class uses CreateNewObject method to create SLIM language statements by following the rules in RuleConstants and SymbolConstants enumerations of TParser class.

Table 6.3 depicts the members variables of Position class. Position class is used to hold the current and home position of the robot.

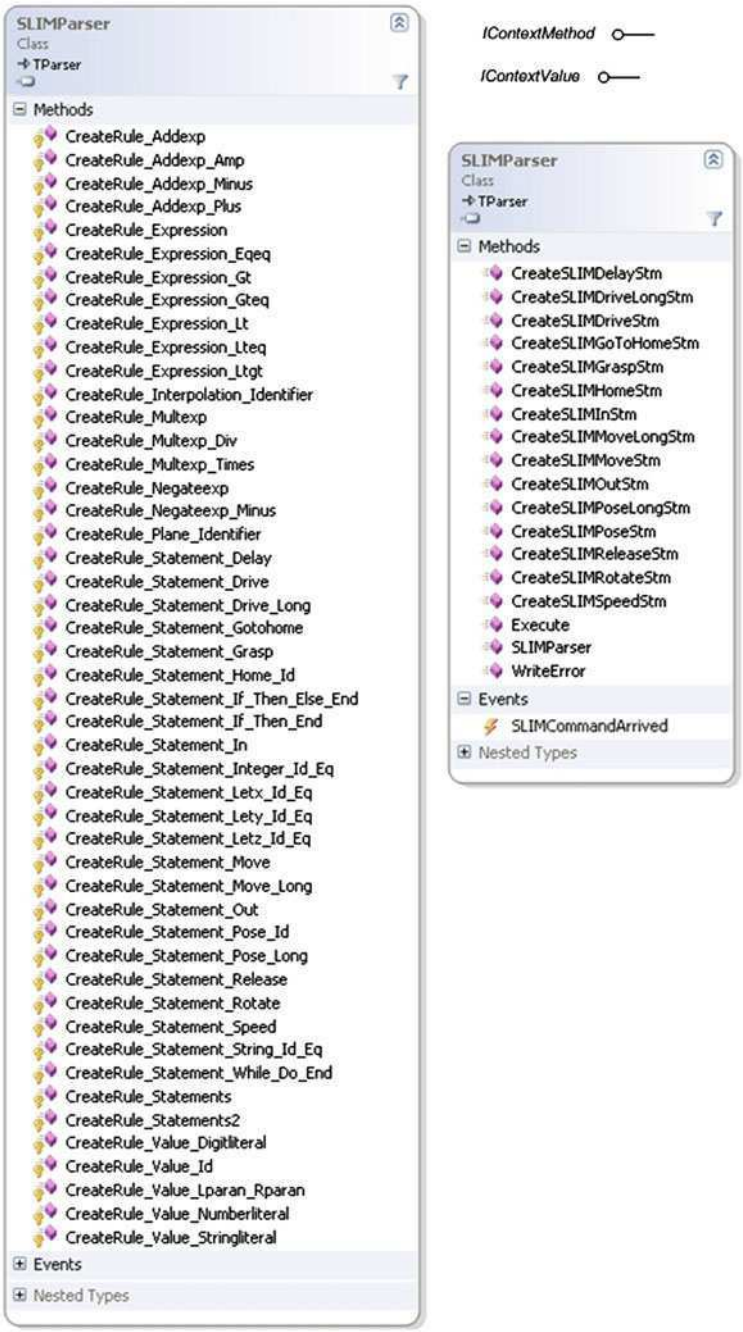


Fig. 6.4 UML Static Diagram of SLIMCompiler (continued)

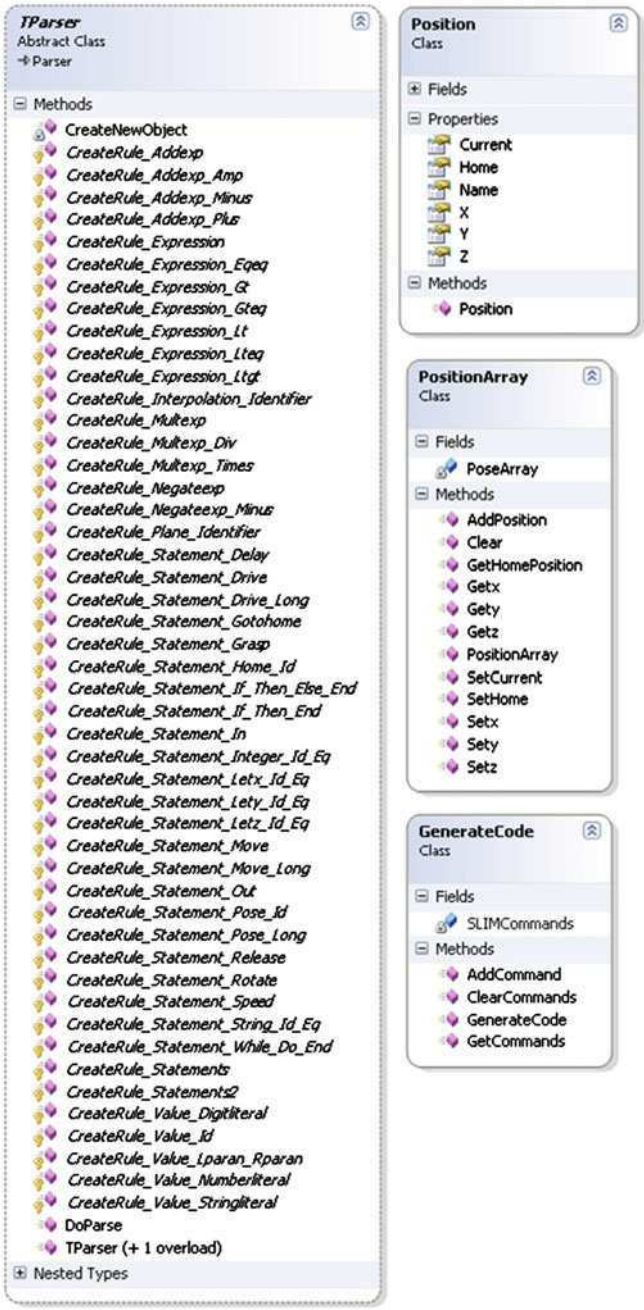


Fig. 6.4 (continued)

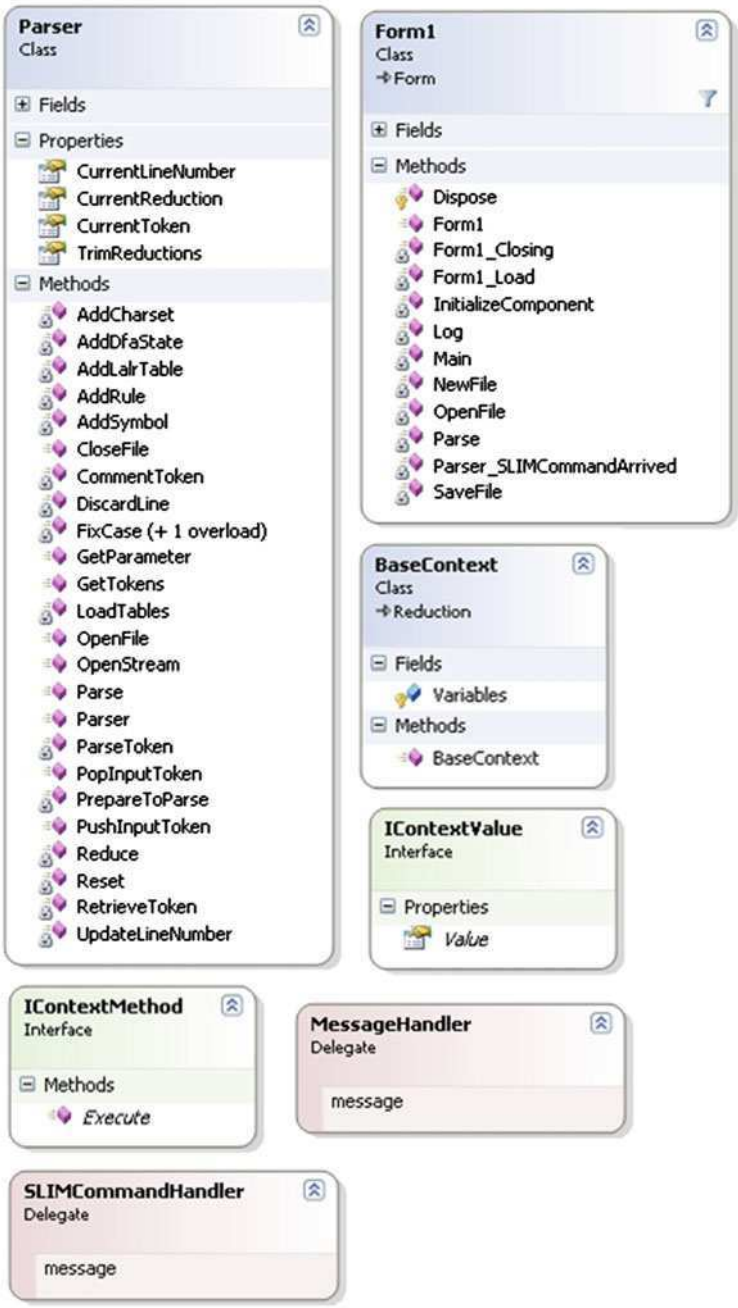


Fig. 6.4 (continued)

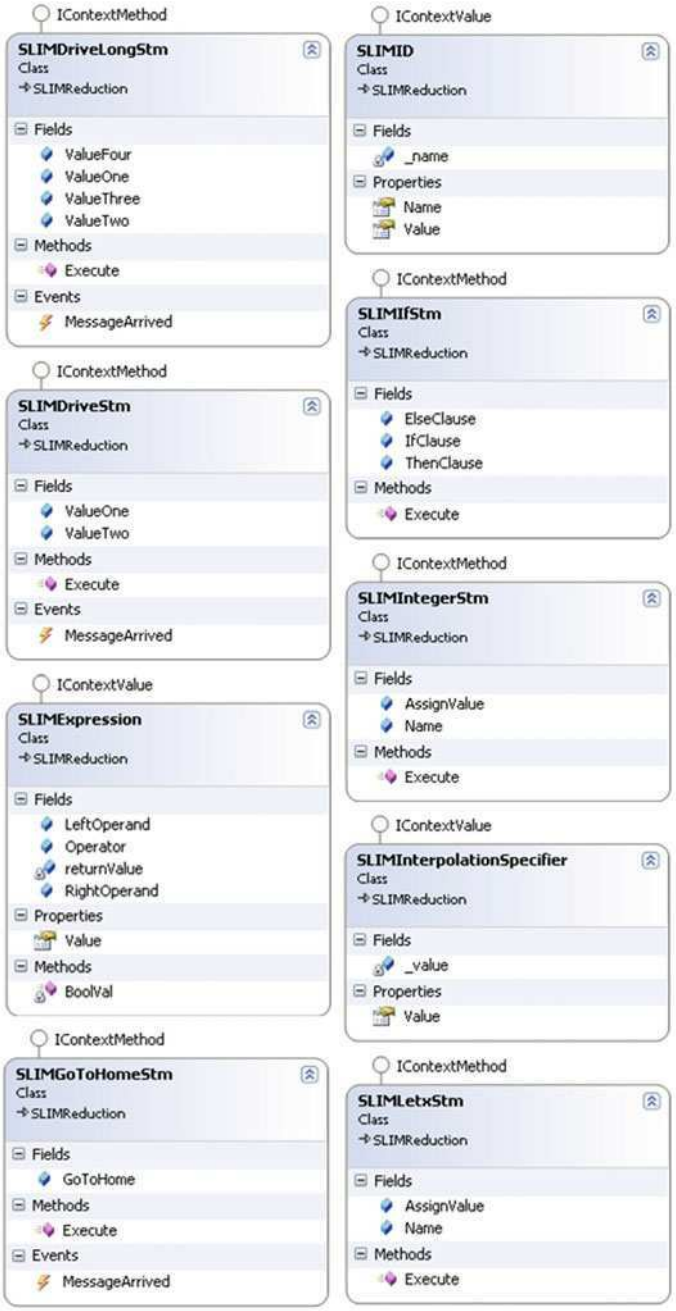


Fig. 6.4 (continued)

Fig. 6.4 (continued)

Fig. 6.4 (continued)

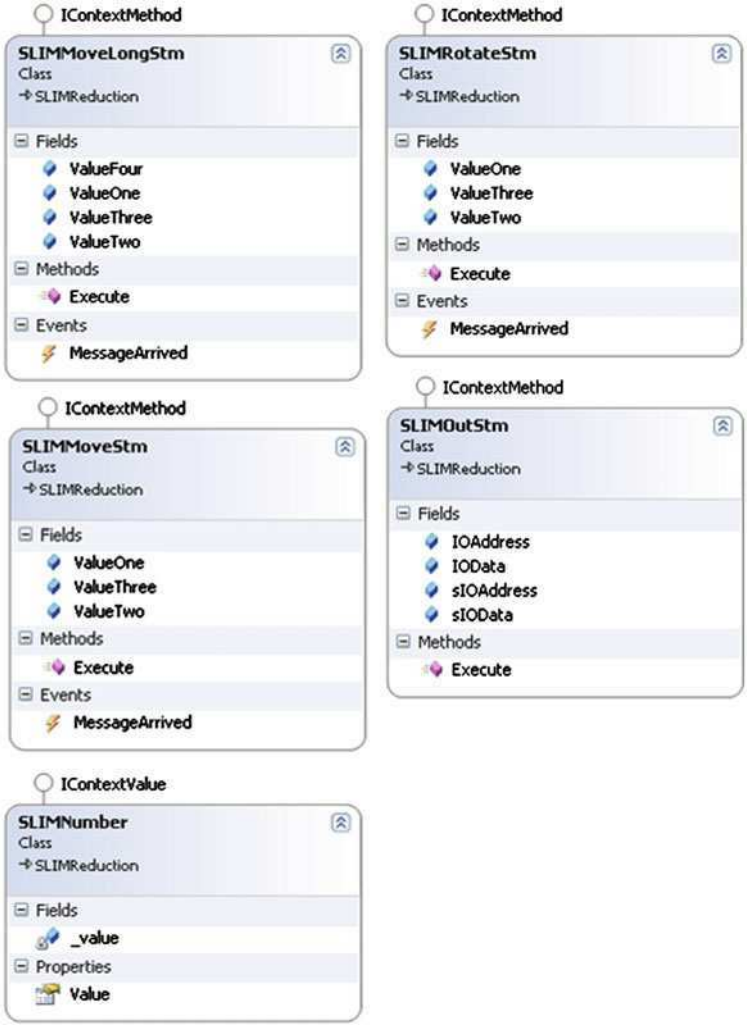


Fig. 6.4 (continued)

Table 6.4 describes the members of PositionArray class. PositionArray class is used to hold the Position objects, to obtain and set positions of robot.

Table 6.5 defines the members of GenerateCode class. GenerateCode class is used to hold the SLIM commands generated by SLIMParser.

Table 6.6 illustrates the summary of SLIMCompiler sequence diagram.

Table 6.7 explains the SLIMCompiler open SLIM program use case.

Table 6.8 outlines the SLIMCompiler new SLIM program use case.

RuleConstants
Enum

RULE_STATEMENTS
RULE_STATEMENTS2
RULE_STATEMENT_INTEGER_ID_EQ
RULE_STATEMENT_STRING_ID_EQ
RULE_STATEMENT_GRASP
RULE_STATEMENT_RELEASE
RULE_STATEMENT_POSE_ID
RULE_STATEMENT_LETX_ID_EQ
RULE_STATEMENT_LETY_ID_EQ
RULE_STATEMENT_LETZ_ID_EQ
RULE_STATEMENT_POSE_ID_LPARAN_COMMA_COMMA_RPARAN
RULE_STATEMENT_HOME_ID
RULE_STATEMENT_GOTOHOME
RULE_STATEMENT_MOVE_COMMA_ID_COMMA_ID
RULE_STATEMENT_MOVE_COMMA_TIMES_COMMA_LPARAN_COMMA_COMMA_RPARAN
RULE_STATEMENT_DRIVE_LPARAN_COMMA_RPARAN
RULE_STATEMENT_DRIVE_LPARAN_COMMA_RPARAN_COMMA_LPARAN_COMMA_RPARAN
RULE_STATEMENT_ROTATE_COMMA_COMMA
RULE_STATEMENT_WHILE_DO_END
RULE_STATEMENT_IF_THEN_END
RULE_STATEMENT_IF_THEN_ELSE_END
RULE_STATEMENT_SPEED
RULE_STATEMENT_DELAY
RULE_STATEMENT_OUT_COMMA
RULE_STATEMENT_IN_ID_EQ
RULE_EXPRESSION_GT
RULE_EXPRESSION_LT
RULE_EXPRESSION_LTEQ
RULE_EXPRESSION_GTEQ
RULE_EXPRESSION_EQEQ
RULE_EXPRESSION_LTGT
RULE_EXPRESSION
RULE_ADDEXP_PLUS
RULE_ADDEXP_MINUS
RULE_ADDEXP_AMP
RULE_ADDEXP
RULE_MULTEXP_TIMES
RULE_MULTEXP_DIV
RULE_MULTEXP
RULE_NEGATEEXP_MINUS
RULE_NEGATEEXP
RULE_PLANE_PLANEIDENTIFIER
RULE_INTERPOLATION_INTERPOLATIONIDENTIFIER
RULE_VALUE_ID
RULE_VALUE_DIGITLITERAL
RULE_VALUE_NUMBERLITERAL
RULE_VALUE_STRINGLITERAL
RULE_VALUE_LPARAN_RPARAN

Fig. 6.4 (continued)

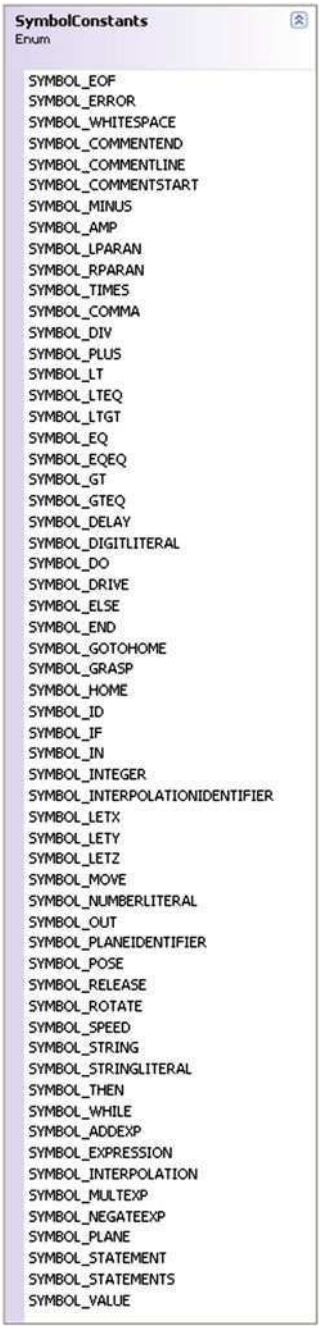


Fig. 6.4 (continued)

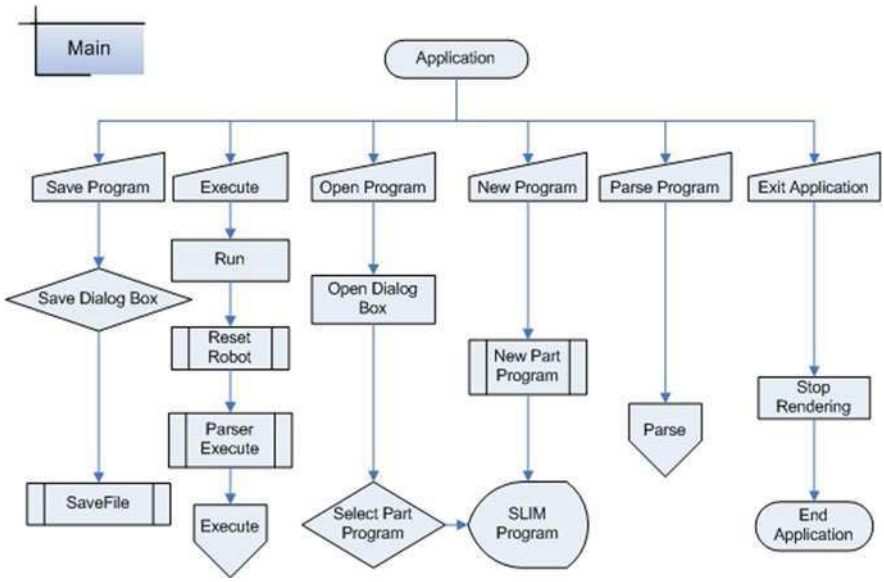


Fig. 6.5 SLIMCompiler Flow Chart

Table 6.9 depicts the SLIMCompiler Parse SLIM program use case.
Table 6.10 outlines the SLIMCompiler Run SLIM program use case.
Table 6.11 defines the SLIMCompiler processes flow charts.
Execution of SLIM commands or SLIM program by using SLIMCompiler is a two-step process.

I. Parse Program
In this process SLIM program is converted into stream of bytes. Then bytes stream are tokenized and each token is check against SLIM grammar before push into Input Token Stack. Any invalid token causes to fail the parse process and update the user about the failure.

II. Execute Program
In Execute process each token is pulled from Input Token Stack. This process is called reduction. Afterward each token Execute method is called. The following code shows the Parse method that calls CreateNewObject with CurrentReduction as parameter:

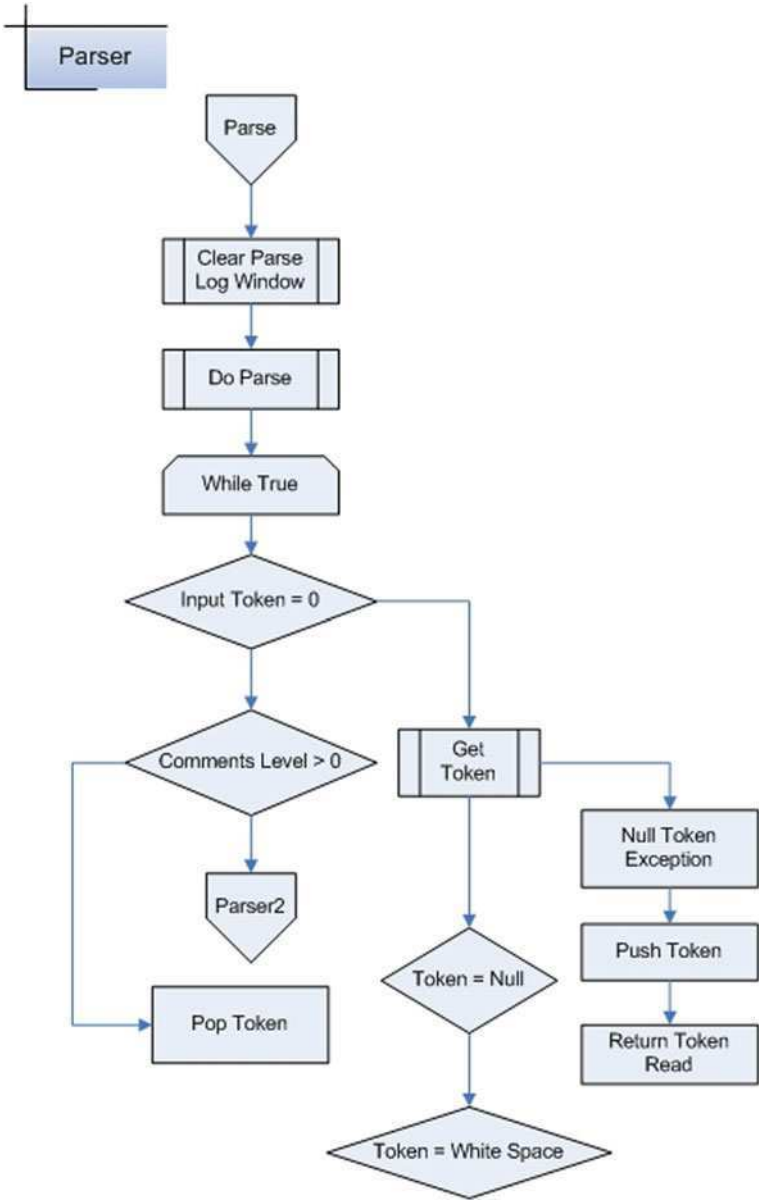


Fig. 6.5 (continued)

Code Description

The following partial code is taken from TParser class. DoParse method of TParser class takes the bytes stream of SLIM program and passes this stream to

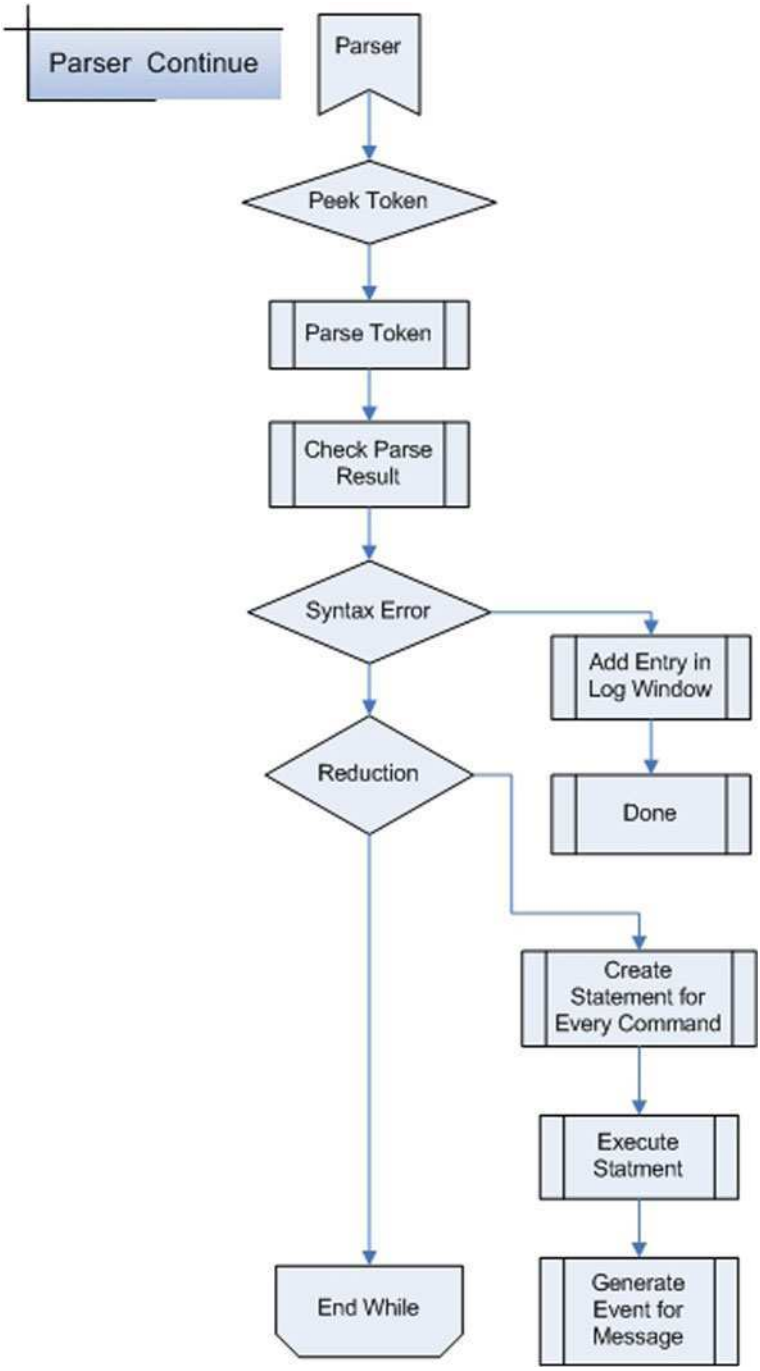


Fig. 6.5 (continued)

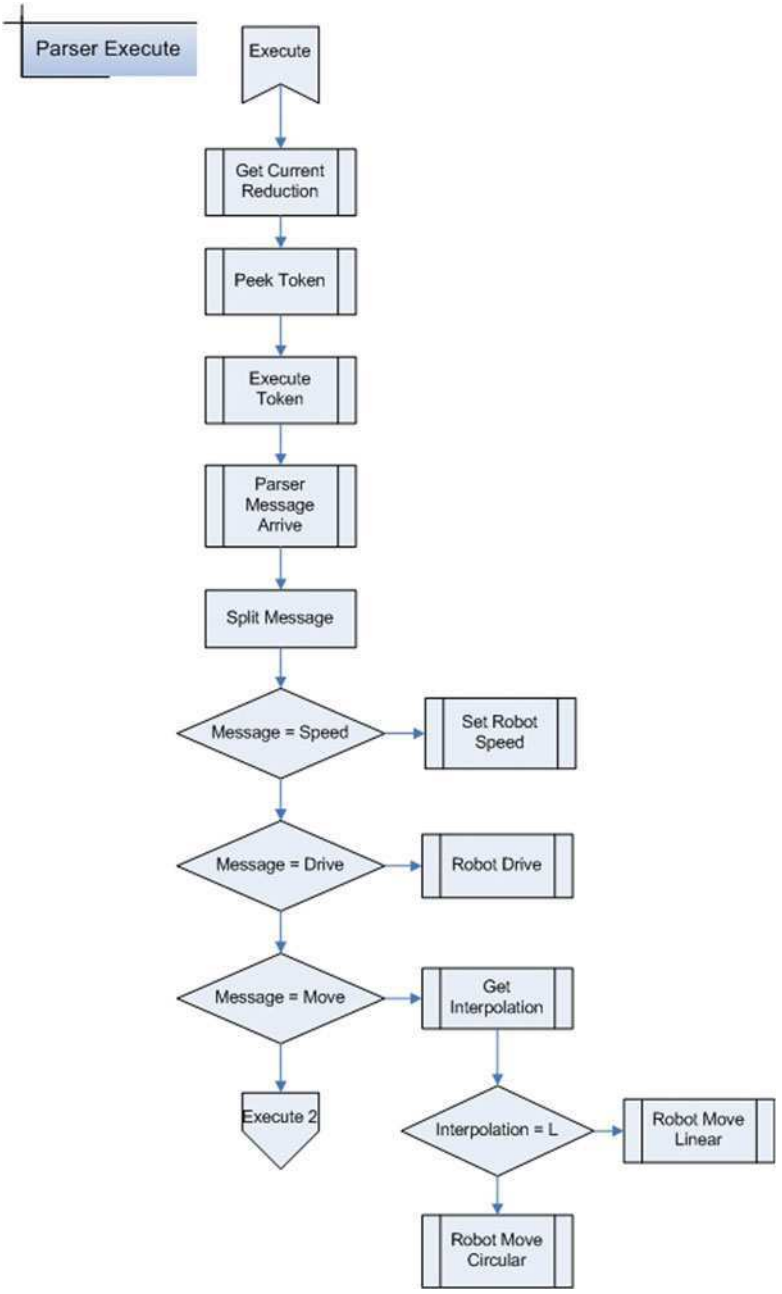


Fig. 6.5 (continued)

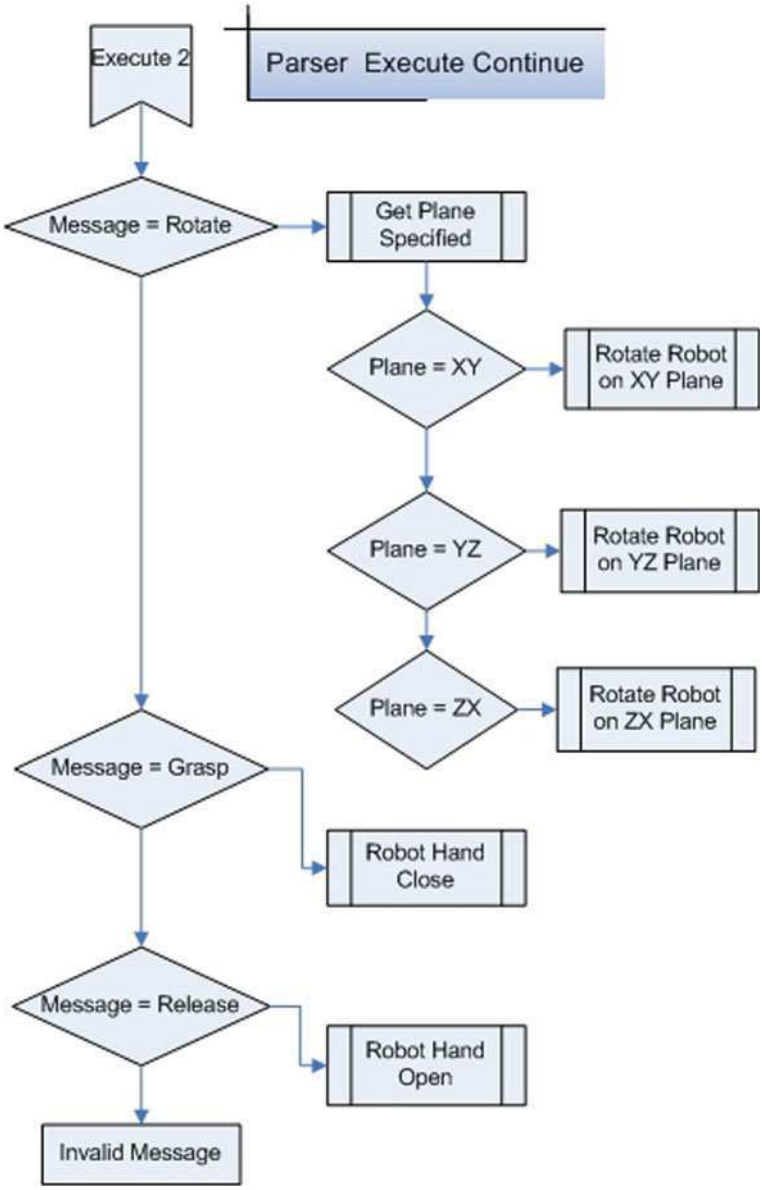


Fig. 6.5 (continued)

OpenStream method. After OpenStream, response is checked against the parse process. If there is an error in response, error message is returned through ParseMessage and if there is reduction in response, a new SLIM statement is created for that reduction by calling TParser CreateNewObject method.

```
public ParseMessage DoParse(System.IO.TextReader Source,bool Genera-
teContext)
                {

                        ParseMessage Response;
                        bool Done; //Controls when we leave the loop

                        OpenStream(Source);
                        TrimReductions = true;
                        Done = false;
                        do
                        {
                                Response = Parse();

                                switch (Response)
                                {
                                        case ParseMessage.LexicalError:

                                                //Cannot recognize token
                                                Done = true;
                                                break;

                                        case ParseMessage.SyntaxError:

                        //Expecting a different token
                                        Done = true; // stop
                                                                break;

                                        case ParseMessage.Reduction:

                                         //Create a customized object to
                                         store the reduction
                                                if (GenerateContext)
                                                {
                                                        CurrentReduction
                                                        = CreateNewOb-
                                                        ject(CurrentRedu
                                                        ction);
                                                }
                                                break;

                                        case ParseMessage.Accept:

                                                //Success!
                                                Done = true;
                                                //Success = true;
                                                break;

                                        case ParseMessage.TokenRead:
```

```
                                        break;
                                //You don't have to do anything here.

                                case ParseMessage.InternalError:
                                        //INTERNAL ERROR

        Done = true;
                                        break;

                                case ParseMessage.CommentError:

                                        //COMMENT ERROR! Unex-
pected end of file
                                        Done = true;
                                        break;
                                }
                        } while (!(Done));
                        return Response;

                }
```

The following partial code is taken from TParser class. This code shows CreateNewObject method of TParser class by calling DoParse method. CreateNewObject takes the current Reduction as parameter and create new SLIM statement for that reduction. Each reduction is checked against RuleConstants enumeration and returns back newly created statement through Result Reduction.

```
private Reduction CreateNewObject(Reduction TheReduction)
               {
                       Reduction Result;
                       switch(TheReduction.ParentRule.TableIndex)
                       {
                                       #region case
                                case
(int)RuleConstants.RULE_STATEMENTS:
                                       //<Statements> ::= <Statement>
<Statements>
                                       Result = Create-
Rule_Statements(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENTS2:
                                       //<Statements> ::= <Statement>
                                       Result = Create-
Rule_Statements2(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENT_SPEED:
                                       //<Statement> ::= speed <Expres-
sion>
                                       Result = Create-
Rule_Statement_Speed(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENT_DELAY:
                                       //<Statement> ::= delay <Expres-
sion>
                                       Result = Create-
Rule_Statement_Delay(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENT_LETX_ID_EQ:
                                       //<Statement> ::= letx Id '='
<Expression>
                                       Result = Create-
Rule_Statement_Letx_Id_Eq(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENT_LETY_ID_EQ:
                                       //<Statement> ::= lety Id '='
<Expression>
                                       Result = Create-
Rule_Statement_Lety_Id_Eq(TheReduction.Tokens);
                                       break;
                                case
(int)RuleConstants.RULE_STATEMENT_LETZ_ID_EQ:
                                       //<Statement> ::= letz Id '='
<Expression>
```

```
                                    Result = Create-
Rule_Statement_Letz_Id_Eq(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_INTEGER_ID_EQ:
                                    //<Statement> ::= integer Id '='
<Expression>
                                    Result = Create-
Rule_Statement_Integer_Id_Eq(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_STRING_ID_EQ:
                                    //<Statement> ::= string Id '='
<Expression>
                                    Result = Create-
Rule_Statement_String_Id_Eq(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_GRASP:
                                    //<Statement> ::= grasp
                                    Result = Create-
Rule_Statement_Grasp(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_GOTOHOME:
                                    // <Statement> ::= gotohome
                                    Result = Create-
Rule_Statement_Gotohome(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_POSE_ID:
                                    // <Statement> ::= pose Id
                                    Result = Create-
Rule_Statement_Pose_Id(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_HOME_ID:
                                    // <Statement> ::= home Id
                                    Result = Create-
Rule_Statement_Home_Id(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_RELEASE:
                                    //<Statement> ::= release
                                    Result = Create-
Rule_Statement_Release(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_DRIVE_LPARAN_COMMA_RPARAN:
                                    // <Statement> ::= drive '('
<Expression> ',' <Expression> ')'
                                    Result = Create-
Rule_Statement_Drive(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_STATEMENT_DRIVE_LPARAN_COMMA_RPARAN_COMMA_LPA
RAN_COMMA_RPARAN:
```

```
                    // <Statement> ::= drive '(' <Expression> ','
<Expression> ')' ',' '(' <Expression> ',' <Expression> ')'
                                        Result = Create-
Rule_Statement_Drive_Long(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_MOVE_COMMA_TIMES_COMMA_LPARAN_COMMA
_COMMA_RPARAN:
                    // <Statement> ::= move <Interpolation> ',' '*'
',' '(' <Expression> ',' <Expression> ',' <Expression> ')'
                                        Result = Create-
Rule_Statement_Move_Long(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_MOVE_COMMA_ID_COMMA_ID:
                    // <Statement> ::= move <Interpolation> ',' Id
',' Id
                                        Result = Create-
Rule_Statement_Move(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_POSE_ID_LPARAN_COMMA_COMMA_RPARAN:
                    // <Statement> ::= pose Id '(' <Expression> ','
<Expression> ',' <Expression> ')'
                                        Result = Create-
Rule_Statement_Pose_Long(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_ROTATE_COMMA_COMMA:
                                        // <Statement> ::= rotate
<Plane> ',' <Expression> ',' <Expression>
                                        Result = Create-
Rule_Statement_Rotate(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_OUT_COMMA:
                                        // <Statement> ::= out <Expres-
sion> ',' <Expression>
                                        Result = Create-
Rule_Statement_Out(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_IN_ID_EQ:
                                        // <Statement> ::= in Id '='
<Expression>
                                        Result = Create-
Rule_Statement_In(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_WHILE_DO_END:
                                        //<Statement> ::= while <Expres-
sion> do <Statements> end
                                        Result = Create-
Rule_Statement_While_Do_End(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_STATEMENT_IF_THEN_END:
```

```
                                        //<Statement> ::= if <Expres-
sion> then <Statements> end
                                        Result = Create-
Rule_Statement_If_Then_End(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_STATEMENT_IF_THEN_ELSE_END :
                                        //<Statement> ::= if <Expres-
sion> then <Statements> else <Statements> end
                                        Result = Create-
Rule_Statement_If_Then_Else_End(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_GT:
                                        //<Expression> ::= <Expression>
'>' <Add Exp>
                                        Result = Create-
Rule_Expression_Gt(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_LT:
                                        //<Expression> ::= <Expression>
'<' <Add Exp>
                                        Result = Create-
Rule_Expression_Lt(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_LTEQ:
                                        //<Expression> ::= <Expression>
'<=' <Add Exp>
                                        Result = Create-
Rule_Expression_Lteq(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_GTEQ:
                                        //<Expression> ::= <Expression>
'>=' <Add Exp>
                                        Result = Create-
Rule_Expression_Gteq(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_EQEQ:
                                        //<Expression> ::= <Expression>
'==' <Add Exp>
                                        Result = Create-
Rule_Expression_Eqeq(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION_LTGT:
                                        //<Expression> ::= <Expression>
'<>' <Add Exp>
                                        Result = Create-
Rule_Expression_Ltgt(TheReduction.Tokens);
                                        break;
                                    case
(int)RuleConstants.RULE_EXPRESSION:
                                        //<Expression> ::= <Add Exp>
```

```
                                    Result = Create-
Rule_Expression(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_ADDEXP_PLUS:
                                    //<Add Exp> ::= <Add Exp> '+'
<Mult Exp>
                                    Result = Create-
Rule_Addexp_Plus(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_ADDEXP_MINUS:
                                    //<Add Exp> ::= <Add Exp> '-'
<Mult Exp>
                                    Result = Create-
Rule_Addexp_Minus(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_ADDEXP_AMP:
                                    //<Add Exp> ::= <Add Exp> '&'
<Mult Exp>
                                    Result = Create-
Rule_Addexp_Amp(TheReduction.Tokens);
                                    break;
                                case (int)RuleConstants.RULE_ADDEXP:
                                    //<Add Exp> ::= <Mult Exp>
                                    Result = Create-
Rule_Addexp(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_MULTEXP_TIMES:
                                    //<Mult Exp> ::= <Mult Exp> '*'
<Negate Exp>
                                    Result = Create-
Rule_Multexp_Times(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_MULTEXP_DIV:
                                    //<Mult Exp> ::= <Mult Exp> '/'
<Negate Exp>
                                    Result = Create-
Rule_Multexp_Div(TheReduction.Tokens);
                                    break;
                                case (int)RuleConstants.RULE_MULTEXP:
                                    //<Mult Exp> ::= <Negate Exp>
                                    Result = Create-
Rule_Multexp(TheReduction.Tokens);
                                    break;
                                case
(int)RuleConstants.RULE_NEGATEEXP_MINUS:
                                    //<Negate Exp> ::= '-' <Value>
                                    Result = Create-
Rule_Negateexp_Minus(TheReduction.Tokens);
                                    break;
                                case (int)RuleConstants.RULE_NEGATEEXP:
                                    //<Negate Exp> ::= <Value>
                                    Result = Create-
Rule_Negateexp(TheReduction.Tokens);
                                     break;
                                 case (int)RuleConstants.RULE_VALUE_ID:
                                     //<Value> ::= Id
```

```
                                        Result = Create-
Rule_Value_Id(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_VALUE_STRINGLITERAL:
                                        //<Value> ::= StringLiteral
                                        Result = Create-
Rule_Value_Stringliteral(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_VALUE_NUMBERLITERAL:
                                        //<Value> ::= NumberLiteral
                                        Result = Create-
Rule_Value_Numberliteral(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_PLANE_PLANEIDENTIFIER:
                                        // <Plane> ::= PlaneIdentifier
                                        Result = Create-
Rule_Plane_Identifier(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_INTERPOLATION_INTERPOLATIONIDENTIFIER:
                    // <Interpolation> ::= InterpolationIdentifier
                                        Result = Create-
Rule_Interpolation_Identifier(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_VALUE_DIGITLITERAL:
                                        //<Value> ::= DigitalLiteral
                                        Result = Create-
Rule_Value_Digitliteral(TheReduction.Tokens);
                                        break;
                                case
(int)RuleConstants.RULE_VALUE_LPARAN_RPARAN:
                                        //<Value> ::= '(' <Expression>
')'
                                        Result = Create-
Rule_Value_Lparan_Rparan(TheReduction.Tokens);
                                        break;
                                default:
                                        Result = TheReduction;
                                        break;
                                        #endregion
                        }
                        return Result;
                }
```

The following code shows that how the TParser creates statement for each reduction. The CreateRule_Statement_Move method is called by CreateNew-Object to create SLIM move statement.

```
protected override Reduction CreateRule_Statement_Move(ArrayList To-
kens)
                {
                        TokenOne = (Token)Tokens[1];
                        TokenTwo = (Token)Tokens[3];
                        TokenThree = (Token)Tokens[5];
                        return CreateSLIMMoveStm(TokenOne.Data, To-
kenTwo.Data.ToString(), TokenThree.Data.ToString());
                }
```

The following code shows how finally SLIM statement is created and return back to SLIMCompiler to take action on this statement. CreateSLIMMoveStm is called by CreateRule_Staement_Move with three Token as parameters.

```
public SLIMMoveStm CreateSLIMMoveStm(object ValueOne, string ValueT-
wo, string ValueThree)
                {
                        // Move L, PoseOne, PoseTwo
                        //        |    |          |
                        //       |    |          |____ ValueThree
                        //       |    |_______________ ValueTwo
                        //        |____________________ ValueOne // In-
terpolation

                        SLIMMoveStm returnValue;
                        SLIMMoveStm Obj = new SLIMMoveStm();

                        Obj.ValueOne = ValueOne;
                        Obj.ValueTwo = ValueTwo;
                        Obj.ValueThree = ValueThree;
                        Obj.MessageArrived +=new MessageHand-
                        ler(Obj_MessageArrived);
                        returnValue = Obj;
                        return returnValue;
                }
```

The CreateSLIMMoveStm create the object of SLIMMoveStm class, shown in the following code. SLIMMoveStm class shall also implement Execute method.

```
public class SLIMMoveStm : SLIMReduction, IContextMethod
                {

                        public event MessageHandler MessageArrived;///
generate the event of message
                        public object ValueOne;
                        public string ValueTwo;
                        public string ValueThree;

                        public object Execute()
                        {
                                StringBuilder sb = new StringBuilder();
                                sb.Append("Move ");

        sb.Append(Convert.ToString(((IContextValue)(ValueOne)).Value)
);
                                sb.Append(", ");
                                sb.Append(PositionArray.Getx(ValueTwo)
+ "," + PositionArray.Gety(ValueTwo) + "," + PositionAr-
ray.Getz(ValueTwo) + "," + PositionArray.Getx(ValueThree) + "," + Po-
sitionArray.Gety(ValueThree) + "," + PositionArray.Getz(ValueThree));
                                PositionAr-
ray.Setx("Current",Convert.ToInt16(PositionArray.Getx(ValueThree)));
                                PositionAr-
ray.Sety("Current",Convert.ToInt16(PositionArray.Gety(ValueThree)));
                                PositionAr-
ray.Setz("Current",Convert.ToInt16(PositionArray.Getz(ValueThree)));
                                MessageArrived(sb.ToString());
                                return null;

                        }
                }
```

The following code shows the Execute method which calls Execute method of current token.

```
public virtual object Execute()
                {
                        if (CurrentReduction != null)
                        {
                                ((IContextMe-
thod)(CurrentReduction)).Execute();
                        }
                        return null;
                }
```

An example of the implementation of SLIMCompiler as an augmented reality is listed in the following pages. A revolute robot is taken as the example. Figures 6.6, 6.7, 6.8, and 6.9 detail use case diagram, UML static diagram, UML sequence diagram and SLIM robot operation flow chart. Figures 6.10 and 6.11 present the SLIM revolute robot and the SLIM Revolute robot performing welding maneuver.

Robot Project is Summarized Below:

The SLIMCompiler namespace for robot project contains six classes. The purpose of SLIMParser class is to parse the tokens of the program file and create grammar for the parsed statements. It inherits from TParser class which contains the basic functions for syntax parsing. The Position class is used to hold the position of the selected robot object. The PositionArray class holds the list of positions values for defining a robot path. GenerateCode class is to generate code based on the provided user input statements after validation and parsing. The Animation class is used to render the visual objects in 3D for displaying robotic arm and its movements. A summary of this project is provided in Tables 6.12, 6.13, 6.14, 6.15, 6.16, 6.17, 6.18, 6.19, 6.20, 6.21, 6.22, and 6.23.

The C# code that performs ‘Rotate’ operation at the revolute robot is listed below:

Following partial code is taken from SLIMRobot’s Robot class which is used to display robot in 3D format. The following Rotate method control the rotation and movement of robot’s arm different parts. The parameters of Rotate method pass the required rotation and required rotation plane information. Rotate method first checks what is the CenterOfRotation. CenterOfRotation can have a value from four different values; this value defines what part of robot has to rotate. After it finds out the CenterOfRotation, it takes the current rotation by using GetRotation method of that value and increases or decreases the rotation by using SetRotation method until it reaches the required rotation define in Angle parameter of Rotate method.

SLIM COMPIER: ROTATE METHOD

```
public void Rotate(PlaneSpecifier PSpecifier, int Angle, int CenterO-
fRotation)
               {
                       bool CheckValue = true;
                       float YAxis;
                       if(CenterOfRotation == 1)
                       {
                               //Stand rotation movement
                               StandPosition =  Mesh-
Stand.GetRotation();
                               YAxis = StandPosition.y;
                               if(Angle > (int)YAxis)

                               {
                                       while(CheckValue)
                                       {
                                               Mesh-
Stand.SetRotation(StandPosition.x,StandPosition.y + 0.5f
,StandPosition.z );
                                               StandPosition = Mesh-
Stand.GetRotation();
                                               Application.DoEvents();
                                               SpeedControl();
                                               Render();

                                               if(StandPosition.y >=
Angle)
                                                       CheckValue =
false;
                                       }
                               }
                               else
                               {
                                       while(CheckValue)
                                       {
                                               Mesh-
Stand.SetRotation(StandPosition.x,StandPosition.y - 0.5f
,StandPosition.z );
                                               StandPosition = Mesh-
Stand.GetRotation();
                                               Application.DoEvents();
                                               SpeedControl();
                                               Render();

                                               if(StandPosition.y <=
Angle)
                                                       CheckValue =
false;
                                       }

                               }

                       }
                       if(CenterOfRotation == 2)
                       {
                               //Backarm rotation movement
                                       BackArmPosition =  MeshBack-
       Arm.GetRotation();
                                       YAxis = BackArmPosition.x;
```

```
                                if(Angle > (int)YAxis)

                                {
                                        while(CheckValue)
                                        {
                                                MeshBack-
Arm.SetRotation(BackArmPosition.x + 0.5f,BackArmPosition.y
,BackArmPosition.z );
                                                BackArmPosition = Mesh-
BackArm.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(BackArmPosition.x >=
Angle)
                                                        CheckValue =
false;
                                        }
                                }
                                else
                                {
                                        while(CheckValue)
                                        {
                                                MeshBack-
Arm.SetRotation(BackArmPosition.x - 0.5f,BackArmPosition.y
,BackArmPosition.z );
                                                BackArmPosition = Mesh-
BackArm.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(BackArmPosition.x <=
Angle)
                                                        CheckValue =
false;
                                        }

                                }
                        }
                        if(CenterOfRotation == 3)
                        {

                                //Forearm rotation movement
                                ForeArmPosition =  MeshFo-
reArm.GetRotation();
                                YAxis = ForeArmPosition.x;
                                if(Angle > (int)YAxis)

                                {
                                        while(CheckValue)
                                        {
                                                MeshFo-
reArm.SetRotation(ForeArmPosition.x + 0.5f,ForeArmPosition.y
,ForeArmPosition.z );
```

```
                                                ForeArmPosition = Mesh-
ForeArm.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(ForeArmPosition.x >=
Angle)
                                                        CheckValue =
false;
                                        }
                                }
                                else
                                {
                                        while(CheckValue)
                                        {
                                                MeshFo-
reArm.SetRotation(ForeArmPosition.x - 0.5f,ForeArmPosition.y
,ForeArmPosition.z );
                                                ForeArmPosition = Mesh-
ForeArm.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(ForeArmPosition.x <=
Angle)
                                                        CheckValue =
false;
                                        }

                                }
                        }
                        if(CenterOfRotation == 4)
                        {
                                //WristJoint rotation movement
                                WristJointPosition =  MeshWrist-
Joint.GetRotation();
                                YAxis = WristJointPosition.x;
                                if(Angle > (int)YAxis)

                                {
                                        while(CheckValue)
                                        {
                                                MeshWrist-
Joint.SetRotation(WristJointPosition.x + 0.5f,WristJointPosition.y
,WristJointPosition.z );
                                                WristJointPosition =
MeshWristJoint.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(WristJointPosition.x
>= Angle)
                                                        CheckValue =
false;
                                        }
                                }
                                else
                                {
                                        while(CheckValue)
                                        {
```

```
                                                MeshWrist-
Joint.SetRotation(WristJointPosition.x - 0.5f,WristJointPosition.y
,WristJointPosition.z );
                                                WristJointPosition =
MeshWristJoint.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(WristJointPosition.x
<= Angle)
                                                        CheckValue =
false;
                                        }

                                }
                        }
                        if(CenterOfRotation == 5)
                        {
                                //Wrist rotation movement
                                WristPosition =  Mesh-
Wrist.GetRotation();
                                YAxis = WristPosition.y;
                                if(Angle > (int)YAxis)

                                {
                                        while(CheckValue)
                                        {
                                                Mesh-
Wrist.SetRotation(WristPosition.x ,WristPosition.y + 0.5f
,WristPosition.z );
                                                WristPosition = Mesh-
Wrist.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(WristPosition.y >=
Angle)
                                                        CheckValue =
false;
                                        }
                                }
                                else
                                {
                                        while(CheckValue)
                                        {
                                                Mesh-
Wrist.SetRotation(WristPosition.x ,WristPosition.y - 0.5f
,WristPosition.z );
                                                WristPosition = Mesh-
Wrist.GetRotation();
                                                Application.DoEvents();
                                                SpeedControl();
                                                Render();

                                                if(WristPosition.y <=
Angle)
                                                        CheckValue =
false;
                                        }

                                }
                        }
```

Table 6.1 Summary of SLIMParser class

Class summary			
Class ID: CP1			Namespace: SLIMCompiler
Class name: SLIMParser			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: TParser			Interfaces with: None
Field name	Data type	Access modifier	Description
TokenOne	Token	Private	Variable to hold Token one type
TokenTwo	Token	Private	Variable to hold Token two type
TokenThree	Token	Private	Variable to hold Token three type
TokenFour	Token	Private	Variable to hold Token four type

Method name	Return type	Access modifier	Description
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	static void	Public	Function to write error details
Execute()	virtual object	Public	Function to execute the program
CreateSLIMIfStm(object IfClause, object ThenClause, object ElseClause)	SLIMIfStm	Private	Function to create If statement
CreateSLIMWhileStm(object WhileClause, object DoClause)	SLIMWhileStm	Private	Function to create While statement
CreateSLIMStmList(object CurrentStm, object NextStm)	SLIMStmList	Private	Function to create Statement List
CreateSLIMID(string Name)	SLIMID	Private	Function to create identifier
CreateSLIMExpression(object LeftOperand, string Operator, object RightOperand)	SLIMExpression	Private	Function to create Expression
CreateSLIMString(string Text)	SLIMString	Private	Function to create string
CreateSLIMGraspStm(object Grasp)	SLIMGraspStm	Public	Function to create Grasp statement

(continued)

Table 6.1 (continued)

Method name	Return type	Access modifier	Description
CreateSLIMGoToHomeStm(object GoToHome)	SLIMGoToHomeStm	Public	Function to create Go to Home statement
CreateSLIMReleaseStm(object Release)	SLIMReleaseStm	Public	Function to create Release statement
CreateSLIMHomeStm(string PoseName)	SLIMHomeStm	Public	Function to create Home Statement
CreateSLIMPoseStm(string PoseName)	SLIMPoseStm	Public	Function to create Position Statement
CreateSLIMSpeedStm(object SpeedClause)	SLIMSpeedStm	Public	Function to create Speed Statement
CreateSLIMDelayStm(object DelayClause)	SLIMDelayStm	Public	Function to create Delay statement
CreateSLIMDriveStm(object ValueOne,object ValueTwo)	SLIMDriveStm	Public	Function to create Drive Statement
CreateSLIMInStm(string IOVariable, object IOAddress)	SLIMInStm	Public	Function to create Input output statement
CreateSLIMPoseLongStm(string Name, object ValueOne, object ValueTwo, object ValueThree)	SLIMPoseLongStm	Public	Function to create long position statement
CreateSLIMMoveLongStm(object ValueOne, object ValueTwo, object ValueThree, object ValueFour)	SLIMMoveLongStm	Public	Function to create long move statement
CreateSLIMMoveStm(object ValueOne, string ValueTwo, string ValueThree)	SLIMMoveStm	Public	Function to create move statement
CreateSLIMDriveLongStm(object ValueOne, object ValueTwo, object ValueThree, object ValueFour)	SLIMDriveLongStm	Public	Function to create long drive statement

(continued)

Table 6.1 (continued)

Method name	Return type	Access modifier	Description
CreateSLIMRotateStm(object ValueOne, object ValueTwo, object ValueThree)	SLIMRotateStm	Public	Function to create rotate statement
CreateSLIMOutStm(object IOAddress, object IOData)	SLIMOutStm	Public	Function to create output statement
CreateSLIMLetxStm(string Name, object AssignValue)	SLIMLetxStm	private	Function to create X value statement
CreateSLIMLetyStm(string Name, object AssignValue)	SLIMLetyStm	private	Function to create Y value statement
CreateSLIMLetzStm(string Name, object AssignValue)	SLIMLetzStm	private	Function to create Z value statement
CreateSLIMIntegerStm(string Name, object AssignValue)	SLIMIntegerStm	private	Function to create integer statement
CreateSLIMStringStm(string Name, object AssignValue)	SLIMStringStm	private	Function to create string statement
CreateSLIMNumber(double Value)	SLIMNumber	private	Function to create number
CreateInterpolationIdentifier(object Value)	SLIMInterpolationSpecifier	private	Function to create interpolation
CreatePlaneIdentifier(object Value)	SLIMPlaneSpecifier	private	Function to create plane
CreateSLIMDigit(int Value)	SLIMDigit	private	Function to create digit
Obj_MessageArrived(string message)	void	private	Function to check the Message Arrive event

Table 6.1 (continued)

Event name	Delegate	Access modifier	Description
SLIMCommandArrived	SLIMCommandHandler	Public	Event to generate when SLIM command arrives
Inner class name	Inherited/Interfaces from	Access modifier	Description
SLIMReduction	Reduction	public	Class for creating reduction object
SLIMIfStm	SLIMReduction, IContextMethod	private	Class for creating If statement
SLIMWhileStm	SLIMReduction, IContextMethod	private	Class for creating While statement
SLIMStmList	SLIMReduction, IContextMethod	private	Class for creating Statement List
SLIMID	SLIMReduction,IContextValue	private	Class for creating identifier
SLIMExpression	SLIMReduction,IContextValue	private	Class for creating Expression
SLIMString	SLIMReduction,IContextValue	private	Class for creating string
SLIMGraspStm	SLIMReduction, IContextMethod	private	Class for creating Grasp statement
SLIMGoToHomeStm	SLIMReduction, IContextMethod	private	Class for creating Go to Home statement
SLIMReleaseStm	SLIMReduction, IContextMethod	private	Class for creating Release statement
SLIMHomeStm	SLIMReduction, IContextMethod	private	Class for creating Home Statement
SLIMPoseStm	SLIMReduction, IContextMethod	private	Class for creating Position Statement
SLIMSpeedStm	SLIMReduction, IContextMethod	private	Class for creating Speed Statement
SLIMDelayStm	SLIMReduction, IContextMethod	private	Class for creating Delay statement
SLIMDriveStm	SLIMReduction, IContextMethod	private	Class for creating Drive Statement
SLIMInStm	SLIMReduction, IContextMethod	private	Class for creating Input output statement
SLIMPoseLongStm	SLIMReduction, IContextMethod	private	Class for creating long position statement
SLIMMoveLongStm	SLIMReduction, IContextMethod	private	Class for creating long move statement
SLIMMoveStm	SLIMReduction, IContextMethod	private	Class for creating move statement
SLIMDriveLongStm	SLIMReduction, IContextMethod	private	Class for creating long drive statement
SLIMRotateStm	SLIMReduction, IContextMethod	private	Class for creating rotate statement
SLIMOutStm	SLIMReduction, IContextMethod	private	Class for creating output statement
SLIMLetxStm	SLIMReduction, IContextMethod	private	Class for creating X value statement
SLIMLetyStm	SLIMReduction, IContextMethod	private	Class for creating Y value statement

(continued)

Table 6.1 (continued)

Event name	Delegate	Access modifier	Description
SLIMLetzStm	SLIMReduction, IContextMethod	private	Class for creating Z value statement
SLIMIntegerStm	SLIMReduction, IContextMethod	private	Class for creating integer statement
SLIMStringStm	SLIMReduction, IContextMethod	private	Class for creating string statement
SLIMNumber	SLIMReduction,IContextValue	private	Class for creating number
SLIMInterpolationSpecifier	SLIMReduction,IContextValue	private	Class for creating interpolation
SLIMPlaneSpecifier	SLIMReduction,IContextValue	private	Class for creating plane
SLIMDigit	SLIMReduction,IContextValue	private	Class for creating digit

Table 6.2 Summary of TParser Class

Class summary			
Class ID: CP2			Namespace: SLIMCompiler
Class Name: TParser			Class Type: abstract
Is base class? Yes			No. of Inherited Classes: 1
Inherited From: Parser			Interfaces With: None
Field name	Data type	Access modifier	Description
SymbolConstants	Enum int	private	Enumeration to hold Symbol constants values
RuleConstants	Enum int	private	Enumeration to hold Rule Constants values
IContextMethod	interface	internal	Interface for implementing Context Method
IContextValue	interface	internal	Interface for implementing Context Value
Method Name	Return Type	Access Modifier	Description
DoParse(System.IO.TextReader Source,bool GenerateContext)	ParseMessage	public	Function to parse the source program
CreateNewObject(Reduction TheReduction)	Reduction	private	Function to create new object for reduction
CreateRule_Statements(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statements > :: = < Statement > < Statements>
CreateRule_Statements2(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statements > :: = < Statement>
CreateRule_Statement_Speed(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = speed < Expression>
CreateRule_Statement_Delay(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = delay < Expression>
Reduction	protected abstract	Function to	CreateRule_Statement_Pose_Id(ArrayList Tokens); handle < Statement > :: = pose Id

(continued)

Table 6.2 (continued)

Field name	Data type	Access modifier	Description
			CreateRule_Statement_Gotohome(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = gotohome
			CreateRule_Statement_Integer_Id_Eq(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = integer Id '='< Expression>
			CreateRule_Statement_Letx_Id_Eq(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = letx Id '=' < Expression>
			CreateRule_Statement_Lety_Id_Eq(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = lety Id '=' < Expression>
			CreateRule_Statement_Letz_Id_Eq(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = letz Id '=' < Expression>
			CreateRule_Statement_Move_Long(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = move < Interpolation > ',' '*' ',' '(' < Expression > ',' < Expression > ',' < Expression > ')'
CreateRule_Statement_Move(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = move < Interpolation > ',' Id ',' Id
			CreateRule_Statement_Pose_Long(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = pose Id '(' < Expression > ',' < Expression > ',' < Expression > ')'
			CreateRule_Statement_Home_Id(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = home Id
			CreateRule_Statement_String_Id_Eq(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = string Id '=' < Expression>
CreateRule_Statement_Grasp(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = grasp
CreateRule_Statement_Release(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = release

(continued)

Table 6.2 (continued)

Field name	Data type	Access modifier	Description
CreateRule_Statement_Drive(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = drive '(' < Expression > ',' < Expression > ')' CreateRule_Statement_Drive_Long(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = drive '(' < Expression > ',' < Expression > ')' ',' '(' < Expression > ',' < Expression > ')'
CreateRule_Statement_Rotate(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = rotate < Plane > ',' < Expression > ',' < Expression>
CreateRule_Statement_Out(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = out < Expression > ',' < Expression>
CreateRule_Statement_In(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = in Id '=' < Expression> CreateRule_Statement_While_Do_End(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = while < Expression > do < Statements > end CreateRule_Statement_If_Then_End(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = if < Expression > then < Statements > end CreateRule_Statement_If_Then_Else_End(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Statement > :: = if < Expression > then < Statements > else < Statements > end
CreateRule_Expression_Gt(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '>' < Add Exp>
CreateRule_Expression_Lt(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<' < Add Exp>
CreateRule_Expression_Lteq(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<= ' < Add Exp>

(continued)

Table 6.2 (continued)

Field name	Data type	Access modifier	Description
CreateRule_Expression_Gteq(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '> = ' < Add Exp>
CreateRule_Expression_Eqeq(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '== ' < Add Exp>
CreateRule_Expression_Ltgt(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<> ' < Add Exp>
CreateRule_Expression(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Add Exp>
CreateRule_Addexp_Plus(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '+' < Mult Exp>
CreateRule_Addexp_Minus(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '-' < Mult Exp>
CreateRule_Addexp_Amp(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '&' < Mult Exp>
CreateRule_Addexp(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Mult Exp>
CreateRule_Multexp_Times(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Mult Exp > '*' < Negate Exp>
CreateRule_Multexp_Div(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Mult Exp > '/' < Negate Exp>
CreateRule_Multexp(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Negate Exp>
CreateRule_Negateexp_Minus(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Negate Exp > :: = '-' < Value>
CreateRule_Negateexp(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Negate Exp > :: = < Value>7

(continued)

Table 6.2 (continued)

Field name	Data type	Access modifier	Description
CreateRule_Value_Id(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = Id
			CreateRule_Value_Stringliteral(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Value > :: = StringLiteral
			CreateRule_Value_Numberliteral(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Value > :: = NumberLiteral
CreateRule_Plane_Identifier(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Plane > :: = PlaneIdentifier
			CreateRule_Interpolation_Identifier(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Interpolation > :: = InterpolationIdentifier
CreateRule_Value_Digitliteral(ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = DigitLiteral
			CreateRule_Value_Lparan_Rparan(ArrayList Tokens);
Reduction	protected	abstract	Function to handle < Value > :: = '(' < Expression > ')'

Inner Class Name	Inherited/ Interfaces From	Access Modifier	Description
BaseContext	Reduction	public	Internal class to handle base context

Table 6.3 Summary of SLIMCompiler Position class

Class summary			
Class ID: CP3			Namespace: SLIMCompiler
Class name: Position			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
iXPosition	int	private	Holds the position for X axis
iYPosition	int	private	Holds the position for Y axis
iZPosition	int	private	Holds the position for Z axis
sName	string	Private	Holds the name of the SLIM command
bHomePosition	bool	Private	Holds the position for home
bCurrentPosition	bool	Private	Holds the position for X axis

Table 6.4 Summary of SLIMCompiler PositionArray class

Class summary			
Class ID: CP4			Namespace: SLIMCompiler
Class name: PositionArray			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
PoseArray	static ArrayList	Private	Array to hold the position
Method Name	Return Type	Access Modifier	Description
Clear()	static void	Public	Function to clear the position array
AddPosition(Position NewPosition)	static void	Public	Function to add the new position to array
Setx(string PositionName, int XValue)	static void	Public	Function to set the value of x axis to a position
Getx(string PositionName)	static string	Public	Function to get the value of x axis to a position
Sety(string PositionName, int YValue)	static void	Public	Function to set the value of y axis to a position
Gety(string PositionName)	static string	Public	Function to get the value of y axis to a position
Setz(string PositionName, int ZValue)	static void	Public	Function to set the value of z axis to a position
Getz(string PositionName)	static string	Public	Function to get the value of z axis to a position
SetHome(string PositionName)	static void	Public	Function to set the home position
GetHomePosition()	static Position	Public	Function to get the home position
SetCurrent(string PositionName)	static void	Public	Function to get the current position

Table 6.5 Summary of SLIMCompiler GenerateCode class

Class summary			
Class ID: CP5			Namespace: SLIMCompiler
Class name: GenerateCode			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces With: None
Field name	Data type	Access modifier	Description
SLIMCommands	static ArrayList	Private	Array to hold the SLIM commands
Method name	Return type	Access modifier	Description
AddCommand(string SLIMCommand)	void	Public	Function to add the SLIM command to the array list
GetCommands()	ArrayList	Public	Function to return the array of SLIM commands
ClearCommands()	void	Public	Function to clear the SLIM command from the array list

6.5 Augmented Reality Design for Gantry Crane

Gantry crane is used to position heavy objects including workpiece from one part of the working area to another. It is mounted close to the roof of the working area and can position itself in a Cartesian coordinate system. A pendent is used to control the movement of the Gantry.

Augmented Reality design for the Gantry Crane is similar to the augmented reality design of the three axes milling machine. However, the complexity of the operation at the Milling Machine is much higher than that at the Gantry Crane.

6.5.1 Interpreter Design for Gantry Crane

Owing to the simplicity of the operations of the Gantry Crane few commands are required to position it. An interpreter suffices this situation. Following command set is used to control the gantry.

Speed

Speed command controls the speed to gantry.

Example speed10

Goto (X,Y,Z)

Goto command controls the moving direction of gantry in three axes i.e. x axis, y axis and z axis.

Table 6.6 Summary of SLIMCompiler sequence diagram

Sequence diagram summary				
Sequence ID: SD-P1 Sequence name: SLIMCompiler Sequence Diagram				
Method name	Type	Description	Source	Destination
DoParse()	Message	Function to set for parsing SLIM Program	Form	SLIMParser
ParseMessage	Return Message	Return message after parsing	Form	SLIMParser
Execute()	Message	Function to execute SLIM Program	Form	SLIMParser
GetMessage	Return Message	Return message after execution	Form	SLIMParser
UpdateInterface()	Self Message	Function to update interface	Form	SLIMParser
Token.Execute()	Self Message	Function to execute a token	SLIMParser	Parser
OpenStream()	Message	Function to open file stream	SLIMParser	Parser
Parse()	Message	Function to parse and validate SLIM Program	SLIMParser	Parser
Response	Return Message	Return message after validation	SLIMParser	Parser
GetCurrentReduction()	Message	Function to get the token object	SLIMParser	Parser
Token()	Message	Function to generate the token	Parser	Token
Reset()	Self Message	Function to reset the robot position	Parser	Token
PrepareToParse()	Self Message	Function to prepare parsing	Parser	Token
PushToken()	Message	Function to push the token to stack	Parser	TokenStack
PullToken()	Message	Function to pull the token from stack	Parser	TokenStack
Token	Return Message	Return message containing token	TokenStack	SLIMParser

Table 6.7 Summary of SLIMCompiler open SLIM program use case

Use case summary
Use case ID: UC-P1
Use case name: Open
Actors: User
Description: The user selects and loads SLIM program through a file with "rbt" extension
Preconditions: SLIM program file is present on the system
Postconditions: SLIM program is loaded on the screen
System parameters: SLIM program file
Success scenario: SLIM program is loaded successfully
Alternative scenario: Error in loading part program
Includes: Open File
Priority: SetFilePath

Table 6.8 Summary of SLIMCompiler new SLIM program use case

Use case summary
Use case ID: UC-P2
Use case name: New
Actors: User
Description: The user inputs a new SLIM program on the screen
Preconditions: None
Postconditions: SLIM program is validated and saved
System parameters: SLIM program with SLIM standard
Success scenario: New SLIM program file is created successfully
Alternative scenario: Error in validating SLIM program or in saving file
Includes: None
Priority: High

Table 6.9 Summary of SLIMCompiler Parse SLIM program use case

Use case summary
Use case ID: UC-P3
Use case name: Parse
Actors: User
Description: The system parses and validates the SLIM program on the screen and saves it to a file with "rbt" extension
Preconditions: None
Postconditions: SLIM program is saved on the file
System parameters: SLIM program with SLIM standard
Success scenario: SLIM program is validated and saved successfully
Alternative scenario: Error in saving part program
Includes: DoParse, OpenStream
Priority: High

Table 6.10 Summary of SLIMCompiler Run SLIM program use case

Use case summary
Use case ID: UC-P4
Use case name: Run
Actors: User
Description: The user executes the SLIM program after loading it on the screen
Preconditions: SLIM program is loaded and validated
Postconditions: SLIM program executes and display the robot movement
System Parameters: SLIM program with SLIM standard
Success Scenario: SLIM program is executed successfully
Alternative Scenario: Error in executing part program
Includes: Parse, Execute, Reset
Priority: High

Table 6.11 Summary of SLIMCompiler processes flow charts

Flow chart summary	
Flow chart ID: FC-P1	
Flow chart name: SLIMCompiler Process Flow Chart	
No. of inputs: 6	
No. of processes: 32	
No. of outputs: 2	
No. of control decisions: 20	
Input name	Description
Save File	Save menu on screen
Execute File	Run menu on screen
Open File	Load menu on screen
New Menu	New menu on screen
Parse File	Reset menu on screen
Exit Application	Exit menu on screen
Process Name	Description
Run	Process to execute SLIM Program
Reset Robot	Process to reset the robot to initial position
Parser Execute	Process to execute parser
New part program	Process to New Part Program
Stop Rendering	Process to close rendering and exit
Clear Parse Log Window	Process to clear the movement log from screen
Do Parse	Process to set the parse flag
Get Token	Process to get the token statement
Pop Token	Process to pop token
Push Token	Process to push token
Parse Token	Process to parse token
Check Parse Result	Process to check the parse result
Add Entry in Log Window	Process to add entry in log window
Create Statement for every command	Process to create statements
Execute Statement	Process to execute statements
Generate Event for message	Process to generate event

(continued)

Table 6.11 (continued)

Input name	Description
Get Current Reduction	Process to get the reduction statement
Peek Token	Process to find token
Execute Token	Process to execute token after parsing
Parse Message Arrive	Process to parse message arrive
Split Message	Process to split message into tokens
Set Robot Speed	Process to set robot's speed
Robot Drive	Process to set robot's drive
Get Interpolation	Process to get interpolation
Robot Move Linear	Process to perform robot's linear movement
Robot Move Circular	Process to perform robot's circular movement
Get Plane Specified	Process to get specified plane
Rotate Robot on XY Plane	Process to rotate robot on XY plane
Rotate Robot on YZ Plane	Process to rotate robot on YZ plane
Rotate Robot on ZX Plane	Process to rotate robot on ZX plane
Robot Hand Close	Process to close robot's hand
Robot Hand Open	Process to open robot's hand
Output Name	Description
Robot's display	Robot loaded on screen
Robot's position display	Robot's movement position values on screen
Control Decision Name	Description
Open Dialog Box	Open Dialog box for choosing the file
Select Part Program	Select the SLIM program file
Save Dialog Box	Save Dialog Box
Input Token = 0	Check if Input Token = 0
Comments Level > 0	Check if Comments Level > 0
Token = NULL	Check if Token = NUL
Token = Whitespace	Check if Token = white space
Peek Token	Search for token
Check Parse Result for Syntax error	Check parse token for syntax
Reduction	Check if reduction is to applied
Message = Speed	Check if Message = Speed
Message = Drive	Check if Message = Drive
Message = Move	Check if Message = Move
Interpolation = L	Check if Interpolation = L
Message = Rotate	Check if Message = Rotate
Message = Grasp	Check if Message = Grasp
Message = Release	Check if Message = Release
Plane = XY	Check if Plane = XY
Plane = YZ	Check if Plane = YZ
Plane = XZ	Check if Plane = XZ

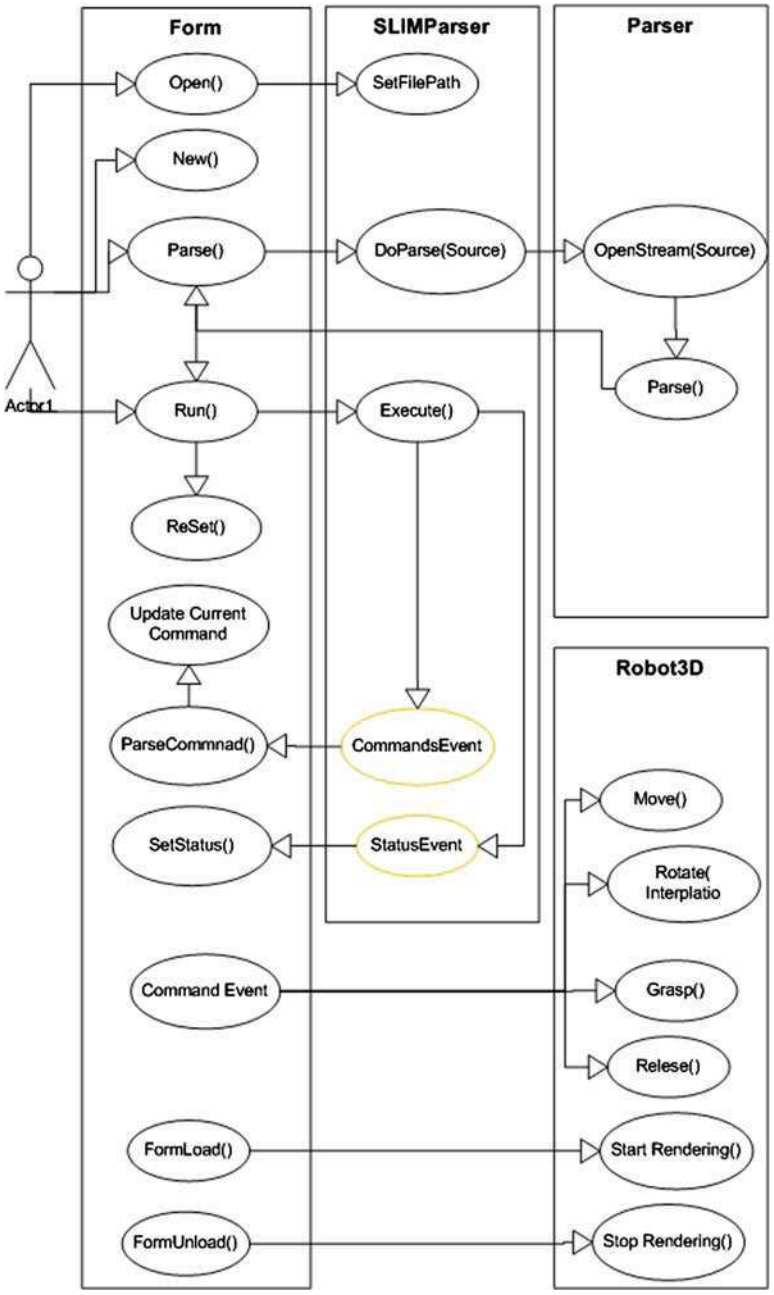


Fig. 6.6 Use Case Diagram of SLIM Robot

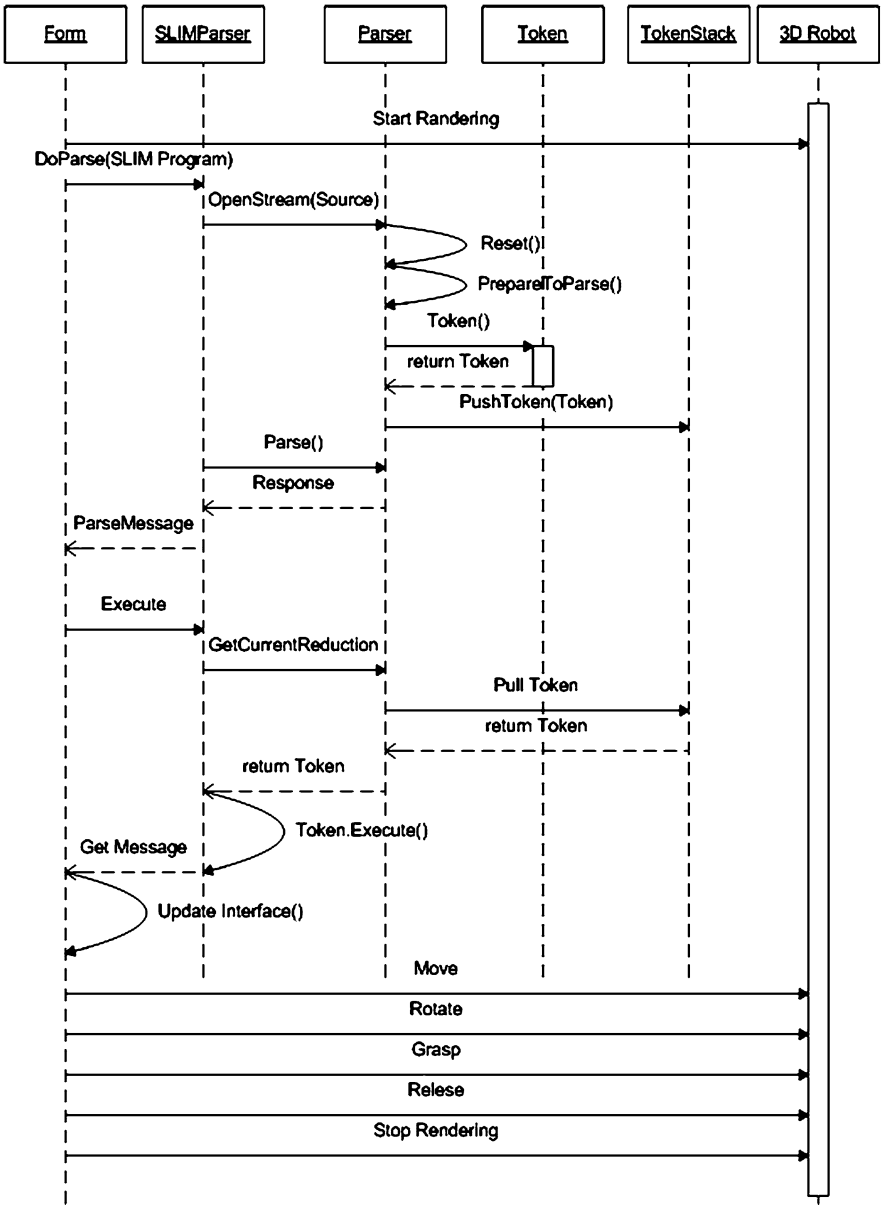


Fig. 6.7 SLIM Robot UML Sequence Diagram

Example goto (200,300, 30)

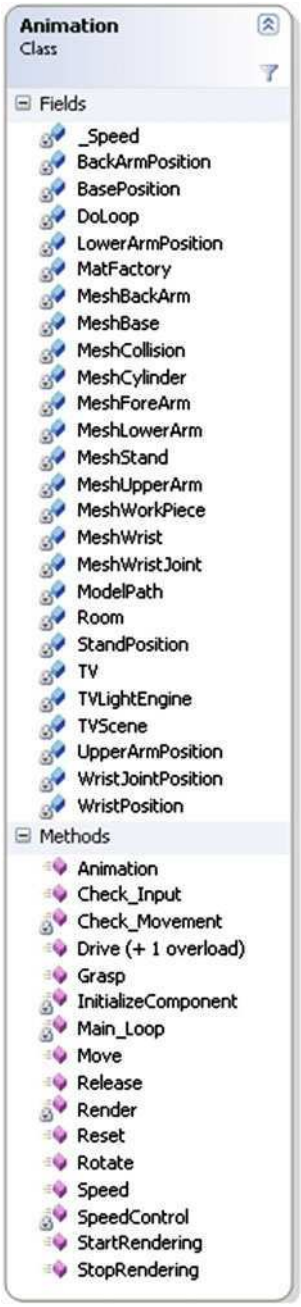


Fig. 6.8 UML Static Diagram of SLIM Robot

Gantry Crane is consists of three projects.

1. GantryEngine: Consists of two classes—PortAccess class, Gantry class, two enumerations, and four events. Object of Gantry class is used in gantry crane software to provide gantry functionality. Object of PortAccess is used in Gantry class to provide access to system physical port. Gantry Engine is a Dynamic Link Library (DLL)-based project. The produced DLL of Gantry Engine is used in gantry crane software to provide separation between application and logic, and logic encapsulation.

   Gantry class has encapsulated all the main functionality required by gantry crane software such as Open File, Save File, Edit File, Execute File, Validate File, and Validate Code. A request from gantry crane software for each function is handled by Gantry class. The enumerations of Gantry Engine are Current, Axis, Spindle, and GantryStatus. Current enumeration identifies the current value of different parameter and GantryStatus identifies the running status of gantry. Axis enumeration holds axis information when gantry is moving and Spindle enumeration holds current spindle status.
   To provide access to physical port on the system, Gantry Engine has PortAccess class. PortAccess class has encapsulated port access and interpolations functions. Only Gantry class has access to PortAccess class, and gantry crane software does not call Gantry class methods. Gantry class calls appropriate interpolation methods of PortAccess, which in turn calls private methods of PortAccess to send and receive physical port signal.
   In order to update the graphical user interface of gantry crane software about the current state of execution, Gantry Engine has three events. StatusArrived

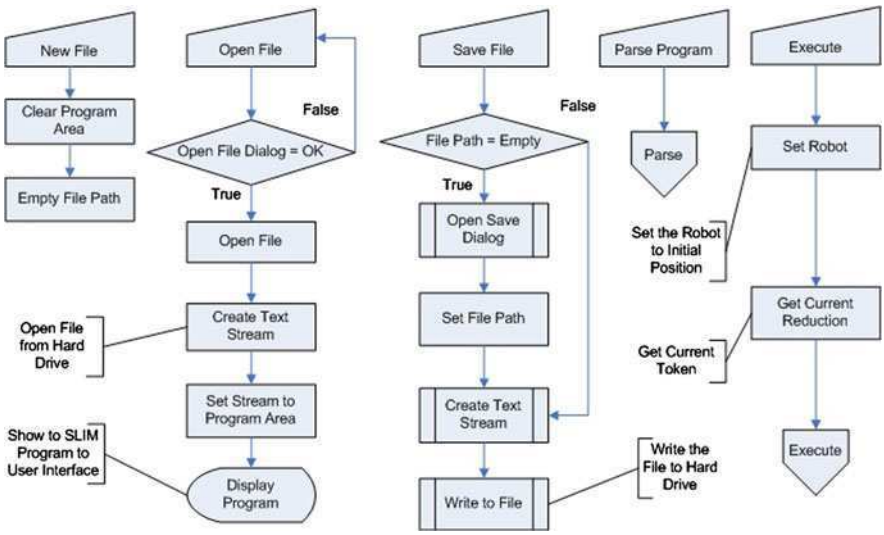


Fig. 6.9 SLIM Robot Flow Chart (continued)

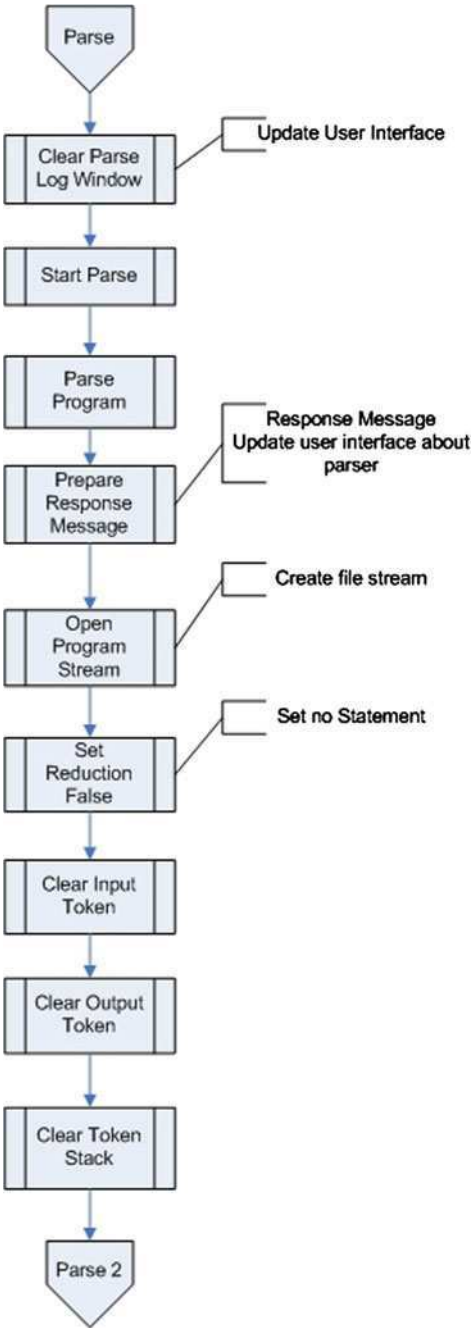


Fig. 6.9 (continued)

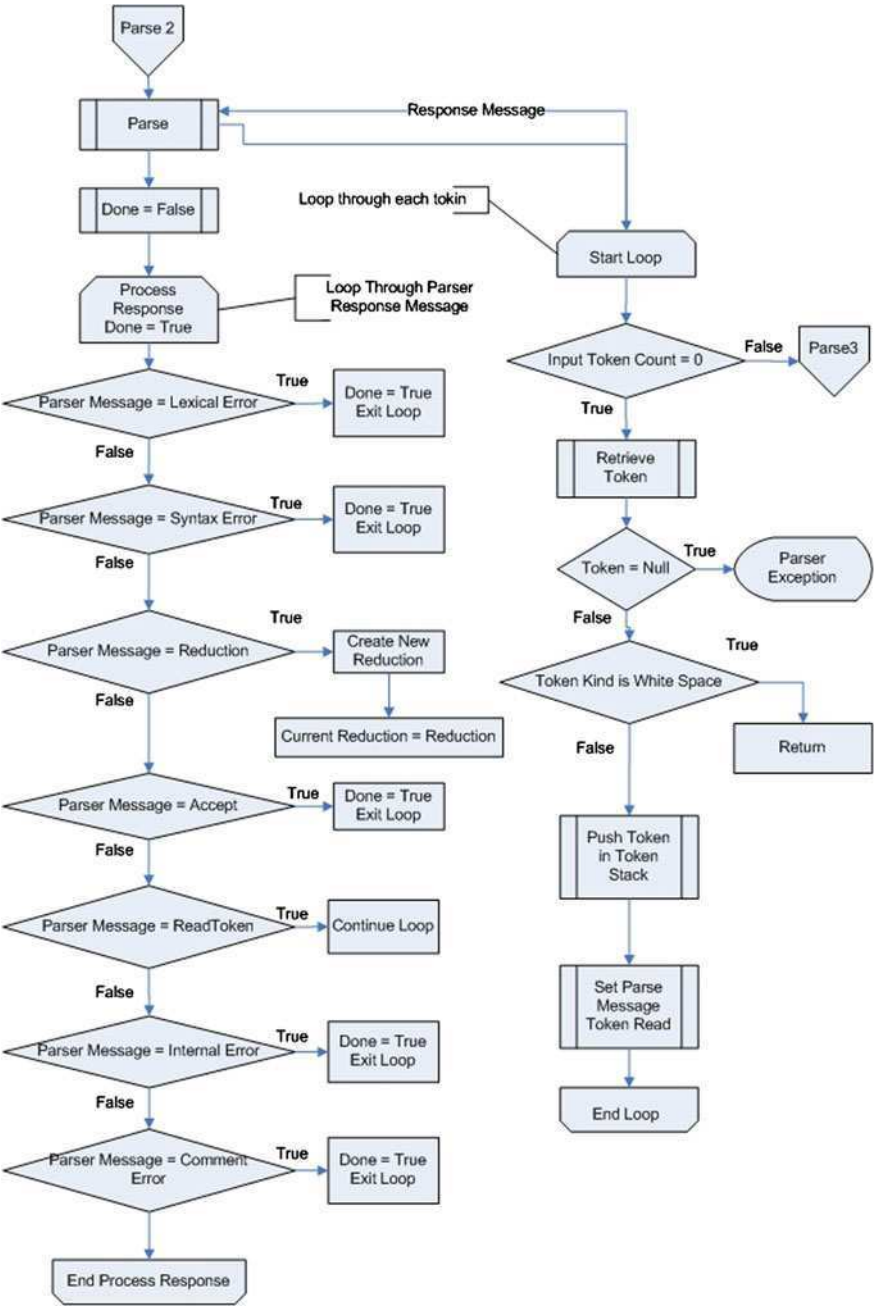


Fig. 6.9 (continued)

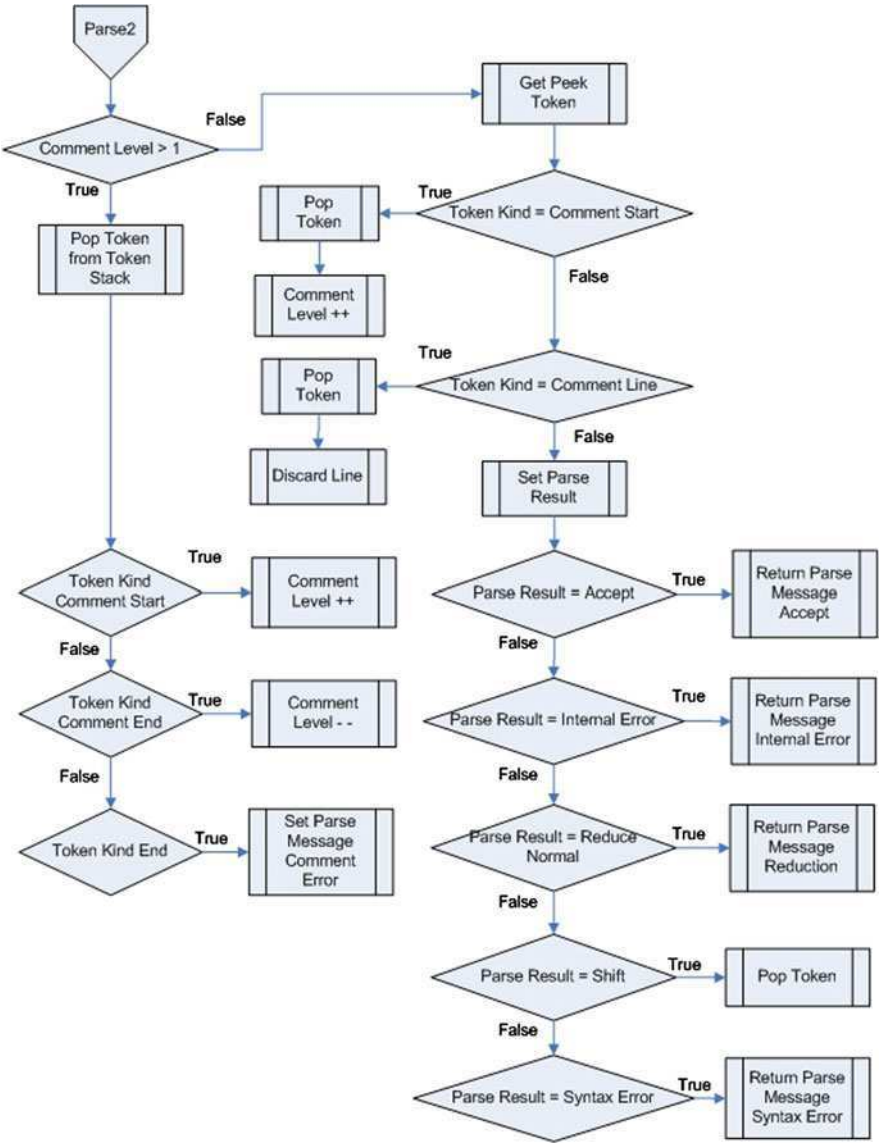


Fig. 6.9 (continued)

event raise by Gantry class to update current status of gantry, Speed event is raised when speed is set or changed by Gantry class, and WorkCompleted event is raised by Gantry class when work is completed. On information received from events by Gantry class, gantry crane passes this information to Gantry3D to update 3D graphics of gantry crane.

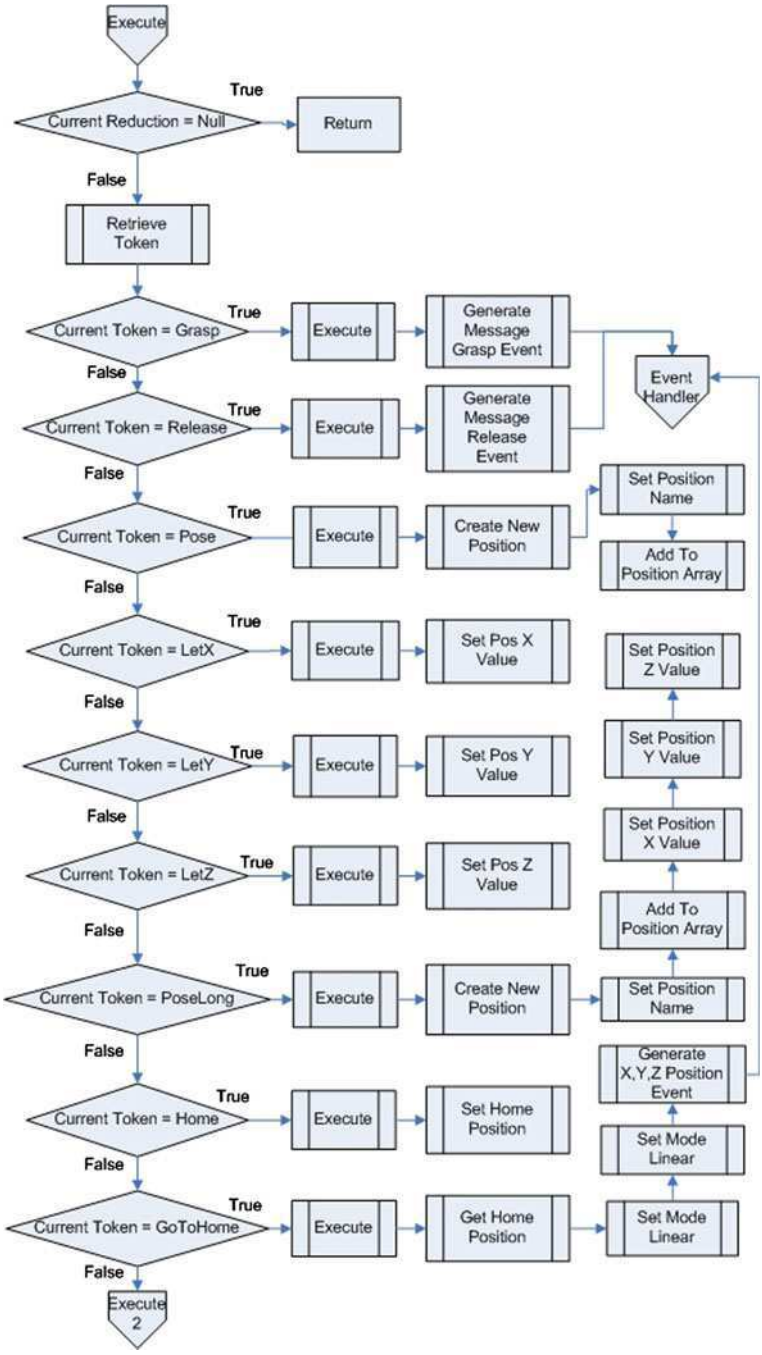


Fig. 6.9 (continued)

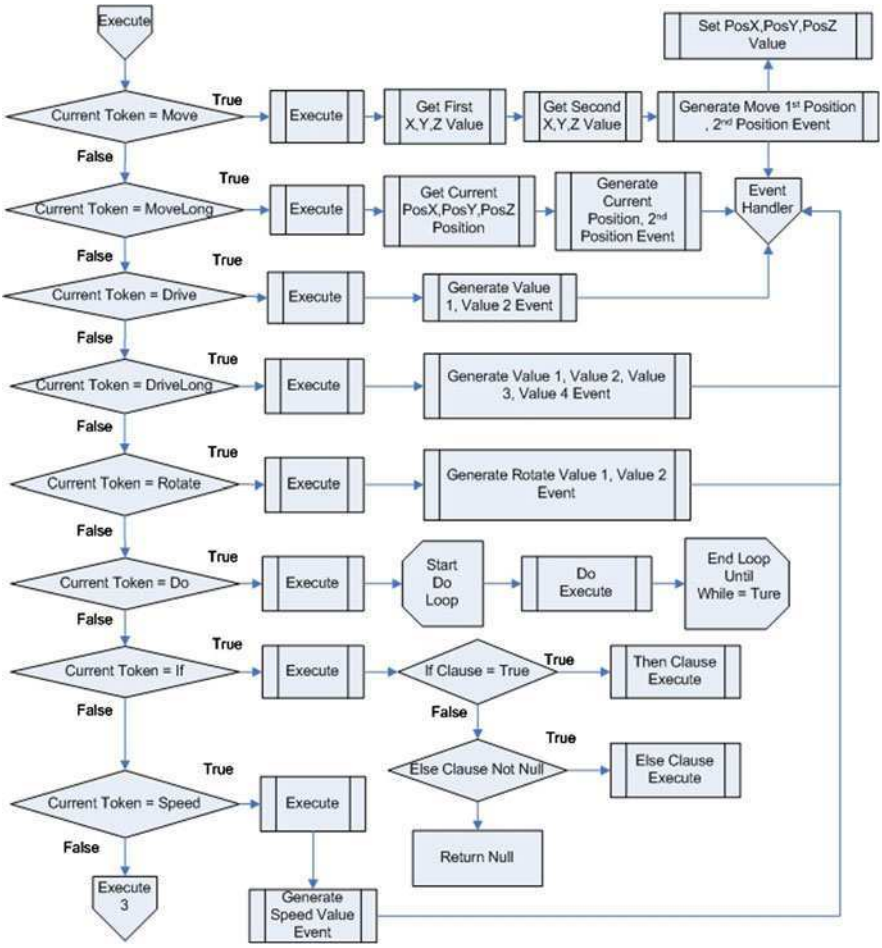


Fig. 6.9 (continued)

2. GantryMachine: Consist of a class 'Form1' which is inherited by System. Windows.Forms.Form to provide Gantry Machine graphical user interface that utilize Gantry Engine object to execute file and related functions. Current status of execution and controlling of 3D animation is also updated from GantryMachine.
3. CoreAnimation: Consist of Gantry3D class and two enumerations. Gantry3D class consists of 3D graphics rendering engine and methods to control movement of individual parts of Gantry. Enumerations of CoreAnimation are Spindle and Axis. Spindle enumeration controls the running status of spindle. Axis enumeration is for controlling the axis movement of the Gantry.

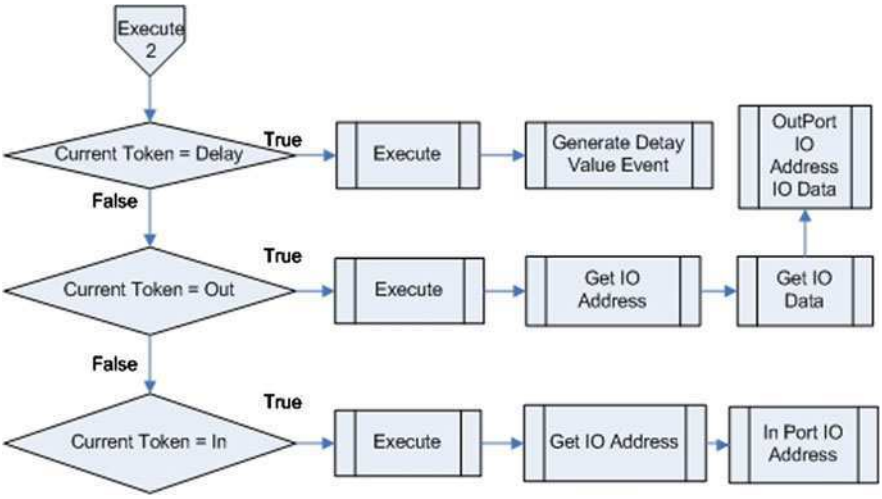


Fig. 6.9 (continued)

Figures 6.12, 6.13, 6.14, and 6.15 present the UML use case diagram, sequence diagram, static diagram and the flow chart for the Gantry crane. Figure 6.16 depicts the augmented reality of the Gantry Crane.

Table 6.24 shows the Gantry PortAccess class members properties, methods, and variables. PortAccess is the helper class of Gantry and this class is used to pass pulses to gantry machine by using parallel port and calculate the pulses, liner interpolation. PortAccess also holds the current position of X, Y, and Z axes and current state of the gantry machine.

Table 6.25 presents the Gantry class members variables, properties and methods. The Gantry class is used to read the part program file, validate it, open it, save it and executes it. This class is also responsible for raising events to update user interface and 3D gantry machine. Interpreter of CNC gantry is also implemented in this class' ExecuteFile method.

Table 6.26 describes the Gantry3D class member variables, properties and methods. Gantry3D class is used to render the graphics of gantry machine in 3D format for display movement of machine. 3D rendering engine is also implemented in this class.

Table 6.27 explains the summary of Gantry sequence diagram.

Table 6.28 depicts the summary of the Gantry Open Part Program use case.

Table 6.29 produces the summary of the Gantry New Part Program use case.

Table 6.30 outlines the summary of the Gantry Save Part Program use case.

Table 6.31 defines the summary of the Gantry Run Part Program use case.

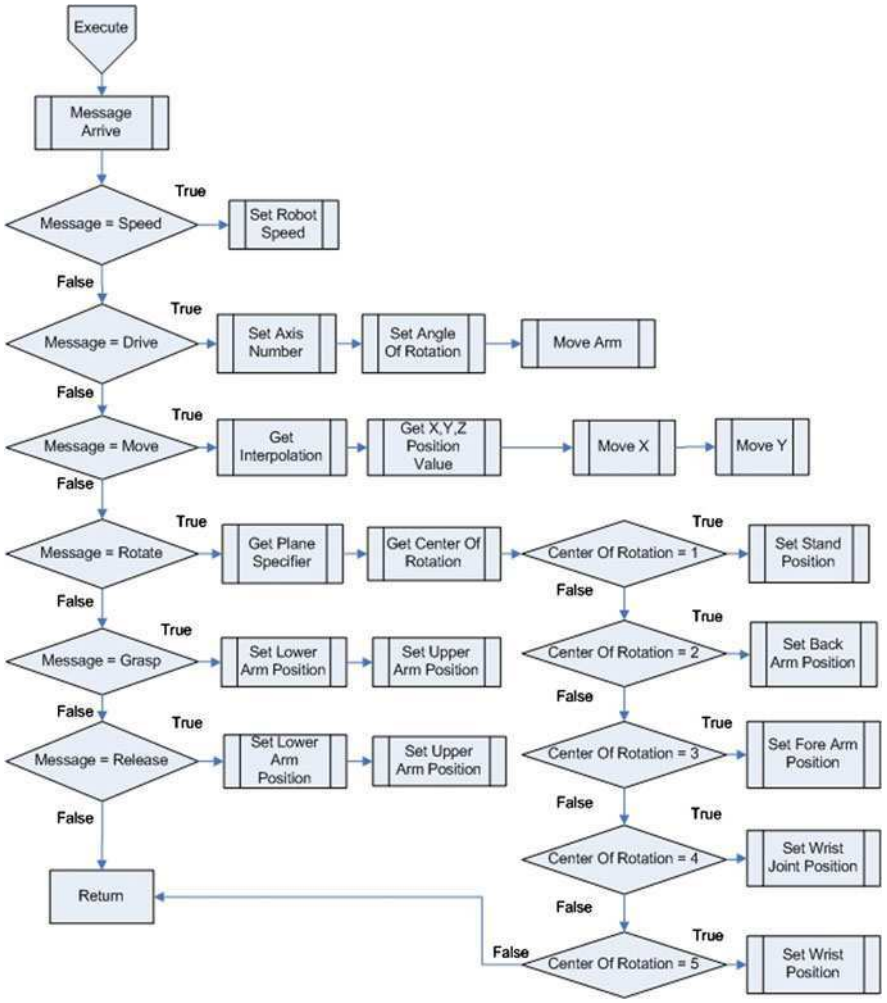


Fig. 6.9 (continued)

Table 6.32 shows the summary of the Gantry processes flow chart

Following Code in C# list the interpreter for the Gantry Crane. This C# code taken from Gantry class's ExecuteFile member method. ExecuteFile method is the interpreter of CNC Gantry crane. In this code it reads part program command one by one from an array and instructs PortAccess class to calculate required number of pulses by using Linear Interpolation and update user interface and 3D machine with the help of events. ValidateFile method is called before Execute method. ValidateFile method validates part program commands with ValideCode method and fills an array with validated commands. In case of any invalid command, ExecuteFile method is not called and error is return to user interface of ganty.

Gantry Interprefer: Execute File

```
public void ExecuteFile()
        {
            string Code;

            if (GantryStatusArrived != null)
                GantryStatusArrived(GantryStatus.On);
            for (int i = 0; i < aList.Count; i++)
            {

                Thread.Sleep(10);
                /// if Emergency stop then exit from loop
                if (PortAccess.EmergencyStop)
                    break;
                Code = aList[i].ToString();
                if (Code == "GoTo" || Code == "goto")
                {
                    // Goto Command
                    // Format is "GoTo (X,Y,Z) for Example GoTo
(35,40,80);
                    // X = X Direction Value
                    // Y = Y Direction Value
                    // Z = Z Direction Value

                    Code = aList[i + 1].ToString();
                    Code = Code.Substring(1);
                    string[] Directions =
Code.Split(",".ToCharArray());
                    StatusArrived(Current.X, Direc-
tions[0].ToString());
                    StatusArrived(Current.Y, Direc-
tions[1].ToString());
                    StatusArrived(Current.Z, Direc-
tions[2].TrimEnd(")".ToCharArray()));
                    i++;
                }

                else if (Code == "Speed" || Code == "SPEED")
                {
                    Code = aList[i + 1].ToString();
                    if (this.StatusArrived != null)
                        this.StatusArrived(Current.Speed, Code);
                    i++;

                }

            }

            if (GantryStatusArrived != null)
                GantryStatusArrived(GantryStatus.Off);

        }
```

Fig. 6.10 AR of SLIM Robot

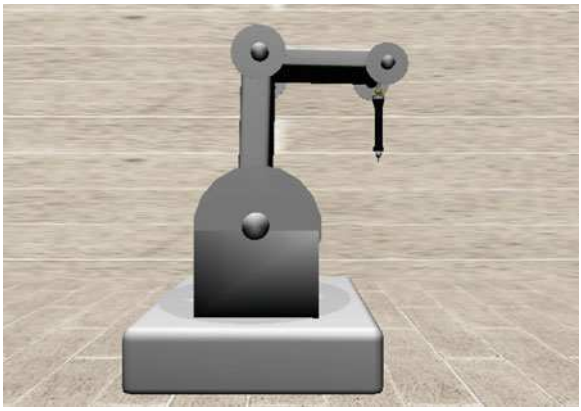


Fig. 6.11 AR of SLIM Welding

6.6 Augmented Reality Design for Conveyors

The conveyor systems are used for transportation of material from one place to another. Several type of conveyor system exists. This section refers to belt Conveyor. Development of augmented reality for belt conveyors requires graphic modeling of basic elements, i.e., supporting frame, pulleys, belt, and motor support. The command set for conveyor is as follows.

I. Speed

Speed command controls the speed of conveyor belt.
Example Speed 10

Table 6.12 Summary of SLIM Robot SLIMParser class

Class summary			
Class ID: CRI			Namespace: SLIMCompiler
Class name: SLIMParser			Class type: concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: TParser			Interfaces with: None
Field name	Data type	Access modifier	Description
TokenOne	Token	Private	Variable to hold Token one type
TokenTwo	Token	Private	Variable to hold Token two type
TokenThree	Token	Private	Variable to hold Token three type
TokenFour	Token	Private	Variable to hold Token four type

Method name	Return type	Access modifier	Description
WriteError(string sErrorMessage,string sErrorSource,string sErrorNumber,string sFunctionThatGenerateError)	static void	Public	Function to write error details
Execute()	virtual object	Public	Function to execute the program
CreateSLIMIfStm(object IfClause, object ThenClause, object ElseClause)	SLIMIfStm	Private	Function to create If statement
CreateSLIMWhileStm(object WhileClause, object DoClause)	SLIMWhileStm	Private	Function to create While statement
CreateSLIMStmList(object CurrentStm, object NextStm)	SLIMStmList	Private	Function to create Statement List
CreateSLIMID(string Name)	SLIMID	Private	Function to create identifier
CreateSLIMExpression(object LeftOperand, string Operator, object RightOperand)	SLIMExpression	Private	Function to create Expression

(continued)

Table 6.12 (continued)

Method name	Return type	Access modifier	Description
CreateSLIMString(string Text)	SLIMString	Private	Function to create string
CreateSLIMGraspStm(object Grasp)	SLIMGraspStm	Public	Function to create Grasp statement
CreateSLIMGoToHomeStm(object GoToHome)	SLIMGoToHomeStm	Public	Function to create Go to Home statement
CreateSLIMReleaseStm(object Release)	SLIMReleaseStm	Public	Function to create Release statement
CreateSLIMHomeStm(string PoseName)	SLIMHomeStm	Public	Function to create Home Statement
CreateSLIMPoseStm(string PoseName)	SLIMPoseStm	Public	Function to create Position Statement
CreateSLIMSpeedStm(object SpeedClause)	SLIMSpeedStm	Public	Function to create Speed Statement
CreateSLIMDelayStm(object DelayClause)	SLIMDelayStm	Public	Function to create Delay statement
CreateSLIMDriveStm(object ValueOne,object ValueTwo)	SLIMDriveStm	Public	Function to create Drive Statement
CreateSLIMInStm(string IOVariable, object IOAddress)	SLIMInStm	Public	Function to create Input output statement
CreateSLIMPoseLongStm(string Name, object ValueOne, object ValueTwo, object ValueThree)	SLIMPoseLongStm	Public	Function to create long position statement
CreateSLIMMoveLongStm(object ValueOne, object ValueTwo, object ValueThree, object ValueFour)	SLIMMoveLongStm	Public	Function to create long move statement
CreateSLIMMoveStm(object ValueOne, string ValueTwo, string ValueThree)	SLIMMoveStm	Public	Function to create move statement
CreateSLIMDriveLongStm(object ValueOne, object ValueTwo, object ValueThree, object ValueFour)	SLIMDriveLongStm	Public	Function to create long drive statement

(continued)

Table 6.12 (continued)

Method name	Return type	Access modifier	Description
CreateSLIMRotateStm(object ValueOne, object ValueTwo, object ValueThree)	SLIMRotateStm	Public	Function to create rotate statement
CreateSLIMOutStm(object IOAddress, object IOData)	SLIMOutStm	Public	Function to create output statement
CreateSLIMLetxStm(string Name, object AssignValue)	SLIMLetxStm	private	Function to create X value statement
CreateSLIMLetyStm(string Name, object AssignValue)	SLIMLetyStm	private	Function to create Y value statement
CreateSLIMLetzStm(string Name, object AssignValue)	SLIMLetzStm	private	Function to create Z value statement
CreateSLIMIntegerStm(string Name, object AssignValue)	SLIMIntegerStm	private	Function to create integer statement
CreateSLIMStringStm(string Name, object AssignValue)	SLIMStringStm	private	Function to create string statement
CreateSLIMNumber(double Value)	SLIMNumber	private	Function to create number
CreateInterpolationIdentifier(object Value)	SLIMInterpolationSpecifier	private	Function to create interpolation
CreatePlaneIdentifier(object Value)	SLIMPlaneSpecifier	private	Function to create plane
CreateSLIMDigit(int Value)	SLIMDigit	private	Function to create digit
Obj_MessageArrived(string message)	void	private	Function to check the Message Arrive event

Event name	Delegate	Access modifier	Description
SLIMCommandArrived	SLIMCommandHandler	Public	Event to generate when SLIM command arrives

(continued)

Table 6.12 (continued)

Inner Class Name	Inherited/Interfaces From	Access Modifier	Description
SLIMReduction	Reduction	public	Class for creating reduction object
SLIMIfStm	SLIMReduction, IContextMethod	private	Class for creating If statement
SLIMWhileStm	SLIMReduction, IContextMethod	private	Class for creating While statement
SLIMStmList	SLIMReduction, IContextMethod	private	Class for creating Statement List
SLIMID	SLIMReduction,IContextValue	private	Class for creating identifier
SLIMExpression	SLIMReduction,IContextValue	private	Class for creating Expression
SLIMString	SLIMReduction,IContextValue	private	Class for creating string
SLIMGraspStm	SLIMReduction, IContextMethod	private	Class for creating Grasp statement
SLIMGoToHomeStm	SLIMReduction, IContextMethod	private	Class for creating Go to Home statement
SLIMReleaseStm	SLIMReduction, IContextMethod	private	Class for creating Release statement
SLIMHomeStm	SLIMReduction, IContextMethod	private	Class for creating Home Statement
SLIMPoseStm	SLIMReduction, IContextMethod	private	Class for creating Position Statement
SLIMSpeedStm	SLIMReduction, IContextMethod	private	Class for creating Speed Statement
SLIMDelayStm	SLIMReduction, IContextMethod	private	Class for creating Delay statement
SLIMDriveStm	SLIMReduction, IContextMethod	private	Class for creating Drive Statement
SLIMInStm	SLIMReduction, IContextMethod	private	Class for creating Input output statement
SLIMPoseLongStm	SLIMReduction, IContextMethod	private	Class for creating long position statement
SLIMMoveLongStm	SLIMReduction, IContextMethod	private	Class for creating long move statement
SLIMMoveStm	SLIMReduction, IContextMethod	private	Class for creating move statement
SLIMDriveLongStm	SLIMReduction, IContextMethod	private	Class for creating long drive statement
SLIMRotateStm	SLIMReduction, IContextMethod	private	Class for creating rotate statement
SLIMOutStm	SLIMReduction, IContextMethod	private	Class for creating output statement
SLIMLetxStm	SLIMReduction, IContextMethod	private	Class for creating X value statement
SLIMLetyStm	SLIMReduction, IContextMethod	private	Class for creating Y value statement

(continued)

Table 6.12 (continued)

Inner Class Name	Inherited/Interfaces From	Access Modifier	Description
SLIMLetzStm	SLIMReduction, IContextMethod	private	Class for creating Z value statement
SLIMIntegerStm	SLIMReduction, IContextMethod	private	Class for creating integer statement
SLIMStringStm	SLIMReduction, IContextMethod	private	Class for creating string statement
SLIMNumber	SLIMReduction,IContextValue	private	Class for creating number
SLIMInterpolationSpecifier	SLIMReduction,IContextValue	private	Class for creating interpolation
SLIMPlaneSpecifier	SLIMReduction,IContextValue	private	Class for creating plane
SLIMDigit	SLIMReduction,IContextValue	private	Class for creating digit

Table 6.13 Summary of SLIM Robot TParser Class

Class summary			
Class ID: CR2			Namespace: SLIMCompiler
Class name: TParser			Class type: abstract
Is base class?: Yes			No. of inherited classes: 1
Inherited from: Parser			Interfaces with: None

Field name	Data type	Access modifier	Description
SymbolConstants	Enum int	private	Enumeration to hold Symbol constants values
RuleConstants	Enum int	private	Enumeration to hold Rule Constants values
IContextMethod	interface	internal	Interface for implementing Context Method
IContextValue	interface	internal	Interface for implementing Context Value

Method Name	Return Type	Access Modifier	Description
DoParse(System.IO.TextReader Source,bool GenerateContext)	ParseMessage	public	Function to parse the source program
CreateNewObject(Reduction TheReduction)	Reduction	private	Function to create new object for reduction
CreateRule_Statements (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statements > :: = < Statement > < Statements>
CreateRule_Statements2 (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statements > :: = < Statement>
CreateRule_Statement_Speed (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = speed < Expression>
CreateRule_Statement_Delay (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = delay < Expression>
CreateRule_Statement_Pose_Id (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = pose Id
CreateRule_Statement_Gotohome (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = gotohome

(continued)

Table 6.13 (continued)

Method Name	Return Type	Access Modifier	Description
CreateRule_Statement_Integer_Id_Eq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = integer Id '=' < Expression>
CreateRule_Statement_Letx_Id_Eq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = letx Id '=' < Expression>
CreateRule_Statement_Lety_Id_Eq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = lety Id '=' < Expression>
CreateRule_Statement_Letz_Id_Eq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = letz Id '=' < Expression>
CreateRule_Statement_Move_Long (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = move < Interpolation > ',' '*' ',' '(' < Expression > ',' < Expression > ',' < Expression > ')'
CreateRule_Statement_Move (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = move < Interpolation > ',' Id ',' Id
CreateRule_Statement_Pose_Long (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = pose Id '(' < Expression > ',' < Expression > ',' < Expression > ')'
CreateRule_Statement_Home_Id (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = home Id
CreateRule_Statement_String_Id_Eq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = string Id '=' < Expression>
CreateRule_Statement_Grasp (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = grasp
CreateRule_Statement_Release (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = release
CreateRule_Statement_Drive (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = drive '(' < Expression > ',' < Expression > ')'

(continued)

Table 6.13 (continued)

Method Name	Return Type	Access Modifier	Description
CreateRule_Statement_Drive_Long (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = drive '(' < Expression > ',' < Expression > ')' ',' '(' < Expression > ',' < Expression > ')'
CreateRule_Statement_Rotate (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = rotate < Plane > ',' < Expression > ',' < Expression>
CreateRule_Statement_Out (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = out < Expression > ',' < Expression>
CreateRule_Statement_In (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = in Id '=' < Expression>
CreateRule_Statement_While_Do_End (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = while < Expression > do < Statements > end
CreateRule_Statement_If_Then_End (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = if < Expression > then < Statements > end
CreateRule_Statement_If_Then_Else_End (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Statement > :: = if < Expression > then < Statements > else < Statements > end
CreateRule_Expression_Gt (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '>' < Add Exp>
CreateRule_Expression_Lt (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<' < Add Exp>
CreateRule_Expression_Lteq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<= ' < Add Exp>
CreateRule_Expression_Gteq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '> = ' < Add Exp>
CreateRule_Expression_Eqeq (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '== ' < Add Exp>

(continued)

Table 6.13 (continued)

Method Name	Return Type	Access Modifier	Description
CreateRule_Expression_Ltgt (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Expression > '<> ' < Add Exp>
CreateRule_Expression (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Expression > :: = < Add Exp>
CreateRule_Addexp_Plus (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '+' < Mult Exp>
CreateRule_Addexp_Minus (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '-' < Mult Exp>
CreateRule_Addexp_Amp (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Add Exp > '&' < Mult Exp>
CreateRule_Addexp (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Add Exp > :: = < Mult Exp>
CreateRule_Multexp_Times (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Mult Exp > '*' < Negate Exp>
CreateRule_Multexp_Div (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Mult Exp > '/' < Negate Exp>
CreateRule_Multexp (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Mult Exp > :: = < Negate Exp>
CreateRule_Negateexp_Minus (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Negate Exp > :: = '-' < Value>
CreateRule_Negateexp (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Negate Exp > :: = < Value>
CreateRule_Value_Id (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = Id
CreateRule_Value_Stringliteral (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = StringLiteral

(continued)

Table 6.13 (continued)

Method Name	Return Type	Access Modifier	Description
CreateRule_Value_Numberliteral (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = NumberLiteral
CreateRule_Plane_Identifier (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Plane > :: = PlaneIdentifier
CreateRule_Interpolation_Identifier (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Interpolation > :: = InterpolationIdentifier
CreateRule_Value_Digitliteral (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = DigitLiteral
CreateRule_Value_Lparan_Rparan (ArrayList Tokens);	Reduction	protected abstract	Function to handle < Value > :: = '(' < Expression > ')'
Inner Class Name	Inherited/ Interfaces From	Access Modifier	Description
BaseContext	Reduction	public	Internal class to handle base context

Table 6.14 Summmary of SLIM Robot Position Class

Class summary			
Class ID: CR3			Namespace: SLIMCompiler
Class name: Position			Class type: concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
iXPosition	int	private	Holds the position for X axis
iYPosition	int	private	Holds the position for Y axis
iZPosition	int	private	Holds the position for Z axis
sName	string	Private	Holds the name of the SLIM command
bHomePosition	bool	Private	Holds the position for home
bCurrentPosition	bool	Private	Holds the position for X axis

II. Direction

Direction commands control the moving direction of conveyor belt either Foreword or Backward.
Example:
Direction Foreword

III. Start

Start command starts the conveyor belt for specified time duration. The duration of time can be specified in hours, minutes and seconds.
Example
start s10 starts the conveyor belt for 10 s
start m10 starts the conveyor belt for 10 min
start h3 starts the conveyor belt for 3 h

IV. Timer

Timer controls running time of conveyor belts. Time can be specified in seconds, minutes, and hours.
Example
start s10 start conveyor belt for 10 s
start m10 starts the conveyor belt for 10 min
start h3 starts the conveyor belt for 3 h

6.6.1 Interpreter Design for Conveyors

Due to the simplicity of operation and use of fewer commands to operate the conveyor, an interpreter is the suitable choice for controlling the conveyor. Figures 6.17, 6.18, 6.19, and 6.20 depict the use case diagram, sequence diagram,

Table 6.15 Summary of SLIM Robot PositionArray Class

Class summary			
Class ID: CR4			Namespace: SLIMCompiler
Class name: PositionArray			Class type: concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field Name: Data Type			Access Modifier: Description
PoseArray: static ArrayList			Private: Array to hold the position
Method Name	**Return Type**	**Access Modifier**	**Description**
Clear()	static void	Public	Function to clear the position array
AddPosition(Position NewPosition)	static void	Public	Function to add the new position to array
Setx(string PositionName, int XValue)	static void	Public	Function to set the value of x axis to a position
Getx(string PositionName)	static string	Public	Function to get the value of x axis to a position
Sety(string PositionName, int YValue)	static void	Public	Function to set the value of y axis to a position
Gety(string PositionName)	static string	Public	Function to get the value of y axis to a position
Setz(string PositionName, int ZValue)	static void	Public	Function to set the value of z axis to a position
Getz(string PositionName)	static string	Public	Function to get the value of z axis to a position
SetHome(string PositionName)	static void	Public	Function to set the home position
GetHomePosition()	static Position	Public	Function to get the home position
SetCurrent(string PositionName)	static void	Public	Function to get the current position

Table 6.16 Summary of SLIM Robot GenerateCode class

Class summary			
Class ID: CR5	Namespace: SLIMCompiler		
Class name: GenerateCode	Class Type: Concrete		
Is base class? No	No. of Inherited Classes: 0		
Inherited from: None	Interfaces with: None		
Field name	Data type	Access modifier	Description
SLIMCommands	static ArrayList	Private	Array to hold the SLIM commands
Method Name	Return Type	Access Modifier	Description
AddCommand (string SLIMCommand)	void	Public	Function to add the SLIM command to the array list
GetCommands()	ArrayList	Public	Function to return the array of SLIM commands
ClearCommands()	void	Public	Function to clear the SLIM command from the array list

static diagram, and the flow chart for the interpreter. Figure 6.21 presents the augmented reality for conveyor belt.

Conveyor consists of three projects.

1. ConveyorEngine: Consist of two classes PortAccess class and, Conveyor class; four enumerations and five events. Object of Conveyor class is used in conveyor belt software to provide conveyor functionality. Object of PortAccess is used in Conveyor class to provide access to system physical port. Conveyor Engine is a Dynamic Link Library (DLL) based project. The produced DLL of Conveyor Engine is used in conveyor belt software to provide separation between application and logic; and, logic encapsulation.

   Class structure of Conveyor Engine: Conveyor class has encapsulated all the main functionality required by conveyor belt software as Open File, Save File, Edit File, Execute File, Validate File and Validate Code. A request from conveyor belt software for each function is handled by Conveyor class. The enumerations of Conveyor Engine are Current, ConveyorStatus, Direction and, TimerUnit. Current enumeration identifies the current value of different parameters, ConveyorStatus identifies the running status. Direction identifies the movement direction of belt and TimerUnit identifies the Time Unit for how long belt has to move.
   To provide access to physical port on the system, Conveyor Engine has PortAccess class. PortAccess Class has encapsulated port access and interpolations functions. Only Conveyor class has access to PortAccess class and conveyor belt software does not call PortAccess functions directly. Conveyor class calls

Table 6.17 Summary of SLIM Robot Animation Class

Class summary			
Class ID: CR6		Namespace: SLIMCompiler	
Class name: Animation		Class type: Concrete	
Is base class? No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Data type	Access modifier	Description
_Speed	int	Private	Array to hold the SLIM commands
TV	TVEngine	public	Declaring the TrueVision Engine object
Inp	TVInputEngine	private	Declaring the Input Engine variable
TVLightEngine	TVLightEngine	private	Declaring the Light Engine variable
MatFactory	TVMaterialFactory	Private	Declaring the Material Factory variable
TVScene	TVScene	Private	Declaring the TVScene object
global	TVGlobals	Private	Declaring the Global variable
light	D3DLIGHT8	Private	Declaring the light variable
MeshBase	TVMesh	Private	Declaring base part of robot's mesh
MeshStand	TVMesh	Private	Declaring stand part of robot's mesh
MeshBackArm	TVMesh	Private	Declaring back arm part of robot's mesh
MeshForeArm	TVMesh	Private	Declaring fore arm part of robot's mesh
MeshWrist	TVMesh	Private	Declaring wrist part of robot's mesh
MeshWristJoint	TVMesh	Private	Declaring wrist joint part of robot's mesh
MeshUpperArm	TVMesh	Private	Declaring upper arm part of robot's mesh
MeshLowerArm	TVMesh	Private	Declaring lower arm part of robot's mesh
MeshWorkPiece	TVMesh	Private	Declaring work piece mesh

(continued)

Table 6.17 (continued)

Field name	Data type	Access modifier	Description
BasePosition	D3DVECTOR	Private	Holds the position of base part of robot
StandPosition	D3DVECTOR	Private	Holds the position of stand part of robot
BackArmPosition	D3DVECTOR	Private	Holds the position of back arm part of robot
ForeArmPosition	D3DVECTOR	Private	Holds the position of fore arm part of robot
WristJointPosition	D3DVECTOR	Private	Holds the position of wrist joint part of robot
WristPosition	D3DVECTOR	Private	Holds the position of wrist part of robot
UpperArmPosition	D3DVECTOR	Private	Holds the position of upper arm part of robot
LowerArmPosition	D3DVECTOR	Private	Holds the position of lower arm part of robot
Room	TVMesh	Private	Declaring room mesh
ModelPath	string	Private	Hold the path to load the mesh models
sngPositionX	float	Private	Holds Object position in X axis for keyboard input
sngPositionY	float	Private	Holds Object position in Y axis for keyboard input
sngPositionZ	float	Private	Holds Object position in Z axis for keyboard input
snglookatX	float	Private	Holds Camera position in X axis for keyboard input
snglookatY	float	Private	Holds Camera position in Y axis for keyboard input
snglookatZ	float	Private	Holds Camera position in Z axis for keyboard input
tmpMouseX	int	Private	Holds Mouse input position in X axis
tmpMouseY	int	Private	Holds Mouse input position in Y axis
tmpMouseB1	short	Private	Holds Mouse input button1
tmpMouseB2	short	Private	Holds Mouse input button2
tmpMouseB3	short	Private	Holds Mouse input button3
tmpMouseScrollOld	int	Private	Holds Mouse input scroll old value
tmpMouseScrollNew	int	Private	Holds Mouse input scroll new value
sngAngleX	float	Private	Holds Mouse input angle in X axis
sngAngleY	float	Private	Holds Mouse input angle in Y axis

(continued)

Table 6.17 (continued)

Field name	Data type	Access modifier	Description
sngWalk	float	Private	For smooth movement while camera is walking
sngStrafe	float	Private	For smooth movement while camera is strafing
DoLoop	bool	Private	Holds the Boolean value true until the scene is rendering
x_move	float	private	Holds the Initial move position for room in x axis
y_move	float	private	Holds the Initial move position for room in y axis
z_move	float	private	Holds the Initial move position for room in z axis
X1	float	private	Holds the Initial position for the machine in x axis
Y1	float	private	Holds the Initial position for the machine in y axis
Z1	float	private	Holds the Initial position for the machine in z axis
X2	float	private	Holds the Initial position for the work piece in x axis
Y2	float	private	Holds the Initial position for the work piece in y axis
Z2	float	private	Holds the Initial position for the work piece in z axis
rot	float	private	Holds the Initial rotation angle
Method name	**Return type**	**Access modifier**	**Description**
InitializeComponent (System.Windows.Forms. PictureBox ToRender)	Void	private	Function to initialize all variables of this class
StartRendering()	Void	Public	Function to Start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to Set the DoLoop variable to false
Check_Movement()	Void	Public	Function to Check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to Check keyboard and mouse input and adjust the camera view in 3D.

(continued)

Table 6.17 (continued)

Field name	Data type	Access modifier	Description
Drive(int AxisNumber, int AngleOfRotation)	Void	Public	Function to drive the robot's arm on a specified axis with a specified angle
Drive(int AxisNumber, int AngleOfRotaion, int SecondAxisNumber, int SecondAngleOfRotation)	Void	Public	Overloaded function to drive the robot's arm on a specified two axes and two specified angles
Release()	Void	Public	Function to set the robot's arm in open position
Grasp()	Void	Public	Function to set the robot's arm in close position
Render()	Void	private	Function to display everything that we have rendered to the screen.
Move(Interpolation MoveInterpolation, int StartX, int StartY, int StartZ, int EndX, int EndY, int EndZ)	Void	Public	Function to move the robot to the specified position
SpeedControl()	Void	private	Function to allow the thread to sleep to slow down speed
Rotate(PlaneSpecifier PSpecifier, int Angle, int CenterOfRotation)	Void	Public	Function to rotate the robot's body according to the provided axis, angle and center of rotation
Reset()	Void	Public	Function to set the robot back to initial position
Speed(int SpeedValue)	Void	Public	Function to set the speed of rotation

Table 6.18 Summary of 8SLIM Robot processes sequence

Sequence diagram summary				
Sequence ID: SD-R1 Sequence Name: CNC SLIM Robot Sequence Diagram				
Method name	Type	Description	Source	Destination
DoParse()	Message	Function to set for parsing SLIM Program	Form	SLIMParser
ParseMessage	Return Message	Return message after parsing	Form	SLIMParser
Execute()	Message	Function to execute SLIM Program	Form	SLIMParser
GetMessage	Return Message	Return message after execution	Form	SLIMParser
UpdateInterface()	Self Message	Function to update interface	Form	SLIMParser
Token.Execute()	Self Message	Function to execute a token	SLIMParser	Parser
OpenStream()	Message	Function to open file stream	SLIMParser	Parser
Parse()	Message	Function to parse and validate SLIM Program	SLIMParser	Parser
Response	Return Message	Return message after validation	SLIMParser	Parser
GetCurrentReduction()	Message	Function to get the token object	SLIMParser	Parser
Token()	Message	Function to generate the token	Parser	Token
Reset()	Self Message	Function to reset the robot position	Parser	Token
PrepareToParse()	Self Message	Function to prepare parsing	Parser	Token
PushToken()	Message	Function to push the token to stack	Parser	TokenStack
PullToken()	Message	Function to pull the token from stack	Parser	TokenStack
Token	Return Message	Return message containing token	TokenStack	SLIMParser
Move()	Message	Function to send the move command to robot	Form	Robot3D
Rotate()	Message	Function to send the rotate command to robot	Form	Robot3D
Grasp()	Message	Function to send the grasp command to robot	Form	Robot3D
Release()	Message	Function to send the release command to robot	Form	Robot3D
StopRendering()	Message	Function to stop rendering the scene	Form	Robot3D
StartRendering()	Message	Function to start rendering the scene	Form	Robot3D

Table 6.19 Summary of SLIM Robot Open use case

Use case summary
Use Case ID: UC-R1
Use Case Name: Open
Actors: User
Description: The user selects and loads part program through a file with "rbt" extension
Preconditions: SLIM program file is present on the system.
Postconditions: SLIM program is loaded on the screen
System parameters: SLIM program file
Success scenario: SLIM program is loaded successfully
Alternative scenario: Error in loading part program
Includes: Open File
Priority: SetFilePath

Table 6.20 Summary of SLIM Robot New use case

Use case summary
Use Case ID: UC-R2
Use Case Name: New
Actors: User
Description: The user inputs a new SLIM program on the screen
Preconditions: None
Postconditions: SLIM program is validated and saved
System parameters: SLIM program with SLIM standard
Success scenario: New SLIM program file is created successfully
Alternative scenario: Error in validating SLIM program or in saving file
Includes: None
Priority: High

Table 6.21 Summary of SLIM Robot Parse use case

Use case summary
Use case ID: UC-R3
Use case name: Parse
Actors: User
Description: The system parses and validates the SLIM program on the screen and saves it to a file with "rbt" extension
Preconditions: None
Postconditions: SLIM program is saved on the file
System parameters: SLIM program with SLIM standard
Success scenario: SLIM program is validated and saved successfully
Alternative scenario: Error in saving part program
Includes: DoParse, OpenStream
Priority: High

Table 6.22 Summary of SLIM Robot Run use case

Use case summary
Use case ID: UC-R4
Use case name: Run
Actors: User
Description: The user executes the SLIM program after loading it on the screen
Preconditions: SLIM program is loaded and validated
Postconditions: SLIM program executes and display the robot movement
System parameters: SLIM program with SLIM standard
Success scenario: SLIM program is executed successfully
Alternative scenario: Error in executing part program
Includes: Parse, Execute, Reset
Priority: High

Table 6.23 Summary of SLIM Robot process flow chart

Flow chart summary	
Flow chart ID: FC-R1	
Flow chart name: CNC SLIM Robot Process Flow Chart	
No. of inputs: 6	
No. of processes: 32	
No. of outputs: 2	
No. of control decisions: 20	
Input name	Description
Save File	Save menu on screen
Execute File	Run menu on screen
Open File	Load menu on screen
New Menu	New menu on screen
Parse File	Reset menu on screen
Exit Application	Exit menu on screen
Process Name	Description
Run	Process to execute SLIM Program
Reset Robot	Process to reset the robot to initial position
Parser Execute	Process to execute parser
New part program	Process to New Part Program
Stop Rendering	Process to close rendering and exit
Clear Parse Log Window	Process to clear the movement log from screen
Do Parse	Process to set the parse flag
Get Token	Process to get the token statement
Pop Token	Process to pop token
Push Token	Process to push token
Parse Token	Process to parse token
Check Parse Result	Process to check the parse result
Add Entry in Log Window	Process to add entry in log window
Create Statement for every command	Process to create statements
Execute Statement	Process to execute statements

(continued)

Table 6.23 (continued)

Input name	Description
Generate Event for message	Process to generate event
Get Current Reduction	Process to get the reduction statement
Peek Token	Process to find token
Execute Token	Process to execute token after parsing
Parse Message Arrive	Process to parse message arrive
Split Message	Process to split message into tokens
Set Robot Speed	Process to set robot's speed
Robot Drive	Process to set robot's drive
Get Interpolation	Process to get interpolation
Robot Move Linear	Process to perform robot's linear movement
Robot Move Circular	Process to perform robot's circular movement
Get Plane Specified	Process to get specified plane
Rotate Robot on XY Plane	Process to rotate robot on XY plane
Rotate Robot on YZ Plane	Process to rotate robot on YZ plane
Rotate Robot on ZX Plane	Process to rotate robot on ZX plane
Robot Hand Close	Process to close robot's hand
Robot Hand Open	Process to open robot's hand
Output Name	Description
Robot's display	Robot loaded on screen
Robot's position display	Robot's movement position values on screen
Control Decision Name	Description
Open Dialog Box	Open Dialog box for choosing the file
Select Part Program	Select the SLIM program file
Save Dialog Box	Save Dialog Box
Input Token = 0	Check if Input Token = 0
Comments Level > 0	Check if Comments Level > 0
Token = NULL	Check if Token = NUL
Token = Whitespace	Check if Token = white space
Peek Token	Search for token
Check Parse Result for Syntax error	Check parse token for syntax
Reduction	Check if reduction is to applied
Message = Speed	Check if Message = Speed
Message = Drive	Check if Message = Drive
Message = Move	Check if Message = Move
Interpolation = L	Check if Interpolation = L
Message = Rotate	Check if Message = Rotate
Message = Grasp	Check if Message = Grasp
Message = Release	Check if Message = Release
Plane = XY	Check if Plane = XY
Plane = YZ	Check if Plane = YZ
Plane = XZ	Check if Plane = XZ

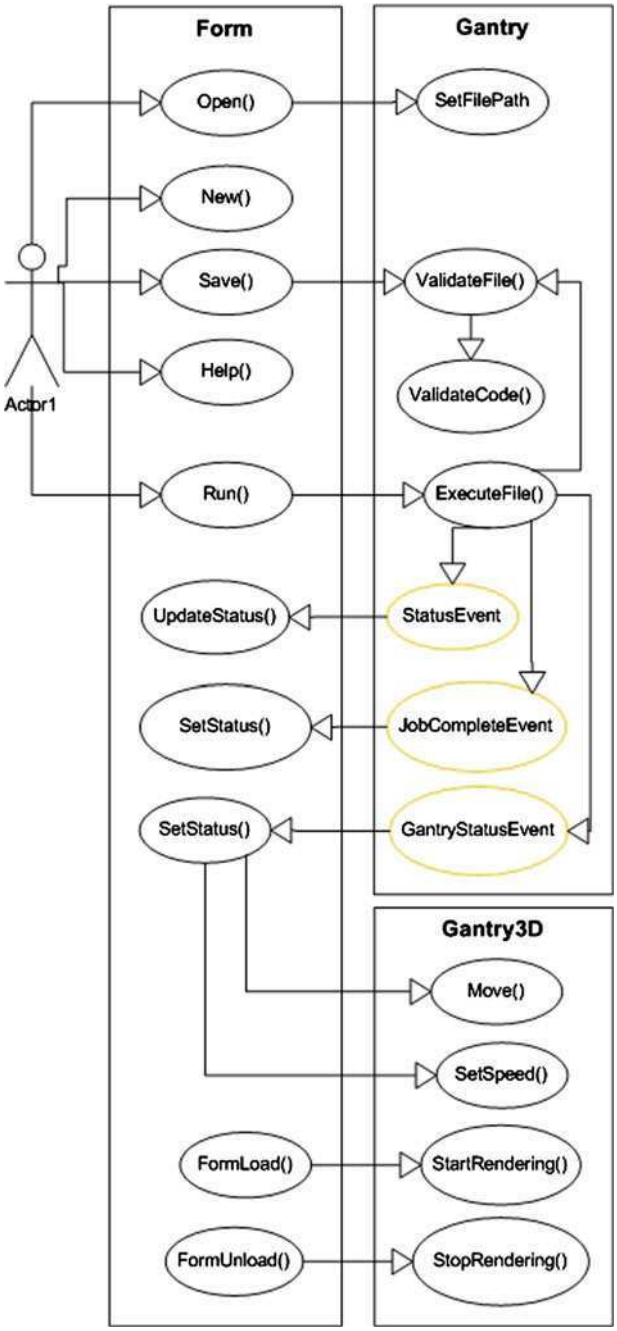


Fig. 6.12 Use Case Diagram of Gantry

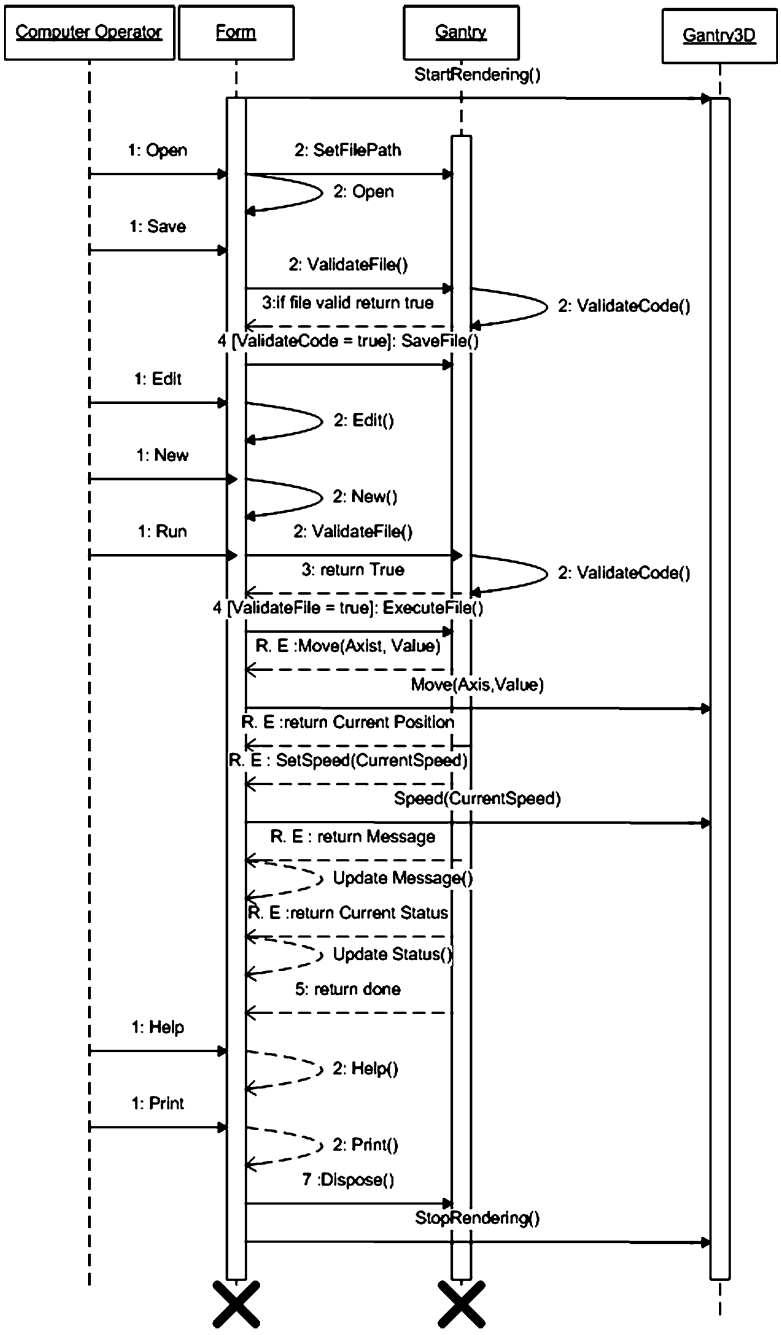


Fig. 6.13 Gantry UML Sequence Diagram

Fig. 6.14 Gantry UML Static Diagram

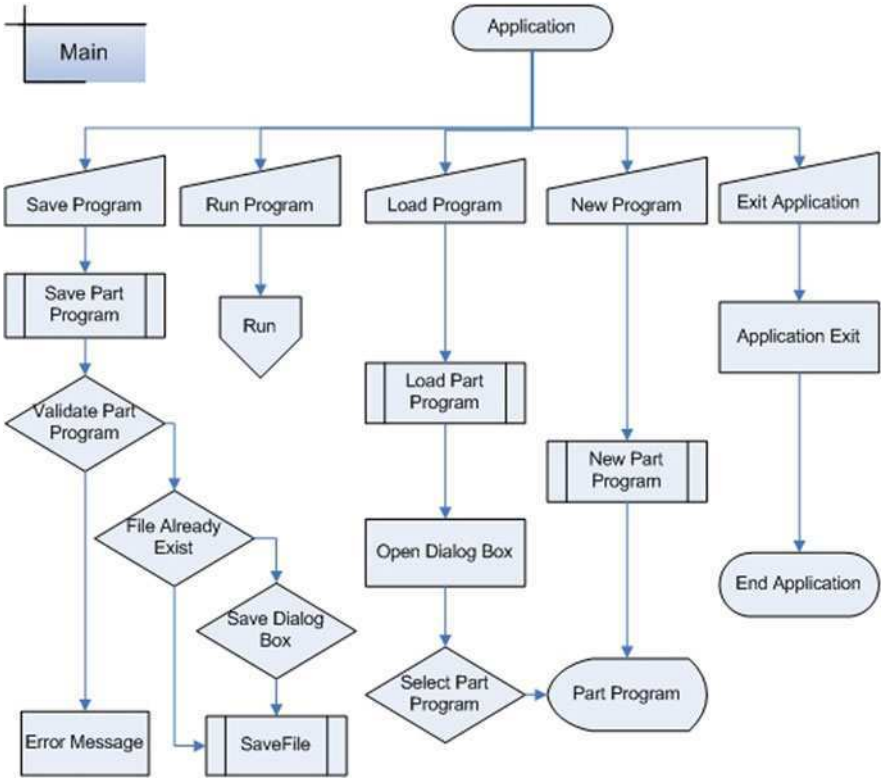


Fig. 6.15 Flow chart for Gantry's Interpreter

appropriate interpolation methods of PortAccess, which in turn calls private methods of PortAccess to send and receive physical port signal.
In order to update the conveyor belt software about the current state of execution Conveyor class has four events, DirectionEvent, SpeedEvent, StartConveyorEvent and StatusEvent. Direction event sets the conveyor belt direction and SpeedEvent sets the moving speed of conveyor belt. StartConveyorEvent start conveyor belt and StatusEvent update the user interface and 3D status of conveyor belt software.

2. ConveyorMachine: Consist of a class 'Form1' which is inherited by System. Windows.Forms.Form to provide conveyor belt graphical user interface that utilizes Conveyor Engine object to execute files and related functions. Current status of execution and control of 3D animation is also updated from ConveyorMachine.
3. CoreAnimation: Consist of Conveyor3D class and three enumerations. Conveyor3D class consists of 3D graphics rendering engine and methods to control movement of individual parts of machine. Enumerations of CoreAnimation are ConveyorType, Spindle and Axis. ConveyorType enumeration is for

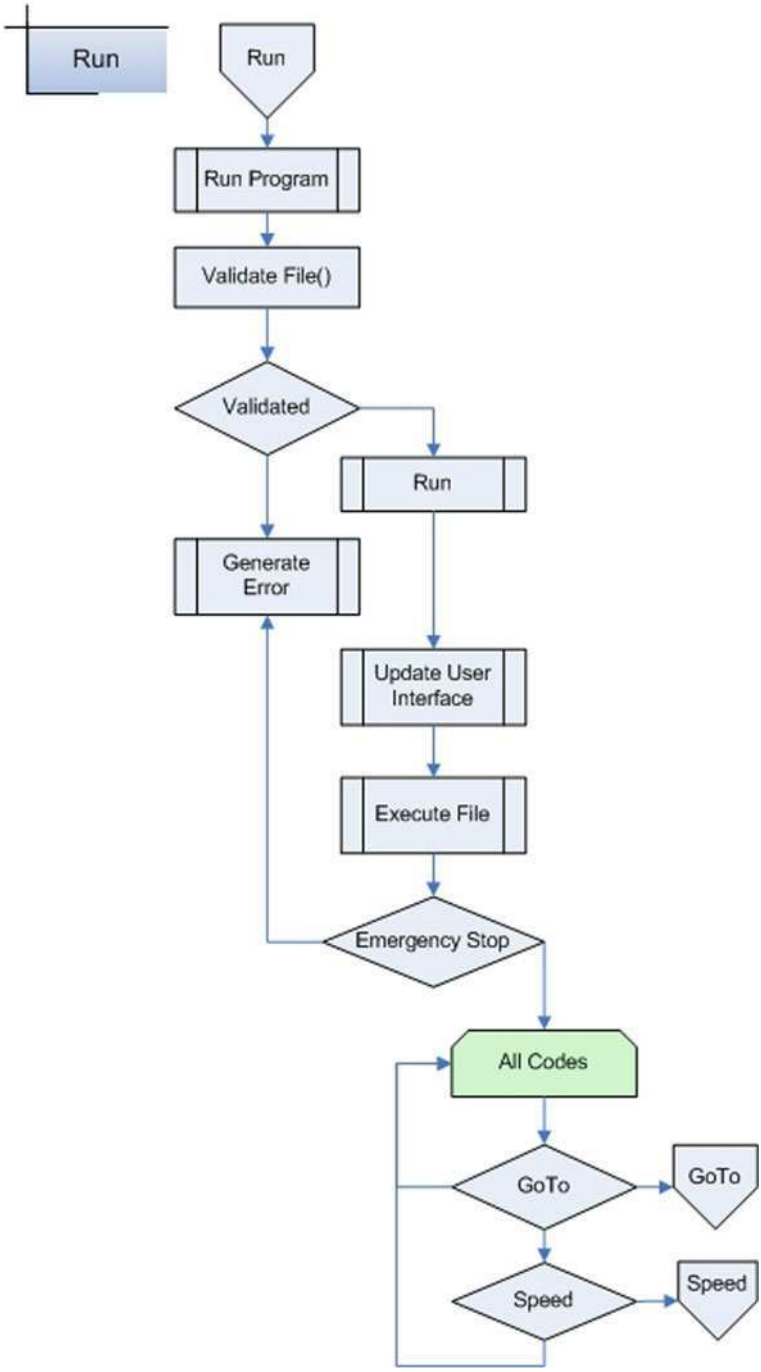


Fig. 6.15 (continued)

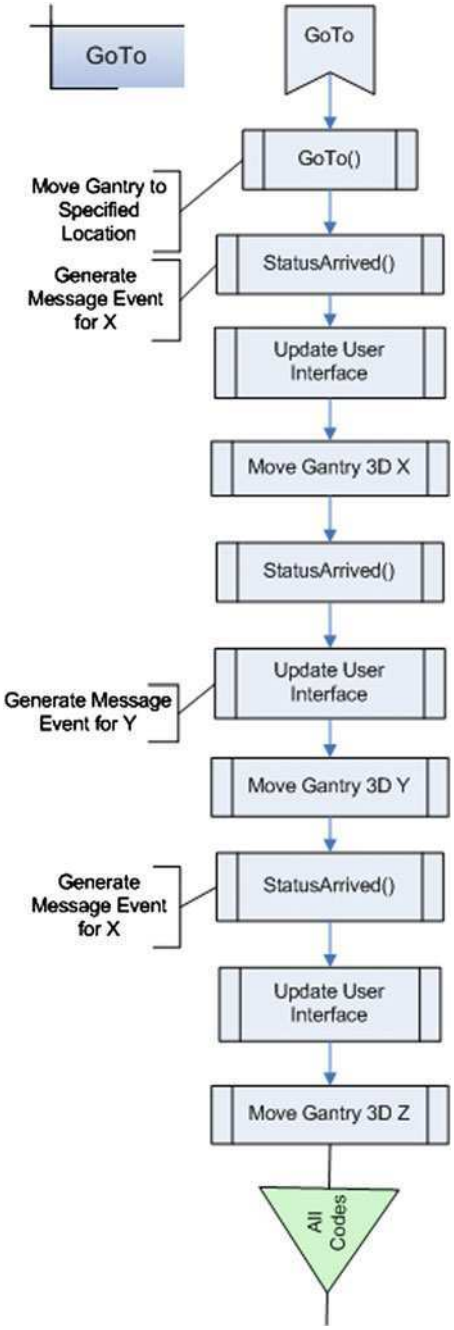


Fig. 6.15 (continued)

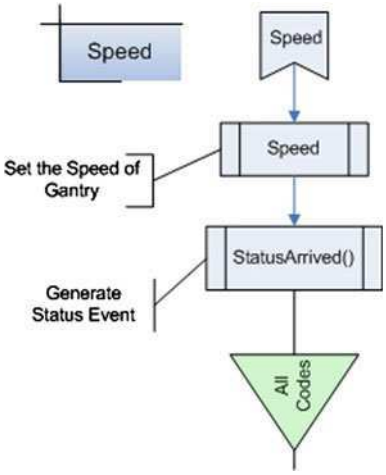


Fig. 6.15 (continued)

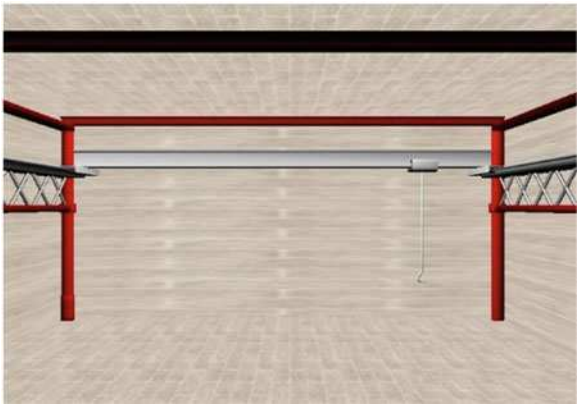


Fig. 6.16 Augmented Reality for Gantry Crane

type of Conveyor. Spindle enumeration controls the running status of spindle. Axis enumeration is for controlling the axis movement of belt.

Table 6.33 shows the PortAccess class member's properties, methods and variables. PortAccess is the helper class of Conveyor class and this class is used to pass pulses to CNC machine by using parallel port and calculate the pulses and timings. PortAccess also holds current state of the conveyor belt.

Table 6.34 outlines the Conveyor class members variables, properties and methods. The Conveyor class is used to read the part program file, validate it, open it, save it and executes it. This class is also responsible for raising events to update

Table 6.24 Summary of Gantry PortAccess class

Class summary	
Class ID: CG1	Namespace: GantryEngine
Class name: PortAccess	Class type: Concrete
Is base class?: No	No. of inherited classes: 0
Inherited from: None	Interfaces with: None

Field name	Data type	Access modifier	Description
_CurrentPositionX	Double	Private	Holds the current position of work piece in X axis.
_CurrentPositionY	Double	Private	Holds the current position of work piece in Y axis
_Radius	Double	Private	Holds the radius of the circular movement
_I	Double	Private	Holds the starting position in X axis
_K	Double	Private	Holds the starting position in Y axis
_NextPositionX	Double	Private	Holds the Next position of work piece in X axis
_NextPositionY	Double	Private	Holds the Next position of work piece in Y axis
_xvalue	Double	Private	Holds the X value
_yvalue	Double	Private	Holds the Y value
Parabola	string	Private	Store the type of parabola
_RapidMovement	Bool	Private	Holds the Boolean value to rapid movement at a machine tool used for positioning the tool.
_LinearInterpolation	Bool	Private	Holds the Boolean value to allow direct movement between two points through linear interpolation
bMetricUnit	Bool	Private	Holds the Boolean value for To check if the Unit is set to Metric System
bAbsoluteCoordinate	Bool	Private	Holds the Boolean value for To check if the coordinate system is set to absolute coordinate
_Speed	short	Private	Holds the value to control machine speed
_Feed	short	Private	Holds the feed of the machine

(continued)

Table 6.24 (continued)

Field name	Data type	Access modifier	Description
_Tool	short	Private	Holds the tool of the machine
xPulses	int	Private	Sets the pulse of the machine in X axis
yPulses	int	Private	Sets the pulse of the machine in Y axis
EmergencyStop	static bool	Private	Holds the Boolean value to control the machine running status
Method name	**Return type**	**Access modifier**	**Description**
CalculateLinearInterpolation (double X1, double Y1, double X2, double Y2)	Void	Public	Function to calculate linear interpolation
Delay()	Void	Public	Function to cause a delay of defined no. of cycles
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to increase the yPulses
Yminus()	Void	Public	Function to decrease the yPulses
G00()	Void	Public	Function to generate Point to Point Positioning
G01()	Void	Public	Function to generate Linear Interpolation
G70()	Void	Public	Sets machine mode for Inch Programming
G71()	Void	Public	Sets machine mode for Metric Programming
G90()	Void	Public	Sets machine mode for Absolute Input
G91()	Void	Public	Sets machine mode for Incremental Input
M02()	Void	Public	Sets machine mode for Program Stop
M03()	Void	Public	Sets machine mode for Machine CW
M04()	Void	Public	Sets machine mode for Machine CCW
M05()	Void	Public	Sets machine mode for Machine off

Table 6.25 Summary of Gantry class

Class summary			
Class ID: CG2			Namespace: GantryEngine
Class name: Gantry			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: IDisposable
Class summary			
Field name	Data type	Access modifier	Description
sFileName	static string	public	Holds the File Name of.cnc file
aList	static ArrayList	private	Holds the GM Codes extracted from.cnc file.
CodeList	static ArrayList	private	Holds all GM Codes
component	Component	Private	Holds other managed resource this class uses.
disposed	bool	Private	Track whether Dispose has been called.
Port	PortAccess	Private	Used to call the port functions for sending GM codes.
status	GantryStatus	Private	Holds and sets the machine status
sMachineName	string	Private	Holds the machine name
Method name	Return type	Access modifier	Description
Dispose()	Void	Private	Function to dispose the rendering
EmergencyStop()	Void	Public	Function to stop the rendering if Emergency Stop is pressed
OpenFile(TextBox textBox)	Void	Public	Function used for open cnc file and display in textbox object based in this function
WriteError (string sErrorMessage, string sErrorSource, string sErrorNumber, string sFunctionThatGenerateError)	static void	public	Function to write errors in error log

(continued)

Table 6.25 (continued)

Method name	Return type	Access modifier	Description
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute
ValidateCode(string sGMCode,ref bool bIsGMCodeValid)	Void	Public	This Function will validate the GMCode
ValidateFile()	bool	Public	This Function will validate the cnc part program file. It will take file name from static variable set by calling routine. If cnc part program file is valid true will be return else false
SaveFile(TextBox textBox)	Void	Public	This Function will Save the Part program file. It will take file name from static variable set by calling routine

Event Name	Delegate	Access Modifier	Description
GantryStatusArrived	GantryStatusHandler	Public	Event to handle the gantry message arrived
StatusArrived	StatusHandler	Public	Event to handle the gantry status arrived
WorkCompleted	Completed	Public	Event to handle work completed status
SpeedArrived	SpeedHandler	Public	Event to handle the speed

Table 6.26 Summary of Gantry3D class

Class summary			
Class ID: CG3			Namespace: CoreAnimation
Class name: Gantry3D			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
TV	TVEngine	public	Declaring the TrueVision Engine object
Inp	TVInputEngine	private	Declaring the Input Engine variable
TVLightEngine	TVLightEngine	private	Declaring the Light Engine variable
MatFactory	TVMaterialFactory	Private	Declaring the Material Factory variable
TVScene	TVScene	Private	Declaring the TVScene object
global	TVGlobals	Private	Declaring the Global variable
light	D3DLIGHT8	Private	Declaring the light variable
sngPositionX	float	Private	Holds Object position in X axis for keyboard input
sngPositionY	float	Private	Holds Object position in Y axis for keyboard input
sngPositionZ	float	Private	Holds Object position in Z axis for keyboard input
snglookatX	float	Private	Holds Camera position in X axis for keyboard input
snglookatY	float	Private	Holds Camera position in Y axis for keyboard input
snglookatZ	float	Private	Holds Camera position in Z axis for keyboard input
tmpMouseX	int	Private	Holds Mouse input position in X axis
tmpMouseY	int	Private	Holds Mouse input position in Y axis
tmpMouseB1	short	Private	Holds Mouse input button1
tmpMouseB2	short	Private	Holds Mouse input button2

(continued)

Table 6.26 (continued)

Field name	Data type	Access modifier	Description
tmpMouseB3	short	Private	Holds Mouse input button3
tmpMouseScrollOld	int	Private	Holds Mouse input scroll old value
tmpMouseScrollNew	int	Private	Holds Mouse input scroll new value
sngAngleX	float	Private	Holds Mouse input angle in X axis
sngAngleY	float	Private	Holds Mouse input angle in Y axis
sngWalk	float	Private	For smooth movement while camera is walking
sngStrafe	float	Private	For smooth movement while camera is strafing
DoLoop	bool	Private	Holds the Boolean value true until the scene is rendering
ModelPath	string	Private	Variable for storing model paths
GantryX	float	Private	Holds Initial position for the machine in X axis
GantryY	float	Private	Holds Initial position for the machine in Y axis
GantryZ	float	Private	Holds Initial position for the machine in Z axis
MeshGantry	TVMesh	Private	Holds the mesh for machine
MeshSlide	TVMesh	Private	Holds the mesh for slide
MeshSlider	TVMesh	Private	Holds the mesh for slider
SpeedControl	D3DVECTOR	Private	Holds the 3D vector of speed in X, Y, Z directions
GantryPosition	float	Private	Holds Initial position for the machine
SlidePosition	float	Private	Holds Initial position for the slide
SliderPosition	float	Private	Holds Initial position for the slider

(continued)

Table 6.26 (continued)

Method name	Return type	Access modifier	Description
InitializeComponent (System.Windows .Forms.PictureBox ToRender)	Void	private	Function to initialize all variables of this class
Move(Axis axis, int Value)	Void	Public	Function to move the gantry crane in X, Y, or Z axis
StartRendering()	Void	Public	Function to Start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to Set the DoLoop variable to false
Check_Movement()	Void	Public	Function to Check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to Check keyboard and mouse input and adjust the camera view in 3D
Rendering()	Void	Public	Function to display everything that we have rendered to the screen

Table 6.27 Summary of Gantry sequence diagram

Sequence diagram summary				
Sequence ID: SD-G1 Sequence name: Gantry Sequence Diagram				
Method name	Type	Description	Source	Destination
Open()	Message	Function to Load Part Program	User	Form
SetFilePath()	Return Message	Function to set the file path to open file on disk	Form	Gantry
Save()	Message	Function to Save Part Program	User	Form
ValidateCode()	Self Message	Function to validate GM codes	Gantry	Gantry
SaveFile()	Message	Function to save file on disk	Form	Gantry
New()	Self Message	Function to New Part Program	User	Form
Run()	Message	Function to Execute Part Program	User	Form
ValidateFile()	Return Message	Function to validate program file	Form	Gantry
ExecuteFile()	Return Message	Function to execute program file	Form	Gantry
Dispose()	Message	Function to close rendering and exit	Form	Gantry
Help()	Message	Function to open the help manual	User	Form
Print()	Message	Function to print the program	User	Form
Move()	Message	Function to move the gantry	Form	Gantry3D
SetSpeed()	Message	Function to set the gantry speed	Form	Gantry3D
StopRendering()	Message	Function to stop rendering the 3D scene	Form	Gantry3D
UpdateStatus()	Return Message	Function to return message about status of gantry	Form	Gantry
UpdateMessage()	Return Message	Function to return message as string	Form	Gantry

Table 6.28 Summary of Gantry Open Part Program use case

Use case summary
Use Case ID: UC-G1
Use Case Name: Open
Actors: User
Description: The user selects and loads part program through a file with “cnc” extension
Preconditions: Part program file is present on the system
Postconditions: Part program is loaded on the screen
System parameters: Part program file
Success scenario: Part program is loaded successfully
Alternative scenario: Error in loading part program
Includes: Set File Path
Priority: High

user interface and 3D conveyor machine. Interpreter of conveyor is also implemented in this class’s ExecuteFile method.

Table 6.35 shows the Conveyor3D class member variables, properties and methods. Conveyor3D class is used to render the graphics of conveyor belt in 3D format for display movement of machine and work piece. 3D rendering engine is also implemented in this class.

Table 6.29 Summary of Gantry New Part Program use case

Use case summary
Use Case ID: UC-G2
Use Case Name: New
Actors: User
Description: The user inputs a new part program on the screen
Preconditions: None
Postconditions: Part program is validated and saved
System Parameters: Part program with GM codes
Success Scenario: New part program file is created successfully
Alternative Scenario: Error in validating part program or in saving file
Includes:
Priority: High

Table 6.30 Summary of Gantry Save Part Program use case

Use case summary
Use case ID: UC-G3
Use Case Name: Save
Actors: User
Description: The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions: None
Postconditions: Part program is saved on the file
System parameters: Part program with GM codes
Success scenario: Part program is validated and saved successfully
Alternative scenario: Error in saving part program
Includes: Validate File
Priority: High

Table 6.31 Summary of Gantry Run Part Program use case

Use case summary
Use case ID: UC-G4
Use case name: Run
Actors: User
Description: The user executes the part program after loading it on the screen
Preconditions: Part program is loaded and validated
Postconditions: Part program executes and display the machine movement and end result of work piece along with task bar description with machine movement in X, Y and Z axes
System parameters: Part program with GM codes
Success scenario: Part program is executed successfully
Alternative scenario: Error in executing part program
Includes: Validate File, Execute File
Priority: High

Table 6.32 Summary of Gantry process flow chart

Flow chart summary	
Flow chart ID: FC-G1	
Flow chart name: Gantry Process Flow Chart	
No. of inputs: 5	
No. of processes: 16	
No. of outputs: 1	
No. of control decisions: 7	
Input name	Description
Save Program	Save button on screen
Run Program	Run button on screen
Load Program	Load button on screen
New Program	New button on screen
Exit Application	Exit button on screen
Process Name	Description
Load Part Program	Function to Load Part Program
Save Part Program	Function to Save Part Program
Open Dialog Box	Function to open file on disk
New Part Program	Function to New Part Program
Update User Interface	Function to update values on user interface screen
Speed	Function to set the speed of machine
End Application	Function to close rendering and exit
Save File	Function to save file on disk
Move Gantry 3D X	Function to move gantry in x axis
Move Gantry 3D Y	Function to move gantry in y axis
Move Gantry 3D Z	Function to move gantry in z axis
Validate File	Function to validate program file
Execute File	Function to Execute Part Program
All Codes Loop	Loop to check the GM code
Message Arrived	Event to check a message
Status Arrived	Event to check a status
Output Name	Description
3D Gantry display	3D Gantry animation on screen
Control Decision Name	Description
Validate Part Program	Check to validate GM codes
Select Part Program	Select the program file
Save Dialog Box	Save Dialog Box
Validated	Check if File is validated
Emergency Stop	Stop rendering and reset
File Already Exists	Check if file already exists on the specified path
GoTo	Move Gantry to specified location

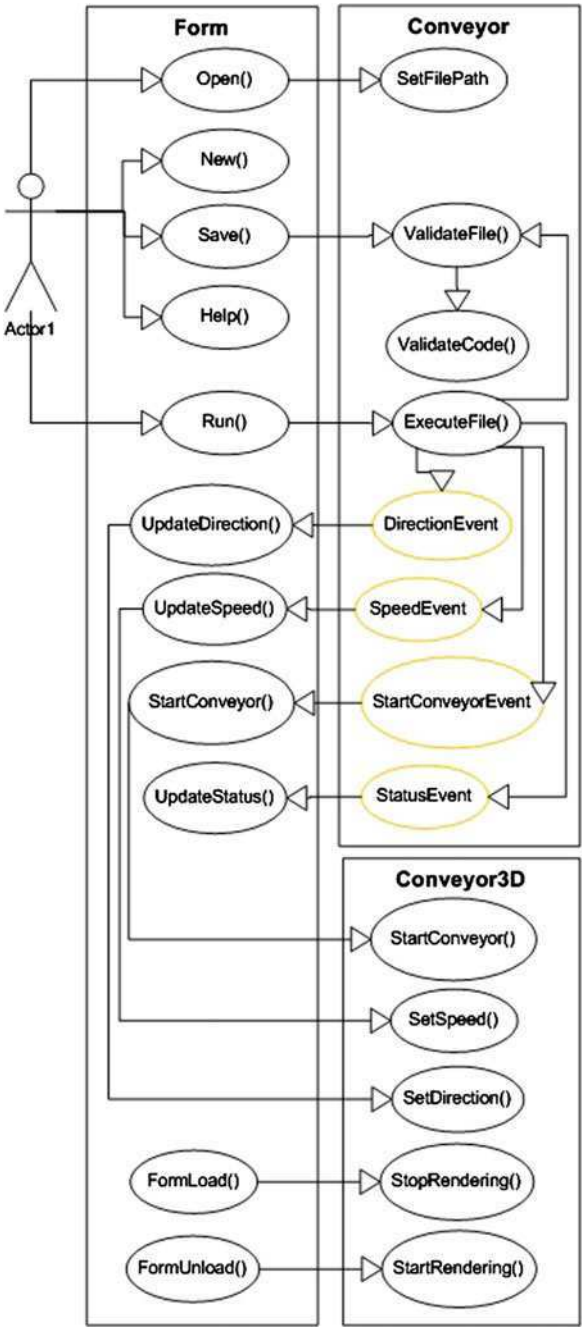


Fig. 6.17 Use Case Diagram of Conveyor Belt

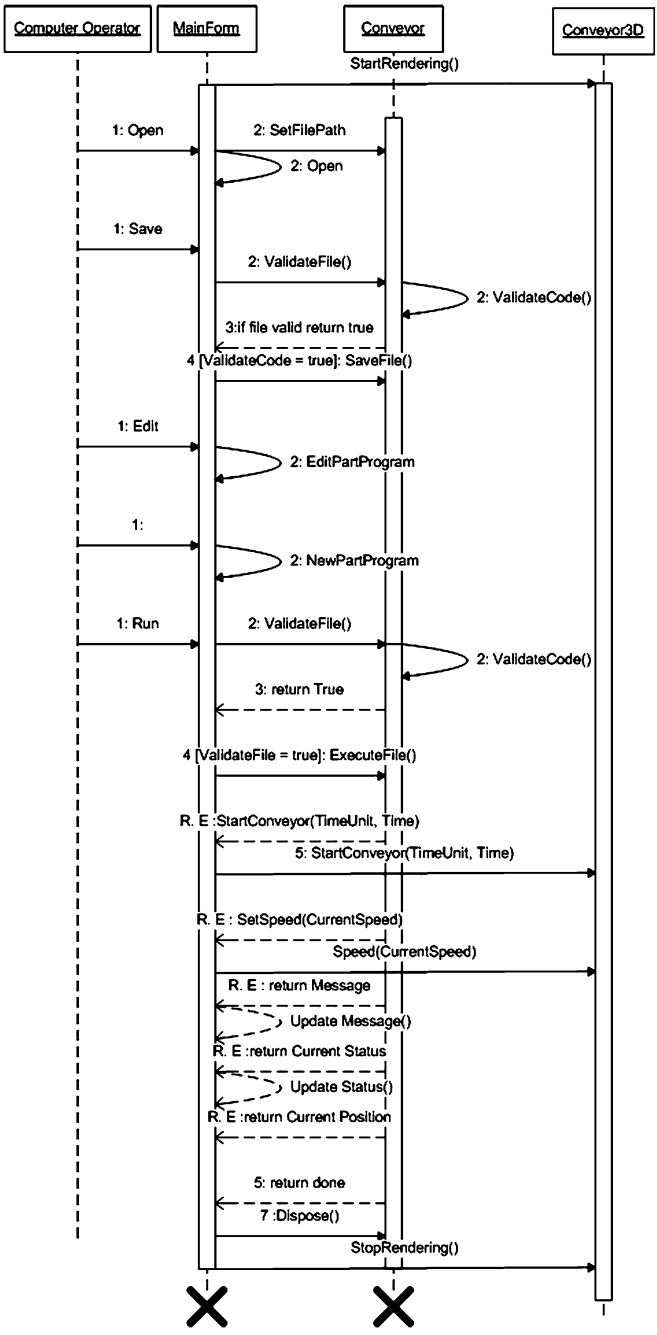


Fig. 6.18 Conveyor Belt UML Sequence Diagram

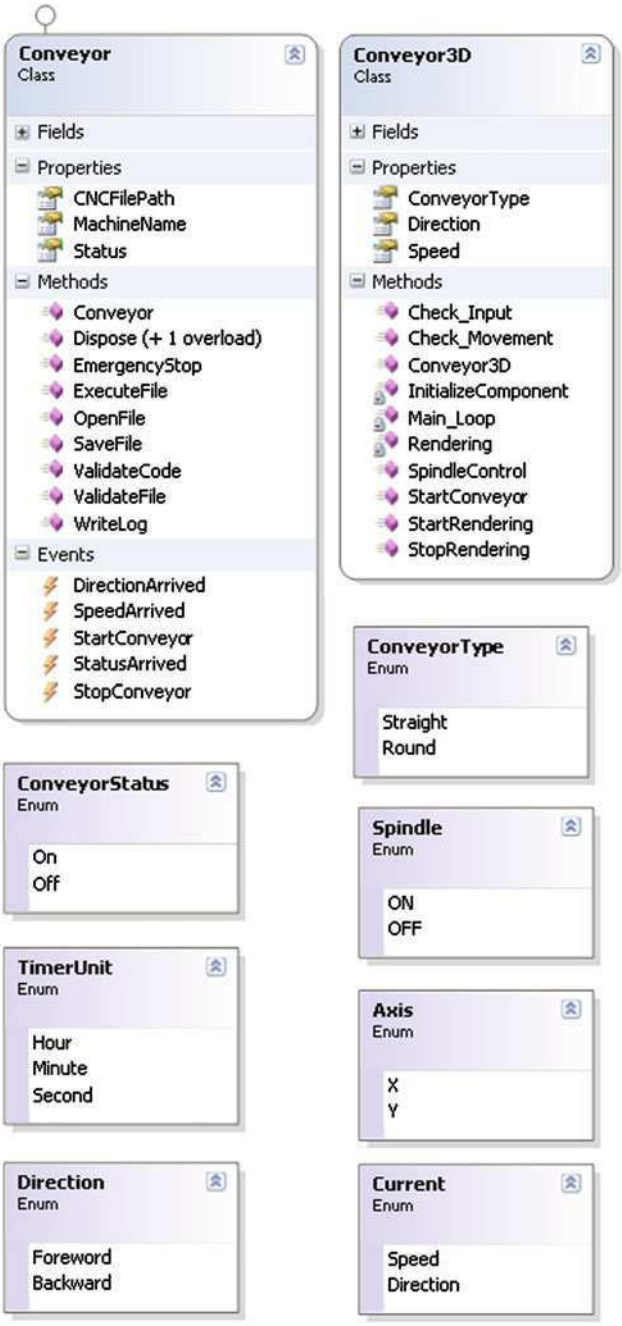


Fig. 6.19 Conveyor Belt Static Diagram

Fig. 6.19 (continued)

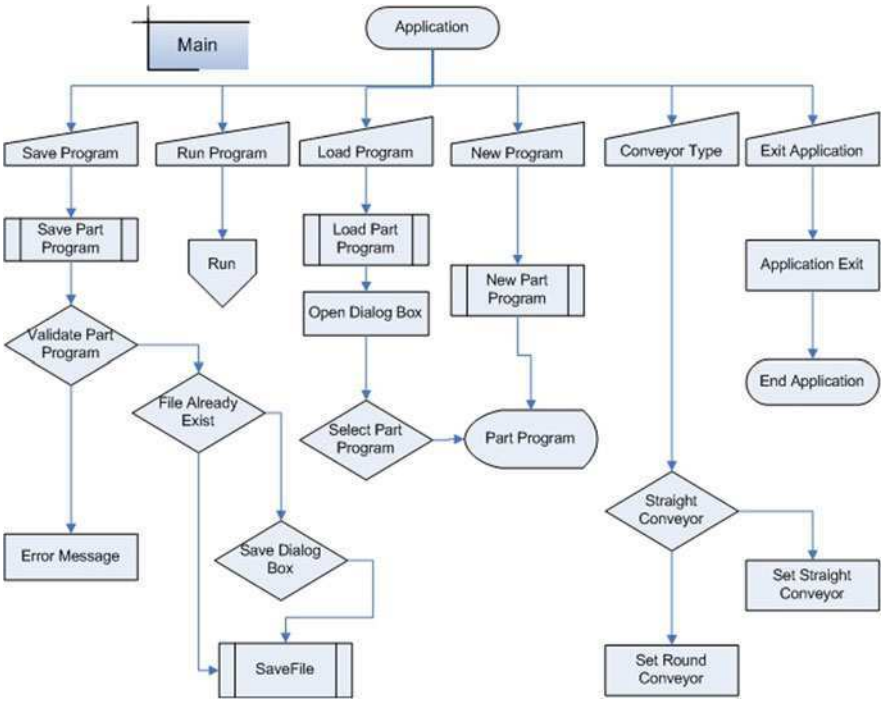


Fig. 6.20 Flow chart for the Conveyor Interpreter

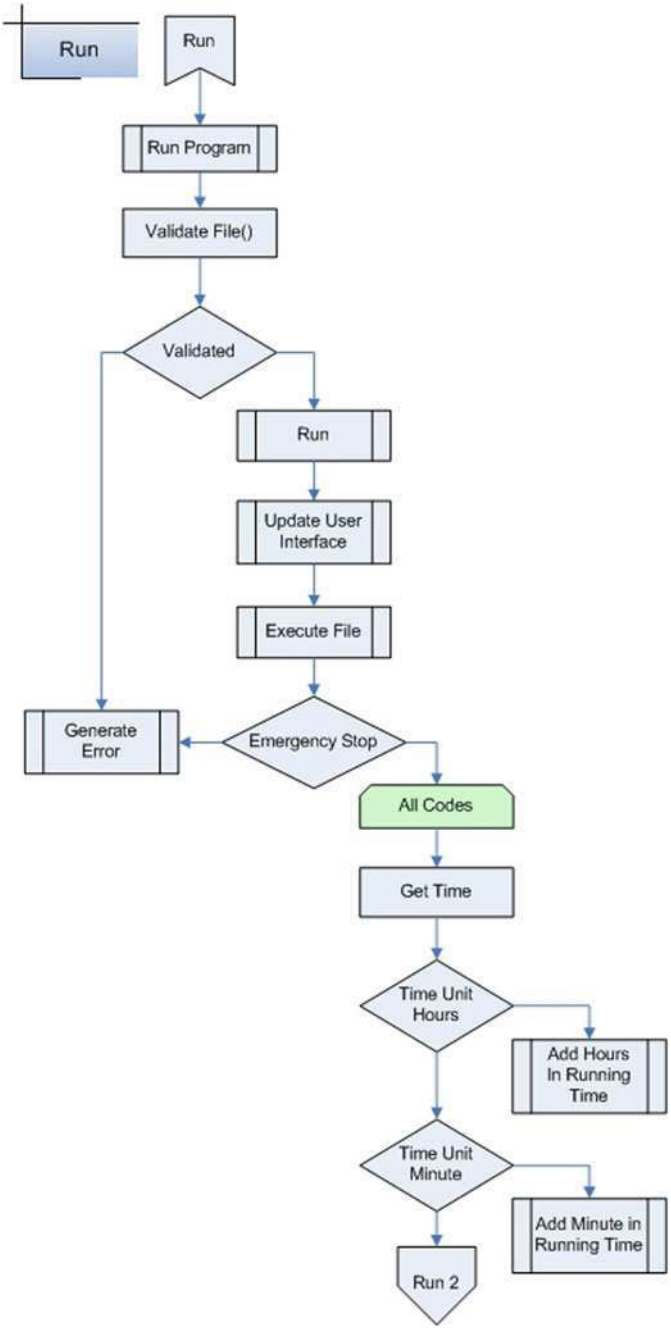


Fig. 6.20 (continued)

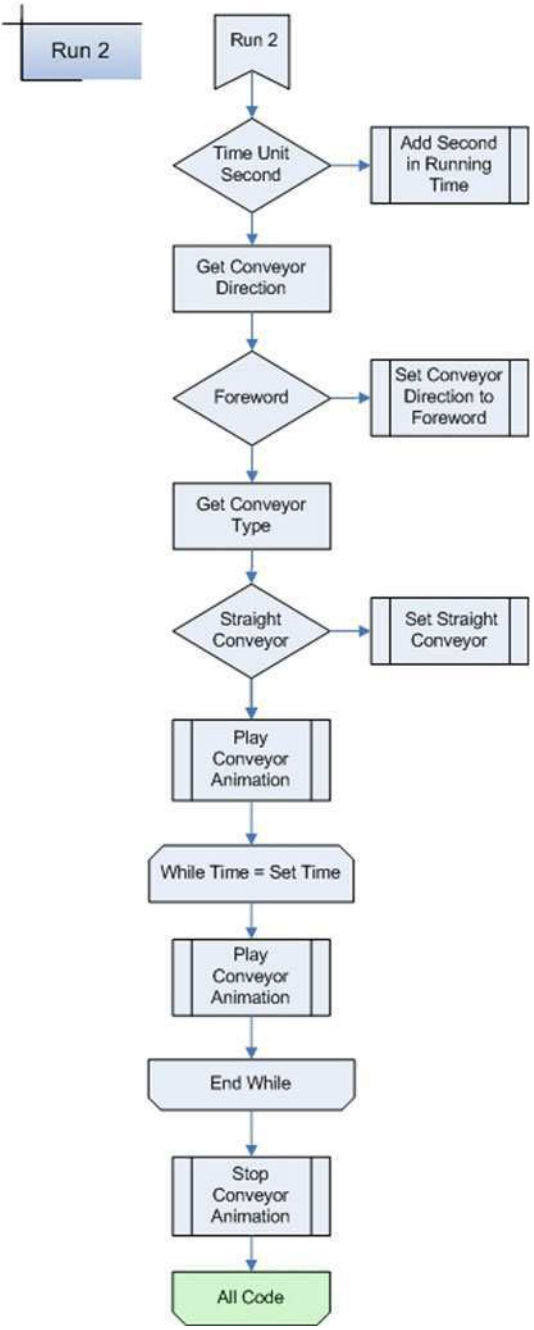


Fig. 6.20 (continued)

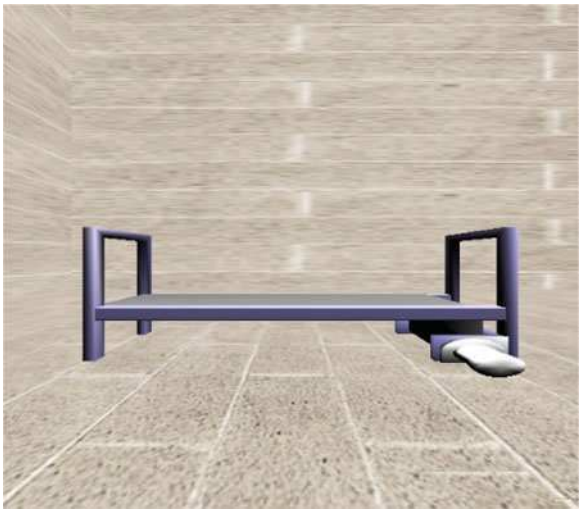


Fig. 6.21 Augmented Reality for the Conveyor

Table 6.36 presents the summary of the Conveyor Belt sequence diagram.

Table 6.37 describes the summary of the conveyor belt Open Part Program use case.

Table 6.38 illustrates the summary of the conveyor belt New Part Program use case.

Table 6.39 outlines the summary of the conveyor belt Save Part Program use case.

Table 6.40 depicts the summary of the conveyor belt Run Part Program use case.

Table 6.41 produces the summary of the conveyor belt processes flow chart.

Following code lists the interpreter for the conveyor. This C# code is taken from Conveyor class' ExecuteFile member method. ExecuteFile method is the interpreter of conveyor belt software. In this code it reads part program command one by one from an array and instructs PortAccess class to calculate required number of pulses by using timers and update user interface and 3D machine with the help of events. ValidateFile method is formerly called Execute method. ValidateFile method validates part program commands with ValideCode method and fills an array with validated commands. Incase of any invalid command, ExecuteFile method is not called and error is returned to the user interface of conveyor.

Intorpreter for Conveyor

```
public void ExecuteFile()
        {


            if (this.StatusArrived != null)
                this.StatusArrived(ConveyorStatus.On);
            string Code;
            string TimeUnit;
            string Command;
            TimerUnit TUnit = TimerUnit.Minute;
            int Time = 0;
            for (int i = 0; i < aList.Count; i++)
            {


                Thread.Sleep(10);
                /// if Emergency stop then exit from loop
                if (PortAccess.EmergencyStop)
                    break;
                Code = aList[i].ToString();
                if (Code == "Start" || Code == "start")
                {
                    // Start Command
                    //  Format  is  "Start  10  M"  OR  "Start  2  H"  OR
"Start 2 S"
                    // m = minute, h = hour , s = second
                    Code = aList[i + 1].ToString();
                    TimeUnit = Code.Substring(0, 1);
                    switch (TimeUnit)
                    {
                        case "H":
                            TUnit = TimerUnit.Hour;
                            break;
                        case "M":
                            TUnit = TimerUnit.Minute;
                            break;
                        case "S":
                            TUnit = TimerUnit.Second;
                            break;
                        case "h":
                            TUnit = TimerUnit.Hour;
                            break;
                        case "m":
                            TUnit = TimerUnit.Minute;
                            break;
                        case "s":
                            TUnit = TimerUnit.Second;
                            break;
                        default:
                            break;
                    }
                    Time = Convert.ToInt16(Code.Substring(1));
                    if (this.StartConveyor != null)
                        this.StartConveyor(TUnit,Time);
                    i++;
```

```
                }
                else if (Code == "Direction" || Code == "DIRECTION")
                {
                    Code = aList[i + 1].ToString();
                    if      (Code.Trim().ToString()      ==      Direc-
tion.Foreword.ToString())
                    {
                        this.ConveyorDirection = Direction.Foreword;
                        if (this.DirectionArrived != null)

this.DirectionArrived(this.ConveyorDirection);
                    }
                    else   if   (Code.Trim().ToString()   ==   Direc-
tion.Backward.ToString())
                    {
                        this.ConveyorDirection = Direction.Backward;
                        if (this.DirectionArrived != null)

this.DirectionArrived(this.ConveyorDirection);
                    }
                    i++;
                }
                else if (Code == "Speed" || Code == "SPEED")
                {
                    Code = aList[i + 1].ToString();
                    if(this.SpeedArrived != null)
                        this.SpeedArrived(Convert.ToInt16(Code));
                    i++;

                }
                else if (Code == "Stop" || Code == "stop")
                {
                    if(this.StopConveyor != null)
                        this.StopConveyor();

                }
            }

            if (this.StatusArrived != null)
                this.StatusArrived(ConveyorStatus.Off);

        }
```

Table 6.33 Summary of Conveyor PortAccess

Class summary			
Class ID: CC1		Namespace: ConveyorEngine	
Class name: PortAccess		Class type: Concrete	
Is base class? No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Data type	Access modifier	Description
_Speed	short	Private	Holds the value to control spindle speed
_Direction	Direction	Private	Holds the direction of the machine
xPulses	int	Private	Sets the pulse of the machine in X axis
yPulses	int	Private	Sets the pulse of the machine in Y axis
EmergencyStop	static bool	Private	Holds the Boolean value to control the machine running status.
Method name	Return type	Access modifier	Description
Delay()	Void	Public	Function to cause a delay of defined no. of cycles
PulsRequestX(double X)	Void	Public	Function to set the xPulses
PulsRequestY(double Y)	Void	Public	Function to set the yPulses
Xplus()	Void	Public	Function to increase the xPulses
Xminus()	Void	Public	Function to decrease the xPulses
Yplus()	Void	Public	Function to increase the yPulses
Yminus()	Void	Public	Function to decrease the yPulses
ON()	Void	Public	Function to start the conveyor machine
OFF()	Void	Public	Function to stop the conveyor machine

Table 6.34 Summary of Conveyor class

Class summary			
Class ID: CC2	Namespace: ConveyorEngine		
Class name: Conveyor	Class type: Concrete		
Is base class? No	No. of inherited classes: 0		
Inherited from: IDisposable	Interfaces with: None		
Field name	Data type	Access modifier	Description
sFileName	static string	public	Holds the File Name of.cnc file
aList	static ArrayList	private	Holds the GM Codes extracted from.cnc file.
CodeList	static ArrayList	private	Hold All Codes
disposed	bool	private	Track whether Dispose has been called.
component	Component	Private	Holds other managed resource this class uses.
disposed	bool	Private	Track whether Dispose has been called.
Port	PortAccess	Private	Used to call the port functions for sending GM codes.
status	ConveyorStatus	Private	Holds and sets the machine status
ConveyorDirection	Direction	private	Holds the direction for the conveyor machine
sMachineName	string	Private	Holds the machine name
Method Name	Return Type	Access Modifier	Description
Dispose()	Void	Private	Function to dispose the rendering
EmergencyStop()	Void	Public	Function to stop the rendering if Emergency Stop is pressed
OpenFile(TextBox textBox)	Void	Public	Function used for open cnc file and display in textbox object based in this function

(continued)

Table 6.34 (continued)

Method Name	Return Type	Access Modifier	Description
WriteError(string sErrorMessage, string sErrorSource, string sErrorNumber, string sFunctionThatGenerateError)	static void	public	Function to write errors in error log
ExecuteFile()	Void	Public	This function takes commands from local array aList one by one and execute.
ValidateCode(string Code, ref bool isValid)	Void	Public	This Function will validate the GMCode
ValidateFile()	bool	Public	This Function will validate the cnc part program file. It will take file name from static variable set by calling routine. If cnc part program file is valid true will be return else false.
SaveFile(string sFile)	Void	Public	This Function will Save the Part program file. It will take file name from static variable set by calling routine.

Event name	Delegate	Access modifier	Description
StopConveyor	StopHandler	Public	Event to handle the stop message arrived
StatusArrived	StatusHandler	Public	Event to handle the conveyor status arrived
StartConveyor	StartHandler	Public	Event to handle the start message arrived
SpeedArrived	SpeedHandler	Public	Event to handle the speed
DirectionArrived	DirectionHandler	Public	Event to handle direction

Table 6.35 Summary of Conveyor3D class

Class summary			
Class ID: CC3		Namespace: CoreAnimation	
Class name: Conveyor3D		Class type: Concrete	
Is base class? No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Data type	Access modifier	Description
TV	TVEngine	public	Declaring the TrueVision Engine object
Inp	TVInputEngine	private	Declaring the Input Engine variable
TVLightEngine	TVLightEngine	private	Declaring the Light Engine variable
MatFactory	TVMaterialFactory	Private	Declaring the Material Factory variable
TVScene	TVScene	Private	Declaring the TVScene object
FactoryFloor	TVMeshClass	private	Declaring the Factory Floor mesh object
global	TVGlobals	Private	Declaring the Global variable
Light	D3DLIGHT8	Private	Declaring the light variable
sngPositionX	float	Private	Holds Object position in X axis for keyboard input
sngPositionY	float	Private	Holds Object position in Y axis for keyboard input
sngPositionZ	float	Private	Holds Object position in Z axis for keyboard input
snglookatX	float	Private	Holds Camera position in X axis for keyboard input
snglookatY	float	Private	Holds Camera position in Y axis for keyboard input
snglookatZ	float	Private	Holds Camera position in Z axis for keyboard input
tmpMouseX	int	Private	Holds Mouse input position in X axis
tmpMouseY	int	Private	Holds Mouse input position in Y axis
tmpMouseB1	short	Private	Holds Mouse input button1
tmpMouseB2	short	Private	Holds Mouse input button2
tmpMouseB3	short	Private	Holds Mouse input button3
tmpMouseScrollOld	int	Private	Holds Mouse input scroll old value
tmpMouseScrollNew	int	Private	Holds Mouse input scroll new value

(continued)

Table 6.35 (continued)

Field name	Data type	Access modifier	Description
sngAngleX	float	Private	Holds Mouse input angle in X axis
sngAngleY	float	Private	Holds Mouse input angle in Y axis
sngWalk	float	Private	For smooth movement while camera is walking
sngStrafe	float	Private	For smooth movement while camera is strafing
DoLoop	bool	Private	Holds the Boolean value true until the scene is rendering
ModelPath	string	Private	Variable for storing model paths
floor_x	float	Private	Holds Initial position for the floor in X axis
floor_y	float	Private	Holds Initial position for the floor in Y axis
floor_z	float	Private	Holds Initial position for the floor in Z axis
MeshConveyor	TVMesh	Private	Holds the mesh for machine
ConveyorDirection	string	Private	Direction of Conveyor Belt to display
ConType	ConveyorType	Private	Type of Conveyor Belt
SpeedControl	D3DVECTOR	Private	Holds the 3D vector of speed in X, Y, Z directions
ConveyorPosition	D3DVECTOR	Private	Holds Initial position for the machine
ConveyorStraightX	float	Private	Holds Initial position for the straight conveyor in X axis
ConveyorStraightY	float	Private	Holds Initial position for the straight conveyor in Y axis
ConveyorStraightZ	float	Private	Holds Initial position for the straight conveyor in Z axis
ConveyorRoundX	float	Private	Holds Initial position for the round conveyor in X axis
ConveyorRoundY	float	Private	Holds Initial position for the round conveyor in Y axis
ConveyorRoundZ	float	Private	Holds Initial position for the round conveyor in Z axis

(continued)

Table 6.35 (continued)

Method name	Return type	Access modifier	Description
InitializeComponent(System.Windows.Forms. PictureBox ToRender)	Void	private	Function to initialize all variables of this class
StartConveyor(string TimeUnit, double Time)	Void	Public	Function to move the conveyor in a specified time slot
StartRendering()	Void	Public	Function to Start rendering the 3D scene in a new thread
StopRendering()	Void	Public	Function to Set the DoLoop variable to false
Check_Movement()	Void	Public	Function to Check keyboard and mouse variables and adjust the camera
Main_Loop()	Void	Public	Function to check input and render the scene while DoLoop is true
Check_Input()	Void	Public	Function to Check keyboard and mouse input and adjust the camera view in 3D.
Rendering()	Void	Public	Function to display everything that we have rendered to the screen.

Table 6.36 Summary of Conveyor Belt Sequence Diagram

Sequence diagram summary				
Sequence ID: SD-G1 Sequence name: Conveyor Sequence Diagram				
Method name	Type	Description	Source	Destination
Open()	Message	Function to Load Part Program	User	MainForm
SetFilePath()	Return Message	Function to set the file path to open file on disk	MainForm	Conveyor
Save()	Message	Function to Save Part Program	User	MainForm
ValidateCode()	Self Message	Function to validate GM codes	Conveyor	Conveyor
SaveFile()	Message	Function to save file on disk	MainForm	Conveyor
New()	Self Message	Function to New Part Program	User	MainForm
Run()	Message	Function to Execute Part Program	User	MainForm
ValidateFile()	Return Message	Function to validate program file	MainForm	Conveyor
ExecuteFile()	Return Message	Function to execute program file	MainForm	Conveyor
Dispose()	Message	Function to close rendering and exit	MainForm	Conveyor
Help()	Message	Function to open the help manual	User	MainForm
Print()	Message	Function to print the program	User	MainForm
Move()	Message	Function to move the Conveyor	MainForm	Conveyor3D
SetSpeed()	Message	Function to set the Conveyor speed	MainForm	Conveyor3D
StopRendering()	Message	Function to stop rendering the 3D scene	MainForm	Conveyor3D
UpdateStatus()	Return Message	Function to return message about status of Conveyor	MainForm	Conveyor
UpdateMessage()	Return Message	Function to return message as string	MainForm	Conveyor
StartConveyor()	Message	Function to start the conveyor machine	MainForm	Conveyor
Speed()	Message	Function to get the current speed	MainForm	Conveyor

Table 6.37 Summary of Conveyor Open Part Program use case

Use case summary
Use Case ID: UC-C1
Use Case Name: Open
Actors: User
Description: The user selects and loads part program through a file with "cnc" extension
Preconditions: Part program file is present on the system
Postconditions: Part program is loaded on the screen
System Parameters: Part program file
Success Scenario: Part program is loaded successfully
Alternative Scenario: Error in loading part program
Includes: Set File Path
Priority: High

Table 6.38 Summary of Conveyor New Part Program use case

Use Case Summary
Use Case ID: UC-C2
Use Case Name: New
Actors: User
Description: The user inputs a new part program on the screen
Preconditions: None
Postconditions: Part program is validated and saved
System parameters: Part program with GM codes
Success scenario: New part program file is created successfully
Alternative scenario: Error in validating part program or in saving file
Includes:
Priority: High

Table 6.39 Summary of Conveyor Save Part Program use case

Use Case Summary
Use Case ID: UC-C3
Use Case Name: Save
Actors: User
Description: The user inputs the part program on the screen and saves it to a file with "cnc" extension
Preconditions: None
Postconditions: Part program is saved on the file
System parameters: Part program with GM codes.
Success scenario: Part program is validated and saved successfully
Alternative scenario: Error in saving part program
Includes: Validate File
Priority: High

Table 6.40 Summary of Conveyor Run Part Program use case

Use Case Summary
Use Case ID: UC-C4
Use Case Name: Run
Actors: User
Description: The user executes the part program after loading it on the screen
Preconditions: Part program is loaded and validated
Postconditions: Part program executes and display the conveyor movement along with task bar description with machine movement in X, Y, and Z axes
System parameters: Part program with GM codes
Success scenario: Part program is executed successfully
Alternative scenario: Error in executing part program
Includes: Validate File, Execute File
Priority: High

Table 6.41 Summary of Conveyor processes flow chart

Flow chart summary	
Flow chart ID: FC-G1	
Flow chart name: ConveyorBelt Process Flow Chart	
No. of inputs: 5	
No. of processes: 16	
No. of outputs: 1	
No. of control decisions: 12	
Input name	Description
Save Program	Save button on screen
Run Program	Run button on screen
Load Program	Load button on screen
New Program	New button on screen
Exit Application	Exit button on screen
Process Name	Description
Load Part Program	Function to Load Part Program
Save Part Program	Function to Save Part Program
Open Dialog Box	Function to open file on disk
New Part Program	Function to New Part Program
Update User Interface	Function to update values on user interface screen
Speed	Function to set the speed of machine
End Application	Function to close rendering and exit
Save File	Function to save file on disk
Get conveyor Direction	Function to Get conveyor Direction
Set Straight Conveyor	Function to Set Straight Conveyor
Set Round Conveyor	Function to Set Round Conveyor
Get conveyor type	Function to Get conveyor type
Set conveyor direction to foreword	Function to set conveyor direction to foreword
Add hours in unit time	Function to add hours in unit time

(continued)

Table 6.41 (continued)

Process Name	Description
Add minutes in unit time	Function to add minutes in unit time
Add seconds in unit time	Function to add seconds in unit time
Play conveyor animation	Function to play conveyor animation
Stop conveyor animation	Function to stop conveyor animation
Validate File	Function to validate program file
Execute File	Function to Execute Part Program
All Codes Loop	Loop to check the GM code
Message Arrived	Event to check a message
Status Arrived	Event to check a status
Output Name	**Description**
3D Conveyor display	3D Conveyor animation on screen
Control Decision Name	Description
Validate Part Program	Check to validate GM codes
Select Part Program	Select the program file
Save Dialog Box	Save Dialog Box
Validated	Check if File is validated
Emergency Stop	Stop rendering and reset
File Already Exists	Check if file already exists on the specified path
GoTo	Move Conveyor to specified location
Straight Conveyor	Check whether to display round or straight conveyor
Time Unit Hours	Check to add hours in unit time
Time Unit Minutes	Check to add minutes in unit time
Time Unit Seconds	Check to add seconds in unit time
Foreword	Check if the direction of conveyor is set to forward

6.7 Interface Design for System Integration

Figures 6.22, 6.23, and 6.24 provide the hardware interfaces between the real and proxy system. Datasheets for the micro-controller and other important integrated circuits are provided in Appendix-IV.

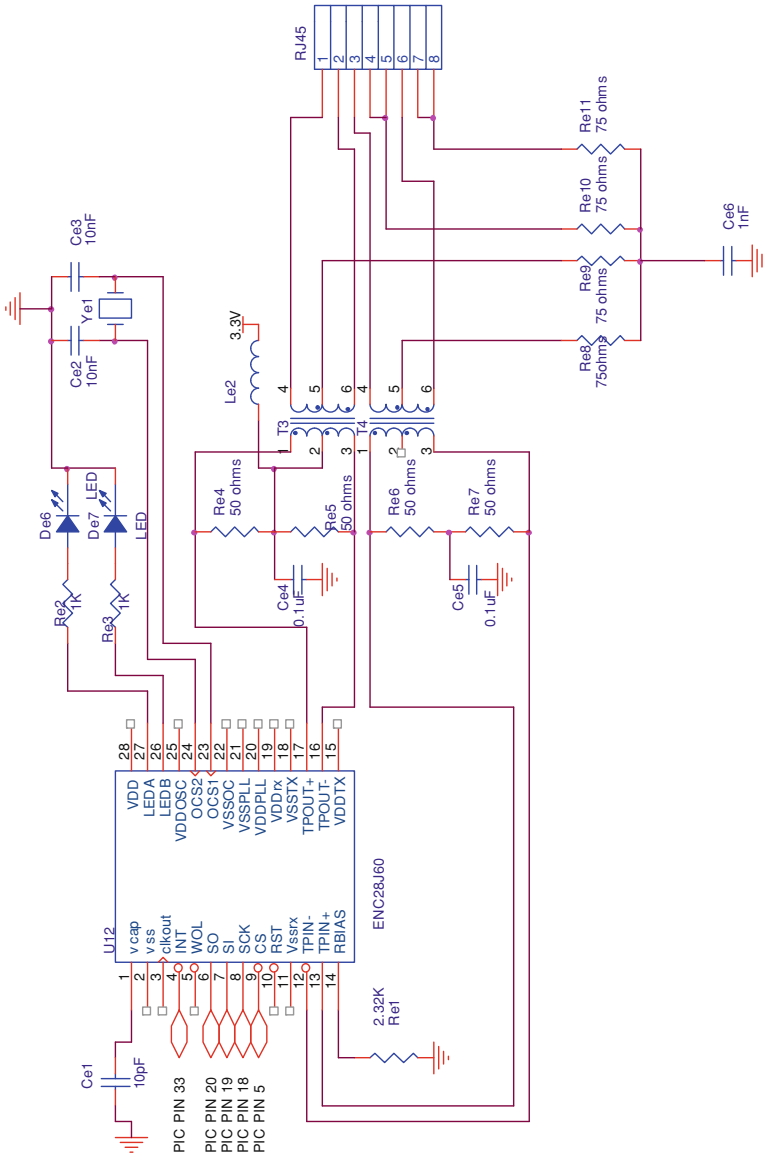


Fig. 6.22 Control of Industrial Manipulator through local area network (continued)

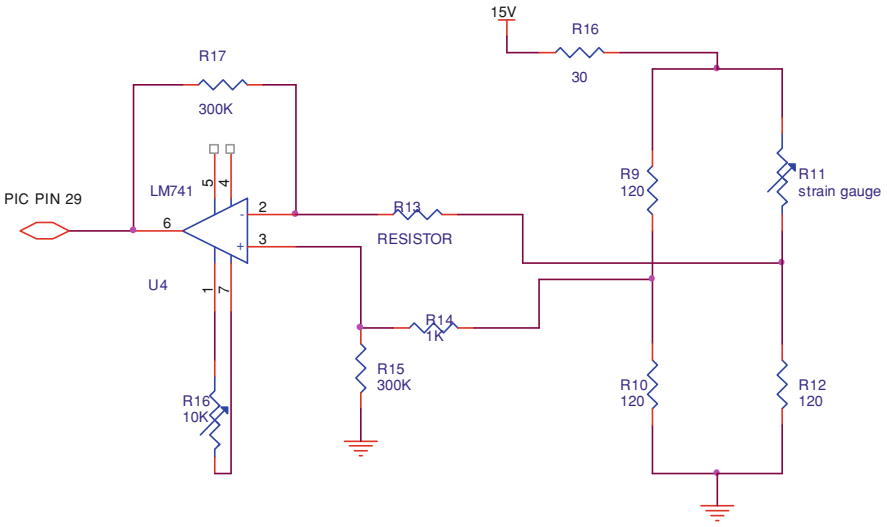


Fig. 6.22 (continued)

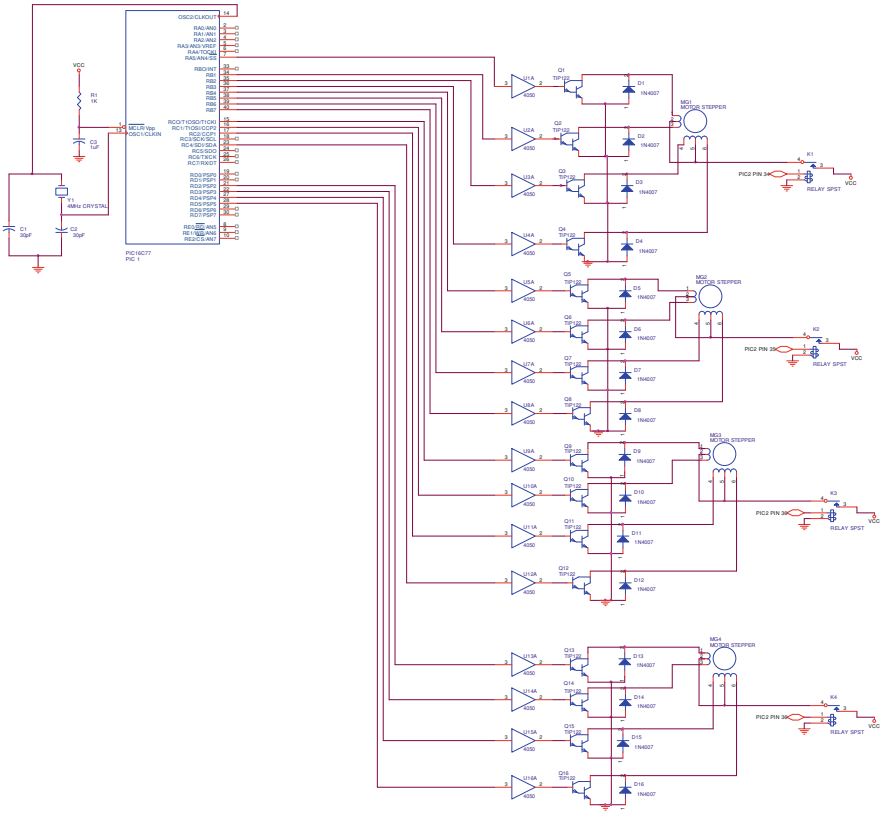


Fig. 6.22 (continued)

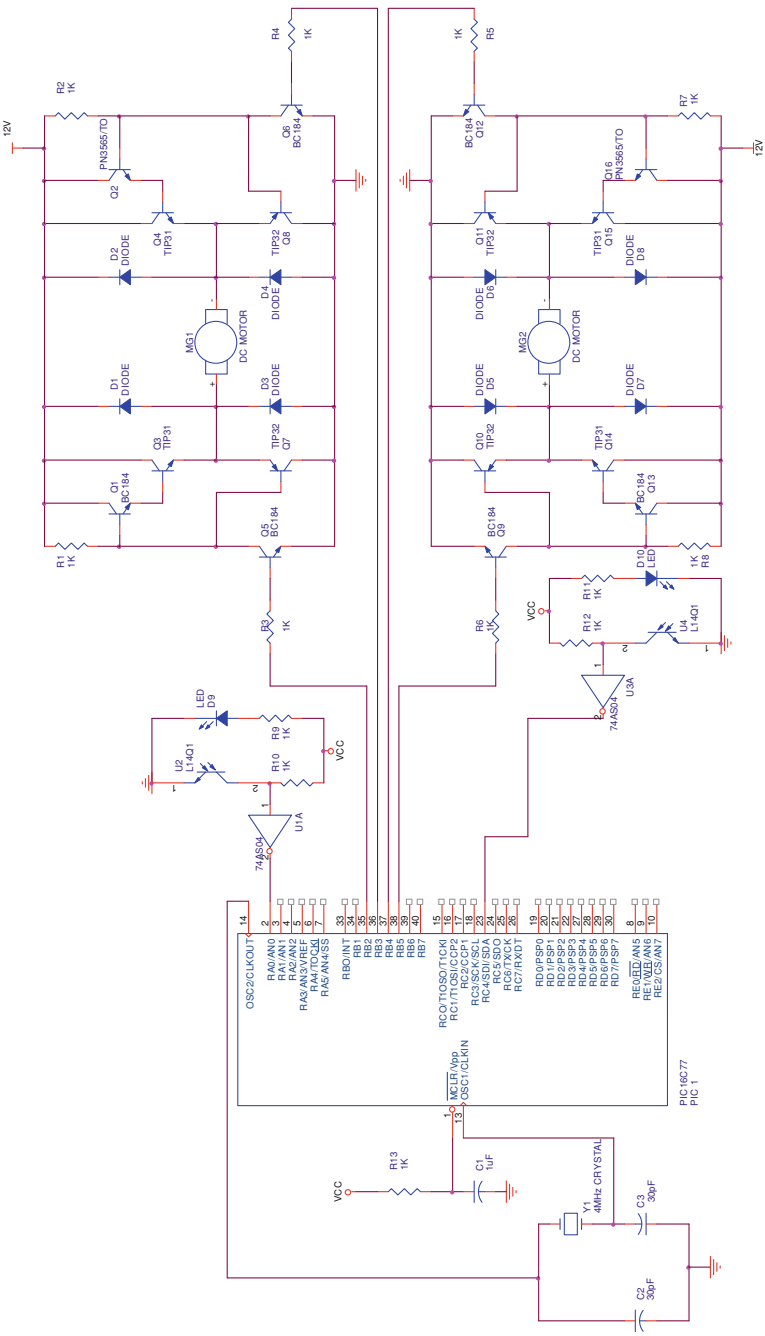


Fig. 6.23 Control of Conveyor belt through local area network

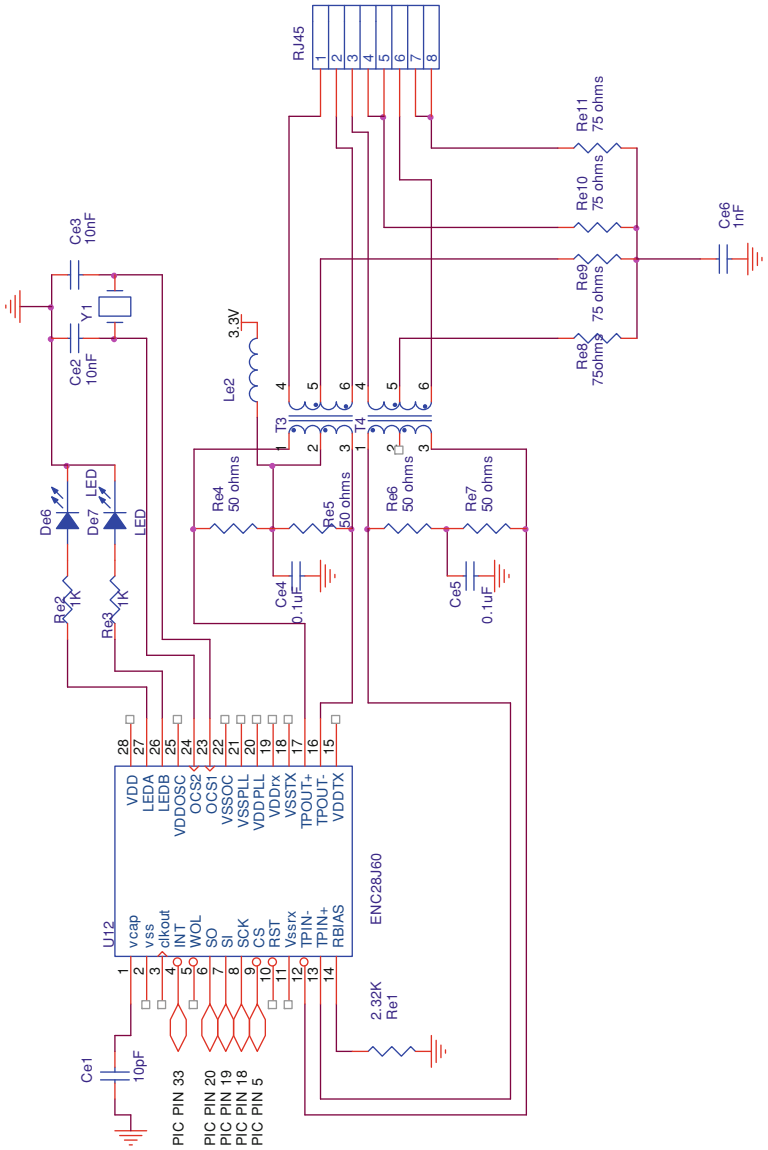


Fig. 6.23 (continued)

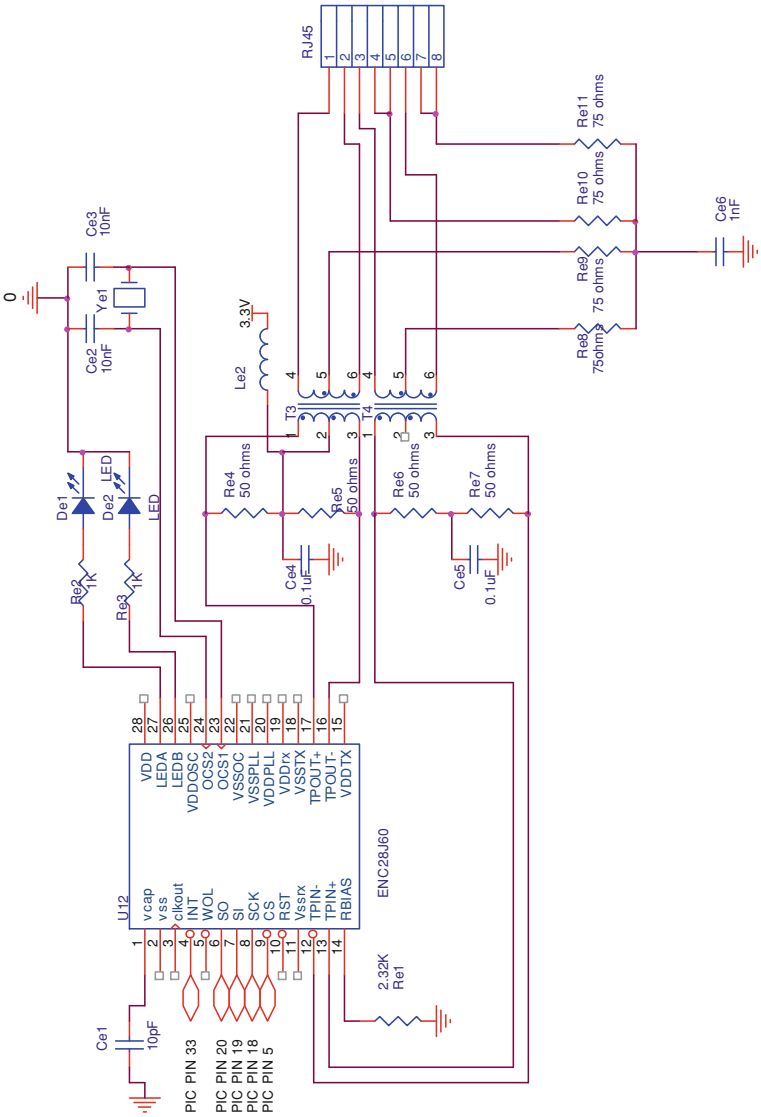


Fig. 6.24 Control of Gantry Crane through local area network (continued)

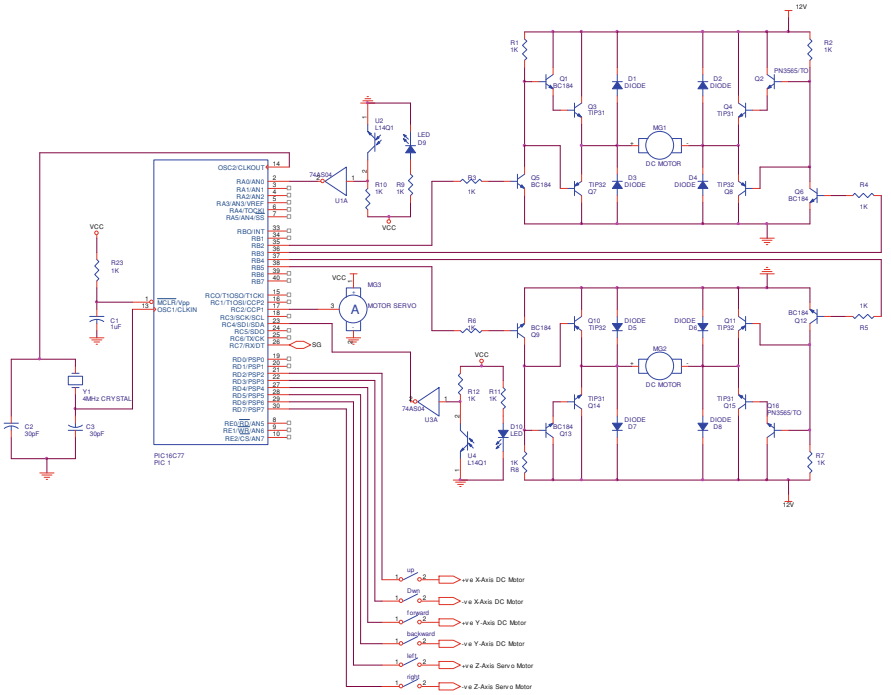


Fig. 6.24 (continued)

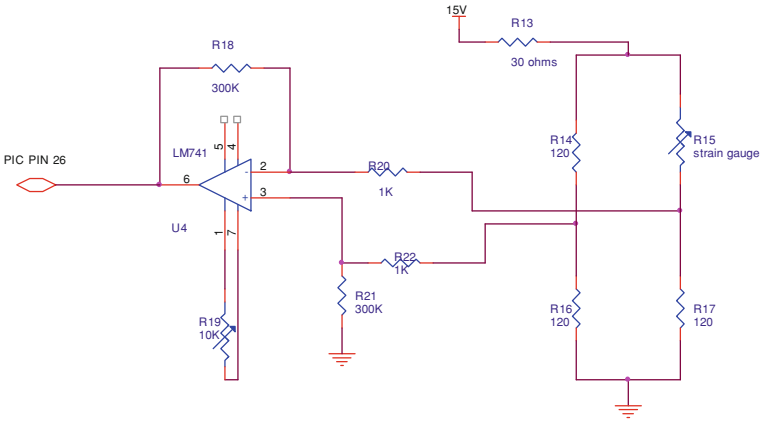


Fig. 6.24 (continued)

Bibliography

1. Baizid K (2009) Human multi-robots interaction with high virtual abstraction level. Lecture Notes in Computer Science 23–32
2. Belousov IR (2001) Virtual reality tools for Internet robotics. In: Proceedings of the IEEE international conference on robotics and automation 1878–1883
3. Bischoff R (2004) Perspectives on augmented reality based human-robot interaction with industrial robots. IEEE/RSJ international conference on intelligent robots and systems, 3226–3231
4. Coman M (2009) Design, simulation and control in virtual reality of a RV-24J robot. In: Proceedings—IECON, 35th annual conference of the IEEE industrial electronics Society
5. Du D (2008) Application of virtual design to belt conveyor. J Comput Inf Syst 4(4):1539–1545
6. Dvorak P (1997) Engineering puts virtual reality to work. Mach Des 69(4):69–73
7. Fang HC (2009) Robot programming using augmented reality. International conference on cyberworlds 13–20
8. Fiala M (2009) A robot control and augmented reality interface for multiple robots. In: Proceedings of the 2009 Canadian Conference on Computer and Robot Vision
9. Freund E (2001) Decentralized virtual reality: making the move to multi-user and multi-screen virtual worlds. IEEE International Conference on Intelligent Robots and Systems 1830–1835
10. Guoqian W (2006) Research on the framework of the crane's virtual design system and its key technologies. IEEE International Conference on Robotics and Biomimetics 1293–1298
11. Hu K (2009) Virtual prototyping of belt conveyor based on Recurdyn. Appl Mech Mater 16–19:776–780
12. http://en.wikipedia.org/wiki/Robot
13. http://en.wikipedia.org/wiki/Gantry_crane
14. http://en.wikipedia.org/wiki/Conveyor
15. Kang SC et al (2009) Numerical method to simulate and visualize detailed crane activities. Comput Aided Civ Infrastruct Eng 24(3):169–185
16. Knivett V (2006) The ideas conveyor belt. New Electron 14–16
17. Korlealaakso PM et al (2007) Development of a real-time simulation environment. Multibody Syst Dyn 17(2–3):177–194
18. Lai JY (1997) Development of a virtual simulation system for crane-operating training. Proceedings of 1997 ASME ASIA congress and exhibition, Singapore
19. Li C et al (2009) Research on 3D remote virtual monitoring for tower crane. Shanghai Jiaotong Daxue Xuebao 43(1)1689–1692
20. Or CKL et al (2009) Perception of safe robot idle time in virtual reality and real industrial environments. Int J Ind Ergonomics 39(5):807–812
21. Reinhart G (2007) A projection-based user interface for industrial robots. In: Proceedings of the 2007 IEEE international conference on virtual environments, human-computer interfaces and measurement system
22. Rocheleau DN (1991) Development of a graphical interface for robotic operation in a hazardous environment. In: Proceedings of the IEEE international conference on systems man and cybermetics, 1077–1081
23. Trebilcock B (2003) Virtual reality. Mod Mater Hand 58(5):55–60
24. Tu H (2005) A web-based remote machining cell. In: Proceedings of the IEEE international conference on machatronics, 953–958
25. Watanuki K et al (2007) Knowledge transfer of crane operating skill on creating a mold using portable VR systemand force feedback device. Transaction of the Japan society of mechanical engineers, 73(1):53–58
26. Wilson BH (1998) Virtual environment for training overhead crane operators: Real-time training. IIE Trans 30(7):589–595
27. Zhou Y (2007) Virtual reality simulation of humanoid robots. In: IECON proceedings of the 33rd annual conference of the IEEE industrial electronics society, 2772–2777

Chapter 7
Virtual Reality Design for Programmable Logic Controller Based Applications

7.1 Introduction

Programmable logic controller (PLC) remained and important control element initially in automobile industry and later in almost all categories of the industries. PLC was basically designed to replace relay circuits in automobile industry. This chapter details design of augmented reality for the PLC.

7.2 Programmable Logic Controller

Programmable logic controller is a device that was invented in 1969 to replace the sequential relay circuits for machine control systems. Control of some thermodynamic parameters using PLC is shown in Fig. 7.1. The PLC works by looking at its inputs and depending upon their state, turning ON/OFF its outputs. Input devices are switches, sensors, and transducers. Output devices can be solenoids, electromagnetic valves, motors, relays, magnetic starters as well as instruments for sound and light signalization. A program is stored in the PLC's non-volatile memory. The primary function of a PLC is to execute one or more sequence control programs in response to data from sensors and, in turn, update actuators. This primary function can be divided conceptually into three functions: sensor scanning, sequence program executing, and actuator updating.

7.3 Programming PLCs

IEC 1131-3 is the international standard that defines the programming language for PLCs. With the IEC-1131 standard, it is now possible to program these devices

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_7, © Springer-Verlag London Limited 2011

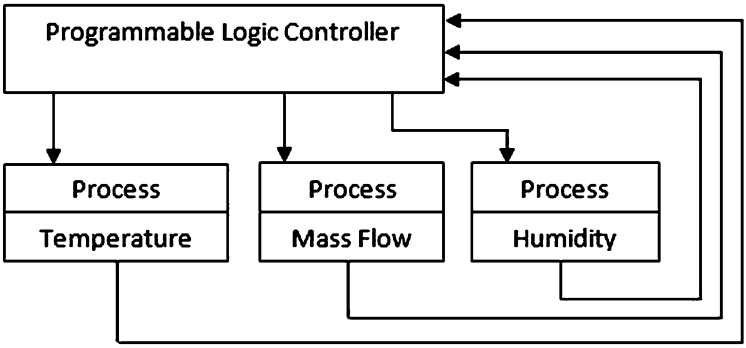


Fig. 7.1 Process parameter control using programmable logic controller

using structured programming languages (such as C#). IEC 1131 consists of five parts:

1. General information,
2. Equipment and test requirements,
3. PLC programming languages,
4. User guidelines,
5. Communications.

The third part of IEC 1131-3 which is called IEC 61131-3 specifies the syntax, semantics, and display for the following suite of PLC programming languages:

1. Ladder diagram (LD),
2. Sequential Function Charts,
3. Function Block Diagram,
4. Structured Text,
5. Instruction List.

IEC 61499, an international standard for distributed function blocks extends IEC 61131. One of the primary benefits of the standard is that it allows multiple languages to be used within the same programmable controller. This allows the program developer to select the language best suited to each particular task. In this PLC simulation application, C# dot NET is used to program PLC simulator.

7.3.1 Basic Instructions Adopted for PLC Simulation

A large number of instructions can be used at a PLC. Since it is not possible to present all these instructions here, therefore, a smaller selection of this instruction is presented. The selected instructions are capable of controlling large number of control situations:

7.3.1.1 Load

The load (LD) instruction is a normally open contact. Sometimes, it is also called examine if ON (XIO) (as in examine the input to see if it is physically ON). The symbol for a load instruction is shown below. The red "F" between the instructions tells that this instruction is currently OFF.

A Load (contact) symbol

Load instruction is used when an input signal is needed to be present for the contact to turn ON. When the physical input is ON, the instruction is True [9]. The PLC simulation provided in the videos examines the input for an ON signal. If the input is physically ON then the contact is ON. An ON condition is also referred to as logic 1 state.

In PLC simulation, this instruction is added by right clicking on LD workspace and by selecting "Add PLC Load." This symbol can normally be used for internal inputs, external inputs, and external output contacts by just setting the property.

7.3.1.2 LoadBar

The LoadBar instruction is a "normally closed contact." Sometimes, it is also called LoaDNot or examine if closed (XIC) (as in examine the input to see if it is physically closed).The symbol for a LoadBar instruction is shown below. The green "T" between the instructions tells that this instruction is currently ON.


A LoaDNot (normally closed contact) symbol

LoadBar is used when an input signal does not need to be present for the symbol to turn ON. When the physical input is OFF the instruction is True. The input is examined for an OFF signal. If the input is physically OFF then the symbol is ON. An OFF condition is also referred to as a logic 0 state.

This symbol is normally used for internal inputs, external inputs, and sometimes, external output contacts. It is the exact opposite of the Load instruction. With most PLCs, this instruction (Load or Loadbar) must be the first symbol on the left of the ladder.

7.3.1.3 Out

The Out instruction is also called an OutputEnergize instruction. The output instruction is like a relay coil. Its symbol looks as shown below.

An OUT (coil) symbol

When there is a path of True instructions preceding this on the ladder rung, it will also be True. When the instruction is True it is physically ON. This instruction is normally open output. This instruction can be used for internal coils and external outputs.

7.3.1.4 Outbar

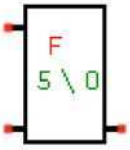
The Outbar instruction is also called an OutNot instruction. Some vendors do not have this instruction. The Outbar instruction is like a normally closed relay coil. Its symbol is shown below.

An Outbar (normally closed coil) symbol

When there is a path of False instructions preceding Outbar on the ladder rung, it will be True. When the instruction is True, it is physically ON. This instruction is a normally closed output. This instruction can be used for internal coils and external outputs. It is the exact opposite of the Out instruction.

7.3.1.5 Counter

Counter instruction is for counting till specified value before opening the contact. Its symbol is shown below.



A counter symbol

PLC simulation's counter is an Up counter and has three contacts. Left upper contact is for resetting the counter and left down contact is for sending ON instructions. Each ON instruction for left down contact will increase counter by 1. Third contact is right contact. It will be open when the counter's counting will be complete. Red "F" is showing that currently counter is in OFF condition.

7.3.1.6 Timer

Programmable logic controller simulation's timer instruction is On-Delay timer. This type of timer simply "delays turning ON." In other words, after input turns ON it waits x-seconds before turning ON its output. Its symbol is shown below.

A Timer Symbol

7.4 Proxy HCI for PLC-Based Processes

It is envisaged that the computer and communication technologies have reached a level whereby the realization of a virtual manufacturing organization is possible. This shall lead to the availability of micro and macro decision variables to the mangers allowing optimum decision making for higher productivity and profitability.

This chapter considers PLCs. The simulated processes should have the capability to mimic real manufacturing processes and are configured on a simulated factory layout. It is also possible to configure each of the tangible and intangible operations in variable quantities and sizes depicting a real factory. The manufacturing system and the manufacturing processes chosen in the virtual domain are reconfigurable thus allowing infinite possibilities for the development of virtual manufacturing organization. The augmented reality for PLC acts as a proxy to the real PLC operating a process and provides real-time status of the process.

7.5 Development of PLC Simulator Using Object-Oriented Design

The object oriented design (OOD) logic for developing real and proxy PLC simulator diagram, which are essentially the same, is divided into five distinct classes. These classes are:

(a) PLC Base Class;
(b) PLC Base Inherited Classes;
(c) Ladder Class Showing Methods, Properties, and Events;
(d) Communication and Connectors Classes;
(e) Serialization (Save Project) Class.

The classes of PLC simulation application are design by keeping object-oriented programming paradigm in mind. All the functionality of PLC simulation is encapsulated in classes and can be utilized in application user interface. Visual components such as Ladder in PLC simulation application which is object of MainLadder class, is inherited by system. Windows.Forms.Control class, is the main object in which user can add rung and PLC instruction to complete LD and all PLC instructions such as Timer, Counter, Relay, Coil, Inputs, Outputs, and Power rails. These are objects of related classes inherited by PLCBase class which is also inherited by Entry class.

In user interface or presentation layer, developer does not have direct access to PLC instructions, instead user just calls appropriate function of ladder to add, remove, and count PLC instructions in order to scan LD.

The above classes and their key components are detailed in class diagrams presented as Figs. 7.2, 7.3, 7.4, 7.5, and 7.6. Figure 7.7 provides the UML sequence diagram for the PLC Control Add system and Fig. 7.9 shows the UML sequence diagram for PLC Connection Add system. Use case diagrams for system run are shown in Fig. 7.10. Figure 7.11 presents use case diagrams Add PLC Control. Figure 7.12 shows use case diagram for Add Connection. Figure 7.13 provides the flow chart for PLC simulation software. The OOD depicted through Figs. 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, and 7.13 gives complete detail for the embedded technique. PLC simulation solution consists of two projects.

(1) PLC Control Library is consist of classes used in PLC simulation software such as Ladder class, all PLC instruction classes, Connection class, Connector class, Serialization class, two interfaces, and four enumerations. PLC Control Library is a Dynamic Link Library (DLL) based project. The produced DLL of PLC Control Library will be use in PLC simulation software to provide separation between application and logic and also logic encapsulation.

Class structure of PLC Control Library comprises two main classes: PLCBase class which is the parent class of all PLC instruction classes and MainLadder class which provide PLC Ladder user control and interface to add PLC instruction objects to complete LD. A request from PLC Scanner for PLC instruction objects are also handles by MainLadder Class. The enumerations of PLC Control Library are ControlFunctionType used to identify the PLC instruction function and it is implemented in PLCBase class by interface IControlFunctionType, PLCControlType is used to identify PLC instruction type, TimerTypes is identify PLC timer type, and PowerRailTypes which identify Power Rail of PLC Ladder.

In order to provide link between two PLC instructions and between power rail, PLC Control Library has Connection class which provide link between two controls with the help of Connector class. Each PLC instruction and power rail implemented Connector object is stored in ConnectorCollection to link between the Connection objects.

In order to facilitate request from PLC Scanner, PLC Control Library also has PLCControlCollection class which is implemented in MainLadder to store each PLC instruction object, connection object, and power rail object. When PLC Control is added in PLC Ladder, it automatically added in PLCControlCollection as well for scanning the Ladder at a later time.

PLC instruction classes are NOInput, NCInput, NORelayContact, InternalRelayCoil, Output, NCRelayContact, PLCPowerRail, PLCTimer, and PLCCounter class, all are child classes of PLCBase class.

In order to save the LD in a file for future use PLC Control Library also has SaveLadder Class which holds all PLC instructions, power rails, and all connections between controls in LD. SaveLadder object serialized in Simple Object Accesses Protocol format. Each PLC Control Type has separate

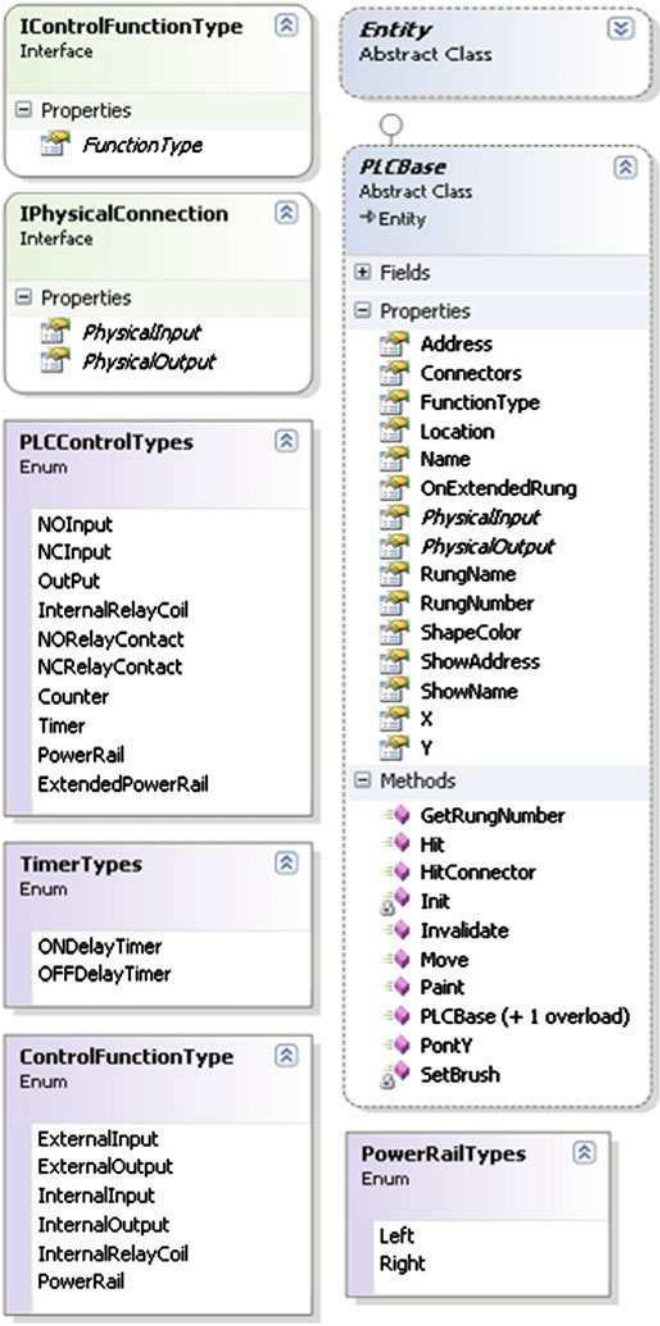


Fig. 7.2 PLC Base Class and enumerations

Fig. 7.3 PLC Base Inherited Classes

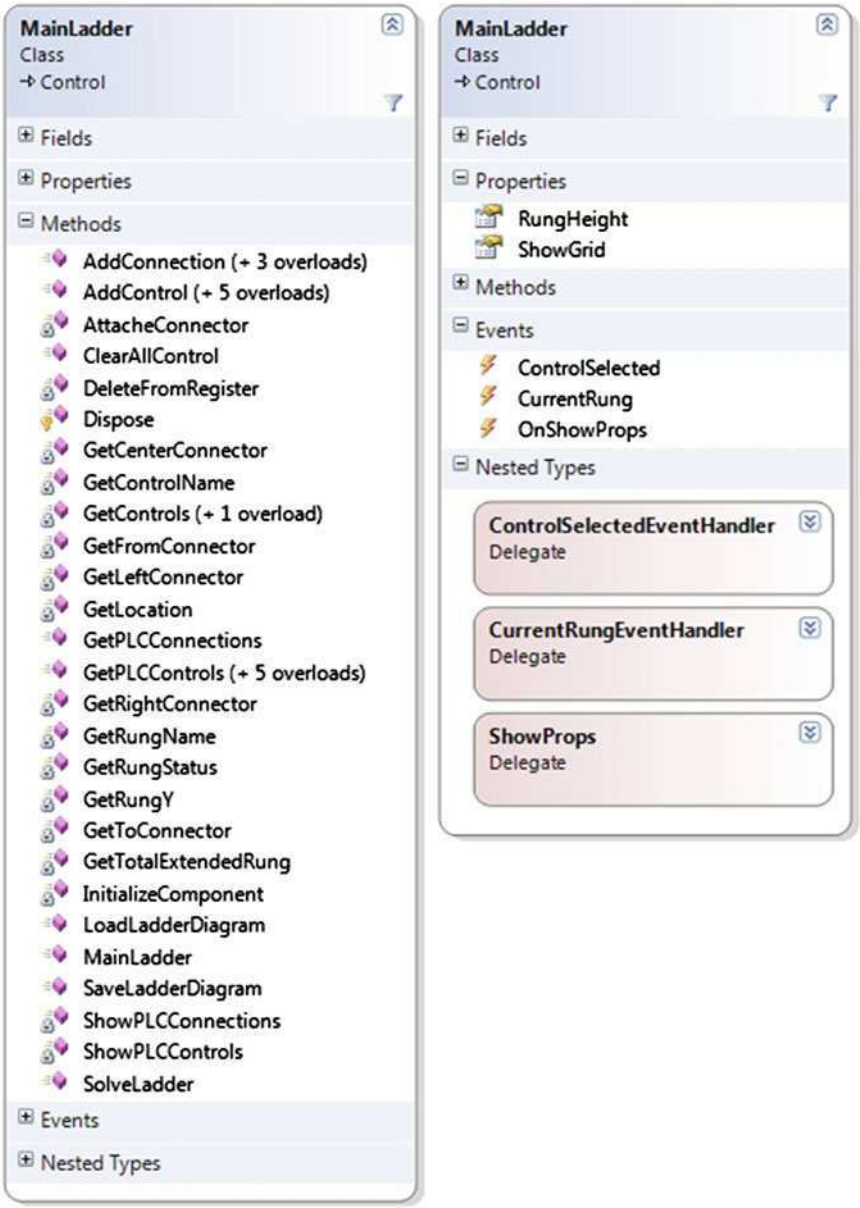


Fig. 7.4 Ladder Class Showing Methods, Properties, and Events

serialized class to facilitate SaveLadder class such as PLCSerializConnection class, PLCSerializControl class, PLCSerializCounter class, PLCSerializPowerRail class, and PLCSerializTimer class.

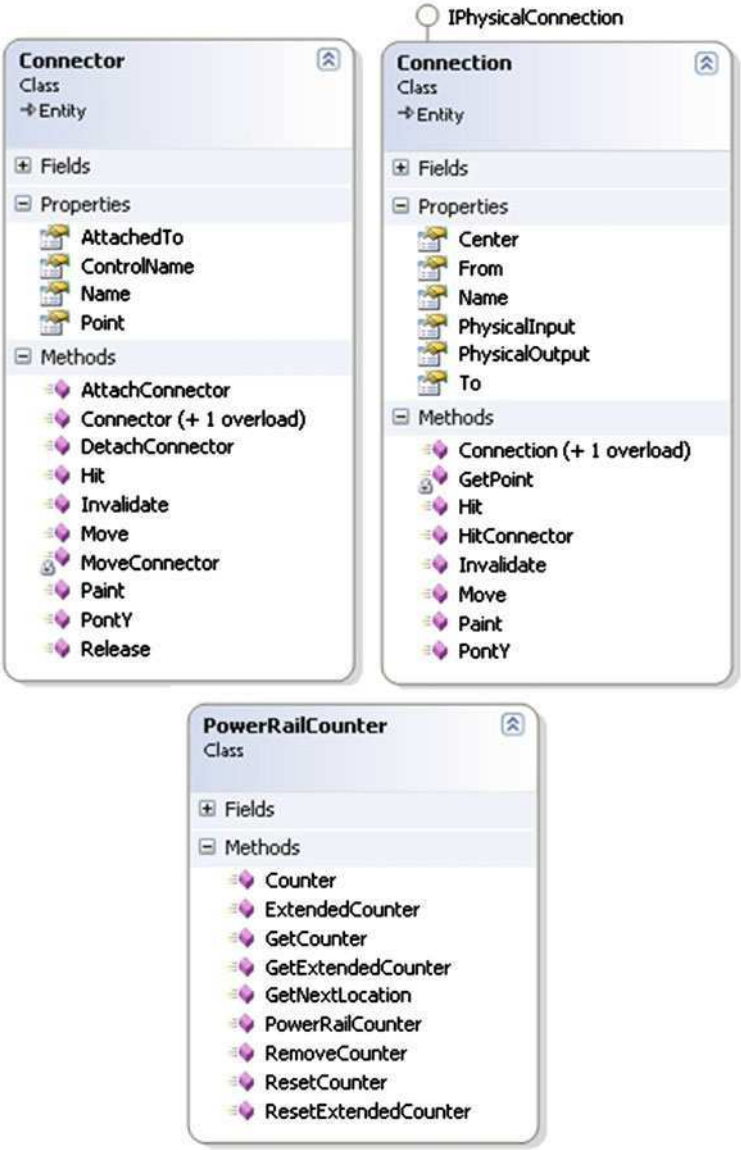


Fig. 7.5 Connection and Connector Classes (drag and drop option)

(2) PLC simulation software consists of PLC application user interface that utilize PLC Control Library objects to simulate PLC. Ladder Analyzer and Ladder Scanner are also implemented in PLC simulation software.

Table 7.1 shows the PLCBase class member properties, methods, and variables. PLCBase inherited from Entity class is an abstract class containing the basic

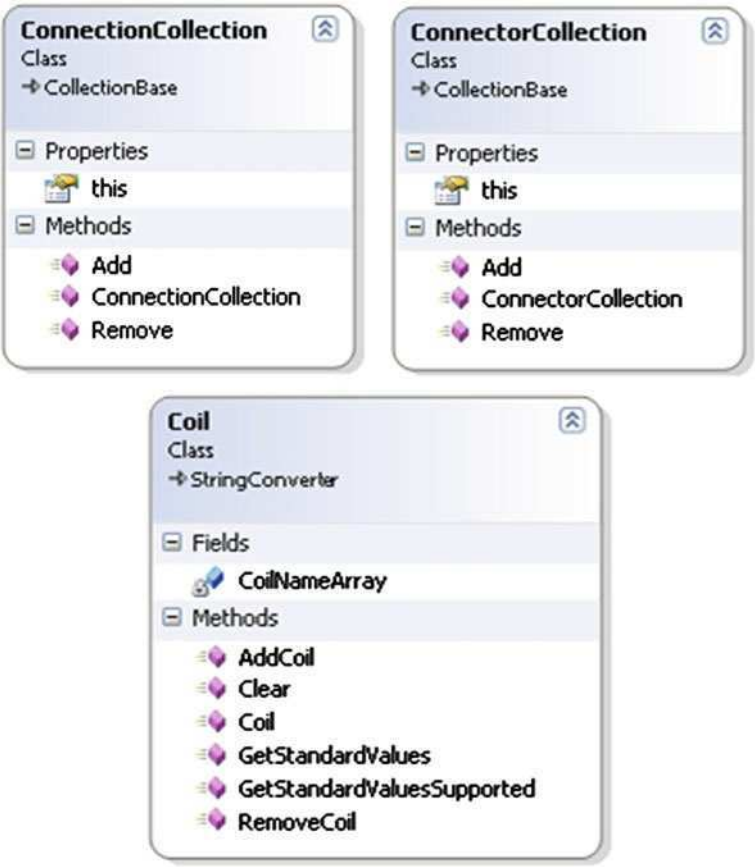


Fig. 7.5 (continued)

parameters of a PLC control and has functionality common to all PLC controls. Each PLC control (PLC Instruction) is inherited from PLCBase class.

Table 7.2 shows the members of PLCControlCollection class. PLCControlCollection is used to provide single location of all PLC controls on ladder. When PLC instruction is added on Ladder, it automatically insert into PLCControlCollection. PLC Scanner loop through each PLC control in PLCControlCollection to determine the state of PLC Control.

Table 7.3 shows the members of Entity class. Entity is the parent class of PLCBase class, Connector class, and Connection class. Entity has the responsibility for displaying the control on Ladder. Entity has functionality common to its child classes.

Table 7.4 shows the members of Coil class. Coil class is used to store coils from other PLC instruction objects. Coil object sets the status of PLC instruction ON or OFF.

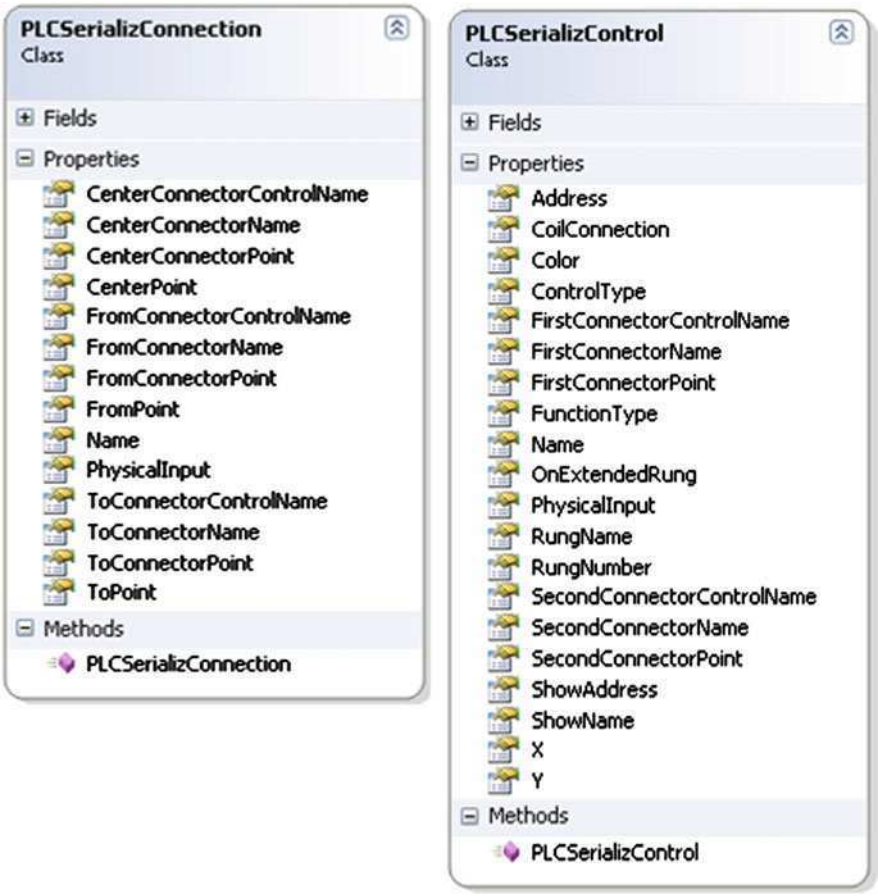


Fig. 7.6 Serialization (Save Project)

Table 7.5 shows the Connection class member variables and methods. Connection class is used to provide logical connection between two PLC instructions or PLC controls by using Connector object. Each PLC control has two or more Connector objects to receive and pass information signals from Connection object to other PLC control objects.

Table 7.6 depicts the members of ConnectionCollection class. ConnectionCollection class is used to holds PLC Connection objects. All Connection objects on LD is store in ConnectionCollection.

Table 7.7 displays the PLCCounter class member variables and methods. PLCCounter class represents the PLC Up Counter instruction. PLCCounter provides the internal input to LD after some predefine counting. PLCCounter is the class that uses three Connector objects to communicate with other PLC instruction.

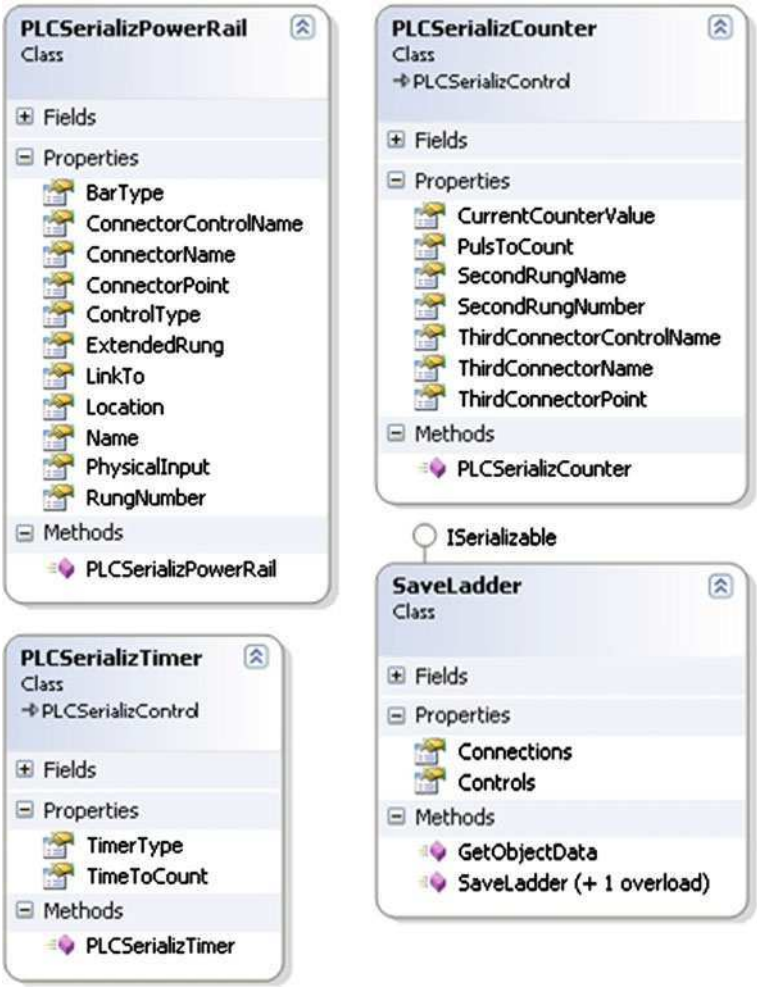


Fig. 7.6 (continued)

Table 7.8 shows the member variables and methods of InternalRelayCoil class. InternalRelayCoil class represents the PLC InternalRelayCoil instruction. InternalRelayCoil just passes signal information on its output Connector which was received from its input Connector.

Table 7.9 depicts the member variables and methods of NCInput class. NCInput class represents the PLC normally closed contact instruction or PLC Control. NCInput just passes opposite signal to its output Connector which was received from its input Connector.

Table 7.10 details the member variables and methods of NCRelayContact class. NCRelayContact class represents the PLC normally closed coil instruction or PLC

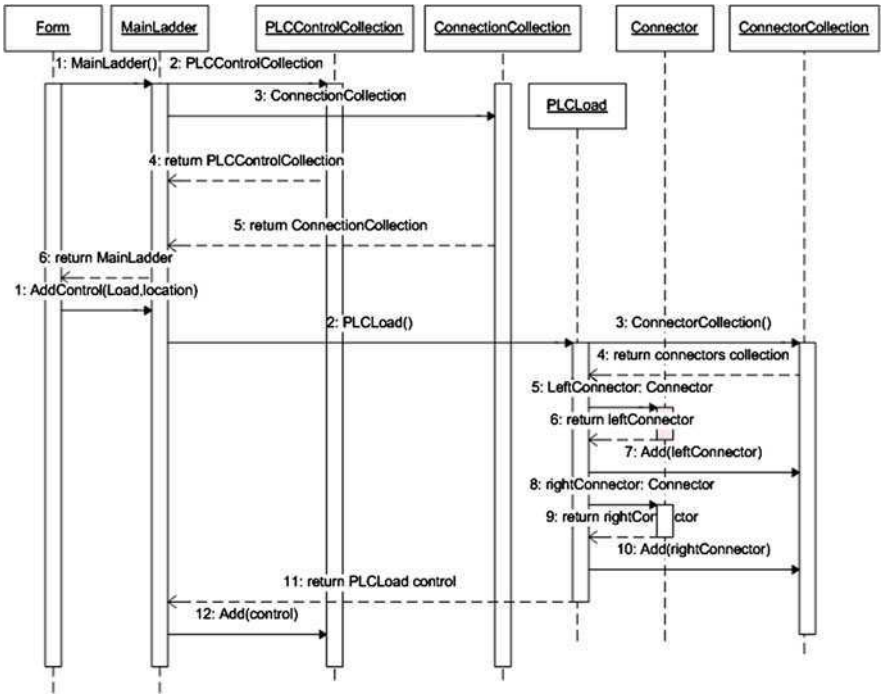


Fig. 7.7 PLC Simulation UML Sequence Diagram (PLC Control Add)

Control and its function type on Ladder is internal input. NCRelayContact just passes apposite signal to its output Connector which was received from its input Connector.

Table 7.11 is an illustration of the member variables and methods of NOInput class. NOInput class represents the PLC normally ON contact instruction or PLC Control and its function type on Ladder is external input. NOInput just passes signal to its output Connector which was receiving it from its input Connector.

Table 7.12 explains the member variables and methods of NORelayContact class. NOInput class represents the PLC normally ON coil instruction or PLC Control and its function type on Ladder is internal input. NORelayContact just passes signal to its output Connector which was receiving it from its input Connector.

Table 7.13 depicts the member variables and methods of Output class. Output class represents the PLC normally Output instruction or PLC Control and its function type on Ladder is external output. Output just passes signal to its output Connector which was receiving from its input Connector.

Table 7.14 shows the member variables and methods of PLCPowerRail class. PLCPowerRail class represents right and left rails of PLC ladder. PLCPowerRail also has two Connector objects, one for right side rail and one for left side rail for sending and receiving signal form PLC Instructions.

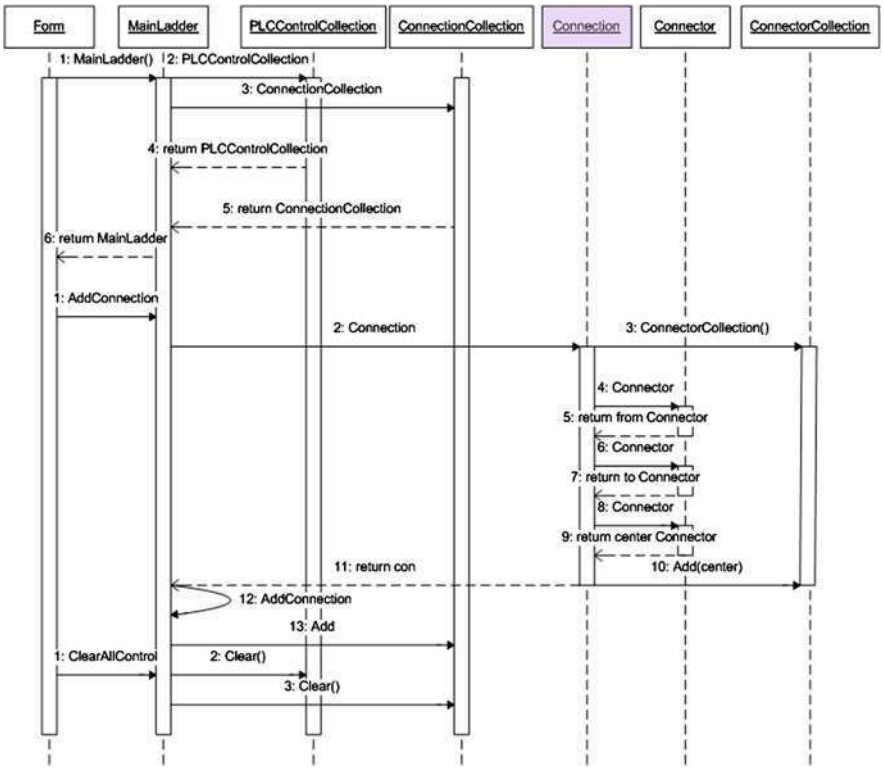


Fig. 7.8 PLC Simulation UML Sequence Diagram (PLC Connection Add)

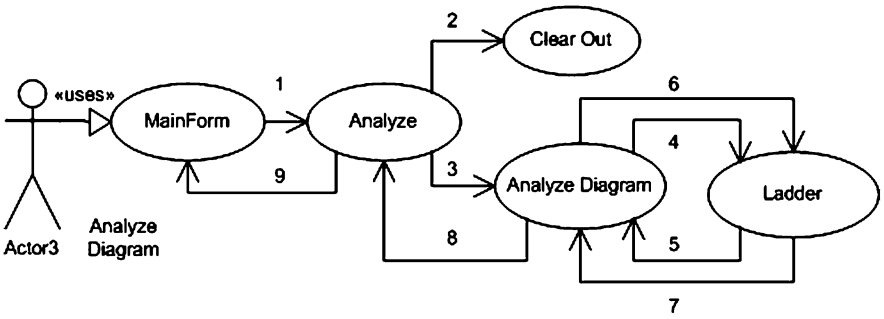


Fig. 7.9 Use Case Diagram for PLC system analysis

Table 7.15 shows the member variables and methods of PowerRailCounter class. PowerRailCounter class keeps track on number of rails PLC Ladder has. Each rung of Ladder has one pair of rail. PLC Scanner scan ladder by number of times provided by PowerRailCounter.

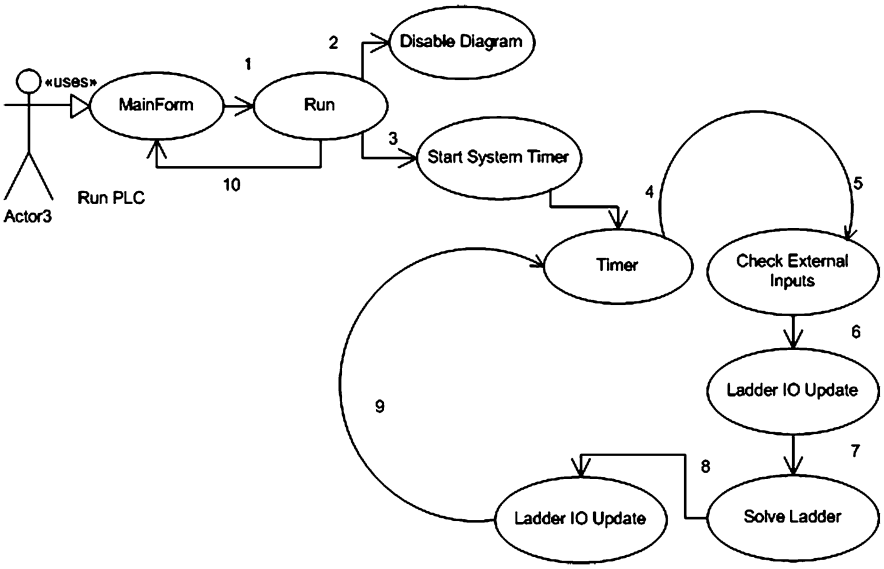


Fig. 7.10 Use Case Diagram for system run

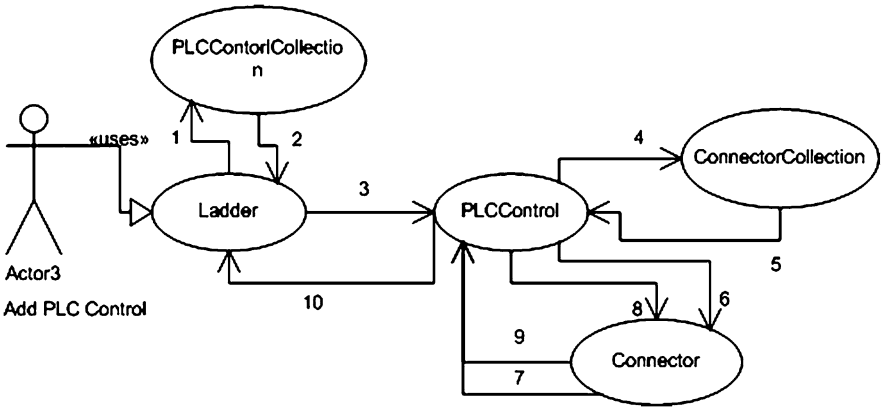


Fig. 7.11 Use Case Diagram for Add PLC Control

Table 7.16 lists the member variables and method of PLCSerializConnection class. PLCSerializConnection class is used to save Connection objects to physical file.

Table 7.17 defines the member variables and method of PLCSerializControl class. PLCSerializControl class is used to save PLC Control objects to physical file.

Table 7.18 presents the member variables and method of PLCSerializCounter class. PLCSerializCounter class is used to save PLC Counter instruction object to physical file.

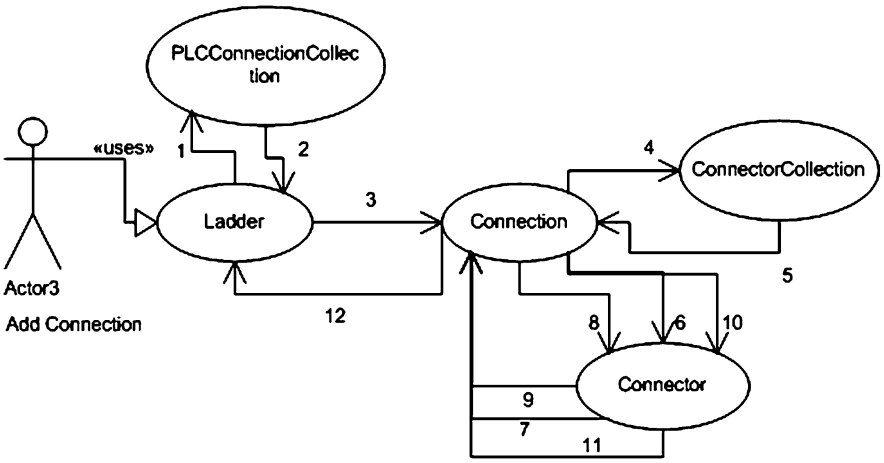


Fig. 7.12 Use Case Diagram for Add Connection

Table 7.19 details the member properties and method of PLCSerializPowerRail class. PLCSerializPowerRail is used to save rail objects of ladder to physical file.

Table 7.20 explains the member properties and method of PLCSerializTimer class. PLCSerializTimer is used to save PLC Timer instruction to physical file.

Table 7.21 presents the member properties and method of SaveLadder class. SaveLadder class is used to save full LD to physical file.

Table 7.22 specifies the PLCTimer class member variables and methods. PLCTimer class represents the PLC Counter instruction. PLCTimer provides the internal input to LD after some predefine timing and can be act as ONDelayTimer or OFFDelayTimer. PLCTimer is the class that uses two Connector objects to communicate with other PLC instruction.

Table 7.23 portrays the MainLadder class member variables and methods. MainLadder class represents the PLC LD user interface to create PLC ladder and it is inherited from Control class. MainLadder is the major class of PLC software.

Table 7.24 depicts the summary of PLC sequence diagram.

Table 7.25 illustrates the summary of Analyze Diagram use case. Analyze Diagram checks and validates each control and connection before ladder scanning begin.

Table 7.26 traces the summary of Run PLC use case. Run PLC is the scanning of PLC LD.

Table 7.27 characterize the summary of Add PLC Control use case.

Table 7.28 outlines the summary of Add Connection use case. Connection is required between two PLC instructions in order to pass signal information. Without Connection, analysis of ladder is fails and scanning process would not be able to start.

Table 7.29 sketches the summary of PLC processes flow charts (Figs. 7.14, 7.15, 7.16, and 7.17).

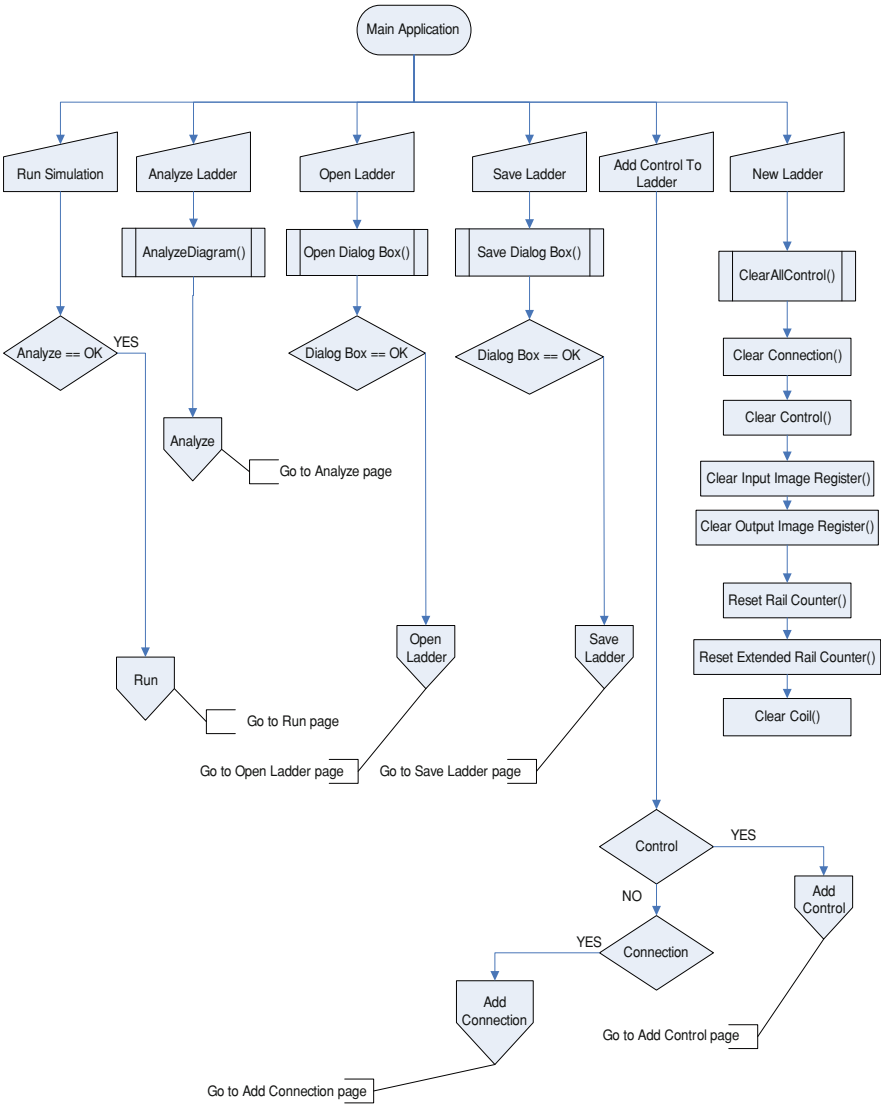
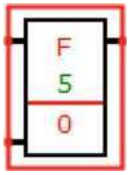
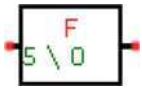


Fig. 7.13 Flow Chart for PLC Simulation Software

7.6 Programmable Logic Controller Simulation Software

Programmable logic controller simulation is windows based software that provides real time facility to create Ladder Diagram (LD) and simulate the functionality of PLC. Following figures provide a flow chart for the operation of this software.

Following PLC instructions have been chosen for the purpose of demonstrating the concept:

I	Load	
II	LoadBar	
III	Out	
IV	OutBar	
V	Counter	
VI	Timer	
VII	Relay	
VIII	Relay Contact	

Ladder Diagram Workspace window screen creates LD for simulation by right clicking on this space and by selecting appropriate instruction (Fig. 7.18).

Output Workspace is the place where current activity of simulation and information about analyzed process is displayed (Fig. 7.19).

Physical Input area allows setting up the status (ON/OFF) for instructions that has external Input property set.

In Fig. 7.20, the load 1 instruction is ON. Setting the instruction ON/OFF shall take effect during simulation is running.

Example given in Fig. 7.21 is used to demonstrate the working of the simulation software.

In circuit of Fig. 7.21, the coil will be energize only if both of SW1 and SW2 will be ON. In order to convert this real world circuit to LD following steps shall be taken:

1. Click on new LD icon on tool bar of PLC simulation.
2. Right click on LD workspace and select Add Rung, left and right power bars will appear.
3. Right click on LD workspace and select Add PLC Load, Load1 will be added to LD workspace.
4. Select newly added load1 by just clicking on it. After the selection of the load1 the property windows will show all properties of load1.
5. Rename it to SW1 by setting the Name property and select true for Show Address property.
6. Repeat the same for load2 or SW2.
7. Right click on LD workspace and select Add PLC Out, out1 will be added to LD workspace.
8. Rename out1 with coil and select true for Show Address property.

Now LD workspace looks like as shown in Fig. 7.22:

1. Two loads and one out instructions is added in the LD, now connections shall be added.
2. Right click on LD workspace and click on Add Connection, a line will appear that have two connectors one will be attached with mouse pointer, just drag this connector to any instruction's left connector and just drop on it. A little square green box will appear that confirms that the connection has been made successfully.
3. Now select second connector of connection and drag it on instruction's right connector and drop on it. A little square green box will appear that confirms the connection has been made successfully.

Now LD workspace looks like as shown in Fig. 7.23.

After successful development of LD, the PLC simulation can be activated by clicking on Run Simulation button. After running simulation, the LD workspace is disabled. The External Input's ON/OFF can be controlled from Physical Input as shown in Fig. 7.24. After running PLC simulation, do the following steps (Fig. 7.25):

In Physical Input, both SW1 and SW2 are OFF. In LD workspace both are OFF as well (Fig. 7.26).

Click on SW1 check box to switch it ON The new status will appear both on work space and message window (Figs. 7.27, 7.28, 7.29, 7.30, and 7.31).

Now both SW1 and SW2 are ON and coil is also switched ON because now it has full connection (Fig. 7.32).

More complex ladder logic can be constructed using the selected functions and by the addition of full set of ladder functions. Figure 7.29 demonstrates displays

Table 7.1 Summary of PLCBase class

Class summary			
Class ID: CP1			Namespace: PLCControlLibrary
Class name: PLCBase			Class type: Abstract
Is base class?: Yes			No. of inherited classes: 9
Inherited from: Entity			Interfaces with: IControlFunctionType

Field name	Data type	Access modifier	Description
rectangle	Rectangle	protected	This field holds the rectangle on which any control lives
shapeColor	Color	protected	This field holds the back color of the control
shapeBrush	Brush	[NonSerialized()] protected	This field holds the brush corresponding to the backcolor
connectors	ConnectorCollection	protected	This field holds the collection of connectors onto which you can attach a connection
sName	string	protected	This field holds the Name of the Control
bShowAddress	bool	protected	This field show the address of Control
bShowName	bool	protected	This field show the name of Control
sAddress	string	protected	This field holds the Address of the Control
iRungNumber	int	protected	This field holds the current rung number of Control
bOnExtendedRung	bool	protected	This field checks if Control is placed On ExtendedRung or MainRung
sRungName	string	Protected	This field holds Rung Name where the control currently located on
FunType	ControlFunctionType	protected	This field holds the FunctionType

Method name	Delegate	Access modifier	Description
GetRungNumber()	Static int	Public	Function to Return the current rung number
Hit(System.Drawing.Point p)	Bool	Public	Function to Override the abstract Hit method
HitConnector(Point p)	Connector	Public	Function to Return the connector hit by the mouse
Init()	void	private	Function to Summarize the initialization used by the constructors
Invalidate()	Void	Public	Function to Override the abstract Invalidate method
Move(Point p)	Void	Public	Function to Move the control with the given shift
Paint(System.Drawing.Graphics g)	Void	Public	Function to Override the abstract paint method

(continued)

Table 7.1 (continued)

Method name	Delegate	Access modifier	Description
PontY(int Y)	Void	Public	Function to Get the current value of Y access of ladder
SetBrush()	void	Public	Function to Set the brush corresponding to the backcolor
PLCBase()	–	Public	Default constructor for the class
PLCBase(MainLadder site): base(site)	–	Public	Special Constructor with the site of the control

Table 7.2 Summary of PLCControlCollection

Class summary			
Class ID: CP2			Namespace: PLCControlLibrary
Class name: PLCControlCollection			Class type: [Serializable] concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: CollectionBase			Interfaces with: None
Method name	Return type	Access modifier	Description
Add(PLCBase control)	int	public	Function to add the PLC control object
Remove(PLCBase control)	void	public	Function to remove the PLC control object
PLCControlCollection(){}	–	Public	Default constructor for the class

such a control. It shows that the analysis, single scan, and continuous scan can be performed (Fig. 7.33).

The PLC simulation software can be upgraded to full set of PLC ladder functions. It is designed to work as proxy to real PLC operation as a micro element of virtual manufacturing organization. Both software and electronics hardware links are required to make the virtual manufacturing organization operational for a real manufacturing set up. By adopting this strategy, micro elements of the decision parameters shall be in access of the concerned manager to enhance quality, productivity, and profitability.

7.7 A Section of Software Code

The purpose of the PLC simulator is to mimic operations at a particular process by a real PLC. PLC simulator is currently a part of VMS training. The current status of the variables at the process should match with that being depicted at the simulator. These parameters are passed to the management software for necessary decision making. A section of the simulator code that performs the analysis of the LD being developed is listed below.

The following partial code is taken from PLC software and consists of a method AnalyseDiagram. AnalyseDiagram method analyses LD for its design and configuration. In the case of any design issue or some missing information in connections then this method returns false and if every thing is configured properly than this method return true to its calling routine.

In the beginning of the following code, two arrays are created, one for PLC controls and one for connections. After creating these arrays, they are filled by MainLadder object's GetPLCControls method and GetPLConnections method. After filling arrays with Controls and Connections, returnValue is initialized with default value true and loop through each control object in ArraysControl array for checking control connector's connection. If PLC control, connector has AttachedTo property set to null than there is no connection with this PLC control.

In this case, returnValue is set to false, and error message is written to message output window. This process is repeated for each control and for each connector. After checking each control and connector, AnalyseDiagram method loop through each connection object for its connector's connection with any other PLC Control.

Section of PLC Simulator – Analyse Diagram

```
private bool AnalyseDiagram()
               {
                       ArrayList ArrayControls = new ArrayList();
                       ArrayList ArrayConnections = new ArrayList();

                       ArrayControls = mainLadder1.GetPLCControls();
                       ArrayConnections = mainLad-
der1.GetPLCConnections();

                       bool returnValue = true;

                       WriteOutput("Analysing controls....");
                       foreach(PLCControlLibrary.PLCBase controls in
ArrayControls)
                       {
                              #region check all plcbase controls
                              if(controls.ControlType == PLCControl-
Types.OutPut || controls.ControlType == PLCControlTypes.NOInput ||
controls.ControlType == PLCControlTypes.NCInput || con-
trols.ControlType == PLCControlTypes.Timer || controls.ControlType ==
PLCControlTypes.InternalRelayCoil || controls.ControlType == PLCCon-
trolTypes.NCRelayContact || controls.ControlType == PLCControl-
Types.NORelayContact)
                              {
                                     WriteOutput("Analysing " + con-
trols.Name);

       if(controls.Connectors[0].AttachedTo == null)
                                     {
                                            WriteOut-
put(controls.Name + "'s left connector is not connected to any con-
nection. Add new connection or re attache the connection with it");
                                            returnValue = false;

                                     }

       if(controls.Connectors[1].AttachedTo == null)
                                     {
                                            WriteOut-
put(controls.Name + "'s right connector is not connected to any con-
nection. Add new connection or re attache the connection with it");
                                            returnValue = false;

                                     }
                              }
                              #endregion
                              if(controls.ControlType == PLCControl-
Types.Counter)
                              {
                                     PLCCounter counter = (PLCCoun-
ter)controls;
                                     WriteOutput("Analysing " + coun-
ter.Name);

       if(counter.Connectors[0].AttachedTo == null)
```

```
                                        {
                                                WriteOut-
put(controls.Name + "'s left upper connector is not connected to any
connection. Add new connection or re attache the connection with
it");
                                                returnValue = false;
                                        }

        if(counter.Connectors[1].AttachedTo == null)
                                        {
                                                WriteOut-
put(controls.Name + "'s right connector is not connected to any con-
nection. Add new connection or re attache the connection with it");
                                                returnValue = false;

                                        }

        if(counter.Connectors[2].AttachedTo == null)
                                        {
                                                WriteOut-
put(controls.Name + "'s left down connector is not connected to any
connection. Add new connection or re attache the connection with
it");
                                                returnValue = false;

                                        }
                                }

                        }
                        #region check all connections
                        WriteOutput("Analysing connections....");
                        foreach(PLCControlLibrary.Connection connec-
tions in ArrayConnections)
                        {

                                WriteOutput("Analysing " + connec-
tions.Name);
                                if(connections.From.AttachedTo == null)
                                {
                                        WriteOutput(connections.Name +
"'s From connector is not connected to any PLC control. Add new PLC
control or re attache the connection with it");
                                        returnValue = false;

                                }
                                if(connections.To.AttachedTo == null)
                                {
                                        WriteOutput(connections.Name +
"'s To connector is not connected to any PLC control. Add new PLC
control or re attache the connection with it");
                                        returnValue = false;

                                }
                        }
                        #endregion

                        return returnValue;
                }
```

Table 7.3 Summary of PLC entity class

Class summary			
Class ID: CP3			Namespace: PLCControlLibrary
Class name: Entity			Class type: [Serializable] abstract
Is base class?: Yes			No. of inherited classes: 3
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
hovered	internal bool	protected	This field tells whether the current entity is hovered by the mouse
site	MainLadder	protected	This fields holds the control to which the entity belongs
isSelected	bool	protected	This field tells whether the entity is selected
PLCControlType	PLCControlTypes	protected	This field holds control type of PLCControl
font	Font	protected	This field holds Default font for drawing text
blackPen	Pen	[NonSerialized()] protected	This field holds Default black pen

Method name	Return type	Access modifier	Description
Paint(Graphics g)	abstract void	public	Function to Paint the entity on the control
Hit(Point p)	abstract void	public	Function to Test whether the control is hit by the mouse
Invalidate()	abstract void	public	Function to Invalidate the entity
PontY(int Y)	abstract void	public	Function to set the Current Point Y value
Move(Point p)	abstract void	public	Function to move the entity on the canvas
Entity()	–	Public	Default constructor for the class
Entity(MainLadder site)	–	Public	Special Constructor with the site of the entity

Table 7.4 Summary of PLC Coil class

Class summary			
Class ID: CP4			Namespace: PLCControlLibrary
Class name: Coil			Class type: [Serializable] abstract
Is base class?: No			No. of inherited classes: 0
Inherited from: StringConverter			Interfaces with: None
Field name	Data type	Access modifier	Description
CoilNameArray	ArrayList	private static	Array to hold the name of the coils

Method name	Return type	Access modifier	Description
GetStandardValuesSupported (ITypeDescriptorContext context)	bool	public	Function to get standard values supported
GetStandardValues(ITypeDescriptorContext context)	StandardValuesCollection	public	Function to get standard values
AddCoil(string CoilName)	static void	public	Function to add coil
RemoveCoil(string CoilName)	static void	public	Function to remove coil
Clear()	static void	public	Function to remove the entity on the canvas

Table 7.5 Summary of PLC Connection class

Class summary	
Class ID: CP7	Namespace: PLCControlLibrary
Class name: Connection	Class type: [Serializable]
Is base class?: No	No. of inherited classes: 0
Inherited from: Entity	Interfaces with: IPhysicalConnection

Field name	Data type	Access modifier	Description
bPhysicalInput	bool	protected	This field holds the boolean value to return true if the connection exist
sName	string	protected	This field holds the name of the connection
From	Connector	protected	This field holds the source connector
To	Connector	protected	This field holds the destination connector
Center	Connector	protected	This field holds the center connector
greenPen	Pen	protected	This field holds the green pen
blackPen	Pen	protected	This field holds the black pen
connectors	ConnectorCollection	protected	This field holds the connector collection

Method name	Return type	Access modifier	Description
Paint(System.Drawing.Graphics g)	void	public override	Function to Paint the connection on the canvas
Invalidate()	void	public override	Function to Invalidates the connection
Hit(Point p)	bool	public override	Function to test if the mouse hits this connection
HitConnector(Point p)	Connector	public	Function to return the connector hit by the mouse, if any
Move(Point p)	void	public override	Function to move the connection with the given shift
PontY(int Y)	void	public override	Function to get the total counter of the added controls
Connection()	–	public	Default constructor for the class
Connection(Point from, Point to)	–	public	Special Constructor with the site of the entity

Table 7.6 Summary of PLC ConnectionCollection class

Class summary			
Class ID: CP8			Namespace: PLCControlLibrary
Class name: ConnectionCollection			Class type: [Serializable]
Is Base Class?: No			No. of inherited classes: 0
Inherited From: CollectionBase			Interfaces with: None
Method name	Return type	Access modifier	Description
Add(Connection con)	int	public	Function to add the connection on the canvas
this[int index]	Connection	public	Function to get the connection
Remove(Connection con)	Void	public	Function to remove the connection
ConnectionCollection()	–	public	Constructor function for this class

Table 7.7 Summary of PLC Counter class

Class summary			
Class ID: CP9			Namespace: PLCControlLibrary
Class name: PLCCounter			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
cDownLeft	Connector	protected	This field holds the downleft connector
newImage	Image	private	This field holds the Image of Control
bPhysicalInput	bool	private	This field holds the physical input
bPhysicalOutput	bool	private	This field holds the physical output
iCounter	int	private	This field holds the counter value
iIncrementCounter	int	private	This field holds the current value of counter
iSecondRungNumber;	int	private	This field Hold the second rung number
sSecondRungName	string	private	This field Hold the Second Rung Name
Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
ReSet()	void	public	Function to reset the counter to 0
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
PLCCounter(MainLadder s): base(s)	–	public override	Constructor for this class

Table 7.8 Summary of PLC InternalRelayCoil class

Class summary			
Class ID: CP10			Namespace: PLCControlLibrary
Class name: InternalRelayCoil			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input
Bred	Brush	[NonSerialized()]	This field holds the red brush
Bgreen	Brush	[NonSerialized()]	This field holds the green brush
Bblack	Brush	[NonSerialized()]	This field holds the black brush

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
InternalRelayCoil(MainLadder s): base(s)	–	public override	Constructor for this class

Table 7.9 Summary of PLC NCInput class

Class summary			
Class ID: CP11			Namespace: PLCControlLibrary
Class name: NCInput			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
NCInput(MainLadder s): base(s)	–	public override	Constructor for this class

Table 7.10 Summary of PLC NCRelayContact class

Class summary			
Class ID: CP12			Namespace: PLCControlLibrary
Class name: NCRelayContact			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input
CoilName	Brush	[NonSerialized()]	This field holds the red brush

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
NCRelayContact(MainLadder s):	base(s)–	public override	Constructor for this class

Table 7.11 Summary of PLC NOInput class

Class summary			
Class ID: CP13			Namespace: PLCControlLibrary
Class name: NOInput			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
NOInput(MainLadder s) : base(s)	–	public override	Constructor for this class

Table 7.12 Summary of PLC NORelayContact class

Class summary			
Class ID: CP14			Namespace: PLCControlLibrary
Class name: NORelayContact			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input
CoilName	Brush	[NonSerialized()]	This field holds the red brush

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
NORelayContact(MainLadder s):	base(s)–	public override	Constructor for this class

Table 7.13 Summary of PLC Output class

Class summary			
Class ID: CP15			Namespace: PLCControlLibrary
Class name: Output			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType
Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	protected	This field holds the Image of Control
bPhysicalInput	bool	protected	This field holds the physical input

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
Output(MainLadder s): base(s)	–	public override	Constructor for this class

Table 7.14 Summary of PLCPowerRail class

Class summary			
Class ID: CP16			Namespace: PLCControlLibrary
Class name: PLCPowerRail			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: None

Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
bPhysicalInput	bool	protected	This field holds the physical input
BluePen	Pen	protected	This field holds the Blue pen
BlackPen	Pen	protected	This field holds the black pen
PowerRail	PowerRailTypes	protected	This field holds the type of bar left or right
PowerRungNumber	int	protected	This field holds the current rung number
bExtendedRung	bool	protected	This field holds the main rung or extended rung
iLinkTo	int	protected	This field holds the link to existing rung by rung number if ExtendedRung

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
PLCPowerRail(MainLadder s, PowerRailTypes Rail, int iPowerRailNumber, bool ExtendedRung): base(s)	–	public override	Constructor for this class

Table 7.15 Summary of PLC PowerRailCounter class

Class summary			
Class ID: CP17			Namespace: PLCControlLibrary
Class name: PowerRailCounter			Class type: concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
iPowerRailCounter	static int	private	This field holds the power rail counter
iExtendedPowerRailCounter	static int	private	This field holds the extended power rail counter
iExtendedShadowCounter	static int	private	This field holds the extended shadow counter

Method name	Return type	Access modifier	Description
Counter()	static void	public	Function to increment the power rail counter
GetCounter()	static int	public	Function to return the no. of power rail counter
RemoveCounter()	static void	public	Function to decrement the power rail counter
ResetCounter()	static void	public	Function to initialize the power rail counter and extended shadow counter to zero
ResetExtendedCounter()	static void	public	Function to initialize extended power rail counter to zero
ExtendedCounter()	static void	public	Function to increment the extended power rail counter and extended shadow counter
GetExtendedCounter()	static void	public	Function to return the no. of extended power rail counter
GetNextLocation()	static int	public	Function to return the next location of extended power rail counter
PowerRailCounter()	–	public	Constructor function for this class

Table 7.16 Summary of PLCSerializConnection class

Class summary			
Class ID: CP18			Namespace: PLCControlLibrary
Class name: PLCSerializConnection			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None

Field name	Data type	Access modifier	Description
_PhysicalInput	bool	private	This field holds boolean value to check if there is a physical input
_Name	string	private	This field holds the name of the connection
_ToPoint	Point	private	This field holds the destination point
_FromPoint	Point	private	This field holds the source point
_CenterPoint	Point	private	This field holds the center point
_ToConnectorControlName	string	private	This field holds the destination connector control name
_FromConnectorControlName	string	private	This field holds the source connector control name
_CenterConnectorControlName	string	private	This field holds the center connector control name
_ToConnectorName	string	private	This field holds the destination connector name
_FromConnectorName	string	private	This field holds the source connector name
_CenterConnectorName	string	private	This field holds the center connector name
_ToConnectorPoint	Point	private	This field holds the destination connector point
_FromConnectorPoint	Point	private	This field holds the source connector point
_CenterConnectorPoint	Point	private	This field holds the center connector point

Method name	Return type	Access modifier	Description
PLCSerializConnection()	–	public	Constructor function for this class

Table 7.17 Summary of PLCSerializControl class

Class summary			
Class ID: CP19			Namespace: PLCControlLibrary
Class name: PLCSerializControl			Class type: [Serializable]
Is base class?: Yes			No. of inherited classes: 2
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
_ControlType	PLCControlTypes	private	This field holds the PLC control type
_FunctionType	ControlFunctionType	private	This field holds the control function type
_PhysicalInput	bool	private	This field holds boolean value to check if there is a physical input
_ShowAddress	bool	private	This field holds the boolean value to show address of control
_ShowName	bool	private	This field holds the boolean value to show name of control
_OnExtendedRung	bool	private	This field holds the boolean value to check if the control is on extended rung
_RungNumber	int	private	This field holds the rung number
_RungName	string	private	This field holds the rung name
_Address	string	private	This field holds the address
_Name	string	private	This field holds the control name
_color	string	private	This field holds the control color
_CoilConnection	string	private	This field holds the coil connection name
_X	int	private	This field holds the X value position
_Y	int	private	This field holds the Y value position
_FirstConnectorControlName	string	private	This field holds the first connector control name
_SecondConnectorControlName	string	private	This field holds the second connector control name
_FirstConnectorName	string	private	This field holds the first connector name
_SecondConnectorName	string	private	This field holds the second connector name

(continued)

Table 7.17 (continued)

Field name	Data type	Access modifier	Description
_FirstConnectorPoint	Point	private	This field holds the first connector point
_SecondConnectorPoint	Point	private	This field holds the second connector point

Method name	Return type	Access modifier	Description
PLCSerializControl()	–	public	Constructor function for this class

Table 7.18 Summary of PLCSerializCounter class

Class summary			
Class ID: CP20			Namespace: PLCControlLibrary
Class name: PLCSerializCounter			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCSerializControl			Interfaces with: None
Field name	Data type	Access modifier	Description
_CurrentCounterValue	int	private	This field holds the current counter value
_PulsToCount	int	private	This field holds the pulse to count value
_SecondRungName	string	private	This field holds the second rung name
_SecondRungNumber	int	private	This field holds the second rung number
_ThirdConnectorControlName	string	private	This field holds the third connector control name
_ThirdConnectorName	string	private	This field holds the third connector name
_ThirdConnectorPoint	Point	private	This field holds the third connector point
Method name	Return type	Access modifier	Description
PLCSerializCounter()	–	public	Constructor function for this class

Table 7.19 Summary of PLCSerializPowerRail class

Class summary			
Class ID: CP21			Namespace: PLCControlLibrary
Class name: PLCSerializPowerRail			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None

Field name	Data type	Access modifier	Description
_ControlType	PLCControlTypes	private	This field holds the PLC control type
_PhysicalInput	bool	private	This field holds boolean value to check if there is a physical input
_ExtendedRung	bool	private	This field holds the boolean value to check if the control is on extended rung
_RungNumber	int	private	This field holds the rung number
_LinkTo	int	private	This field holds the link number
_Name	string	private	This field holds the control name
_Location	Point	private	This field holds the location of connection
_BarType	PowerRailTypes	private	This field holds the bar type of power rail
_ConnectorControlName	string	private	This field holds the connector control name
_ConnectorName	string	private	This field holds the connector control name
_ConnectorPoint	Point	private	This field holds the connector point

Method name	Return type	Access modifier	Description
PLCSerializPowerRail()	–	public	Constructor function for this class

Table 7.20 Summary of PLCSerializTimer class

Class summary			
Class ID: CP22			Namespace: PLCControlLibrary
Class name: PLCSerializTimer			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCSerializControl			Interfaces with: None
Field name	Data type	Access modifier	Description
_TimeToCount	double	private	This field holds the time counter value
_TimerType	TimerTypes	private	This field holds timer type
Method name	Return type	Access modifier	Description
PLCSerializTimer()	–	public	Constructor function for this class

Table 7.21 Summary of SaveLadder class

Class summary			
Class ID: CP23			Namespace: PLCControlLibrary
Class name: SaveLadder			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: ISerializable
Field name	Data type	Access modifier	Description
Conns	ArrayList	private	This field holds array of connections
Conts	ArrayList	private	This field holds array of controls

Method name	Return type	Access modifier	Description
GetObjectData(SerializationInfo info, StreamingContext context)	void	public	Function to add a new connection with given control type
SaveLadder(SerializationInfo info, StreamingContext context)	–	public	This is the special constructor called during deserialization
SaveLadder()	–	public	Default Constructor function for this class

Table 7.22 Summary of PLCTimer class

Class summary			
Class ID: CP24			Namespace: PLCControlLibrary
Class name: PLCTimer			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: PLCBase			Interfaces with: IControlFunctionType

Field name	Data type	Access modifier	Description
cLeft	Connector	protected	This field holds the left connector
cRight	Connector	protected	This field holds the right connector
newImage	Image	private	This field holds the Image of Control
bPhysicalInput	bool	private	This field holds the physical input
bPhysicalOutput	bool	private	This field holds the physical output
iTimeToCount	int	private	This field holds the timer value
iTimeCounter	int	private	This field holds the counter value of timer
bInput	int	private	This field holds the bool value for timer
_TimerType	string	private	This field holds the TimerTypes value of timer

Method name	Return type	Access modifier	Description
Hit(System.Drawing.Point p)	bool	public override	Function to test whether the mouse hits this control
Paint(System.Drawing.Graphics g)	void	public override	Function to paint the Control on the canvas
Invalidate()	void	public override	Function to invalidate the control
PLCTimer(MainLadder s): base(s)	–	public	Constructor for this class

Table 7.23 Summary of PLC MainLadder class

Class summary			
Class ID: CP25			Namespace: PLCControlLibrary
Class name: MainLadder			Class type: concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: System.Windows.Forms.Control			Interfaces with: none
Field name	Data type	Access modifier	Description
iOutPutCounter	int	private	This field holds the output connector
iTimerCounter	int	private	This field holds the timer counter
iConnectionCounter	int	private	This field holds the connection counter
MousePoint	Point	private	This field holds the mouse point
p	Point	private	This field holds the point variable
pMousePosition	Point	private	This field holds the mouse position
SelectedControl	PLCControlTypes	public	This field holds the selected control
cMenu	ContextMenu	private	This field holds the context menu
ExternalInputArray	ArrayList	private	This field holds the external input array
LoadList	ArrayList	private	This field holds the load list array
ControlList	ArrayList	private	This field holds the control list array
BluePen	Pen	private	This field holds the blue pen
LightGrayPen	Pen	private	This field holds the light gray pen
iRungHeight	int	private	This field holds the rung height
connections	PLCControlLibrary.ConnectionCollection	protected	This field holds the connections collection
saveLadder	PLCControlLibrary.SaveLadder	protected	This field holds the ladder properties
refp	Point	protected	This field holds the reference point
rnd	Random	protected	This field holds the randomly selected point
RungNumber	int	protected	This field holds the rung number
PlcCollection	PLCControlCollection	protected	This field holds the collection of PLC Controls on the ladder diagram

(continued)

Table 7.23 (continued)

Field name	Data type	Access modifier	Description
tracking	bool	protected	This field holds the boolean value to check tracking
showGrid	bool	protected	This field holds the boolean value to check if grid is to be shown
gridSize	Size	protected	This field holds the default gridsize used in the paint-background method
hoveredEntity	Entity	protected	This field holds the entity hovered by the mouse
selectedEntity	Entity	protected	This field holds the unique entity currently selected
proxy	Proxy	protected	This field holds the simple proxy for the propsgrid of the control
components	private System. ComponentModel. Container	protected	This field holds the system components container

Method name	Return type	Access modifier	Description
OnPaint(PaintEventArgs e)	void	protected override	Function to handle the paint event
OnPaintBackground(PaintEventArgs e)	void	protected override	Function to handle the paint-background event
OnMouseMove(MouseEventArgs e)	void	protected override	Function to handle the mouse move event
ShowPLCControls(ArrayList PLCControlArray)	void	private	Function to show PLC controls
GetFromConnector(string sConnectorName)	Connector	private	Function to get the source connector
GetToConnector(string sConnectorName)	Connector	private	Function to get the destination connector
AttacheConnector(ArrayList PLCConnectionArray)	void	private	Function to attach a connector to the PLC connection array
GetRightConnector(string sConnectorName)	void	private	Function to get the right connector
GetCenterConnector(string sConnectorName)	void	private	Function to get the center connector
GetLeftConnector(string sConnectorName)	void	private	Function to get the left connector
ClearAllControl()	void	public	Function to clear all controls
ShowPLCConnections(ArrayList PLCConnectionArray)	void	private	Function to show PLC connections
LoadLadderDiagram(string sFileName)	void	public	Function to load ladder diagram
Dispose(bool disposing)	void	protected override	Function to clean up any resources being used.

(continued)

Table 7.23 (continued)

Method name	Return type	Access modifier	Description
OnClick(EventArgs e)	void	protected override	Function to handle button click event
GetPLCControls(int RungNumber)	ArrayList	public	Function to return the collection of all control that have property set to ControlFunctionType.ExternalInput and belong to same rung number
GetRungName(int iRungNumber)	string	private	Function to get rung name
GetPLCControls(ControlFunctionType FunctionType,ControlFunctionType SecondFunctionType, int RungNumber)	ArrayList	public	Function to return the collection of all control that have property set to ControlFunctionType and contained in RungNumber
GetPLCControls(ControlFunctionType FunctionType,ControlFunctionType SecondFunctionType, int RungNumber, int XLocation)	ArrayList	public	Function to return the collection of all control that have property set to ControlFunctionType and contained in RungNumber and have X Access less then XLocation
GetPLCControls(ControlFunctionType FunctionType, int RungNumber)	ArrayList	public	Function to return the collection of all control that have property set to ControlFunctionType and contained in RungNumber
GetPLCControls(ControlFunctionType FunctionType)	ArrayList	public	Function to return the collection of all control that have property set to ControlFunctionType
GetPLCControls()	ArrayList	public	Function to return the collection of all controls
GetPLCConnections()	ArrayList	public	Function to return the collection of all connections
OnMouseUp(MouseEventArgs e)	void	protected override	Function to handle the mouse-up event
OnMouseDown(MouseEventArgs e)	void	protected override	Function to handle the mouse-down event
AddConnection(Connection con)	Connection	public	Function to add a connection to the diagram
AddConnection(Point startPoint)	Connection	public	Function to add a connection with a start point
AddConnection(Connector from, Connector to)	Connection	public	Function to add a connection with specified source and destination
AddConnection(Point from, Point to)	Connection	public	Function to add a connection with specified source and destination
GetRungY(int RungNumber)	int	private	Function to get the specified rung
GetControlName(int YValue)	string	private	Function to get the specified control name

(continued)

Table 7.23 (continued)

Method name	Return type	Access modifier	Description
GetRungStatus(string sRungName)	bool	private	Function to get the rung status
DeleteFromRegister(PLCBase control)	void	private	Function to delete from register
GetTotalExtendedRung(int iMainRungNumber)	ArrayList	private	Function to get the total extended rung
GetControls(string sRungName)	ArrayList	private	Function to get controls with specified name
GetControls(string sRungName, int XLocation)	ArrayList	private	Function to get controls with specified name and location
IOUpdate()	void	public	Function to update the input output of the control
SolveLadder()	void	public	Function to solve the ladder
SaveLadderDiagram(string sFileName)	void	public	Function to save ladder diagram
HoverNone()	void	private	Function to reset the hovering status of the control, i.e., the hoverEntity is set to null.
AddControl(PLCBase control)	PLCBase	public	Function to add a PLC Control to the Ladder Diagram
AddControl(PLCPowerRail controlToAdd, PLCSerializPowerRail prail)	void	private	Function to add a PLC Control to the Ladder Diagram with specified power rail and serialized power rail
AddControl(PLCCounter controlToAdd,PLCSerializCounter counter)	void	private	Function to add a PLC Control to the Ladder Diagram with specified counter and serialized counter
AddControl(PLCTimer controlToAdd,PLCSerializTimer timer)	void	private	Function to add a PLC Control to the Ladder Diagram with specified timer and serialized timer
AddControl(PLCBase controlToAdd,PLCSerializControl controL)	void	private	Function to add a PLC Control to the Ladder Diagram with specified control and serialized control
AddControl(PLCControlTypes type, Point location)	PLCBase	public	Function to add a PLC Control to the Ladder Diagram with specified control type and location
BuildMenu()	void	private	Function to build menu
GetLocation()	int	private	Function to get location
InitializeComponent()	void	private	Required method for Designer support
menuAddConnection_Click(object sender, EventArgs e)	void	private	Function to handle add connection menu

(continued)

Table 7.23 (continued)

Method name	Return type	Access modifier	Description
menuAddRung_Click(object sender, EventArgs e)	void	private	Function to handle add rung menu
menuAddExtendedRung_Click(object sender, EventArgs e)	void	private	Function to handle add extended rung menu
menuNOInput_Click(object sender, EventArgs e)	void	private	Function to handle NO Input menu
menuNCInput_Click(object sender, EventArgs e)	void	private	Function to handle NC input menu
menuInternalRelayCoil_Click(object sender, EventArgs e)	void	private	Function to handle internal relay coil menu
menuNORelayContact_Click(object sender, EventArgs e)	void	private	Function to handle NO relay contact menu
menuNCRelayContact_Click(object sender, EventArgs e)	void	private	Function to handle NC relay contact menu
menuOutput_Click(object sender, EventArgs e)	void	private	Function to handle output menu
menuCounter_Click(object sender, EventArgs e)	void	private	Function to handle counter menu
menuTimer_Click(object sender, EventArgs e)	void	private	Function to handle timer menu
menuDelete_Click(object sender, EventArgs e)	void	private	Function to handle delete menu
MainLadder()	–	private	Constructor for this class

Event name	Delegate name	Data type	Description
ControlSelected	ControlSelectedEventHandler	public	Event for handling selected control
CurrentRung	CurrentRungEventHandler	public	Event for handling current rung

Table 7.24 Summary of PLC sequence diagram

Sequence diagram summary				
Sequence ID: SD-P1 Sequence name: PLC System Sequence Diagram				
Method name	Type	Description	Source	Destination
MainLadder()	Return message	Function to add main ladder diagram on the screen on which the controls are to be added	Form	MainLadder
PLCControlCollection	Return message	Function to load and return collection of PLC controls	MainLadder	PLCControlCollection
ConnectionCollection	Return message	Function to load and return collection of control connections	MainLadder	ConnectionCollection
Add Connection()	Message	Function to add a new connection	Form	MainLadder
Connection	Return message	Function to return a new connection	MainLadder	Connection
ConnectorCollection()	Message	Function to get a connector from the collection	Connection	ConnectorCollection
Connector	Return message	Function to return a connector to the connection class	Connection	Connector
Center Connector	Return message	Function to return the center connector to the connection class	Connection	Connector
Add (Center)	Message	Function to add the center connector	Connection	ConnectorCollection
Clear All Control()	Return message	Function to clear all controls	Form	MainLadder
Clear()	Message	Function to clear the control and connection collection	MainLadder	PLCControlCollection, ConnectionCollection
Add Connection()	Self message	Function to add connection	MainLadder	MainLadder
Add Conrol(Load.location)	Message	Function to add control at the specified location	Form	MainLadder
PLCLoad()	Return message	Function to load PLC diagram	MainLadder	PLCLoad
Add (control)	Message	Function to add a control	MainLadder	PLCControlCollection
LeftConnector	Return message	Function to get the left connector	PLCLoad	Connector
Add(LeftConnector)	Return message	Function to add the left connector	PLCLoad	ConnectorCollection
RightConnector	Return message	Function to get the right connector	PLCLoad	Connector
Add(RightConnector)	Return message	Function to add the right connector	PLCLoad	ConnectorCollection

Table 7.25 Summary of PLC Analyze Diagram use case

Use case summary
Use case ID: UC-P1
Use case name: Analyze Diagram
Actors: User
Description: The user selects Analyze Diagram button on the MainForm to analyze the connections and the flow of PLC system before running it
Preconditions: PLC controls and their connections are defined
Post conditions: Display message to indicate the status of the system analysis
System parameters: PLC ladder
Success scenario: Ladder is analyzed and success result is displayed
Alternative scenario: Problems in ladder and control connections are displayed and the ladder is cleared out
Includes: Analyze, Clear Out, Analyze Diagram, Ladder
Priority: High

Table 7.26 Summary of Run PLC use case

Use case summary
Use case ID: UC-P2
Use case name: Run PLC
Actors: User
Description: The user selects Run PLC on the MainForm to run the PLC simulation. Its starts the timer and checks I/O until the ladder is completed
Preconditions: The PLC system is designed and analyzed successfully
Post conditions: The output displays the executed PLC system with illuminated LEDs in 2D
System parameters: PLC ladder
Success scenario: The timer starts, the system checks the external inputs, perform ladder I/O updates and solve ladder in a cycle until all the controls are completed
Alternative scenario: If there is a problem, the system disables the PLC diagram
Includes: Run, Disable Diagram, Start System timer, Timer, Check External Input, Solve Ladder, Ladder I/O Update
Priority: High

Table 7.27 Summary of Add PLC Control

Use case summary
Use case ID: UC-P3
Use case name: Add PLC Control
Actors: User
Description: The user adds a PLC control on the ladder while designing the PLC system
Preconditions: None
Post conditions: A new PLC control is added on the ladder
System parameters: PLC Control
Success scenario: The user selects from PLC Control Collections and adds to the ladder at the required position
Alternative scenario: PLC Control Collections have no sufficient controls to select from
Includes: Ladder, PLC Control Collection, PLC Control, Connector Collection, Connector
Priority: High

Table 7.28 Summary of Add Connection use case

Use case summary
Use case ID: UC-P4
Use case name: Add Connection
Actors: User
Description: The user adds a connector to the PLC control on the ladder while designing the PLC system
Preconditions: None
Post conditions: A new connection to PLC control is added on the ladder
System parameters: Connector
Success scenario: The user selects from PLC Connector Collections and adds to the PLC control on the ladder
Alternative scenario: PLC Connector Collections have no sufficient connections to select from
Includes: Ladder, PLC Connector Collection, Connection, Connector Collection, Connector
Priority: High

Table 7.29 Summary of PLC processes flow charts

Flow chart summary	
Flow chart ID: FC-P1	
Flow chart name: PLC Process Flow Chart	
No. of inputs: 6	
No. of processes: 71	
No. of outputs: 6	
No. of control decisions: 22	
Input name	**Description**
Save Ladder	Save Ladder button on screen
Run Simulation	Run Simulation menu on screen
Open Ladder	Open Ladder menu on screen
New Ladder	New Ladder menu on screen
Analyze Ladder	Analyze Ladder menu on screen
Add Control to Ladder	Add control to ladder menu on screen
Process name	**Description**
Analyze Diagram	Function to analyze all connectors and controls
Save Dialog Box	Function to call save dialog box
Open Dialog Box	Function to call open dialog box
Clear All Controls	Function to clear all controls
Clear Connection	Function to clear connections collection
Clear Control	Function to clear controls collection
Clear Input Image Register	Function to clear input image register
Clear Output Image Register	Function to clear output image register
Reset Rail Counter	Function to Reset Rail Counter
Reset Extended Rail Counter	Function to Reset Extended Rail Counter
Clear Coil	Function to Clear Coil

(continued)

Table 7.29 (continued)

Process name	Description
Open Ladder	Function to Open Ladder
Load Ladder Diagram	Function to Load Ladder Diagram
Create Soap Formatter	Function to Create Soap Formatter
Create File Stream	Function to Create File Stream
Deserialize File Stream	Function to Deserialize File Stream
Show Connections from File Stream	Function to Show Connections from File Stream
Show Controls from File Stream	Function to Show Controls from File Stream
Attach Connection with Controls	Function to Attach Connection with Controls
Close File Stream	Function to Close File Stream
Analyze	Function to Analyze
Get All Connections From Ladder	Function to Get All Connections From Ladder
Get All Controls From Ladder	Function to Get All Controls From Ladder
Loop through each connection	Function to Loop through each connection for verification
Write Output "Analyzing Controls"	Function to Write Output "Analyzing Controls"
Write Error to Output Window	Function to Write Error to Output Window
Write Error Disable Simulation	Function to Write Error Disable Simulation
Ready for Run Simulation Enable Simulation	Function to Ready for Run Simulation Enable Simulation
Check All External Inputs	Function to Check All External Inputs
OFF Connections attached with control	Function to OFF Connections attached with control
ON Connections attached with control	Function to ON Connections attached with control
IO Update	Function to perform IO Update
Set Output Control OFF	Function to Set Output Control OFF
Set Output Control ON	Function to Set Output Control ON
Solve Ladder Logic	Function to Solve Ladder Logic
Loop for each Main Rung	Function to Loop for each Main Rung

(continued)

Table 7.29 (continued)

Process name	Description
Loop for each Extended Rung	Function to Loop for each Extended Rung
Check Extended Rung Logic	Function to Check Extended Rung Logic
Check Main Rung Logic	Function to Check Main Rung Logic
Calculate Final rung Logic	Function to Calculate Final rung Logic
Add Connection	Function to Add Connection
Get Connection Start Point	Function to Get Connection Start Point
Get Connection End Point	Function to Get Connection End Point
Attaches From Connector	Function to Attaches From Connector
Attaches To Connector	Function to Attaches To Connector
Add Connection to Connections Collection	Function to Add Connection to Connections Collection
Add Control	Function to Add Control
Create New Normally Open Contact	Function to Create New Normally Open Contact
Add Control to Control Connection	Function to Add Control to Control Connection
Create New Normally Close Contact	Function to Create New Normally Close Contact
Create New Internal Relay Coil	Function to Create New Internal Relay Coil
Create New Right Rung	Function to Create New Right Rung
Create New Left Rung	Function to Create New Left Rung
Create New Right Extended Rung	Function to Create New Right Extended Rung
Create New Left Extended Rung	Function to Create New Left Extended Rung

Output name	Description
Display ladder diagram	Open and display ladder diagram on screen
Analysis Result	Display analysis result
Run Simulation	Display running simulation
Save ladder diagram	Save ladder diagram on disk
Add connection	Add connection to the ladder diagram
Add control	Add control to the ladder diagram

(continued)

Table 7.29 (continued)

Control decision name	Description
Analyze == OK	Check if the user pressed OK on Analyze dialog
Dialog Box == OK	Check if the user pressed OK on Open and Save Dialog Box
Control	Check if the type is Control
Connection	Check if the type is Connection
Control Left Connector connected to any connection	Check if the Control Left Connector connected to any connection
Control right Connector connected to any connection	Check if the Control right Connector connected to any connection
Connection From Connector Attached to Any Control	Check if the Connection From Connector Attached to Any Control
Connection To Connector Attached to Any Control	Check if the Connection To Connector Attached to Any Control
External Input == ON	Check if the External Input == ON
Check Input State of Inputs Controls	Check Input State of Inputs Controls is ON or OFF
Have any Extended Run	Check if the selected rung have any Extended Run
Final Rung Logic	Check if the Final Rung Logic is YES or NO
Normally Open Contact	Check if the user selected Normally Open Contact
Normally Close Contact Internal Relay Coil	Check if the user selected Normally Close Contact Internal Relay Coil
Output	Check if the user selected Output
Normally Open Relay Contact	Check if the user selected Normally Open Relay Contact
Normally Close Relay Contact	Check if the user selected Normally Close Relay Contact
Timer	Check if the user selected Timer
Counter	Check if the user selected Counter
Rung	Check if the user selected Rung
Extended Rung	Check if the user selected Extended Rung

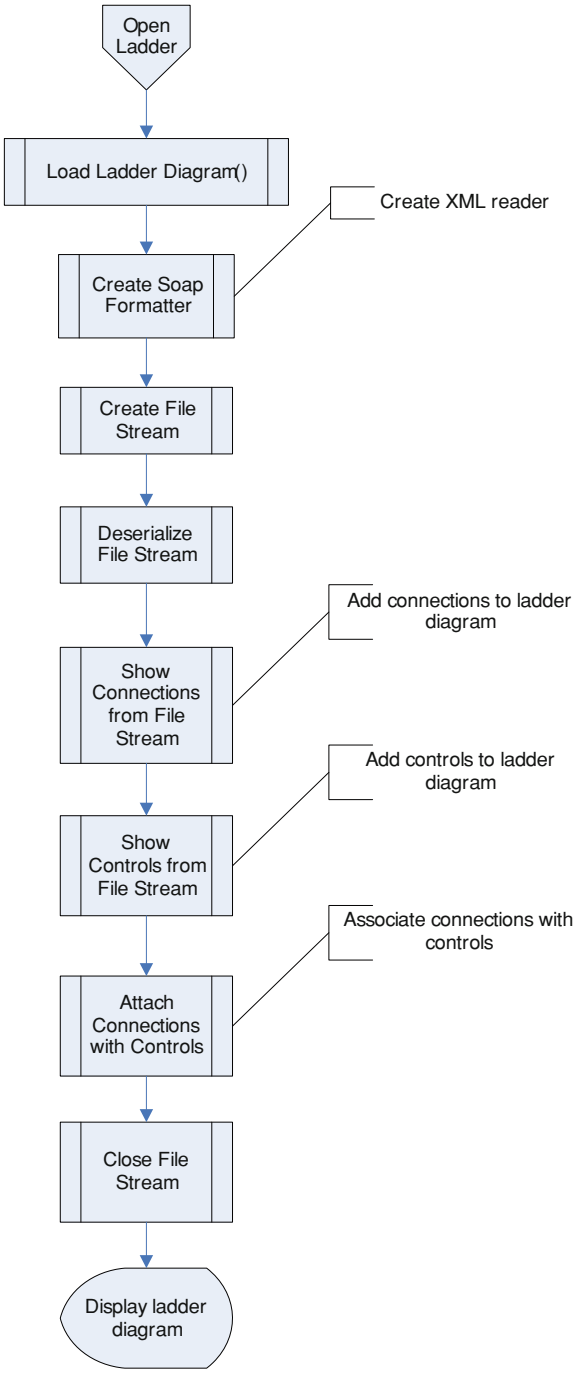


Fig. 7.14 Function for opening the Ladder Diagram

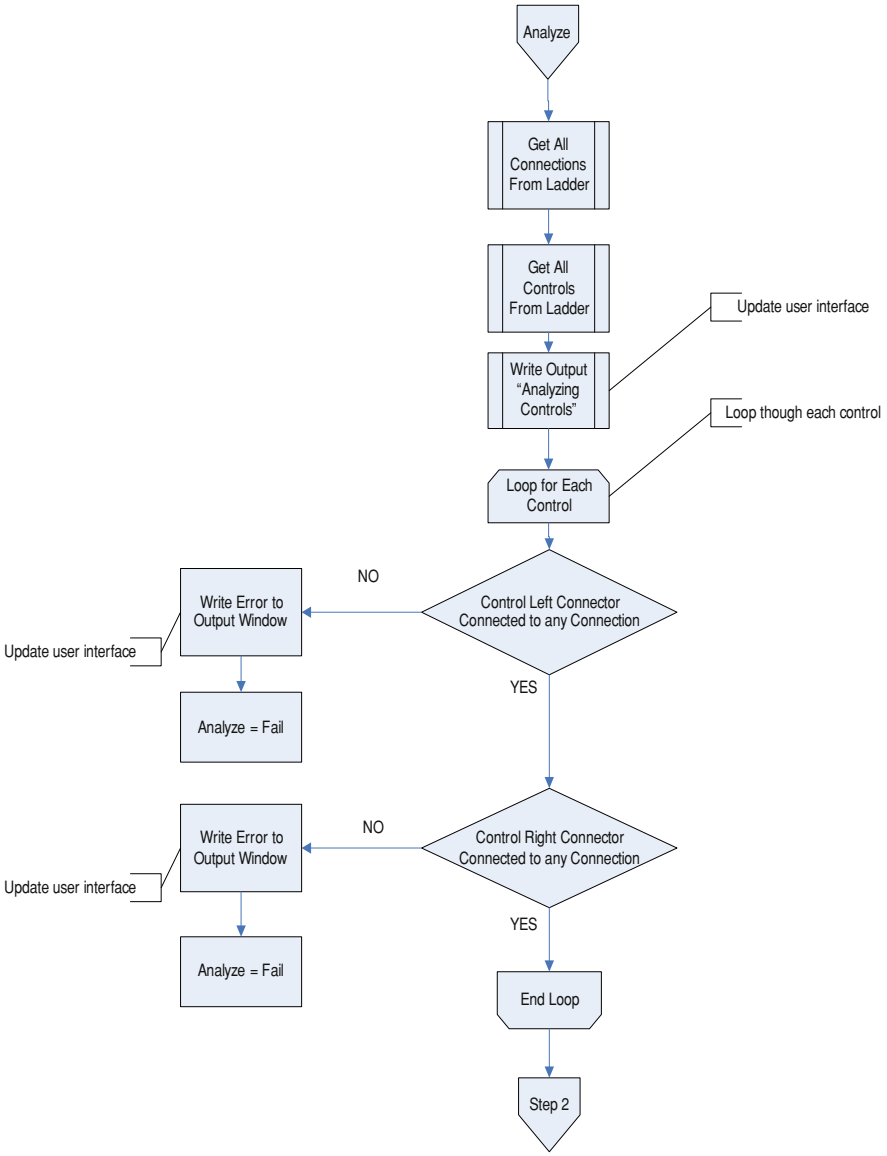


Fig. 7.15 Function for analyzing the Ladder Diagram

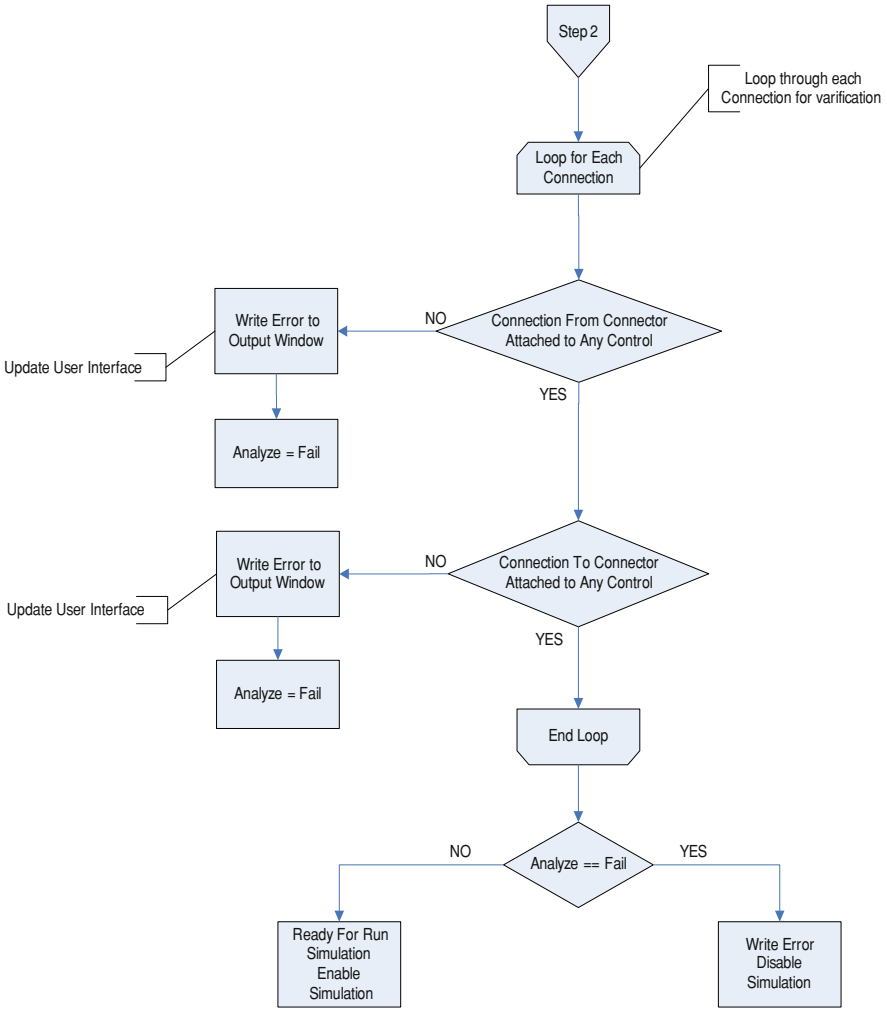


Fig. 7.15 (continued)

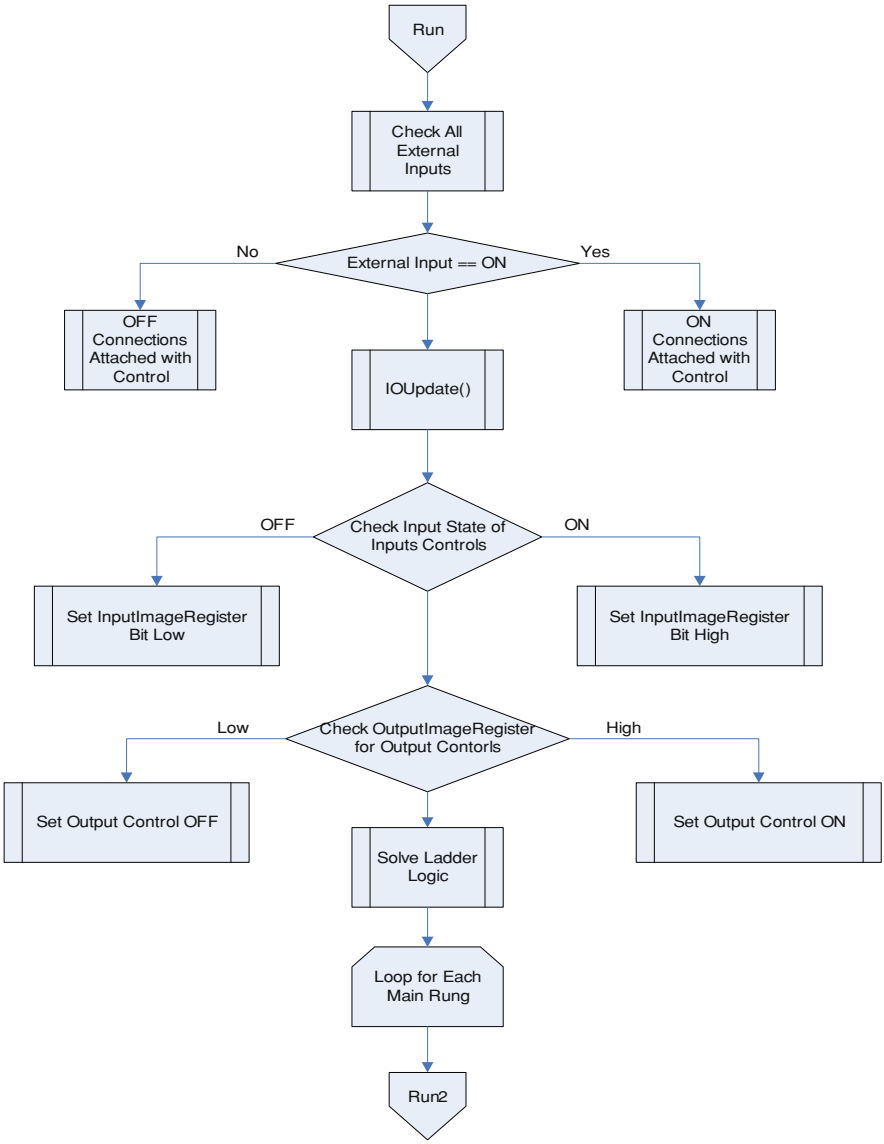


Fig. 7.16 Function for simulating the Ladder Diagram

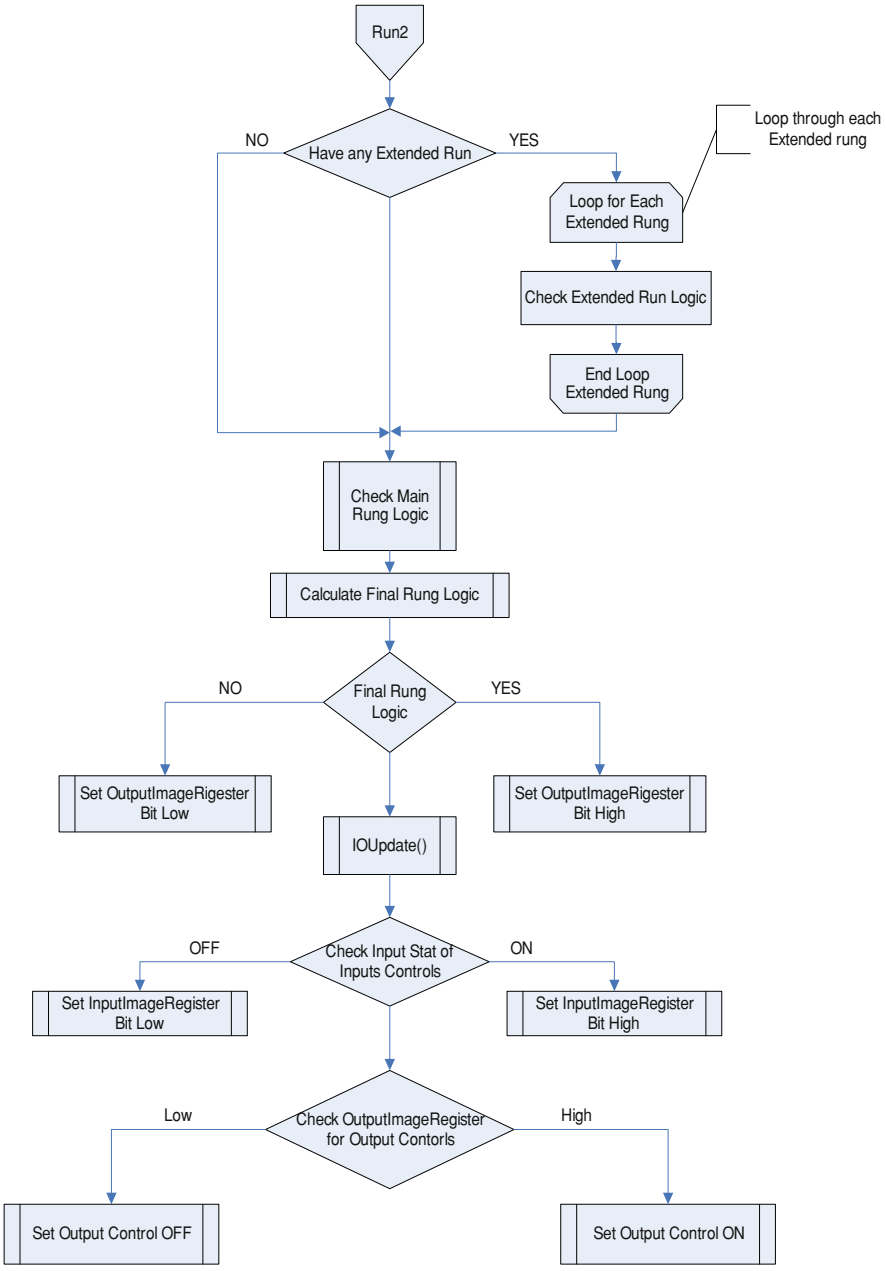


Fig. 7.16 (continued)

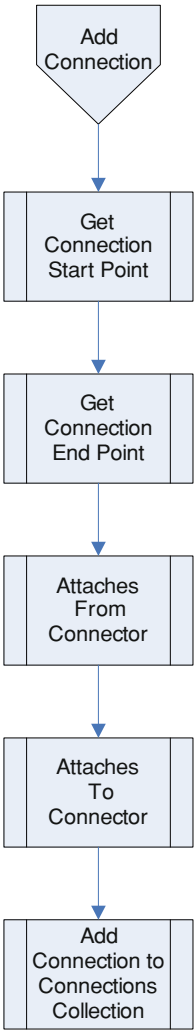


Fig. 7.17 Function for Add Control

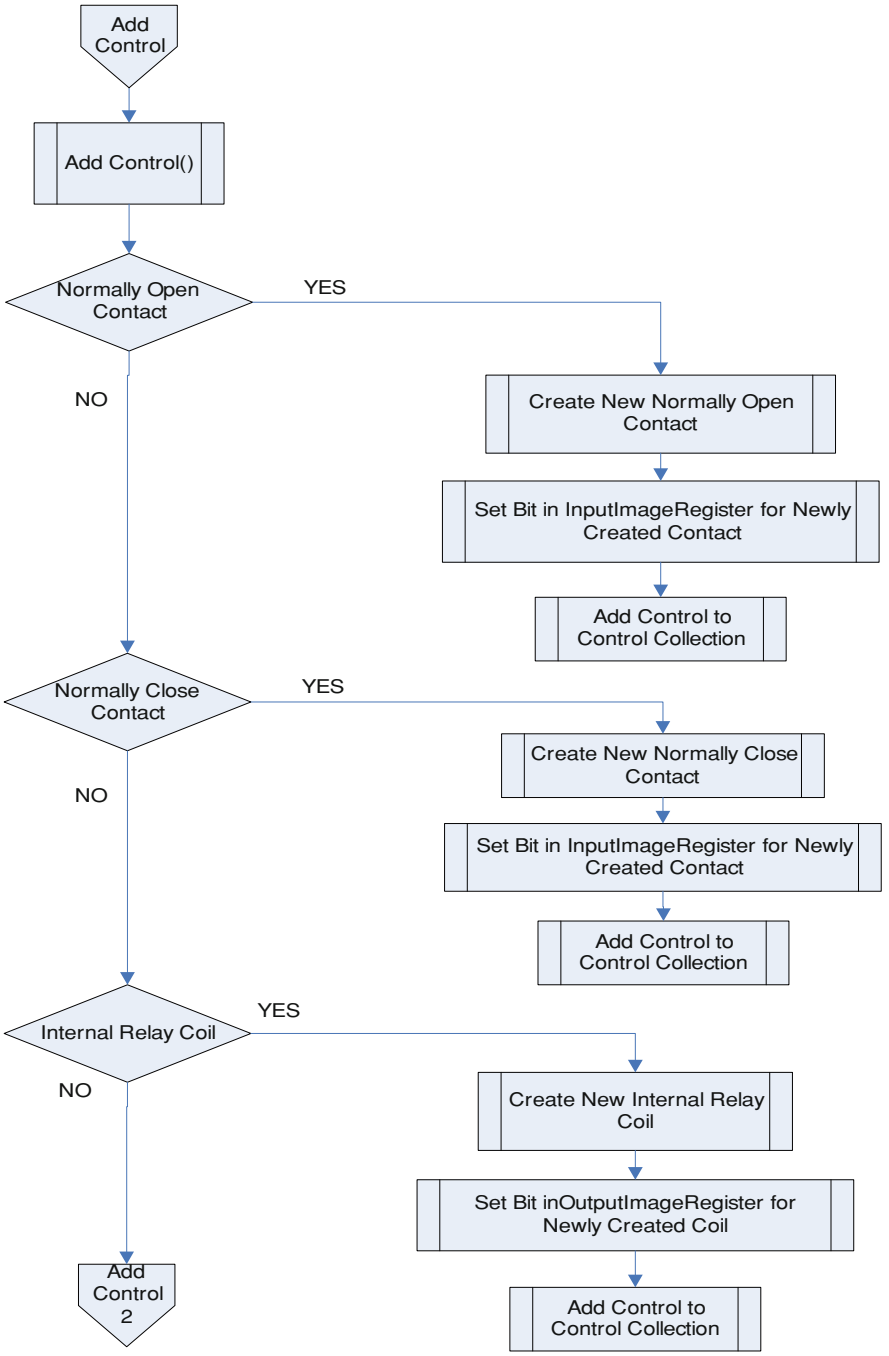


Fig. 7.17 (continued)

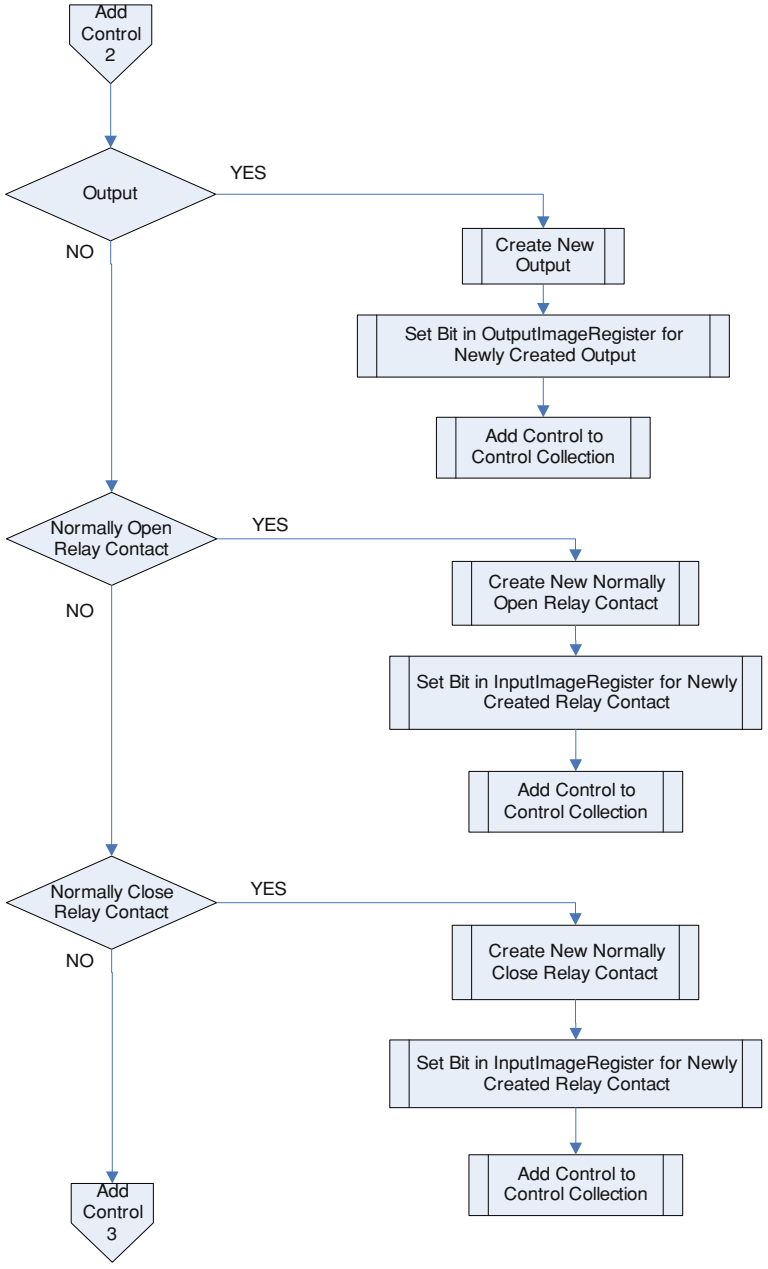


Fig. 7.17 (continued)

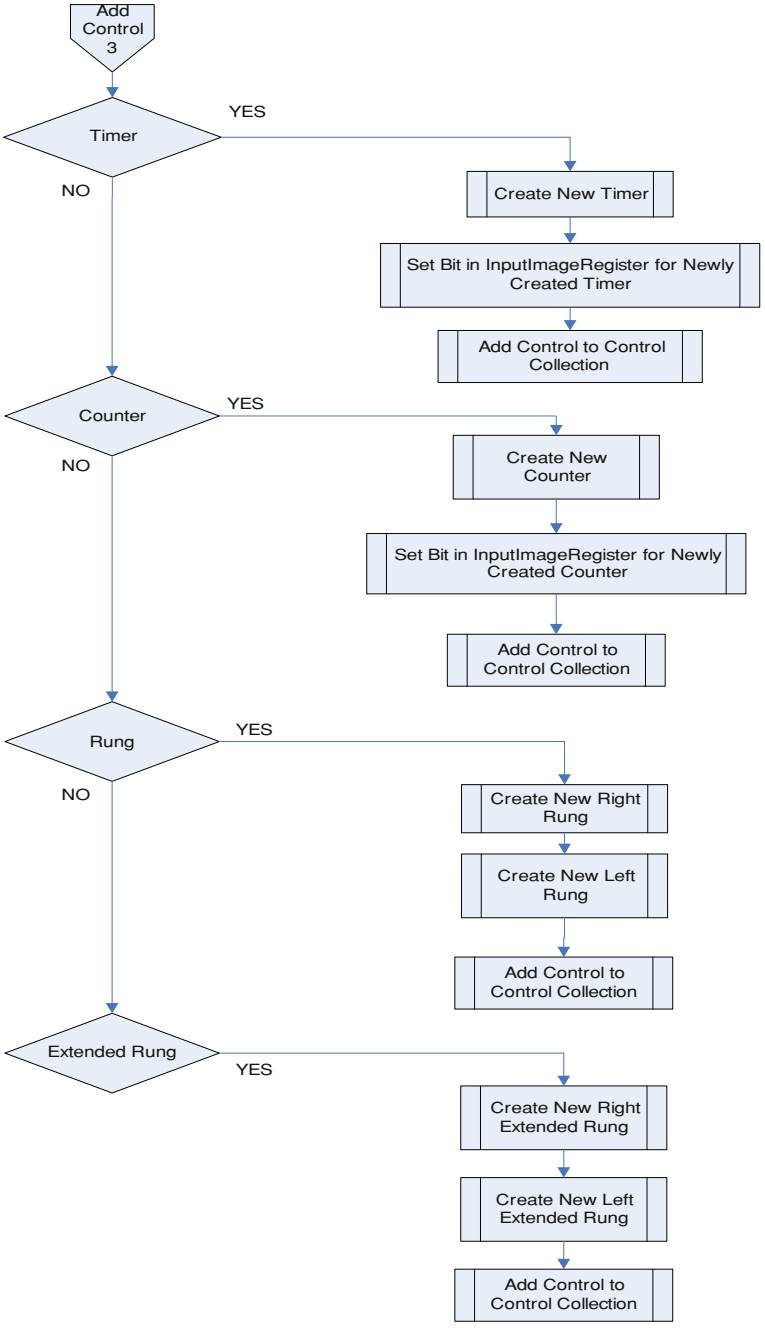


Fig. 7.17 (continued)

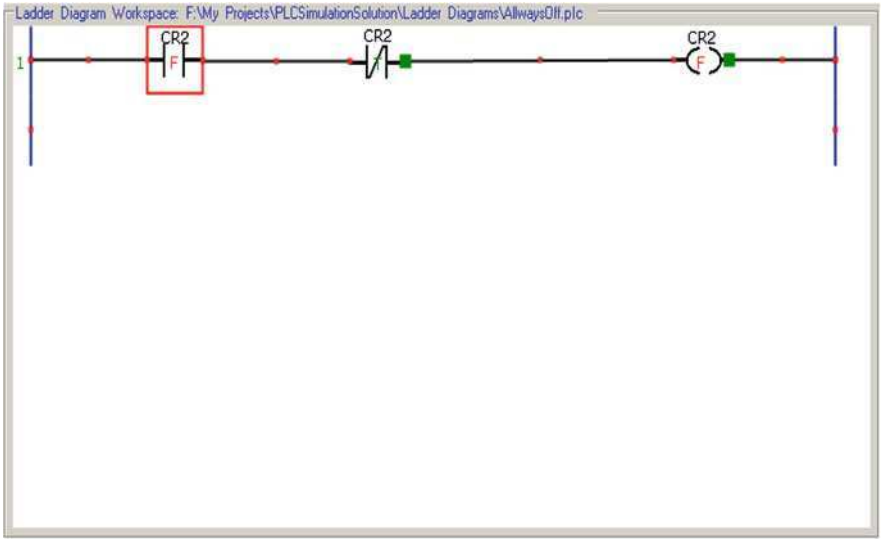


Fig. 7.18 Ladder Diagram Workspace

Fig. 7.19 Output Workspace

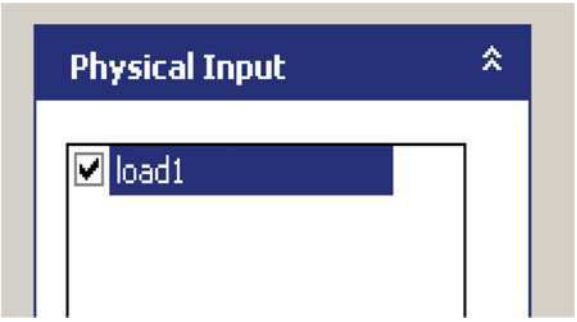


Fig. 7.20 Physical Input

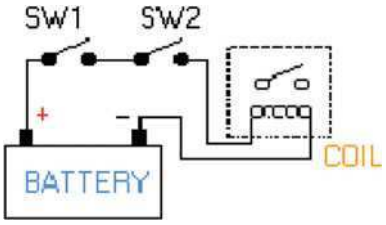


Fig. 7.21 Example Circuit

Fig. 7.22 Ladder Diagram for the Example Circuit

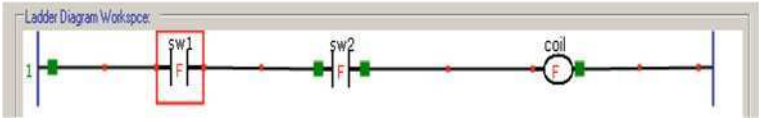


Fig. 7.23 Complete Ladder Diagram for the Example Circuit

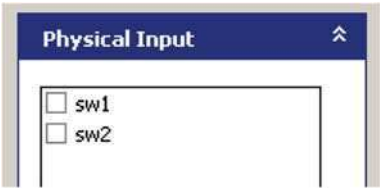


Fig. 7.24 SW1 and SW2 status (both SW1 and SW2 are OFF)

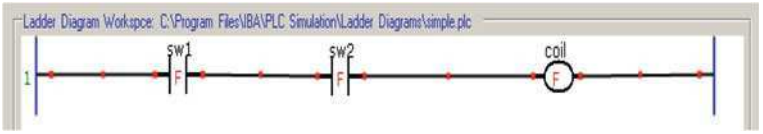


Fig. 7.25 Alignment with actual condition

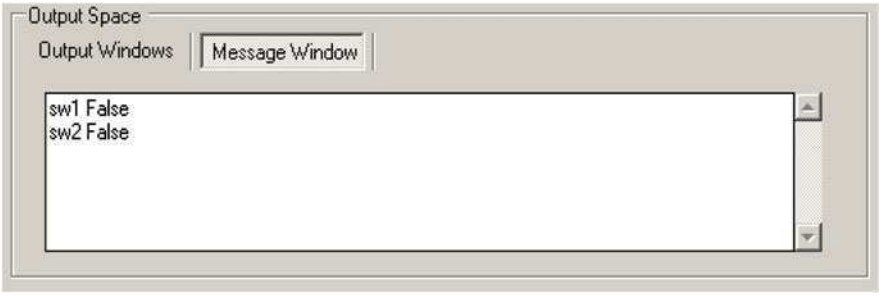


Fig. 7.26 Output Space Message Window shows that SW1 and SW2 both are OFF

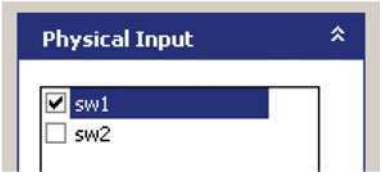


Fig. 7.27 Switching ON

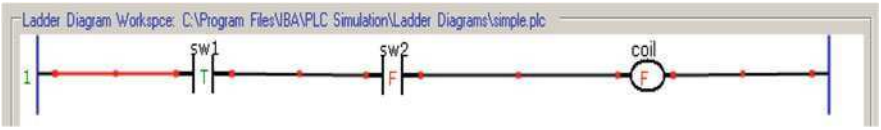


Fig. 7.28 Ladder Diagram Status after switching ON SW1

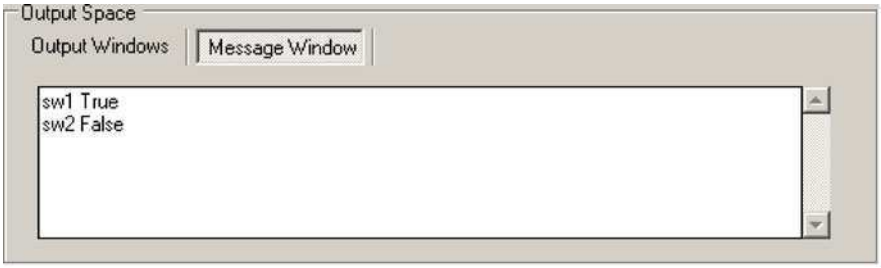


Fig. 7.29 Output Space Message Window also showing that SW1 is ON

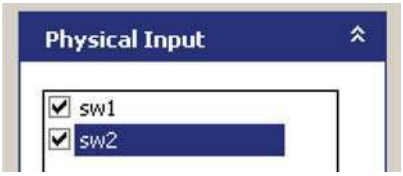


Fig. 7.30 Click on SW2 check box to switch it ON as well

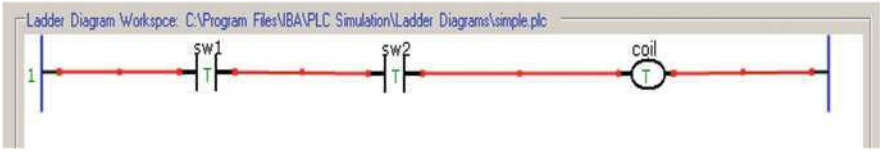


Fig. 7.31 Ladder Diagram Status after switching ON SW2

Fig. 7.32 Output Space Message Window also showing that both SW1 and SW2 are ON

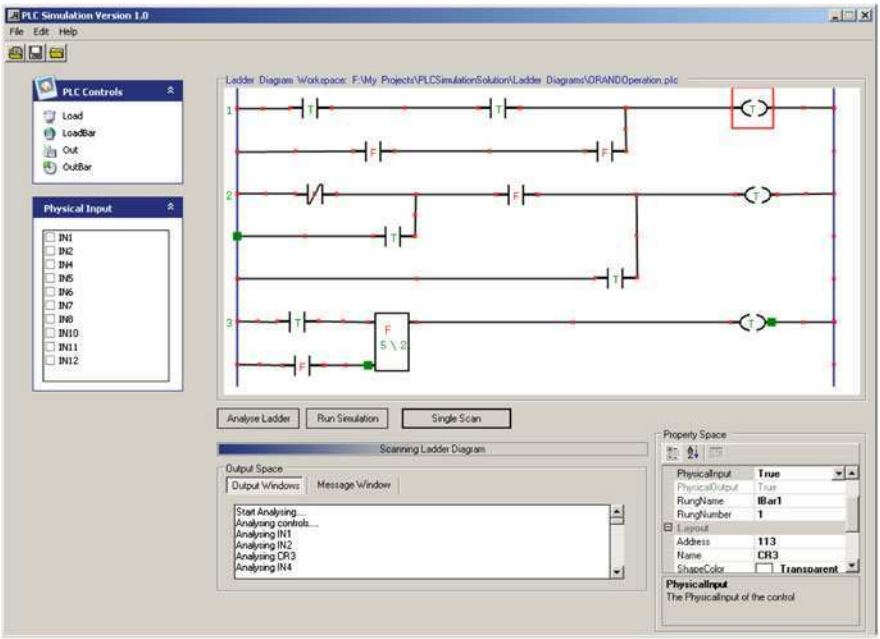


Fig. 7.33 A Proxy PLC Ladder Diagram during scanning mode

7.8 Interface Design for System Integration

A simplified hardware interface design between real and proxy system is provided in Fig. 7.34.

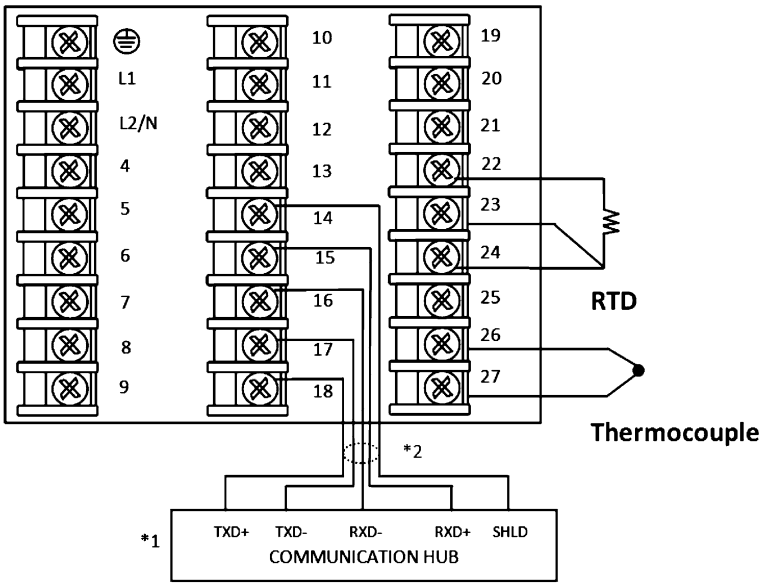


Fig. 7.34 Control of PLC over local area network. **1* Connecting directly to a PC will require the use of an ethernet crossover cable, **2* Use shielded twisted-pair, category 5 ethernet cable

Bibliography

1. Alting L (1994) Manufacturing engineering process. Marcel Dekker, New York
2. Azuma R, Bailout Y, Behringer R, Feiner S, Julier S, MacIntyre B (2001) Recent advances in augmented reality. IEEE Comput Graph Appl 21(6):34–47
3. Bedworth DD et al (1991) Computer integrated design and manufacturing. McGraw Hill, New York
4. Berger AS (2005) Embedded system design. Replika Press, Narela
5. Bishop HR (ed) (2002) Mechatronics. CRC Press, Boca Raton
6. Boyer SA (2004) SCADA. ISA, Decatur
7. Chen J (2006) A survey of 3D Graphics software tools.CS ready notes IEEE Comput Soc.
8. Chryssolouris G (1992) Manufacturing systems: theory and practice. Springer Verlag, New York
9. Hackworth JR, Hackworth FD Jr (2004) Programmable logic controller. Pearson, Upper Saddle River
10. Hitomi K (1975) Manufacturing systems engineering. Taylor & Francis, Bristol
11. Hsieh SJ (2004) Integrated virtual learning system for programmable logic controller. J Eng Educ 93(2):169–178
12. http://en.wikipedia.org/wiki/Ladder_logic
13. http://en.wikipedia.org/wiki/Programmable_logic_controller
14. http://www.iec.org
15. http://www.mikroelektronika.co.yu/english/product/books/PLCbook/chapter2/chapter2.htm
16. Johnson B et al (2003) Inside Microsoft Visual Studio.NET 2003. Microsoft Press, Redmond

17. Kalpakjian S (1992) Manufacturing engineering process and technology. Addison-Wesley, New York
18. Kief HB (1986) Flexible automation—the international CNC reference book. Becker Publishing, Eden Prairie
19. Park SC (2008) A PLC programming environment based on a virtual plant. Int J Adv Manuf Technol 39:1262–1270
20. Park SC (2008) PLCStudio: simulation based PLC code verification. In: Proceedings—winter simulation conference, pp 222–228
21. Quatrni T (1999) Visual modeling with Rational Rose 2000 and UML. Addison-Wesley, Reading
22. Rumbaugh J et al (1999) The unified modeling language reference manual. Addison-Wesley, Reading
23. Silva V (2005) Grid computing for developers (programming series). Charles River Media, Rockland
24. Trebicock B et al (2003) Virtual reality. Mod Mater Handl 85(5):55–60
25. Webb JW, Reis RA (2003) Programmable logic controllers. Prentice Hall, Engelwood Cliffs

Chapter 8
Augmented Reality for Embedded Systems

8.1 Embedded System Characteristics

A dedicated computer performing single or multiple functions in real time is known as embedded system. The embedded systems are used in variety of areas such as in aviation for autopilot of an airplane; in transportation systems for traffic control, and in manufacturing for machine tool control.

Single or multiple microcontrollers or Digital Signal Processors control the embedded system. Embedded systems have limited memory and processing power. These systems perform their function completely or partially independent of human intervention in most efficient manner. Most embedded systems are time critical in nature. One of the downfalls of the embedded system is that normally it can only be programed once. These systems are installed in systems where unreliability is not acceptable. The embedded system requires thorough testing before the system leaves the production line.

8.2 Real-Time Operating Systems

Real time operating system (RTOS) is the key element of an embedded system. It is a special type of operating system that nearly operates on real time based on scheduling as per the design of operation of the application. The scheduling at a RTOS relies on advance algorithms. The level of consistency in time table to accept and complete an application's task is paramount. This reflects the accuracy of the RTOS which ranges from hard, medium to soft. A RTOS that usually meets a deadline is a soft RTOS while RTOS that meets the deadline deterministically is a hard real-time OS. Major consideration in a RTOS is given to minimal interrupt latency and minimal thread switching latency.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 509
DOI: 10.1007/978-0-85729-186-8_8, © Springer-Verlag London Limited 2011

8.3 Embedded Systems in Augmented Reality Environment

Control gadgets based on embedded system in discrete virtual manufacturing environment are treated in similar manner as those of other control technologies. The input and output signals are copied to the virtual manufacturing controller minimizing process operation as augmented reality.

8.4 Augmented Reality Model for Embedded System

Tables 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, and 8.10 provide the detail of classes, methods, and variables related to the augmented reality for embedded system. The use case diagram, sequence diagram, and UML static diagram and the flow chart for the envisaged system are depicted in Figs. 8.1, 8.2, 8.3, and 8.4. A selection of sources with appropriate explanation is also provided. A factory clock is taken as an example. This embedded system is consists of a single project.

1. EmbeddedSystem: The embedded system project consists of Digits class, Colon class, Task class, TimeTask class, ProductionTimeTask class, TaskStack class, Kernel class, and Form1 class.

Digits class and Color class are inherited from System.Windows.Form.User-Control class. Digits and Colon are user control classes represent single digit and single colon in digital clock, respectively. Total eight Digits class objects, two for second, two for minute, two for hour, and two Colon class object, for separation of

Table 8.1 Summary of embedded system TaskStack class

Class summary			
Class ID: CE1			Namespace: EmbeddedSystem
Class name: TaskStack			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
Taskstack	static ArrayList	public	This field holds the array of the task stack
StackCount	static int	public	This field holds the stack count
Method name	Return type	Access modifier	Description
TaskIn(Task TaskToAdd)	static void	Public	Function to push a new task on the stack
TaskOut(int Index)	static Task	Public	Function to pop the task out of the stack
RemoveAll()	static void	Public	Function to remove all tasks from the stack
RemoveTask(int Index)	static void	Public	Function to remove the selected task from the stack
TaskStack()	–	public	Constructor function for the class

Table 8.2 Summary of embedded system Kernel class

Class summary			
Class ID: CE2			Namespace: EmbeddedSystem
Class name: Kernel			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None

Field name	Data type	Access modifier	Description
SystemClock	System.Windows.Forms.Timer	private	This field holds the system clock object
RunTimer	bool	private	This field holds the boolean value to check if timer is to be run
task	Task	private	This field holds the task object

Method name	Return type	Access modifier	Description
StartKernel()	void	public	Function to start kernel
StopKernel()	void	public	Function to stop kernel
ClockInterrupt(object sender, EventArgs e)	void	Private	Function to handle the clock interrupt event
PTTask_UpdateTotalProduction(TimeUnit Time, string Value)	void	Private	Function to handle the task update total production event for production task
PTTask_UpdateProduction(TimeUnit Time, string Value)	void	Private	Function to handle the task update production event for production task
TTask_Update(TimeUnit Time, string Value)	void	Private	Function to handle the task update event for task task
mainLoop()	void	public	Function to execute the main loop
Kernel()	–	public	Constructor function for the class

Event name	Delegate type	Access modifier	Description
MessageArrived	MessageHandler	public	Event to handle the message arrive event
UpdateTime	UpdateTimeHandler	public	Event to handle the update time event
UpdateProduction	UpdateTimeHandler	public	Event to handle the update production event
UpdateTotalProduction	UpdateTimeHandler	public	Event to handle the update total production event

Table 8.3 Summary of embedded system Task class

Class summary			
Class ID: CE3			Namespace: EmbeddedSystem
Class name: Task			Class type: Concrete
Is base class?: Yes			No. of inherited classes: 2
Inherited from: None			Interfaces with: IDisposable
Field name	Data type	Access modifier	Description
component	Component	private	This field holds the other managed resource this class uses
disposed	bool	private	This field tracks whether Dispose has been called
_TaskPriority	Priority	private	This field holds the priority value of the task
_TaskState	State	private	This field holds the state of the task
Terminate	bool	public	This field tracks whether to terminate the task

Method name	Return type	Access modifier	Description
Dispose()	void	private	Function to clear the task object
Dispose(bool disposing)	void	Private	Overriden Function to clear the task object
Task()	–	public	Constructor function for the class

Table 8.4 Summary of embedded system TimeTask

Class summary			
Class ID: CE4			Namespace: EmbeddedSystem
Class name: TimeTask			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: Task			Interfaces with: IContextMethod
Method name	Return type	Access modifier	Description
Execute()	void	public	Function to execute the time task
Event name	Delegate type	Access modifier	Description
Update	UpdateHandler	public	Function to handle the update event

hour and minute and minute and second, are required to display full digital clock. Digits class has a single property, Value to get and set its current value of digit (Fig. 8.5).

Task class is the parent class of TimeTask class and ProductionTimeTask class. Both TimeTask and ProductionTimeTask represent the task and have Execute method which kernel of embedded system has to perform by calling Execute method. Execute method of TimeTask and ProductionTimeTask has events UpdateProduction and UpdateTotalProduction to update user interface of embedded system for the current time of production clock and total production clock.

TaskStack class represents the stack of tasks. On each clock interrupt of Kernel class, one TimeTask and one ProductionTimeTask object are created and pushed on TaskStack for execution by Kernel.

Table 8.5 Summary of embedded system ProductionTimeTask class

Class summary			
Class ID: CE5			Namespace: EmbeddedSystem
Class name: ProductionTimeTask			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: Task			Interfaces with: IContextMethod
Field name	Data type	Access modifier	Description
Now	static DateTime	Private	This field holds the current time value
NextTime	static DateTime	private	This field holds the next time value
Method name	Return type	Access modifier	Description
Execute()	void	public	Function to execute the time task
Event name	Delegate type	Access modifier	Description
UpdateTotalProduction	UpdateHandler	public	Function to handle the update total production event
UpdateProduction	UpdateHandler	public	Function to handle the update production event

Kernel class represents the kernel of embedded system that executes Task by calling Task Execute method. On kernel loop, each Task pull from TaskStack and execute it and remove from TaskStack after execution.

Form1 class represents the user interface of embedded system.

Table 8.1 shows the member variables and methods of TaskStack class. TaskStack class is used to maintain the stack of task that Kernel of embedded system has to perform or execute.

Table 8.2 shows the member variables, methods, and events of Kernel class. Kernel class represents the kernel of embedded system and is used to execute task. Kernel gets system clock interrupt from Timer. On each clock interrupt, tasks are created and executed by Kernel.

Table 8.3 shows the member variables and methods of Task class. The Task class is the base class containing the generalized fields and methods related to task which is inherited by TimeTask and ProductionTimeTask classes.

Table 8.4 shows the member method and Event of TimeTask class. The TimeTask class represents the factory time, executed by embedded system kernel.

Table 8.5 shows the member method and Event of ProductionTimeTask class. The ProductionTimeTask class represents production time, executed by embedded system kernel.

Table 8.6 shows the member variables and methods of Colon class. Colon class represents the separator between hours and minutes and between minutes and seconds. Total two objects of Colon class are required to present digital clock.

Table 8.6 Summary of embedded system Colon class

Class summary			
Class ID: CE6			Namespace: EmbeddedSystem
Class name: Colon			Class type: partial
Is base class?: No			No. of inherited classes: 0
Inherited from: UserControl			Interfaces with: None
Field name	Data type	Access modifier	Description
timer1	Timer	private	This field holds the timer object
bColorControl	bool	private	This field tracks whether color control is active
segPoints	Point[][]	private	This field holds the 2D array of the segment points
gridHeight	int	private	This field holds the height of the grid
gridWidth	int	private	This field holds the width of the grid
elementWidth	int	private	This field holds the Width of LED segments
colorBackground	Color	private	This field holds the Background color of the seven-segment display
colorDark	Color	private	This field holds the Color of inactive LED segments
colorLight	Color	private	This field holds the Color of active LED segments
customPattern	int	private	This field Sets a custom bit pattern to be displayed on the seven segments. This is an integer value where bits 0–6 correspond to each respective LED segment

Method name	Return type	Access modifier	Description
timer1_Tick(object sender, EventArgs e)	void	private	Function to handle the timer tick event
Colon_Resize(object sender, EventArgs e)	void	private	Function to handle the colon resize event
OnPaddingChanged(EventArgs e)	void	protected override	Function to handle the On padding changed event
OnPaintBackground(PaintEventArgs e)	void	protected override	Function to handle the On paint background event
Colon_Paint(object sender, PaintEventArgs e)	void	private	Function to handle the paint event
Colon()	–	public	Constructor function for the class

Table 8.7 Summary of embedded system Digits class

Class summary			
Class ID: CE7		Namespace: EmbeddedSystem	
Class name: Digits		Class type: partial	
Is base class?: No		No. of inherited classes: 0	
Inherited from: UserControl		Interfaces with: None	
Field name	Data type	Access modifier	Description
italicFactor	float	private	This field holds the Shear coefficient for italicizing the displays. Try a value like −0.1
ValuePattern	enum	public	These are the various bit patterns that represent the characters that can be displayed in the seven segments. Bits 0 through 6 correspond to each of the LEDs, from top to bottom!
theValue	string	private	This field holds the Character to be displayed on the seven segments. Supported characters are digits and most letters
segPoints	Point[][]	private	This field holds the 2D array of the segment points
gridHeight	int	private	This field holds the height of the grid
gridWidth	int	private	This field holds the width of the grid
elementWidth	int	private	This field holds the Width of LED segments
colorBackground	Color	private	This field holds the Background color of the 7-segment display
colorDark	Color	private	This field holds the Color of inactive LED segments
colorLight	Color	private	This field holds the Color of active LED segments

(continued)

Table 8.7 (continued)

Field name	Data type	Access modifier	Description
customPattern	int	private	This field Set a custom bit pattern to be displayed on the seven segments. This is an integer value where bits 0 through 6 correspond to each respective LED segment
showDot	bool	private	Specifies if the decimal point LED is displayed
dotOn	bool	private	Specifies if the decimal point LED is active

Method name	Return type	Access modifier	Description
RecalculatePoints()	void	private	Function to Recalculate the points that represent the polygons of the seven segments, whether we are just initializing or changing the segment width
SevenSegment_Resize(object sender, EventArgs e)	void	private	Function to handle the segment resize event
OnPaddingChanged(EventArgs e)	void	protected override	Function to handle the On padding changed event
OnPaintBackground(PaintEventArgs e)	void	protected override	Function to handle the On paint background event
SevenSegment_Paint(object sender, PaintEventArgs e)	void	private	Function to handle the paint event
Digits()	–	public	Constructor function for the class

Table 8.8 Summary of embedded system sequence diagram

Sequence diagram summary				
Sequence ID: SD-E1 Sequence name: Embedded System Sequence Diagram				
Method name	Type	Description	Source	Destination
Start kernel	Self Message	Function to start the kernel process for handling clock interrupts	kernel	kernel
Clock Interrupt	Self Message	Function to call a clock interrupt	kernel	kernel
Create Time Task	Return Message	Function to create a new time task	kernel	TimeTask
Create Production Time Task	Return Message	Function to create a new production time task	TimeTask	ProductionTimeTask
Create Task	Return Message	Function to create a new task	TimeTask, ProductionTimeTask	Task
Add Task in Stack (Task in)	Message	Function to push a new task in stack	kernel	Task Stack
Start kernel loop	Self Message	Function to start the kernel loop	kernel	kernel
Get Task (Task Out)	Return Message	Function to pop a task from the stack	kernel	Task Stack
Execute Task	Self Message	Function to execute task	kernel	kernel
Delete Task (Remove Task)	Message	Function to remove task	kernel	Task Stack
Update Clock	Message	Function to update clock	kernel	Form
Update Digits	Self Message	Function to update digits in a loop	Form	Form
Remove Task	Self Message	Function to remove a task	Task Stack	Task Stack

Table 8.9 Summary of embedded system Start Application use case

Use case summary
Use case ID: UC-E1
Use case name: Start Application
Actors: User
Description: The user executes the application
Preconditions: None
Post conditions: The screen shows the factory time, total production hours and production elapsed time in digital format
System parameters: Kernel
Success scenario: The screen is loaded with required time information
Alternative scenario: The system gives an error and exits application
Includes: Start kernel loop, kernel control interrupt
Priority: High

Table 8.10 Summary of embedded system flow charts

Flow chart summary	
Flow chart ID: FC-E1	
Flow chart name: Embedded System Process Flow Chart	
No. of inputs: 1	
No. of processes: 13	
No. of outputs: 1	
No. of control decisions: 0	
Input name	Description
Start Kernel	User starts application
Process name	Description
Clock Interrupt	Function to call a clock interrupt
Create Time Task	Function to create time task
Add Task in Task Stack	Function to add a new task
Out In	Function to push a new task on the stack
Task Stack	Function to set the task stack object
Execute Task	Function to execute task
Out Task	Function to pop the task from the stack
Kernel Loop	Main Loop for kernel tasks
Remove Task	Function to remove the selected task
Update Clock and Production Time	Function to Update clock and production time
Create Production Time Task	Function to create production time task
Add Production Task in Task Stack	Function to add production time task on stack
Start kernel main loop	Start the main loop for kernel
Output name	Description
Graphics display for digital clock	Digital clocks with time displayed on screen

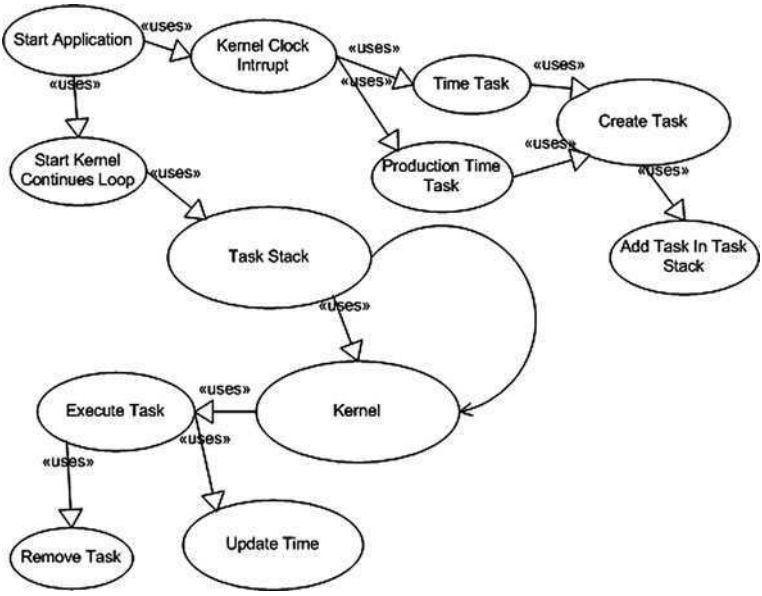


Fig. 8.1 Use case diagram for embedded system

Table 8.7 shows the member variables and methods of Digits class. Digits class represents the single digit on digital clock. Total six objects of Digits class are required to present digital clock.

Table 8.8 shows the process of embedded system sequence diagram.

Table 8.9 defines the embedded system Start Application use case.

Table 8.10 portrays the embedded system processes flow charts.

The following partial code is taken from Factory clock embedded system. In the following code, it defines internal interface IContextMethod, which has only one method Execute() signature. ProductionTimeTask class and TimeTask class are inherited from Task class as well as from IContextMethod interface.

UpdateHandler is a delegate with two parameter, Time and time value used to update the digital clock values.

In ProductionTime class, two DateTime objects Now and NextTime are initialized with 12:0:0 and 0:0:0, respectively. Execute method called by embedded system Kernel, subtracts 1 s from Now time to create elapsed production time and adds 1 s in NextTime to create total production time in each call of Execute method. After getting both times, UpdateTotalProduction and UpdateProduction events are raised to update the embedded system clock.

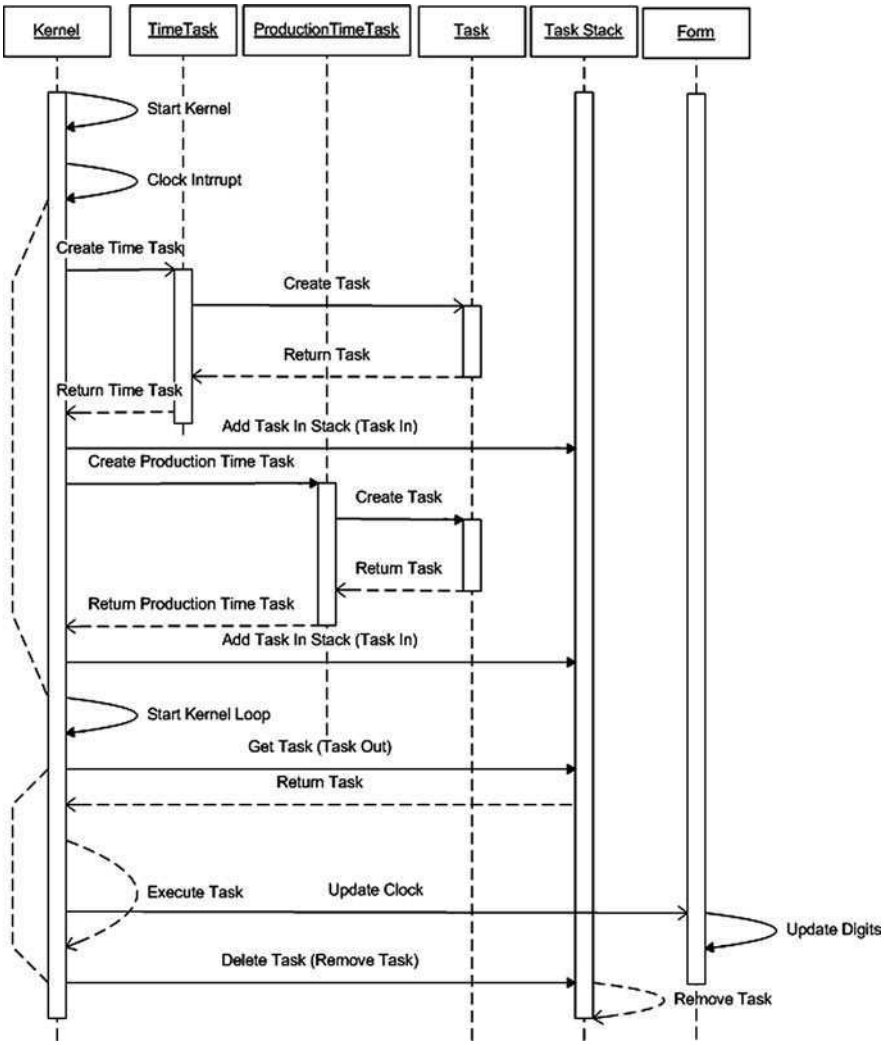


Fig. 8.2 Sequence diagram for embedded system

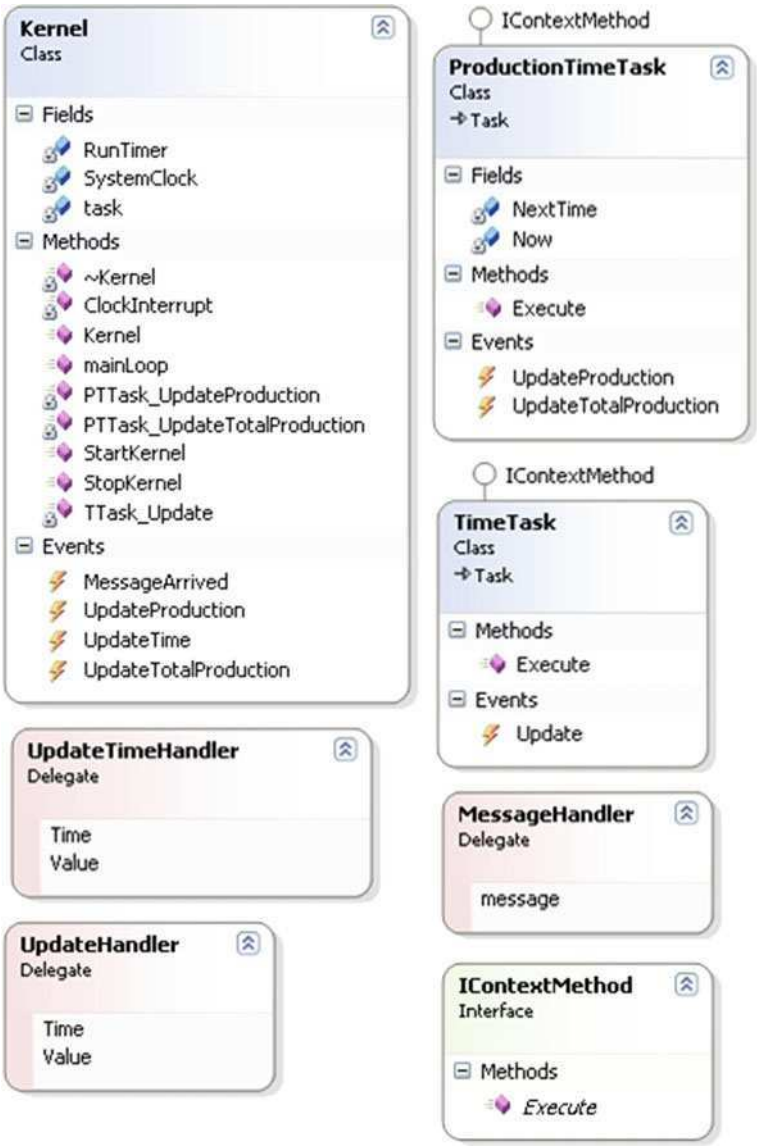


Fig. 8.3 Static diagram for embedded system

Fig. 8.3 (continued)

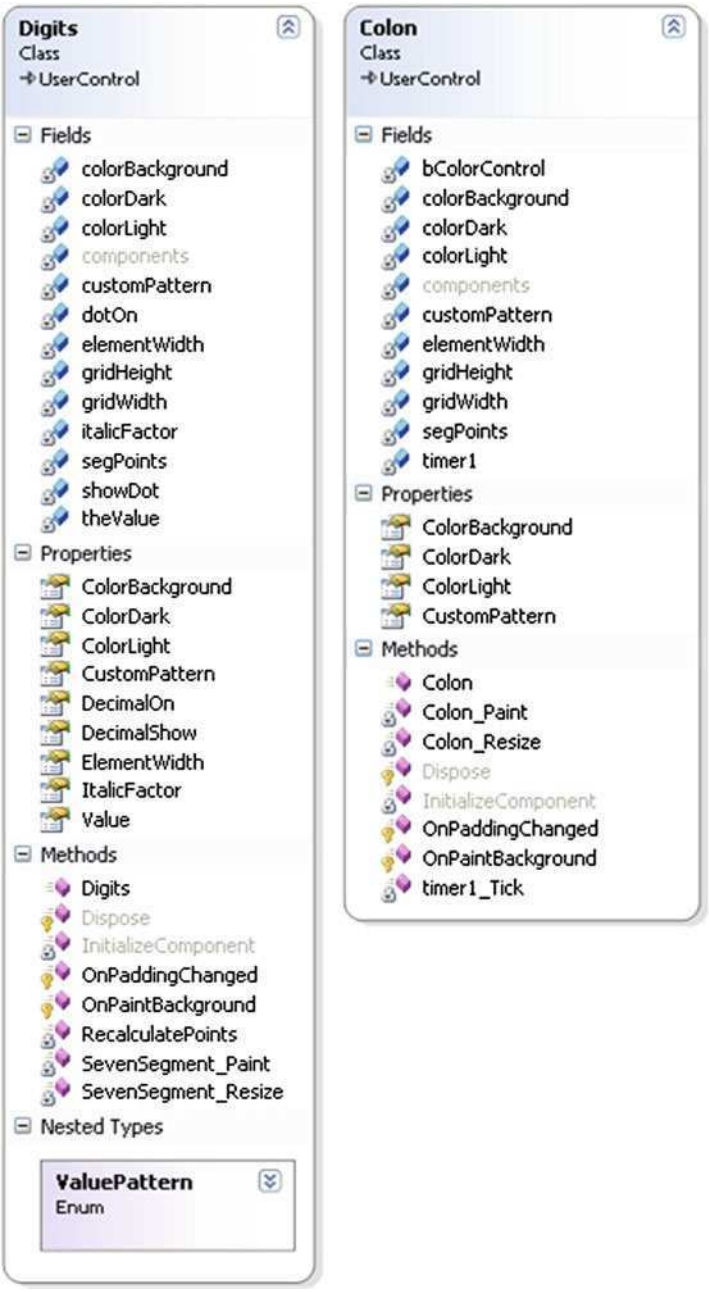


Fig. 8.3 (continued)

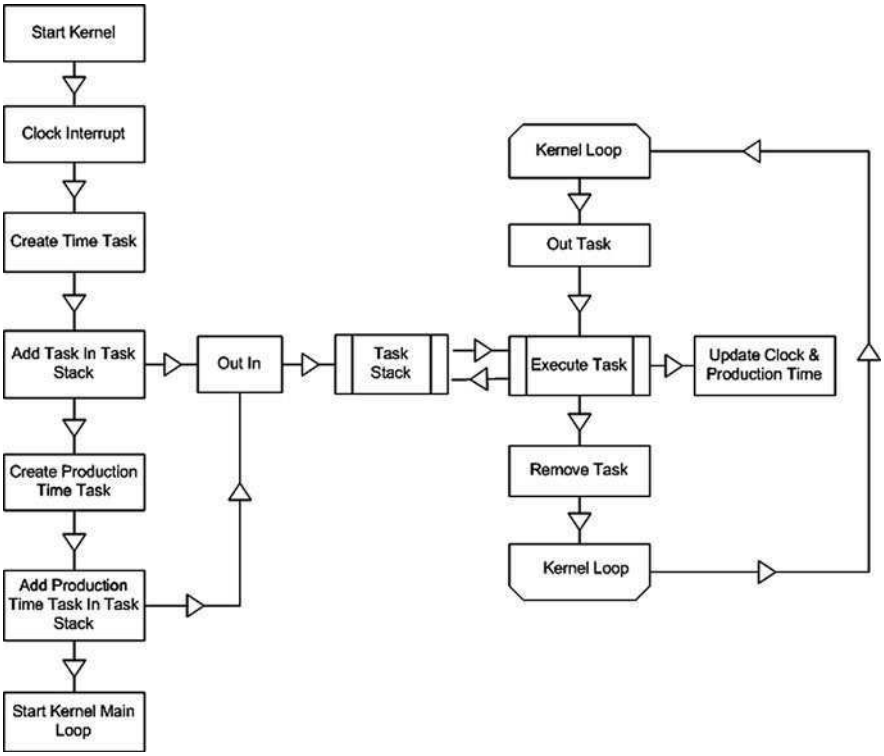


Fig. 8.4 Typical flow chart for embedded system

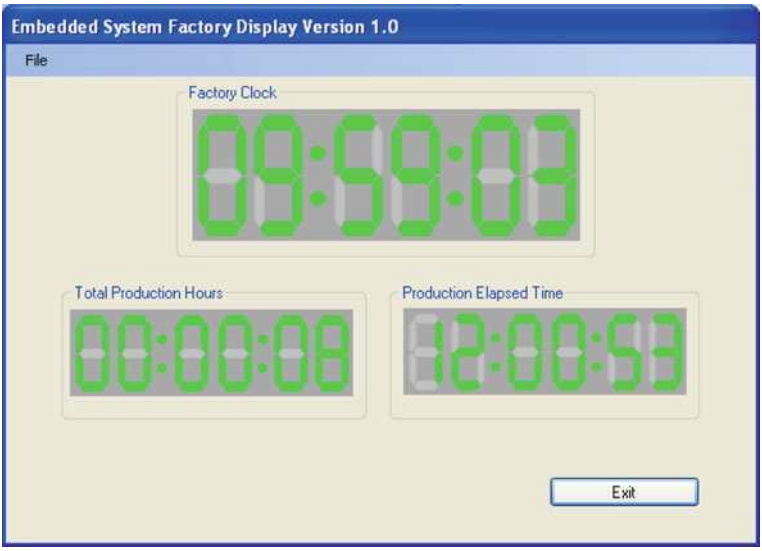


Fig. 8.5 Augmented reality for embedded system

```
internal interface IContextMethod
    {
        void Execute();
    }
    public delegate void UpdateHandler(TimeUnit Time, string Value);

    public class ProductionTimeTask : Task, IContextMethod
    {
        static DateTime Now = new DateTime(DateTime.Now.Year,DateTime.Now.Month,DateTime.Now.Day,
12, 1, 1);
        static DateTime NextTime = new  DateTime(DateTime.Now.Year,DateTime.Now.Month,
DateTime.Now.Day, 0, 0, 0);

        public event UpdateHandler UpdateTotalProduction;
        public event UpdateHandler UpdateProduction;

        public void Execute()
        {
            try
            {

                DateTime NextNow = new DateTime(Now.Year, Now.Month, Now.Day, 0, 0, 1);
                TimeSpan InnerNow = Now.Subtract(NextNow);
                Now = new DateTime(Now.Year, Now.Month, Now.Day, InnerNow.Hours,
InnerNow.Minutes, InnerNow.Seconds);
                DateTime InnerNextTime = NextTime.AddSeconds(1);
                NextTime = InnerNextTime;
                UpdateTotalProduction(TimeUnit.Hour, InnerNextTime.Hour.ToString());
                UpdateTotalProduction(TimeUnit.Minute, InnerNextTime.Minute.ToString());
                UpdateTotalProduction(TimeUnit.Second, InnerNextTime.Second.ToString());
                UpdateProduction(TimeUnit.Hour, InnerNow.Hours.ToString());
                UpdateProduction(TimeUnit.Minute, InnerNow.Minutes.ToString());
                UpdateProduction(TimeUnit.Second, InnerNow.Seconds.ToString());
            }
            catch (Exception exp)
            {
                MessageBox.Show(exp.Message);
            }

        }
    }
```

The following code is taken from TimeTask class of embedded system. In TimeTask class, Update event is defined for update the factory clock of embedded system. In Execute method called by Kernel, a DateTime object Now is initialized with the current system time. After initializing Now object, Execute method separates the seconds, minutes, and hours from Now time and raised Update event to update digital clock values.

```
public class TimeTask : Task, IContextMethod
    {
        public event UpdateHandler Update;

        public void Execute()
        {
            DateTime Now = DateTime.Now;
            String Sec = Now.Second.ToString();
            String Min = Now.Minute.ToString();
            String Ho = Now.Hour.ToString();
            Update(TimeUnit.Hour, Ho);
            Update(TimeUnit.Minute, Min);
            Update(TimeUnit.Second, Sec);


        }

    }
```

The following code is taken from embedded system Task class. In the constructor of Task class, it sets the default priority of task to medium and task state to

ready. After constructor, Task class has definition of generalized properties to set Task state. In the last of Task class, it defines Dispose method to clean terminated task from system memory.

```
public class Task : IDisposable
    {
        private Component component = new Component();// Other managed resource this class uses.
        private bool disposed = false;// Track whether Dispose has been called.

        private Priority _TaskPriority;
        private State _TaskState;
        public bool Terminate = false;

        public Task()
        {
            this._TaskPriority = Priority.Medium;
            this._TaskState = State.Ready;
        }
        #region Properties
        public bool IsTerminated
        {
            get
            {
                return Terminate;
            }

        }
        public Priority TaskPriority
        {
            get
            {
                return _TaskPriority;
            }
            set
            {
                _TaskPriority = value;
            }

        }
        public State TaskState
        {
            set
            {
                _TaskState = value;
            }
            get
            {
                return _TaskState;
            }
        }
        #endregion

        #region IDisposable Members

        public void Dispose()
        {
            Dispose(true);
            // This object will be cleaned up by the Dispose method.
            GC.SuppressFinalize(this);
        }
        private void Dispose(bool disposing)
        {
            // Check to see if Dispose has already been called.
            if (!this.disposed)
            {
                // If disposing equals true, dispose all managed
                // and unmanaged resources.
                if (disposing)
                {
                        // Dispose managed resources.
                        component.Dispose();
                    }
                }
                disposed = true;
            }

            #endregion
        }
```

The following code is taken from Kernel class of embedded system. Kernel class uses two delegates to define two events one for Kernel messages and one for update the factory clock. In class definition, it creates Task object for holding TimeTask class object and TotalTimeTask class object from TaskStack on temporarily basis.

In the constructor of Kernel, it creates and initialized system clock of type Timer and sets it clock interval to 1 s. The methods StartKernel and StopKernel start and stop system clock, respectively.

In the ClockInterrupt method of Kernel, two objects of TaskTime class and TotalTaskTime are created and push to TaskStack array and call MainLoop method.

In the MainLoop method, it loop through the number of entries on TaskStack and pull Task objects from TaskStack and calls Task's Execute method to execute each task on TaskStack and raise appropriate events to update factory clock.

```
public delegate void MessageHandler(string message);
    public delegate void UpdateTimeHandler(TimeUnit Time,string Value);
    class Kernel
    {
        private System.Windows.Forms.Timer SystemClock;
        private bool RunTimer = true;

        Task task = new Task();
        public event MessageHandler MessageArrived;
        public event UpdateTimeHandler UpdateTime;
        public event UpdateTimeHandler UpdateProduction;
        public event UpdateTimeHandler UpdateTotalProduction;

        public Kernel()
        {
            #region System Clock
            SystemClock = new System.Windows.Forms.Timer();
            SystemClock.Interval = 1000;
            SystemClock.Tick += new EventHandler(ClockInterrupt);
            #endregion
        }

        public void StartKernel()
        {
            RunTimer = true;
            SystemClock.Enabled = true;
            SystemClock.Start();

        }
        public void StopKernel()
        {
            RunTimer = false;
            MessageArrived("Stop of Kernel");
        }

        private void ClockInterrupt(object sender, EventArgs e)
        {

            MessageArrived("Clock Interrupt");

            TimeTask TTask = new TimeTask();
            TTask.Update += new UpdateHandler(TTask_Update);
            TaskStack.TaskIn((Task)TTask);

            ProductionTimeTask PTTask = new ProductionTimeTask();
            PTTask.UpdateProduction += new UpdateHandler(PTTask_UpdateProduction);
            PTTask.UpdateTotalProduction += new UpdateHandler(PTTask_UpdateTotalProduction);
            TaskStack.TaskIn((Task)PTTask);
            Application.DoEvents();
            if (RunTimer)
            {
                mainLoop();

            }
```

```
        }

        void PTTask_UpdateTotalProduction(TimeUnit Time, string Value)
        {
            UpdateTotalProduction(Time, Value);
        }

        void PTTask_UpdateProduction(TimeUnit Time, string Value)
        {
            UpdateProduction(Time, Value);

        }

        void TTask_Update(TimeUnit Time, string Value)
        {

            UpdateTime(Time, Value);
        }
        ~Kernel()
        {
            SystemClock.Stop();
        }


        public void mainLoop()
        {
             try
            {
                //int StackCounter = TaskStack.StackCount;

                for (int Index = 0; Index < TaskStack.StackCount; Index++)
                {
                    //Getting task from task stack
                    task = TaskStack.TaskOut(Index);
                    //Start Executing Task
                    MessageArrived("Start Executing Task");
                    ((IContextMethod)task).Execute();
                    task.Dispose();
                    MessageArrived("Removing Task");
                    //removing task from task stack after executing task
                    TaskStack.RemoveTask(Index);
                    GC.Collect();


                }

            }
            catch (Exception e)
            {
                MessageBox.Show("Error in Kernel Module" + e.Message);
            }
        }

    }
```

The following code is taken from Form1 class. Form1 class has subscription for Kernel's UpdateTime event. UpdateTime event has two parameters, one for time unit and one for time value. First it checks TimeUnit value by using switch case statement. TimeUnit can have one of three values Second, Minute, or Hour. After making decision of TimeUnit, it sets the value of Digits class objects to set the factory digital clock.

```
void kernel_UpdateTime(TimeUnit Time, string Value)
        {
            //Kernel Event for Update Factory Clock
            switch (Time)
            {
                case TimeUnit.Second:
                    //Update Factory Time Second
                    if (Value.Length == 1)
                    {
                        S2.Value = "0";
                        S1.Value = Value;
                    }
                    else
                    {
                        S2.Value = Value.Substring(0, 1);
                        S1.Value = Value.Substring(1, 1);

                    }
                   break;
                case TimeUnit.Minute:
                    //Update Factory Time Minute
                    if (Value.Length == 1)
                    {
                        M2.Value = "0";
                        M1.Value = Value;
                    }
                    else
                    {
                        M2.Value = Value.Substring(0, 1);
                        M1.Value = Value.Substring(1, 1);

                    }

                    break;
                case TimeUnit.Hour:
                    //Update Factory Time Hour
                    if (Value.Length == 1)
                    {
                        H2.Value = "0";
                        H1.Value = Value;
                    }
                    else
                    {
                        H2.Value = Value.Substring(0, 1);
                        H1.Value = Value.Substring(1, 1);

                    }

                    break;
                default:
                    break;
            }
        }
```

8.5 Interface Design for System Integration

Figure 8.6 shows hardware interface design for AR for embedded systems. The data sheet for the micro controller is provided as Appendix IV. The power supply circuit used in conjunction with the main circuit is shown in Appendix V.

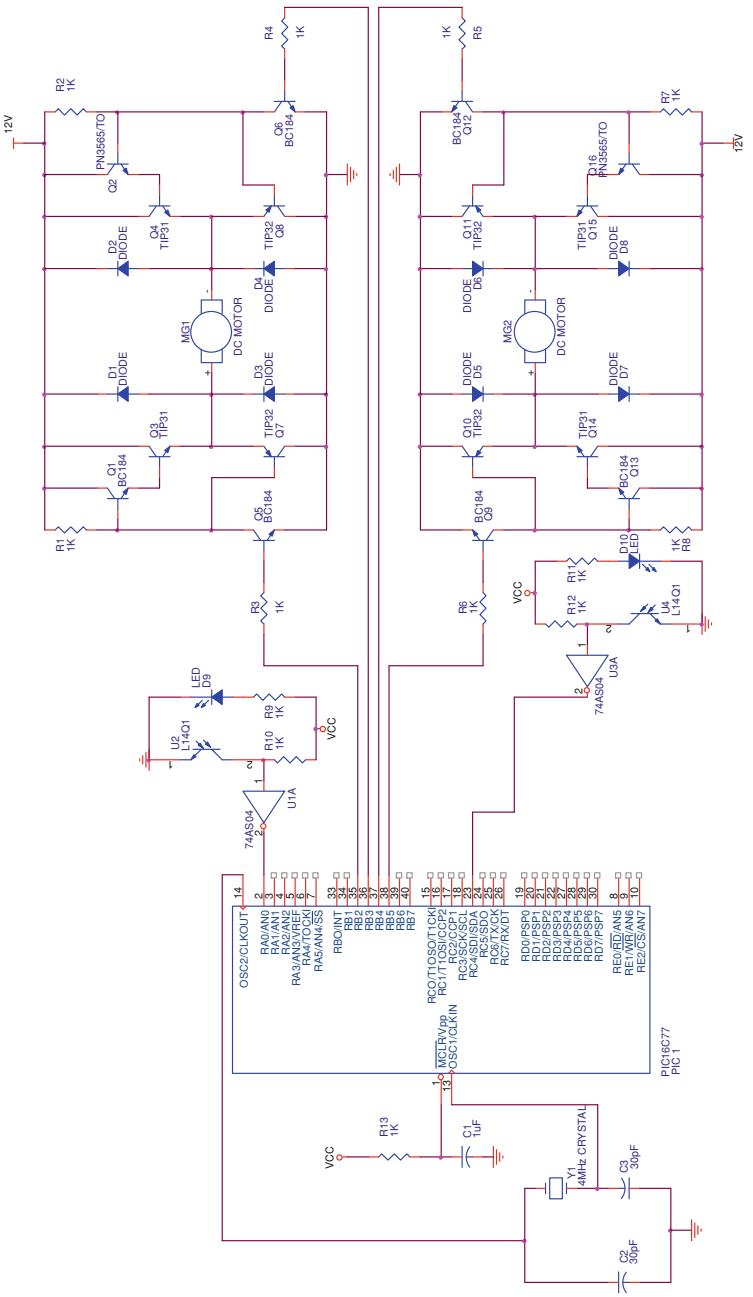


Fig. 8.6 Control of embedded system over local area network (continued)

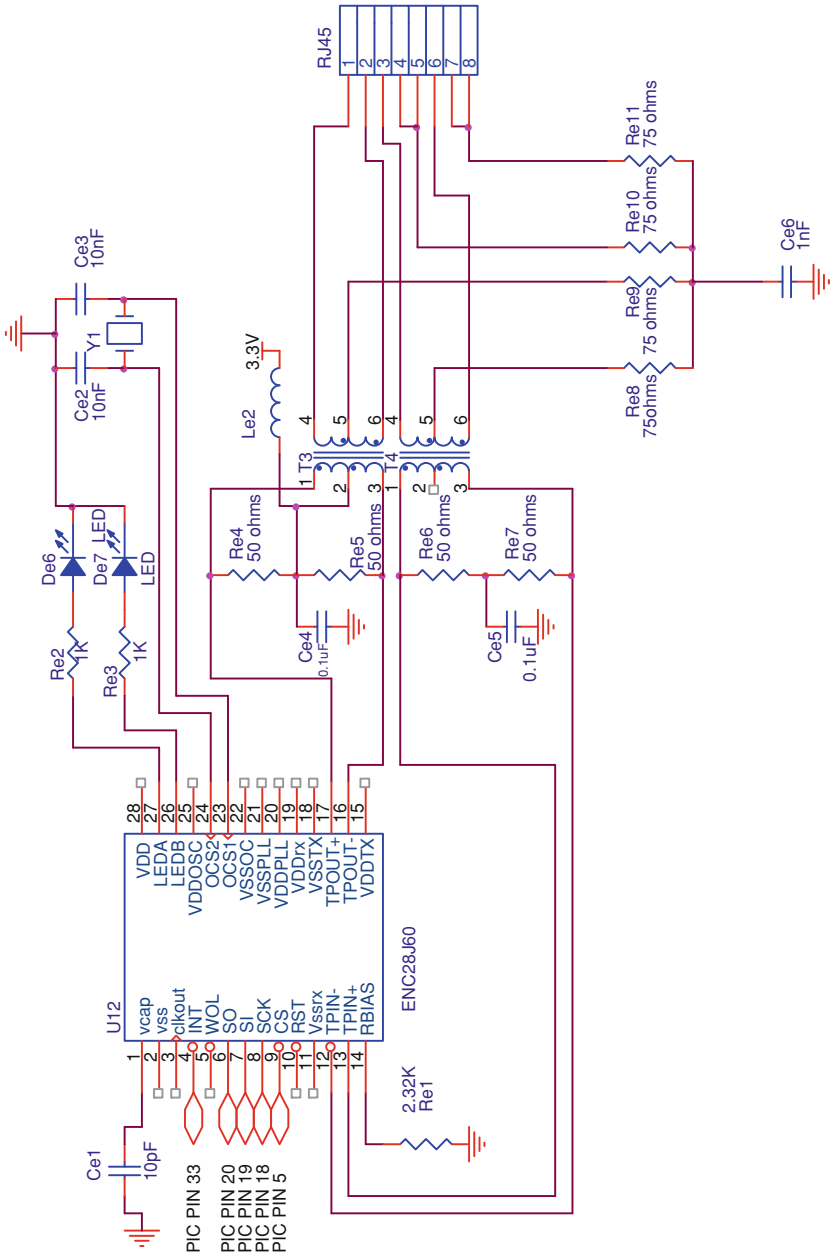


Fig. 8.6 (continued)

Bibliography

1. Cheng X (2003) Software programming of an embedded virtual instrument system. In: Proceedings of the international symposium on test and measurement, pp 4695–4698
2. Cho SY (2007) A flexible virtual development environment for embedded system. In: Proceedings of the 5th international workshop on intelligent solutions in embedded systems, pp 37–47
3. Hori T (2009) Improvement of reality on input device for virtual walk system. In: ICCAS-SICE, pp 1100–1104
4. http://en.wikipedia.org/wiki/Embedded_system
5. Koppen V (2009) An architecture for interoperability of embedded systems and virtual reality. IETE Tech Rev 26:350–356
6. Li X (2009) The research and development of fitness equipment embedded system based on distributed virtual environment. In: Proceedings—2009 international conference on information technology and computer science, pp 28–31
7. Mauro AD (2009) Virtual reality training embedded in neurosurgical microscope. In: Proceedings—IEEE virtual reality, pp 233–234
8. Sung RCW (2009) Automated design process modeling and analysis using immersive virtual reality. CAD Comput Aided Des 41(12):1082–1094
9. Tsang WM (2005) A finger-tracking virtual mouse realized in an embedded system. In: Proceedings of 2005 international symposium on intelligent signal processing and communication systems, pp 781–784
10. Xiong Z (2005) Virtual embedded operating system for hardware/software co-design. ASICON 6th international conference on ASIC, pp 858–861

Chapter 9
Augmented Reality for Supervisory Control and Data Acquisition-Based Application

9.1 Characteristics of SCADA-Based System

The supervisory control and data acquisition (SCADA) based system is characterized by its operation at very large distances and in most cases by real time operation. SCADA facilitate operation in areas where human access is difficult or impossible. The basic components of the SCADA are Sensors, Transducers and Actuators. Remote Terminal Unit (RTU) conveys signals from sensors and transducers through leased telephone lines, data communication network, or through satellite communication to Master Terminal Unit (MTU): host computer or server.

SCADA-based systems are used in diverse areas such as

1. Oil and Gas Production Facilities;
2. Pipelines for Gas, Oil, Chemical or Water;
3. Waste Water Collection and Treatment;
4. Potable Water Treatment and Distribution;
5. Electrical Transmission System;
6. Large Communication System;
7. Irrigation System;
8. Group of Small Hydroelectric Generating Stations;
9. Weather Forecasting System;
10. Large process based continues Manufacturing System, e.g., Refineries, Sugar production plants, Cement Production Plant, etc.;
11. Large Flow Lines, etc.

9.2 Augmented Reality for SCADA-Based System

Augmented Reality (AR) for SCADA-based system is in reality used in many applications outlined in Sect. 9.1. The technology used may not be utilizing the

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 533
DOI: 10.1007/978-0-85729-186-8_9, © Springer-Verlag London Limited 2011

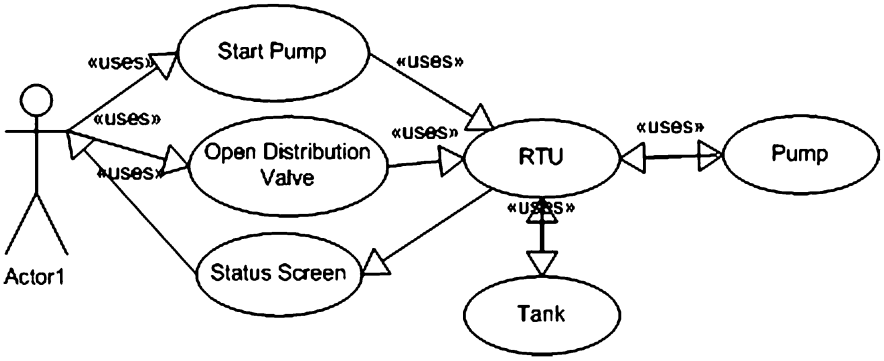


Fig. 9.1 SCADA use case diagram

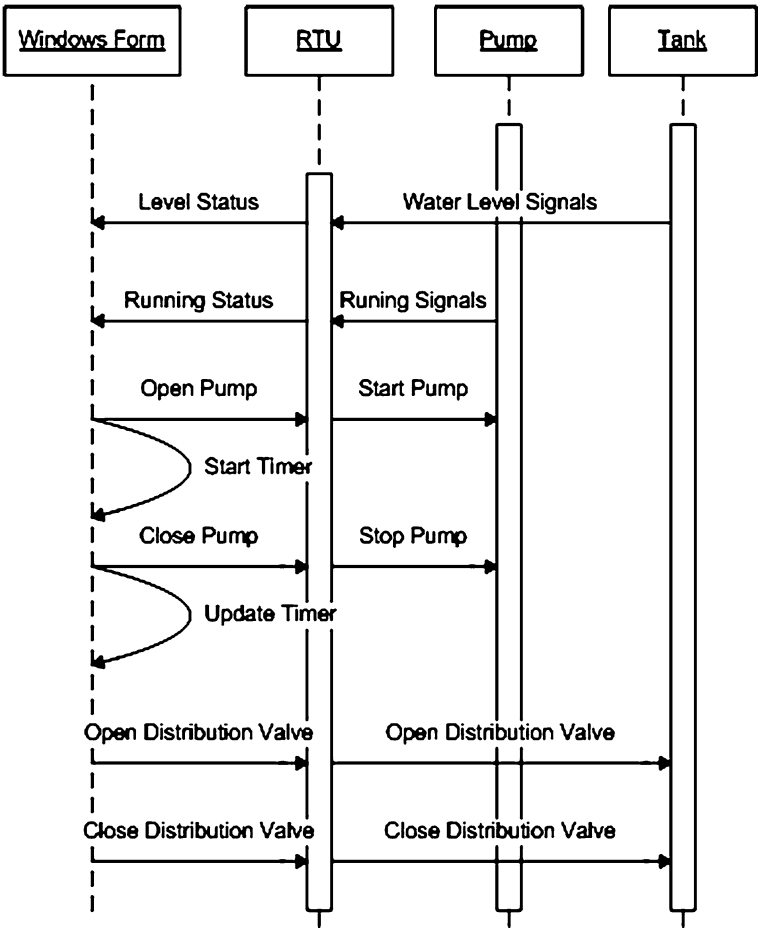


Fig. 9.2 SCADA processes sequence diagram

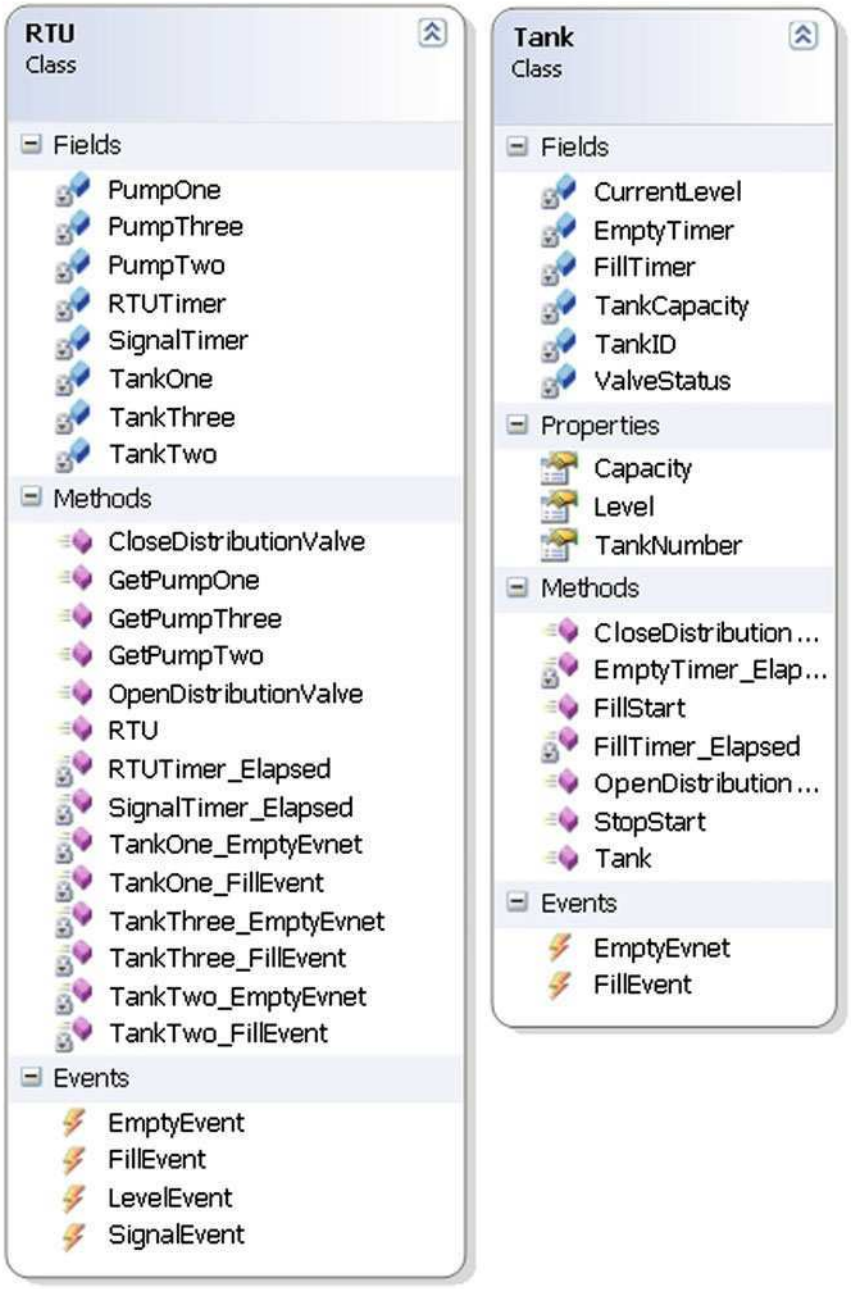


Fig. 9.3 SCADA classes, delegates, and enumeration static diagram

Fig. 9.3 (continued)

modern age human–computer interface. In this section, the use of AR for a section of potable water distribution system is detailed. The use case diagram, UML sequence diagram, and UML Static diagram are presented in Figs. 9.1, 9.2, and 9.3. The flow chart to run AR for SCADA system is given in Fig. 9.4. The graphical user interface for this case is provided in Fig. 9.5. Tables 9.1, 9.2, 9.3,

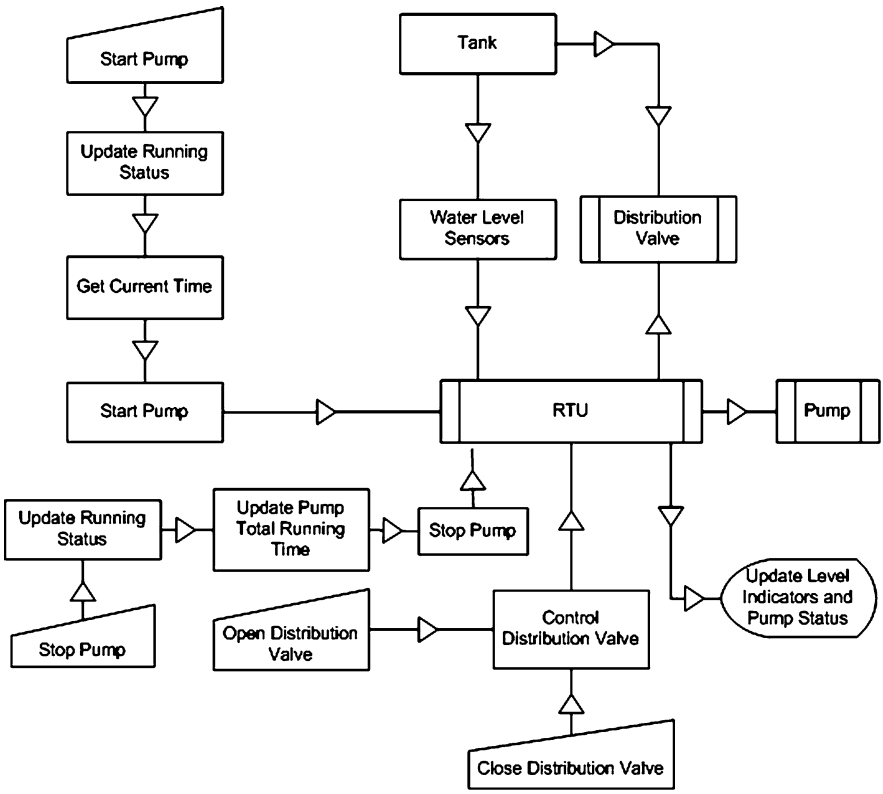


Fig. 9.4 SCADA processes flow chart diagram

9.4, 9.5, 9.6, 9.7 provide the summary of the UML use case, sequence diagrams, and static diagrams. A section of c# code is also described in this section.

SCADA consists of three projects:

1. Field: consists of two classes—Pump class, Tank class—and two delegates. Tank class represents water tank and Pump class represents the water pump. Objects of Tank class and Pump class are created in RemoteTerminalUnit to provide water pump and tank functionality for SCADA system. Field is Dynamic Link Library (DLL) based project. The produced DLL of Field will be used in RemoteTerminalUnit to provide separation between logic and logic encapsulation.

   Pump and Tank classes have encapsulated all the main functionality required by RemoteTerminalUnit as Open Distribution Valve and Close Distribution Valve. A request from RemoteTerminalUnit for each function is handled by Pump class. Pump and Tank object are tightly coupled, because for each Pump object,

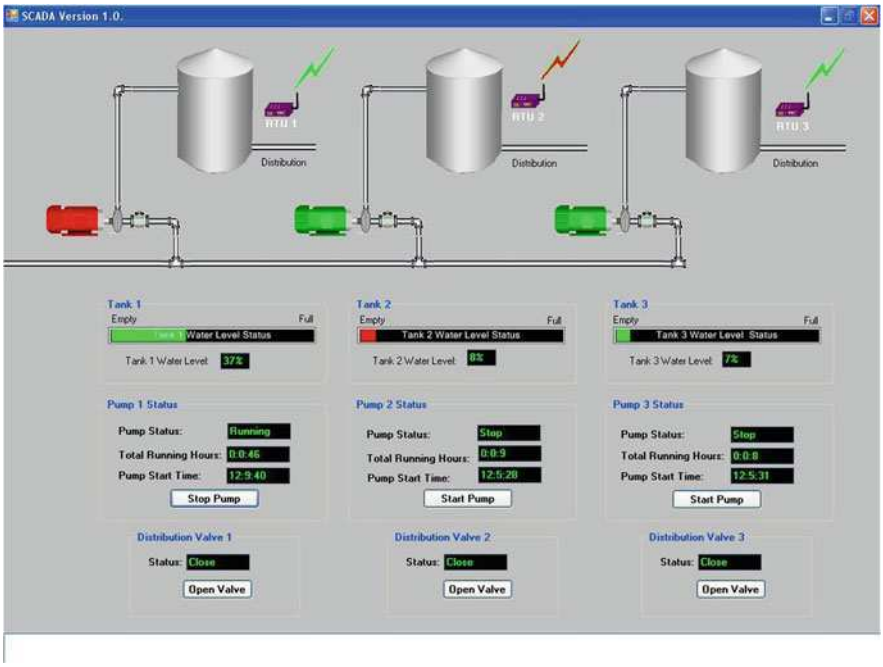


Fig. 9.5 SCADA for portable water distribution system

Table 9.1 Summary of SCADA Pump class

Class summary			
Class ID: CA1			Namespace: Field
Class name: Pump			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	**Data type**	**Access modifier**	**Description**
PumpStatus	bool	private	This field holds the status of the pump
AssociatedTank	Tank	private	This field holds the associated tank with the pump
Method name	**Return type**	**Access modifier**	**Description**
Start()	void	Public	Function to start the pumping process
Stop()	void	Public	Function to stop the pumping process
GetPumpStatus()	bool	Public	Function to return the pump status
Pump (Tank AssociateTank)	–	Public	Constructor function for the class

there must be a Tank object. Tank class has two events EmptyEvent is raised when tank is empty and FillEvent is raised when tank is full to update RemoteTerminalUnit object.

Table 9.2 Summary of SCADA Tank class

Class summary			
Class ID: CA2			Namespace: Field
Class name: Tank			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
FillTimer	Timer	private	This field holds the fill timer object
EmptyTimer	Timer	private	This field holds the empty timer object
TankID	Int	private	This field holds the tank number
TankCapacity	Int	private	This field holds the total tank capacity
CurrentLevel	Int	private	This field holds the current water level of the tank
ValveStatus	bool	private	This field holds the status of the distribution valve
Method name	Return type	Access modifier	Description
EmptyTimer_Elapsed (object sender, ElapsedEventArgs e)	void	Private	Function handle the empty timer event
FillTimer_Elapsed (object sender, ElapsedEventArgs e)	void	Private	Function to handle the fill timer event
OpenDistributionValve()	void	Public	Function to Open Distribution Valve
CloseDistributionValve()	void	Public	Function to Close Distribution Valve
Tank(int Capacity, int TankNumber)	–	Public	Constuctor function for the class
Event name	Delegate type	Access modifier	Description
FillEvent	TankFillHandler	Public	Event to handle the tank filling process
EmptyEvnet	TankEmptyHandler	Public	Event to handle the tank emptying process

2. RemoteTerminalUnit: The RemoteTerminalUnit consists of a RTU class, one enumeration and four events. The objects of Pump class and Tank class are created in RTU class. The object of RTU class is initialized in SCADA application to provide access to tank and pump functionality. Information from EmptyEvent and FillEvent of Tank class is passed to SCADA application by RTUs own events EmptyEvent and FillEvent. RTU class has two more events, LevelEvent and SignalEvent for level of water and RTU signals. All the four events are used to update the interface of SCADA application.
3. SCADA: consists of a class Form1 which is inherited by System.Windows. Form provides SCADA graphical user interface and utilizes RTU object to control Pump and valve to fill and distribute water. Current status of filling and distribution of water is also updated from SCADA project.

Table 9.3 Summary of SCADA RTU class

Class summary			
Class ID: CA3			Namespace: RemoteTerminalUnit
Class name: RTU			Class type: Concrete
Is base class? No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
TankOne	Tank	private	This field holds the tank 1 object
TankTwo	Tank	private	This field holds the tank 2 object
TankThree	Tank	private	This field holds the tank 3 object
PumpOne	Pump	private	This field holds the pump 1 object
PumpTwo	Pump	private	This field holds the pump 2 object
PumpThree	Pump	private	This field holds the pump 3 object
RTUTimer	Timer	private	This field holds the RTU timer object
SignalTimer	Timer	private	This field holds the signal timer object

Method name	Return type	Access modifier	Description
SignalTimer_Elapsed(object sender, ElapsedEventArgs e)	void	private	Function to handle the signal timer event
TankThree_EmptyEvnet(string message)	void	Private	Function to handle the tank 3 empty timer event
TankTwo_EmptyEvnet(string message)	void	Private	Function to handle the tank 2 empty timer event
TankOne_EmptyEvnet(string message)	void	Private	Function to handle the tank 1 empty timer event
TankOne_FillEvent(string message)	void	Private	Function to handle the tank 1 fill timer event
TankThree_FillEvent(string message)	void	Private	Function to handle the tank 3 fill timer event
TankTwo_FillEvent(string message)	void	Private	Function to handle the tank 2 fill timer event
RTUTimer_Elapsed(object sender, ElapsedEventArgs e)	void	Private	Function to handle the RTU timer event
GetPumpOne()	void	public	Function to return the pump 1 object
GetPumpTwo()	void	public	Function to return the pump 2 object
GetPumpThree()	void	public	Function to return the pump 3 object

(continued)

Table 9.3 (continued)

Method name	Return type	Access modifier	Description
OpenDistributionValve(int TankNumber)	void	public	Function to Open Distribution Valve
CloseDistributionValve(int TankNumber)	void	public	Function to Close Distribution Valve
RTU()	–	public	Constructor function for the class

Event name	Delegate type	Access modifier	Description
FillEvent	TankFillHandler	public	Event to handle the tank filling process
EmptyEvnet	TankEmptyHandler	public	Event to handle the tank emptying process
LevelEvent	LevelStatusHandler	public	Event to handle the water level event
SignalEvent	SignalHandler	public	Event to handle the signal event

Table 9.4 Summary of SCADA processes sequence diagram

Sequence Diagram Summary				
Sequence ID: SD-A1				
Sequence name: SCADA system sequence diagram				
Method name	Type	Description	Source	Destination
Start Pump	Message	Function to start the pumping process of water in the tank	Form	RTU
Stop Pump	Message	Function to stop the water pumping process	Form	RTU
Open valve	Message	Function to open the distribution valve of the tank to release water	Form	RTU
Close valve	Message	Function to close the distribution valve of the tank	Form	RTU
Level status	Return Message	Function to get the status of water level in the tank	RTU	Form
Running status	Return Message	Function to get the status of the tank in running mode	RTU	Form
Start timer	Self Message	Function to start the timer for measuring the total time for running	Form	Form
Update timer	Self Message	Function to update timer on the display screen	Form	Form
Level signals	Return Message	Function to get the signals of the water level in the tank	Tank	RTUs
Running signals	Return Message	Function to get the signals of the tank in running mode	Pump	RTU

Table 9.5 Summary of SCADA Start Pump user case

Use case summary
Use case ID: UC-A1
Use case name: Start Pump
Actors: User
Description: The user presses Start Pump button on the screen to start the water pumping in the tank
Preconditions: The pump is in STOP state
Post conditions: The screen shows the water level movement in the tank, pump starting time and total running hours and changes the status of the pump to RUNNING
System parameters: Pump status
Success scenario: The screen is loaded with required time and status information and motor image. The user is shown a message if the tank is full
Alternative scenario: The user can press Stop Pump in between the running stage. This will stop the filling water process in the tank. If the user press Open Valve button during the running state, then the water inflow and outflow will occur simultaneously
Includes: RTU, Tank, Pump
Priority: High

Table 9.6 Summary of SCADA Open Distribution Valve

Use case summary
Use case ID: UC-A2
Use case name: Open Distribution Valve
Actors: User
Description: The user selects Open Valve button to open the valve of the tank to distribute water out of the tank
Preconditions: The distribution valve is in CLOSE state
Post conditions: The screen shows the water level movement in the tank and changes the status of the valve to OPEN
System parameters: Distribution valve status
Success scenario: The screen is loaded with required time and status information. The user is shown a message if the tank is empty
Alternative scenario: The user can press Close Valve in between the running stage. This will stop the distributing water process from the tank. If the user press Start Pump button during the running state, then the water inflow and outflow will occur simultaneously
Includes: RTU, Tank, Pump
Priority: High

The Table 9.1 shows the member variables and methods of Pump class. Pump class used to represent the water pump for a Tank class. Constructor of Pump class receives Tank object as parameter to attach pump with a Tank.

The Table 9.2 describes the member variables, methods and events of Tank class. Tank class is used to represent the water tank for SCADA application. Constructor of Tank class receives tank number for Tank identification and tank capacity information. Tank class also defines two methods for water distribution, Open Distribution Valve and Close Distribution Valve method. Tank class uses two events for water level to update SCADA application through RTU object.

The Table 9.3 defines the member variables, methods, and events of RTU class. RTU class is used to represent remote terminal unit and provides communication bridge between SCADA application and Tank and Pump. Pump and Tank class objects are created in RTU class. Events information is passed on to SCADA application using RTU own events.

The Table 9.4 illustrates the SCADA processes sequence diagram.

The Table 9.5 presents the SCADA Start Pump user case.

The Table 9.6 outlines the SCADA Open Distribution Valve.

The Table 9.7 specifies the SCADA processes flow charts.

The following code is taken from SCADA system RTU class. In RTU class definition, it creates and initializes three Tank objects and creates three Pump objects with two status timers and four events for updating SCADA application.

Table 9.7 Summary of SCADA process flow charts

Flow chart summary	
Flow Chart ID: FC-A1	
Flow Chart Name: SCADA Process Flow Chart	
No. of Inputs: 4	
No. of Processes: 12	
No. of Outputs: 2	
No. of Control Decisions: 0	
Input name	Description
Start Pump	Start Pump button on screen
Stop Pump	Stop Pump button on screen
Open Distribution Valve	Open Valve button on screen
Close Distribution Valve	Close Valve button on screen
Process name	Description
Update running status	Function to Update running status
Get current time	Function to Get current time
Start Pump	Function to Start Pump
Tank	Function to update Tank
Water Level Sensors	Function to set Water Level Sensors
Distribution Valve	Function to set Distribution Valve
Update running time	Function to Update running time
Stop pump	Function to Stop pump
Control distribution valve	Function to Control distribution valve
Update level indicators	Function to Update level indicators
RTU	Function to update RTU
PUM	Function to update PUM
Output name	Description
Graphics display	Tank, water level bar and RTU display on screen
Text display	Status and other information display on screen

In the constructor of RTU, it initializes three Pump objects with three Tank objects as parameters. After initialization of Pump objects, it initializes and start status Timer, Fill and Empty events for all Tank objects.

After the constructor definition, it define Empty and Fill event handlers for all Tank objects and methods that returns Pump and methods for controlling distribution valve of Tank objects.

SCADA SYSTEM: RTU class

```
     public class RTU
    {
        Tank TankOne = new Tank(100,1);
        Tank TankTwo = new Tank(100,2);
        Tank TankThree = new Tank(100,3);

        Pump PumpOne;
        Pump PumpTwo;
        Pump PumpThree;

          Timer RTUTimer;
        Timer SignalTimer;

        public event FillHandler FillEvent;
        public event EmptyHanler EmptyEvent;
        public event LevelStatusHandler LevelEvent;
        public event SignalHandler SignalEvent;

        public RTU()
        {
            PumpOne = new Pump(TankOne);
            PumpTwo = new Pump(TankTwo);
            PumpThree = new Pump(TankThree);
            RTUTimer = new Timer(100);
            RTUTimer.Elapsed += new
ElapsedEventHandler(RTUTimer_Elapsed);
            RTUTimer.Start();

            SignalTimer = new Timer(100);
            SignalTimer.Elapsed += new
ElapsedEventHandler(SignalTimer_Elapsed);
            SignalTimer.Start();


            //Tank One Events
            TankOne.FillEvent += new
TankFillHandler(TankOne_FillEvent);
            TankOne.EmptyEvnet += new
TankEmptyHandler(TankOne_EmptyEvnet);

            //Tank Two Events
            TankTwo.FillEvent += new
TankFillHandler(TankTwo_FillEvent);
            TankTwo.EmptyEvnet += new
TankEmptyHandler(TankTwo_EmptyEvnet);

            //Tank Three Events
            TankThree.FillEvent += new
TankFillHandler(TankThree_FillEvent);
            TankThree.EmptyEvnet += new
TankEmptyHandler(TankThree_EmptyEvnet);
        }


 void SignalTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            try
```

```
            {
                SignalEvent();
            }
            catch (Exception exp)
            { }
        }


        void TankThree_EmptyEvnet(string message)
        {
            TankThree.CloseDistributionValve();
            EmptyEvent(3);

        }
        void TankTwo_EmptyEvnet(string message)
        {
            TankTwo.CloseDistributionValve();
            EmptyEvent(2);

        }
        void TankOne_EmptyEvnet(string message)
        {
            TankOne.CloseDistributionValve();
            EmptyEvent(1);

        }
        void TankOne_FillEvent(string message)
        {
            TankOne.StopStart();
            FillEvent(1);

        }
        void TankThree_FillEvent(string message)
        {
            TankThree.StopStart();
            FillEvent(3);
        }
        void TankTwo_FillEvent(string message)
        {
            TankTwo.StopStart();
            FillEvent(2);
        }




void RTUTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            try
            {

                LevelEvent(1, TankOne.Level);
                LevelEvent(2, TankTwo.Level);
                LevelEvent(3, TankThree.Level);
            }
            catch (NullReferenceException exp)
            {
```

```
        }
    }


    public Pump GetPumpOne()
    {
        return PumpOne;
    }
    public Pump GetPumpTwo()
    {
        return PumpTwo;
    }
    public Pump GetPumpThree()
    {
        return PumpThree;
    }
    public void OpenDistributionValve(int TankNumber)
    {
        switch (TankNumber)
        {
            case 1:
                TankOne.OpenDistributionValve();
                break;
            case 2:
                TankTwo.OpenDistributionValve();
                break;
            case 3:
                TankThree.OpenDistributionValve();
                break;
            default:
                break;
        }

    }
    public void CloseDistributionValve(int TankNumber)
    {
        switch (TankNumber)
        {
            case 1:
                TankOne.CloseDistributionValve();
                break;
            case 2:
                TankTwo.CloseDistributionValve();
                break;
            case 3:
                TankThree.CloseDistributionValve();
                break;
            default:
                break;
        }

    }
}
```

9.3 Interface Design for Systems Integration

An illustration of interface of SCADA components with virtual factory is presented in following Figures.

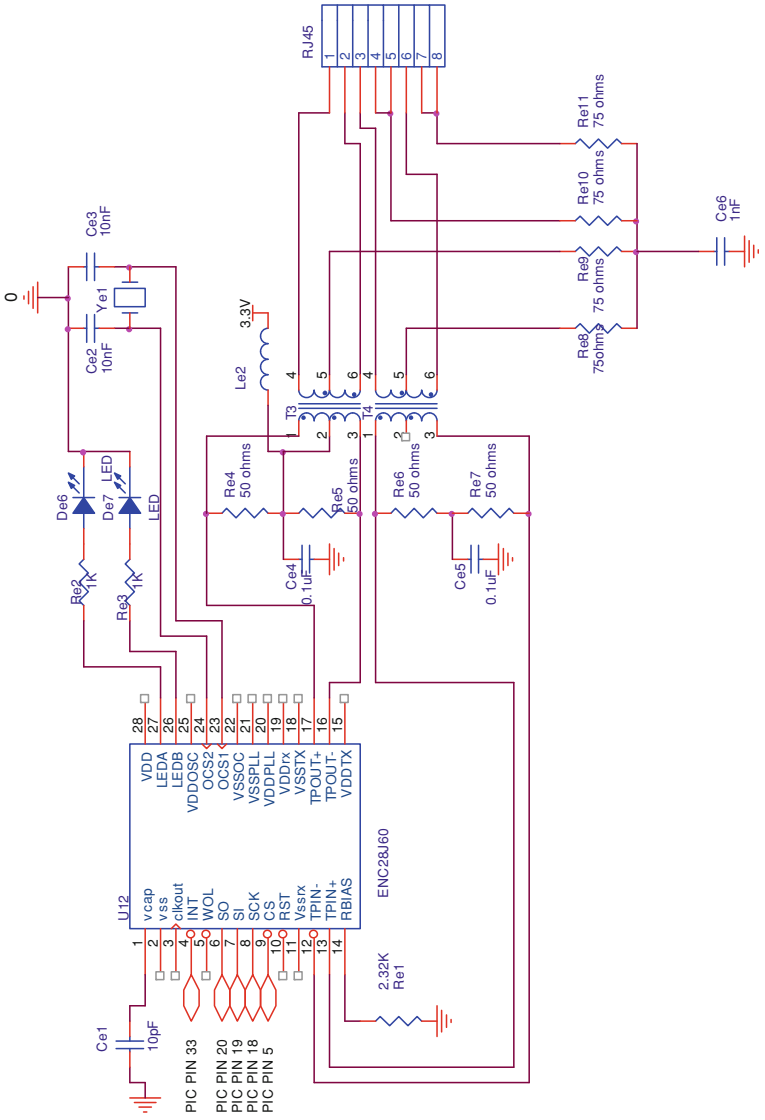


Fig. 9.6 Control of section of SCADA system over Local Area Network

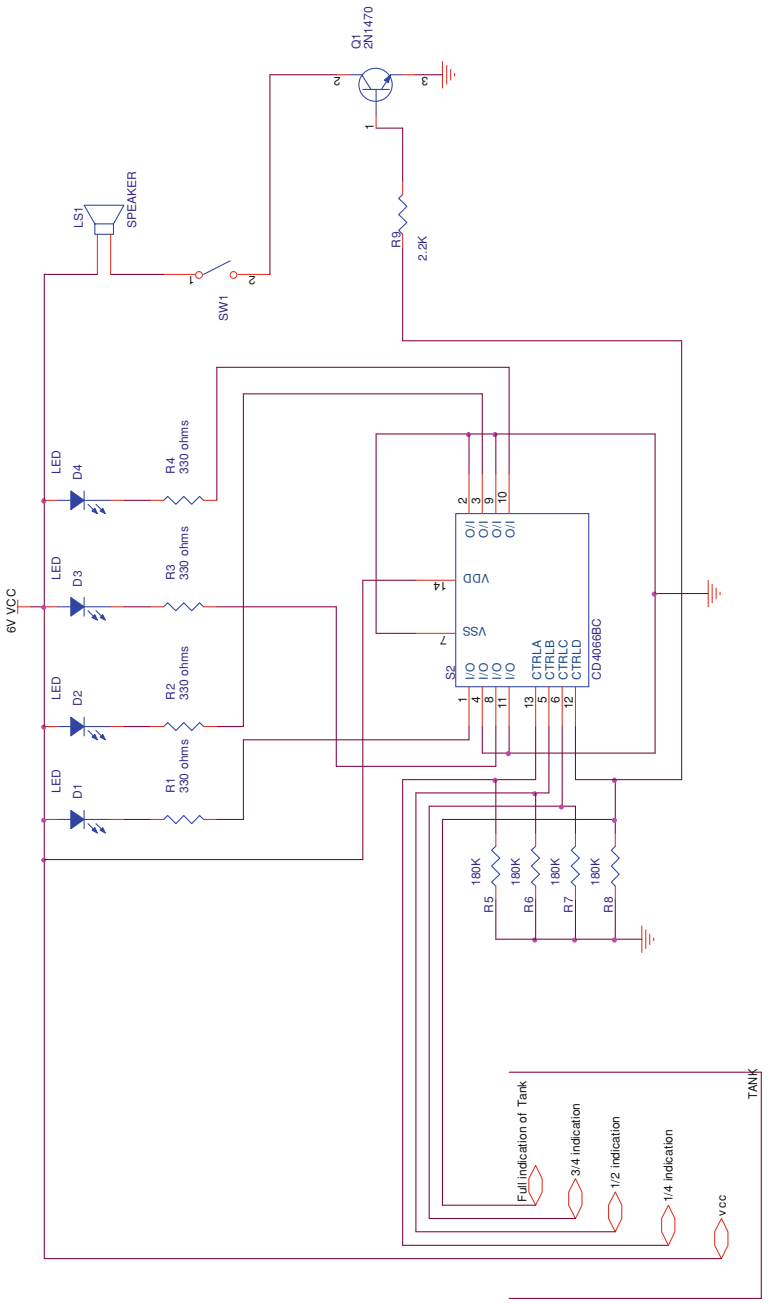


Fig. 9.6 (continued)

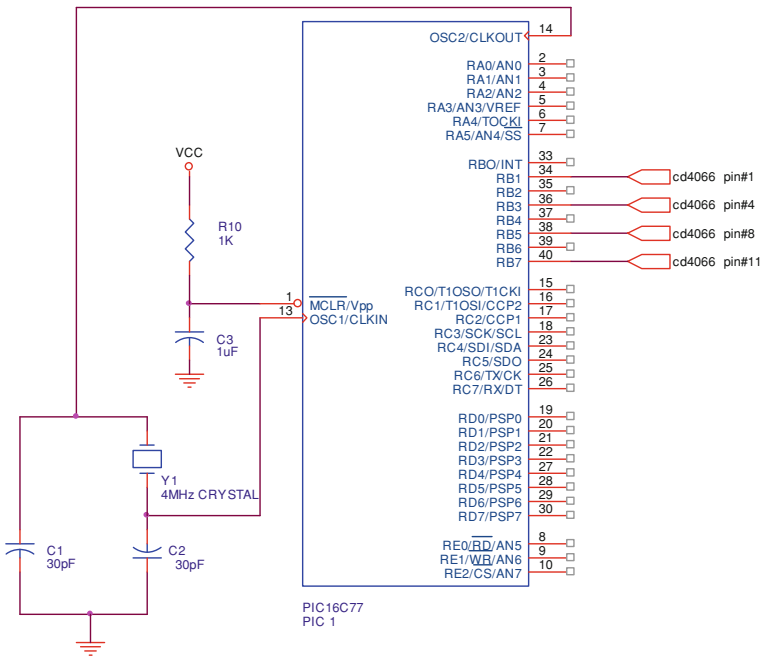


Fig. 9.6 (continued)

Bibliography

1. Darwish KW (2008) Virtual SCADA simulation system for power substation. In: 4th international conference on innovations in information technology, pp 322–326
2. David H (2001) User-centred design of supervisory control systems. IEE Conference Publication, pp 291–296
3. Freund E (2001) Space robot commanding and supervision by means of projective virtual reality: the ERA experiences. In: Proceedings of SPIE–The International Society for Optical Engineering, pp 312–322
4. Freund E (2004) Using supervisory control methods for model based control of multi-agent systems. In: IEEE Conference on Robotics, Automation and Mechatronics, pp 649–654
5. Freund E (2001) State oriented modeling as enabling technology for projective virtual reality. In: IEEE International Conference on Intelligent Robots and Systems, pp 1842–1847
6. Gustavsson I (2002) A remote laboratory for electrical experiments. In: ASEE Annual Conference Proceedings, pp 11285–11293
7. http://en.wikipedia.org/wiki/SCADA
8. Promel F (2005) Simulated reality. ABB Rev 1(2):62–65
9. Salazar C (2001) Building boundaries and negotiating work at home. In: Proceedings of the international ACM SIGGROUP conference on supporting group work, pp 162–170
10. Teng JH (2001) Integration of internet and virtual instruments to develop an industrial SCADA. In: Proceedings of the IEEE power engineering society transmission and distribution conference, pp 1509–1514

Chapter 10
Augmented Reality for Mechatronics-Based Applications

10.1 Characterstics of Mechatronics-Based Application

A Mechatronics system relies on various branches of engineering. It automatically controls gadgets used in all aspect of life. Key characteristics of the Mechatronics systems are

(i) Contrary to (Transister Transister Logic) TTL or CMOS voltage levels only, the Mechatronics system may also be operating on different voltage levels.
(ii) Standard series or parallel transmission protocols, such as RS232 or IEEE 488 may not necessarily be implemented in their true spirit.
(iii) Communication protocols such as ISO OSI layers protocol may not be fully implemented or augmented with additional layers.
(iv) Computer Numerical Control-related standards may be used with modification.
(v) Programmable Logic Controller-related standards and principles may be used with modification.
(vi) Standards and practices in designing embedded or SCADA systems may not be adhered to.
(vii) An equipment designed and manufactured based on pertinent standards related to CNC, PLC, Embedded System or SCADA are distinguished distinctly. On the other hand, the equipment designed and manufactured as Mechatronics based equipment may not be distinctly identified as belonging to either of well-known control gadgetry.
(viii) A Mechatronics system involves presence of a mechanical component or mechanism.
(ix) Mechatronics-based systems are produced as one off or in small quantity.
(x) A CNC system, PLC system, embedded system or a SCADA system may be a part of Mechatronics system.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 551
DOI: 10.1007/978-0-85729-186-8_10, © Springer-Verlag London Limited 2011

10.2 Augmented Reality for Mechatronics Applications

The augmented reality development for mechatronics based system uses the same principles as that of CNC, PLC, Embedded System or SCADA-based system for augmented reality design. A number of examples of mechatronics system may be taken to explain the augmented reality design for the mechatronics system. In Sect. 10.3, an example of a linear actuator coupler with a stepper motor and a shaft encoder is taken. This is a sub-assembly of a CNC milling machine. The design and operation of such system is adequately given in Chap. 5.

10.3 System Integration Methodology

Mechatronics based systems are employed at diverse types of the products. Most of the circuit presented in the earlier chapters belong to this type of control application. Figure 10.1 provides circuits for controlling a sub-assembly of a CNC milling machine.

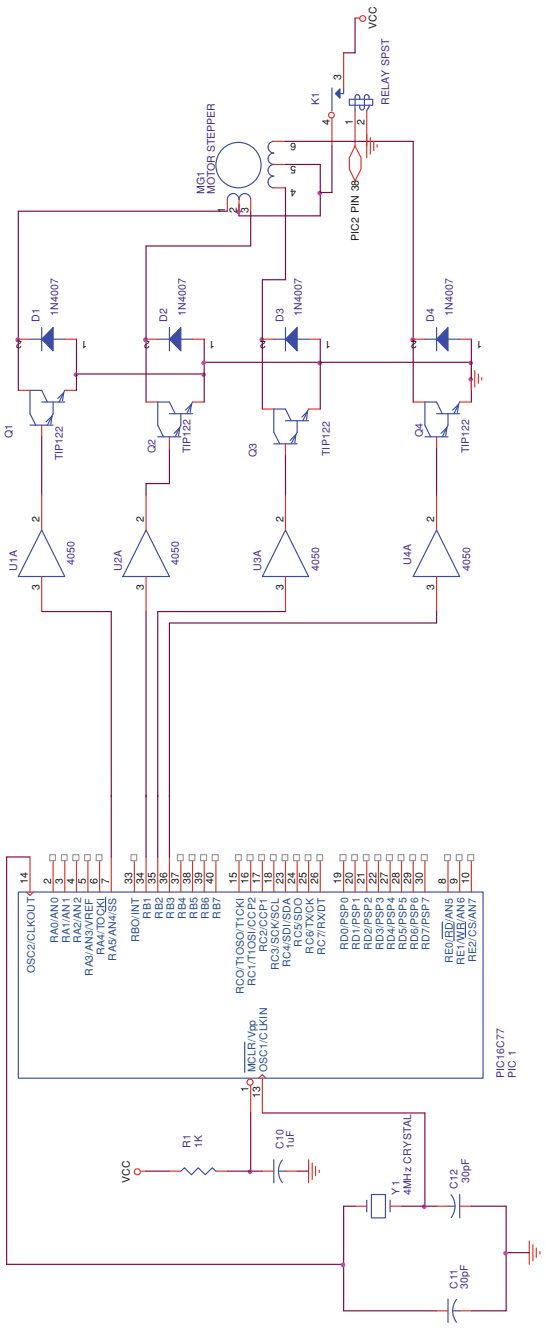


Fig. 10.1 Control of mechatronics (continued)

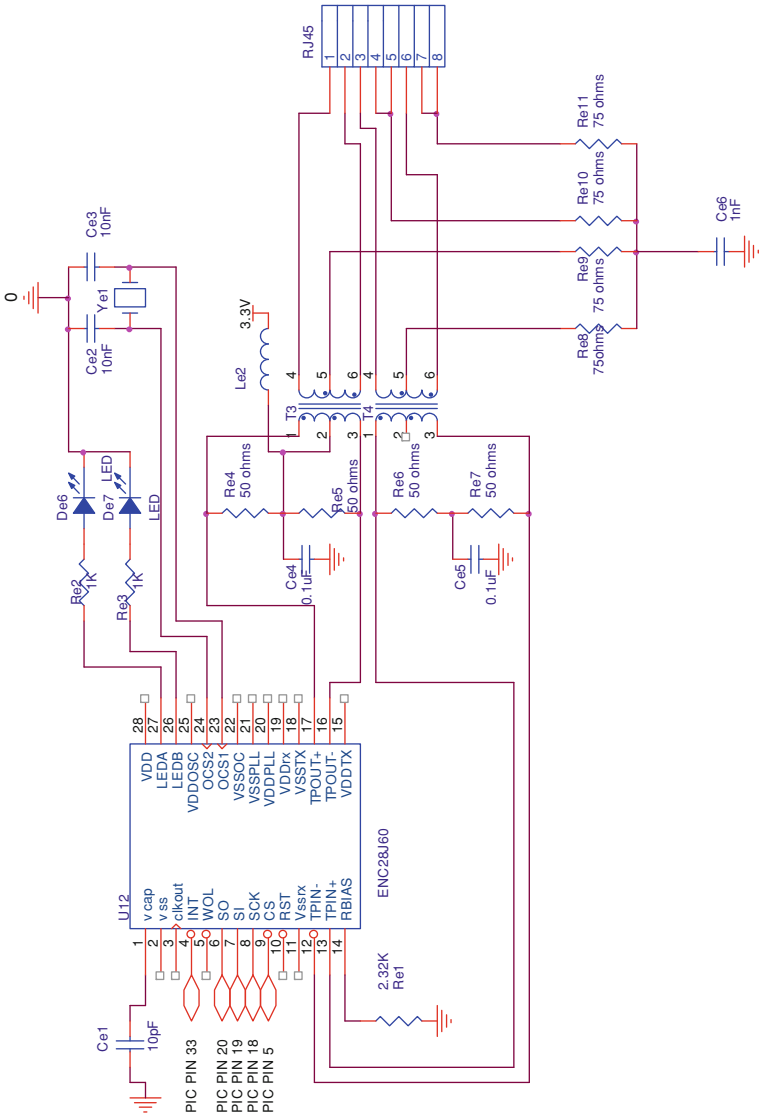


Fig. 10.1 (continued)

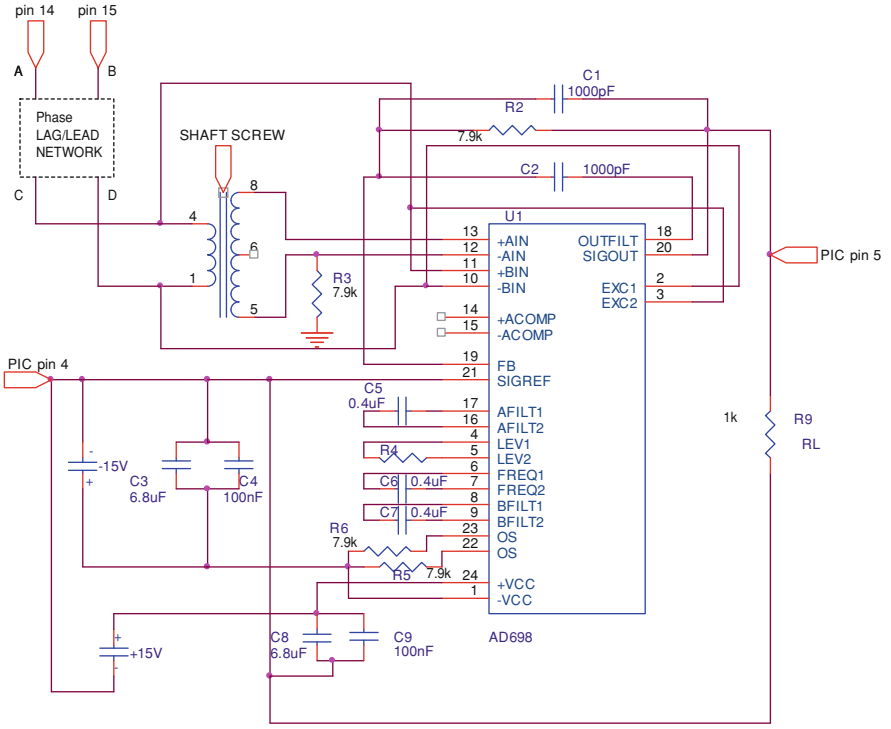


Fig. 10.1 (continued)

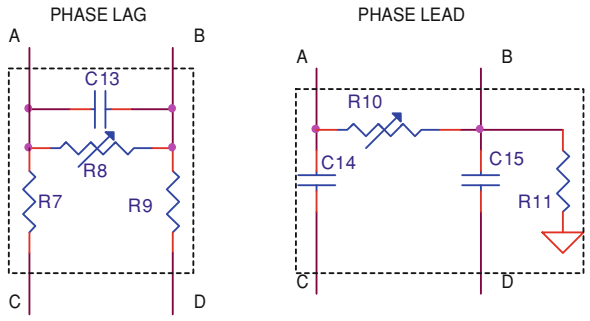


Fig. 10.1 (continued)

Bibliography

1. Bellamine M (2004) A remote diagnosis system for rotating machinery using virtual reality. In: IEEE conference on robotics, automation and mechatronics, pp 716–723
2. Coman M (2009) Design, simulation and control in virtual reality of RV-24J robot. In: IECON proceedings, pp 20206–20310
3. Cui G (2009) Study of tolerance products modeling based on virtual reality. Appl Mech Mater 16–19:781–785
4. Elton MD (2009) A virtual reality operator interface station with hydraulic hardware-in-the-loop simulation for prototyping excavator control systems. In: IEEE/ASME international conference on advanced intelligent mechatronics, pp 250–255
5. Green SA (2007) Human robot collaboration: an augmented reality approach a literature review and analysis. In: Proceedings of the ASME international design engineering technical conferences and computers and information in engineering conference, pp 117–126
6. Green SA (2008) Evaluating the augmented reality human-robot collaboration system. In: International conference on mechatronics and machine vision in practice, pp 521–526
7. http://en.wikipedia.org/wiki/Mechatronics
8. Lubczynski TR (2009) Virtual simulation of mechatronics laboratory. Diffusion and Defect Data. Part B: Solid State Phenomena, 147–149:930–935
9. Padilla MA (2009) Virtual reality simulator of transurethral resection of the prostate. In: Pan American Health Care Exchanges, pp 116–119
10. Park H (2007) Teleoperation of a multi-purpose robot over the internet using augmented reality. In: ICCAS— International conference on control, automation and systems, pp 2456–2461
11. Radkowski R (2008) An augmented reality-based approach for the visual analysis of intelligent mechatronic systems. In: Proceedings of the ASME international design engineering technical conference and computers and information in engineering, pp 1579–1587
12. Sonar AV (2005) Development of a virtual reality-based power wheel chair simulator. In: IEEE international conference on mechatronics and automation, pp 222–229
13. White D (2005) A virtual reality application for stroke patient rehabilitation. In: IEEE international conference on mechatronics and automation, pp 1081–1086
14. Yuji W (2009) Space teleoperation with large time delay based on vision feedback and virtual reality. In: IEEE/ASME international conference on advance intelligent mechatronics, pp 1200–1205

Chapter 11
Virtual Manufacturing for Discrete Manufacturing Systems

11.1 Introduction

The evolution of manufacturing practices since industrial revolution relates directly to the advancement in scientific methodologies and management principles. Some examples of the manifestations of these principles are:

1. Development of coal-fired furnaces for iron making
2. Development of steam engines and steam-driven machines
3. Development of transportation, communication, and electricity networks
4. Establishment and advancement in automobile industry
5. Development of mass production systems
6. Development and advancement in aviation industry1
7. Development of modern manufacturing processes
8. Development in electronics industry
9. Development of computer science and engineering
10. Development in Aerospace industry
11. Advancement in weapon technology

Several production management philosophies were presented since industrial revolution in order to enhance efficiency and profitability. These philosophies are[1]:

1. Group technology
2. Kanban (Just in Time)
3. Material Resource Planning (MRP)
4. MRP II
5. Enterprise Resource Planning (ERP)
6. Theory of Constraints
7. Lean Manufacturing

[1] Extracted with permission from Khan WA, Raouf A (2006) Standards for engineering design and manufacturing, CRC/Taylor and Francis, USA.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-0-85729-186-8_11,

8. Six Sigma
9. Computer Integrated Manufacturing
10. KAIZEN
11. Agile Manufacturing

These philosophies aim at improving the efficiency of a system and to increase the profits of an organization. Some of the parameters used in these philosophies are similarity of products, flow of information, flow of material, manufacturing facilities layouts, elimination of defect, optimization of slowest link, supply of raw material, produced goods inventories, automation, and continuous improvement.

This chapter is the culmination of knowledge described in previous chapters. It takes into account the Enterprise Resources Planning as the underlying manufacturing philosophy and describes a software product entitled Virtual Manufacturing System (VMS). The various VMS modules as described in Chap. 1 (refer to system requirement specification, Sect. 1.9.4) are further elaborated.

Discrete virtual manufacturing described here operates on ERP philosophy. However, VMS being software based when applied to tangible and intangible functions of the production, it can handle all parameters for any of the production philosophies that do not involve movement of the installed machinery. Also refer to Sect. 12.4.

11.2 Components of the VMS

Virtual Manufacturing System is an elaborate application software. Its various modules provide functionality appropriate to selected domain. The functionality of each module is listed below:

1. VMS Design
   - Machine Database Window
   - Properties Window
   - Factory Layout Window
   - Output Window
   - Save Project
   - Open Project
   - Create New Project
   - Products
   - Products Categories
   - Products Classification
   - Help
2. VMS Planning
   - Process Control Window
   - Factory Layout Window

- Output Window
- Process Planning
  - Route Sheet
  - Machines Database Reports
- Machine Database
  - Milling
  - Turning
  - Drilling
- Costing Reports
- Scheduling Reports
- Inventory Management
  - Raw Material Status
  - Consumable Items Status
  - Non-consumable Items Status
- Management Information System (MIS)
  - Reports
- Help

3. VMS Monitoring
   - Process Control Window
   - Factory Layout Window
   - Factory 3D Window
   - Machine Output Window
   - Properties Window
   - Job Status Window
   - Output Window
   - MIS
     - Reports
   - Process Planning
     - Route Sheet Report
     - Machine Database Report
   - Costing Reports
   - Scheduling Reports
   - Machine Database
   - Help
4. VMS Fault Diagnostic
   - Process Control Window
   - Factory Layout Window

- Factory 3D Window
- Controller Information Window
- Machine Output Window
- Output Window
- Scheduling Reports
- Help

5. VMS Quality Assurance
   - Sales
   - Purchase
   - Help
6. VMS Venders
   - Venders
   - Help
7. VMS Business
   - Procurement
     - Requisition
     - Purchasing
   - Sales
     - Finish Goods Entry and Store Status
     - Sales Order and Order Status
     - Process Sales Order
   - Accounts and Finance
     - Reports

       Charts of Account
       Income Statement
       General Ledger Trial Balance
       Balance Sheet
       General Ledger
   - Accounts Types
     - General Entries
     - Budget Allocation
   - Business
     - Search Business
     - Search Reports
   - Help

8. VMS Assembly
    - Stationary operator moving workpiece animation
9. VMS Training
10. VMS Administrator
    - User Accounts and Permissions
    - Departments
11. VMS Programs
    - Samples Programs
12. VMS Videos
    - Samples Videos
13. VMS Help
    - System Help

Details of following major VMS components are given in the following pages:

1. Factory Layout
2. Discrete Manufacturing Processes
3. Pick and Place Technology
4. Costing
5. Accounting and Finance
6. Sales
7. Inventory Management
8. Procurement
9. Process Planning
10. Quality Assurance
11. Scheduling
12. MIS

The above components with their detailed implementation are part of one or more VMS software module. In most of the cases the report generated for these components is accessible once the virtual factory has completed a project and the current data have been stored in appropriate databases.

11.2.1 Factory Layout

Virtual Manufacturing System software allows definition of factory floor area initially for planning purpose. Multiple cells can be defined at the factory floor. VMS permits drag and drop operation of available machinery to cell or anywhere in the factory layout. The envelop encompassing the factory is graduated and

allows machinery to be positioned precisely. This activity is the most basic activity in defining the virtual factory.

11.2.2 Discrete Manufacturing Processes

Discrete Functionality of Manufacturing processes is demonstrated through augmented reality of metal cutting processes such as Milling, Turning, Drilling, and Sawing. This is a very small selection from a very large number of manufacturing processes as listed in Chap. 2. Manufacturing system based on job shop or cellular manufacturing can be defined. This activity compliments the definition of factory floor area. There is a provision at VMS to define the properties of manufacturing system selected for any given setup of the virtual factory. This also includes loading the virtual cutting process with the required part program. The part program is defined using selection of G and M code as per EIA 274D standard, refer to Chap. 5. The part program is developed manually or at any computer Aided Manufacturing CAM facility and stored on hard disk of the computer operating the VMS software. Option is available at VMS Monitor to load and run appropriate part program at the virtual machine.

11.2.3 Pick and Place Technology

Pick and Place Technology is represented through revolute robot, conveyor, and gantry crane. The Virtual processes for these objects operate on differing computer software technology. The robot uses a compiler constructed according to Japanese Industrial Stand "Standard Language for Industrial Manipulator" (JIS SLIM). The gantry and the conveyor operate on user-defined commands with command translation using an interpreter. VMS Monitor allows full functionality of these machines including their positioned and operation parameters.

11.2.4 Costing

In a business organization, the Activity Based Costing (ABC) methodology assigns an organization's resource costs through activities to the products and services provided to its customers. It is generally used as a tool for understanding product and customer cost and profitability. As such, ABC has predominantly been used to support strategic decisions such as pricing, outsourcing, and identifying and measuring process improvement initiatives.

Virtual Factory has the capabilities to generate costing report for a particular project based on different parameters. User can view the total cost of a selected project. Project must be run once to generate costing report. Costing Report of virtual factory includes the material cost, machine cost, consumable item cost,

Virtual Factory

Production Costing Report

Date: 06-April-2009

Costing Detail For :	Dispenser Prodection	**Quantity to be produced:**	5
Project Code:	PKM45	**Material:**	Slabs
Process Detail:	Milling To Drilling	**Material Cost:**	5 S.

Machine Type:	**Milling**	**Time / Piece:**	
Machine Cost / Minute:	5 S.	**Total Time:**	4:30:660
Coolants Cost:	2 S.		
Lubricants Cost:	2 S.		
Cost / Piece:	9.00 S.		
Process 1 Total Cost:	45.00 $.		
Project Total Cost:	70.00 $.		

Fig. 11.1 Production costing report

non-consumable item cost and shows cost/product and total cost of the project. Figure 11.1 presents a typical production costing report generated by VMS.

11.2.5 Accounts and Finance

Accounts and Finance module in virtual factory controls all account- and finance-related task. Accounts and Finance menu has four sub-items as shown in the following picture.

I. *Reports*
Reports menu item has five sub-items as shown in the following picture. These reports provide full detail of accounting and finance.

(A) Charts of Accounts
Chart of Accounts report shows all charts of accounts that virtual factory currently used to maintain accounts and finance (Fig. 11.2).

Fig. 11.2 Chart of Accounts

Virtual Factory
Chart of Accounts
As of 06-Apr-2009

Account ID	Account Description	Active ?	Account Type
1	Cash in Hand	Yes	Cash
2	Sales	Yes	Income
3	Purchase	Yes	Cost of Sales
4	Salaries	Yes	Expenses
5	Capital	Yes	Equity-Retained Earnings
6	Machines	Yes	Fixed Assets
8	Precision Engineered	Yes	Accounts Payable
9	Honda Corporation	Yes	Accounts Receivable
10	Handy Twins International	Yes	Accounts Payable
11	Taizhou Hisource	Yes	Accounts Payable
12	Taizhou Papers International	Yes	Accounts Payable

Fig. 11.3 Income Statement

Virtual Factory
Income Statement

	Current Month
Income	
Sales	24,375.00
Total Income	**24,375.00**
Cost of Sales	
Purchase	8,700.00
Gross Profit	**8,700.00**
Expenses	
Salaries	10,000.00
Total Expenses	**10,000.00**

(B) Income Statement
Income Statement reports show the current status of income with sales, purchase, and salaries account heads being major contributor (Fig. 11.3).

(C) General Ledger Trial Balance
General Ledger Trial Balance reports show the current General Ledger Trail Balance (Fig. 11.4).

(D) Balance Sheet
Balance Sheet report shows the current balance sheet of the virtual factory (Fig. 11.5).

(E) General Ledger
General Ledger report shows the General Ledger of Virtual Factory (Fig. 11.6).

II. *Accounts Type*
From Account Type window, user can add new account type (Fig. 11.7).

III. *General Entries*
From the General Entries windows, user can post new entries (Fig. 11.8).

Fig. 11.4 General Ledger Trial Balance

Virtual Factory

General Ledger Trial Balance

As of 06-Apr-2009

	Account Description	Debit Amount	Credit Amount
1	Cash in Hand	971,500.00	0.00
2	Sales		24,375.00
3	Purchase	8,700.00	
4	Handy Twins International Co		1,600.00
4	Precision Engineered Products		800.00
4	Taizhou Hisource International		2,500.00
4	Taizhou Papers International		3,800.00
5	Salaries	10,000.00	
6	Jonson Engineering Co.	4,375.00	
6	Matthew Motors	2,500.00	
7	Capital		1,000,000.00
8	Machines	36,000.00	
	Total:	**1,033,075.00**	**1,033,075.00**

Fig. 11.5 Balance Sheet

Virtual Factory

Balance Sheet

Assets

Accounts Receivable	
Jonson Engineering Co.	4,375.00
Accounts Receivable	
Matthew Motors	2,500.00
Total Accounts Receivable	**6,875.00**
Cash	
Cash in Hand	971,500.00
Total Cash in Hand	**971,500.00**
Fixed Assets	
Machines	36,000.00
Total Fixed Assets	**36,000.00**
Total Assets	**1,014,375.00**

IV. *Budget Allocation*

Budget Allocation is the responsibilities of Finance Department. User can allocate budget to a particular account head or increase or decrease the

Virtual Factory
General Ledger

Account Description	Date	Trans Description	Debit Amt	Credit Amt	Balance
Capital		Beginning Balance			
	31-Jan-2009	Capital Introduce		1,000,000	
		Current Period Change		1,000,000	-1,000,000
		Ending Balance			**-1,000,000**
Cash in Hand		Beginning Balance			
	2-Feb-2009	Salary Paid to Manager		5,000	
	23-Feb-2009	Drilling Machine DK53		6,000	
	23-Feb-2009	Turning Machine GK25		8,000	
	31-Jan-2009	Milling Machine GK 150		10,000	
	24-Feb-2009	Sale Finish Product to Matthew	17,500		
	2-Feb-2009	Salaries Paid to Employee		5,000	
	23-Feb-2009	Milling Machine GM300		12,000	
	31-Jan-2009	Cash Introduce	1,000,000		
		Current Period Change	1,017,500	46,000	971,500
		Ending Balance			**971,500**
Handy Twins International Co		Beginning Balance			
	23-Feb-2009	Raw Material for Production		400	
	23-Feb-2009	Raw Material for Production		500	
	23-Feb-2009	Raw Material for Production		700	
		Current Period Change		1,600	-1,600
		Ending Balance			**-1,600**

Fig. 11.6 General Ledger

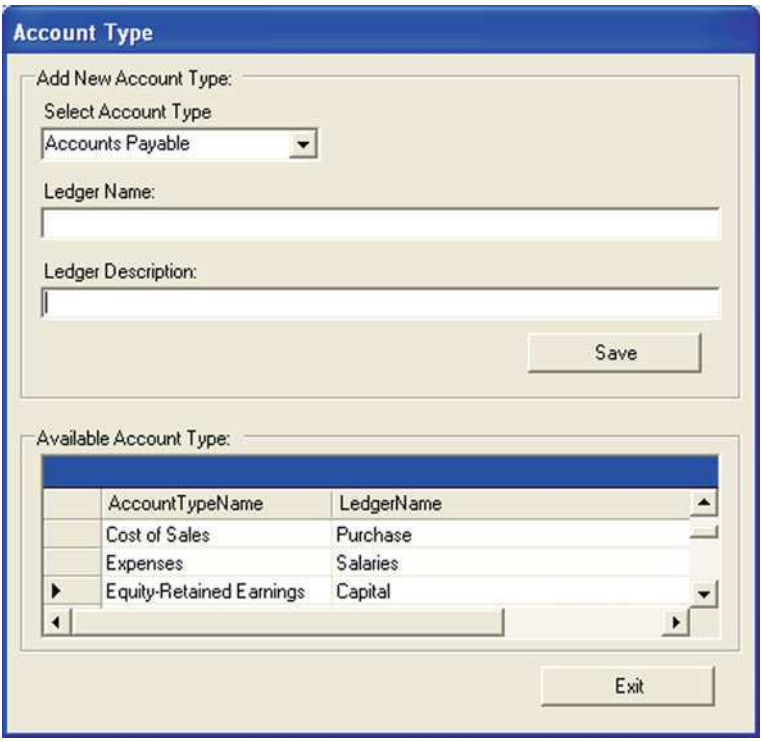


Fig. 11.7 Accounts Selection

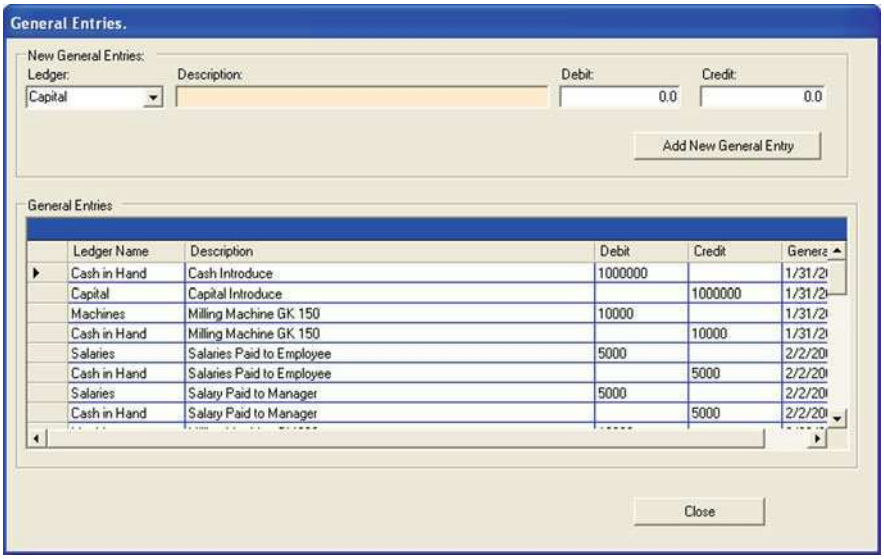


Fig. 11.8 New Entries to General Ledger

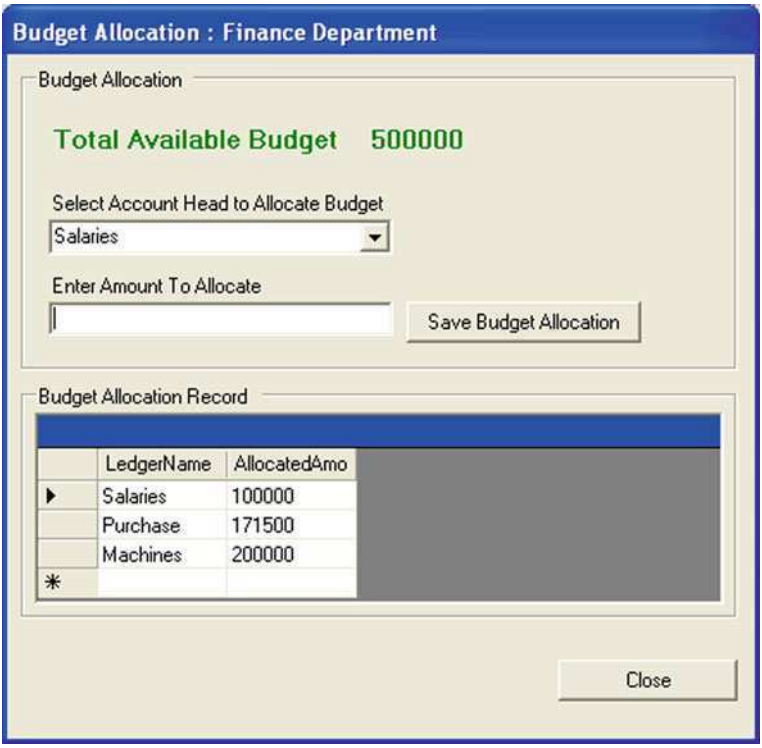


Fig. 11.9 Budget Allocation

allocated amount of a particular account head. Budget allocation is used to control the amount that particular account head cannot exceed from allocated budget. This window also shows the total available budget as well allocated budget for each account head (Fig. 11.9).

11.2.6 Sales

Sales in virtual factory include management of finished goods, Sales order and tracking orders and processing of sales orders.

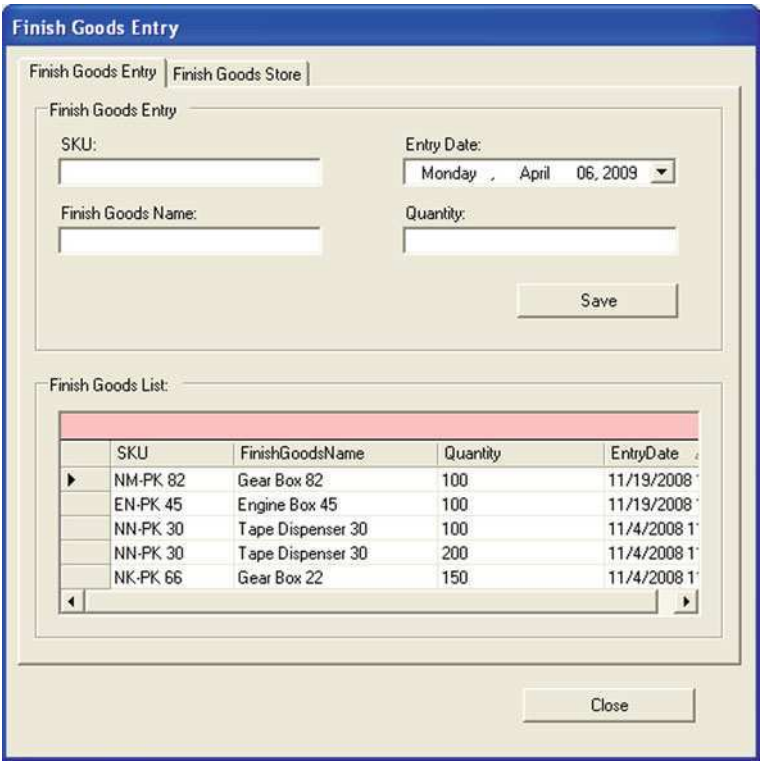


Fig. 11.10 Finish Goods Entry

I. *Finish Goods Entry and Store Status*
User can add finish goods that Virtual Factory has produced. Finish Goods must be entered before any sales. When user add new finish good, finish good store will be updated automatically but those goods must pass quality assurance test before the processing of sales order (Figs. 11.10 and 11.11). Finished Goods store tab shows the current status of the store. When user enters sales entry, store status will update automatically.

II. *Sales Order and Order Status*
In virtual factory, user with sales right must fill sales order form before sales process begins. From sales order tab user can add new sales order with full order detail (Fig. 11.12).
From sales order status tab user can track sales order status and delivery status. Figure 11.13 shows the current status of sales order.

III. *Process Sales Order*
After adding new sales order, sales order must be processed in order to give delivery to customer. When user process the order Debit and Credit entry will be updated in accounts and finance modules as well as in store inventory (Fig. 11.14).

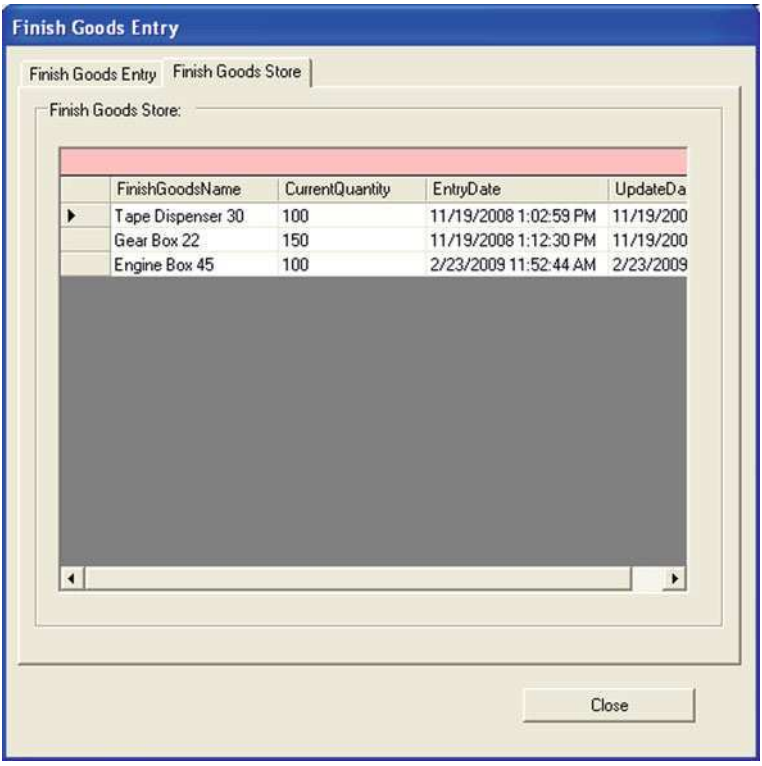


Fig. 11.11 Finish Goods Store

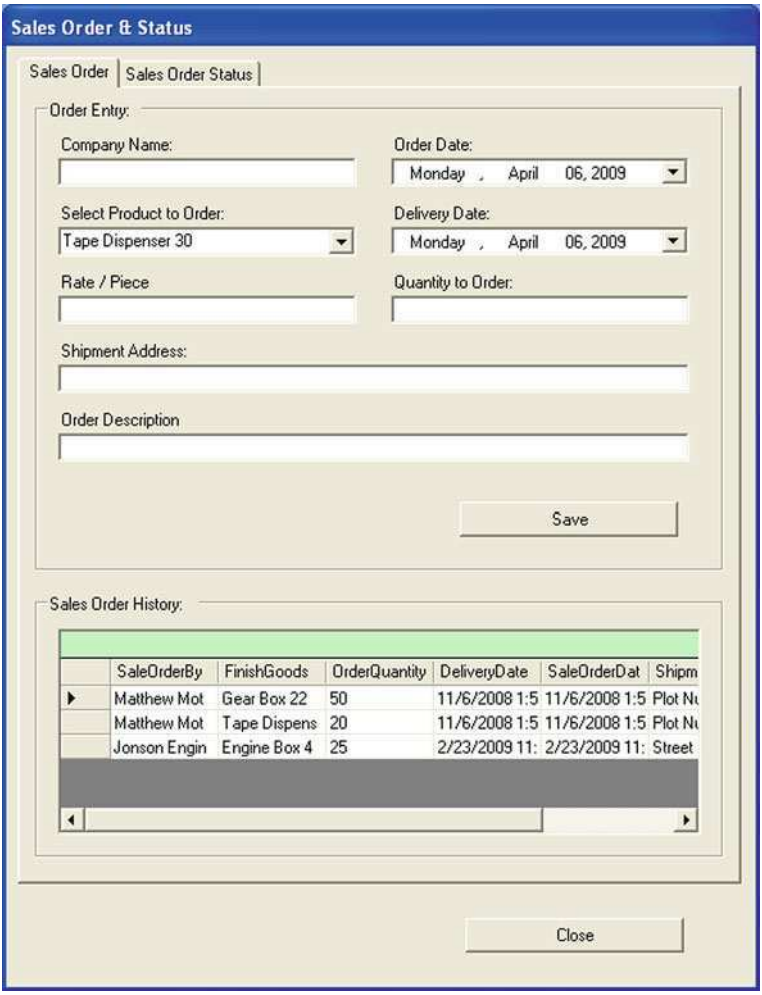


Fig. 11.12 Sales Order Entry

Fig. 11.13 Sales Order Status

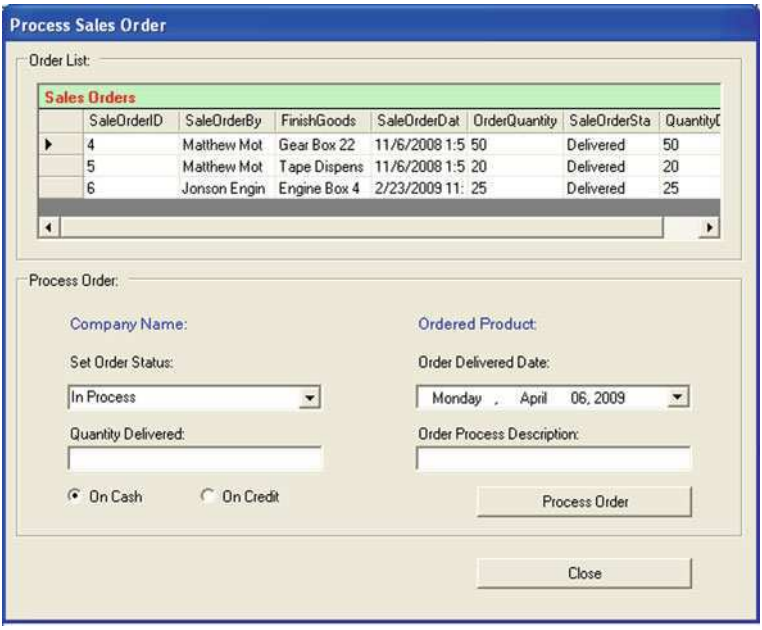


Fig. 11.14 Sales Order List

11.2.7 Inventory Management

Virtual Factory has Inventory Management system to manage inventory of raw material, consumable items, non-consumable item, and finished product. Item can be added in inventory by using purchasing module discussed latter in this chapter. By using Inventory Management user can only view current status of inventory items. Updating of these inventory items will be done automatically by virtual factory when user runs a project.

Inventory Management menu item has three sub items as shown in the following picture.

Inventory Management Submenu

I. *Raw Material Status*
Raw material in Virtual Factory is used to manufacturing finished goods. Virtual Factory automatically updates the quantity of raw material when

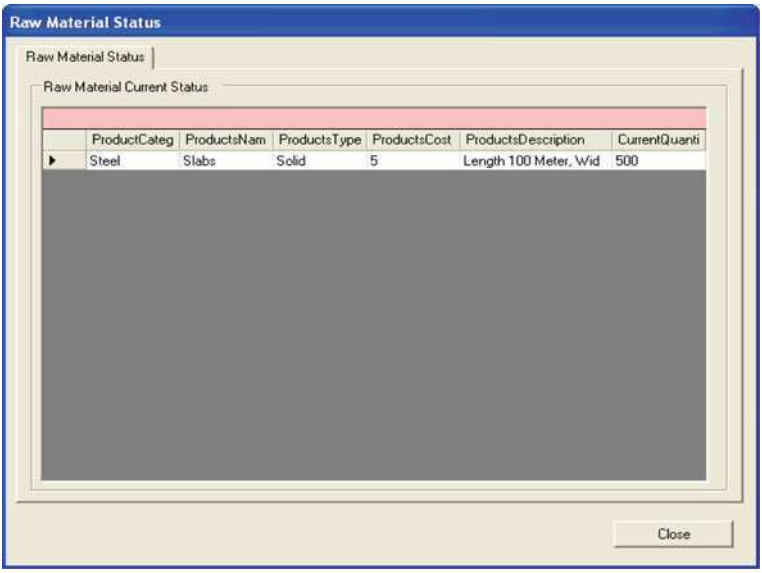


Fig. 11.15 Raw Material Status

finish product is created and when user purchase raw material for production. Raw material status window shows the current status of all raw materials used in virtual factory. Table in raw material status window displays information related to raw material including product category, product name, product type, cost of the product, description of the product and current quantity of material available for manufacturing in virtual factory (Fig. 11.15).

II. *Consumable Items Status*

Inventory Management system also manages consumable items. Consumable item are those items which are not directly used to create finished product but helps to create finished products such as lubricants, oil, cooling water or cutting fluid, etc.

Consumable items status window shows the current status of all consumable items used in virtual factory. Table in consumable items status window display all information related to consumable items including product name, product type, cost of the product, description of the product and current quantity of items available for consumption in virtual factory (Fig. 11.16).

III. *Non-Consumable Items Status*

Inventory Management system also manages non-consumable items. Non-consumable items are those items which are not directly used to create finished product but helps to create finished products such as tools.

Non-consumable items status window shows the current status of all non-consumable items used in virtual factory. Table in non-consumable items status window display all information related to non-consumable items

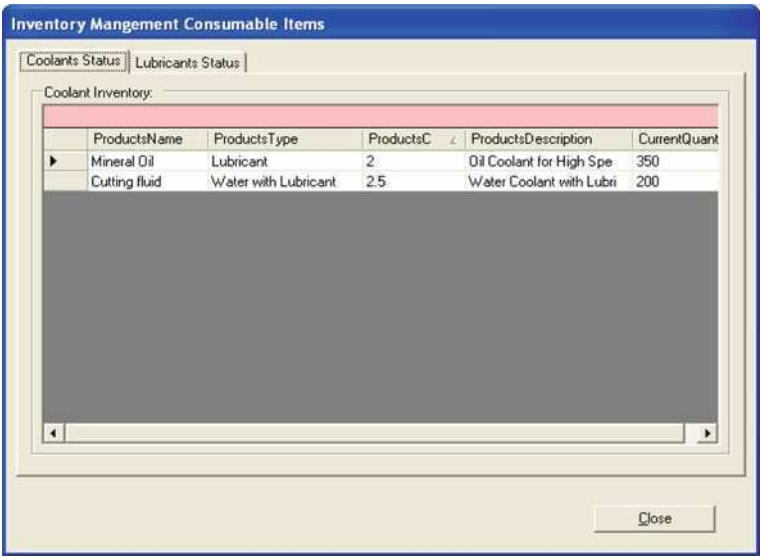


Fig. 11.16 Coolant Inventory

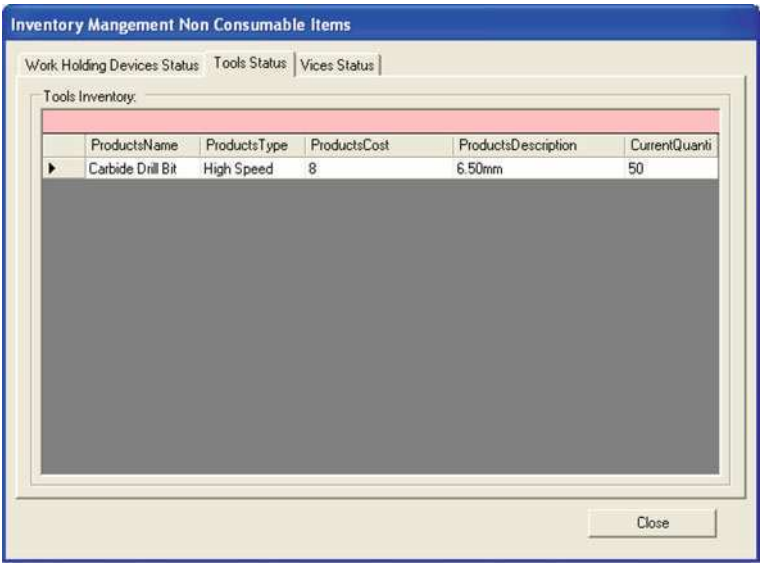


Fig. 11.17 Tools Inventory

including product name, product type, cost of the product, description of the product and current quantity of items available for used in virtual factory (Fig. 11.17).

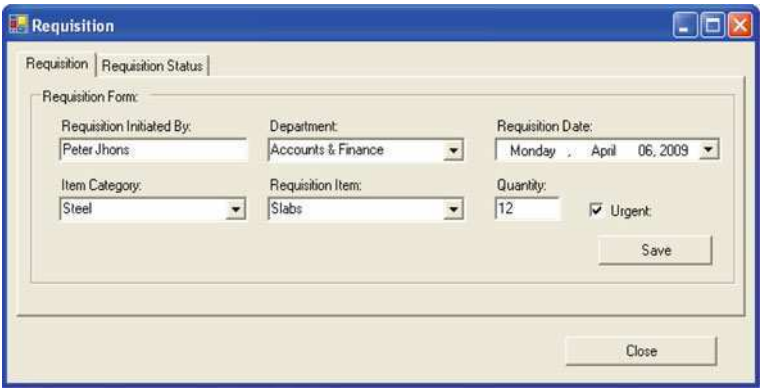


Fig. 11.18 Requisition Form

11.2.8 Procurement

Procurement is the largest module of virtual factory to handle requisition, purchasing, products, product categories, product classification, and product suppliers. Procurement menu has seven sub items as shown in the following picture and in the following each discussed.

I. *Requisition*
From the requisition tab, user can add new requisition as there requirement by selecting name, department, category of required product, name of required product and required quantity of the product. User can also set the importance of the requisition by setting the urgent check box. User with purchasing rights can approve or reject requisition based on the current status of store inventory (Fig. 11.18).

From the Requisition Status tab on requisition form user can view the status of requisition that was previously raised. When purchasing manager or any user with purchasing rights approve or reject requisition, Requisition Status window automatically updates to reflect status of requisition (Fig. 11.19).

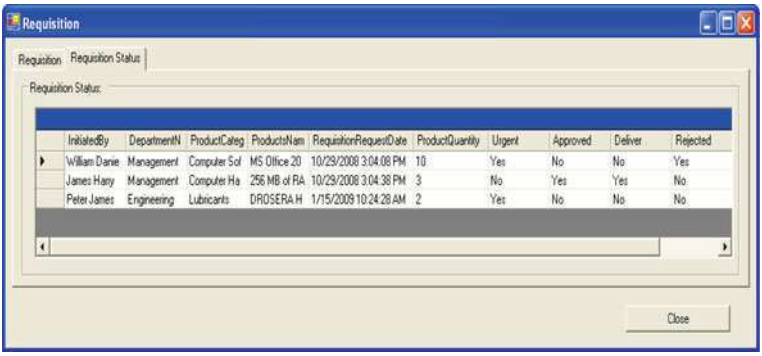


Fig. 11.19 Requisition Status

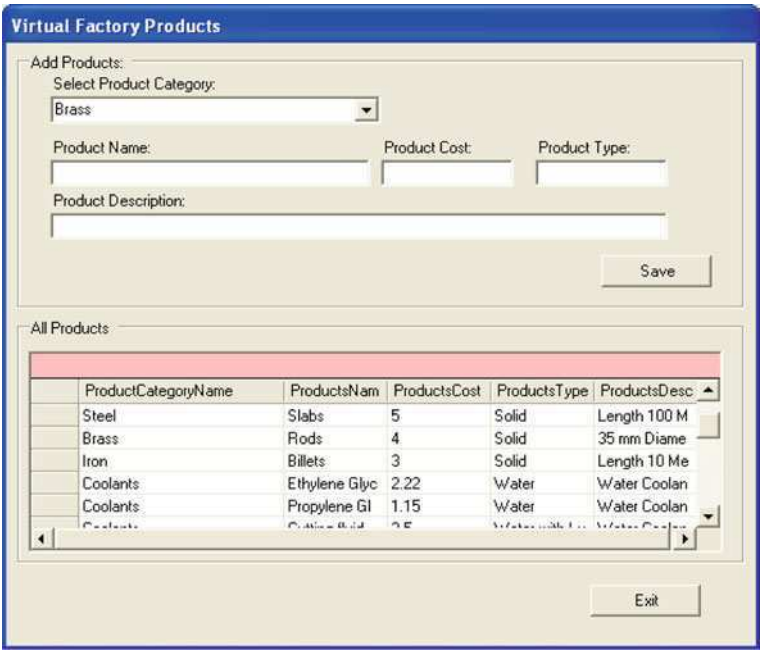


Fig. 11.20 Product Details

II. *Products*
User can add new products that will be used in virtual factory including raw material, consumable items, and non-consumable items. Product category must be created and selected before adding new product (Fig. 11.20).

III. *Products Category*
User can add new product category. Product classification must be created and selected before adding new product category (Fig. 11.21).

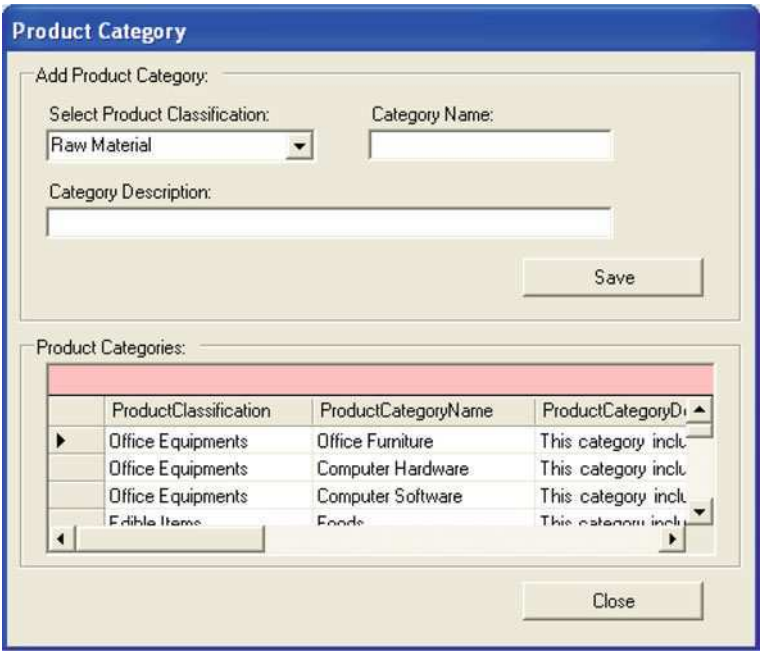


Fig. 11.21 Appending new Product Category

IV. *Purchasing*
After creating product and attach product with supplier, user can purchase those product on based on requirements. User can purchase items or product from purchase history form (Fig. 11.22). When user select a supplier, all attached product with selected supplier will be display in Select Product drop down menu automatically. After purchasing new product the store inventory will be maintain with Debit and Credit entries in Accounts and Finance. After purchase items, items will not directly went to store, instead each item has to pass quality assurance. QA is discussed in detail later in this chapter.

V. *Products Classification*
Product classifications are used to identify the nature of each product such as raw material, consumable items or non-consumable items. User can add new classification from Product Classification window (Fig. 11.23).

11.2.9 Process Planning

Virtual Factory has the process planning capabilities from where user can control the flow of job, materials of job and number of times job should be performed.

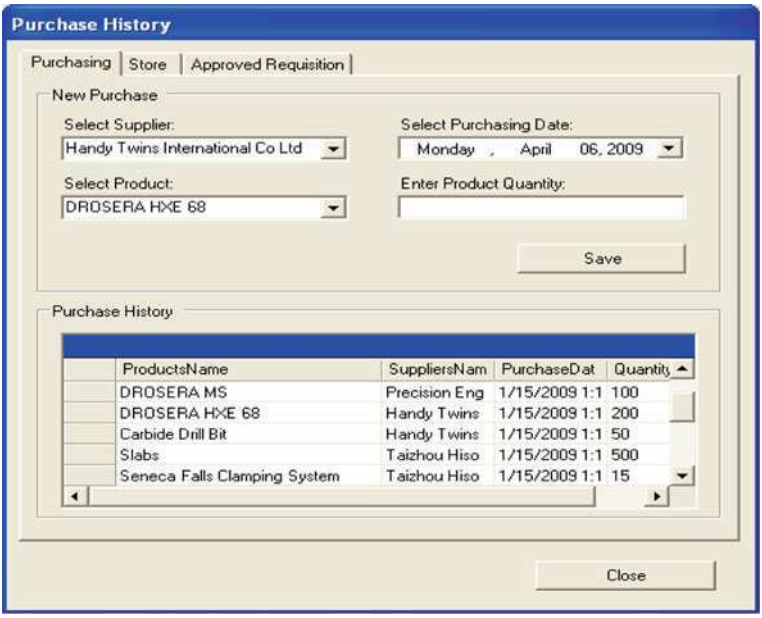


Fig. 11.22 Purchase History

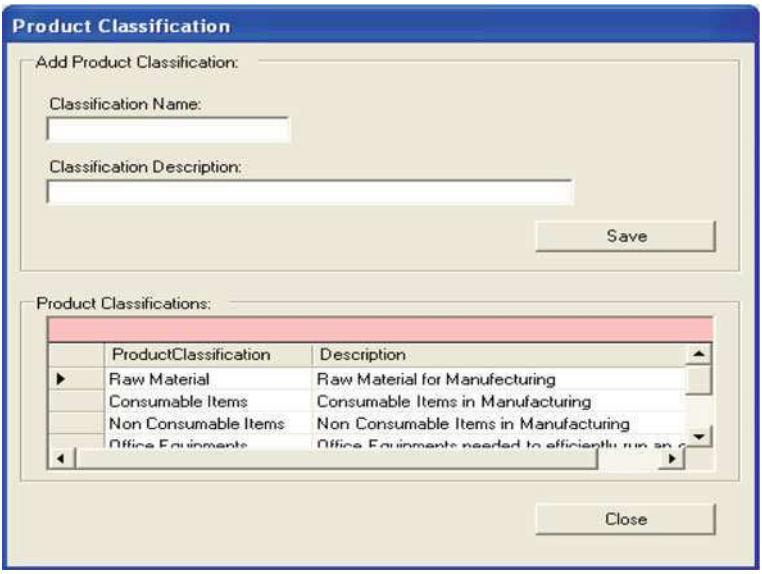
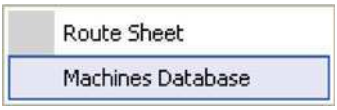


Fig. 11.23 Appending Product Classification

Defining new process planning is the part of creating new project. From Process planning menu user can generate reports that display the process planning information of a particular project.

Process planning menu item has two sub items as shown in the following picture.



Route sheet will display window from where use can generate the following reports.

I. *Procedure Sheet*
User can generate procedure sheet for a particular project. Procedure Sheet will display all procedure related to selected project (Fig. 11.24).

II. *Process Sheet*
User can generate process sheet for a particular project. Process Sheet will display all process related to selected project (Fig. 11.25).

Virtual Factory

Procedure Sheet

Project Code No: IB030
Process Sheet No: 2525
Revision No: 4545
Issued Ac:

Project: Lever Production	Planner: Khalid	Material: Slabs	Batch No. PK-333 S. No. From. PD To. ED			Part Name: Milling Set Part No. PK333			
Qty. to be Processed 5		Job Order No. 12	STD			Actual		Sign	
Date	Op. No.	W.S. No.	Tools / Operations	Met Dtm	S/T Min	Run Min	S/T Min	Run Min	Opr	Sup Q/A
4/28/2008	747	MT 23	Mill Surface 5 mm				3:6:486	3:6:486		
11/26/2008	78	MT30	Mill Sarface 1.5 mm				1:14:814	1:14:814		

Fig. 11.24 Procedure Sheet

Virtual Factory

Process Sheet

Project Code No: IB030
Process Sheet No: 2525
Revision No: 4545
Issued Ac:

Project: Lever Production	Planner: Khalid	Material: Slabs	Batch No. PK-333 S. No. From. PD To. ED		Part Name: Milling Set Part No. PK333			
Qty. to be Processed 5		Job Order No. 12	STD		Actual		Sign	
Date	Pr. No.	Process Detail	S/T Min	Run Min	S/T Min	Run Min	Opr	Sup Q/A
4/28/2008	PK333	Mill Surface 5 mm			3:6:486	3:6:486		
11/26/2008	NP88	Mill Sarface 1.5 mm			1:14:814	1:14:814		

Fig. 11.25 Process Sheet

III. *Route Sheet*
User can generate route sheet for a particular project. Route Sheet will display all Routes related to selected project (Fig. 11.26).

IV. *Machines Database*
User can generate machines report that virtual factory has or used for manufacturing finished goods. User can view full information about the machines, including the available quantity (Figs. 11.27, 11.28, and 11.29).

Virtual Factory

Route Sheet

Project Code No: IB030
Route Sheet No: 1001
Procedure No: 747
Process Sheet No: 2525

Project	Planner	Material	Batch No. PK-333 S. No. From. PD To. ED	Part Name Milling Set Part No. PK333
Lever Production	Khalid	Slabs		
Qty. to be Processed 5		Job Order No. 12		Page No. 1110

Op. No.	Order / Process	W.S. No.	SECT.	Qty. Load	Loading Date	Qty. Accp	Qty. Rej	Operator Sign	Read by Date
	Mill Surface 5	MT 23	Engineerin	5		5	0.00		
	Mill Sarface	MT30	Engineerin	5		2	3.00		

Fig. 11.26 Route Sheet

Milling Machine Specification

S.No	Machine No.	No of Machines	Table Size (mm) X	Y	Additional No of Axes	Automatic Tool Change	No of Tools	Quill Travels	Auxiliary Feature	Networked	Max Distance	No of Spindle Speeds	Speed Range (rpm) Min	Max	No of Feeds
5	GK1	12	150	180	4	Yes	12	52.5	Yes	Yes	1550	23	0	500	44
6	GM3	2	150	180	4	Yes	12	52.5	Yes	Yes	1550	23	0	500	44

Fig. 11.27 Milling Machine Specification

Turning Machine Specification

S. No.	Machine No.	No of Machine	Center Distance (mm)	Clamping Diameter (mm)	Swing Over BED (mm)	Swing Over Cross Slide (mm)	Speed Range Spindle Min	Max	Positioning Min	Max	Torque KN M	Auto Tool Change	Max Drive Rating (KW)	Length of Carriage (mm)	Max Travel of Facing Slide (mm)
1	GK25	5	56	25	55	60	50	600	25	100	10	Yes	150	25	100

Fig. 11.28 Turning Machine Specification

Virtual Factory

Drill Machine Specification

S. No.	Machine No.	No of Machine	Table Size X	Y	Additional Axis	Automatic Tool Change	Number of Tools	Quill Travels	Auxiliary Feature	Networked	Cross Max	Min
1	DK53	3	150	300	2	Yes	1	5.2	Yes	Yes	55 mm	11 mm

Fig. 11.29 Drill Machine Specification

11.2.10 Quality Assurance

Quality assurance module is used to control the quality of raw materials and finished good that are ready for sale. For the control of quality QA test must be performed before any purchase and sales of finished goods.

Quality Assurance menu has two sub items as shown in the following picture.

I. *Sales Quality Control*
User cannot sale finish goods product without the Quality Check Pass. After adding sales order, an entry will be shown in Quality Control Salas window's Quality Control tab. User must select an entry in the following list to Pass Quality Control (Fig. 11.30).
User can view the Sales order status that has passed Quality Check Passed (Fig. 11.31).

II. *Purchasing Quality Control*
User cannot purchase any product or raw material without the Quality Control check of that product. User must select an entry in the following list to pass quality control check (Fig. 11.32).
After passing the quality control check, user can view the list of products that have passed the quality control check (Fig. 11.33).

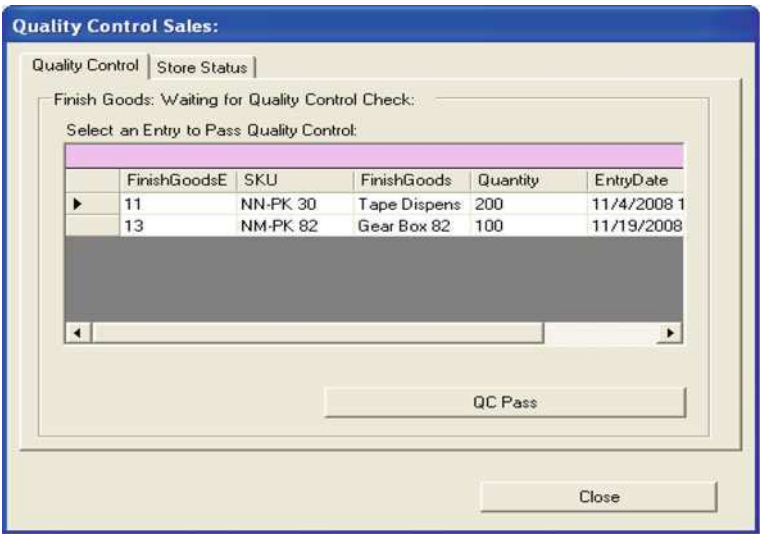


Fig. 11.30 Quality Control Status for products to be sold

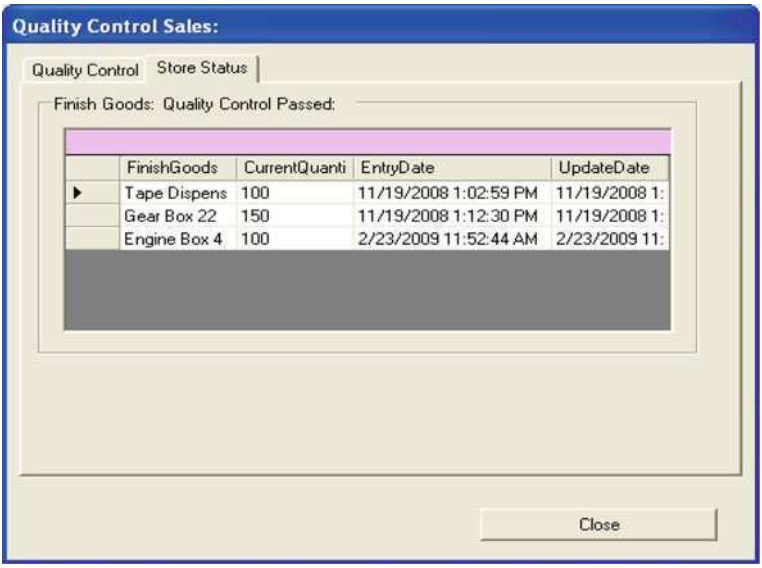


Fig. 11.31 Finished Goods Quality Control Status

Fig. 11.32 Quality Control Status for Purchase Goods

11.2.11 Scheduling

Scheduling is an important tool for manufacturing and engineering, where it can have a major impact on the productivity of a process. In manufacturing, the

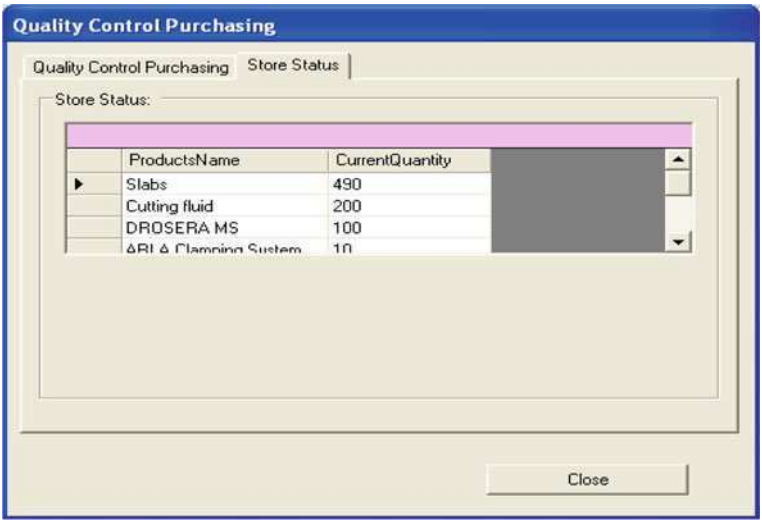


Fig. 11.33 Status of Quality Control Passed Goods in Store

purpose of scheduling is to minimize the production time and costs by projecting which production facility should make it, when to make it, which staff to be manned, and on which equipment to perform the manufacturing operations. Production scheduling aims to maximize the efficiency of the operation and reduce costs.

Forward scheduling is planning the tasks from the date resources become available to determine the shipping date or the due date.

Backward scheduling is planning the tasks from the due date or required-by date to determine the start date and/or any changes in capacity required.

The benefits of production scheduling include:

- Process change-over reduction
- Inventory reduction, leveling
- Reduced scheduling effort
- Increased production efficiency
- Labor load leveling
- Accurate delivery date quotes
- Real-time information

Scheduling report is based on process planning where the user defines how should or what will be the route of a job. Job route is based on cell number. User can view production scheduling report for a selected project. Production scheduling report shows the job schedule (Fig. 11.34).

Virtual Factory

Production Report 4/6/2009

JobName	Jobs Assign	Jobs Completed	Total Completion Time	Jobs Completion Date	Material
JobTwo	5	5	3:6:486	4/28/2008 12:00:00	Plane
JobThree	5	1	0:32:875	4/28/2008 12:00:00	Plane
Production	5	2	1:14:814	11/26/2008 12:00:00	Plane
TESTJOB	5	5	4:36:332	1/22/2009 12:00:00	Slabs

Fig. 11.34 MIS Production Report

Virtual Factory

Production Report 4/6/2009

JobName	Jobs Assign	Jobs Completed	Total Completion Time	Jobs Completion Date	Material
JobTwo	5	5	3:6:486	4/28/2008 12:00:00	Plane
JobThree	5	1	0:32:875	4/28/2008 12:00:00	Plane
Production	5	2	1:14:814	11/26/2008 12:00:00	Plane
TESTJOB	5	5	4:36:332	1/22/2009 12:00:00	Slabs

Fig. 11.35 MIS Production Report

11.2.12 Management Information System

Management Information System in Virtual Factory provides the facility to generate report to view the status of production. The main purpose of production report is to analyze the setup of project. Its shows the project setup performance by having columns of how many jobs are completed and how much time is consumed in manufacturing a job. Products Status reports of MIS show the current status of products in Virtual Factory. MIS Reports window has two tabs, Production Report and Product Status Report.

I. *Production Report*
In production report user can generate report for particular job that has been manufactured in Virtual Factory by running a project. Report can be generated for a single job or all jobs. Production report has very informative columns such as Jobs assigned and how many jobs are completed from assigned job, jobs completion time, and material used to create finished products (Fig. 11.35).

II. *Products Status Report*
In products status report user can generate report for a product category, all categories, and by-product purchase date. In products status report user can view classification of product, category of product, type of product, cost of product, and how much quantity is available for production in virtual factory (Fig. 11.36).

Virtual Factory

Products Status Report

4/6/2009

Classification	Category	Name	Type	Cost		Current Quantity
Raw Material	Steel	Slabs	Solid	5	S	500
Consumable Items	Coolants	Cutting fluid	Water with	2.5	S	200
Consumable Items	Lubricants	DROSERA MS	Machining	8	S	100
Non Consumable Items	Work Holding Devices	AR1.A Clamping	Hydraulic	350	S	10
Consumable Items	Coolants	Mineral Oil	Lubricant	2	S	350
Non Consumable Items	Tools	Carbide Drill Bit	High Speed	8	S	50

Fig. 11.36 MIS Product Status Report

11.3 Virtual Manufacturing System

Virtual Manufacturing System is a software package that allows provision for a proxy user interface. This proxy user interface allows monitoring discrete manufacturing facility for the collection of micro and macro level decision parameters allows managers to deduce optimal decisions and implements this optimal decision by overriding the current setting. Currently the overriding facility is not operational, however, electronic circuits are provided to make this function operational.

Virtual Manufacturing System is aimed at contributing to creation of wealth through searching business over the internet and outsource part or complete factory to derived businesses. Both business to client and business to business operation option are possible. Following sections provide details of each VMS module. The VMS components laving been discussed earlier (Fig. 11.37).

11.3.1 VMS Design

Virtual Manufacturing System Design provides the facility to design the factory layout. In design mode user can add machines, robots, conveyor belts from machines database windows, and can set properties and load appropriate part program of machines by using properties window. In design mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled (Fig. 11.38).

11.3.2 VMS Planner

Virtual Manufacturing System Planning provides the facility to plan the job control and scheduling. In planning mode user can set the job properties including job name, job operation, and Job arrival time, total number of jobs, job material

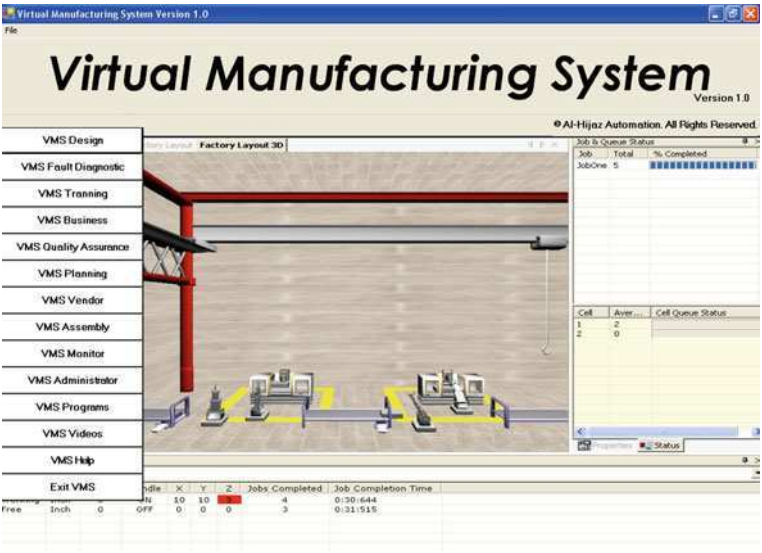


Fig. 11.37 VMS User Interface

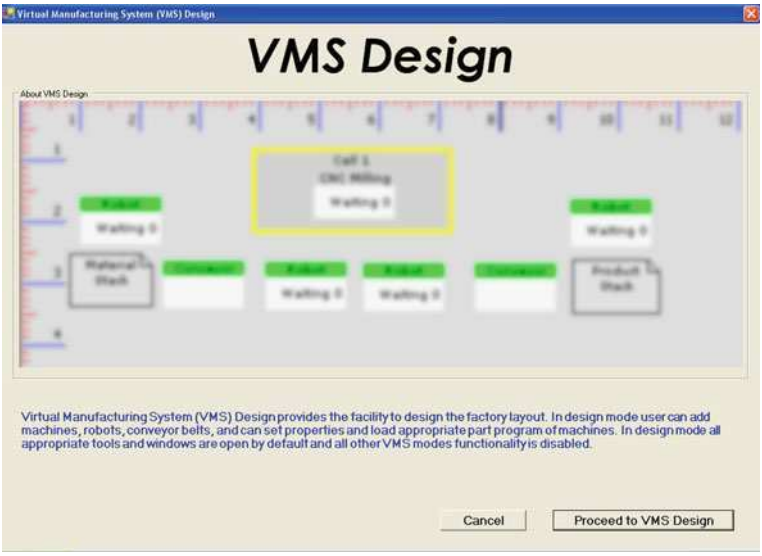


Fig. 11.38 VMS Design User Interface

type, job material source and job routes (Scheduling). In planning mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled. A project must be created in Process Control window in order to define jobs (Fig. 11.39).

Fig. 11.39 VMS Planning User Interface

11.3.3 VMS Monitor

After VMS design and VMS planning, user can use VMS monitor to verify and examine setup of virtual factory. VMS Monitoring provides the facility to monitor the performance of factory setup and status of each machine and job during execution of virtual factory. User also can see scheduling, performance and costing reports offline after execution. In monitoring mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled (Fig. 11.40).

11.3.4 VMS Fault Diagnostic

Virtual Manufacturing System Fault Diagnostic provides the facility to diagnose the faults of machine by using virtual micro-controller feedback system during the execution and monitoring of virtual factory. Each machine has several pins for sending and receiving control signals. User can see status of each pin by green and red LED on Controller Information tab at status window. In fault diagnostic mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled (Fig. 11.41).

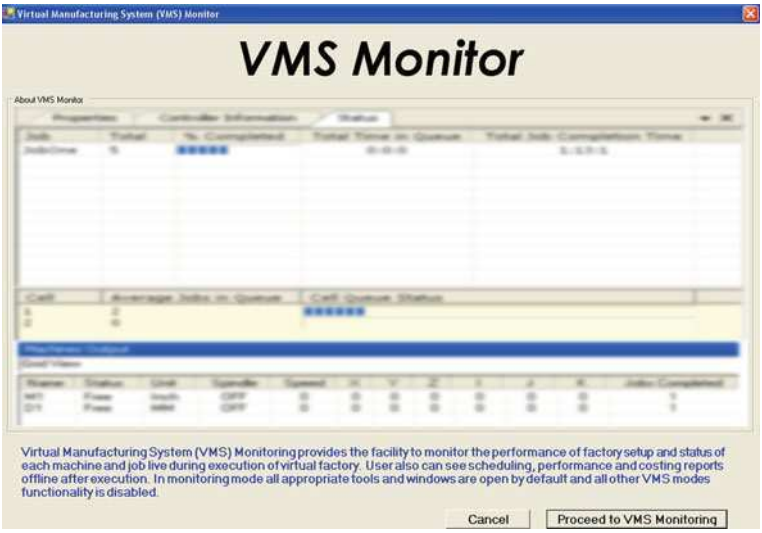


Fig. 11.40 VMS Monitor User Interface

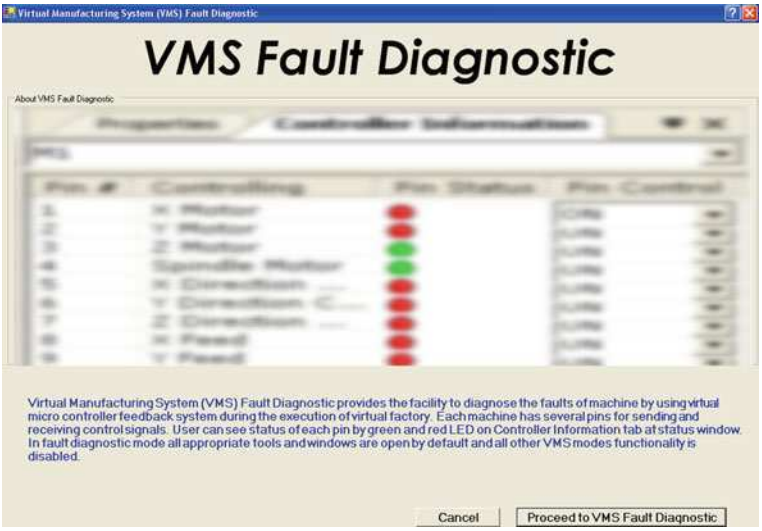


Fig. 11.41 VMS Fault Diagnostic User Interface

11.3.5 VMS Training

Virtual Manufacturing System Training provides the facility to train individuals for CNC based machines. VMS Training includes VR module of CNC Sawing,

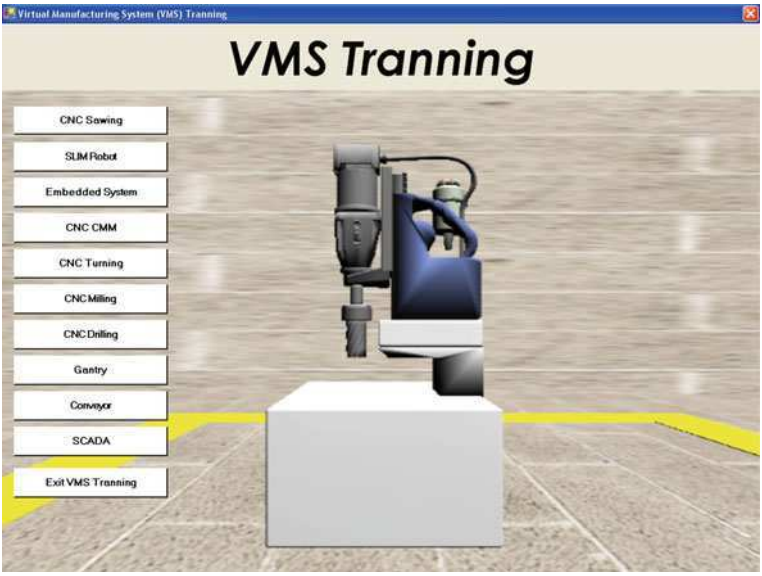


Fig. 11.42 VMS Training User Interface

CNC CMM, CNC Turning, CNC Milling, CNC Drilling, SLIM Robot, Embedded System, Gantry, Conveyer, and SCADA system. Each module of VMS Training is a separate application with its own user interface (Fig. 11.42).

11.3.6 VMS Quality Assurance

Virtual Manufacturing System Quality Assurance provides the facility to control the quality of materials and items at the stage of purchasing and selling. The quality control of purchasing and selling are handled separately. Without quality control pass, no item can inter or exit from inventory database. In Quality Assurance mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled (Fig. 11.43).

11.3.7 VMS Assembly

Assembly is an essential part of the production system. Basically four types of assembly configurations exist. These are Stationary Operator Stationary Workpiece, Stationary Operator Moving Workpiece, Moving Operator Stationary Workpiece and Moving Operator Moving Workpiece. VMS Assembly

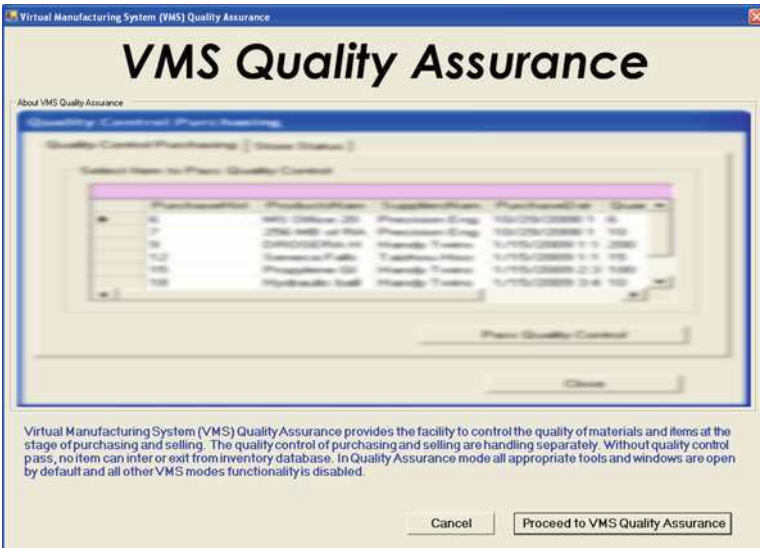


Fig. 11.43 VMS Quality Assurance User Interface

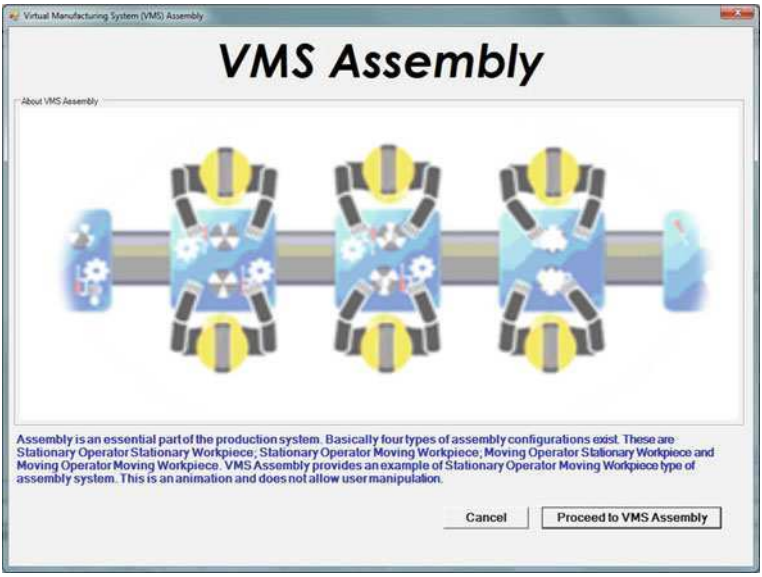


Fig. 11.44 VMS Assembly User Interface

provides an example of Stationary Operator Moving Workpiece type of assembly system. This is an animation and does not allow user manipulation (Fig. 11.44).

Fig. 11.45 VMS Business User Interface

11.3.8 VMS Business

Virtual Manufacturing System Business provides the facility to search manufacturing business online by using semantic web technology and manufacturing ontology. VMS business mode also includes product management, procurement, sales, purchase, accounts, and finance. In business mode all appropriate tools and windows are open by default and all other VMS modes functionality is disabled (Fig. 11.45).

Virtual Manufacturing System Business provides the facility to search business on the Web related to manufacturing by using the semantic Web technology. VMS Business uses a Java-based Semantic Web Agent which implemented a Semantic Web API JENA. IP address of default gateway must be configured in order to get connectivity with internet by using Proxy Server IP option in Business menu (Fig. 11.46).

11.3.8.1 Search Business

User can search business online by using Search Business submenu item in Business Menu. Semantic Web Business windows has Search Name which is mandatory for reporting purposes and two types of searching, predefined companies type and user can write its own searching criteria (Fig. 11.47).

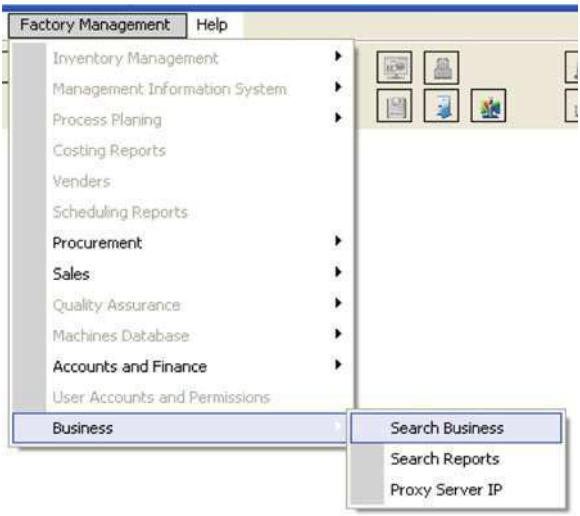


Fig. 11.46 VMS Business Menu

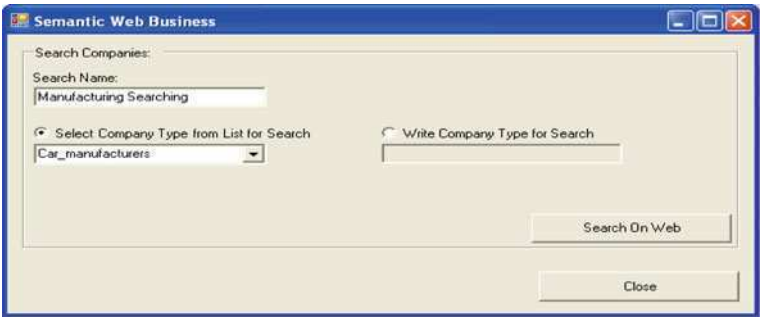


Fig. 11.47 Business Search Window

11.3.8.2 Search Reports

After the completion of search, user can generate reports by using Search Name (Fig. 11.48).

11.3.9 VMS Vender

Virtual Manufacturing System Vender provides the facility to manage vender's database. Venders are suppliers that provide raw material and other products used in virtual factory to produce finished goods. In vender mode all appropriate tools and windows are opened by default and all other VMS modes functionality is disabled (Fig. 11.49).

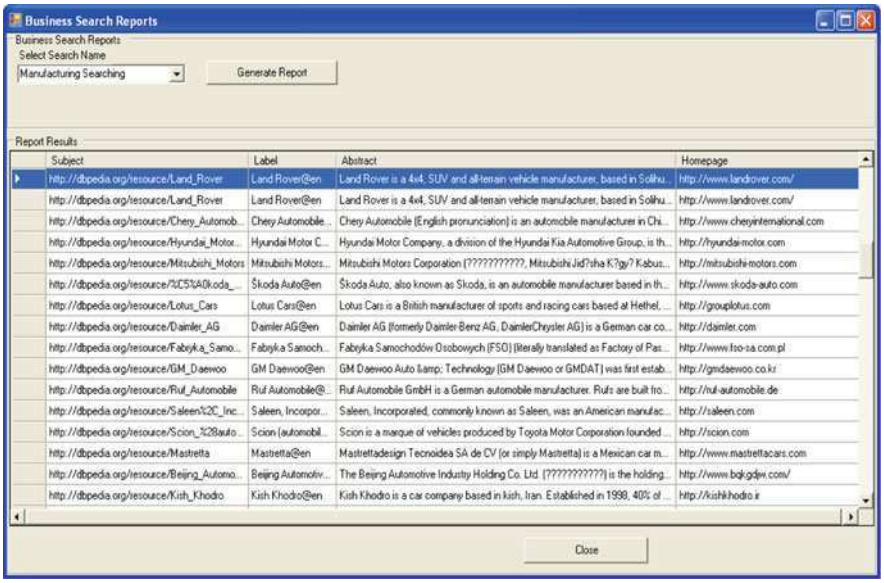


Fig. 11.48 Semantic Web Search Report

Fig. 11.49 VMS Vender User Interface

I. *Vender*
Suppliers are those individuals or companies in virtual factory that provides raw material, tools, and products used in virtual factory to create finished

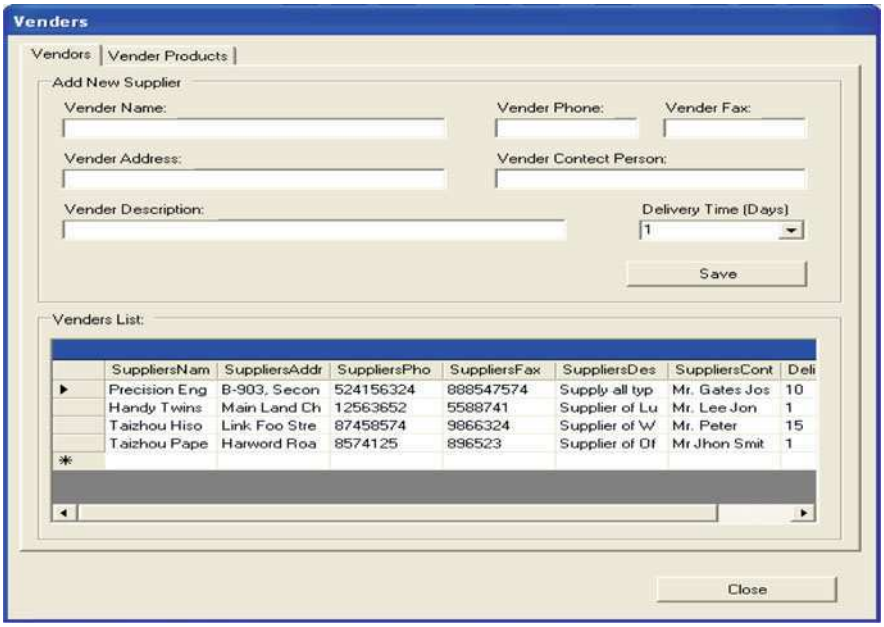


Fig. 11.50 Developing Vender List

goods. Multiple suppliers can provide the same product. In virtual factory supplier and products are created separately. After created suppliers and products. Product can be attached with a supplier or with multiple suppliers from Suppliers Product tab on Suppliers window. User can create or add new supplier in Suppliers window. Supplier must be created before purchasing the product. Delivery time on supplier's window is the number of days in which supplier can provide required products (Fig. 11.50).

From Suppliers Product tab, user can attach supplier with a product. Product must be created before attaching supplier with the product. Multiple suppliers can be attaching with a single product (Fig. 11.51).

11.3.10 VMS Administrator

Virtual Manufacturing System Administrator provides the facility to manage virtual factory's user login accounts, user's permission, and user's designations. After login to virtual factory, permissions set by administrator decides what features of virtual factory cannot or can be access by user.

For the proper administration of virtual factory, administrator can create post, user account and set permission for a particular user. For access the functionality of virtual factory there must be user account. After login, permissions assign by

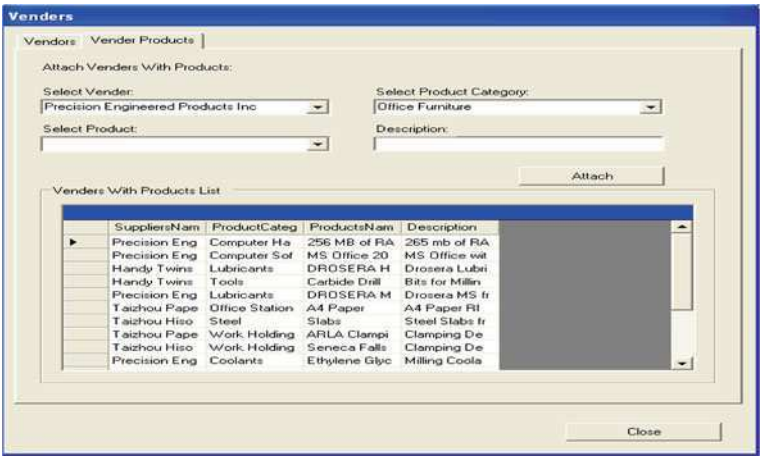


Fig. 11.51 Vender list with category of items

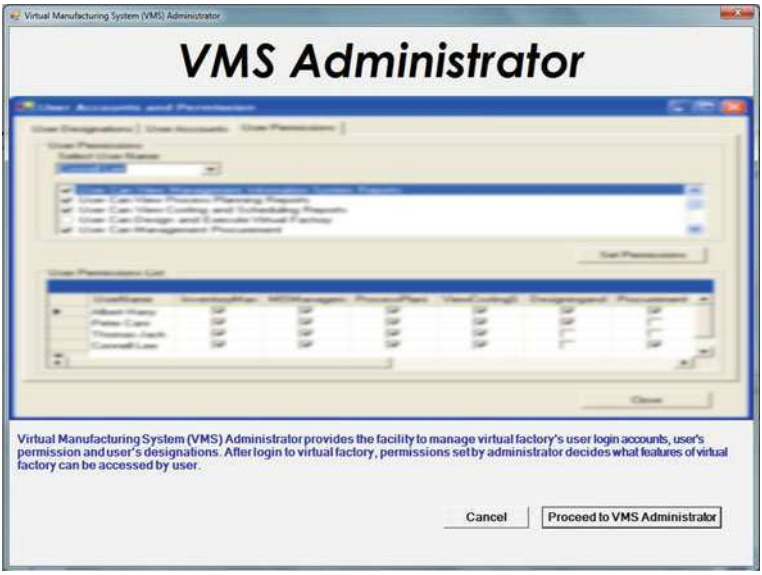


Fig. 11.52 VMS Administrator User Interface

administrator decide what functionality of virtual factory user can use. By default administrator has full control of virtual factory.

From user accounts and permission window, user can manage user designation, user accounts, and permission to use particular feature of Virtual Factory (Fig. 11.52).

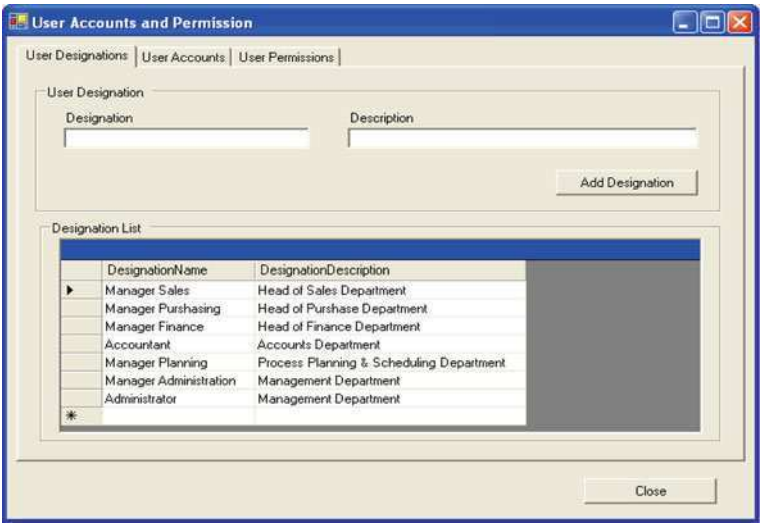


Fig. 11.53 VMS Administrator User Designations

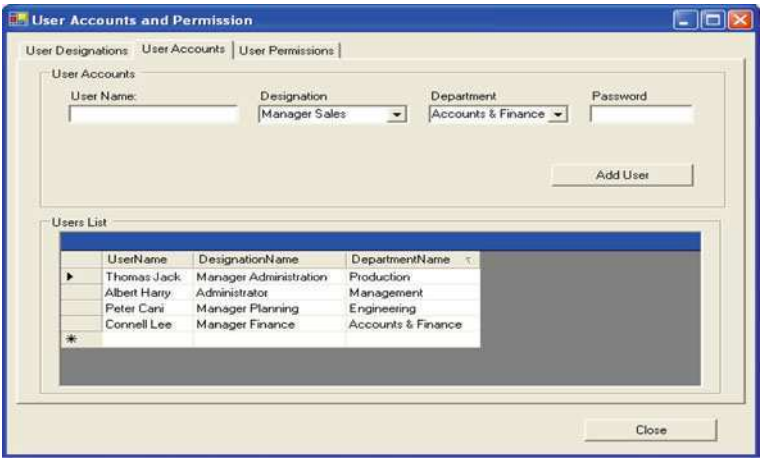


Fig. 11.54 VMS Administrator User Accounts

I. *User Designation*
User can create new designation in the following form as shown in Fig. 11.53.

II. *User Accounts*
User can create new user account by selecting designation, department, login name, and password (Fig. 11.54).

III. *User Permissions*
User can set permission for a user to use virtual factory features (Fig. 11.55).

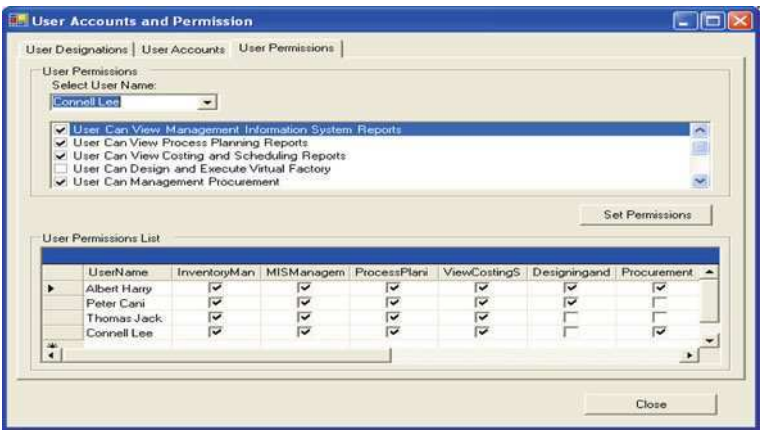


Fig. 11.55 VMS Administrator User Permissions

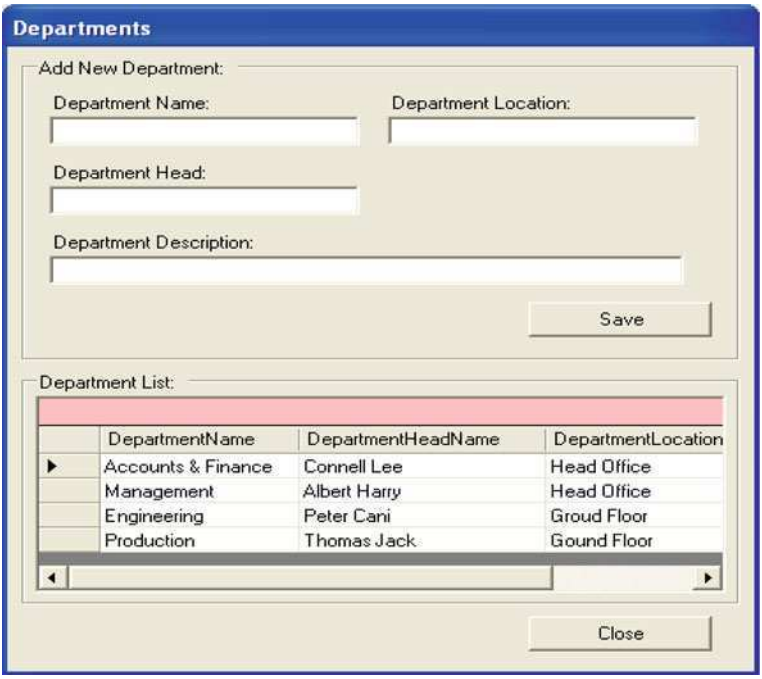


Fig. 11.56 VMS Administrator Developing Departmental Hierarchy

IV. *Department*

Each user in Virtual Factory belongs to a department. Only a user with full admin rights can add new department. Departments are used to categorize user in more efficient way in virtual factory (Fig. 11.56).

Fig. 11.57 VMS Programs User Interface

11.3.11 VMS Programs

Virtual Manufacturing System Programs provides the pre written sample part program for CNC-based machines that are provided with VMS. Part programs provided with VMS Program can also be used with VMS Training (Fig. 11.57).

11.3.12 VMS Videos

Virtual Manufacturing System Videos provides help for VMS in the form of different videos. Videos help can provide better understanding of how to use VMS and machines included in VMS Training (Fig. 11.58).

11.3.13 VMS Help

Virtual Manufacturing System Help provides detailed help on functionality of Virtual Factory, user interface of Virtual Factory and how to setup Virtual Factory (Fig. 11.59).

Fig. 11.58 VMS Videos User Interface

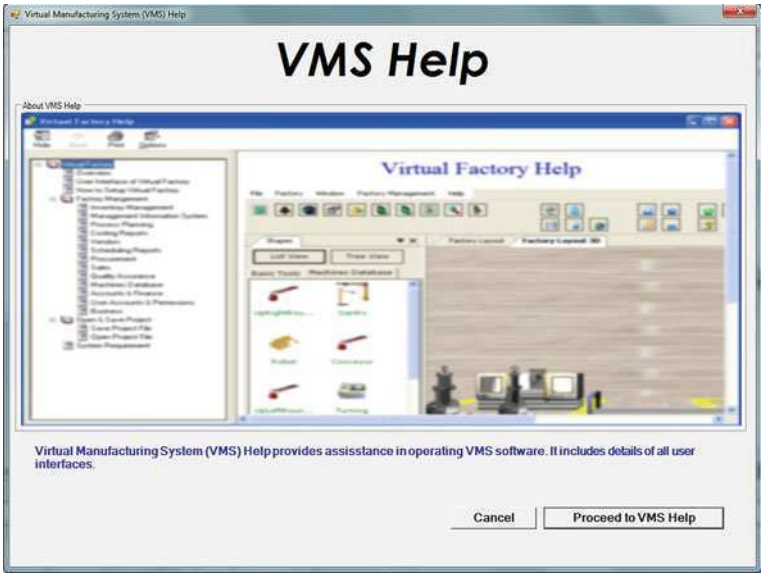


Fig. 11.59 VMS Help User Interface

11.4 AR Design of Virtual Manufacturing Facility

Virtual Manufacturing based on augmented reality relies on same principle as those of its ingredient components. It amalgamates AR for control technologies such as Computer Numerical Control, Programmable Logic Controller, Embedded Systems, Supervisory Control and data acquisition in VMS architecture.

Virtual Factory consists of eight projects:

1. BasicShapes: Consist of classes used in Virtual Factory to design factory layout such as Cell and TextLabel. Cell class represents the specific location marked with yellow color on factory floor to group similar or different machine to setup job shop or cellular manufacturing setup. In Virtual Factory, Cell is also used to setup job routes or job scheduling. TextLabel is used to write note or tag for references on machines at factory floor. Both Cell class and TextLabel class are inherited from Shape class member of VirtualFactoryLib project. BasicShapes is a Dynamic Link Library (DLL) based project. The produced DLL of BasicShapes is referenced in VirtualFactory project.
2. DataAccess: This project is consisted of a class DataHelper, which is used to provide database related functionality to all projects of Virtual Factory. DataHelper class handles connection of database, runs SQL queries for, e.g., Inserts or updates record and return records from database by DataReader object. DataAccess is a DLL based project. The produced DLL of DataAccess is referenced in other projects of Virtual Factory those require database access.
3. EventLog: is consisted of a class used in Virtual Factory to log unwanted events in windows system log. Any internal error or exception of Virtual Factory is logged using Event Log with informative message. Windows event viewer is used to display all events logged by Virtual Factory. EventLog is a DLL based project and used by all other projects of Virtual Factory.
4. Factory3D: Factory3D consists of a Factory3D class. Factory3D class consists of 3D graphics rendering engine that displays factory in 3D format and methods to create 3D machines and controlling direction and movement of 3D objects. Factory3D is also responsible for creating 3D world including walls, roof, floor, camera movement, and start and stop 3D rendering engine. Factory3D is a DLL based project. Produced DLL of Factory3D is referenced by Virtual Factory project to present factory in 3D mode.
5. Factory3DLibrary: consists of classes to create 3D objects including all types of machines, conveyor belts, robots, cells and work pieces. Each type of machine has its own 3D class such as CNC Milling has Milling3D class in Factory3DLibrary project and also each 3D class has properties and methods to control axis and direction of machine and its parts. Factory3DLibrary is a DLL based project referenced in Factory3D project.
6. Machines: consist of classes to create machines such as milling and turning in Virtual Factory. Each machine class has it own compiler or interpreter to produce its functionality and methods to validate and execute part program file;

and, each machine object generate update event in order to control movement of its 3D parts and user interface of Virtual Factory.

7. VirtualFactory: consists of classes to create user interface including menu, tool bars, forms and reports of Virtual Factory. VirtualFactory also has Mediator class that provides Communication Bridge between different projects and parts of Virtual Factory to exchange information. VitualFactory is an executable project and product.exe file to run Virtual Factory application.
8. VirtualFactoryLib: It is the library of core and base classes of Virtual Factory application, common enumerations, attributes, delegates and collections. VirtualFactoryLib is a DLL based project referenced by all other projects.

Table 11.1 shows the member variables, methods, and events of Entity class. The Entity is the parent class of Shape class. All machine classes are inherited from Shape class.

Table 11.2 details the member variables, methods and events of Shape class. All machines on factory floor including robots and conveyors belts are inherited from Shape class. Such as CNC milling machine is an object of CNCMilling class inherited from Shape class. Shape class has generalized methods, properties and events that are common to all machines. For example, Run method implemented in Shape class, called from machine object by VF simulation engine to start a particular machine. Shape initiates several events to update its current running position and status. Following is the brief description of each event of Shape class.

1. MessageDelegate is implemented by MessageArrive event in Shape class. MessageArrive event is used to update the current status of machine or when machine object has some information to display on user interface of Virtual Factory. MessageArrive event used two parameters to pass information. Parameter includes its own object as sender and informative message as message parameter. VF user interface has subscribed MessageArrive event raised by machine to update its interface.
2. DelegateShowMessage which is implemented in Entity class as RaiseMessage method. Its functionality is same as MessageDelegate but it is for general information and not for particular machine information.
3. CompletedDelegate is implemented in Shape class as RaiseComplete method. It is raised when machine completed assigned job and notifies VF about completion of job with three parameters. Parameter includes its own object as sender and a number of completed jobs as counter parameter and the time machine takes to complete a job as JobCompletionTime parameter. Event raised by machine is handled by simulation engine to update VF user interface and route the job to next machine if define in job route. VF simulation engine identifies the machine that raised the event by using sender parameter.
4. RunningStatusDelegate is implemented in Shape class as RaiseRunningStatusArrive event is raised when machine notifies VF to running status with two parameters. Parameter includes its own object as sender and running status of machine, i.e., ON or OFF as status parameter. Event raised by a machine is

Table 11.1 Summary of Virtual Factory Entity class

Class summary			
Class ID: V1			Namespace: VirtualFactory.VirtualFactoryLib
Class name: Entity			Class type: abstract
Is base class?: No			No. of inherited classes: 1
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
recalculateSize	bool	protected	This field checks whether to recalculate the shape size, speed up the rendering
bag	PropertyBag	[NonSerialized] protected internal	This field holds the property bag for the entity. An Entity is an element of the plex, can be either a connection, a connector, a Shape (box, element)
bluepen	Pen	[NonSerialized] protected internal	This field holds the default blue pen
blackpen	Pen	[NonSerialized] protected internal	This field holds the default black pen
mPenWidth	float	protected internal	This field holds the pen width
pen	Pen	[NonSerialized] protected internal	This field holds the default pen
mIsReset	bool	protected internal	This field checks whether the entity is reset
mFontFamily	string	protected	This field holds the font family
mDirection	Direction	protected	This field holds the Direction of Machine OR Face
mEntityName	string	protected	This field holds the entity name
mIsSelected	Boolean	protected	This field checks whether the entity is selected
mHover	Boolean	protected	This field checks whether this entity is being hovered
mUID	Guid	protected	This field holds the unique identifier
mText	string	protected	This field holds the mText or label

(continued)

Table 11.1 (continued)

Field name	Data type	Access modifier	Description
mShowLabel	internal bool	protected	This field checks whether to show the mText label
mTextColor	Color	protected	This field holds the default mText color
mFontSize	float	protected	This field holds the default font size in points
mFont	Font	protected	This field holds the default font for shapes
mSite	IGraphSite	[NonSerialized] protected	This field holds the mSite of the entity

Method name	Return type	Access modifier	Description
GetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected virtual	Function to determine which properties are accessible via the property grid
SetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected virtual	Function to set the values passed by the property grid
InitEntity()	void	protected internal virtual	Function to Initialize the class. This method is necessary when deserializing since various elements like the Pen cannot be serialized to file and have to be, hence, set after deserialization
Paint(Graphics g)	void	protected internal abstract	Function to Create the actual visual element on screen
GetCursor(PointF p)	Cursor	public abstract	Function to Get the cursor for the current position of the mouse
Delete()	void	protected internal abstract	Function to delete the entity from the plex
Hit(RectangleF r)	Boolean	public abstract	Function to Check whether, for the given rectangle, the underlying shape is contained in it
Invalidate()	void	public abstract	Function for Invalidating refreshes part or all of a control
GenerateNewUID()	void	Private internal	Function to regenerate a GUID for this entity
RaiseMouseDown(MouseEventArgs e)	void	Private internal	Function to Raise the mouse down event
RaiseMouseUp(MouseEventArgs e)	void	Private internal	Function to Raise the mouse up event
RaiseMouseMove(MouseEventArgs e)	void	Private internal	Function to Raise the mouse move event
Entity()	–	public	Default Constructor for the entity class, initializes a new GUID for the entity
Entity(IGraphSite mSite)	–	public	Overloaded Constructor Creates a new entity, specifying the mSite

(continued)

Table 1.1 (continued)

Event name	Delegate	Access modifier	Description
OnNewMessage	NewMessage	public	Occurs when the new message arrives
OnMouseDown	MouseEventHandler	public	Occurs when the mouse is pressed on this shape
OnMouseUp	MouseEventHandler	public	Occurs when the mouse is released while above this shape
OnMouseMove	MouseEventHandler	public	Occurs when the mouse is moved above this shape

Table 11.2 Summary of Virtual Factory Shape class

Class summary			
Class ID: V2			Namespace: VirtualFactory.VirtualFactoryLib
Class name: Shape			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 12
Inherited from: Entity			Interfaces with: IAutomataCell, ILayoutNode
Field name	Data type	Access modifier	Description
CurrentStatus	MachineCurrentStatus	protected	This field holds the current status of machine
mMobile	bool	protected	This field holds whether you can move the shape
mTag	object	protected	This field holds volatile all-purpose mTag
mZOrder	int	private	This field holds the z-order of the shapes
mControls	NetronGraphControl Collection	protected	This field holds the array list of .Net mControls the shape contains
mdx	double	protected	This field holds the infinitesimal x-shift used by layout
mdy	double	protected	This field holds the infinitesimal y-shift used by layout
isfixed	bool	protected	This field holds fixed node boolean
mNodeColor	Color	protected	This field holds the default node color
Parent	GraphAbstract	Private internal	This field holds the abstract object to which this shape belongs. A shape cannot exist on its own and will not be drawn outside an abstract structure
mResizable	Boolean	protected	This field tells wether or not the user can resize the mRectangle
connectors	ConnectorCollection	private	This field holds the internal collection of connectors attached to this shape object
mRectangle	RectangleF	Protected internal	This field holds the floating-point mRectangle associated to the shape. It determines the shape's size or boundaries
mShapeTracker	ShapeTracker	private	This field holds the internal tracker object, representing the mRectangle and grips with which one can resize the shape

(continued)

Table 11.2 (continued)

Field name	Data type	Access modifier	Description
mShapeType	ShapeType	protected	This field holds the Shape Type
mMachineType	MachineType	protected	This field holds the Machine Type
machineNumber	string	private	This field holds machine number
mWorkStationUID	Guid	protected	This field holds the work station UID
mWorkStationNumber	int	protected	This field holds the WorkStation Number
FeedingRobotName	string	protected	This field holds the Feeding robot name
OutPutRobotName	string	protected	This field holds the Taking robot name
OperationName	string	protected	This field holds the machine operation name
JobStartConveyor	bool	protected	This field checks if the Conveyor is Job Start Conveyor
JobEndConveyor	bool	protected	This field checks if the Conveyor is Job End Conveyor
FeederConveyorName	string	protected	This field holds the feeding conveyer name
OutPutConveyorName	string	protected	This field holds the Taking conveyer name
machineModel	string	private	This field holds Turning Machine Type
picBox	PictureBox	private	This field holds the picture box
Lines	ArrayList	protected	This field holds the array of lines
_MajorInterval	int	private	This field holds the major interval period
_Scale	int	private	This field holds the scale value
_StartValue	double	private	This field holds the start value
_ScaleX	string	private	This field holds the scaling value in X axis
_ScaleY	string	private	This field holds the scaling value in Y axis
Shape()	–	public	Default constructor for this class
Shape(IGraphSite site) : base(site)	–	public	Overloaded constructor for specifying the site for this shape

Method name	Return type	Access modifier	Description
CalculateValue(float iOffset)	double	private	Function to calculate next value
IsConnectedTo(Shape shape)	bool	public	Function to check if the shape is connected

(continued)

Table 11.2 (continued)

Method name	Return type	Access modifier	Description
RaiseMessageArrive(string message)	void	public	Function to Raise the MessageArrive event
RaiseControllerStatusArrive(Pins PinNumber, PinsStatus PinStatus)	void	public	Function to Raise the ControllerStatusArrive event
RaiseRuningStatusArrive(MachineCurrentStatus status)	void	public	Function to Raise the RunningStatusArrive event
RaisePositionArrive(MachineAxis axis, double dvalue)	void	public	Function to Raise the PositionArrive event
RaiseSpindleArrive(MachineSpindle Status)	void	public	Function to Raise the SpindleArrive event
RaiseStatusArrive(CurrentInformation Info,string svalue)	void	public	Function to Raise the Current Information for machine Event
RaiseAddLine(double x1, double y1, double x2, double y2)	void	public	Function to Raise the Add Line Event for Work Piece
RaiseComplete(int Counter, TimeSpan JobCompletionIime)	void	public	Function to Raise the Complete OR Job Done Event
InitAutomata()	void	public virtual	Function to initiate interface implementation
Insert(GraphAbstract p)	void	Internal Private	Function to Add the shape to a GraphAbstract collection
Delete()	void	internal protected override	Function to Remove the shape from a GraphAbstract. The connectors are deleted as part of this deletion process
Hit(RectangleF r)	Boolean	public override	Function to Return true if the given mRectangle contains the shape (this)
ConnectionPoint(Connector c)	PointF	public virtual	Function to Return the coordinates of a given connector attached to this shape
GetThumbnail()	Bitmap	public virtual	Function to Return the thumbnail of the shape (for the shape viewer)

(continued)

Table 11.2 (continued)

Method name	Return type	Access modifier	Description
Paint(Graphics g)	void	protected internal override	Function to Override the paint method
FitSize()	void	public	Function to Recalculate the size of the mRectangle to fit the size of the text
FitSize(bool square)	void	public	Function to set the shape of the square with the maximum otherwise only the width will be resized to fit the content
Invalidate()	void	public override	Function to Refresh the shape
GetCursor(PointF p)	Cursor	public override	Function to Return the cursor for this shape
MoveControls()	void	public virtual	Function to Move the shape controls (if any) when the shape has been moved
AddProperties()	void	public override	Function to add properties like the size of the shape, the location of the shape, graph specs, the node's color, feeding and output conveyor and robot and machine model
GetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected override	Function to get properties like the size of the shape, the location of the shape, graph specs, the node's color, feeding and output conveyor and robot and machine model
SetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected override	Function to set properties like the size of the shape, the location of the shape, graph specs, the node's color, feeding and output conveyor and robot and machine model
ShapeMenu()	MenuItem[]	public virtual	Function to load and set the menu items
OnKeyDown(KeyEventArgs e)	void	public virtual	Function to handle the keydown event

(continued)

Table 11.2 (continued)

Method name	Return type	Access modifier	Description
OnKeyPress(KeyPressEventArgs e)	void	public virtual	Function to handle the keypress event
RaiseDragEnter(DragEventArgs e)	void	public virtual	Function to Raise the DragEnter event
RaiseDragDrop(DragEventArgs e)	void	public virtual	Function to Raise the DragDrop event

Event name	Delegate	Access modifier	Description
MessageArrive	MessageDelegate	public	Event of New Message
AddLineArrive	AddLineDelegate	public	Event of Add Line
RuningStatusArrive	RunningStatusDelegate	public	Event of Running Status of Machine
ControllerStatusArrive	ControllerStatusDelegate	public	Event of Controller Status
PositionArrive	PositionDelegate	public	Event of New Position of WorkPiece
SpindleArrive	SpindleDelegate	public	Event of Spindle Status
StatusArrive	StatusDelegate	public	Event of Current Status of whole machine
Complete	CompletedDelegate	public	Event of Work Finish or completed

handled by simulation engine to update VF user interface. VF simulation engine identify the machine that raise the event by using sender parameter.

5. SpindleDelegate is implemented in Shape class as RaiseSpindleArrive event raise when machine notify VF to its Spindle status with two parameters. Parameter includes its own object as sender and Spindle status of machine, i.e., ON or OFF as status parameter. Event fired by a machine is handled by simulation engine to update VF user interface. VF simulation engine identifies the machine by using sender parameter.
6. PositionDelegate is implemented in Shape class as RaisePositionArrive event. It is raised when machine notifies VF to current position of work piece on which machine is currently working with three parameters. Parameter includes its own object as sender and position of work piece in X, Y, Z axis as coordinate parameter and value of coordinate as value parameter. Event fired by machine is handles by simulation engine to update VF user interface. VF simulation engine identify the machine that fired the event by using sender parameter.
7. StatusDelegate is implemented in Shape class as RaiseStatusArrive event. It is raised when machine wants to notify its different types of status with three parameters. Parameter includes its own object as sender and speed of spindle, movement of work piece, current tool, current feed, and current status of coolants as current parameter and value of different status as value parameter.

Table 11.3 shows the member variables and method of Job class. Job class is used to represents job performed on machines in VF. The number of job is controlled by the VF user in Process Planning phase. If user decides five jobs to be completed by VF then five separate objects of Job class are created by VF simulation engine. Each job object is identified separately by its JobName property. Job object has several properties to handle Job efficiently by VF simulation engine such as JobOperation, MaterialType, MaterialSource. Variables to handles these properties are shown in Table 11.3.

Table 11.4 defines the member methods of JobCollection class. All the Job class objects created by VF simulation engine are stored temporarily in JobCollection class. JobCollection class is inherited from CollectionBase class which is the built-in class of .Net framework to provide ArrayList functionality. JobCollection class has Add and Remove methods to add or remove Job objects.

Table 11.5 depicts the members variables and method JobScheduling class. The JobScheduling class is used to handles the scheduling of Jobs in VF by using its member properties.

Table 11.6 describes the member variables, methods and events of SimulationProject class. SimulationProject class is the heart of VF and used to simulate the VF. It handles the simulation of VF by implementing RunLoop method which handles each and every activity in VF including arrival of job, departure of job, update inventory database, update MIS reports, controlling machines and 3D animation. SimulationProject has several properties to represent a VF project created during process planning phase in VF such as ProjectName, Department,

Table 11.3 Summary of Virtual Factory Job class

Class summary			
Class ID: V3			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: Job			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None

Field name	Data type	Access modifier	Description
materialSource	string	private	This field holds the Source of Material
materialType	string	private	This field holds the Type of Material
jobName	string	private	This field holds the Name of the Job
jobOperation	string	private	This field holds the Name of the JobOperation
arrivalValue	Int	private	This field holds the Arrival Value
arrivalUnit	string	private	This field holds the Arrival Unit
maxArriveLimited	Int	private	This field holds the Max Arrival Limited Value
workStationCollection	WorkStation Collection	private	This field holds the Collection of WorkStations
availableWorkStations	ArrayList	private	This field holds the Collection of Available WorkStation
includedWorkStations	ArrayList	private	This field holds the Collection of Included Workstation in Job

Method name	Return type	Access modifier	Description
Job()	void	public	Constructor function for the class

Table 11.4 Summary of Virtual Factory JobCollection class

Class summary			
Class ID: V4			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: JobCollection			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: CollectionBase			Interfaces with: None
Method name	Return type	Access modifier	Description
Clone()	object	public	Function to clone the given object
Add(Job job)	Int	public	Function to add one job to the queue
this[int index]	Job	public	Function to return the current job
Contains(object job)	Bool	public	This field check if there exists a job object
Remove(Job job)	void	public	This field removes the selected job

Table 11.5 Summary of Virtual Factory JobScheduling class

Class summary			
Class ID: V5			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: JobScheduling			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
projectName	string	private	This field holds the project name
jobName	string	private	This field holds the job name
operation	string	private	This field holds the operation name
machineOperation	string	private	This field holds the machine operation name
location	string	private	This field holds the location
machine	string	private	This field holds the machine name
schedulingDate	string	private	This field holds the scheduling date
schedulingStartTime	string	private	This field holds the scheduling start time
Method name	Return type	Access modifier	Description
JobScheduling()	void	public	Constructor function for the class

NameOfPlanner. SimulationProject class has Run and SaveProject public methods to start simulation and save entire project to physical file in computer.

SimulationProject has several events to update VF user interface related to current status and simulation internal current activities. The brief description of SimulationProject events are as follows.

1. CurrentJobStatus is used to update the Job status. CurrentJobStatus event implemented by JobStatus delegate in SimulationProject class with four parameters. Parameter includes NameOfJob, NumberOfJob, TotalQueueTime,

Table 11.6 Summary of Virtual Factory SimulationProject class

Class summary			
Class ID: V6			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: SimulationProject			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Field name	Data type	Access modifier	Description
workStationQueue	WorkStationQueue	private	This field holds the WorkStation Queue
SimulationThread	Thread	private	This field holds the Simulation Thread
DataThread	Thread	private	This field holds the Data Thread
MISThread	Thread	private	This field holds the MIS Thread
JobsCompletionTotalTime	DateTime []	private	This field holds the Jobs Completion Total Time
TotalTimeInQueue	DateTime []	private	This field holds the Total Time in Queue
JobsCompleted	int []	private	This field holds the Jobs Completed
NumberOfTask	int []	private	This field holds the Number of Task for each Job
NumberOfWorkStations	int	private	This field holds the Max Number of WorkStation
NumberOfJobs	int	private	This field holds the Max Number of Jobs
JobRoute	int [,]	private	This field holds the Route of Job
EventQueue	ArrayList	private	This field holds the Simulation Event Queue
JobQueue	ArrayList	private	This field holds the Simulation Job Queue
GlobalSimulation EventType	EventType	private	This field holds the Simulation Event Type
VFTimer	SimulationTimer	private	This field holds the Simulation Timer
graphControl	GraphControl	private	This field holds the GraphControl
ErrorInAnyMachine	bool	private	This field holds the Error in any machine
ProjectAlreadyRun	bool	private	This field holds the project run status
projectID	string	private	This field holds the project ID

(continued)

Table 11.6 (continued)

Field name	Data type	Access modifier	Description
nameOfProject	string	private	This field holds the name of the project
nameOfPlanner	string	private	This field holds the name of the planner
organization	string	private	This field holds the name of the organization
department	string	private	This field holds the name of the department
nameOfPart	string	private	This field holds the name of the part
city	string	private	This field holds the name of the city
projectCodeNumber	string	private	This field holds the project code number
batchNumber	string	private	This field holds the batch number
partNumber	string	private	This field holds the part number
routeSheetNumber	string	private	This field holds the route sheet number
procedureNumber	string	private	This field holds the procedure number
from	string	private	This field holds the source machine
to	string	private	This field holds the destination machine
jobOrderNumber	string	private	This field holds the job order number
procedureSheetNumber	string	private	This field holds the procedure sheet number
placeOfIssue	string	private	This field holds the place of issue
pageNumber	string	private	This field holds the page number
revisionNumber	string	private	This field holds the revision number
jobCollection	JobCollection	private	This field holds the job collection
DHandling	DataHandling	private	This field holds the datahandling object
DAccess	DataHelper	private	This field holds the data helper object

Method name	Return type	Access modifier	Description
SaveWizardController()	void	public	Constructor function for the class
SaveProject()	bool	public	Function to save project

(continued)

Table 11.6 (continued)

Method name	Return type	Access modifier	Description
Start()	void	public	Function to start simulation
Stop()	void	public	Function to stop simulation
Run()	void	public	Function to run the machine
GetSimulationEvent()	void	private	Function to get the simulation event
RunLoop()	void	private	Function to run the rendering loop
VFTimer_Tick(object sender, EventArgs e)	void	Private	Function to handle the timer tick event
Arrive(int Job)	void	private	Function to handle the job arrive event
Depart()	void	private	Function to depart the process
Report()	void	private	Function to report the simulation output
SubscribeEvent(Shape shape)	void	private	Function to add event for shape
UnsubscribeEvent(Shape shape)	void	private	Function to remove event for shape
eng_Complete(object sender, int Counter, TimeSpan JobCompletionTime)	void	private	Function to update the job completion information
UpdateData()	void	private	Function to update data
UpdateMIS()	void	private	Function to update MIS
SimulationProject()	void	private	Default Constructor for this class

Event name	Delegate	Access modifier	Description
OnAddJobInQueue	AddJobInQueue	public	Event of Job Add In Queue
OnRemoveJobInQueue	RemoveJobInQueue	public	Event of Job Remove In Queue
OnJobMove	JobMove	public	Event of Job 3D Movement
OnJobComplete	JobComplete	public	Event of Job Arrive
OnJobArrive	JobArrive	public	Event of Job Arrive
CurrentJobStatus	JobStatus	public	Event of Job Completion Status
CurrentQueueStatus	QueueStatus	public	Event of Queue Status

TotalCompletionTime. CurrentJobStatus event raised by SimulationProject's Departure method is handled by Mediator class. After getting CurrentJobStatus event in Mediator class, all four parameters will be passed into 3D animation form by calling UpdateJob method of AnimationForm class for controlling movement of object in 3D view of VF.

2. CurrentQueueStatus is used to update the job queue status. CurrentQueueStatus event implemented by QueueStatus delegate in SimulationProject class with three parameters. Parameter includes CellNumber, Average, and NumberInQueue. CurrentQueueStatus event raised by SimulationProject's Arrive method and SimulationProject's Depart method is handled by Mediator class. After receiving CurrentQueueStatus event in Mediator class, all three parameters will be passed into 3D animation form by calling UpdateQueue method of AnimationForm class for controlling conveyor belts status and work piece 3D object movement.
3. OnAddJobInQueue is used to adding job in queue. OnAddJobInQueue event implemented by AddJobInQueue delegate in SimulationProject class with four parameters. Parameters include TaskNumber, JobName, JobNumber, and WorkStation. OnAddJobInQueue event is raised by SimulationProject's Arrive method when a job is added in a queue. OnAddJobInQueue is handled by Mediator class. After getting OnAddJobInQueue event in Mediator class all four parameters will be passed into 3D animation form by calling AddInQueue method of AnimationForm class for adding 3D object of a work piece in queue.
4. OnJobArrive is used to update the user interface of VF when new job arrives. OnJobArrive event implemented by JobArrive delegate in SimulationProject class with four parameters. Parameters includes NameOfJob, NumberOfJob, MaterialType, and MaterialSource. OnJobArrive event fired by SimulationProject's VF Timer Tick and EngComplete methods when a new job is created. VF Timer Tick method is an event of SimulationTimer class and EngComplete method is an event of Shape class, both events handled by SimulationProject. OnJobArrive event is handled by Mediator class. After receiving OnJobArrive event in Mediator class all four parameters will be passed into 3D animation form by calling CreateWorkPiece method of AnimationForm class for creating a new 3D work piece object.
5. OnJobComplete is used to update the user interface of VF when a job is complete by any machine. OnJobComplete event implemented by JobComplete delegate in SimulationProject class with three parameters. Parameters includes Shape, JobName and JobNumber. OnJobComplete event is raised by SimulationProject's EngComplete method. EngComplete method is an event of Shape class handled by SimulationProject class. OnJobComplete event is handled by Mediator class. After receiving OnJobComplete event in Mediator class all three parameters will be passed into 3D animation form by calling WorkComplete method of AnimationForm class for controlling robot to pick completed work piece from machine.
6. OnJobMove is used to update the movement of work piece from machine to machine. OnJobMove event is implemented by JobMove delegate in

SimulationProject class with three parameters. Parameter includes Shape, JobName and JobNumber. OnJobMove event is raised by Arrive and Depart methods of SimulationProject. OnJobMove event handled by Mediator class. After getting OnJobMove event in Mediator class, all three parameters will be passed into 3D animation form by calling MoveWorkPiece method of AnimationForm class for controlling the movement of work piece.

7. OnRemoveJobInQueue is used to removing job from queue. OnRemove JobInQueue event implemented by RemoveJobInQueue delegate in SimulationProject class with four parameters. Parameters includes TaskNumber, JobName, JobNumber and WorkStation. OnRemoveJobInQueue event is raised by SimulationProject's Depart method when a job is being removed from queue. OnRemoveJobInQueue is handled by Mediator class. After getting OnRemoveJobInQueue event in Mediator class all four parameters will be passed into 3D animation form by calling RemoveQueue method of AnimationForm class for adding 3D object of a work piece in queue.

Table 11.7 shows the member variables and methods of SimulationTimer class. SimulatioTimer class provides the system clock that generates the clock interrupts for VF simulation. SimulationTimer inherited from System.Form.Timer class.

Table 11.7 Summary of Virtual Factory SimulationTimer class

Class summary			
Class ID: V7		Namespace: VirtualFactory.VirtualFactoryLib.SimulationEngine	
Class name: SimulationTimer		Class type: [Serializable]	
Is base class?: No		No. of inherited classes: 0	
Inherited from: System.Windows.Forms.Timer		Interfaces with: None	
Field name	**Data type**	**Access modifier**	**Description**
TimerName	string	private	This field holds the timer name
materialType	string	private	This field holds the material type
materialSource	string	private	This field holds the material source
TimerTag	object	private	This field holds the timer tag
TimerCounter	int	private	This field holds the timer counter
TimerNumber	int	private	This field holds the timer number
TimerLoop	int	private	This field holds the timer loop
TimerTotalTask	int	private	This field holds the timer total task
Method name	**Return type**	**Access modifier**	**Description**
OnTick(EventArgs e)	void	protected override	Function to handle the event of On tick timer
SimulationTimer()	–	public	Default Constructor function for the class
SimulationTimer():base()	–	public	Overloaded Constructor function for the class

Table 11.8 Summary of Virtual Factory GraphControl class

Class summary			
Class ID: V8			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: GraphControl			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: ScrollableControl			Interfaces with: IGraphSite, IGraphLayout
Field name	Data type	Access modifier	Description
mAllowDeleteShape	bool	protected	This field holds the boolean value to allow delete shape
mAllowMoveShape	bool	protected	This field holds the boolean value to allow move shape
mAllowAddShape	bool	protected	This field holds the boolean value to allow add shape
mAllowAddConnection	bool	protected	This field holds the boolean value to allow add connection
freeArrows	FreeArrowCollection	protected	This field holds the free, non-graph dependent arrow, useful for debugging purposes
insertionPoint	PointF	protected	This field holds the insertion point
format	DataFormats.Format	static	This field holds the custom format (see bug problem in the copy/paste code
bag	PropertyBag	[NonSerialized] protected	This field holds the property bag of the canvas
mContextMenu	ContextMenu	protected	This field holds the mContextMenu of the canvas
CtrlShift	bool	protected	This field holds the Shortcut key
workingSize	Size	protected	This field holds the working size
connectionPath	string	protected	This field holds the default path style of the new connections
connectionEnd	ConnectionEnds	protected	This field holds the default connection end
mGridSize	int	protected	This field holds the grid size
mShowGrid	bool	protected	This field holds the boolean value to show grid?
mSnap	bool	protected	This field holds the boolean value to check whether to mSnap to the grid

(continued)

Table 11.8 (continued)

Field name	Data type	Access modifier	Description
lastAddedShapeKey	string	protected	This field holds the last shape key for easy addition via ALT + click
mEnableContextMenu	bool	protected	This field checks whether to enable the context menu
mRestrictToCanvas	bool	protected	This field checks whether to restrict the shapes to the canvas size
mLibraries	GraphObjectsLibraryCollection	protected	This field holds the shape mLibraries
ctrlKey	const int	protected	This field holds the constant used when checking for the CTRL key during drop operations
layoutFactory	LayoutFactory	protected	This field holds the reference to the layout factory which organizes the layout algo's
mEnableLayout	bool	protected	This field checks is the layout active?
ts	ThreadStart	protected	This field holds the thread pointer which will run on the layout thread
thLayout	Thread	protected	This field holds the thread for laying out the graph
mAutomataPulse	int	protected	This field holds the timer interval in milliseconds that updates the state of the automata in the plex
mBackgroundType	CanvasBackgroundTypes	protected	This field holds the kinda background the canvas has
mGradientBottom	Color	protected	This field holds the color of the lower gradient part
mGradientTop	Color	protected	This field holds the color of the upper gradient part
mBackgroundImagePath	string	protected	This field holds the image path for the background
mBackgroundColor	Color	protected	This field holds the background color
currentZoomFactor	float	protected	This field holds the currentZoomFactor factor
mFileName	string	protected	This field holds the current filename of the plex
extract	GraphAbstract	protected	This field holds the extract contains the data of the plex and its structure
shapeObject	Shape	protected	This field holds the volatile object not connected to extract structure, used for the current added or selected shape

(continued)

Table 11.8 (continued)

Field name	Data type	Access modifier	Description
connection	Connection	protected	This field holds the volatile connection, used to manipulate the current connection
Hover	Entity	protected	This field holds the Entity with current mouse focus. Is automatically set by the HitHover and HitEntity handlers
selector	Selector	protected	This field holds the object for tracking selection state, represents the dashed selection rectangle
mDoTrack	Boolean	protected	This field indicates track mode, i.e., moving a shape around
transmissionTimer	System.Windows.Forms.Timer	protected	This field holds the timer controlling the refresh rate of the graph
_Scale	int	private	This field holds the scale value
_bDrawLine	bool	private	This field holds the boolean value to draw line
_iMouseXPosition	int	private	This field holds the mouse X position
_iMouseYPosition	int	private	This field holds the mouse Y position
_StartValue	double	private	This field holds the start value
_RulerCreated	bool	private	This field holds the boolean value to check if the ruler has been created
_iHeight	int	private	This field holds the height of the graph control
_iWidth	int	private	This field holds the width of the graph control
_Orientation	enumOrientation	private	This field holds the orientation of the graph control
_ScaleMode	enumScaleMode	private	This field holds the scale mode
_RulerAlignment	enumRulerAlignment	private	This field holds the ruler alignment
_i3DBorderStyle	Border3DStyle	private	This field holds the border style
_iMajorInterval	int	private	This field holds the major interval
_iNumberOfDivisions	int	private	This field holds the number of divisions
_DivisionMarkFactor	int	private	This field holds the division mark factor
_MiddleMarkFactor	int	private	This field holds the middle mark factor
_ZoomFactor	double	private	This field holds the zoom factor

(continued)

Table 11.8 (continued)

Field name	Data type	Access modifier	Description
_bMouseTrackingOn	bool	private	This field holds the boolean value to set mouse tracking ON
_VerticalNumbers	bool	private	This field holds the vertical numbers
_bShowRuler	bool	private	This field holds the boolean value to show ruler
_bActualSize	bool	private	This field holds the boolean value to set to actual size
_DpiX	float	private	This field holds the pi X value

Method name	Return type	Access modifier	Description
Dispose(bool disposing)	void	protected override	Function to Clean up any resources being used
CalculateValue(int iOffset)	double	private	Function to calculate next value
OnScaleModeChanged (ScaleModeChangedEventArgs e)	void	protected	Function to handle OnScaleModeChanged event
DefaultScale(enumScaleMode iScaleMode)	int	private	Function to Set scaling of the ruler by determining the *relative* proportions of each scale (foot or meter)
DefaultMajorInterval (enumScaleMode iScaleMode)	int	private	Function to Set interval of the ruler by determining the *relative* proportions of each scale (foot or meter)
Start()	int	private	Function to Return the start position of the ruler based on the selected borderstyle
Line(Graphics g, int x1, int y1, int x2, int y2,System.Drawing.Color lineColor)	void	private	Function to Draw the solid line using Pen
OnHooverValue(HooverValueEventArgs e)	void	protected	Function to handle event to set the HooverValue to true if mouse is at hover position
DrawValue(Graphics g, int iValue, int iPosition, int iSpaceAvailable)	void	private	Function to Draw the text on the ruler with provided orientation, alignment and position
DivisionMark(Graphics g, int iPosition, int iProportion)	void	private	Function to draw a line division and is affected by the RulerAlignment setting

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
Offset()	int	private	Function to Return the offset position of the ruler based on the selected borderstyle
RenderTrackLine(Graphics g)	void	public	Function to Optionally render Mouse tracking line
DrawControl(Graphics g)	void	private	Function to Draw the control (shape, connector, connection) over the ruler
ZoomPoint(Point originalPt)	Point	public	Function to Zoom given point
UnzoomPoint(Point originalPt)	Point	public	Function to Unzoom given point
ZoomPoint(PointF originalPt)	PointF	public	Function to Zoom given point to the current zoom factor
UnzoomPoint(PointF originalPt)	PointF	public	Function to Unzoom given point to the current zoom factor
ZoomRectangle(RectangleF originalRect)	RectangleF	public	Function to Zoom a given rectangle to the current zoom factor
UnzoomRectangle(RectangleF originalRect)	RectangleF	public	Function to Unzoom a given rectangle to the current zoom factor
ZoomRectangle(Rectangle originalRect)	Rectangle	public	Function to zoom given rectangle
UnzoomRectangle(Rectangle originalRect)	Rectangle	public	Function to Unzoom given rectangle
InvalidateRectangle(Rectangle rect)	void	public	Function to Let the site invalidate the rectangle
HitEntity(PointF p)	Entity	protected	This is the event handler for the mousemove and returns an entity when it hits anything from the plex
HitHover(PointF p)	void	protected	Function to simply check if the HitEntity returns something different than the previous hit. If it is something different than the previous hit it sets the internal "Hover" entity to the new one
Deselect()	void	public	Function to Collect all controls (shapes, connectors and connections) of the abstract and sets the IsSelected to False
SetCursor(PointF p)	void	protected	Function to return a cursor for the given point
OnMouseDown(MouseEventArgs e)	void	protected override	Function to Handle the mouse down event
AddConnectionPoint(object sender, EventArgs e)	void	private	Function to add connection point

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
RemoveConnectionPoint(object sender, EventArgs e)	void	private	Function to remove connection point
OnMouseUp(MouseEventArgs e)	void	protected override	Function to Handle the mouse up event
OnMouseMove(MouseEventArgs e)	void	protected override	Function to override the Mouse move event handler if the mousemove is a dragging action on a tracker grip it will enlarge the tracker
PaintArrow(Graphics g, PointF startPoint, PointF endPoint,Color lineColor, bool filled, bool showLabel)	void	public	Function to Paint an arrow
OnPaint(PaintEventArgs e)	void	protected override	Function to Override the paint method and calls the paint method of the extract structure which will loop through all the shapes, connections and connectors
OnPaintBackground(PaintEventArgs e)	void	protected override	Function to Paint the background of the canvas
OnDragDrop(DragEventArgs drgevent)	void	protected override	Function to Store the data as a string so that it can be accessed from the mnuCopy and mnuMove click events
OnDragEnter(DragEventArgs drgevent)	void	protected override	Function to Show the standard Move or Copy icon
OnNewGraph(object sender, EventArgs e)	void	protected override	Function to handle the OnNewGraph event
OnSelectAll(object sender, EventArgs e)	void	protected override	Function to handle OnSelectAll event
OnCut(object sender,EventArgs e)	void	protected override	Function to handle OnCut event
OnCopy(object sender,EventArgs e)	void	protected override	Function to handle OnCopy event

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
OnPaste(object sender,EventArgs e)	void	protected override	Function to handle OnPaste event
OnProperties(object sender,EventArgs e)	void	protected override	Function to Raise Show Properties Form
On3DModel(object sender, EventArgs e)	void	protected override	Function to Raise 3DModel
OnJobAdd(object sender,EventArgs e)	void	protected override	Function to Show Job Form
OnDelete(object sender,EventArgs e)	void	protected override	Common gate for a delete action on the plex. Deletes the selected shapes and goes via the history (of course)
AddToContextMenu(ShapeSummary summary)	void	protected override	Function to Add the given shape summary to the context menu
OnContextMenuItem(object sender, EventArgs e)	void	protected override	Function to Fire when a contextmenu item is selected
AddBaseMenu()	void	protected override	Function to Adds a base menu containing Delete and Properties menus
ResetToBaseMenu()	void	protected override	Function to Reset the menu to its base state
PrintCanvas(object Sender, PrintPageEventArgs e)	void	public	Function to print canvas
StartAutomata()	void	public	This function starts the heart beat of the automata data flow, it simply calls the start method of the timer
StopAutomata()	void	public	This function stops the heart beat of the automata data flow
OnTransmissionTimer(object Sender, EventArgs e)	void	public	This function is the event handler when the transmission timer fires. Handles the transmissions of data over the connections
StartLayout()	void	public	Function to Start the graph layout thread using a previously defined algorithm

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
StopLayout()	void	public	Function to Stop the graph layout thread
SaveGraphMLAs(string filePath)	void	public	Function to save graph MLAs
BitBlt(IntPtr hdcDest, int nXDest, int nYDest, int nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc, System.Int32 dwRop)	bool	private static extern	A function imported from the gdi32.dll library to take a snapshot of a (actually any) control
SaveImage(string path)	void	public	Function to Save an image of the canvas in JPG format
LoadLibraries()	void	public	Function to Load all library classes from VirtualFactoryLibs
AddLibrary(string path)	void	public	Function to Add a shape library to the collection
ImportShapes(string path)	void	protected	Function to import shape from library
SaveAs(string mFileName)	bool	public	Function to Save the plex to the selected path using the wonderful.Net serialization
OutputInfo(object obj)	void	protected internal	Function to output information
RaiseShowProps(PropertyBag props)	void	public	Function to raise property form
OnKeyPress(object sender, KeyPressEventArgs e)	void	private	Function to Handle the key press
OnKeyDown(object sender, KeyEventArgs e)	void	private	This is the general event handler for key events
OnMouseWheel(MouseEventArgs e)	void	protected override	Function to Move the scrollbar up-down
AddShape(Shape sob, PointF position)	Shape	public	Function to Add a given shape to the canvas at the specified position
CreateConveyorName()	string	private	Function to create conveyor
CreateRobotName()	string	private	Function to create robot
AddShape(string key, PointF position)	Shape	public	Function to Add a key shape to the canvas at the specified position
RemoveShape(Shape shape)	void	public	Function to remove shape
AddShape(Shape sob)	Shape	protected	Function to Add a given shape to the canvas at the center
AddNode(BasicShapeTypes shapeType, string label)	Shape	public	Function to Add a special node with the specified label at the center

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
AddNode(BasicShapeTypes shapeType, string label, PointF position)	Shape	public	Function to Add a special node with the specified label at the specified position
AddNode()	Shape	public	Function to Add a simple node at the center
AddNode(string label)	Shape	public	Function to Add a simple node with the specified label at the center
AddNode(string label,PointF position)	Shape	public	Function to Add a simple node with the specified label at the specified position
AddNode(PointF position)	Shape	public	Function to Add a simple node at the specified position
AddBasicNode(BasicShapeTypes shapeType, string nodeLabel, PointF position)	Shape	private	Function to Add a special node with the specified label at the specified position
ConnectionExists(Connector b,Connector e)	Boolean	protected	Function to Return whether for the given two connectors there is a connection
GetMachinesByWorkStationNumber(int WorkStationNumber)	ShapeCollection	public	Function to Return the ShapeCollection with the given WorkStation number
ValidateShapeByName(string ShapeName)	bool	public	Function to Return the true if found the shape otherwise false
GetShapeByWorkStationUID(Guid UID)	ShapeCollection	public	Function to Return the ShapeCollection with the specified WorkStation UID
GetWorkStaionNumberByName(string WorkStationName)	int	public	Function to Return the WorkStation Number
GetNumberOfMachinesByWorkStationName (string WorkStationName)	Int	public	Function to Return the Number of Machines Hold by WorkStation
GetGantryShape()	Shape	public	Function to Return the Gantry
GetConveyorByName(string ConveyorName)	Shape	public	Function to Return the Shape Object of Conveyor Belt Machine
GetSawing()	Shape	public	Function to Return the Shape Object of Sawing Machine
GetConveyors()	ShapeCollection	public	Function to Return the ShapeCollection with the ShapeType is ShapeType.Conveyor

(continued)

Table 11.8 (continued)

Method name	Return type	Access modifier	Description
GetRobots()	ShapeCollection	public	Function to Return the ShapeCollection with the ShapeType is ShapeType.Robot
GetWorkStations()	ShapeCollection	public	Function to Return the ShapeCollection with the ShapeType is ShapeType.WorkStation
GetNumberOfWorkStations()	int	public	Function to Return the Number of WorkStations
GetEntity(string UID)	Entity	public	Function to Return the entity with the specified UID
GetEntity(PointF p)	Entity	public	Function to Return the entity with the specified Point
Copy()	void	public	Function to Implement the copy function to the clipboard
Paste()	void	public	Function to The clipboard paste action
AddEdge(Connector From, Connector To)	Connection	public	Function to specify a connection: by specifying the two connectors
SelectAll(bool includeAll)	void	public	Function to Select all shapes and things of the plex
GetShapeInstance(string shapeKey)	Shape	public	Function to get shape
Clear()	void	public	Function to Clear the canvas of nodes and edges, goes via the clearing of the attached GraphAbstract class
GetNodeByLabel(string label)	Shape	public	Function to get the selected node
New()	void	public	Function to Create a blank new canvas
GetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected virtual	Function to Determine which properties are accessible via the property grid
SetPropertyBagValue(object sender, PropertySpecEventArgs e)	void	protected virtual	Function to Set the values passed by the property grid
AddProperties()	void	protected	Function to add properties
GraphControl()	–	public	Constructor function for the class

SimulationTimer has the OnTick event that control the Simulation engine's job arrive frequency.

Table 11.8 shows the member variables and methods of GraphControl class. GraphControl class represents the VF in 2D mode and provides the facilities to design a factory layout by providing drag and drop features. GraphControl is a windows user interface control inherited from System.Form.ScrollableControl class. GraphControl has several properties such as BackColor to change factory layout color, Grid to show grid on top of factory layout, Grid Size to change to size of grid, Scale to change to scale properties.

When user drags a machine from machine database and drop into factory layout, GraphControl raised OnShapeAdded event which is implemented by NewShape delegate with two parameters. Parameter of OnShapeAddedd includes sender and shape. After creating new machine it is placed into MachinesCollection array for use in simulation. All methods of GraphControl that is start with "Get" such as GetRobot, GetWorkStations, GetShapesByWorkStationID, GetConveyors in GraphControl class uses MachineCollection to search through required type of objects.

GraphControl is just a place holder to hold machines and all types of equipments and return back required machines as demanding during execution of VF simulation.

Table 11.9 presents the member variables, method and event of ProcessForm class. ProcessForm class represents the process planning window, where user can plan VF processes. ProcessForm class uses ProcessTree control to add project and multiple jobs on each project.

Table 11.10 explains the member variables and methods of Mediator class. Mediator class acts as a bridge for passing information between different classes belonging to different projects of VF. All VF windows such as factory layout window, property window, process planning window, information output window and others are created in Mediator class. VF user interface can access those windows by respective properties. All events of SimulationProject are handled by Mediator. After getting events from SimulationProject, Mediator passes that information to a specific window. For example queue status window can change queue status according to latest information it received from Mediator class.

Table 11.11 defines the member variables and methods of Cell class. Cell class is used to represents the specific area where factory layout designer group similar or dissimilar machines to form a job shop or cellular manufacturing system. Each cell is identified separately by CellNumber property. When factory layout designer drop machine into cell, machine automatically detect the cell number to attach itself with Cell.

Table 11.12 outlines the member variables and methods of DataHelper class. DataHelper class represents the Database Link Layer (DLL) functionality. DataHelper class hides the database handling complexities and provides two properties and seven methods to handle database. DBName and DBPath properties are used to set the VF database name and path of database file in computer. SQL query can be run directly by using RunSQL method of DataHelper class. Connect and

Table 11.9 Summary of Virtual Factory ProcessForm class

Class summary			
Class ID: V9			Namespace: VirtualFactory.Factory
Class name: ProcessForm			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: DockContent			Interfaces with: None
Field name	Data type	Access modifier	Description
Tree	ProcessTree	private	This field holds the process tree object
mediator	Mediator	private	This field holds the mediator class object
label1	System.Windows.Forms.Label	private	This field holds the windows label
components	System.ComponentModel.IContainer	private	This field holds the components class
Project	SimulationProject		This field holds the project
Method name	Return type	Access modifier	Description
Dispose(bool disposing)	void	protected override	Function to Clean up any resources being used
InitializeComponent()	void	private	Required method for Designer support
Tree_OnShowMessage (string message)	void	public	Function to call OnProcessShowMessage event
ProcessForm(Mediator mediator)	–	public	Constructor function for the class
Event name	Delegate	Access modifier	Description
OnProcessShowMessage	DelegateShowMessage	public	Event of On Process Show Message

Disconnect methods are private methods and call automatically by RunSQL, GetDataAdaperBySQL and GetDataReaderBySQL methods.

Table 11.13 lists the member methods of ShapeCollection class. ShapeCollection class is used to represents the array list for holding all Shape class objects. ShapeCollection is inherited from CollectionBase class. Each object that is on factory floor is added into ShapeCollection class by using Add method for using in simulation. Remove method is use for removing any Shape object from collection of shapes. Contains method is use for verify that a specific Shape object is in a collection. Simulation Engine uses this ShapeCollection to loop through each Shape object to receive and set property or can call specific method of Shape object.

Table 11.14 shows the member method of Log class. Log class represents the logging functionality for any unexpected event during execution of VF. The event log into windows application log and can be viewed by event viewer. Any system

Table 11.10 Summary of Virtual Factory Mediator class

Class summary			
Class ID: V10			Namespace: VirtualFactory.Factory
Class name: Mediator			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None

Field name	Data type	Access modifier	Description
dockPanel	DockPanel	private	This field holds the dockPanel
parent	MainForm	private	This field holds the parent Main form
cortexForm	GraphForm	private	This field holds the cortex Graph form
miniForm	MiniMapForm	private	This field holds the mini map form
controllerInfo	ControllerInfo	private	This field holds the controller info form
animatorForm	AnimationForm	private	This field holds the animation form
propsForm	PropertiesForm	private	This field holds the properties form
shapesForm	ShapeLibraryForm	private	This field holds the shapes form
outputForm	OutputForm	private	This field holds the output form
sb	StatusBar	private	This field holds the status bar
jobForm	JobForm	Private	This field holds the job form
processForm	ProcessForm	Private	This field holds the process form
machineoutputForm	MachineOutputForm	Private	This field holds the machine output form
factory3DForm	Factory3DForm	Private	This field holds the factory 3D form
jobScheduling	JobScheduling	Private	This field holds the job scheduling form
StartingConveyorCount	int	Private	This field holds the number of starting conveyors
EndingConveyorCount	int	Private	This field holds the number of ending conveyors
DAccess	DataHelper	Private	This field holds the data helper object

Method name	Return type	Access modifier	Description
StopRendering()	void	public	Function to stop rendering 3D animation
New()	void	Public	Function to create new simulation project

(continued)

Table 11.10 (continued)

Method name	Return type	Access modifier	Description
Open (string fileName)	void	public	Function to open project file on disk
RunFactory()	void	Public	Function to run factory simulation
LoadJobInformation()	bool	Private	Function to load job information
LoadCellInformation()	Bool	Private	Function to load cell information
Validate()	bool	Private	Function to validate all the processes
FillControllerByMachineName(string MachineName)	ArrayList	Private	Function to fill controller by machine name
LoadGridList()	bool	Private	Function to load grid list
CreateForms()	void	Public	Function to create all forms
CreateJobForm()	void	public	Function to create job form
CreateOutputForm()	Void	Private	Function to create output form
CreateMachineOutputForm()	Void	Private	Function to create machine output form
CreateCortexForm()	Void	Private	Function to create cortex form
CreateFactory3DForm	Void	Private	Function to create factory 3D form
CreateControllerInfo()	Void	Private	Function to create controller info
controllerInfo_PortStatusChangeEvent(string MachineName, int PinNumber, PinsStatus PinStatus)	Void		Function to handle port status change event
CreateMiniForm()	Void	Private	Function to create mini map form
CreateAnimatorForm()	Void	Private	Function to create animator form
CreatePropsForm()	Void	Private	Function to properties form
CreateShapesForm()	void	Private	Function to create shapes form
CreateProcessForm()	Void	Private	Function to create process form
OnShowProperties(object sender, VirtualFactory.VirtualFactoryLib.PropertyBag props)	Void	Private	Function to handle On Show Properties event
ShowProperties(object selectedObject)	Void	Public	Function to show properties on the form
MachineOutput(string message)	Void	Public	Function to show machine output on the form
Output(string message)	Void	Public	Function to show output to the form

(continued)

Table 11.10 (continued)

Method name	Return type	Access modifier	Description
ShowCortexProperties()	Void	Public	Function to show cortex properties
processForm_OnProcessShowMessage(string message)	void	Public	Function to handle On Process Show Message event
cortexForm_OnShow3DModel(object sender)	void	Private	Function to handle On Show 3D model event
Project_CurrentQueueStatus(int CellNumber, int Average, int NumberInQueue)	Void	Private	Function to handle project current queue status event
Project_CurrentJobStatus(string NameOfJob, int NumberOfJob, string TotalQueueTime, string TotalCompletionTime)	Void	Private	Function to handle current job status event
Project_OnJobComplete(Shape shape, string JobName, int JobNumber)	Void	Private	Function to handle On Job complete event
Project_OnAddJobInQueue(int TaskNumber, string JobName, int JobNumber, int WorkStation)	Void	Private	Function to handle On Add Job In Queue event
Project_OnRemoveJobInQueue(int TaskNumber, string JobName, int JobNumber, int WorkStation)	Void	Private	Function to handle on remove job in Queue event
Project_OnJobMove(Shape shape, string JobName, int JobNumber)	Void	Private	Function to handle on job move event
GetOutPutRobotLocation()	PositionVector	public	Function to get output robot location
GetFeedingRobotLocation()	PositionVector	Private	Function to get feeding robot location
Project_OnJobArrive(string NameOfJob, int NumberOfJob, string MaterialType, string MaterialSource)	Void	Private	Function to handle on job arrive event
Mediator(MainForm parent)	–	public	Constructor function for the class

Table 11.11 Summary of Virtual Factory Cell class

Class summary			
Class ID: V12		Namespace: VirtualFactory.VirtualFactoryLib.BasicShapes	
Class name: Cell		Class type: [Serializable]	
Is base class?: No		No. of inherited classes: 0	
Inherited from: Shape		Interfaces with: None	
Field name	Return type	Access modifier	Description
TopNode	Connector	private	This field holds the top node
BottomNode	Connector	private	This field holds the bottom node
LeftNode	Connector	private	This field holds the left node
RightNode	Connector	private	This field holds the right node

Method name	Return type	Access modifier	Description
AddProperties()	void	public override	Function to add properties to the cell
GetThumbnail()	Bitmap	public override	Function to get thumbnail of the selected cell
Paint(Graphics g)	void	protected override	Function to Paint the shape of the person object in the plex
ConnectionPoint (Connector c) e)	PointF	public override	Function to return the connection point
Cell() : base()	–	public	Default Constructor function for the class
Cell(IGraphSite site) : base(site)	–	public	Specialized Constructor function for the class

error or any internal message logged and notifies user by a message box that a new event has logged. Log message includes project name, method name, and error message with error code.

Table 11.15 presents member variables and method of Event class. Event class is used to represent the simulation events such as job arrive event or job depart event. Event class has nine properties to present event functionality. EventTime property holds the time when the event is created. EventType is used to identify the event type arrival or departure. Job property holds the job object for which the event is created. JobName and JobNumber properties are use by simulation engine to control the job flow. TaskNumber and TotalTaskNumber properties used to control the job flow between cells.

Table 11.16 explains the member variable and methods of MIS class. MIS class is used to represents the functionality of MIS department. All type of information related to MIS reports is added into database by using MIS class object during execution of simulation. MIS class does not have direction access to VF database, instead it uses DataHelper class object DAccess to add or update database records by using AddEntry and UpdateEntry methods, respectively.

Table 11.17 describes the member variable and methods of RobotProperty class. RobotProperty class represents array list for robot objects. RobotProperty class holds all robots that are on factory floor. AddRobot and RemoveRobot methods can be use for add robot into list and remove robot from list, respectively.

Table 11.12 Summary of Virtual Factory DataHelper class

Class summary			
Class ID: V13			Namespace: VirtualFactory.DataAccess
Class name: DataHelper			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: IDisposable			Interfaces with: None
Field name	Return type	Access modifier	Description
_dbName	string	private	This field holds the database name
_dbPath	string	private	This field holds the database path
component	Component	private	This field holds the component object
disposed	bool	private	This field holds the boolean value to dispose form

Method name	Return type	Access modifier	Description
Dispose()	void	public	Function to dispose the form
Dispose(bool disposing)	void	Private	Function to dispose the form If disposing equals true, dispose all managed and unmanaged resources
GetDataAdapterBySQL (string sSql)	OleDbDataAdapter	public	Function to get Data Adapter objet GetDataReaderBySQL(string sSql
OleDbDataAdapter	public		Function to get Data Reader object
RunSql(string sSql)	int	Public	Function to run SQL string
RunSql(string sSql, string sTableName)	DataSet	Public	Function to run SQL string for the provided table name
Connect()	OleDbConnection	public	Function to return the connection object Disconnect(OleDbConnection oCn)
void	public		Function to disconnect the connection
DataHelper()	–	private	Default Constructor function for the class
DataHelper (string dbName, string dbPath)	–	private	Specialized Constructor function for the class

Table 11.18 shows the member variables and methods of GenericMachineEditor class. GenericMachineEditor represents the windows form control for selection of machines model during the setup of machine properties. GenericMachineEditor provide the facility to select, e.g., milling machine models. GenericMachineEditor is inherited from windows form control.

Table 11.19 depicts the member variable and methods of ConveyorProperty class. ConveyorProperty class represents array list for conveyor belt objects. ConveyorProperty class holds all conveyor belts that are on factory floor. AddConveyor and RemoveConveyor methods can be used to add conveyor into list and remove conveyor from list, respectively.

Table 11.13 Summary of Virtual Factory ShapeCollection class

Class summary			
Class ID: V14			Namespace: VirtualFactory.VirtualFactoryLib
Class name: ShapeCollection			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: CollectionBase			Interfaces with: None
Method name	Return type	Access modifier	Description
Clone()	object	public	Function to clone the shape object
int Add(Shape shape)	int	public	Function to add a new shape object
Contains(object shape)	bool	public	Function to check if the shape exist
Sort(string sortParameter, SortDirection direction)	bool	public	Function to sort the added shapes in a specified direction
Remove(Shape shape)	bool	Public	Function to remove the shape
ShapeCollection()	–	private	Default Constructor function for the class

Table 11.14 Summary of Virtual Factory Log class

Class summary			
Class ID: V15			Namespace: virtualFactory.EventWriter
Class name: Log			Class type: [Serializable]
Is base class?: No			No. of inherited classes: 0
Inherited from: None			Interfaces with: None
Method name	Return type	Access modifier	Description
Write(string ProjectName, string MethodName,string Entry, EventLogEntryType EntryType, int EventID)	object	public static	Function to write the log with project details
Log()	–	private	Default Constructor function for the class

Table 11.20 details the member variables and methods of ScriptEditor class. ScriptEditor class is used to represents the windows form control for writing part program for CNC machines or robots. ScriptEditor is inherited from windows form control.

Table 11.21 portrays the member variables and methods of AnimationForm class. AnimationForm class is used to represents the windows form control to display Job Status List and Queue status List. AnimationForm class is inherited from DockContent class to help this control to dock with other windows. AddJob and AddQueue methods create rows in job status list and queue status list, respectively. AddJob and AddQueue are executed by Mediator class after verification of factory floor design and properties. ClearQueue method clears the queue status list. UpdateJob and UpdateQueue methods update the job status list and

Table 11.15 Summary of Virtual Factory Event class

Class summary			
Class ID: V18		Namespace: VirtualFactory.VirtualFactoryLib.SimulationEngine	
Class name: Event		Class type: Concrete	
Is base class?: No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: IDisposable	
Field name	Return type	Access modifier	Description
materialSource	String	Private	This field hold the material source name
materialType	String	Private	This field holds the material type
jobName	String	Private	This field holds the job name
eventType	EventType	Private	This field holds the event type
jobNumber	int	Private	This field holds the job number
job	int	Private	This field holds the job
taskNumber	int	private	This field holds the task number
totalTaskNumber	int	Private	This field holds the total task number
eventTime	DateTime	private	This field holds the event time
Method name	Return type	Access modifier	Description
Event()	–	public	Default Constructor function for the class

Table 11.16 Summary of Virtual Factory MIS class

Class summary			
Class ID: V20		Namespace: VirtualFactory.VirtualFactoryLib.MIS	
Class name: MIS		Class type: Concrete	
Is base class?: No		No. of inherited classes: 0	
Inherited from: None		Interfaces with: None	
Field name	Return type	Access modifier	Description
DAccess	DataHelper	private	This field holds the data helper class object
Method name	Return type	Access modifier	Description
AddEntry()	void	public	Function to add a new entry to the information system
UpdateEntry()	void	public	Function to update the added entry
MIS()	–	public	Default Constructor function for the class

queue status list. UpdateJob and UpdateQueue methods are executed by simulation engine to update the current status of job and queue.

Table 11.22 details the member variables and methods of Factory3DForm class. Factory3DForm class is used to represents 3D window for displaying factory in 3D format. Factory3DForm is inherited from DockContents class to help this window to dock with others windows. Factory3DForm class has StartRender and StopRender methods to start and stop 3D engine. Position method controls the position of Milling, Turning, Drilling and Sawing machines. Robot and Welding

Table 11.17 Summary of Virtual Factory RobotProperty class

Class summary			
Class ID: V22		Namespace: VirtualFactory.VirtualFactoryLib	
Class name: RobotProperty		Class type: concrete	
Is base class?: No		No. of inherited classes: 0	
Inherited from: StringConverter		Interfaces with: None	
Field name	Return type	Access modifier	Description
RobotNameArray	ArrayList	Private static	This field holds the array of robots

Method name	Return type	Access modifier	Description
GetStandardValuesSupported (ITypeDescriptorContext context)	bool	public override	Function to check if standard values supported
GetStandardValues (ITypeDescriptorContext context)	StandardValues Collection	public override	Function to return the standard values
AddRobot(string RobotName)	Void	public static	Function to add a new robot
RemoveRobot(string RobotName)	void	public static	Function to remove the robot
Clear()	void	public static	Function to clear the screen
RobotProperty()	–	public	Default Constructor function for the class

machines have their own methods to control position of parts such as RobotPosition Method and WeldingPosition Method. CreateWorkPiece creates 3D work piece at run time as number of job defined in simulation. AddInQueue and RemoveQueue methods manage the job queue.

Table 11.23 shows the member variables and methods of OutPutForm class. OutPutForm class is used to represents the message output window. OutPutForm class is inherited from DockContents class to help this control to dock with others windows. ClearOut method clears output window. OutPut method displays any informative message such as any improper configuration of factory setup notified in OutPut form window.

Table 11.24 illustrates the summary of virtual factory processes sequence.

Table 11.25 presents the summary of virtual factory Run Simulation use case.

Table 11.26 explains the summary of virtual factory Add Project use case.

Table 11.27 depicts the summary of virtual factory Update Project use case.

Table 11.28 produces the summary of virtual factory Add Job use case.

Table 11.29 displays the summary of virtual factory Update Job use case.

Table 11.30 defines the summary of virtual factory Open Project use case.

Table 11.31 outlines the summary of virtual factory Save Project use case.

Table 11.32 shows the summary of virtual factory processes flow chart.

The UML use case diagrams for Run VF project, Create VF project, Open VF project, and Save VF project are shown in Figs. 11.60, 11.61, 11.62, and 11.63, respectively. UML sequence diagram for the Run VF Project, VF job arrive event handling and VF Simulation event handling are given as Figs. 11.64, 11.65, and

Table 11.18 Summary of Virtual Factory GenericMachineEditor class

Class summary			
Class ID: V23			Namespace: VirtualFactory.VirtualFactoryLib.UI
Class name: GenericMachineEditor			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: System.Windows.Forms.Form			Interfaces with: None
Field name	Return type	Access modifier	Description
OKButton	System.Windows.Forms.Button	private	This field holds the button object
mCancelButton	System.Windows.Forms.Button	Private	This field holds the button object
groupBox1	System.Windows.Forms.GroupBox	Private	This field holds the group box object
label1	System.Windows.Forms.Label	Private	This field holds the label object
Label2	System.Windows.Forms.Label	Private	This field holds the label object
dataGrid1	System.Windows.Forms.DataGrid	Private	This field holds the data grid object
DAccess	DataHelper	Private	This field holds the data helper object
timer1	System.Windows.Forms.Timer	private	This field holds the timer object
cmbMachines	System.Windows.Forms.ComboBox	Private	This field holds the machine combo list
components	System.ComponentModel.IContainer	private	This field holds the components container object

Method name	Return type	Access modifier	Description
Dispose(bool disposing)	void	protected override	Function to dispose the class object
InitializeComponent()	Void	private	Function to initialize the components. Required method for Designer support
OKButton_Click(object sender, System.EventArgs	Void	Private	Function to handle the OK button click event
timer1_Tick(object sender, System.EventArgs e)	Void	Private	Function to handle the timer tick event
LoadMachines()	Void	Private	Function to load all the machines
GenericMachineEditor_Load(object sender, System.EventArgs e)	Void	Private	Function to handle the form load event
cmbMachines_SelectedIndexChanged (object sender, System.EventArgs e)	Void	private	Function to handle the machine list
LoadGrid()	void	private	Function to load the data grid after user selected the machine
GenericMachineEditor()	–	public	Default Constructor function for the class

Table 11.19 Summary of Virtual Factory ConveyorProperty class

Class summary			
Class ID: V24			Namespace: VirtualFactory.VirtualFactoryLib
Class name: ConveyorProperty			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: StringConverter			Interfaces with: None
Field name	Return type	Access modifier	Description
ConveyorNameArray	ArrayList	private static	This field holds the conveyor array
Method name	Return type	Access modifier	Description
GetStandardValues Supported(ITypeDescriptor Context context)	bool	public override	Function to check if standard values supported
GetStandardValues (ITypeDescriptorContext context)	StandardValues Collection	public override	Function to return the standard values
AddConveyor(string ConveyorName)	void	public static	Function to add a new conveyor
RemoveConveyor(string ConveyorName)	Void	public static	Function to remove the conveyor
Clear()	void	public static	Function to clear the screen
ConveyorProperty()	–	public	Default Constructor function for the class

11.66, respectively. Figures 11.67, 11.68, 11.69, 11.70, and 11.71 show various UML classes static structures.

Figure 11.72 provides UML static structure of Factory3DLibrary classes. The description of each class is listed below.

(a) *Robot3D class description*
Robot3D class represents the 3D robot in VF. Robot3D has two properties and six methods. Robot3D load the DirectX files and create robot for 3D rendering. Each Robot3D object identify by MachineName property. Movement of robot parts such as lower arm and upper arm can be controlled by using Robot3D's Move method. Movement of robot its self can be change by calling Move-Robot method. Current position can be achieved by calling GetRobotPosition method. ModelsPath property can be use for setting the drive location where DirectX files of robot are located. Factory3D delete or remove 3D robot from 3D space by calling Destroy method.

(b) *MillingWorkPiece3D class description*
MillingWorkPiece3D class represents the 3D work piece in VF. Milling-WorkPiece3D has seven properties and five methods. MillingWorkPiece3D loads the DirectX files and create work piece for 3D rendering. Each work piece object identified by WorkPieceNumber property. Position of the work piece in 3D space can be achieved by calling GetWorkPiecePosition.

Table 11.20 Summary of Virtual Factory ScriptEditor class

Class summary			
Class ID: V25			Namespace: VirtualFactory.Machines
Class name: ScriptEditor			Class type: [NonSerialized]
Is base class?: No			No. of inherited classes: 0
Inherited from: System.Windows.Forms.Form			Interfaces with: None
Field name	Return type	Access modifier	Description
components	System.ComponentModel.Container	private	This field holds the components container object
mainMenu1	System.Windows.Forms.MainMenu	Private	This field holds the main menu object
menuItem1	System.Windows.Forms.MenuItem	Private	This field holds the menu item object
mnuOpen	System.Windows.Forms.MenuItem	Private	This field holds the open menu item
mnuNew	System.Windows.Forms.MenuItem	Private	This field holds the new menu item
mnuSave	System.Windows.Forms.MenuItem	Private	This field holds the save menu item
menuItem5	System.Windows.Forms.MenuItem	Private	This field holds the menu item
menuItem6	System.Windows.Forms.MenuItem	private	This field holds the Exit menu item
mnuSaveAs	System.Windows.Forms.MenuItem	Private	This field holds the Save As menu item
results	CompilerResults	Private	This field holds the compiler results after compiling the part programs
panel1	System.Windows.Forms.Panel	private	This field holds the panel object
btnOk	System.Windows.Forms.Button	Internal	This field holds the ok button
lvwErrors	System.Windows.Forms.ListView	Internal	This field holds the listview for displaying errors
ColumnHeader1	System.Windows.Forms.ColumnHeader	Internal	This field holds the column header object
ColumnHeader2	System.Windows.Forms.ColumnHeader	Internal	This field holds the column header object
btnCancel	System.Windows.Forms.Button	internal	This field holds the cancel button
splitter1	System.Windows.Forms.Splitter	private	This field holds the splitter object
txtScript	System.Windows.Forms.TextBox	Private	This field holds the text box object
menuItem2	System.Windows.Forms.MenuItem	Private	This field holds the menu item
menuItem2	System.Windows.Forms.MenuItem	Private	This field holds the menu item

(continued)

Table 11.20 (continued)

Field name	Return type	Access modifier	Description
mnuBiomorph	System.Windows.Forms.MenuItem	private	This field holds the bio morph menu item
_compiledScript	IScript	private	This field returns the compiled script

Method name	Return type	Access modifier	Description
Dispose(bool disposing)	void	protected override	Function to dispose the class object
InitializeComponent()	Void	private	Function to initialize the components. Required method for Designer support
btnOk_Click(object sender, System.EventArgs e)	Void	Private	Function to handle the OK button click event
lvwErrors_ItemActivate(object sender, System.EventArgs e)	Void	Private	Function to handle the errors list view item activate event
mnuNew_Click(object sender, System.EventArgs e)	Void	Private	Function to handle the New button click event
SaveAs(string FileName)	Void	Private	Function to Save the file with specified name
mnuSaveAs_Click(object sender, System.EventArgs e)	Void	private	Function to handle the Save As button click event
menuItem6_Click(object sender, System.EventArgs e)	void	private	Function to handle the Item button click event
mnuSave_Click(object sender, System.EventArgs e)	void	private	Function to handle the Save button click event
btnCancel_Click(object sender, System.EventArgs e)	Void	private	Function to handle the Cancel button click event
mnuBiomorph_Click(object sender, System.EventArgs e)	void	private	Function to handle the Bio morph button click event
ScriptEditor()	–	public	Default Constructor function for the class

Table 11.21 Summary of Virtual Factory AnimationForm class

Class summary	
Class ID: V28	Namespace: VirtualFactory.Factory
Class name: AnimationForm	Class type: Concrete
Is base class?: No	No. of inherited classes: 0
Inherited from: DockContent	Interfaces with: None

Field name	Return type	Access modifier	Description
mediator	Mediator	private	This field holds the mediator object for connecting other classes
lstJob	ListViewEx	Private	This field holds the list view for Job
lstQueue	ListViewEx	Private	This field holds the list view for Queue
splitter1	System.Windows.Forms.Splitter	Private	This field holds the Splitter
JobNumber	System.Windows.Forms.ColumnHeader	Private	This field holds the column header for job number
Percent	System.Windows.Forms.ColumnHeader	Private	This field holds the column header for percent
Total	System.Windows.Forms.ColumnHeader	Private	This field holds the column header for total
Cell	System.Windows.Forms.ColumnHeader	private	This field holds the column header for cell
Queue	System.Windows.Forms.ColumnHeader	Private	This field holds the column header for queue
Average	System.Windows.Forms.ColumnHeader	private	This field holds the column header for average
columnHeader1	System.Windows.Forms.ColumnHeader	private	This field holds the column header for total time in queue
columnHeader2	System.Windows.Forms.ColumnHeader	Private	This field holds the column header for total job completion time
components	System.ComponentModel.Container	private	This field holds the components container

Method name	Return type	Access modifier	Description
UpdateJob(string JobName, int JobNumber, string TotalQueueTime, string TotalCompletionTime)	Void	public	Function to update job when the event of update job occurs
UpdateQueue(int CellNumber, int Average, int NumberInQueue)	void	public	Function to update queue when event of update queue occurs

(continued)

Table 11.21 (continued)

Method name	Return type	Access modifier	Description
RemoveJobs()	void	public	Function to remove job
AddJob(string [] JobValues)	Void	Public	Function to add a new job
ClearQueue()	void	public	Function to clear queue
AddQueue(string [] QueueValues)	Void	public	Function to add queue
InitializeComponent()	Void	private	Function to initialize form components. Required method for Designer support
Dispose(bool disposing)	void	protected override	Function to Clean up any resources being used
AnimationForm_Load(object sender, System.EventArgs e	Void	private	Function to handle the load event of the form
AnimationForm_Closing(object sender, System.ComponentModel. CancelEventArgs e)	void	private	Function to handle the close event of the form
AnimationForm(Mediator mediator)	–	public	Default Constructor function for the class

Table 11.22 Summary of Virtual Factory Factory3DForm class

Class summary			
Class ID: V29			Namespace: VirtualFactory.Factory
Class name: Factory3DForm			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: DockContent			Interfaces with: None
Field name	Return type	Access modifier	Description
mediator	Mediator	private	This field holds the mediator object for connecting other classes
picBox	System.Windows.Forms.PictureBox	Private	This field holds the picture box object for rendering 3D object
factory3D	Factory3D	Private	This field holds the factory3D object for accessing all 3D functionality
contextMenu	System.Windows.Forms.ContextMenu	Private	This field holds the menu object
menuItem1	System.Windows.Forms.MenuItem	Private	This field holds the menu item 1 for Load Factory
menuItem2	System.Windows.Forms.MenuItem	Private	This field holds the menu item 2 for Clear factory
menuItem3	System.Windows.Forms.MenuItem	Private	This field holds the menu item 3 for Refresh Rendering
components	System.ComponentModel.Container	private	This field holds the components container
Machine3DCollection	ArrayList	Private	This field holds the array of 3D Machines. Machine3DCollection holds all 3D machines for accessing during execution
WorkPiece3DCollection	ArrayList	private	This field holds the array of 3D WorkPiece
WorkPiece3DMillingCollection	ArrayList	private	This field holds the array of 3D work piece for milling
MoveWorkThread	Thread	Private	This field holds the thread for moving work piece
MoveRobotThread	Thread	private	This field holds the thread for moving robot
GantryThread	Thread	Private	This field holds the thread for gantry

(continued)

Table 11.22 (continued)

Field name	Return type	Access modifier	Description
gFeedRobot	string	Private	This field holds the feeding robot
gPVector	PositionVector	private	This field holds the position of the machine
QueueAddArray	ArrayList	Private	This field holds the array list for queue
WorkDoneArray	ArrayList	Private	This field holds the array list for work completed
MoveArray	ArrayList	Private	This field holds the array list for moving work piece object
ChangeThreadArray	ArrayList	private	This field holds the array list for changing process threads
factoryLoaded	bool	Private	This field holds the boolean value to check if factory is loaded

Method name	Return type	Access modifier	Description
factory3D_OnSetFocus(object sender)	void	private	Function to handle on set focus event
InitializeComponent()	Void	private	Function to initialize form components. Required method for Designer support
Dispose(bool disposing)	void	protected override	Function to clean up any resources being used
StartRender()	void	public	Function to start 3D rendering
StopRender()	void	public	Function to stop 3D rendering
Factory3DForm_Load(object sender, System.EventArgs e)	Void	private	Function to handle the form load event
CreateWorkPiece(string Name, int JobNumber, string MaterialType, int WorkPieceNumber, string MaterialSource, PositionVector Position)	void	public	Function to create work piece for a job with specified material at a specified position
SetWorkPieceQueueStatus(string JobName, int JobNumber, bool Status)	Void	private	Function to set work piece queue status
DestroyWorkPiece(string JobName,int JobNumber)	Void	private	Function to destroy work piece

(continued)

Table 11.22 (continued)

Method name	Return type	Access modifier	Description
MoveWorkPiece(string JobName,int JobNumber, PositionVector PVector, bool QueueStatus)	Void	private	Function to move work piece to another location with a queue
GetWorkPiece(string JobName,int JobNumber)	WorkPiece3D	Private	Function to return the work piece object
GetWorkPieceMaterialType(string JobName,int JobNumber)	string	Private	Function to return the work piece material type
MoveWorkPiece(string JobName,int JobNumber, PositionVector PVector)	Void	private	Function to move work piece to another location
MoveWorkPiece(string JobName,int JobNumber, PositionVector PVector,Shape shape)	Void	Private	Function to move work piece to another location in the specified shape
GetRobotLocationByName(string RobotName)	PositionVector	Private	Function to return the position for robot
GetRobot(string RobotName)	Robot3D	Private	Function to return the robot object
GetGantry()	Gantry3D	private	Function to return the gantry object
GetWorkPositionOfSawing()	PositionVector	private	Function to return the position for Sawing machine
GetConveyor(string ConveyorName	ConveyorBase3D	Private	Function to return the conveyor object
RunQueueAdd()	Void	private	Function to run the added queue
AddInQueue(int TaskNumber, string JobName, int JobNumber, int WorkStation)	Void	public	Function to add the task in the queue
RemoveQueue(int TaskNumber, string JobName, int JobNumber, int WorkStation)	Void	Public	Function to remove the task from the queue
WorkComplete(Shape shape, string JobName, int JobNumber,bool Complete)	Void	Public	Function to mark the status of work complete
ChangeMeshThreadRun()	Void	Private	Function to change the mesh
MoveThread()	Void	Private	Function to start the move thread
MoveGantry()	Void	Private	Function to start the gantry
RobotThreadMove()	Void	Private	Function to start the robot move thread
RobotThread()	Void	Private	Function to start the robot

(continued)

Table 11.22 (continued)

Method name	Return type	Access modifier	Description
MoveWorkPiece(Shape shape, string JobName, int JobNumber)	Void	Public	Function to move work piece in specified shape with no location
ClearWorkPiece()	Void	Public	Function to clear work piece
ClearFactory()	Void	Public	Function to clear whole factory
LoadFactory()	Void	Public	Function to load factory
menuItem1_Click(object sender, System.EventArgs e)	Void	private	Function to handle menu item click event
menuItem2_Click(object sender, System.EventArgs e)	Void	private	Function to handle menu item click event
menuItem3_Click(object sender, System.EventArgs e)	Void	private	Function to handle menu item click event
RobotPosition(string RobotName, string message)	Void	Public	Function to set the robot position
WeldingPosition(string MachineName, string message)	Void	Public	Function to set the welding machine position
MoveMillingWorkPiece(int JobNumber, string JobName, string MachineName, MachineAxis Coordinate, float Value,object o)	Void	private	Function to move the milling work piece
Position(Shape shape,MachineAxis coordinate, double dvalue)	Void	public	Function to set the position of the selected shape
Factory3DForm(Mediator mediator)	–	public	Default Constructor function for the class

Table 11.23 Summary of Virtual Factory OutputForm class

Class summary			
Class ID: V30			Namespace: VirtualFactory.Factory
Class name: OutputForm			Class type: Concrete
Is base class?: No			No. of inherited classes: 0
Inherited from: DockContent			Interfaces with: None
Field name	Return type	Access modifier	Description
Outputter	System.Windows.Forms.TextBox	private	This field holds the text box for displaying message
mediator	Mediator	Private	This field holds the mediator object for connecting other objects
components	System.ComponentModel.Container	Private	This field holds the components container

Method name	Return type	Access modifier	Description
Dispose(bool disposing)	Void	protected override	Function to Clean up any resources being used
InitializeComponent()	Void	private	Function to initialize form components. Required method for Designer support
Output(string message)	Void	public	Function to display the output message
ClearOutput()	void	public	Function to clear the output form
OutputForm(Mediator mediator)	–	public	Default Constructor function for the class

MoveWorkPiece method is used to move work piece at specified location. The current place of work piece can be obtained by using property InsideMachine. ModelsPath property can be used for setting the drive location where DirectX files of work piece are located. Factory3D deletes or removes 3D work piece from 3D space by calling Destroy method.

(c) *Milling3D class description*
Milling3D class represents the 3D Milling machine in VF. Milling3D class has two properties and four methods. Milling3D loads the DirectX files and create milling machine for 3D rendering. Each milling machine is identified by MachineName property. Move method is used to control the movement of 3D work piece inside milling machine. GetWorkPieceStartPosition method is used by Robot to decide where it should drop/pick work piece inside machine. ModelsPath property can be use for setting the drive location where DirectX files of milling machine are located. Factory3D deletes or removes 3D milling machine from 3D space by calling Destroy method.

(d) *Factory3D class description*
Factory3D class represent 3D world for VF and is responsible for creating 3D factory including walls, floor, sealing and all 3D object of VF. Factory3D class

Table 11.24 Summary of virtual factory processes sequence

Sequence diagram summary				
Sequence ID: SD-V1 Sequence name: Virtual Factory Sequence Diagram				
Method name	Type	Description	Source	Destination
Run Factory	Return Message	Function to start factory animation	Main Form	Mediator
Validate()	Self Message	Function to validate the factory layout, job information and cells information	Mediator	Mediator
Load Grid List()	Self Message	Function to load the factory design on the grid	Mediator	Mediator
Load Job Information()	Self Message	Function to load the job information to the factory collection	Mediator	Mediator
Load Cell Information()	Self Message	Function to load the cell information to the factory layout	Mediator	Mediator
Start Rendering()	Message	Function to start the 3D scene rendering	Mediator	Factory3D
Load 3D Factory()	Message	Function to load all the objects in the factory design	Mediator	Factory3D
Get Project()	Return Message	Function to return the selected Project from the collection	Mediator, Process Form	Process Form
On Add job in Queue	Self Message	Event to add a new job to the queue	Mediator	Mediator
On job Arrive	Self Message	Event to set a new job status	Mediator	Mediator
On job Move	Self Message	Event to start the job	Mediator	Mediator
On job Complete	Self Message	Event to complete the job	Mediator	Mediator
Run Project	Message	Function to run the project	Mediator	Process Form
Run Project	Message	Function to run the project	Process Form	Project
Run Machine	Message	Function to run the specific machine	Project	Machine
Validate File	Self Message	Function to validate the program file for running the specific machine	Machine	Machine

(continued)

Table 11.24 (continued)

Method name	Type	Description	Source	Destination
Start Execution	Self Message	Function to start execution of the machine	Machine	Machine
Get Start Time	Self Message	Function to return start time when machine executed	Machine	Machine
Update 3D	Message	Function to continuously update and animate 3D scene	Mediator	Factory 3D
Start 3D Machine	Self Message	Function to start 3D rendering of machine	Factory 3D	Factory 3D
Move 3D Work Piece	Message	Function to move 3D work piece	Mediator, Factory 3D	Factory 3D
Move 3D Robot	Self Message	Function to move 3D robot	Factory 3D	Factory 3D
Job Complete	Message	Function to fire the event of Job Complete	Main Form	Project
Job Complete	Message	Function to fire the event of Job Complete	Mediator	Factory 3D
Start Stack Jobs	Self Message	Function to start jobs in the queue	Factory 3D	Factory 3D
Update Database	Self Message	Function to update tools database	Project	Project
Calculate Jobs	Self Message	Function to calculate the total jobs	Project	Project
Run Machine	Message	Function to run the selected machine	Project	Machine
Create Work Piece	Message	Function to create the work piece object	Mediator	Factory 3D
Add work piece in collection	Self Message	Function to add work piece in collection for the required job	Factory 3D	Factory 3D
Get work start position	Message	Function to get the starting position of work piece object	Factory 3D	Milling 3D
Get starting conveyor	Return Message	Function to get the starting position of conveyor	Factory 3D	Milling 3D
Get feeding conveyor	Return Message	Function to get the source/feeding conveyor	Factory 3D	Milling 3D
Get feeding robot	Return Message	Function to get the source/feeding robot	Factory 3D	Milling 3D
Get taking robot	Return Message	Function to get the destination robot	Factory 3D	Milling 3D
Conveyor()	Return Message	Function to load the conveyor machine	Milling 3D	Conveyor 3D
Robot()	Return Message	Function to load the robot	Milling 3D	Robot 3D

(continued)

Table 11.24 (continued)

Method name	Type	Description	Source	Destination
Move(Location)	Message	Function to move the work piece to a specified location	Factory 3D	Work Piece 3D
Update()	Return Message	Function to update the work piece status to machine	Work Piece 3D	Milling 3D
Rotate(Interpolation, Location)	Message	Function to rotate the robot arm in a specified direction	Factory 3D	Robot 3D
Rotate()	Self Message	Function to rotate the robot arm in predefined direction	Robot 3D	Robot 3D
Create Job tag	Self Message	Function to create a job tag	Project	Project
Move Machine 3D	Message	Function to move the spindle of the machine	Mediator	Milling 3D
3D Move	Self Message	Function to move the 3D spindle meshes of the machines	Milling 3D	Milling 3D
Project Run	Message	Function to load and run the project information	Mediator	Project
Clear Event Queue	Self Message	Function to clear all events from the queue	Project	Project
Clear job Queue	Self Message	Function to clear all jobs from the queue	Project	Project
Add job route	Self Message	Function to add a new job route	Project	Project
SimulationThread.Start()	Self Message	Function to start the 3D simulation process	Project	Project
Start()	Message	Function to start the timer for the simulation process	Project	VF Timer
TimerTick()	Self Message	Event to return the timer tick	VF Timer	VF Timer
Event()	Return Message	Function to start the 3D simulation event	VF Timer	Simulation Event
EventQueue.Add (SimulationEvent)	Self Message	Function to add the simulation event to the queue	VF Timer	VF Timer
Get Simulation Event	Self Message	Function to get the specified event from the queue	Project	Project
Arrive()	Self Message	Event to check the simulation	Project	Project

(continued)

Table 11.24 (continued)

Method name	Type	Description	Source	Destination
Get Machines by Work station Number	Return Message	Function to get machines by work station number	Project	Factory
Run()	Return Message	Function to run the factory	Project	Factory
Departure()	Self Message	Event to finish the assigned job	Project	Project
Remove Job In Queue	Self Message	Function to remove the job from the queue	Project	Project
ExitLoop()	Self Message	Function to exit the 3D rendering loop	Project	Project
EventQueue.Add (DepartureEvent)	Self Message	Function to add the departure event to the queue	Project	Project
CreateMachine (MachineType)	Return Message	Function to create a machine of selected type	Main Form	Factory
Machine(MachineType)	Return Message	Function to return the specific machine type	Factory	Machine
Machine.Add(Machine)	Self Message	Function to add a machine object	Machine	Machine

Table 11.25 Summary of virtual factory Run Simulation use case

Use case summary
Use case ID: UC-V1
Use case name: Run Simulation
Actors: User
Description: The user presses the Run Simulation button and the factory simulation starts after validation and loading cell, job and machine information
Preconditions Factory layout is designed Job shop/cells are defined Jobs are created Machines are loaded with connection properties Robots are connected with the machines
Post conditions: 3D simulation is shown with the current status information in the status bar below
System parameters: Factory Project
Success scenario: Complete 3D simulation is displayed successfully
Alternative scenario: Error in rendering/loading the project or in executing the job is displayed
Includes: Run Factory
Priority: High

Table 11.26 Summary of virtual factory Add Project use case

Use case summary
Use case ID: UC-V2
Use case name: Add Project
Actors: User
Description: The user creates a new project, enter project information and save project
Preconditions: None
Post conditions: Project node is added to the tree
System parameters: Project information
Success scenario: Project is successfully created
Alternative scenario: Error in creating project is displayed
Includes: Create Project
Priority: High

Table 11.27 Summary of virtual factory Update Project use case

Use case summary
Use case ID: UC-V3
Use case name: Update Project
Actors: User
Description: The user updates an existing project, enter project information and save project
Preconditions: None
Post conditions: Project information is updated
System parameters: Project information
Success scenario: Project is successfully updated
Alternative scenario: Error in updating project is displayed
Includes: Get project node from tree
Priority: High

Table 11.28 Summary of virtual factory Add Job use case

Use case summary
Use case ID: UC-V4
Use case name: Add job
Actors: User
Description: The user creates a new job, enter job information and save job
Preconditions: None
Post conditions: Job node is added to the tree
System parameters: Job information
Success scenario: Job is successfully created
Alternative scenario: Error in creating job is displayed
Includes: Create job
Priority: High

Table 11.29 Summary of virtual factory Update Job use case

Use case summary
Use case ID: UC-V5
Use case name: Update job
Actors: User
Description: The user updates an existing job, enter job information and save job
Preconditions: None
Post conditions: Job information is updated
System parameters: Job information
Success scenario: Job is successfully updated
Alternative scenario: Error in updating job is displayed
Includes: Get job node from tree
Priority: High

Table 11.30 Summary of virtual factory Open Project use case

Use case summary
Use case ID: UC-V6
Use case name: Open project
Actors: User
Description: The user selects the project file by browsing on the system
Preconditions: Project file should exist on the system
Post conditions: Project is successfully loaded
System parameters: Project information
Success scenario
The user selects the project file and opens it
The system creates file stream and desterilizes the file
The cells information is loaded
The machines information is loaded
The project information is loaded
The jobs information is loaded
Alternative scenario: Error in opening the project file is displayed
Includes: Open Dialog box
Priority: High

Table 11.31 Summary of virtual factory Save Project use case

Use case summary
Use case ID: UC-V7
Use case name: Save project
Actors: User
Description: The user selects the project file by browsing on the system
Preconditions: Project information is added and validated successfully
Post conditions: Project is successfully saved
System parameters: Project information
Success scenario
The user selects to save the project file
The system creates file stream and desterilizes the file
The cells information is saved
The machines information is saved
The project information is saved
The jobs information is saved
Alternative scenario: Error in saving the project file is displayed
Includes: Save Dialog box
Priority: High

is also responsible for creating 3D engine that control the rendering of all 3D object. All other functions called by Mediator class create 3D object. For example Mediator class calls CreateMilling method to create milling machine in 3D space. DestroyMachine is use by Mediator class delete to or remove all machines in 3D space.

(e) *Drilling3D class description*
Drilling3D class represents the 3D Drilling machine in VF. Drilling3D class has two properties and four methods. Drilling3D loads the DirectX files and create drilling machine for 3D rendering. Each drilling machine is identified by MachineName property. Move method is used to control the movement of 3D work piece inside drilling machine. GetWorkPieceStartPosition method is use by Robot to decide where it should drop/pick work piece inside machine. ModelsPath property can be use for setting the drive location where DirectX files of drilling machine are located. Factory3D deletes or removes 3D drilling machine from 3D space by calling Destroy method.

(f) *ConveyorBase3D class description*
ConveyorBase3D class represents the base class of all type of 3D conveyor belts in VF. ConveyorBase3D class has four properties and five methods which all of which are common to all type of 3D conveyor belts. Each conveyor belt is identified by MachineName property. The number of work piece currently placed on conveyor belt is counted by NumberOfWorkPiece property. ModelsPath property can be used for setting the drive location where DirectX files of conveyor belt are located.

(g) *Conveyor3D class description*
Conveyor3D class represents the straight conveyor belt. Conveyor3D class is inherited from ConveyorBase3D class. Conveyor3D class has all the base

Table 11.32 Summary of virtual factory processes flow chart

Flow chart summary
Flow chart ID: FC-V1
Flow chart name: Virtual Factory Process Flow Chart
No. of inputs: 17
No. of processes: 317
No. of outputs: 3
No. of control decisions: 141

Input name	Description
Save Project	Save Project Button on screen
Run Simulation	Run Simulation Button on screen
Create Process Plan	Create Process Plan Button on screen
Save Requisition	Save Requisition Button on screen
Save Department	Save Department Button on screen
Save Supplier	Save Supplier Button on screen
Save Category	Save Category Button on screen
Save Product	Save Product Button on screen
Approve Requisition	Approve Requisition Button on screen
Purchasing	Purchasing Button on screen
Add User	Add User Button on screen
User Permissions	User Permissions Button on screen
General Entries	General Entries Button on screen
Create Ledger	Create Ledger Button on screen
Quality Control Purchase	Quality Control Purchase Button on screen
Quality Control Sales	Quality Control Sales Button on screen
Exit Application	Exit Application Button on screen

Process name	Description
Run Factory	Function to Run Factory
Update User about Deny of Feature	Function to Update User about Deny of Feature
Exit Run	Function to Exit Run
Mediator Run Factory	Function to call Mediator object of Run Factory
Validate Factory Setup	Function to Validate Factory Setup
Load Grid List	Function to Load Grid List
Load Job Information	Function to Load Job Information
Load Cell Information	Function to Load Cell Information
Show 3D Window	Function to Show 3D Window
Start 3D Rendering	Function to Start 3D Rendering
Load 3D Factory	Function to Load 3D Factory
Save Project	Function to Save Project
Set Project Event Handler For OnJobArrive	Function to Set Project Event Handler For OnJobArrive
Set Project Event Handler for OnJobMove	Function to Set Project Event Handler for OnJobMove
Set Project Event Handler for OnJobComplete	Function to Set Project Event Handler for OnJobComplete

(continued)

Table 11.32 (continued)

Process name	Description
Set Project Event Handler for OnAddJobInQueue	Function to Set Project Event Handler for OnAddJobInQueue
Set Project Event Handler for OnRemoveJobFromQueue	Function to Set Project Event Handler for OnRemoveJobFromQueue
Clear 3D Work Piece Collection	Function to Clear 3D Work Piece Collection
Clear Queue Event	Function to Clear Queue Event
Clear Job Queue	Function to Clear Job Queue
Set Job Number = 1	Function to Set Job Number = 1
Check Each Job	Function to Check Each Job
Add Job Information in MIS Table	Function to Add Job Information in MIS Table
Create New Event Timer For Each Job	Function to Create New Event Timer For Each Job
Start Job Timer	Function to Start Job Timer
Set Task Number = 1	Function to Set Task Number = 1
Check Each Cell	Function to Check Each Cell
Set Job Route With Job Number and Task Number	Function to Set Job Route With Job Number and Task Number
Increase Task Number By 1	Function to Increase Task Number By 1
Increase Job Number By 1	Function to Increase Job Number By 1
Create New Thread For Simulation Loop	Function to Create New Thread For Simulation Loop
Start Simulation Loop Thread	Function to Start Simulation Loop Thread
Get Simulation Event From Event Queue	Function to Get Simulation Event From Event Queue
Simulation Event Queue	Function to call Simulation Event Queue
Set event Type = None	Function to Set event Type = None
Continue Loop	Function to Continue Loop
Exit Loop	Function to Exit Loop
Set Simulation Event Task Number = 1	Function to Set Simulation Event Task Number = 1
Get Cell From Route Matrix	Function to Get Cell From Route Matrix
Get Machines By Cell Number	Function to Get Machines By Cell Number
Check Each Machine	Function to Check Each Machine
Add Job In Queue	Function to Add Job In Queue
Add Cell in Queue	Function to Add Cell in Queue
Make Machine Free True	Function to Make Machine Free True
Create New Job Tag With Job Parameters	Function to Create New Job Tag With Job Parameters
Attach Job Tag With Machine	Function to Attach Job Tag With Machine
Raise OnJobMove Event	Function to Raise OnJobMove Event
Raise OnAddJobInQueue Event	Function to Raise OnAddJobInQueue Event
Raise CurrentQueueStatus Event	Function to Raise CurrentQueueStatus Event
Start Machine	Function to Start Machine
Add Job in Queue Collection	Function to Add Job in Queue Collection
Update Queue Status Window	Function to Update Queue Status Window

(continued)

Table 11.32 (continued)

Process name	Description
Update User No Part Program Found	Function to Update User No Part Program Found
Validate Part Program	Function to Validate Part Program
Update User Invalid Part Program	Function to Update User Invalid Part Program
Create New Run Thread	Function to Create New Run Thread
Start Run Thread	Function to Start Run Thread
Clear Code List	Function to Clear Code List
Create Stream Reader	Function to Create Stream Reader
Read File	Function to Read File
Extract Line	Function to Extract Line
Read Each Line	Function to Read Each Line
Split Line by Space	Function to Split Line by Space
Validate Word	Function to Validate Word
Validate = False	Function to Validate = False
Validate = True	Function to Validate = True
Add Work in Code List	Function to Add Work in Code List
Get Machine Start Time	Function to Get Machine Start Time
Set Status = Working	Function to Set Status = Working
Check Each Code in Code List	Function to Check Each Code in Code List
Port Access for Each Code	Function to call Port Access for Each Code
Raise Current Status	Function to Raise Current Status
Update User Interface	Function to Update User Interface
Get Simulation Event Task Number	Function to Get Simulation Event Task Number
Get Simulation Event Job Number	Function to Get Simulation Event Job Number
Get Cell Using Task number and Job number	Function to Get Cell Using Task number and Job number
Check Each Job in Job Queue	Function to Check Each Job in Job Queue
Get Machines By Cell Number	Function to Get Machines By Cell Number
Create New Job Tag	Function to Create New Job Tag
Calculate Queuing Time	Function to Calculate Queuing Time
Run Machine	Function to Run Machine
Set Delete From Queue = True	Function to Set Delete From Queue = True
Remove Job From Job Queue	Function to Remove Job From Job Queue
Remove job From Cell Queue	Function to Remove job From Cell Queue
Raise Current Queue Status	Function to Raise Current Queue Status
Get Task Number By Current Job Number	Function to Get Task Number By Current Job Number
Raise Arrive Event	Function to Raise Arrive Event
Factory 3D Move Work Piece	Function to Factory 3D Move Work Piece
Get Machine name Job name Job number	Function to Get Machine name Job name Job number
Check each machine in 3D Machines Collection	Function to Check each machine in 3D Machines Collection

(continued)

Table 11.32 (continued)

Process name	Description
Get Machine 3D Feeding Conveyor	Function to Get Machine 3D Feeding Conveyor
Get Machine Feeding Conveyor	Function to Get Machine Feeding Conveyor
Get Conveyor Start Position	Function to Get Conveyor Start Position
Check each Work Piece in 3D Work Piece Collection	Function to Check each Work Piece in 3D Work Piece Collection
Move Work Piece at Conveyor Start Position	Function to Move Work Piece at Conveyor Start Position
Get Conveyor Feeding Robot	Function to Get Conveyor Feeding Robot
Get Robot Home Position	Function to Get Robot Home Position
Rotate Robot to Take Work Piece	Function to Rotate Robot to Take Work Piece
Pick Work Piece	Function to Pick Work Piece
Take Conveyor Start Position	Function to Take Conveyor Start Position
Move Robot to Conveyor Start Position	Function to Move Robot to Conveyor Start Position
Drop Work Piece on Conveyor	Function to Drop Work Piece on Conveyor
Get Feeding Conveyor End Position	Function to Get Feeding Conveyor End Position
Get Machine Feeding Robot	Function to Get Machine Feeding Robot
Get Machine Home Position	Function to Get Machine Home Position
Move Robot To feeding Conveyor End Position	Function to Move Robot To feeding Conveyor End Position
Take Work Piece from Feeding Conveyor	Function to Take Work Piece from Feeding Conveyor
Get Machine Location	Function to Get Machine Location
Move Robot To Machine Location	Function to Move Robot To Machine Location
Drop Work Piece on Machine	Function to Drop Work Piece on Machine
Move Robot to Home Position	Function to Move Robot to Home Position
Get Machine 3D Feeding Conveyor	Function to Get Machine 3D Feeding Conveyor
Get Machine Feeding Conveyor	Function to Get Machine Feeding Conveyor
Get Conveyor Start Position	Function to Get Conveyor Start Position
Get Information for this Job	Function to Get Information for this Job
Create New Job Arrive Event	Function to Create New Job Arrive Event
Raise on Job Arrive Event	Function to Raise on Job Arrive Event
Add Job Arrive Event in Event Queue	Function to Add Job Arrive Event in Event Queue
Get Job Information	Function to Get Job Information
3D Create Work Piece	Function to 3D Create Work Piece
Get Initial Position of Work Piece From Sawing Machine	Function to Get Initial Position of Work Piece From Sawing Machine
Create Work Piece	Function to Create Work Piece
Add Work Piece in 3D Work Piece Collection	Function to Add Work Piece in 3D Work Piece Collection
Connect Database	Function to Connect Database
Update Project Information	Function to Update Project Information

(continued)

Table 11.32 (continued)

Process name	Description
Get Existing Project ID	Function to Get Existing Project ID
Set Project Already Run True	Function to Set Project Already Run True
Insert New Project Into Database	Function to Insert New Project Into Database
Get New Project ID	Function to Get New Project ID
Set Project Already Run False	Function to Set Project Already Run False
Save Job Information	Function to Save Job Information
Get All Job From Project	Function to Get All Job From Project
Get Cells Information From Job	Function to Get Cells Information From Job
Get Machines From a Cell	Function to Get Machines From a Cell
Get Machine Type	Function to Get Machine Type
Get Machine Costing Information	Function to Get Machine Costing Information
Save Job Information into Database	Function to Save Job Information into Database
Load All Cells	Function to Load All Cells
Exit Grid List	Function to Exit Grid List
Clear Entries in Machines status From	Function to Clear Entries in Machines status From
Check Each Cell	Function to Check Each Cell
Get Machines on a Cell	Function to Get Machines on a Cell
Get Name of Machine	Function to Get Name of Machine
Add entry in Machines Status Window	Function to Add entry in Machines Status Window
Load Project Information	Function to Load Project Information
Update User No Project Found	Function to Update User No Project Found
Exit Load Job Information	Function to Exit Load Job Information
Load Job Information	Function to Load Job Information
Clear Jobs Status	Function to Clear Jobs Status
Clear Cell Queue Status	Function to Clear Cell Queue Status
Update User No Job Found	Function to Update User No Job Found
Check Each Job	Function to Check Each Job
Get Name of Job	Function to Get Name of Job
Insert New entry in Job Status Window	Function to Insert New entry in Job Status Window
Load All Cell	Function to Load All Cell
Update User No Cell found	Function to Update User No Cell found
Exit Load Cell Information	Function to Exit Load Cell Information
Clear Queue Status	Function to Clear Queue Status
Check Each Cell	Function to Check Each Cell
Get Number of Cell	Function to Get Number of Cell
Insert New Entry in Queue Status Window	Function to Insert New Entry in Queue Status Window
Set Speed	Function to Set Speed
Raise Status Event	Function to Raise Status Event
Set Current Tool	Function to Set Current Tool
Set Feed	Function to Set Feed

(continued)

Table 11.32 (continued)

Process name	Description
XValue	Function to get the XValue
PulseRequestX()	Function to get the PulseRequestX()
Xminus()	Function to get the Xminus()
Xplus()	Function to get the Xplus()
Raise Position Event	Function to Raise Position Event
Raise X Position Event	Function to Raise X Position Event
YValue	Function to get the YValue
PulseRequestY()	Function to set the PulseRequestY()
Yminus()	Function to set the Yminus()
Yplus()	Function to set the Yplus()
Raise Current Position Event	Function to Raise Current Position Event
Raise Current Status	Function to Raise Current Status
ZValue()	Function to get the ZValue()
PulseRequestZ()	Function to set the PulseRequestZ()
Zminus()	Function to set the Zminus()
Zplus()	Function to set the Zplus()
Raise Current Status Event	Function to Raise Current Status Event
NextPositionX()	Function to set NextPositionX()
NextPositionY()	Function to set NextPositionY()
Calculate Linear Interpolation()	Function to Calculate Linear Interpolation()
AddLine()	Function to call AddLine()
Raise Current X Position Event	Function to Raise Current X Position Event
Raise Current Y Position Event	Function to Raise Current Y Position Event
NextValueZ()	Function to set NextValueZ()
I()	Function to set I()
J()	Function to set J()
CalculateParabolicInterpolation()	Function to call CalculateParabolicInterpolation()
Raise Position X− Event	Function to Raise Position X− Event
Raise Position X+ Event	Function to Raise Position X+ Event
CalculateCirculateInterpolation()	Function to call CalculateCirculateInterpolation()
Create Process Plan	Function to Create Process Plan
Open Project Window for Parameters Modification	Function to Open Project Window for Parameters Modification
Open Project Window	Function to Open Project Window
Get Project Parameters	Function to Get Project Parameters
Add New Project in Project Tree	Function to Add New Project in Project Tree
Update Tree	Function to Update Tree
Get All Job Related to Project	Function to Get All Job Related to Project
Open job Window	Function to Open job Window
Get Job Parameters	Function to Get Job Parameters
Add New Job in Project Tree	Function to Add New Job in Project Tree
Open Window For Parameters Modifications	Function to Open Window For Parameters Modifications
Get All Cells from Factory Layout	Function to Get All Cells from Factory Layout

(continued)

Table 11.32 (continued)

Process name	Description
Get All Robots from Factory Layout	Function to Get All Robots from Factory Layout
Get All Conveyors from Factory Layout	Function to Get All Conveyors from Factory Layout
Update User about Limits of Robots	Function to Update User about Limits of Robots
Exit Validate	Function to Exit Validate
Update User about Limits of Conveyors	Function to Update User about Limits of Conveyors
Validate Each Conveyor	Function to Validate Each Conveyor
Update User A Conveyor cant be Job Starting and Job Ending Conveyor at the same time	Function to Update User A Conveyor cant be Job Starting and Job Ending Conveyor at the same time
Increase Job Ending Conveyor Counter	Function to Increase Job Ending Conveyor Counter
Update User Output Robot is required for Job Ending Conveyor	Function to Update User Output Robot is required for Job Ending Conveyor
Update User Cant find Output Robot	Function to Update User Cant find Output Robot
Increase job Starting Conveyor Counter	Function to Increase job Starting Conveyor Counter
Update User Feeding Robot is required for Job Starting Conveyor	Function to Update User Feeding Robot is required for Job Starting Conveyor
Update User Cant find Feeding Robot	Function to Update User Cant find Feeding Robot
Sawing Found True	Function to Sawing Found True
Update User Job Starting Conveyor is required	Function to Update User Job Starting Conveyor is required
Update User Job Ending Conveyor is required	Function to Update User Job Ending Conveyor is required
Update User Job Ending Conveyor Cant be more than 1	Function to Update User Job Ending Conveyor Cant be more than 1
Update User Job Starting Conveyor Cant be more than 1	Function to Update User Job Starting Conveyor Cant be more than 1
Update User At least 1 Cell is required	Function to Update User At least 1 Cell is required
Update User Cells cant be more than 2	Function to Update User Cells cant be more than 2
Update User Invalid Cell Number	Function to Update User Invalid Cell Number
Update User At least 1 Machine is required on a Cell	Function to Update User At least 1 Machine is required on a Cell
Update User Cell Cant hold more than 2 Machines	Function to Update User Cell Cant hold more than 2 Machines
Update User Machine Name cant be Null	Function to Update User Machine Name cant be Null

(continued)

Table 11.32 (continued)

Process name	Description
Update User Machine Part Program is not valid	Function to Update User Machine Part Program is not valid
Update User Cannot find Machine Feeding Conveyor	Function to Update User Cannot find Machine Feeding Conveyor
Update User Cannot find Machine Output Conveyor	Function to Update User Cannot find Machine Output Conveyor
Update User Cannot find Machine feeding Robot	Function to Update User Cannot find Machine feeding Robot
Update User Cannot find Machine Output Robot	Function to Update User Cannot find Machine Output Robot
Inform User Deny of Service	Function to Inform User Deny of Service
Inform User Name is Required	Function to Inform User Name is Required
Inform User Department is Required	Function to Inform User Department is Required
Inform User Item is Required	Function to Inform User Item is Required
Inform User Quantity is Required	Function to Inform User Quantity is Required
Make Requisition Urgent	Function to Make Requisition Urgent
Exit Save Requisition	Function to Exit Save Requisition
Insert Record in Database	Function to Insert Record in Database
Update User	Function to Update User
Update Requisition List	Function to Update Requisition List
Inform User Location	Function to Inform User Location
Inform User Department Head Required	Function to Inform User Department Head Required
Inform User Description is Required	Function to Inform User Description is Required
Exit Save Department	Function to Exit Save Department
Update Departments List	Function to Update Departments List
Inform User Phone Number is Required	Function to Inform User Phone Number is Required
Inform User Fax Number is Required	Function to Inform User Fax Number is Required
Inform User Supplier Address is Required	Function to Inform User Supplier Address is Required
Inform User Contact Person is Required	Function to Inform User Contact Person is Required
Inform User Delivery Time is Required	Function to Inform User Delivery Time is Required
Exit Save Supplier	Function to Exit Save Supplier
Update Supplier List	Function to Update Supplier List
Inform User Select Product Classification	Function to Inform User Select Product Classification
Inform User Category Name is Required	Function to Inform User Category Name is Required
Inform User Description is Required	Function to Inform User Description is Required

(continued)

Table 11.32 (continued)

Process name	Description
Exit Save Category	Function to Exit Save Category
Update Category List	Function to Update Category List
Inform User Select Product Category	Function to Inform User Select Product Category
Inform User Product Name is Required	Function to Inform User Product Name is Required
Inform User Product Cost is Required	Function to Inform User Product Cost is Required
Inform User Product Type is Required	Function to Inform User Product Type is Required
Inform User Description is Required	Function to Inform User Description is Required
Exit Save Product	Function to Exit Save Product
Update Product List	Function to Update Product List
Exit Purchasing Items	Function to Exit Purchasing Items
Inform User Select Requisition to be Approve	Function to Inform User Select Requisition to be Approve
Mark Requisition Not Approved	Function to Mark Requisition Not Approved
Mark Requisition Approved	Function to Mark Requisition Approved
Mark Requisition Un Delivered	Function to Mark Requisition Un Delivered
Mark Requisition Out of Stock	Function to Mark Requisition Out of Stock
Inform User Select Supplier	Function to Inform User Select Supplier
Inform User Select Product to Purchase	Function to Inform User Select Product to Purchase
Load Products Related to Selected Supplier	Function to Load Products Related to Selected Supplier
Inform User Quantity is Required	Function to Inform User Quantity is Required
Exit Purchasing Items	Function to Exit Purchasing Items
Make Product QC Pass = No	Function to Make Product QC Pass = No
Update Purchasing List	Function to Update Purchasing List
Inform User Designation is Required	Function to Inform User Designation is Required
Inform User Password is Required	Function to Inform User Password is Required
Exit Create New User	Function to Exit Create New User
Set All Permissions No	Function to Set All Permissions No
Update User List	Function to Update User List
Inform User Select User to Set Permissions	Function to Inform User Select User to Set Permissions
Set Manage Inventory Yes	Function to Set Manage Inventory Yes
Exit User Permissions	Function to Exit User Permissions
Set MIS Reports Yes	Function to Set MIS Reports Yes
Set Planning Reports Yes	Function to Set Planning Reports Yes
Set Costing Reports Yes	Function to Set Costing Reports Yes
Set Design & Execute Yes	Function to Set Design & Execute Yes
Set Manage Users	Function to Set Manage Users
Inform User Selection of Ledger	Function to Inform User Selection of Ledger

(continued)

Table 11.32 (continued)

Process name	Description
Inform User Entry Description is Required	Function to Inform User Entry Description is Function to Required
Inform User Debit or Credit Amount is Required	Function to Inform User Debit or Credit Amount is Required
Inform User Entry can be Debit or Credit	Function to Inform User Entry can be Debit or Credit
Exit Create General Entries	Function to Exit Create General Entries
Mark General Entry As Credit	Function to Mark General Entry As Credit
Mark General Entry As Debit	Function to Mark General Entry As Debit
Update Entries List	Function to Update Entries List
Inform User Account Type is Required	Function to Inform User Account Type is Required
Inform User Ledger Name is Required	Function to Inform User Ledger Name is Required
Inform User Description is Required	Function to Inform User Description is Required
Update Ledger List	Function to Update Ledger List
Exit Create New Ledger	Function to Exit Create New Ledger
Exit Quality Control Sales	Function to Exit Quality Control Sales
Inform User Select Product for Quality Control	Function to Inform User Select Product for Quality Control
Mark Product QC Pass Fail	Function to Mark Product QC Pass Fail
Mark Product QC Pass OK	Function to Mark Product QC Pass OK
Add Product in Finish Goods Store	Function to Add Product in Finish Goods Store
Update Store List	Function to Update Store List
Update Product List	Function to Update Product List
Exit Quality Control Purchase	Function to Exit Quality Control Purchase
Output Name	Description
3D Factory Simulation	The Virtual Factory is displayed in 3D with complete job cycle as defined by the user
Inventory and Accounts Management Reports	The user can manage and view inventory control and finance reports
Multiple View Windows	The user can view multiple windows for building factory project for, e.g., properties window, Mini map, tools window, machine window, etc.
Control decision name	**Description**
Operator has Design and Execution Permission	Check if Operator has Design and Execution Permission
Factory Loaded in 3D	Check if Factory Loaded in 3D
Event is Arrive	Check if Event is Arrive
Event is Departure	Check if Event is Departure
Event is None	Check if Event is None
Event is End of Simulation	Check if Event is End of Simulation
Job Number = 1	Check if Job Number = 1
Machine Status = Free	Check if Machine Status = Free

(continued)

Table 11.32 (continued)

Control decision name	Description
Machine is not Sawing	Check if Machine is not Sawing
Part Program Path = Null	Check if Part Program Path = Null
Validate = True	Check if Validate = True
Word = Null	Check if Word = Null
Word in GM Code List	Check if Word in GM Code List
S	Check if S
F	Check if F
T	Check if T
Rapid Movement	Check if Rapid Movement
Linear Interpolation	Check if Linear Interpolation
Parabolic Interpolation	Check if Parabolic Interpolation
Circular Interpolation	Check if Circular Interpolation
Delete From Queue = True	Check if Delete From Queue = True
Current Task Number < Task Number	Check if Current Task Number < Task Number
Machine is Milling	Check if Machine is Milling
Machine Name = Milling Name	Check if Machine Name = Milling Name
Conveyor = Job Starting Conveyor	Check if Conveyor = Job Starting Conveyor
Work Piece Number = Job Number	Check if Work Piece Number = Job Number
Machine is Turning	Check if Machine is Turning
Machine Name = Turning Name	Check if Machine Name = Turning Name
Machine is Drilling	Check if Machine is Drilling
Machine Name = Drilling Name	Check if Machine Name = Drilling Name
Machine is Welding	Check if Machine is Welding
Machine Name = Welding Name	Check if Machine Name = Welding Name
Job Material Source is Sawing	Check if Job Material Source is Sawing
Material Source is Sawing	Check if Material Source is Sawing
Project Exist in Database	Check if Project Exist in Database
Cell Count = 0	Check if Cell Count = 0
Project = Null	Check if Project = Null
Job Counts = 0	Check if Job Counts = 0
IGMCode Start With X	Check if IGMCode Start With X
IGMCode Start With Y	Check if IGMCode Start With Y
IGMCode Start With Z	Check if IGMCode Start With Z
Absolute Coordinate	Check if Absolute Coordinate
CurrentPositionX > XValue	Check if CurrentPositionX > XValue
CurrentPositionY > YValue	Check if CurrentPositionY > YValue
CurrentPositionZ > ZValue	Check if CurrentPositionZ > ZValue
Check Y Coordinate	Check if Check Y Coordinate
NextPositionX ! = CurrentPositionX	Check if NextPositionX ! = CurrentPositionX
NextPositionX > CurrentPositionX	Check if NextPositionX > CurrentPositionX
NextPositionY > CurrentPositionY	Check if NextPositionY > CurrentPositionY
Project Created	Check if Project Created
Change Job OR Create Job	Check if Change Job OR Create Job

(continued)

Table 11.32 (continued)

Control decision name	Description
Robot Counts > 6	Check if Robot Counts > 6
Conveyor Counts > 4	Check if Conveyor Counts > 4
Job ending Conveyor is True and Job Starting Conveyor is True	Check if Job ending Conveyor is True and Job Starting Conveyor is True
Job Ending is True	Check if Job Ending is True
Conveyor Output Robot Set	Check if Conveyor Output Robot Set
Output Robot name is valid	Check if Output Robot name is valid
Job Starting is True	Check if Job Starting is True
Conveyor Feeding Robot Set	Check if Conveyor Feeding Robot Set
Feeding Robot name is valid	Check if Feeding Robot name is valid
Starting Conveyor Count = 0	Check if Starting Conveyor Count = 0
Machine is Sawing	Check if Machine is Sawing
Sawing Found True	Check if Sawing Found True
Job Ending Conveyor = 0	Check if Job Ending Conveyor = 0
Job Ending Conveyor > 1	Check if Job Ending Conveyor > 1
Job Starting Conveyor > 1	Check if Job Starting Conveyor > 1
Cell Count > 2	Check if Cell Count > 2
Cell Number = 0	Check if Cell Number = 0
Machine Count = 0	Check if Machine Count = 0
Machine Count > 2	Check if Machine Count > 2
Machine Name = null	Check if Machine Name = null
Part Program is Valid	Check if Part Program is Valid
Check Machine Feeding Conveyor	Check Machine Feeding Conveyor
Check Machine Output Conveyor	Check Machine Output Conveyor
Check Machine Feeding Robot	Check Machine Feeding Robot
Check Machine Output Robot	Check Machine Output Robot
Operator has Save Requisition Permission	Check if Operator has Save Requisition Permission
Requisition Initiator = Null	Check if Requisition Initiator = Null
Initiator Department = Null	Check if Initiator Department = Null
Requisition Item = Null	Check if Requisition Item = Null
Required Quantity = 0	Check if Required Quantity = 0
Urgent Box = Checked	Check if Urgent Box = Checked
Operator has Permission to Create Department	Check if Operator has Permission to Create Department
Department Name = Null	Check if Department Name = Null
Location = Null	Check if Location = Null
Department Head = Null	Check if Department Head = Null
Description = Null	Check if Description = Null
Operator has Permission to Add New Supplier	Check if Operator has Permission to Add New Supplier
Supplier Name = Null	Check if Supplier Name = Null
Phone = Null	Check if Phone = Null
Fax = Null	Check if Fax = Null
Address = Null	Check if Address = Null

(continued)

Table 11.32 (continued)

Control decision name	Description
Contact Person = Null	Check if Contact Person = Null
Description = Null	Check if Description = Null
Delivery Time = Null	Check if Delivery Time = Null
Operator has Permission to Create New Category	Check if Operator has Permission to Create New Category
Selection of Product Classification	Check if Selection of Product Classification
Category Name = Null	Check if Category Name = Null
Category Description = Null	Check if Category Description = Null
Operator has Permission to Add New Product	Check if Operator has Permission to Add New Product
Selection of Product Category	Check if Selection of Product Category
Product Name = Null	Check if Product Name = Null
Product Cost = Null	Check if Product Cost = Null
Product Type = Null	Check if Product Type = Null
Product Description = Null	Check if Product Description = Null
Operator has Permission to Approve Requisition	Check if Operator has Permission to Approve Requisition
Selection of Requisition	Check if Selection of Requisition
Select Approve	Check if Select Approve
Select Delivered	Check if Select Delivered
Select Out of Stock	Check if Select Out of Stock
Operator has Permission to Purchase New Items	Check if Operator has Permission to Purchase New Items
Selection of Supplier	Check if Selection of Supplier
Selection of Product	Check if Selection of Product
Purchasing Quantity = 0	Check if Purchasing Quantity = 0
Operator has Permission to Create New User	Check if Operator has Permission to Create New User
User Name = Null	Check if User Name = Null
Select Designation = Null	Check if Select Designation = Null
Select Department = Null	Check if Select Department = Null
User Password = Null	Check if User Password = Null
Operator has Permission to Set User Permissions	Check if Operator has Permission to Set User Permissions
Select User = Null	Check if Select User = Null
Manage Inventory Checked	Check if Manage Inventory Checked
View MIS Reports Checked	Check if View MIS Reports Checked
View Planning Reports Checked	Check if View Planning Reports Checked
View Costing Report Checked	Check if View Costing Report Checked
Design & Execute Checked	Check if Design & Execute Checked
Manage User Checked	Check if Manage User Checked
Operator has Permission to Create New General Entries	Check if Operator has Permission to Create New General Entries
Ledger = Null	Check if Ledger = Null
Entry Description = Null	Check if Entry Description = Null

(continued)

Table 11.32 (continued)

Control decision name	Description
Debit AND Credit = 0	Check if Debit AND Credit = 0
Debit AND Credit ! = 0	Check if Debit AND Credit ! = 0
Credit ! = 0	Check if Credit ! = 0
Operator has Permission to Create New Ledger	Check if Operator has Permission to Create New Ledger
Select Account Type = Null	Check if Select Account Type = Null
Ledger Name = Null	Check if Ledger Name = Null
Ledger Description = Null	Check if Ledger Description = Null
Operator has Permission to Control the Quality of Sales	Check if Operator has Permission to Control the Quality of Sales
Selection of Product	Check if Products are selected
Pass Quality Control	Check if Pass Quality Control
Operator has Permission to Control the Quality of Purchase	Check if Operator has Permission to Control the Quality of Purchase

class properties and two override methods. GetWorkPieceStartPosition method is used by robot to identify the location or position in X, Y and Z coordinates to drop work piece on conveyor belts. GetWorkPieceEndPosition is the same as GetWorkPieceStartPosition, but it identifies the position from where robot can take the work piece from conveyor belt. Destroy method is implemented in each conveyor class to delete or remove conveyor belt from 3D space.

(h) *LeftRoundConveyor3D class description*
LeftRoundConveyor3D class represents the round conveyor belts curving on left side. LeftRoundConveyor3D class inherited from ConveyorBase3D class. LeftRoundConveyor3D class has all the base class properties and three override methods. GetWorkPieceStartPosition method is used by robot to identify the location or position in X, Y and Z coordinates to drop work piece on conveyor belts. GetWorkPieceEndPosition is the same as GetWorkPieceStartPosition, but it identifies the position from where robot can take the work piece from conveyor belt. GetWorkPieceCenterPosition is used by 3D animation engine to identify the position from where work piece should take turn. Destroy method is implemented in each conveyor class to delete or remove conveyor belt from 3D space.

(i) *RightRoundConveyor3D class description*
RightRoundConveyor3D class represents the round conveyor belts curving on right side. RightRoundConveyor3D class inherited from ConveyorBase3D class. RightRoundConveyor3D class has all the base class properties and three override methods. GetWorkPieceStartPosition method is used by robot to identify the location or position in X, Y and Z coordinates to drop work piece on conveyor belts. GetWorkPieceEndPosition is the same as

Fig. 11.60 Run VF Project UML Use Case

GetWorkPieceStartPosition, but it identifies the position from where robot can take the work piece from conveyor belt. GetWorkPieceCenterPosition is used by 3D animation engine to identify the position from where work piece should take turn. Destroy method is implemented in each conveyor class to delete or remove conveyor belt from 3D space.

Figure 11.73 provides UML static structure of Factory3DLibrary classes. The description of each class is listed below.

(a) *UpLeftRoundConveyor3D class description*
UpLeftRoundConveyor3D class represents the round conveyor belts curving from upper left side. UpLeftRoundConveyor3D class inherited from ConveyorBase3D class. UpLeftRoundConveyor3D class has all the base class

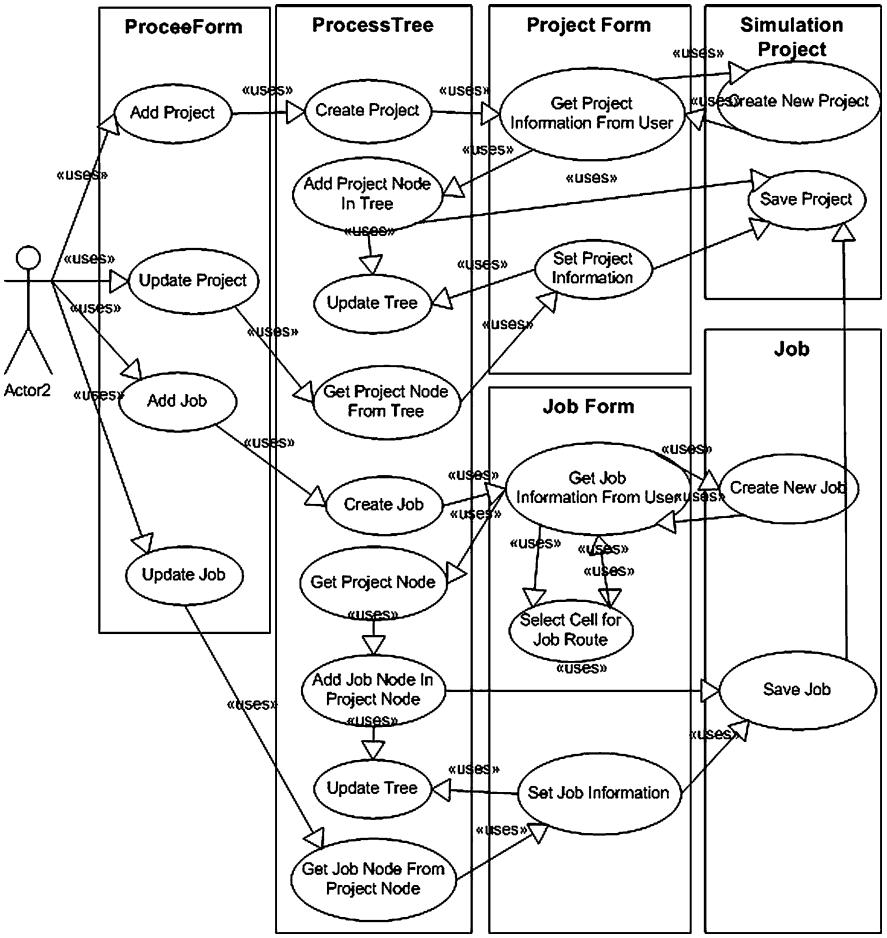


Fig. 11.61 Create VF Project UML Use Case

properties and three override methods. GetWorkPieceStartPosition method is used by robot to identify the location or position in X, Y and Z coordinates to drop work piece on conveyor belts. GetWorkPieceEndPosition is the same as GetWorkPieceStartPosition, but it identifies the position from where robot can take the work piece from conveyor belt. GetWorkPieceCenterPosition is used by 3D animation engine to identify the position from where work piece should take turn alongside conveyor. Destroy method is implemented in each conveyor class to delete or remove conveyor belt from 3D space.

(b) *Sawing3D class description*

Sawing3D class represents the 3D Sawing machine in VF. Sawing3D class has three properties and six methods. Sawing3D loads the DirectX files and create

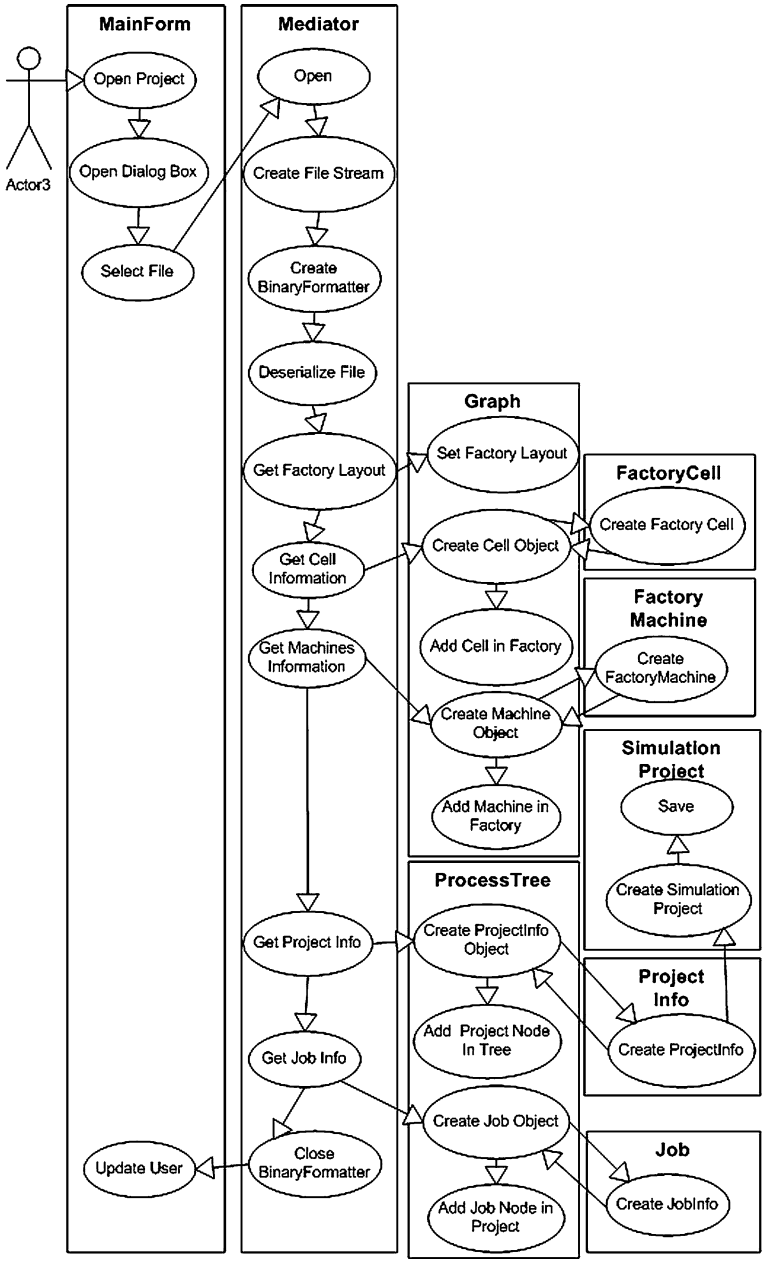


Fig. 11.62 Open VF Project UML Use Case

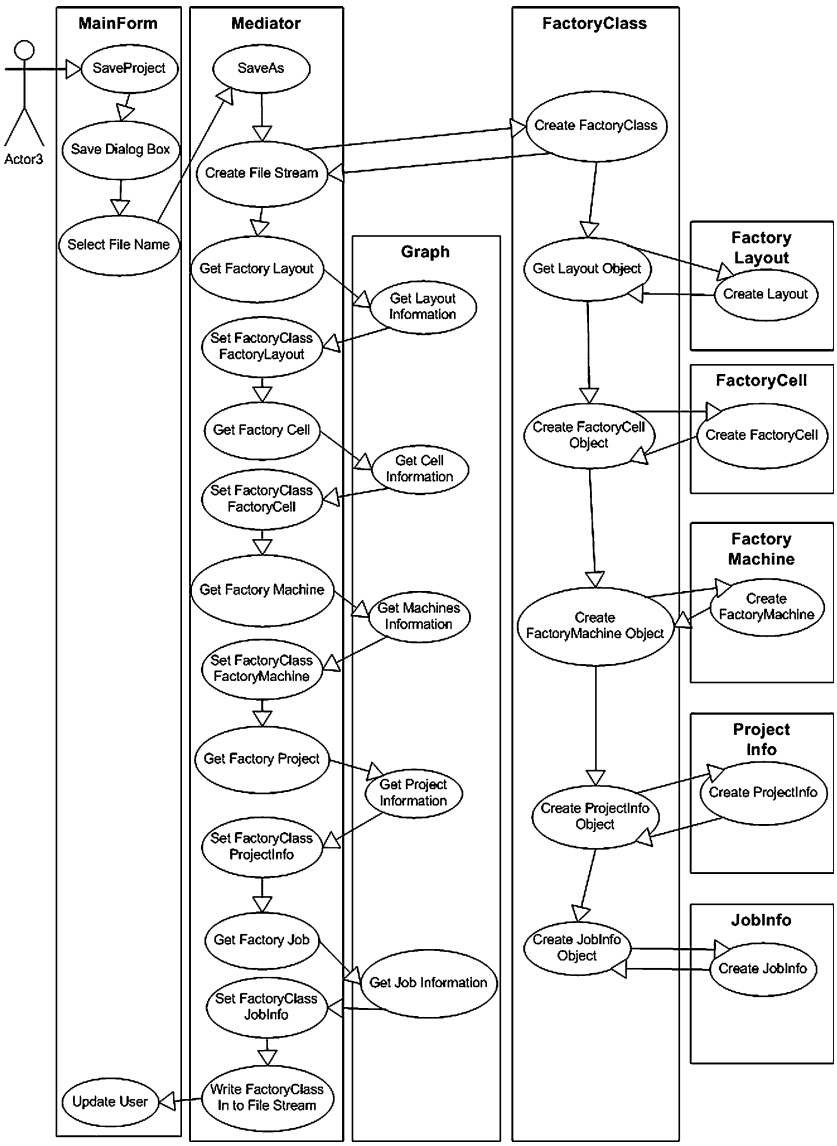


Fig. 11.63 Save VF Project UML Use Case

sawing machine for 3D rendering. Each sawing machine is identified by MachineName property. Move method is used to control the movement of 3D work piece inside sawing machine. GetWorkPieceStartPosition method is use by Robot to decide where it should drop/pick work piece inside machine. ModelsPath property can be use for setting the drive location where DirectX files of sawing machine are located. SpindleControl is used to control the

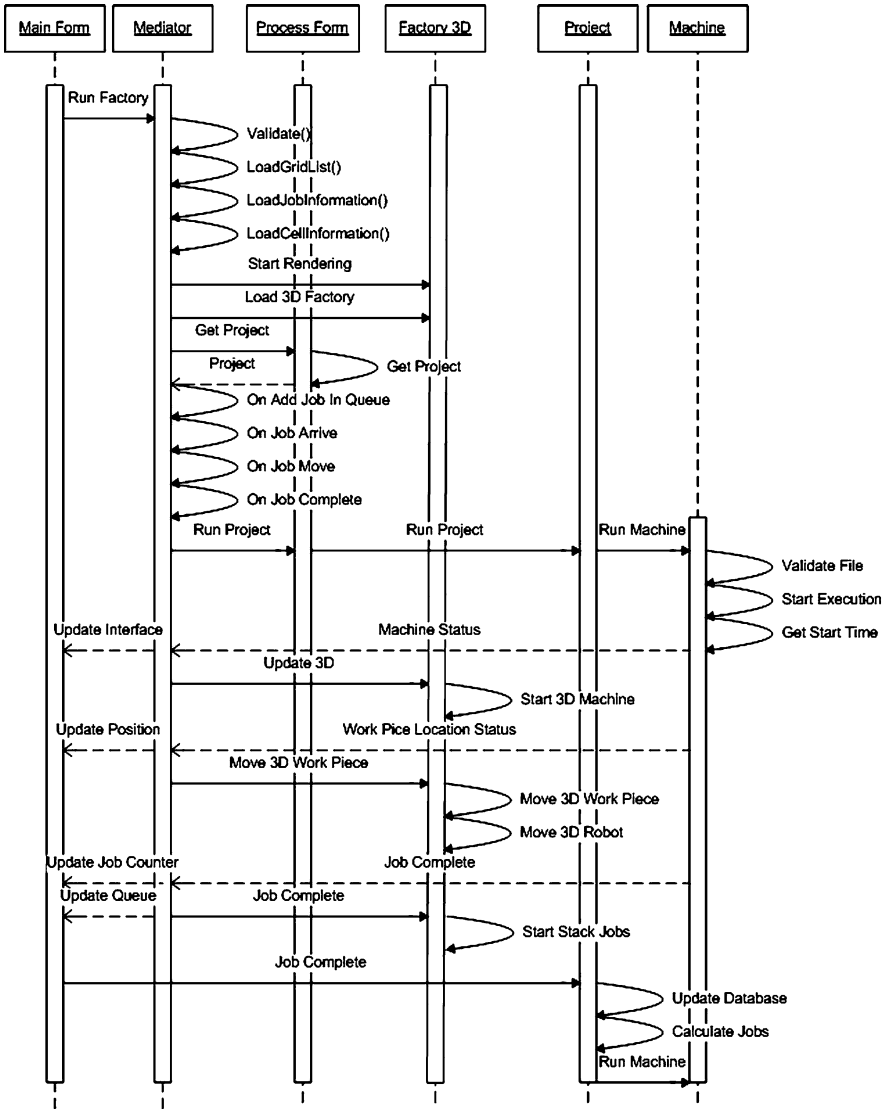


Fig. 11.64 Run VF Project (Overview) UML Sequence

sawing blade speed. Factory3D delete or remove 3D sawing machine from 3D space by calling Destroy method.

(c) *WorkPiece3D class description*

WorkPiece3D class represents the 3D work piece in VF. WorkPiece3D class has seven properties and six methods to control work piece functionality and movement. InQueue and InsideMachine properties can be use by simulation

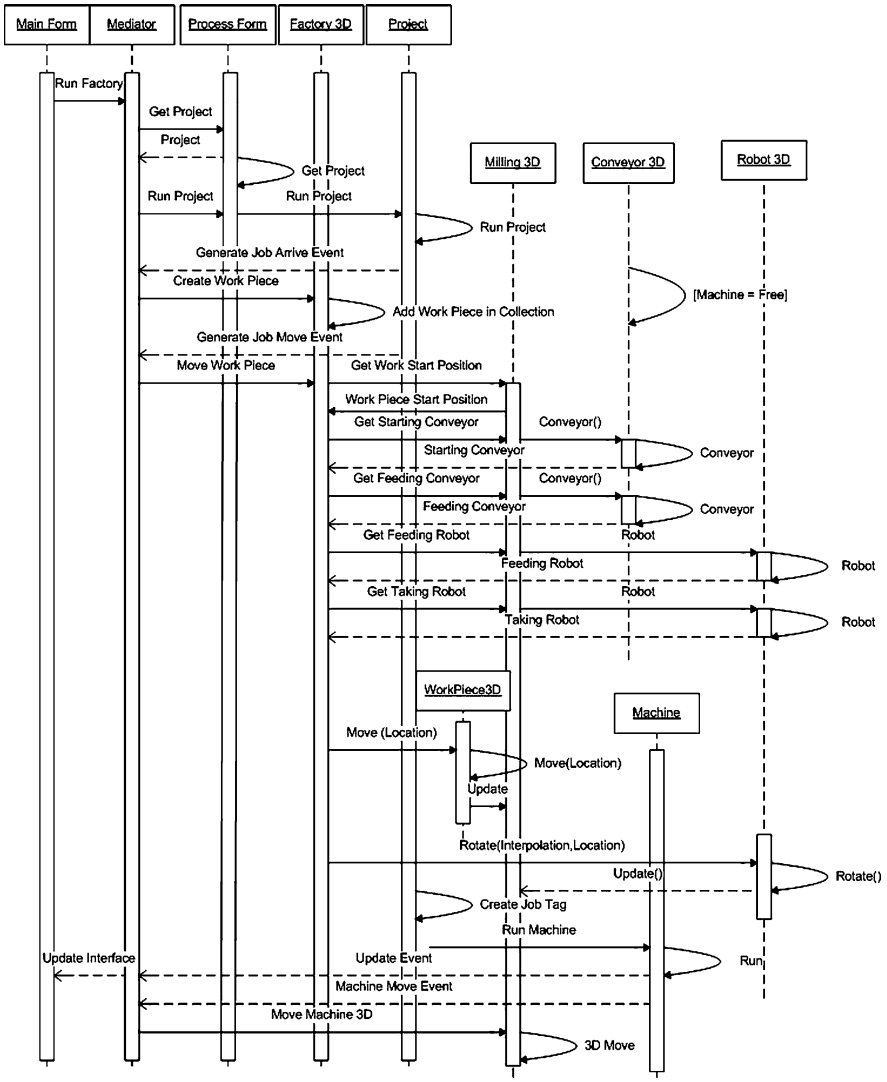


Fig. 11.65 VF Job Arrive Event Handling UML Sequence

engine to decide weather work piece is in process queue or not and work piece is currently inside machine or out side machine, respectively. JobName and WorkPieceNumber properties are used to identify each work piece separately. MachineName property is used to find out the machine name in which work piece is currently inside the machine. MaterialType property can be used to set the material of work piece. ModelsPath property can be used for setting the

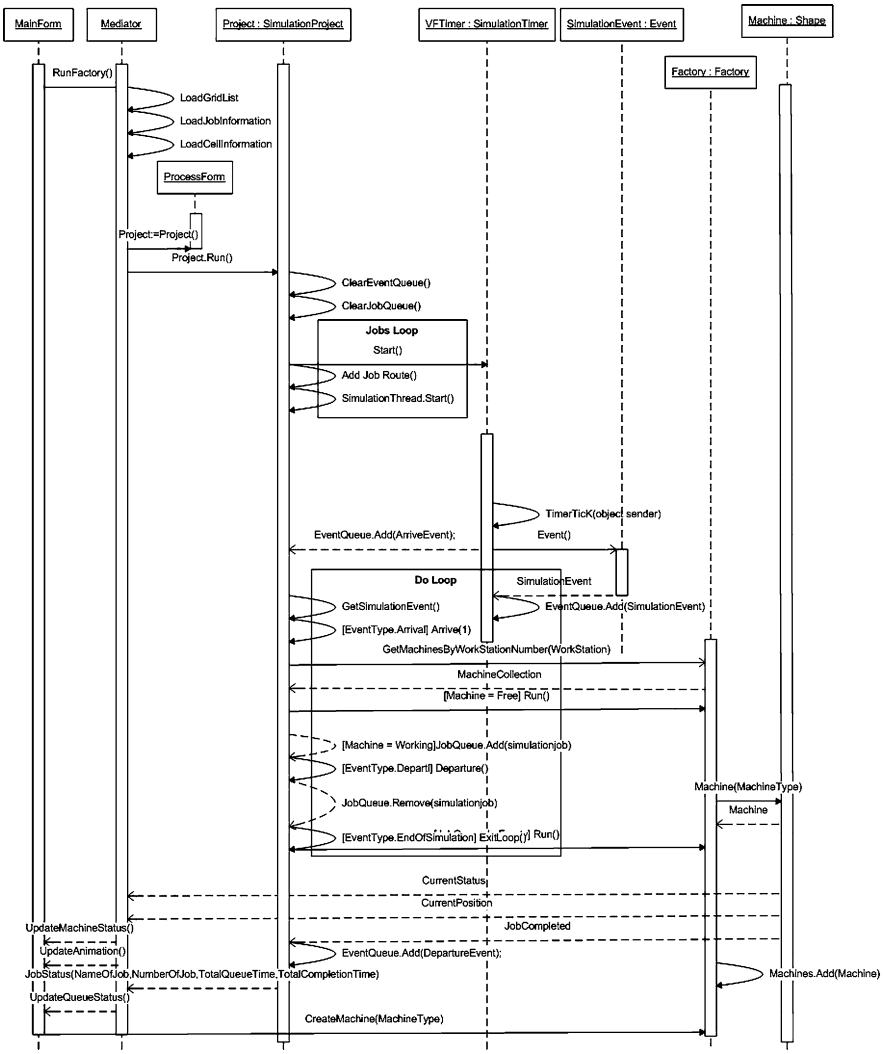


Fig. 11.66 VF Simulation Events Handling UML Sequence

drive location where DirectX files of work piece are located. ChangeMesh method is used to draw cutting effect on work piece. GetWorkPiecePosition is used to find out the current location or position in X, Y and Z coordinates in 3D space. MoveWorkPiece method is used to control the movement of work piece in 3D space. Rotate method is used to rotate the work piece in 3D space in X, Y and Z coordinates.

(d) *Gantry3D class description*

Gantry3D class represents the 3D gantry machine in VF. Gantry3D class has two properties and six methods. Gantry3D loads the DirectX files and create

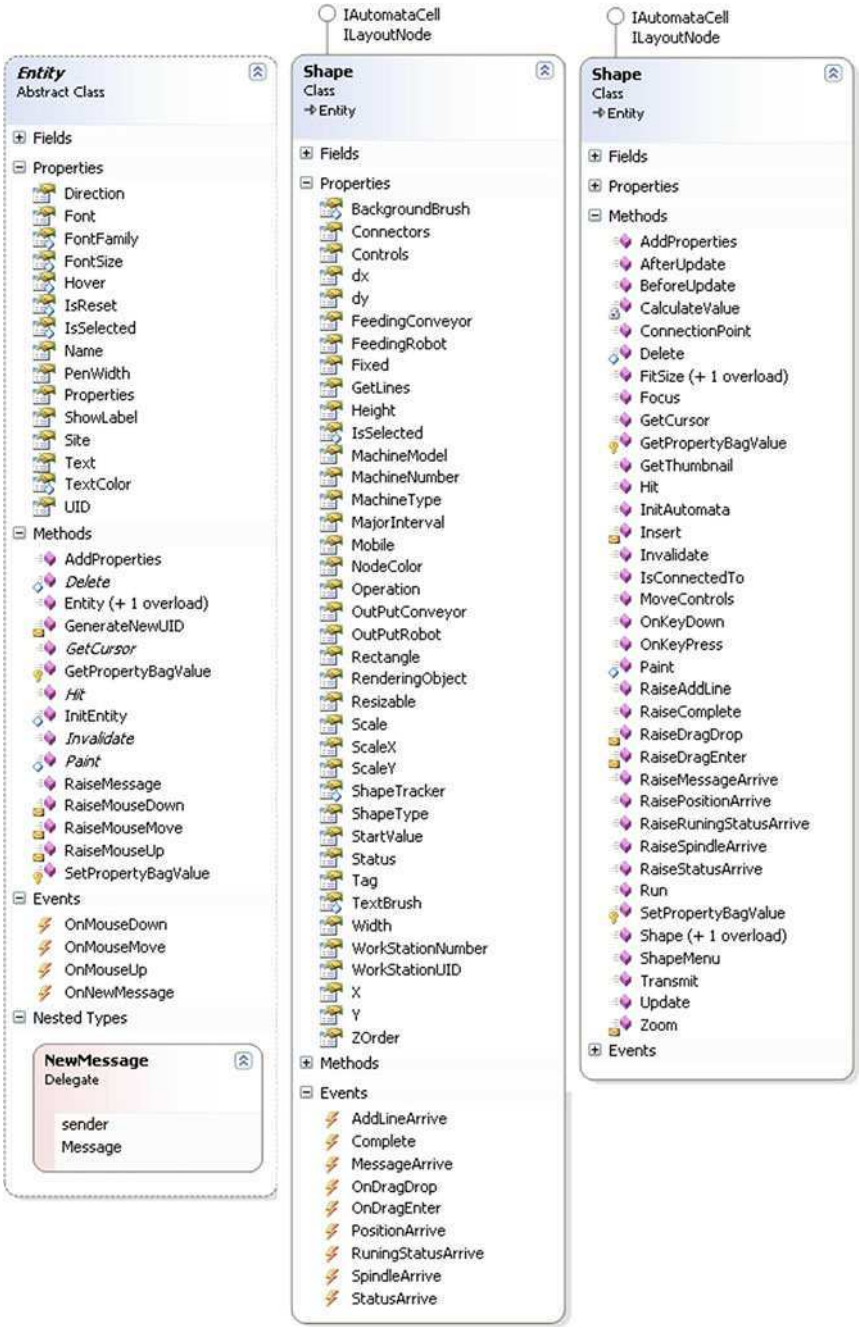


Fig. 11.67 UML Static Structure of VirtualFactory.VirtualFactoryLib Classes

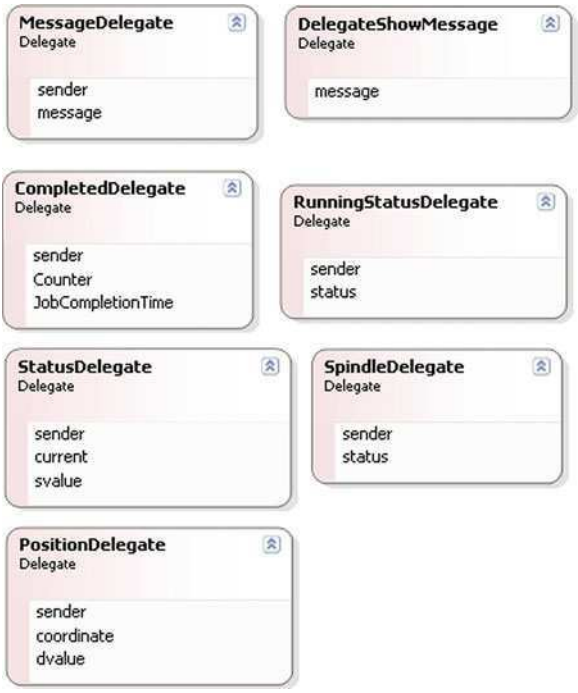


Fig. 11.68 UML Static Structure of VirtualFactory. VirtualFactoryLib Delegates

gantry machine for 3D rendering. Each gantry machine is identified by MachineName property. Move method is used to control the movement of 3D hook of gantry machine. ModelsPath property can be use for setting the drive location where DirectX files of gantry machine are located. GetSlidePosition and GetSliderPosition methods are used to find out the current position of Slide and Slider, respectively. Factory3D delete or remove 3D gantry machine from 3D space by calling Destroy method.

(e) *WorkStation3D class description*
WorkStation3D class represents the 3D cell in VF. Cell defines a location on factory floor that is used to group machines. WorkStation3D class has only two methods WorkStation3D which is a class constructor and a Destroy method used to remove or delete work station from 3D space.

(f) *Turning3D class description*
Turning3D class represents the 3D Turning machine in VF. Turning3D class has two properties and four methods. Turning3D loads the DirectX files and create turning machine for 3D rendering. Each turning machine is identified by MachineName property. Move method is used to control the movement of 3D work piece inside turning machine. GetWorkPieceStartPosition method is use by Robot to decide where it should drop/pick work piece inside machine. ModelsPath property can be used for setting up the drive location where DirectX files of turning machine are located. Factory3D delete or remove 3D turning machine from 3D space by calling Destroy method.

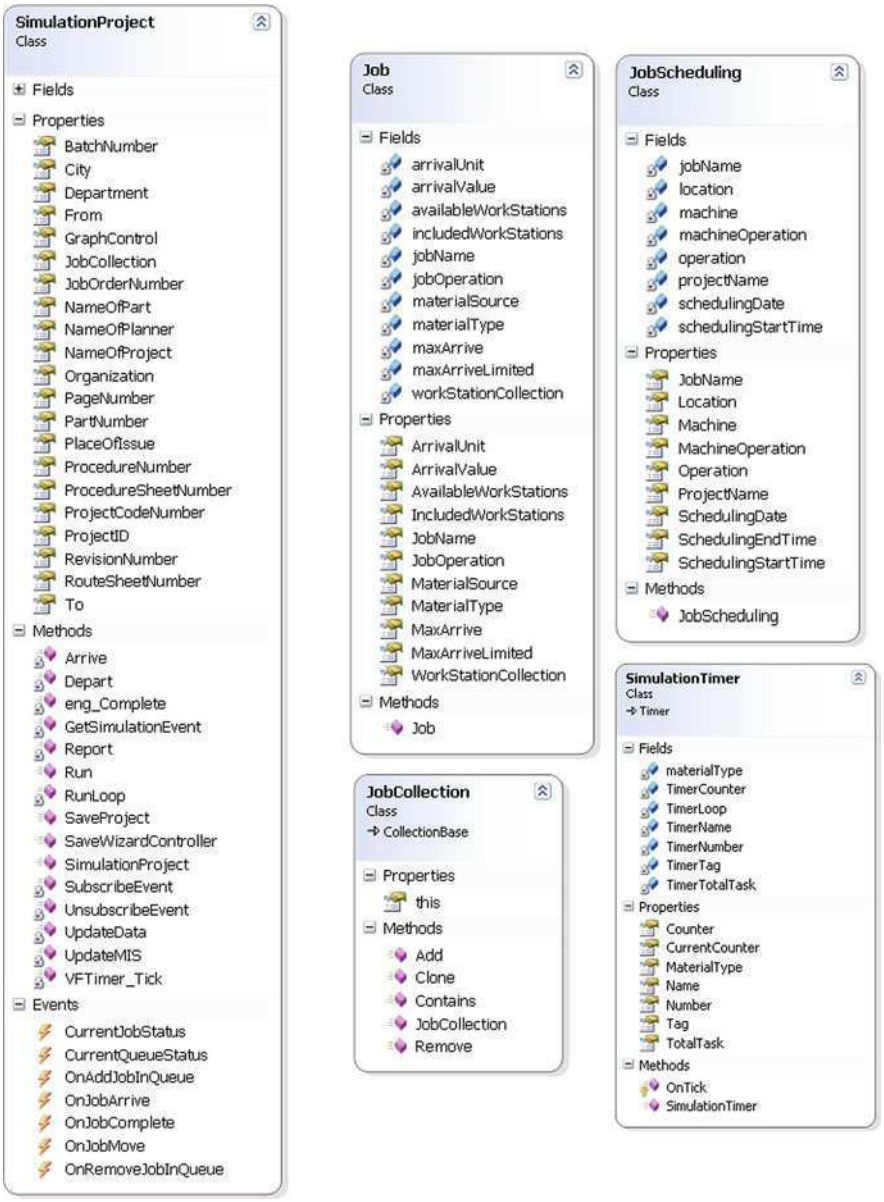


Fig. 11.69 UML Static Structure of VirtualFactory.VirtualFactoryLib Classes

(g) *Welding3D class description*

Welding3D class represents the 3D Robot holding welding torch in VF. Welding3D class has two properties and seven methods. Welding3D loads the DirectX files and create welding machine for 3D rendering. Each welding

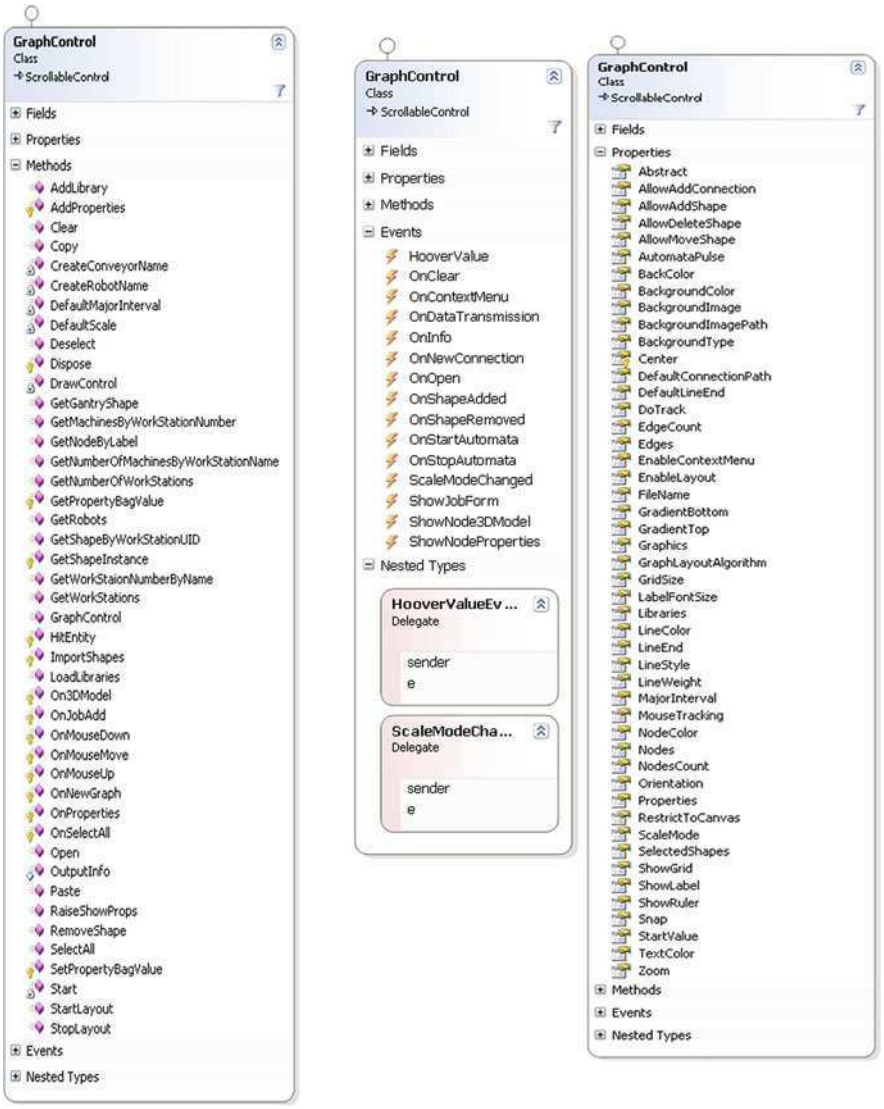


Fig. 11.70 UML Static Structure of VirtualFactory.VirtualFactoryLib Classes

machine is identified by MachineName property. Move method is used to control the movement of robot's arms and MoveRobot method is used to control the movement of robot itself. GetWorkPieceStartPosition method is use by Robot to decide where it should drop/pick work piece. Reset method is used to reset robot's all parts to its original position. GetRobotPosition is used to find out the current welding machine position in 3D space in X, Y and Z

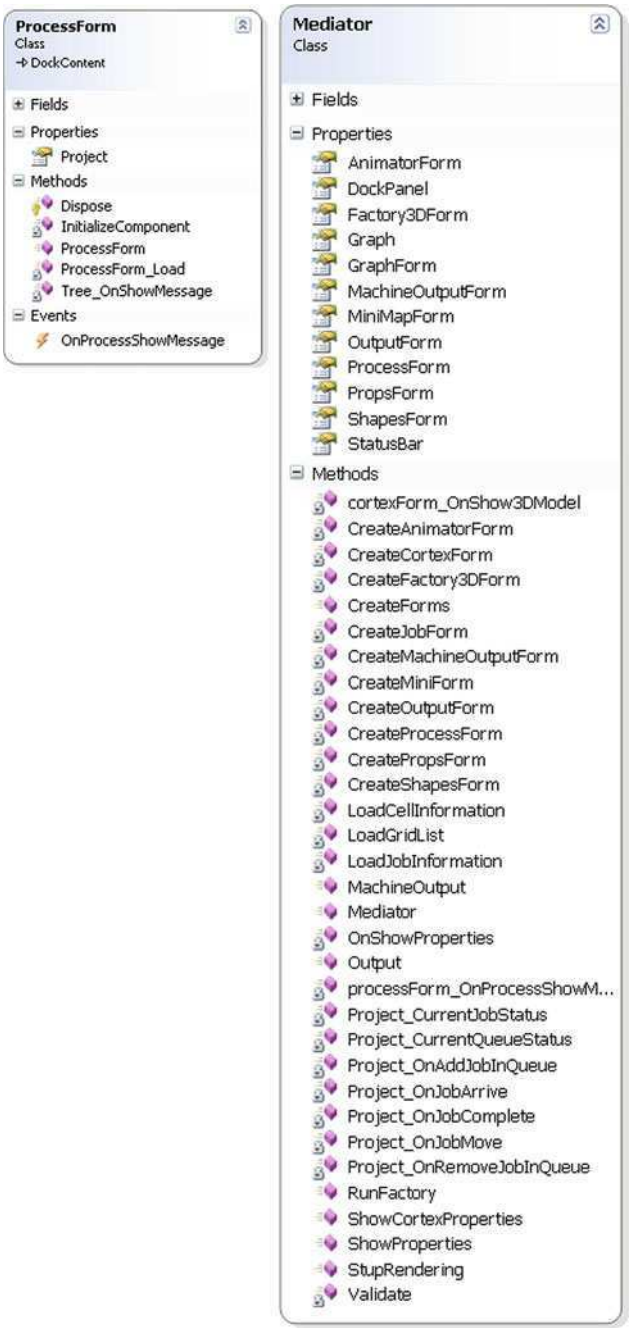


Fig. 11.71 UML Static Structure of VirtualFactory Classes

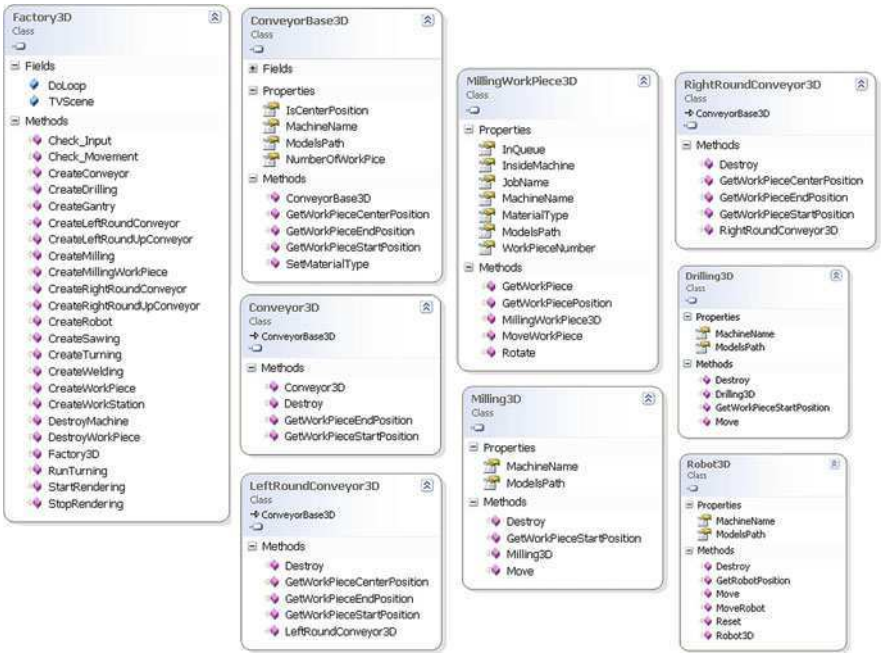


Fig. 11.72 UML Static Structure of VirtualFactory.Factory3DLibrary Classes

coordinates. ModelsPath property can be used for setting the drive location where DirectX files of welding machine are located. Factory3D delete or remove 3D welding machine from 3D space by calling Destroy method.

Figure 11.74 details the UML static structure of Machine classes. Each class is described below.

(a) *CNCMilling class description*
CNCMilling class represents the functionality of CNC milling machine in VF. CNCMilling class is inherited from Shape class. This class has three properties and several methods to control presence of milling machine. GetMachine property is used by simulation engine and this return object of itself. PartProgramFile property used by factory layout designer to set the location of part program file in local computer. Status property returns the current status of the machine. More concern methods of CNCMilling class are ExecuteFile, Run, ValidateCode, ValidateFile, and Update. Run method is use by Simulation Engine to start a machine. ExecuteFile method called by Run method to start execution of part program. Validate method is used to validate the part program file by calling ValidateCode. ValidateCode verifies each code in part program according to defined standards. ExecuteFile method implemented its own interpreter for CNC milling to interpret part program codes.

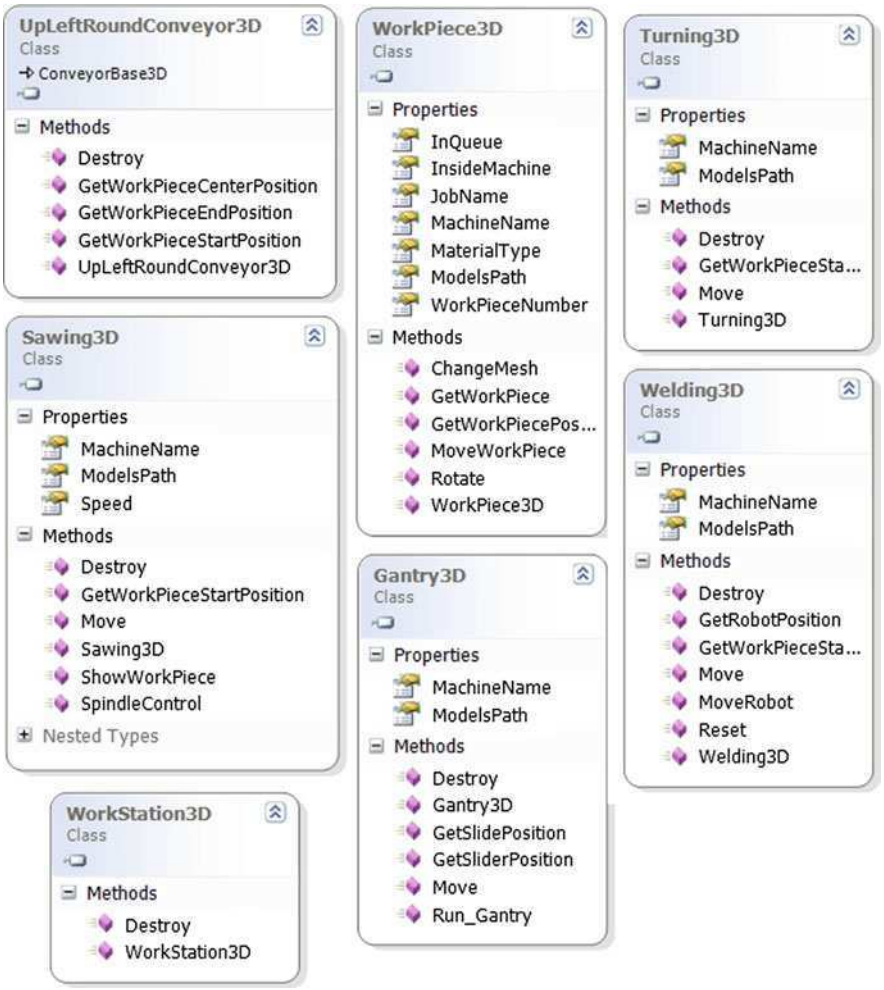


Fig. 11.73 UML Static Structure of VirtualFactory.Factory3DLibrary Classes

(b) *CNCDrilling class description*
CNCDrilling class represents the functionality of CNC drilling machine in VF. CNCDrilling class is inherited from Shape class. This class has three properties and several methods to control presence of milling machine. GetMachine property is used by simulation engine and this return object of itself. PartProgramFile property used by factory layout designer to set the location of part program file in local computer. Status property returns the current status of the machine. More concern methods of CNCMilling class are ExecuteFile, Run, ValidateCode, ValidateFile and Update. Run method is use by Simulation Engine to start a machine. ExecuteFile method called by Run method to start execution of part program. Validate method is used to validate the part

program file by calling ValidateCode. ValidateCode verifies each code in part program according to defined standards. ExecuteFile method implements its own interpreter for CNC drilling to interpret part program codes. The AddLine event is fired by ExecuteFile with current position in X, Y, and Z coordinates of work piece and spindle as parameters. AddLine event captured by Simulation Engine draws 3D cutting effects on work piece.

(c) *CNCDrilling class description*

CNCSawing class represents the functionality of CNC sawing machine in VF. CNCSawing class inherited from Shape class. This class has three properties

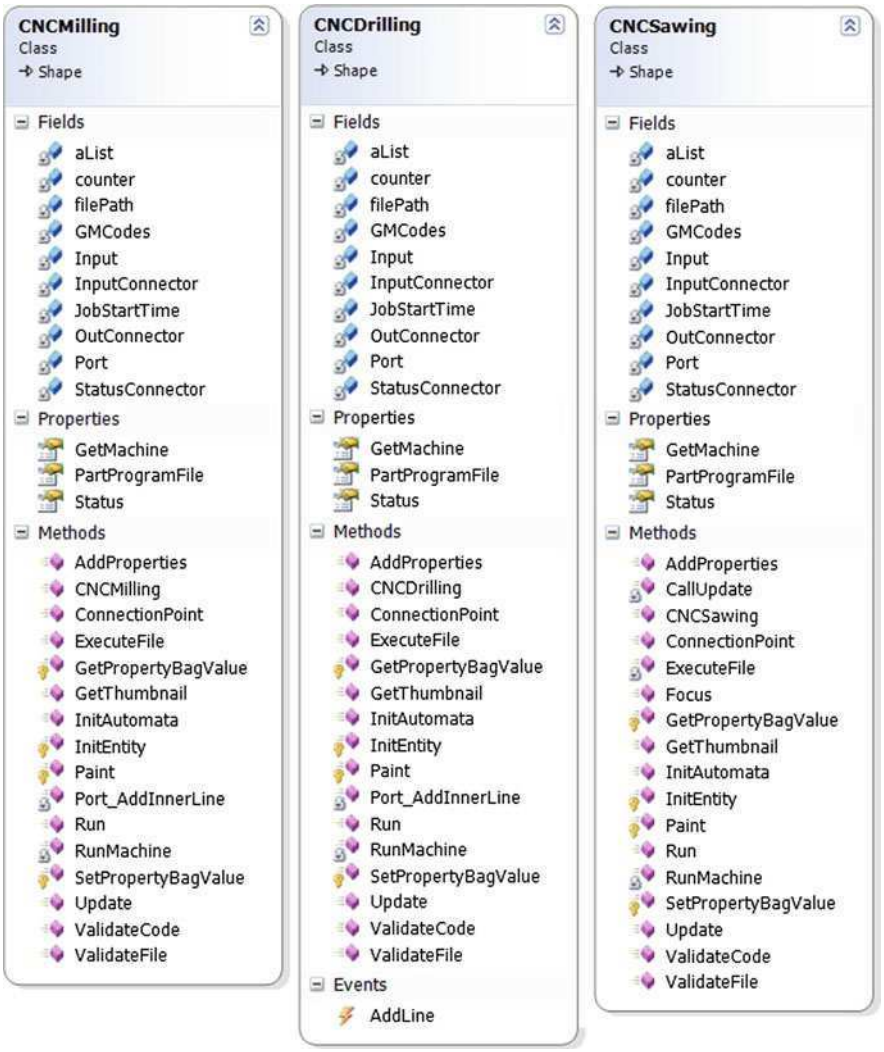


Fig. 11.74 UML Static Structure of VirtualFactory.Machines Classes

and several methods to control presence of sawing machine. GetMachine property is used by simulation engine and this return object of itself. PartProgramFile property used by factory layout designer to set the location of part program file in local computer. Status property returns the current status of the machine. Other concerned methods of CNCSawing class are ExecuteFile, Run, ValidateCode, ValidateFile and Update. Run method is used by Simulation

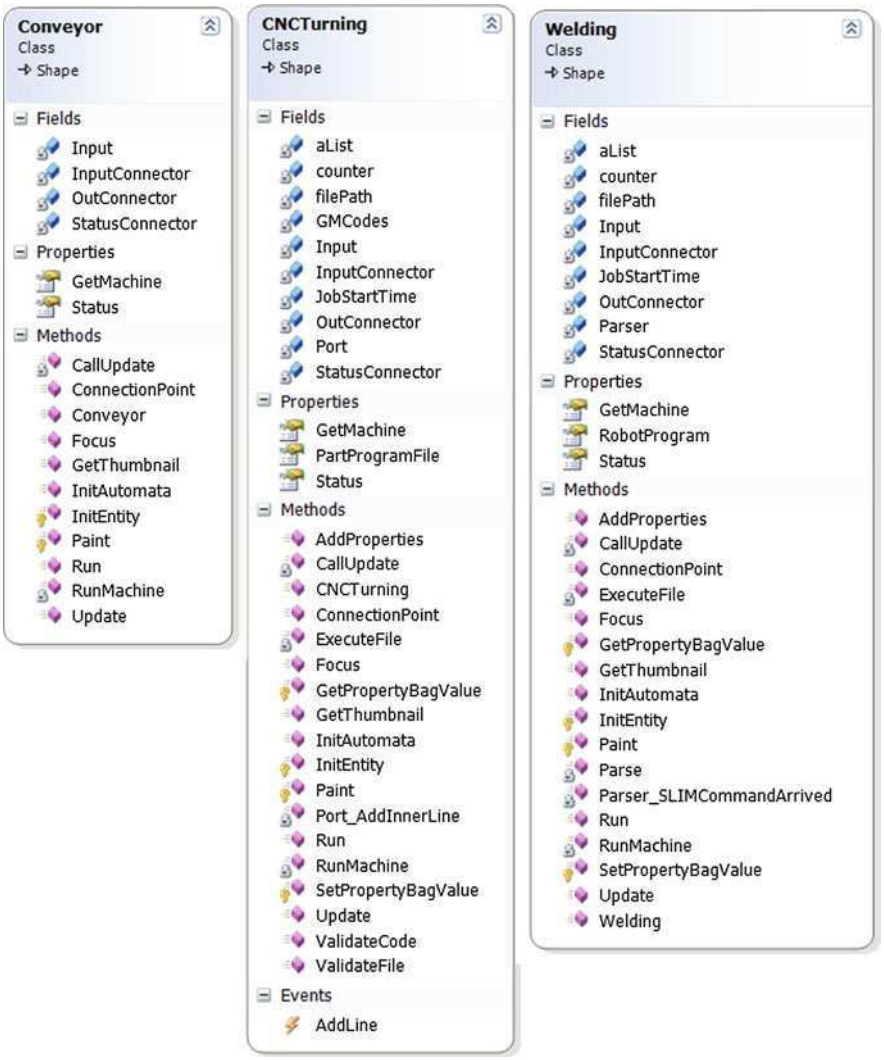


Fig. 11.75 UML Static Structure of VirtualFactory.Machines Classes

Engine to start a machine. ExecuteFile method called by Run method to start execution of part program. Validate method is used to validate the part program file by calling ValidateCode. ValidateCode verifies each code in part program according to defined standards. ExecuteFile method implements its own interpreter for CNC sawing to interpret part program codes.

Figure 11.75 gives more machine classes. Each of these classes is described below.

(a) *Conveyor class description*
Conveyor class represents the functionality of conveyor belt in VF. Conveyor class inherited from Shape class. This class has two properties and several methods to control presence of conveyor belt. GetMachine property is used by simulation engine and this return object of itself. Status property returns the current status of the conveyor belt. Run method is used by Simulation Engine to start conveyor belt. RunMachine method is called by Run method and is used to execute the conveyor belts instructions.

(b) *CNCTurning class description*
CNCTurning class represents the functionality of CNC turning machine in VF. CNCTurning class is inherited from Shape class. This class has two properties and several methods to control presence of turning machine. GetMachine property is used by simulation engine and this returns object of itself. PartProgramFile property used by factory layout designer to set the location of part program file in local computer. Status property returns the current status of machine. More concerned methods of CNCTurning class are ExecuteFile, Run, ValidateCode, ValidateFile, and Update. Run method is use by Simulation Engine to start a machine. ExecuteFile method is called by Run method to start execution of part program. Validate method is used to validate the part program file by calling ValidateCode. ValidateCode verifies each code in part program according to defined standards. ExecuteFile method implements its own interpreter for CNC turning to interpret part program codes. The AddLine event is fired by ExecuteFile with current position in X, Y, and Z coordinates of work piece and spindle as parameters. AddLine event captured by Simulation Engine draws 3D cutting effects in work piece.

(c) *Welding class description*
Welding class represents the functionality of welding machine in VF. Welding class is inherited from Shape class. This class has three properties and several methods to control presence of welding machine. GetMachine property is used by simulation engine and this returns object of itself. RobotProgram property used by factory layout designer to set the location of SLIM robot program file in local computer. Status property returns the current status of welding machine. Run method is use by Simulation Engine to start machine. ExecuteFile method called by Run method to start execution of SLIM robot program. ExecuteFile method implements its own SLIM compiler to execute robot program. Parse method is called by ExecuteFile method to parse whole robot program before actual execution of program.

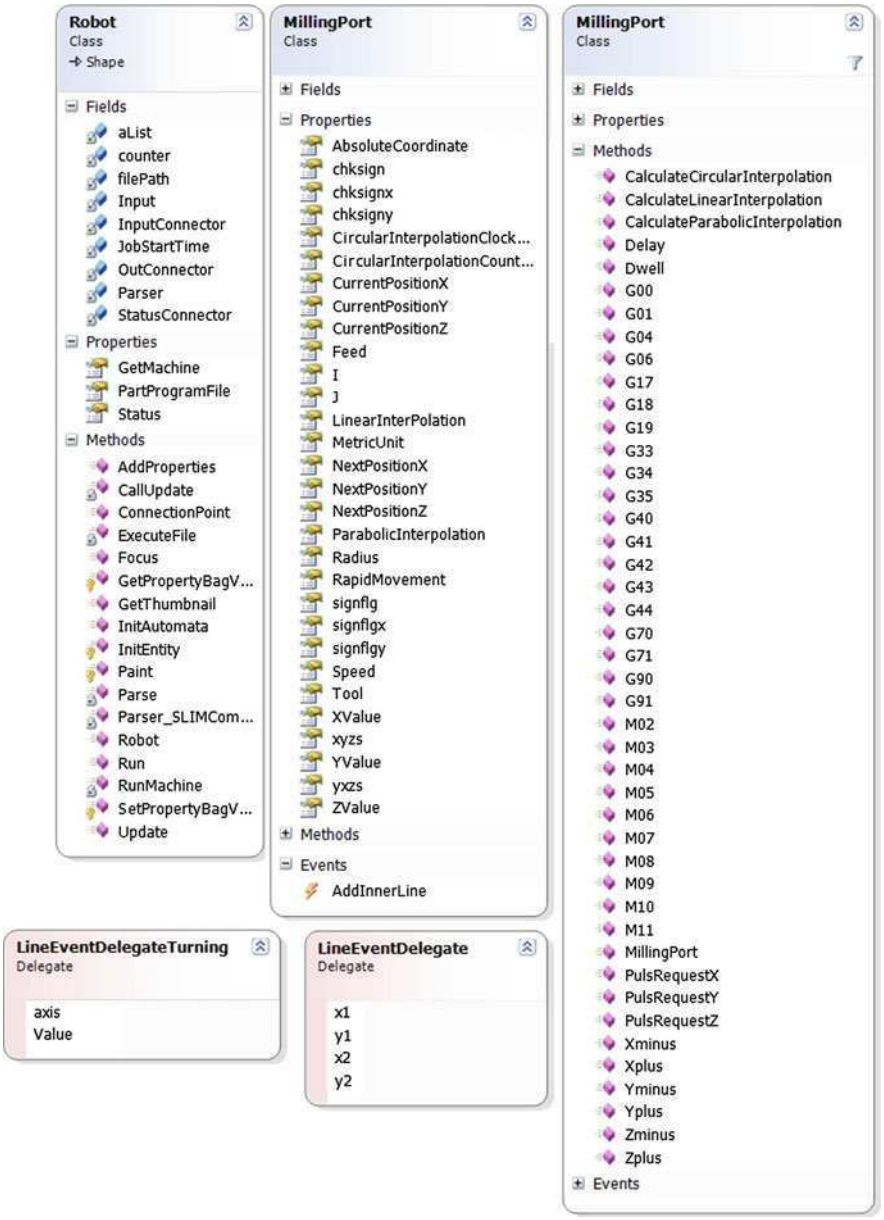


Fig. 11.76 UML Static Structure of VirtualFactory.Machines Classes and Delegates

The Parser_SLIMCommandArrived event is fired by ExecuteFile which is currently executing SLIM command. Parser_SLIMCommandArrived event captured by Simulation Engine controls 3D robot parts movements.

Figure 11.76 presents UML static structure for machine class and delegates. A description of these classes is given below.

(a) *MillingPort class description*
MillingPort class represents the interpreter of CNC milling part program. MillingPort class is also responsible for reading and writing signals to physical port of computer for controlling real CNC milling machine. The object of MillingPort class is created in CNCMilling class. ExecuteFile method in CNCMilling class use MillingPort object to interprets and controls the execution of part program. The property of MillingPort is used to pass part program information between CNCMilling class and MillingPort class. CalculateCircularInterpolation method is used to calculate the circular interpolation and it takes values to perform calculation from CurrentPositionX, CurrentPositionY, CurrentPositionZ, I and J property variables. After performing calculation CalculateCircularInterpolation method sets the NextPositionX, NextPositionY and NextPositionZ properties for handling work piece position and sets the PulseRequestX, PulseRequestY, and PulseRequestZ for physical port signaling. CalculateLinearInterpolation and CalculateParabolicInterpolation methods are used to perform linear and parabolic interpolation, respectively, and both methods use the same concept to perform calculation as CalculateCircularInterpolation method. In MillingPort class each code of part program has its own method to execute code, such as G00 and G17 code has its own methods to handle. The AddInnerLine event implemented by LineEventDelegate is fired by all three interpolations method such as CalculateParabolicInterpolation method with X1, Y1, X2, and Y2 parameters for creating 3D cutting effects on work piece at run time.

(b) *Robot class description*
Robot class represents the functionality of robot in VF. Robot class inherited from Shape class. This class has three properties and several methods to control presence of robot. GetMachine property is used by simulation engine and this return object of its self. PartProgramFile property used by factory layout designer to set the location of SLIM robot program file in local computer. Status property returns the current status of robot. Run method is use by Simulation Engine to start robot. ExecuteFile method called by Run method to start execution of SLIM robot program. ExecuteFile method implements its own SLIM compiler to execute robot program. Parse method is called by ExecuteFile method to parse whole robot program before actual execution of program. The Parser_SLIMCommandArrived event is fired by ExecuteFile while currently executing SLIM command. Parser_SLIMCommandArrived event catch by Simulation Engine to control 3D robot parts movements.

Figure 11.77 presents the UML static structure for more machine classes. The description of these classes is listed below.

(a) *TurningPort class description*
TurningPort class represents the interpreter of CNC turning part program. TurningPort class is also responsible for reading and writing signals to

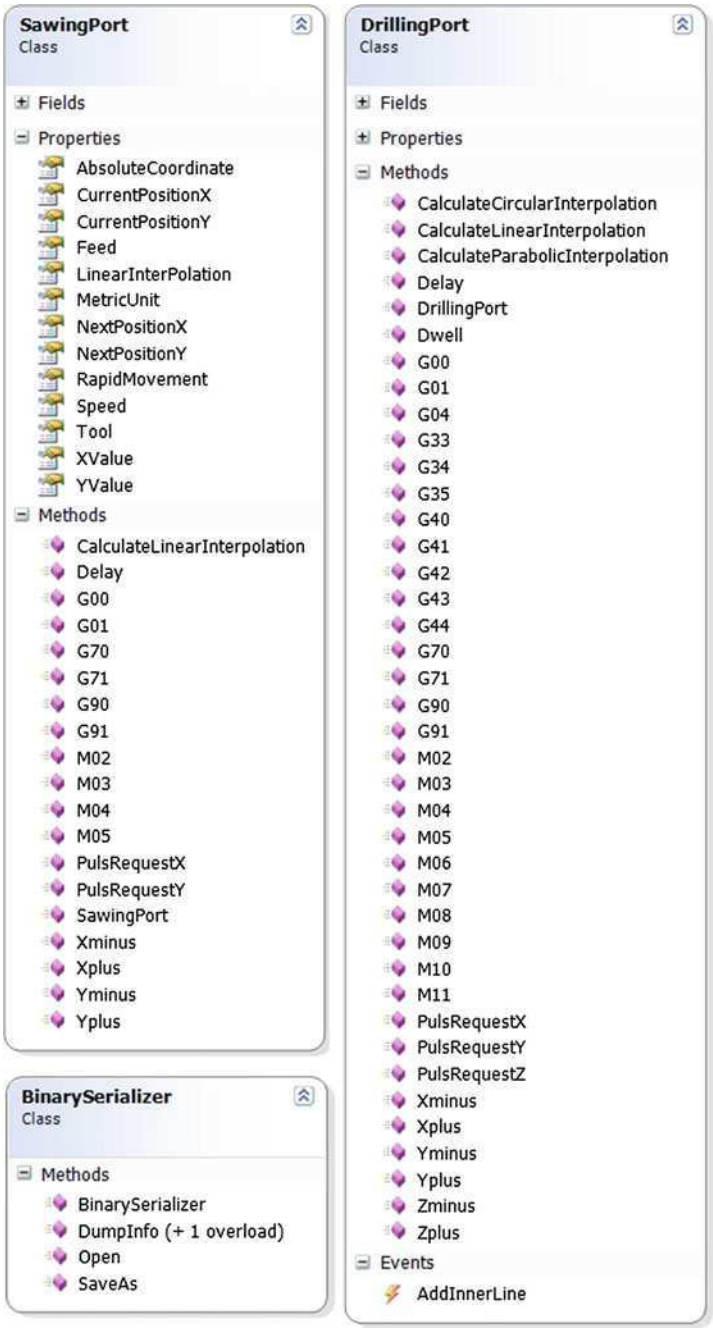


Fig. 11.77 UML Static Structure of VirtualFactory.Machines Classes

physical port of computer for controlling external physical CNC Turning machine. The object of TurningPort class is created in CNCTurning class. ExecuteFile method in CNCTurning class use TurningPort object to interpret and control the execution of part program. The properties of TurningPort used to pass part program information between CNCTurning class and TurningPort class. CalculateCircularInterpolation method is used to calculate the circular interpolation and it takes values to perform calculation from CurrentPositionX, CurrentPositionZ, I and K property variables. After performing calculation CalculateCircularInterpolation method sets the NextPositionX and NextPositionZ properties for handling work piece position and sets the PulseRequestX and PulseRequestZ for physical I/O port signaling. CalculateLinearInterpolation and CalculateParabolicInterpolation methods are used to perform linear and parabolic interpolation, respectively, and both methods use the same concept to perform calculation as CalculateCircularInterpolation method. In TurningPort class each code of part program has its own method to handle code, such as G90 and G91 code has its own methods to execute. The AddInnerLine event implemented by LineEventDelegateTurning is fired by all three interpolations method such as CalculateParabolicInterpolation method with coordinate and its value as parameters for creating 3D cutting effects on work piece at run time.

(b) *SawingPort class description*

SawingPort class represents the interpreter of CNC sawing part program. SawingPort class is also responsible for reading and writing signals to physical port of computer for controlling external physical CNC sawing machine. The object of SawingPort class is created in CNCSawing class. ExecuteFile method in CNCSawing class use SawingPort object to interprets and controls the execution of part program. The properties of SawingPort used to pass part program information between CNCSawing class and SawingPort class. CalculateLinearInterpolation method is used to calculate the linear interpolation and it takes values to perform calculation from CurrentPositionX and CurrentPositionY property variables. After performing calculation CalculateLinearInterpolation method sets the NextPositionX and NextPositionY properties for handling work piece position and sets the PulseRequestX and PulseRequestY for physical port signaling. In SawingPort class each code of part program has its own method to handle code, such as G90 and G91 code has its own methods to execute.

(c) *DrillingPort class description*

DrillingPort class represents the interpreter of CNC drilling part program. DrillingPort class is also responsible for reading and writing signals to physical port of computer for controlling external physical CNC drilling machine. The object of DrillingPort class is created in CNCDrilling class. ExecuteFile method in CNCDrilling class uses DrillingPort object to interpret and control the execution of part program. The properties of DrillingPort used to pass part program information between CNCDrilling class and DrillingPort class. CalculateCircularInterpolation method is used to calculate the circular

interpolation and it takes values to perform calculation from CurrentPositionX, CurrentPositionY, CurrentPositionZ, I and J property variables. After performing calculation CalculateCircularInterpolation method sets the NextPositionX, NextPositionY and NextPositionZ properties for handling work piece position and sets the PulseRequestX, PulseRequestY, and PulseRequestZ for physical port signaling. CalculateLinearInterpolation and CalculateParabolicInterpolation methods are used to perform linear and parabolic interpolation, respectively, and both methods used the same concept to perform calculation as CalculateCircularInterpolation method. In TurningPort class each code of part program has its own method to handle code, such as G70 and G71 code has its own methods to execute. The AddInnerLine event implemented by LineEventDelegate is fired by all three interpolations method such as CalculateParabolicInterpolation method with X1, Y1, X2, and Y2 parameters for creating 3D cutting effects on work piece at run time.

Figures 11.78, 11.79, and 11.80 present UML static structure for virtual factory BasicShapes and other minor classes. Enumerations are displayed in Fig. 11.81. A description of this enumeration is given below.

(a) *BasicShapestypes enumeration description*
BasicShapesTypes enumeration is used to distinguish between BaseShapes types. This value set in the constructor of SimpleNode and TextLabel objects. After creating the SimpleNode and TextLabel it is easy for Simulation Engine to identify whether that it is SimpleNode or TextLabel. Using BasicShapesTypes there is no fourth option to check and type of BasicShapes.

(b) *ConvasBackGround enumeration description*
ConvasBackGround enumeration is used to set the back ground of factory layout. Factory layout designer just have only three options to choose from, which are FlatColor, Gradient and Image.

(c) *Direction enumeration description*
Direction enumeration is used to set the physical direction the machine on factory floor. Factory layout designer only has four directions to choose from, Front, Back, Right, and Left. These directions are related to 3D view and default camera position in 3D space.

(d) *MachineCurrentStatus enumeration description*
MachineCurrentStatus enumeration is used to update the Simulation Engine about the running status of the machine. There are two status of machine, Free and Working.

(e) *MachineSpindle enumeration description*
MachineSpindle enumeration is used to update the Simulation Engine about the status of machine spindle. Machine spindle can be in one of two states, ON or OFF state.

(f) *ConnectionWeight enumeration description*
ConnectionWeight enumeration is used to control the thickness of line between two shapes. Thickness of line can be one of three choices, Thin, Medium, and Thick.

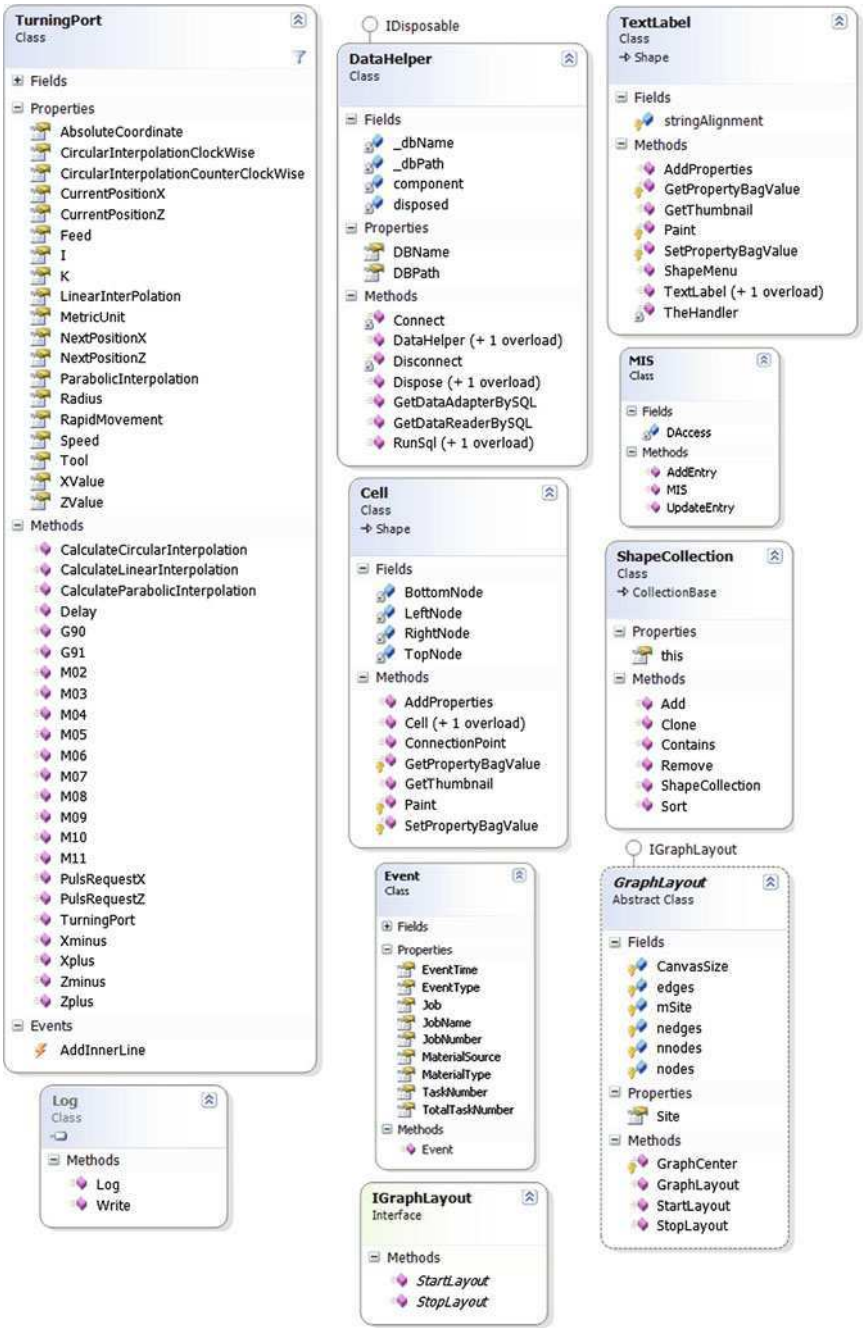


Fig. 11.78 UML Static Structure of VirtualFactory and VirtualFactory.BasicShapes Classes

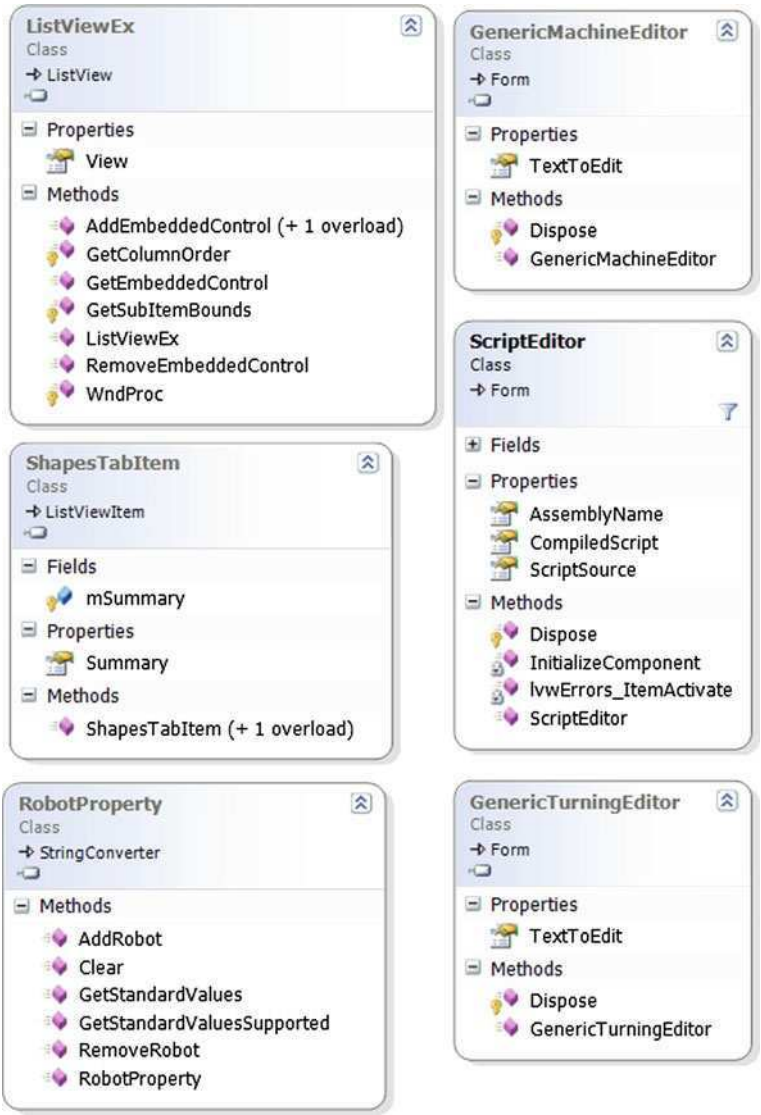


Fig. 11.79 UML Static Structure of VirtualFactory Classes

(g) *CurrentInformation enumeration description*
CurrentInformation enumeration is used to update Simulation Engine about the current status of different parameter of machine. Machine uses CurrentStatus Arrive event to pass current status to Simulation Engine. CurrentStatusArrive event has three parameters: Sender, Current, svalue. Current parameter is a variable of type CurrentInformation and holds the one of defined values of

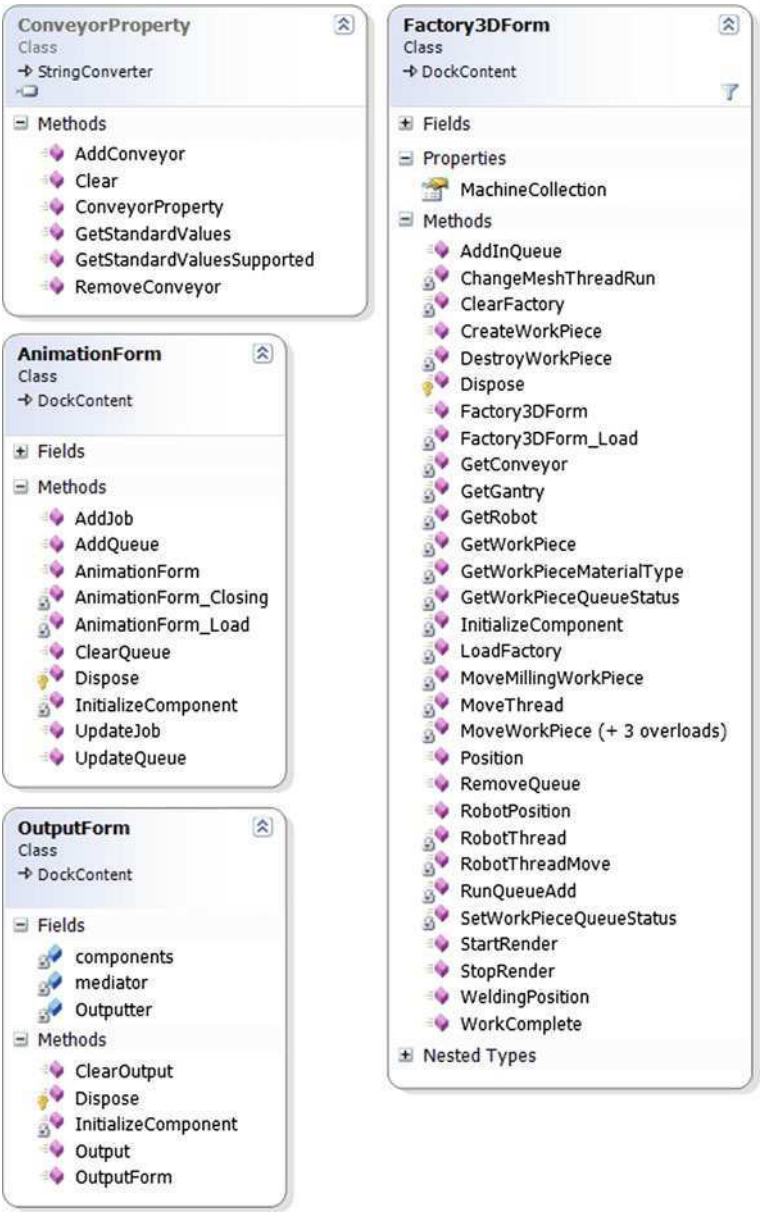


Fig. 11.80 UML Static Structure of VirtualFactory Classes

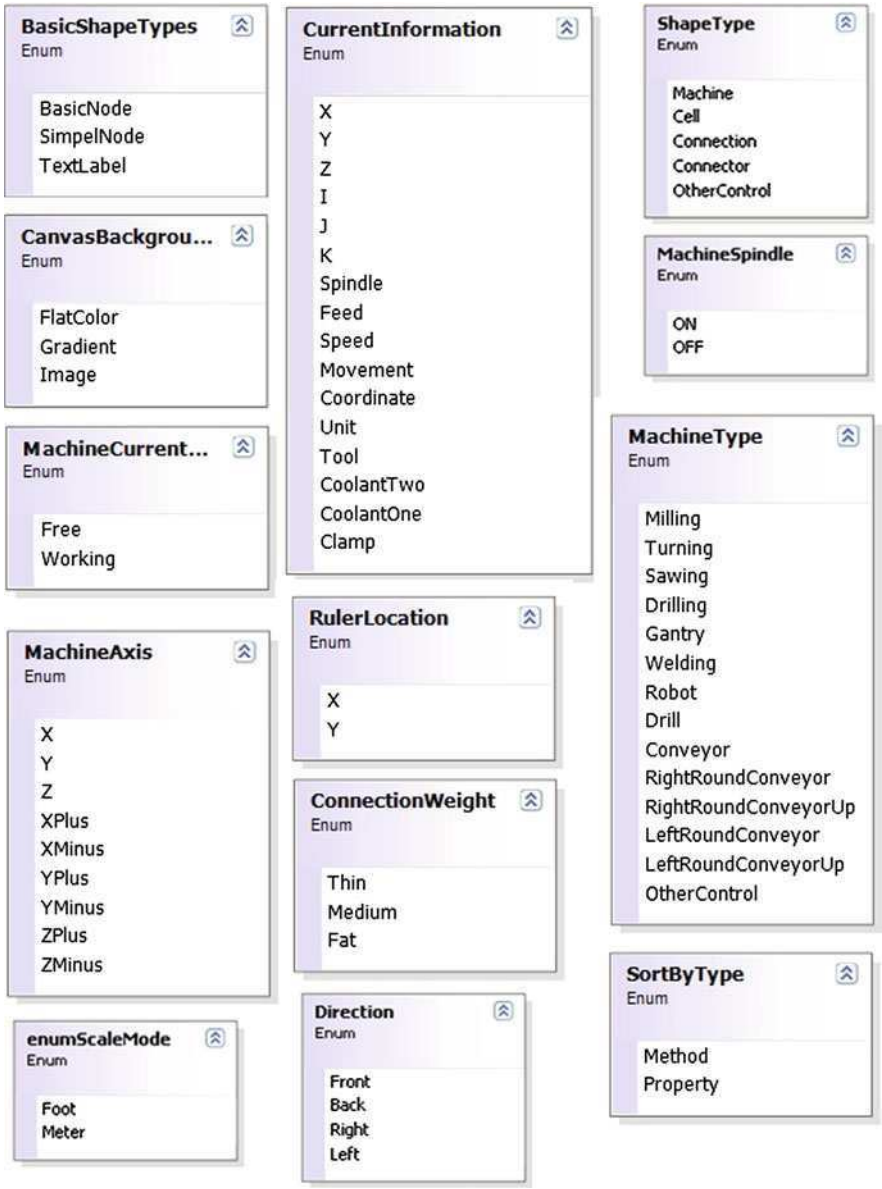


Fig. 11.81 UML Static Structure of VirtualFactory Enumerations

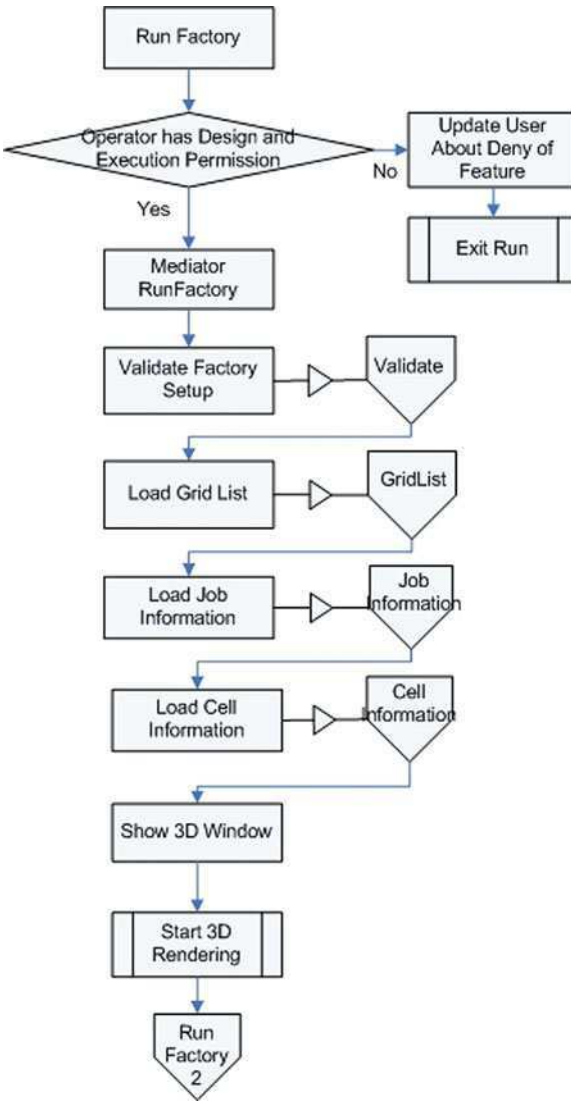


Fig. 11.82 Virtual Factory Flow Chart

CurrentInformation enumeration. Some of CurrentInformation enumeration choices are Spindle, Feed, Speed, Movement, Coordinate, Tool, and others.

(h) *MachineType enumeration description*
MachineType enumeration is used to identify which type of Shape object it is. When new Shape object is created, MachineType variable is set in Shape constructor. Each Shape object can be one of these: Machine, Cell, Connection, Connector, or OtherControl.

(i) *enumScaleMode enumeration description*
enumScalMode enumeration is used to set the scale mode of factory layout window. Factory scale can be one of two, which are Foot and Meter. The default setting of layout window is Foot scale mode.

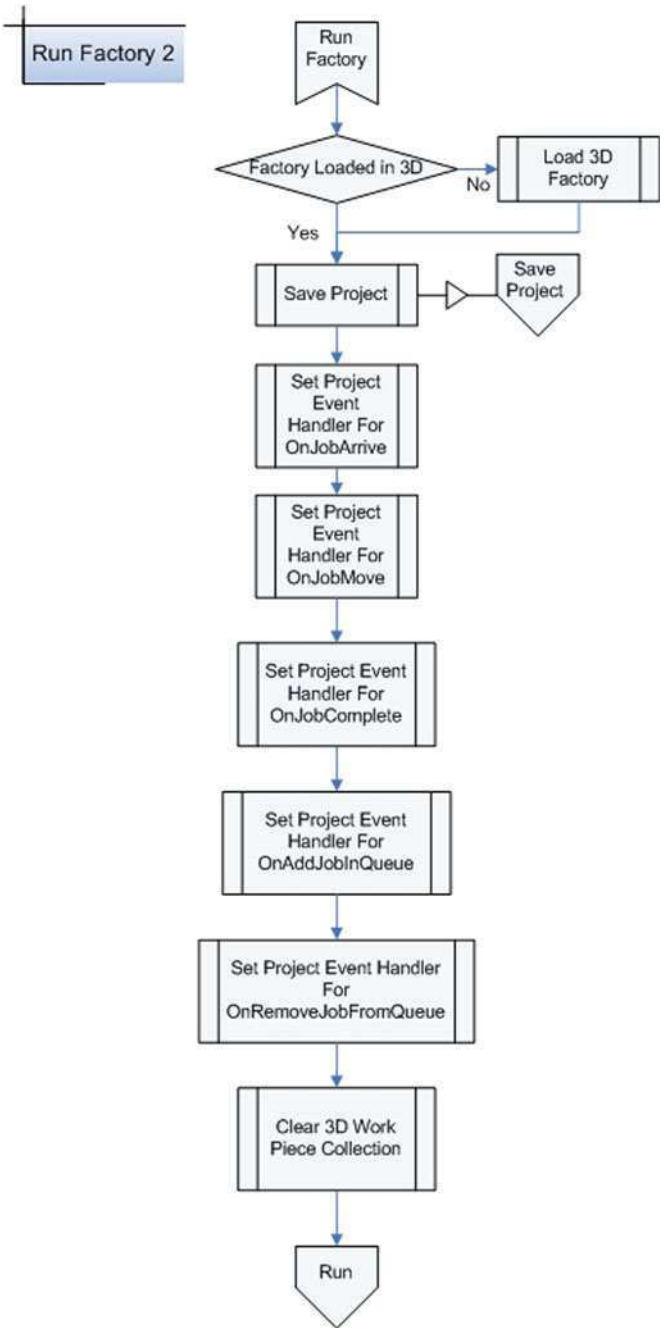
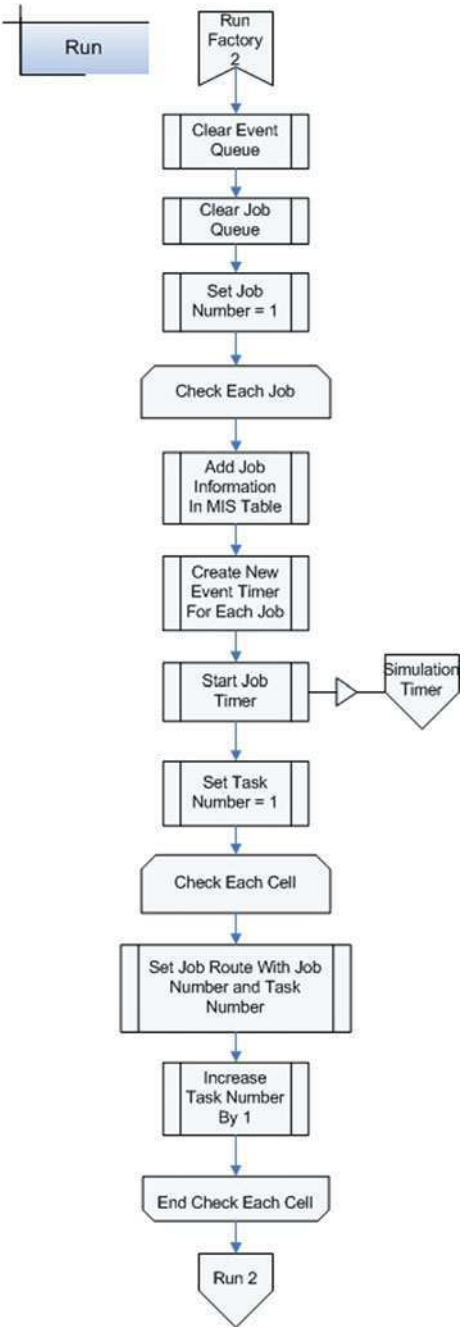


Fig. 11.82 (continued)

Fig. 11.82 (continued)



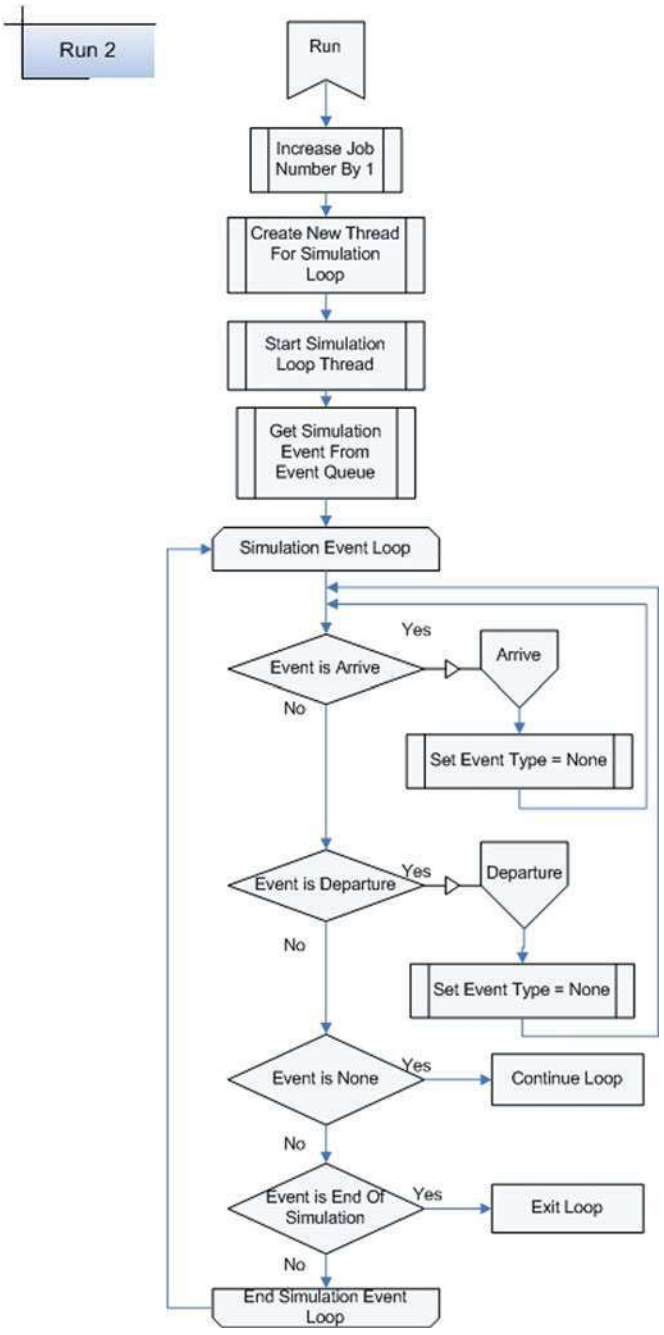


Fig. 11.82 (continued)

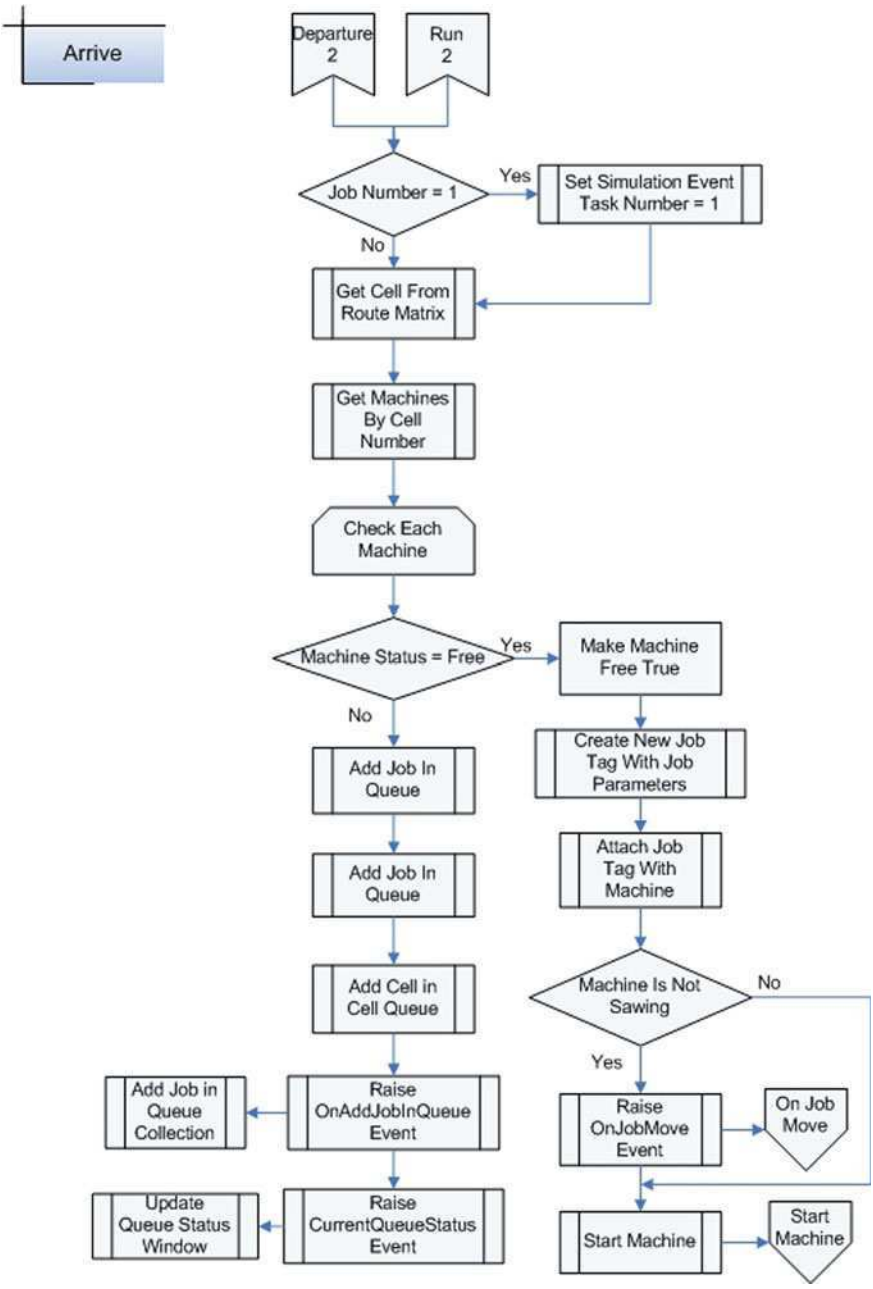


Fig. 11.82 (continued)

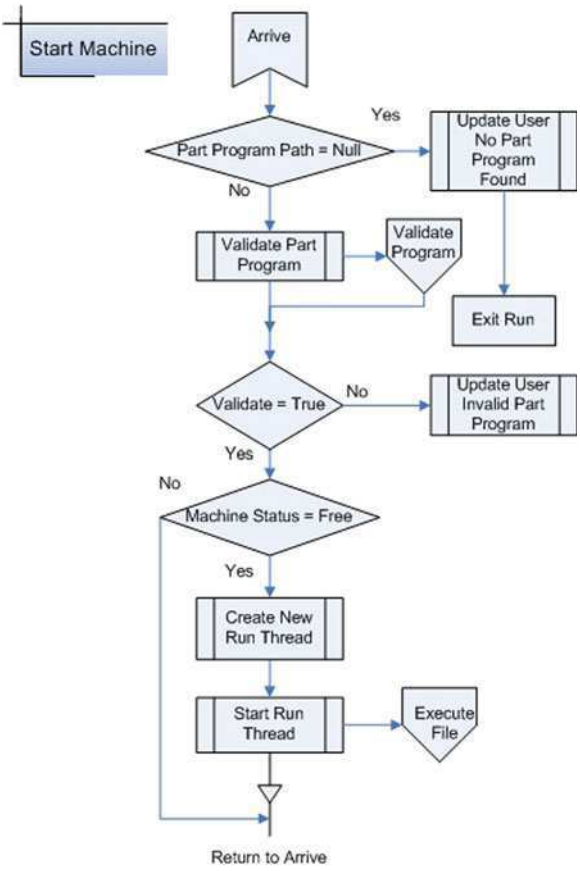


Fig. 11.82 (continued)

(j) *MachineType enumeration description*
MachineType enumeration is used to identify the type of machine. Simulation Engine first uses ShapeType enumeration to decide that selected Shape object is type of Machine. After making decision on machine Simulation Engine use MachineType enumeration to decide which type of machine it is. Shape object can be one of these machines: Milling, Turning, Sawing, Drilling, Gantry, Welding, Robot, Drill, Conveyor, RightRoundConveyor, RightRoundConveyorUp, LeftRoundConveyor, LeftRoundConveyorUp, and OtherControl.

(k) *RulerLocation enumeration description*
RulerLocation enumeration is used to set the scale location on factory layout window. RulerLocation variable can be X or Y.

(l) *MachineAxis enumeration description*
MachineAxis enumeration is used to pass current values of coordinates between machine class and its port class such as CNCMilling class use

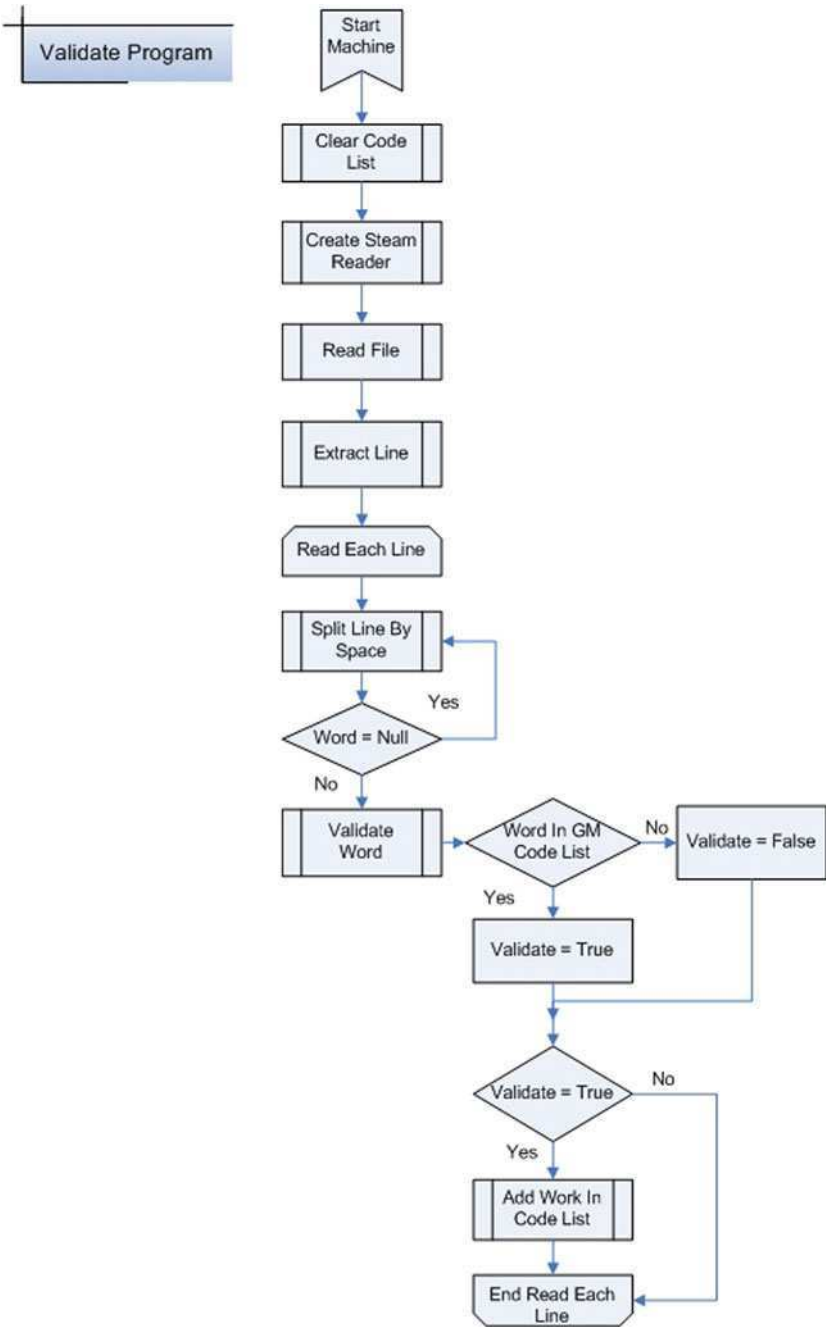


Fig. 11.82 (continued)

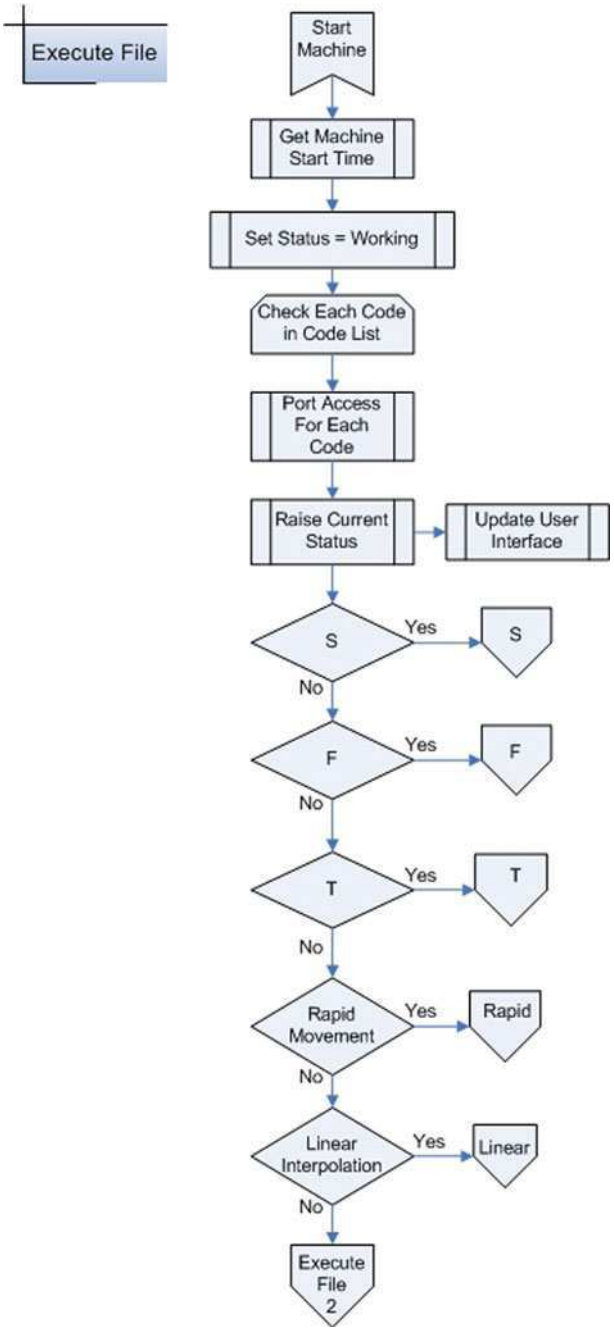


Fig. 11.82 (continued)

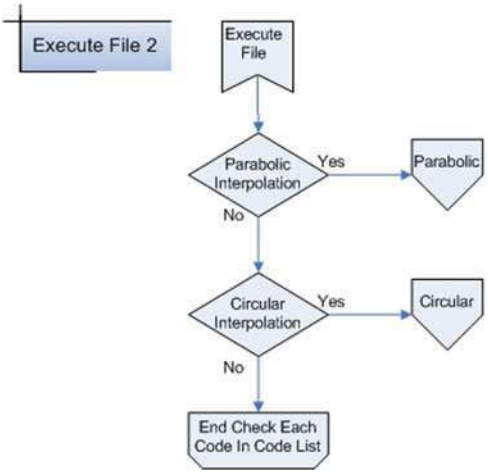


Fig. 11.82 (continued)

MachineAxis enumeration to pass coordination values to MillingPort class. Variable of type MachineAxis can hold one of these values: X, Y, Z, XPlus, XMinus, YPlus, YMinus, ZPlus, and ZMinus.

VF flow char presented in Fig. 11.82 elaborates working of the virtual factory. The flow chart is self-explanatory. The description of symbols used in the flow chart is given in Appendix III.

The intangible parameters are handled afterward. Figure 11.83 shows the flow chart for VF process plan. Figure 11.84 provides the flow chart for Setup Validation. Procurement related flow chart is given in Figs. 11.85, 11.86, 11.87, and 11.88. Save VF product function is depicted as a flow chart in Fig. 11.89. Purchasing related flow charts are shown as Figs. 11.90 and 11.91. User administration flow charts are presented as Figs. 11.92 and 11.93. The ledger functionality is shown as flow chart in Figs. 11.94 and 11.95. Quality assurance related flow charts are given in flow charts of Figs. 11.96 and 11.97.

11.5 System Integration for Virtual Manufacturing Facility

Hardware interface for integrating the virtual factory with the real manufacturing system is enormous. A section of such an interface is shown in Fig. 11.98. This interface shows control of a three-axis milling machine through an augmented reality controller (Figs. 11.99 and 11.100).

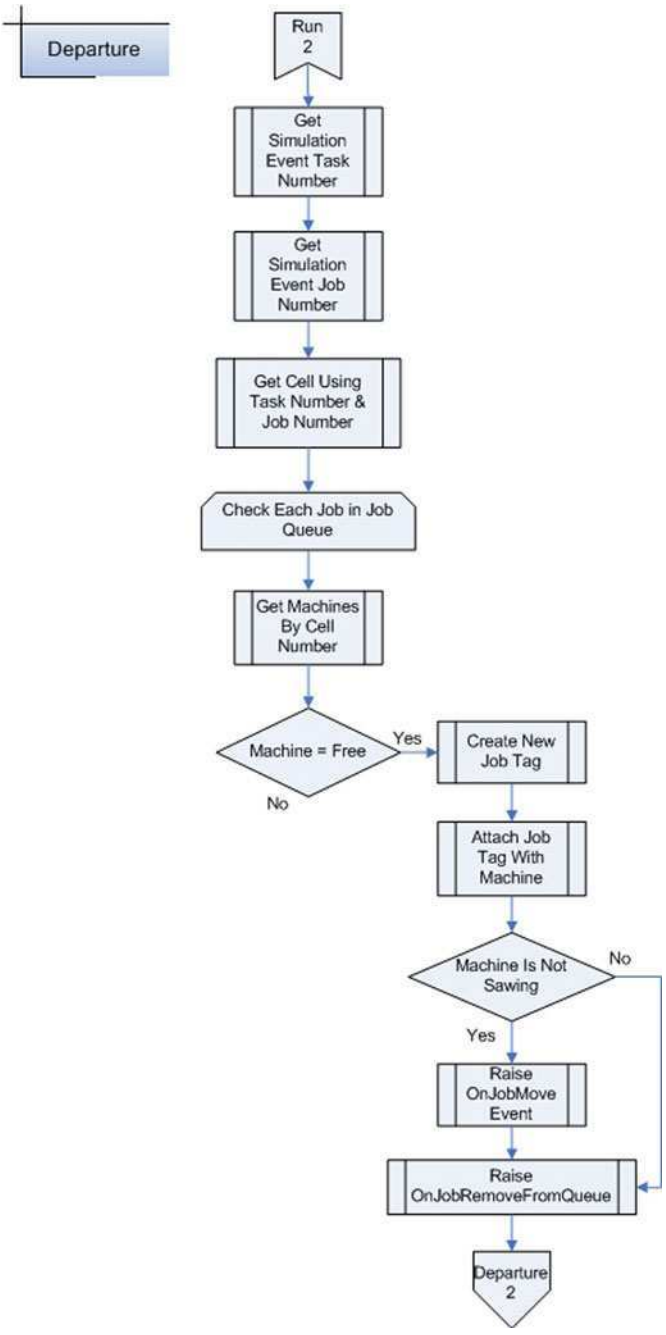


Fig. 11.82 (continued)

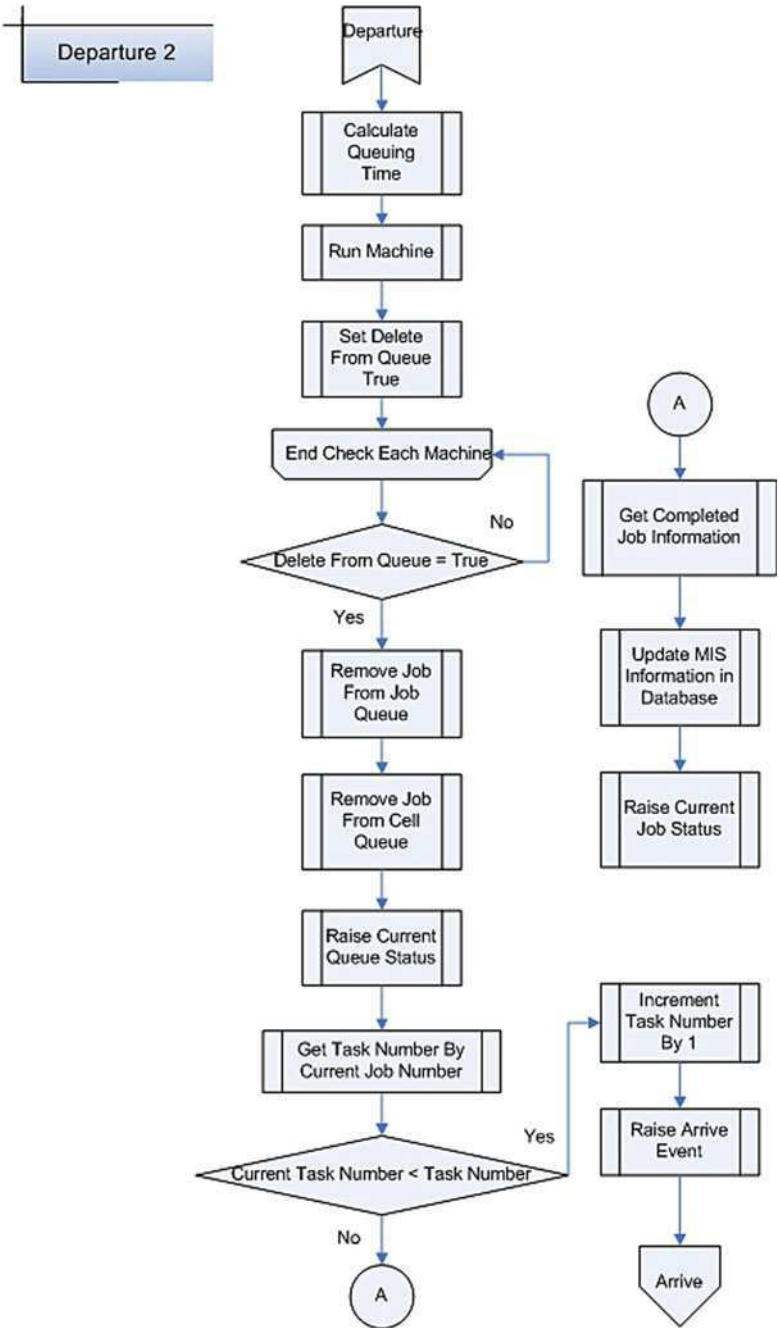


Fig. 11.82 (continued)

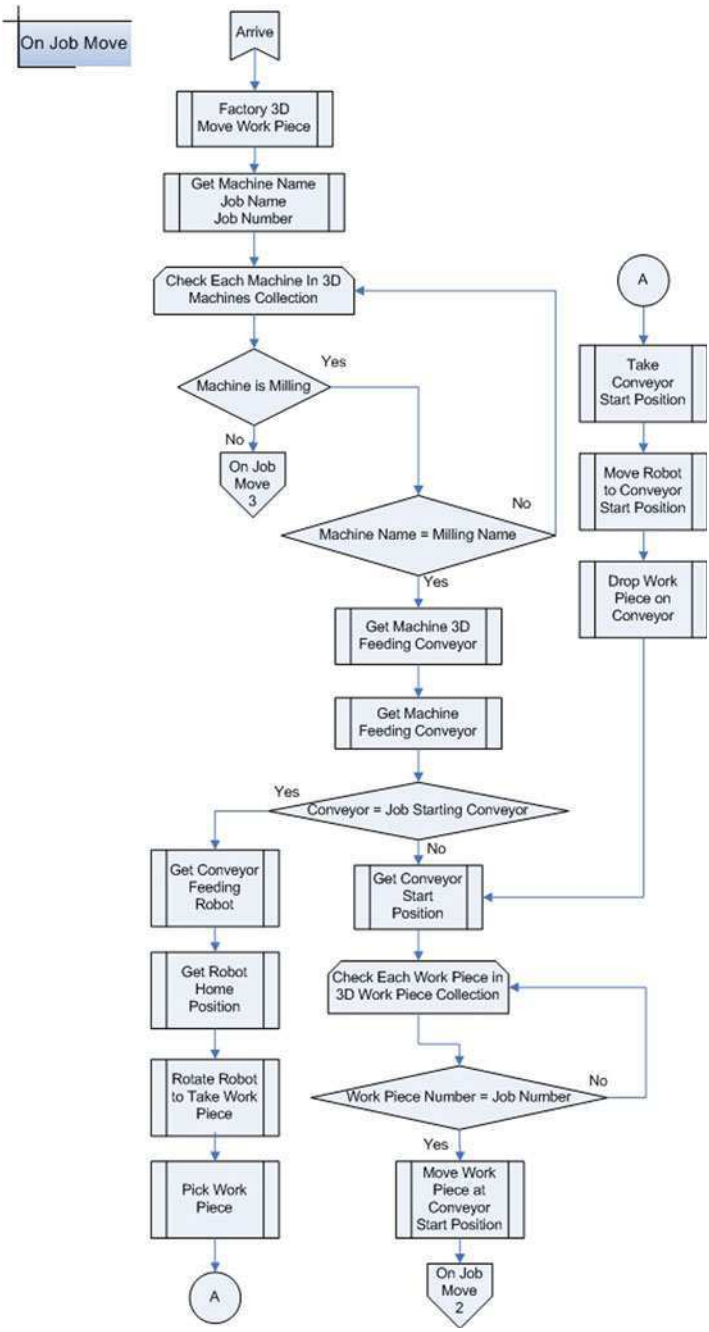


Fig. 11.82 (continued)

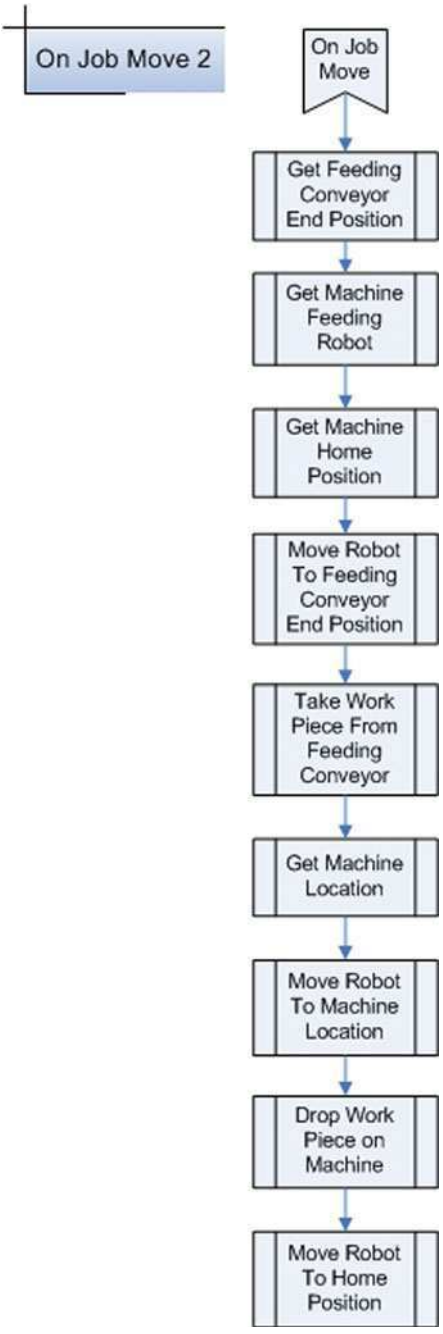


Fig. 11.82 (continued)

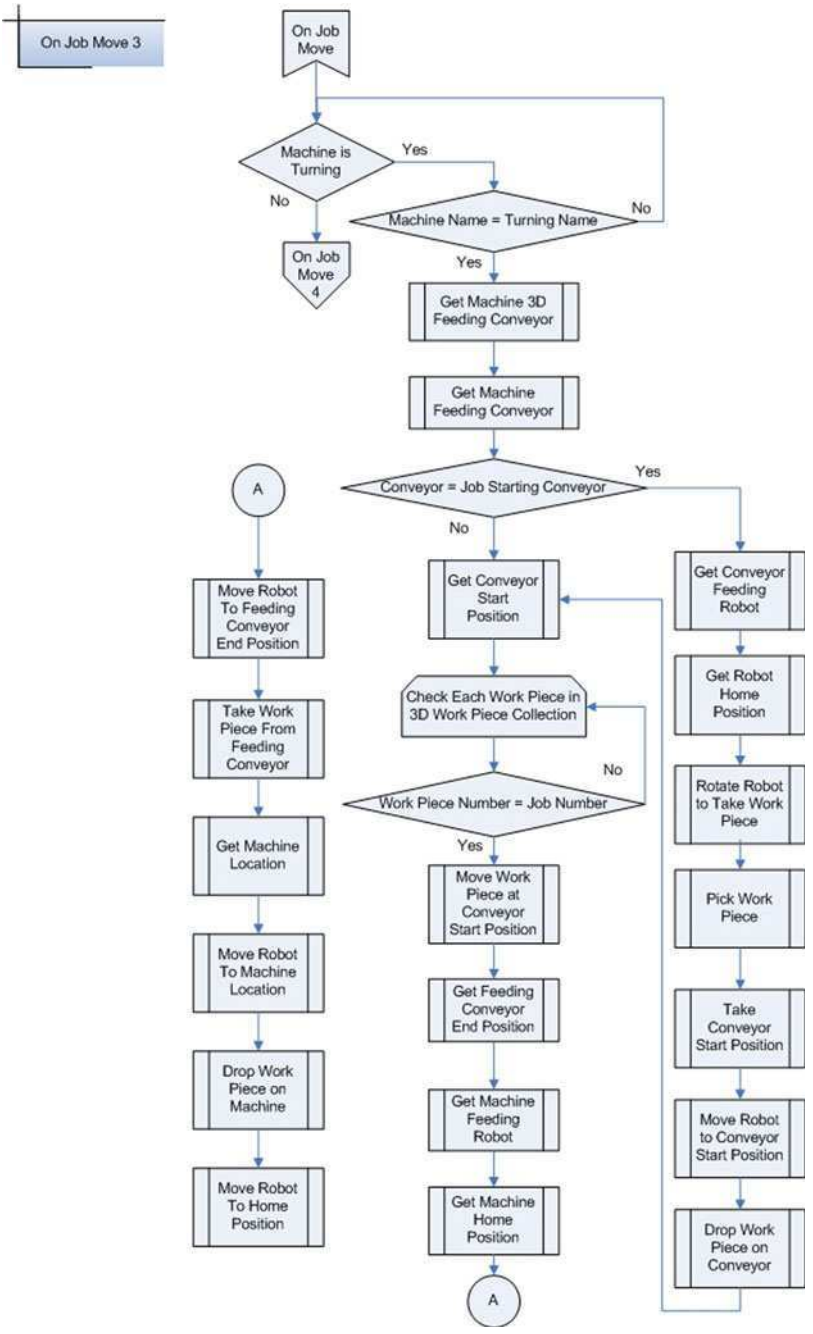


Fig. 11.82 (continued)

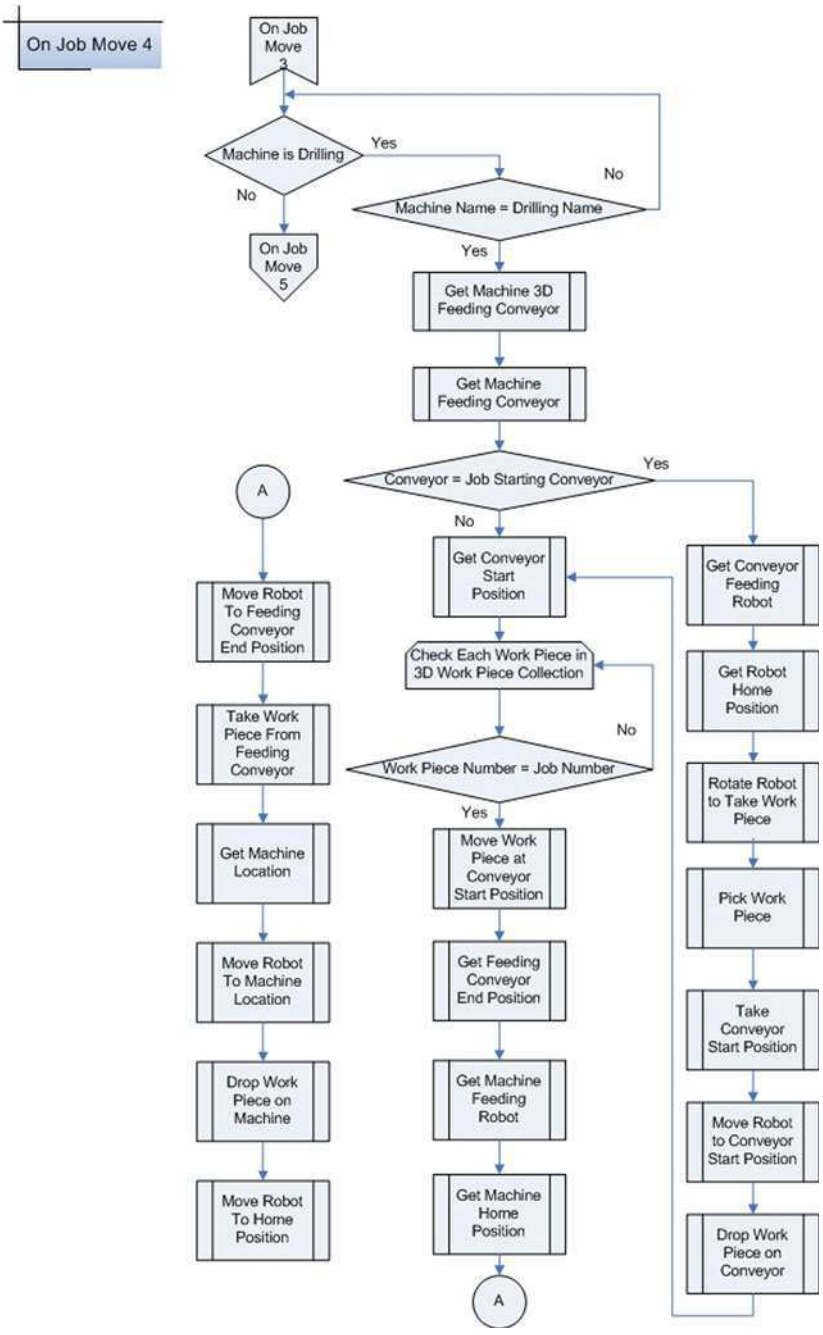


Fig. 11.82 (continued)

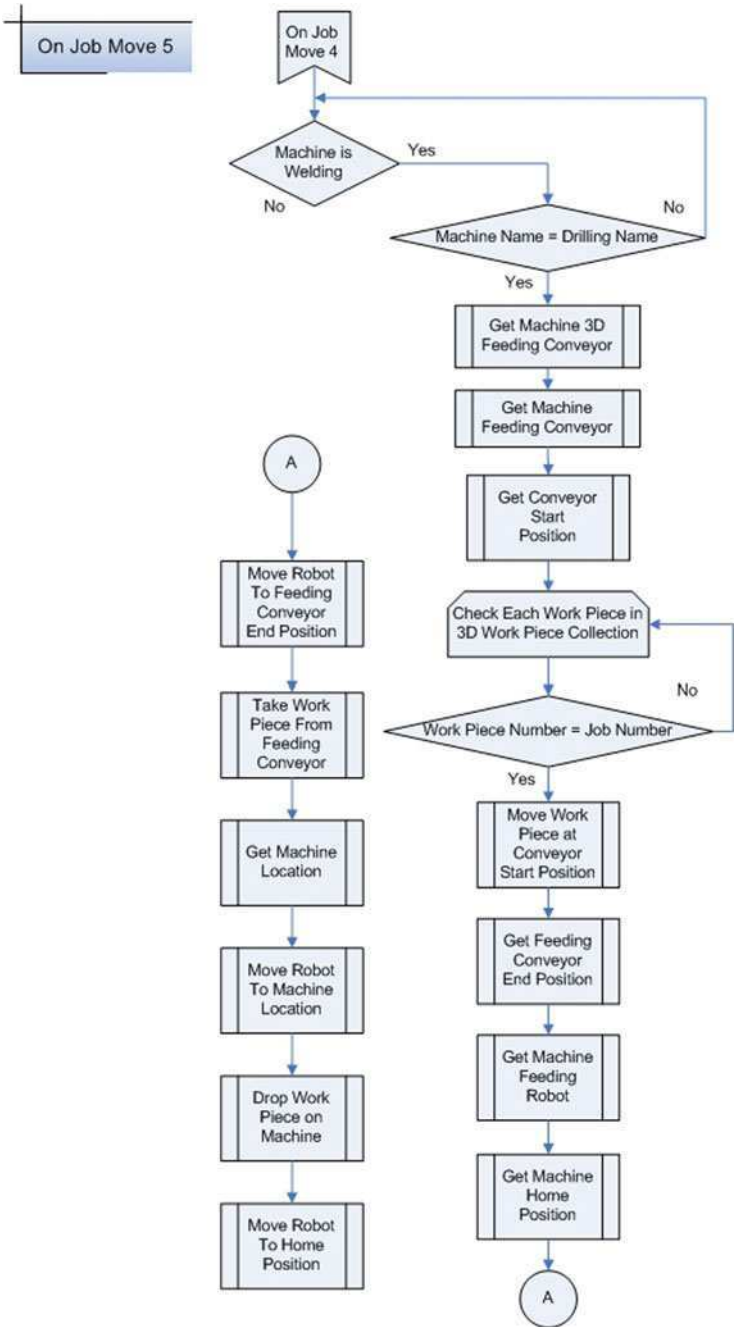


Fig. 11.82 (continued)

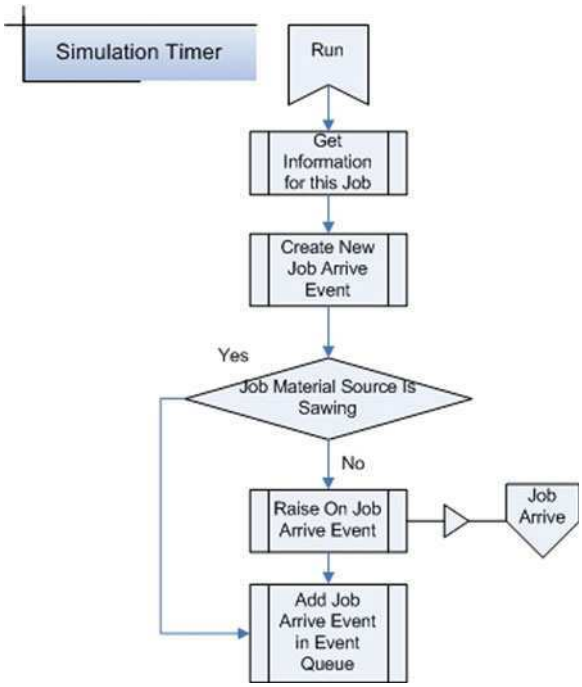


Fig. 11.82 (continued)

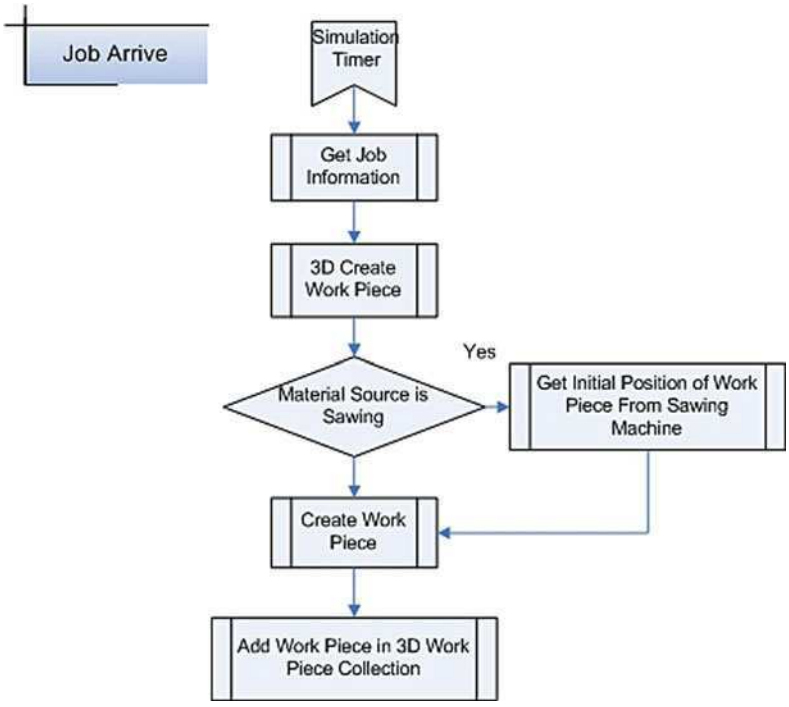


Fig. 11.82 (continued)

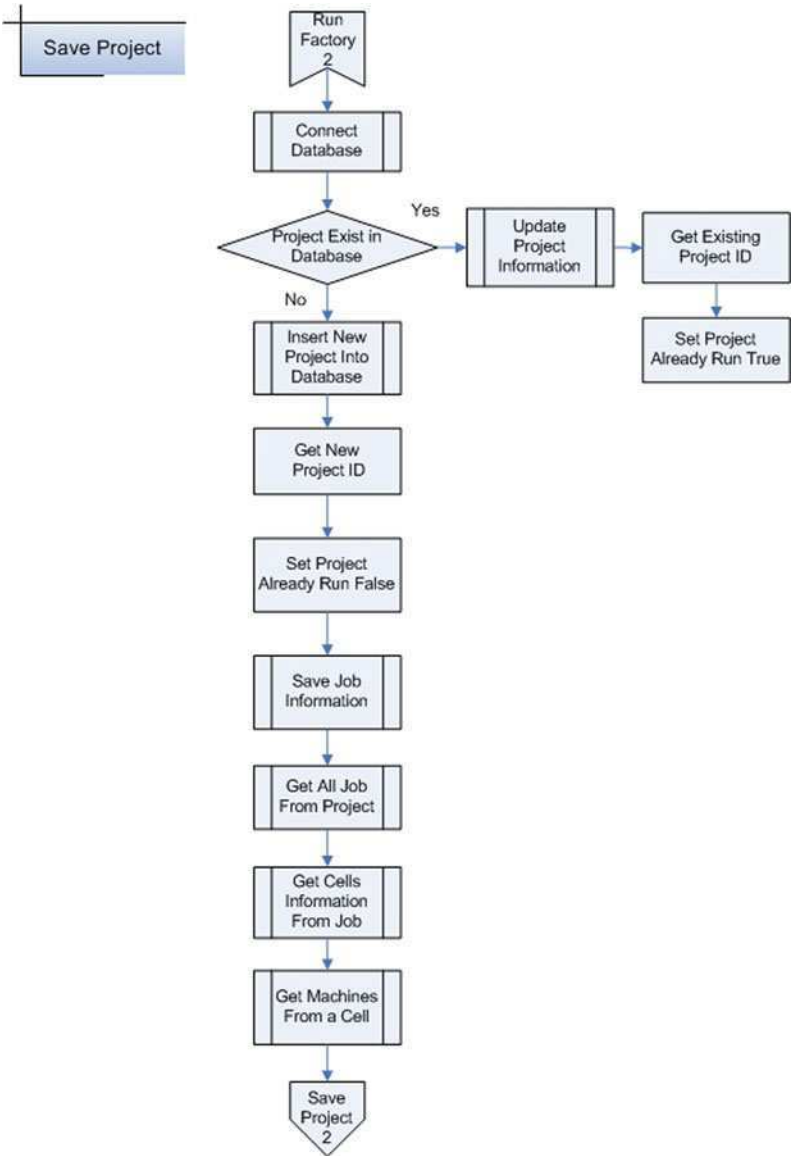


Fig. 11.82 (continued)

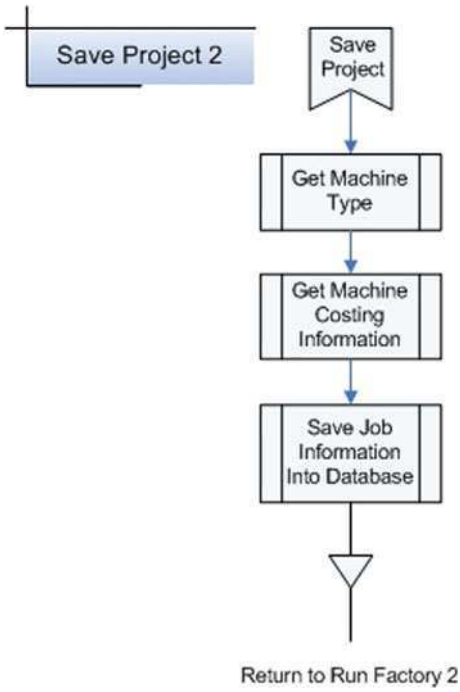


Fig. 11.82 (continued)

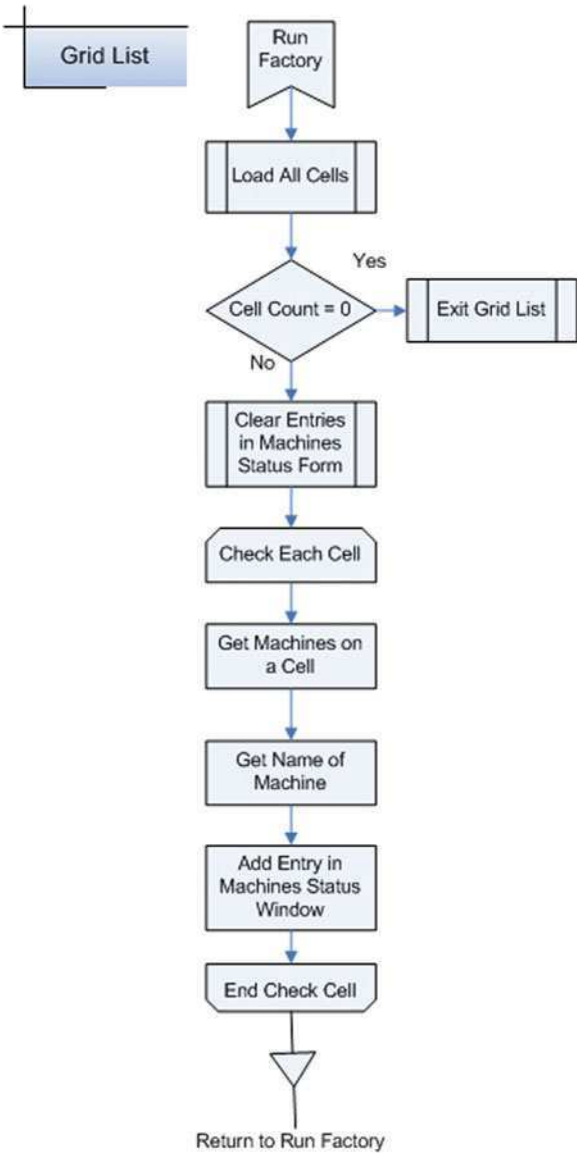


Fig. 11.82 (continued)

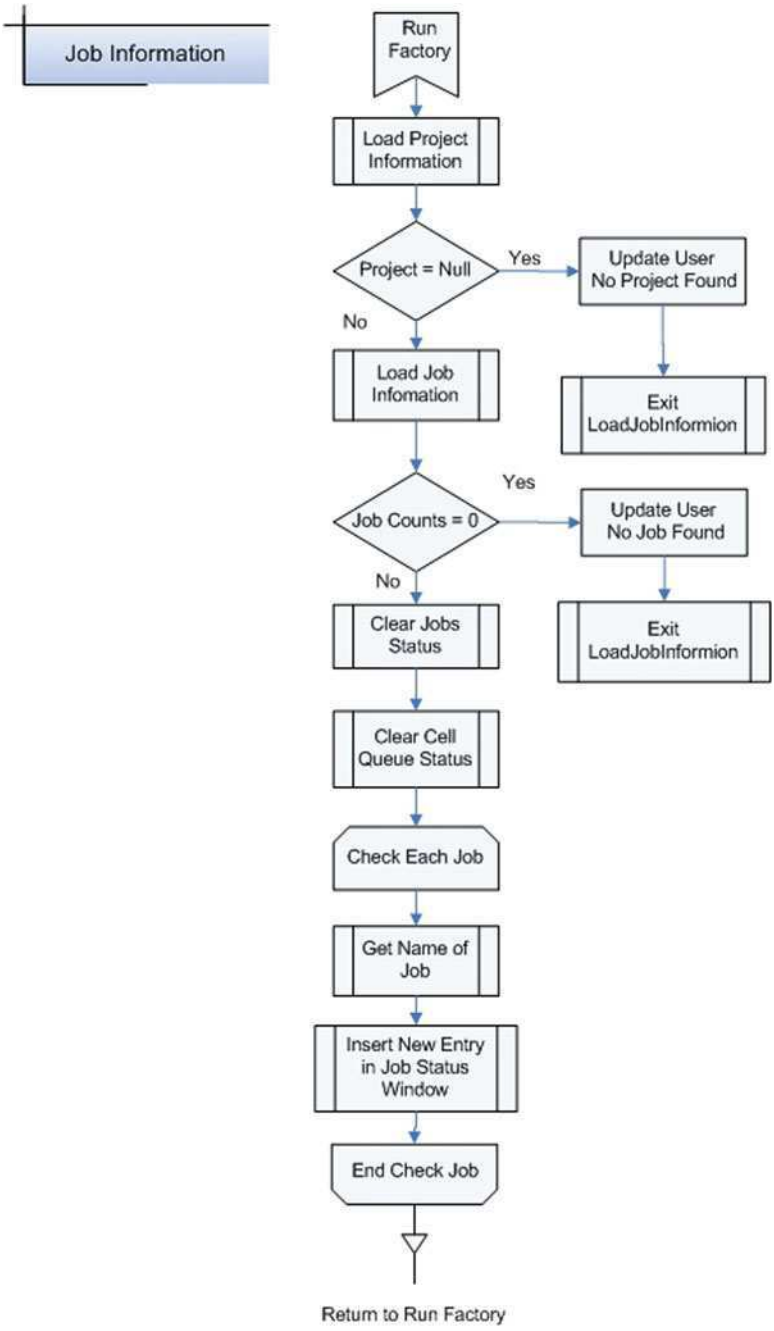


Fig. 11.82 (continued)

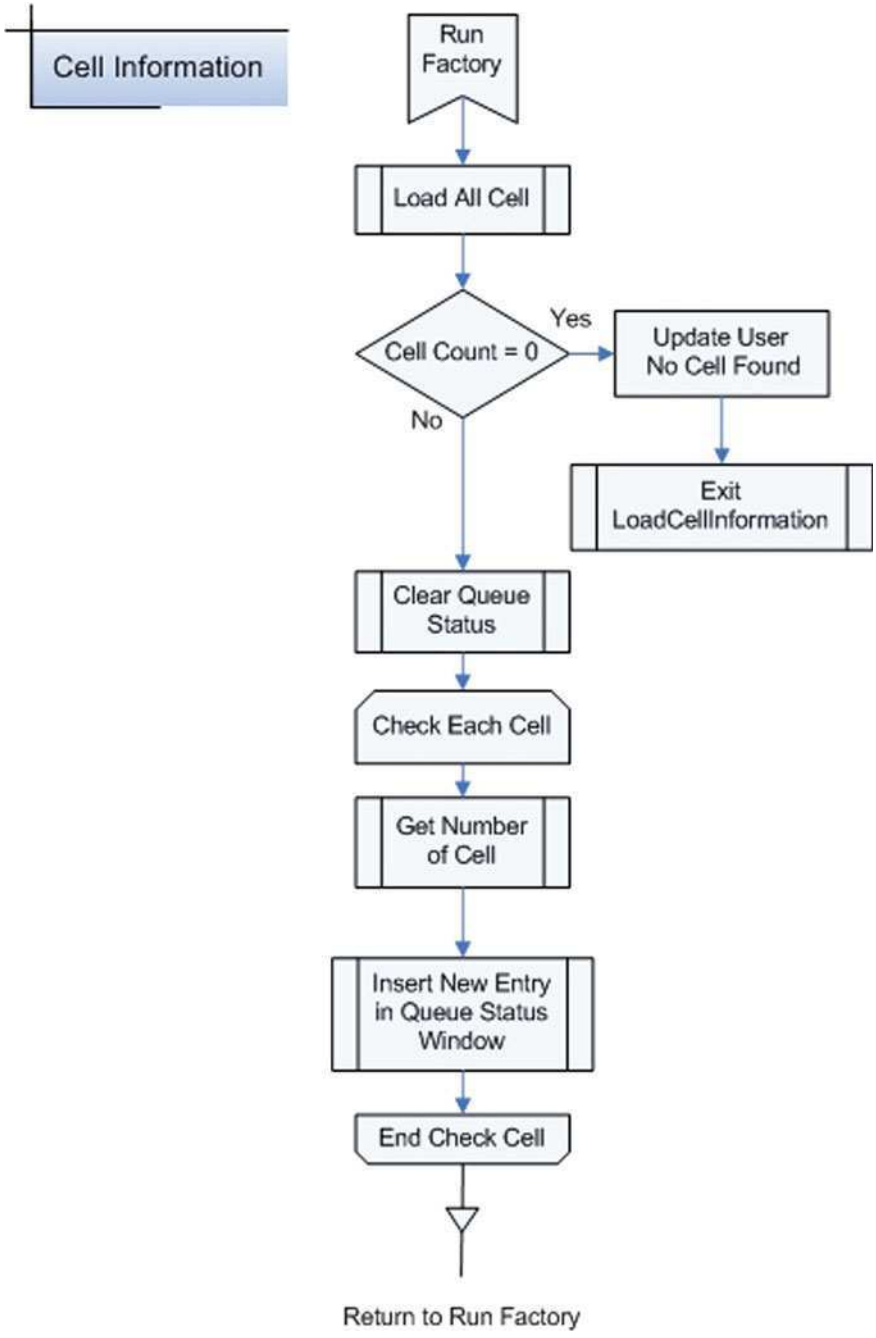


Fig. 11.82 (continued)

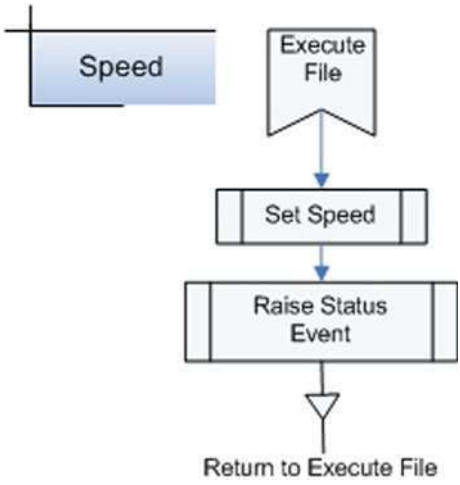


Fig. 11.82 (continued)

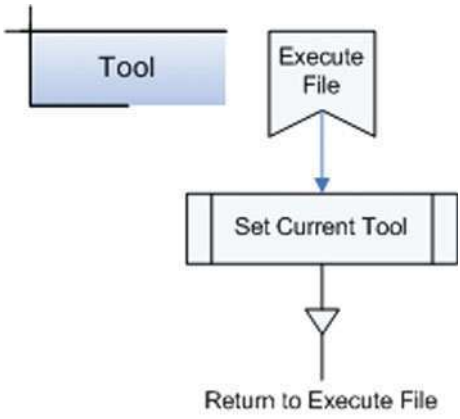


Fig. 11.82 (continued)

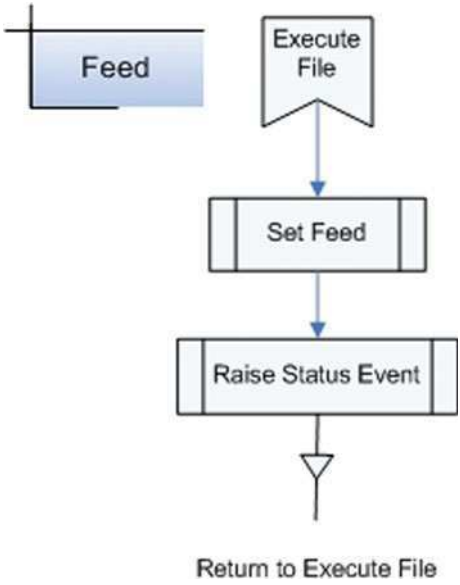


Fig. 11.82 (continued)

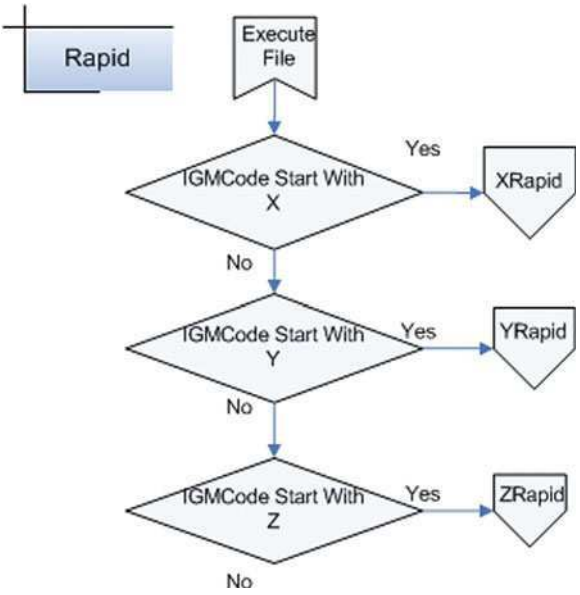


Fig. 11.82 (continued)

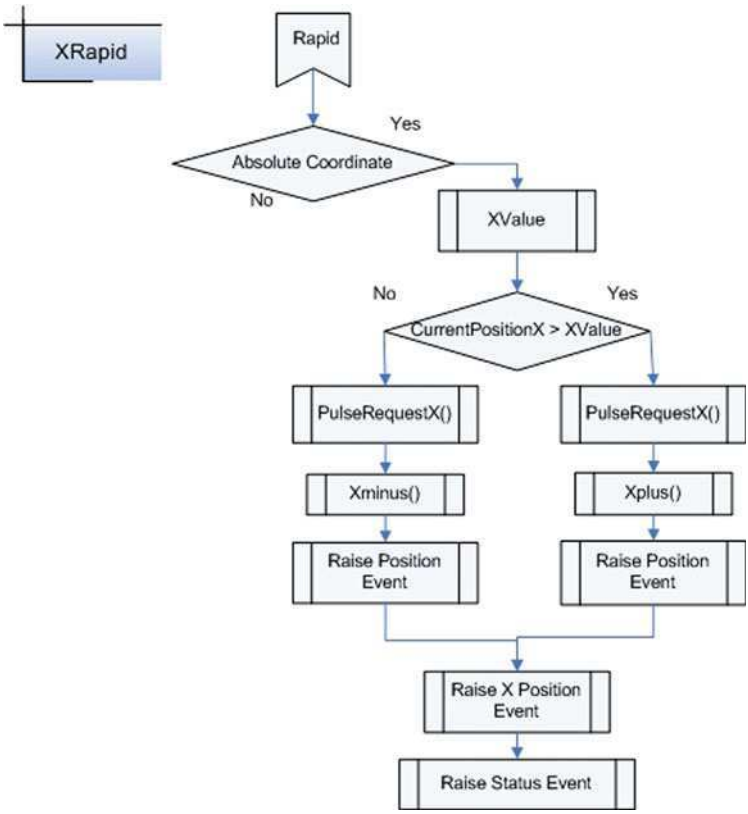


Fig. 11.82 (continued)

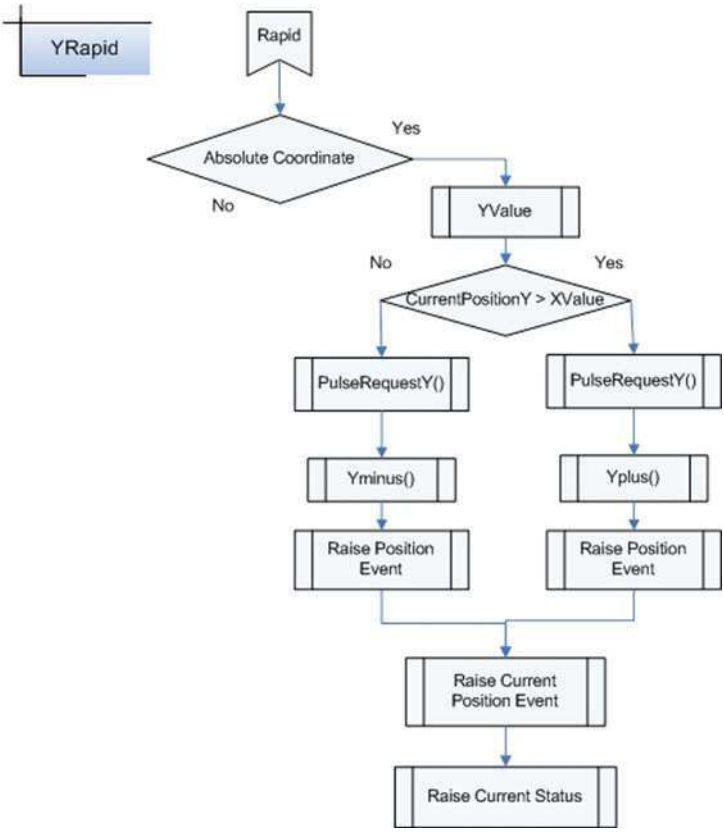


Fig. 11.82 (continued)

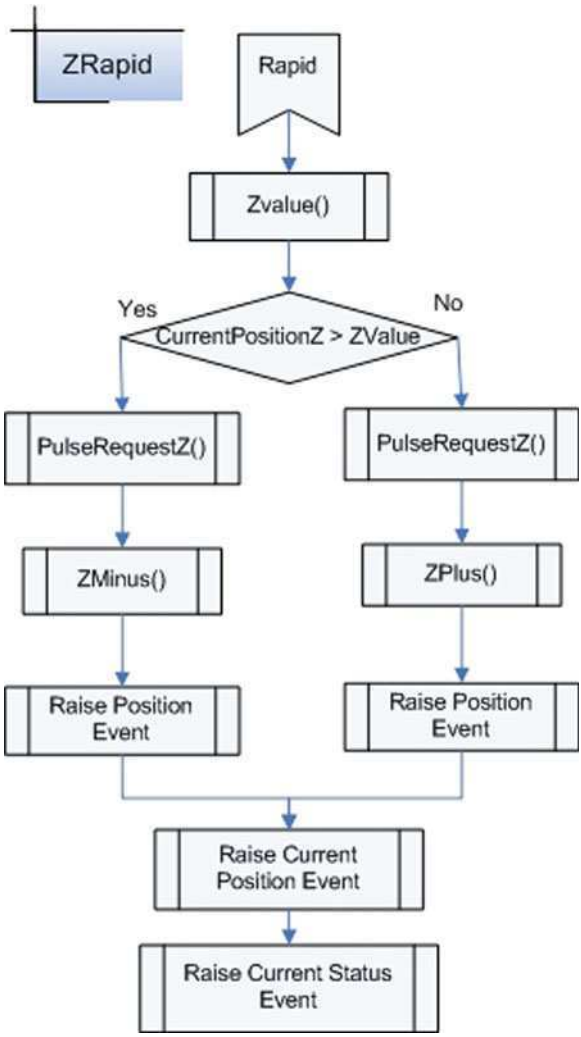


Fig. 11.82 (continued)

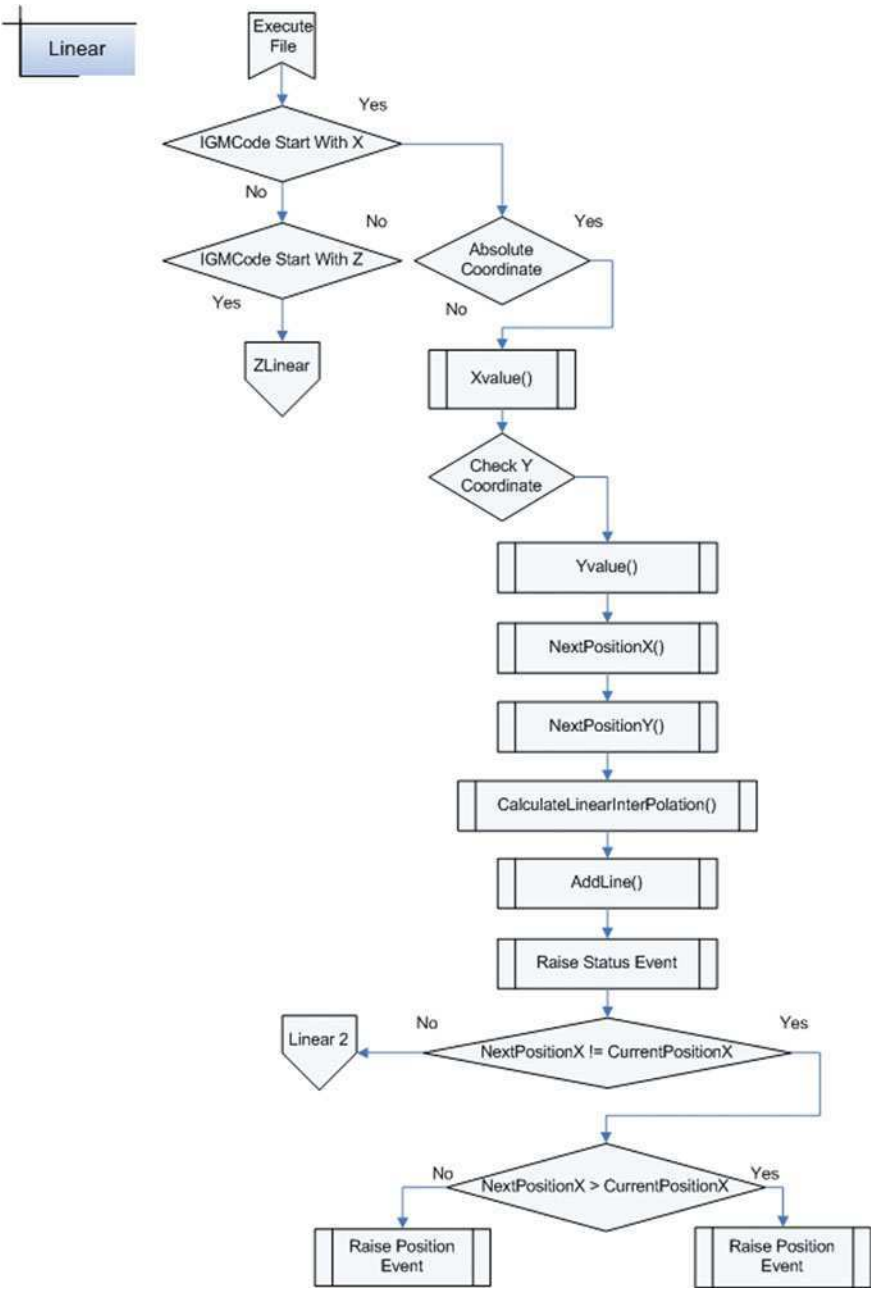


Fig. 11.82 (continued)

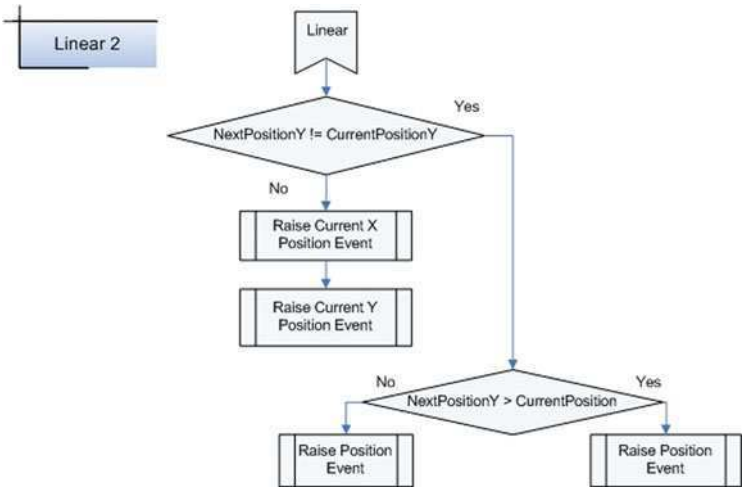


Fig. 11.82 (continued)

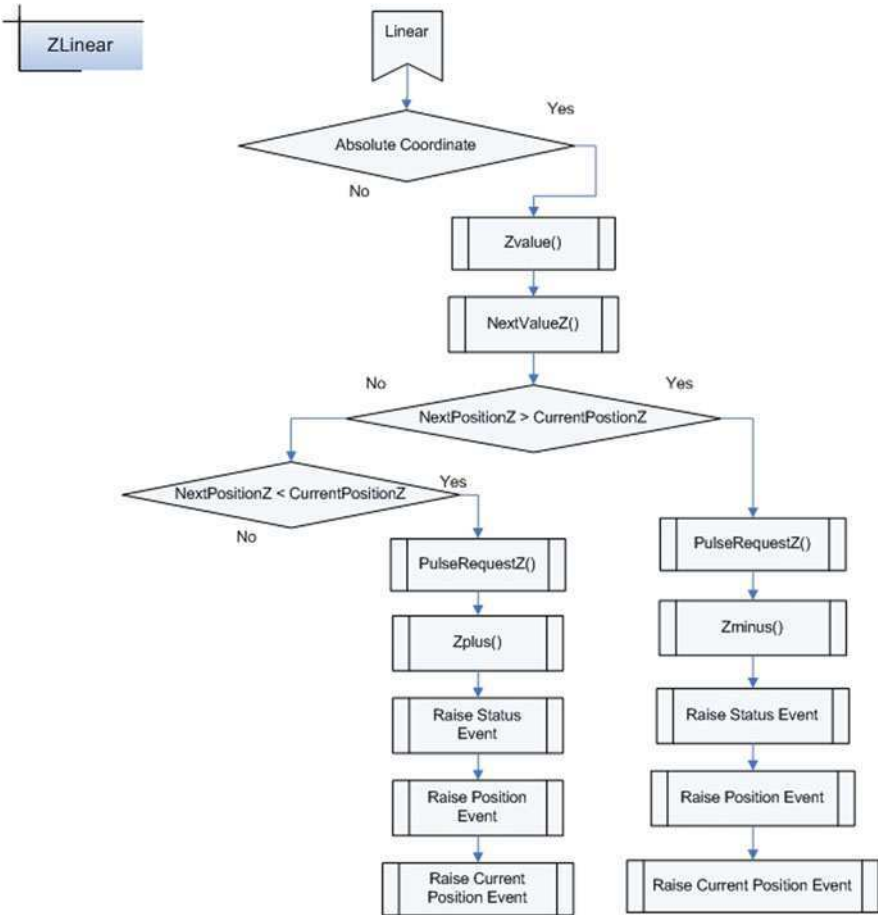


Fig. 11.82 (continued)

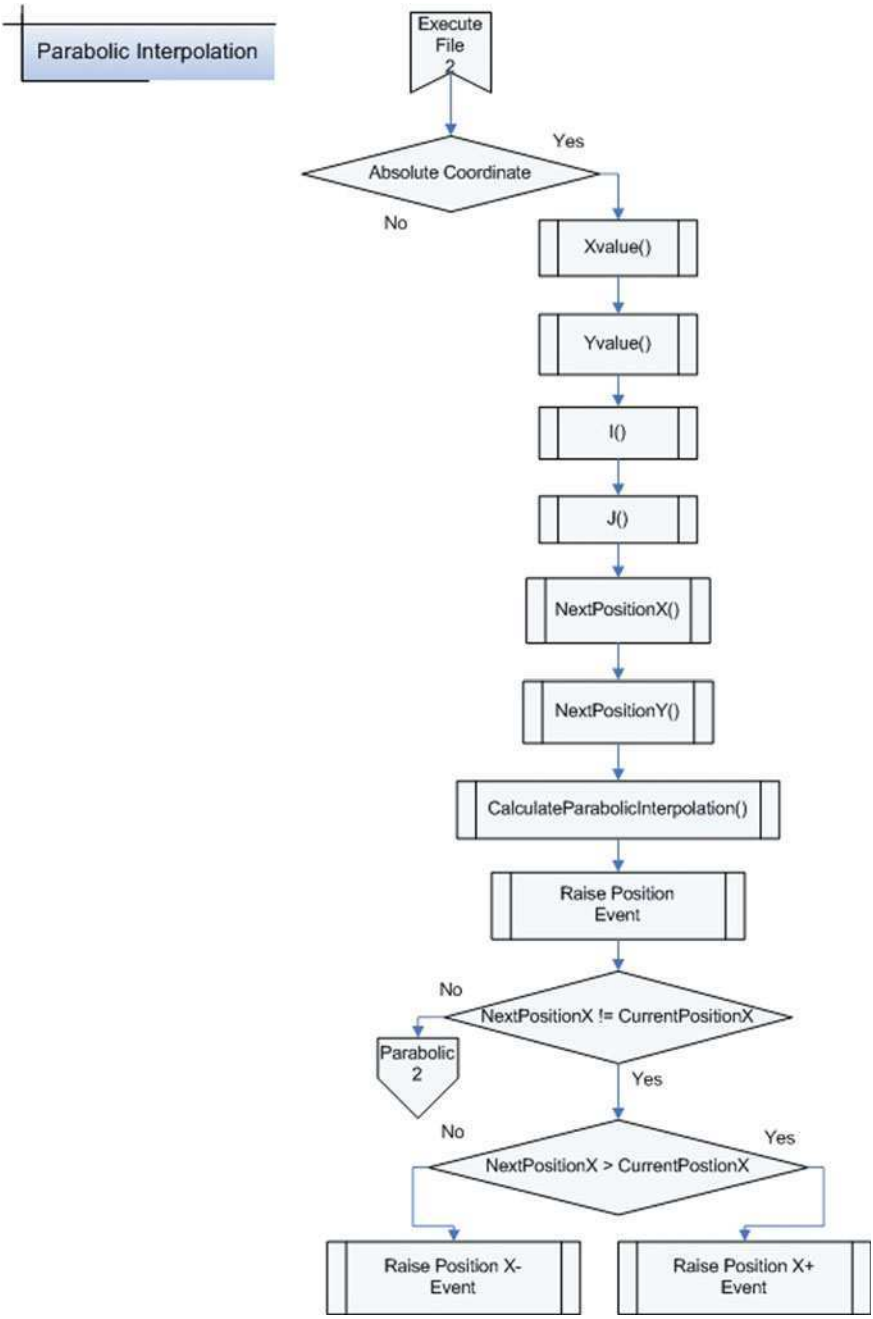


Fig. 11.82 (continued)

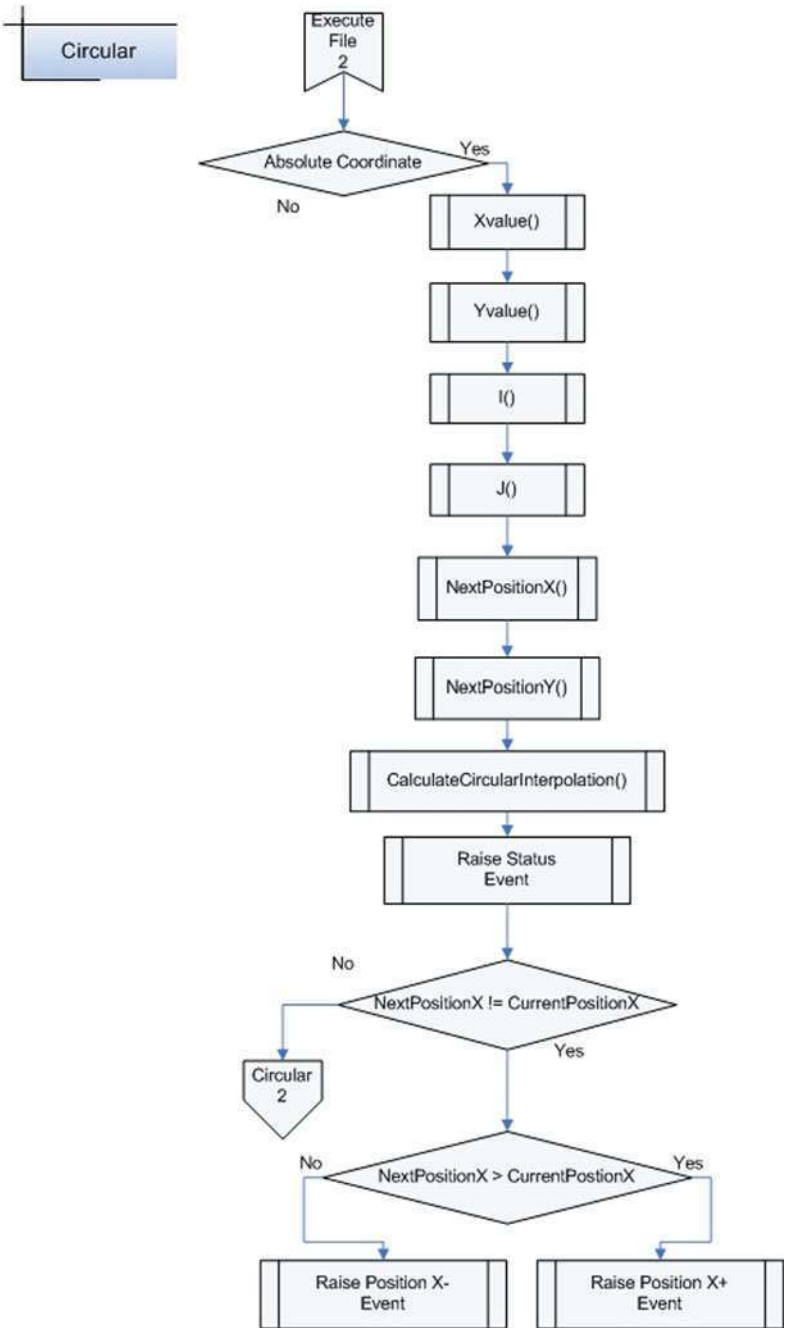


Fig. 11.82 (continued)

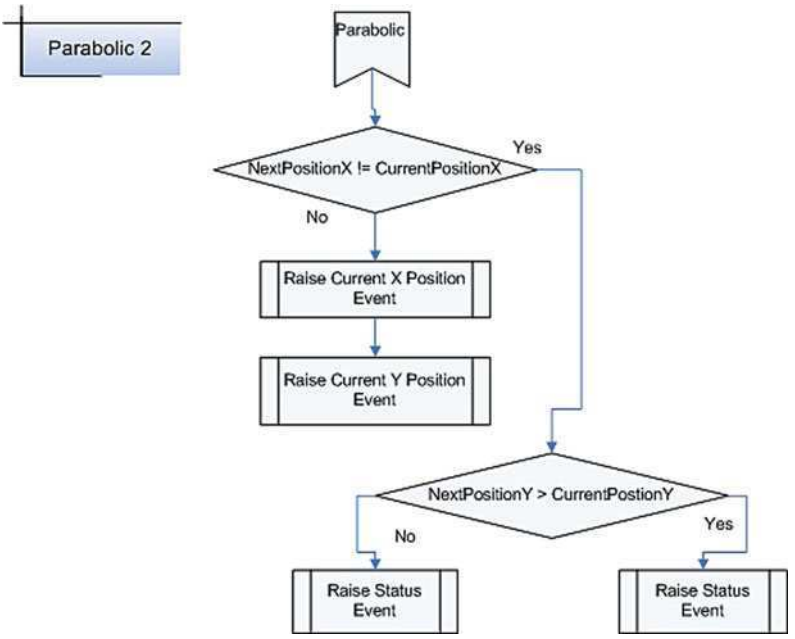


Fig. 11.82 (continued)

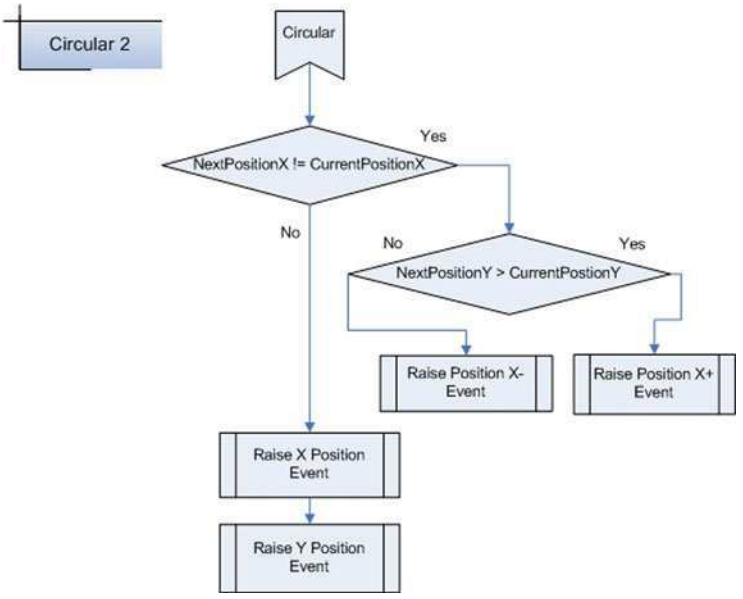


Fig. 11.82 (continued)

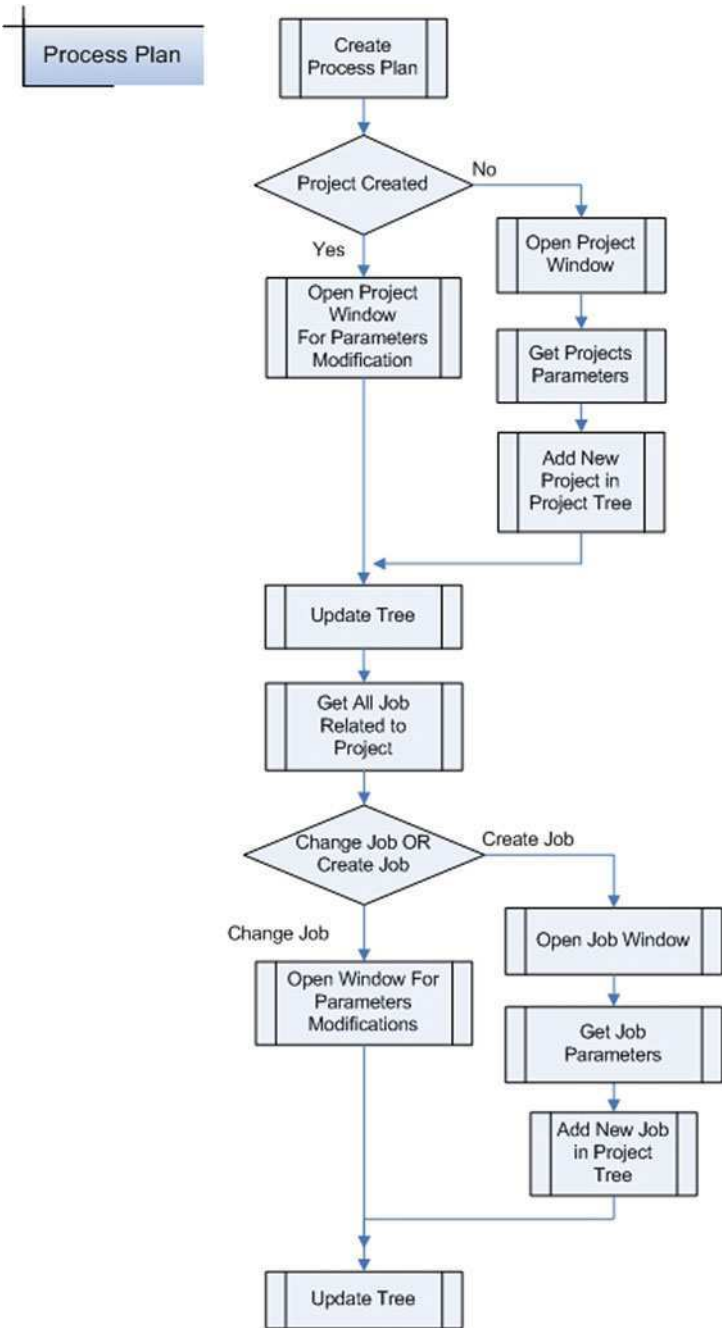


Fig. 11.83 Virtual Factory Process Plan Flow Chart

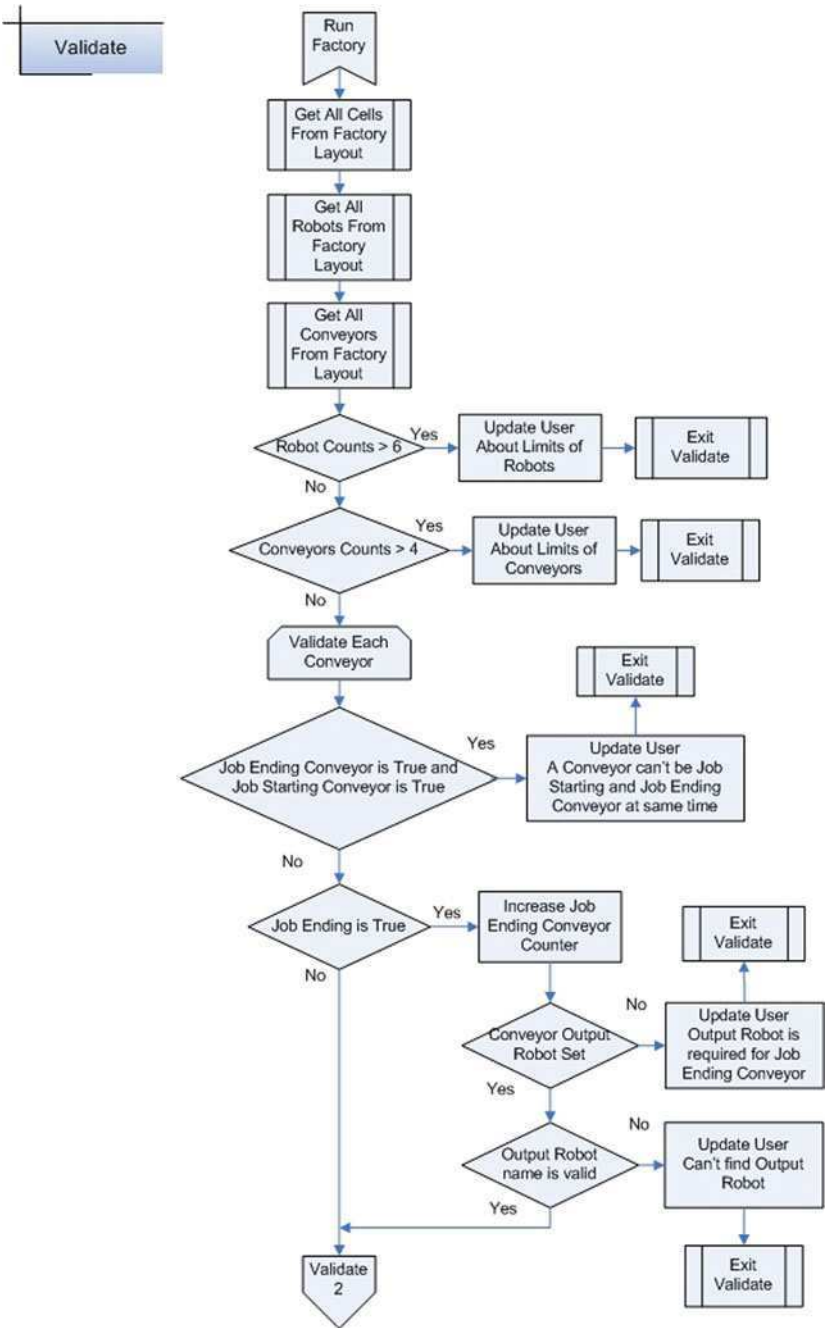


Fig. 11.84 Virtual Factory Validate Setup Flow Chart

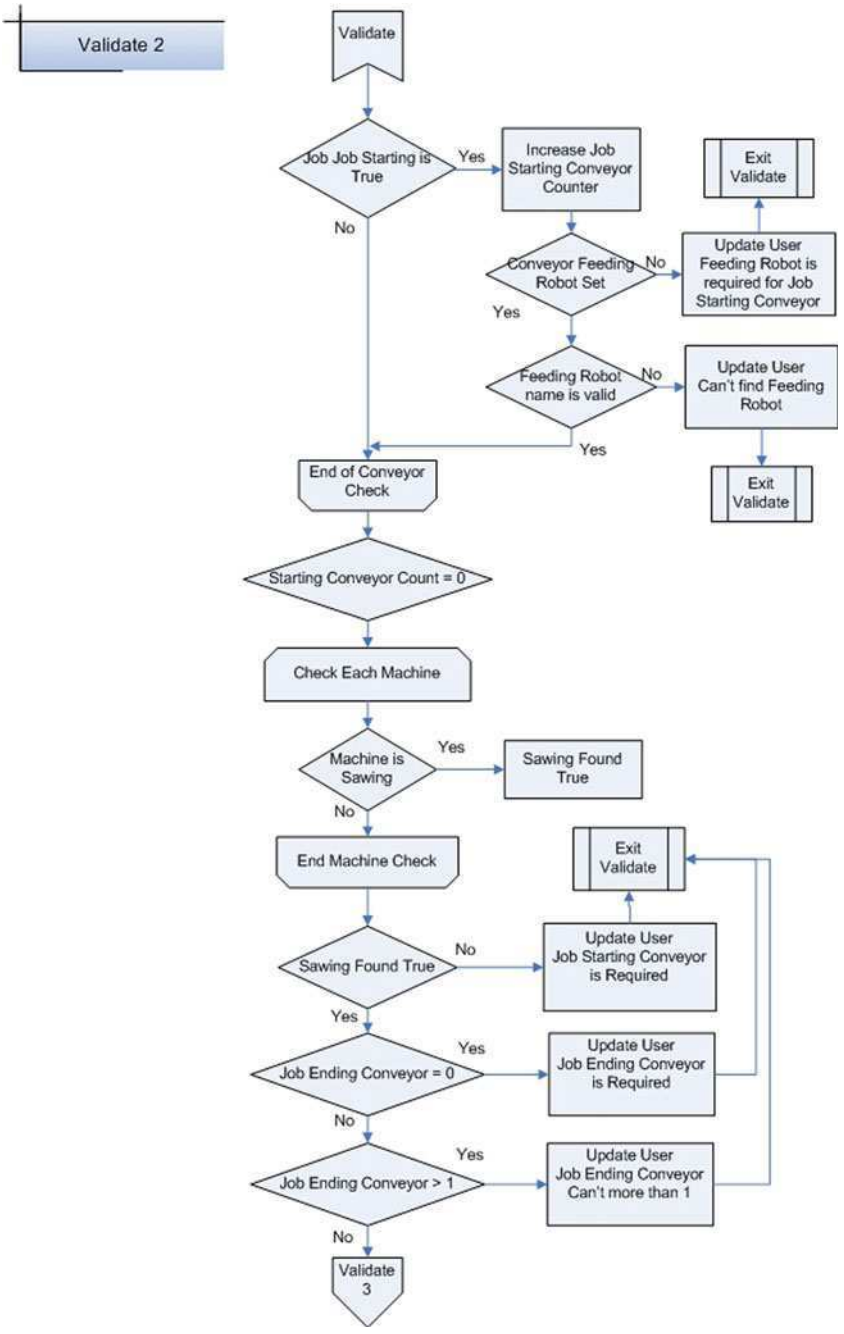


Fig. 11.84 (continued)

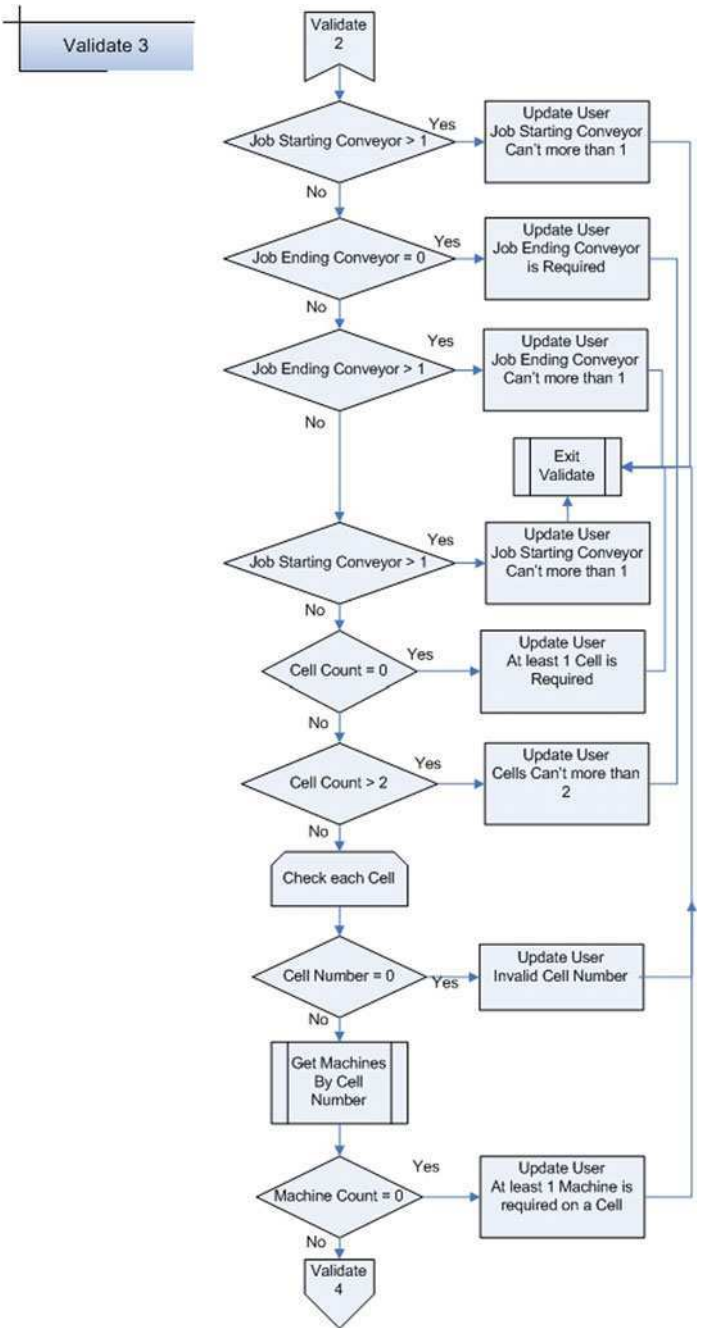


Fig. 11.84 (continued)

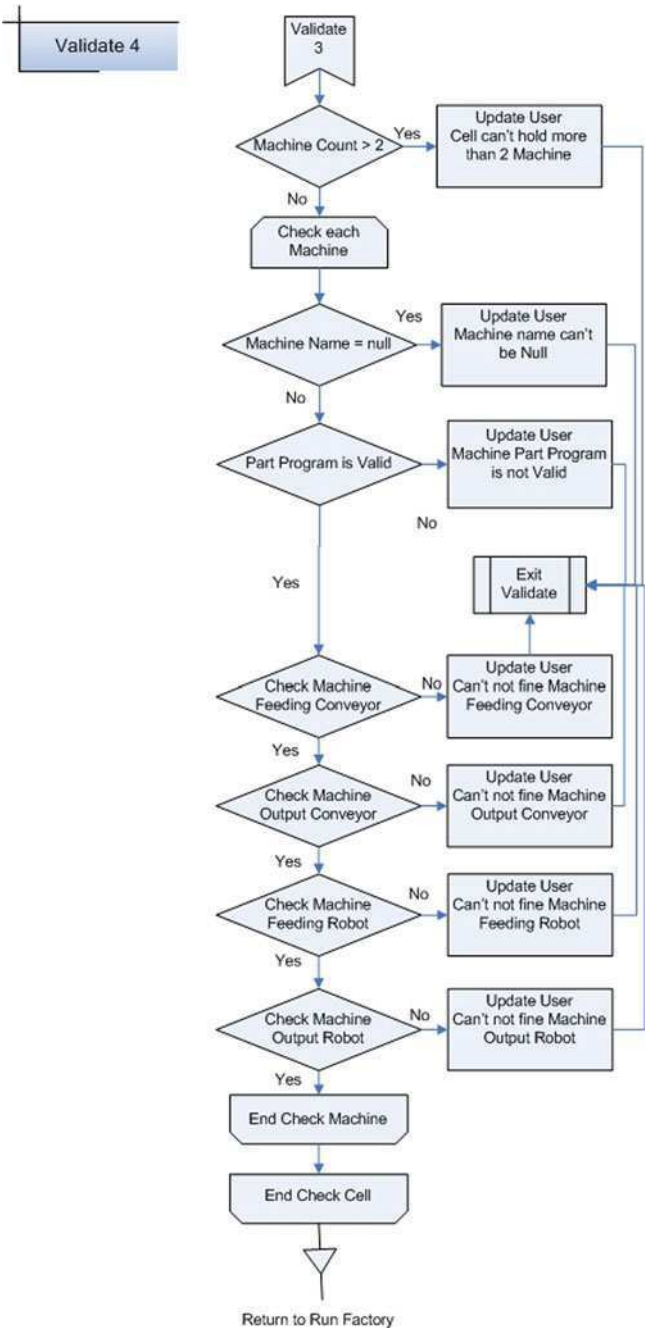


Fig. 11.84 (continued)

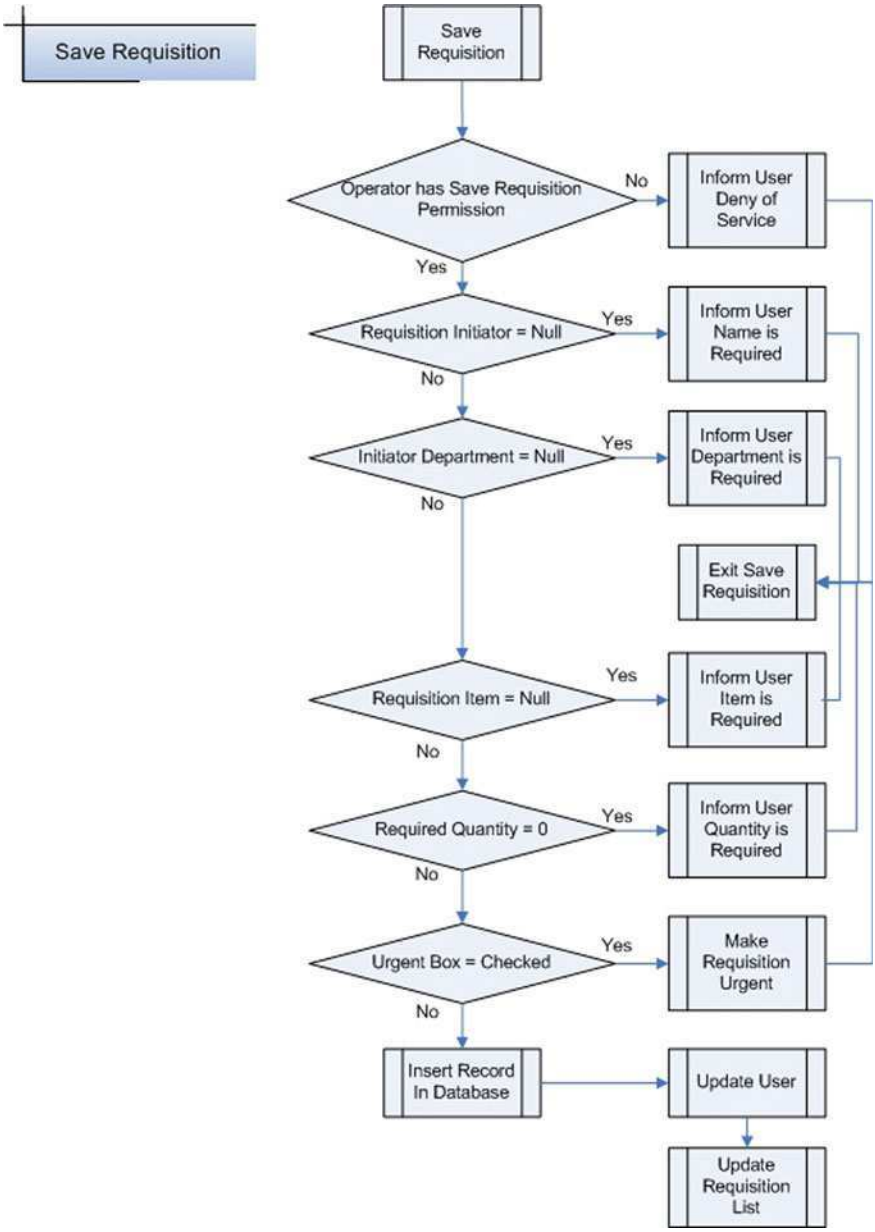


Fig. 11.85 Virtual Factory Save Requisition Flow Chart

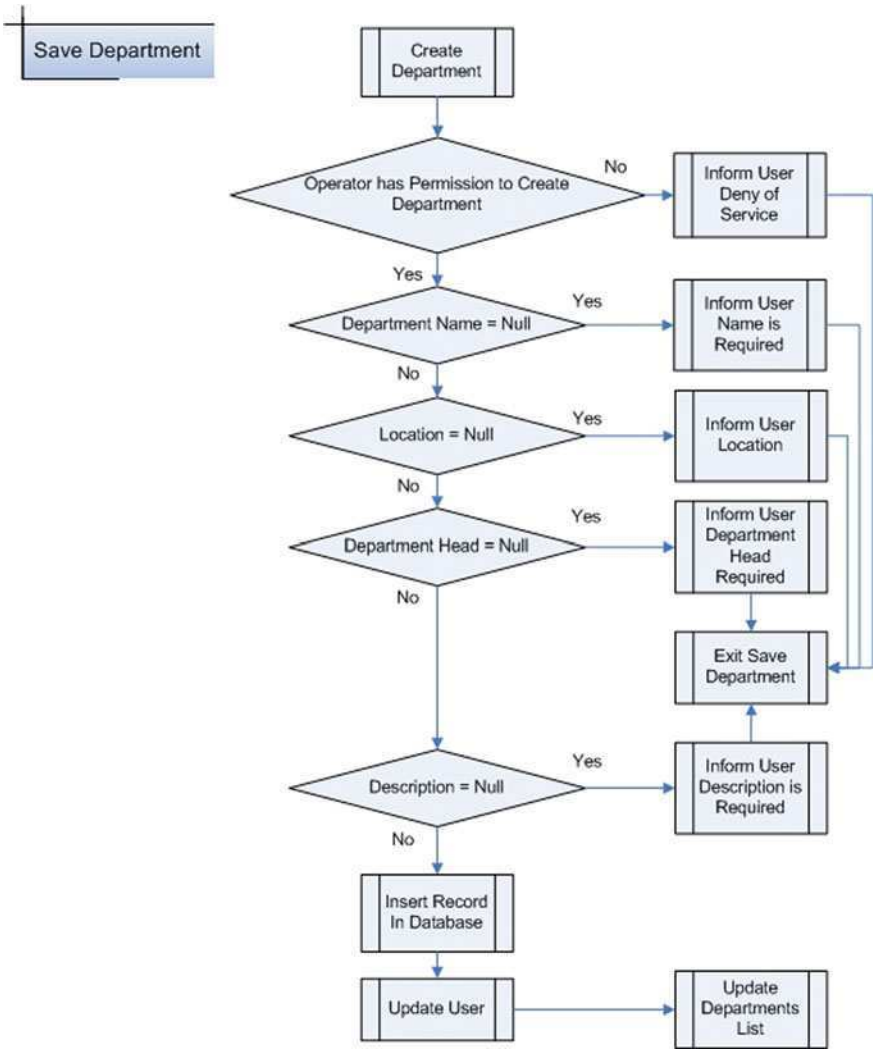


Fig. 11.86 Virtual Factory Save Department Flow Chart

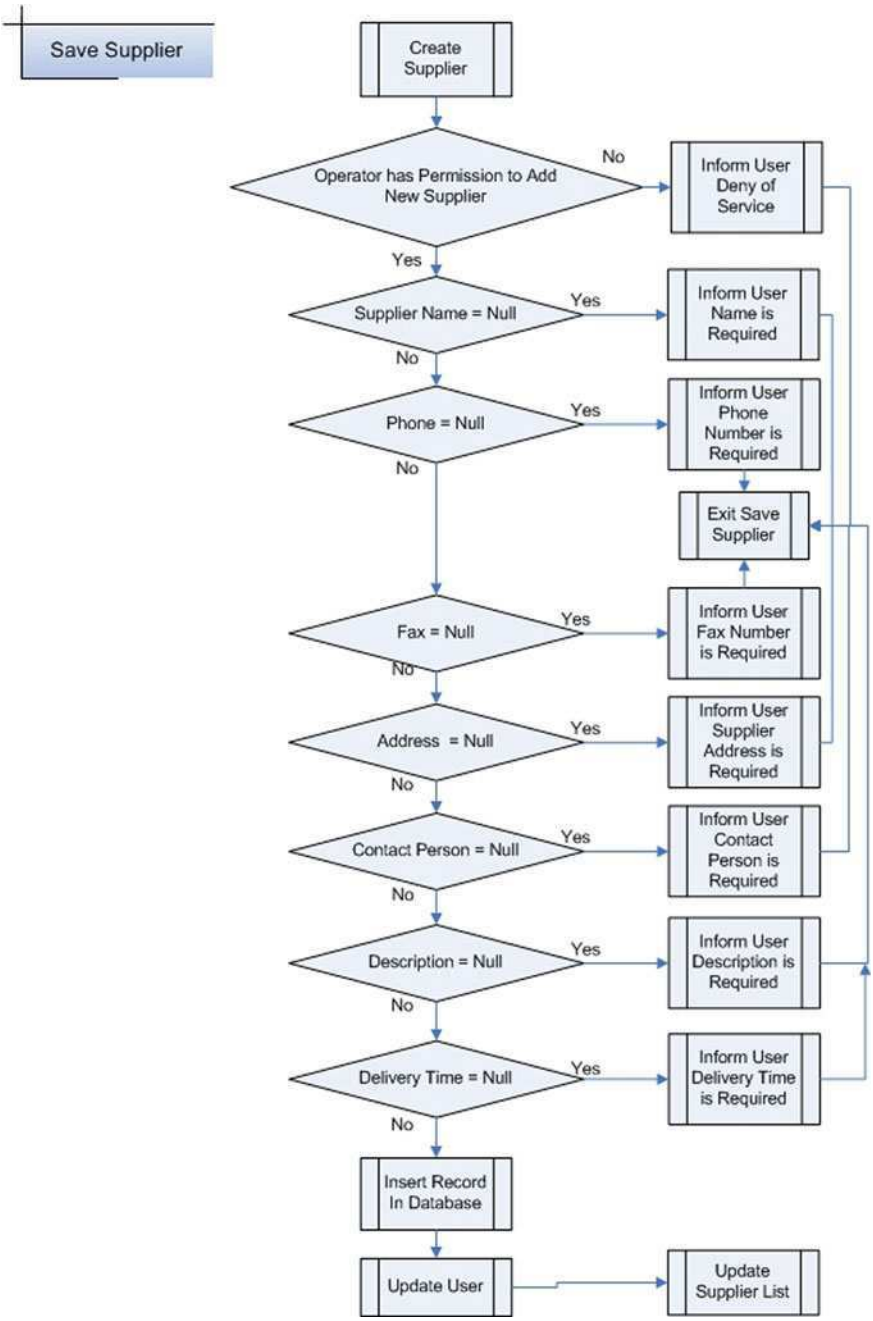


Fig. 11.87 Virtual Factory Validate Add Supplier Flow Chart

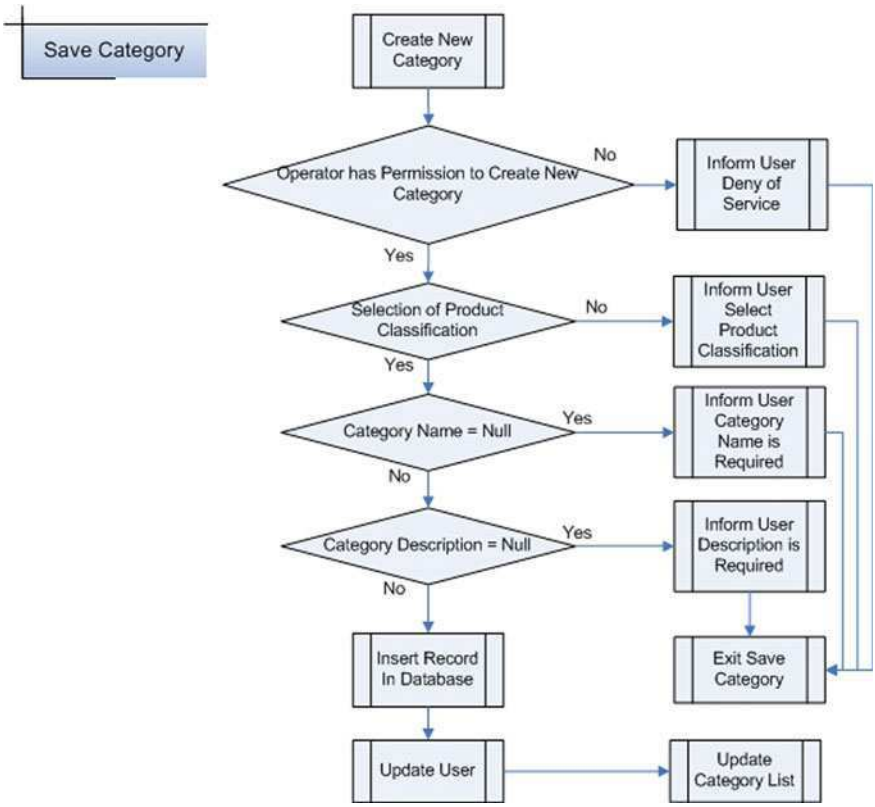


Fig. 11.88 Virtual Factory Add Category Flow Chart

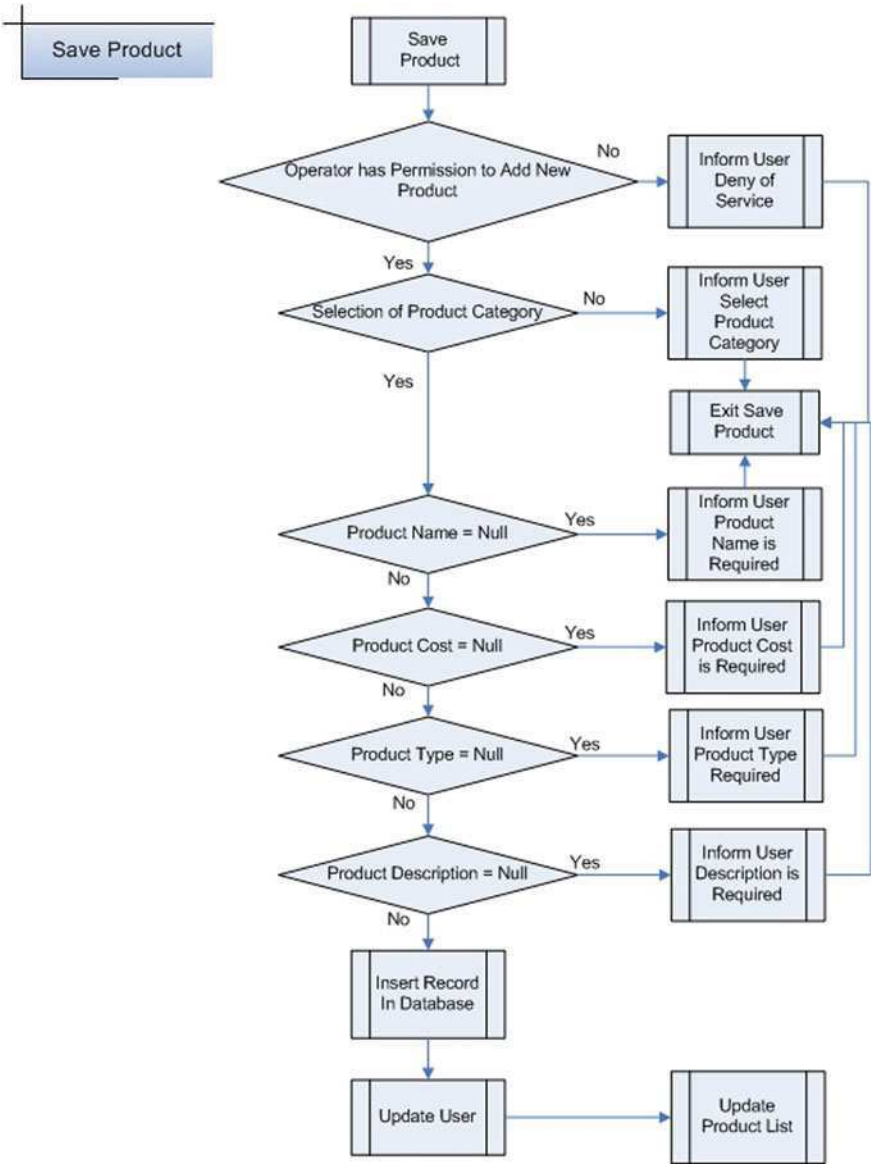


Fig. 11.89 Virtual Factory Save Product Flow Chart

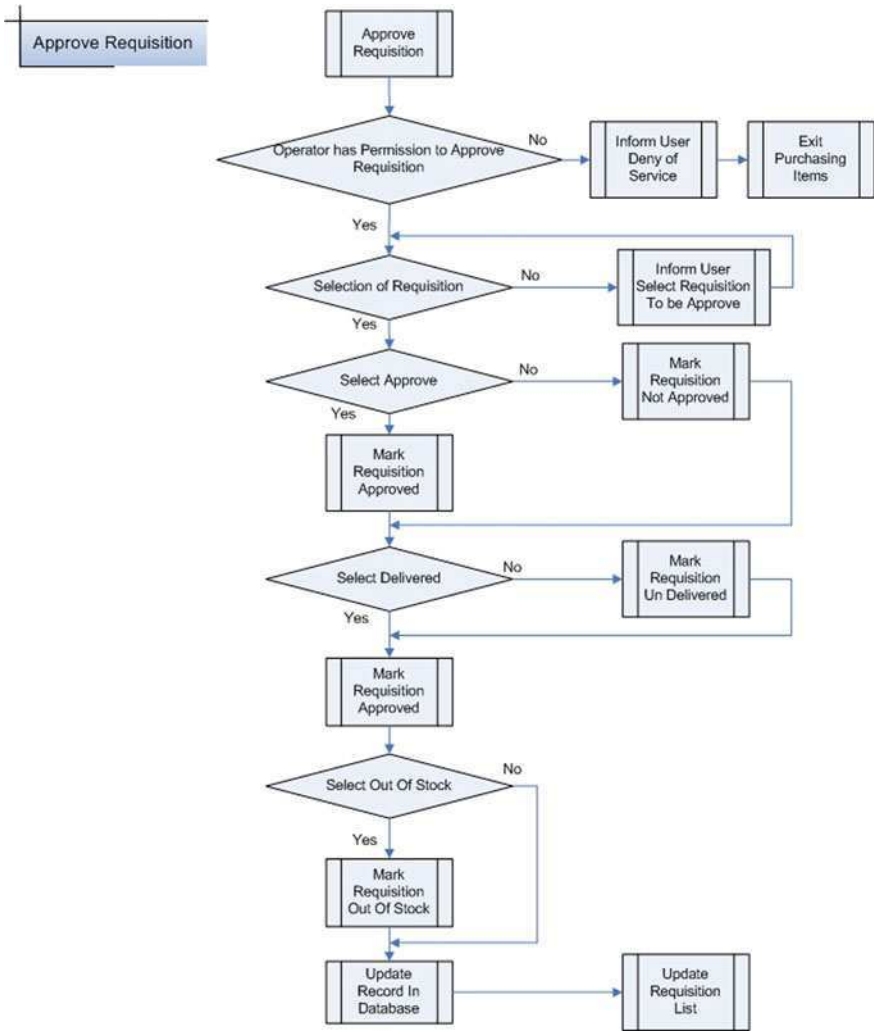


Fig. 11.90 Virtual Factory Approve Requisition

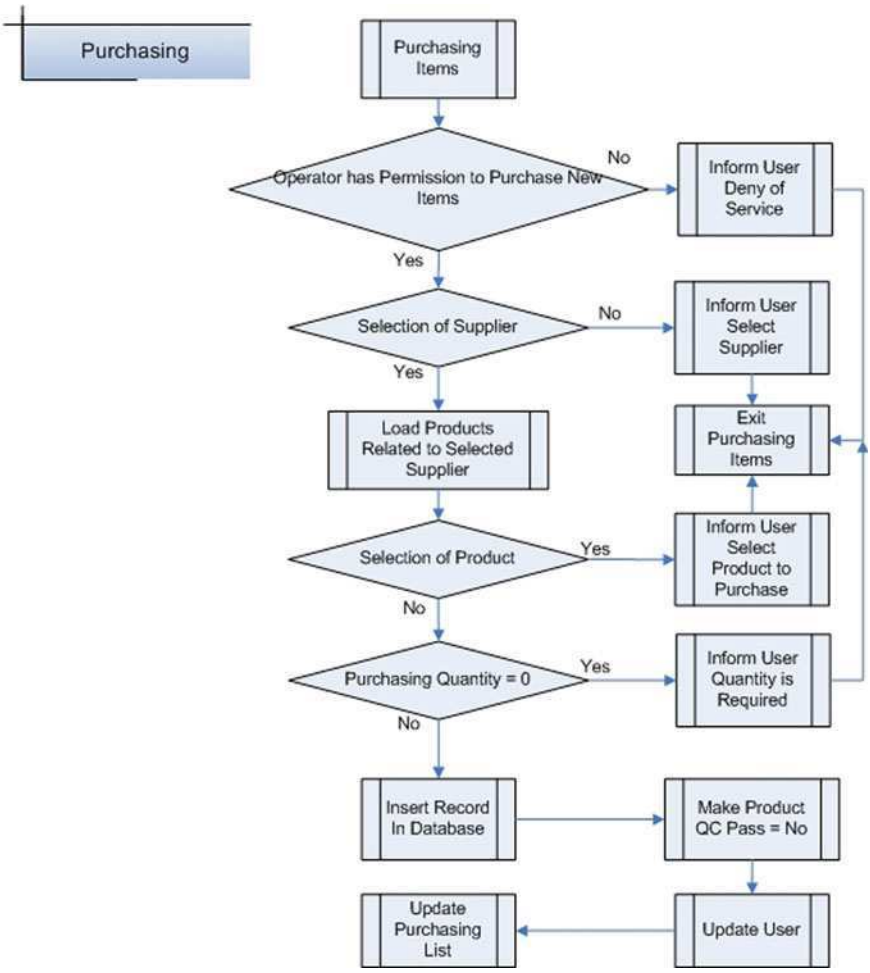


Fig. 11.91 Virtual Factory Purchasing Flow Chart

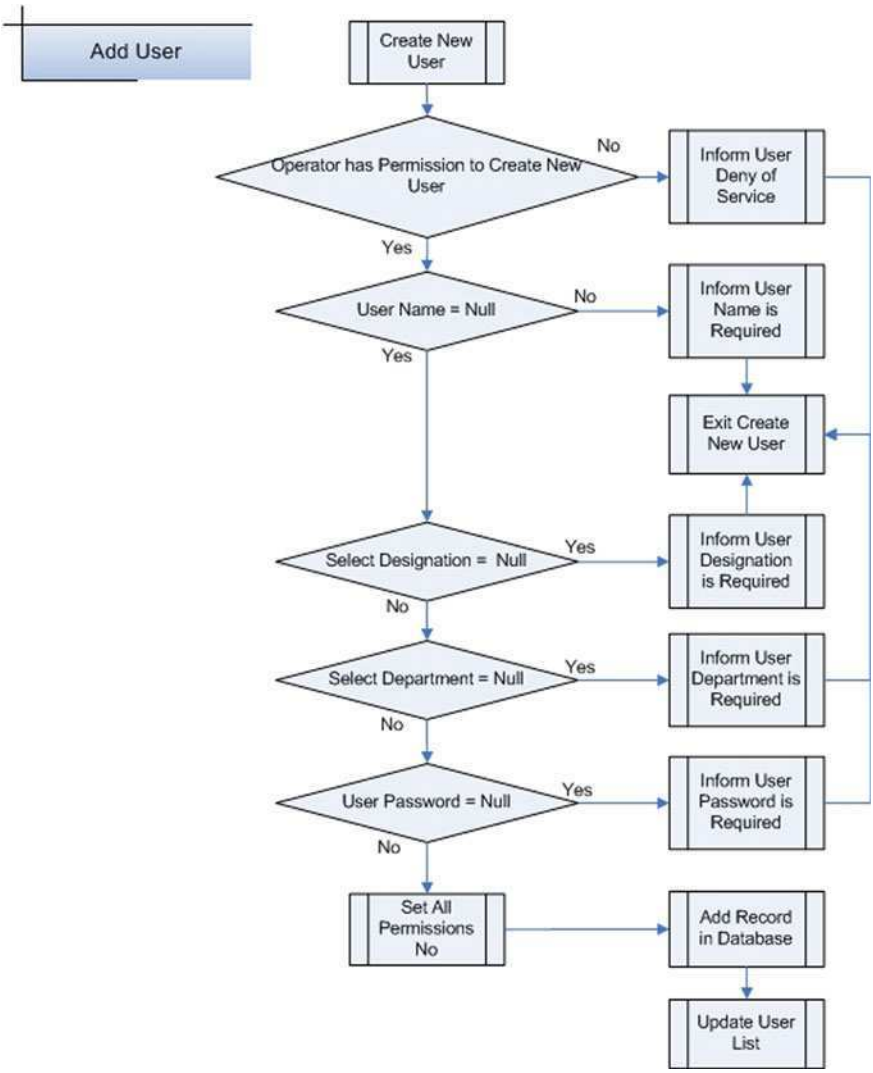


Fig. 11.92 Virtual Factory Add User Flow Chart

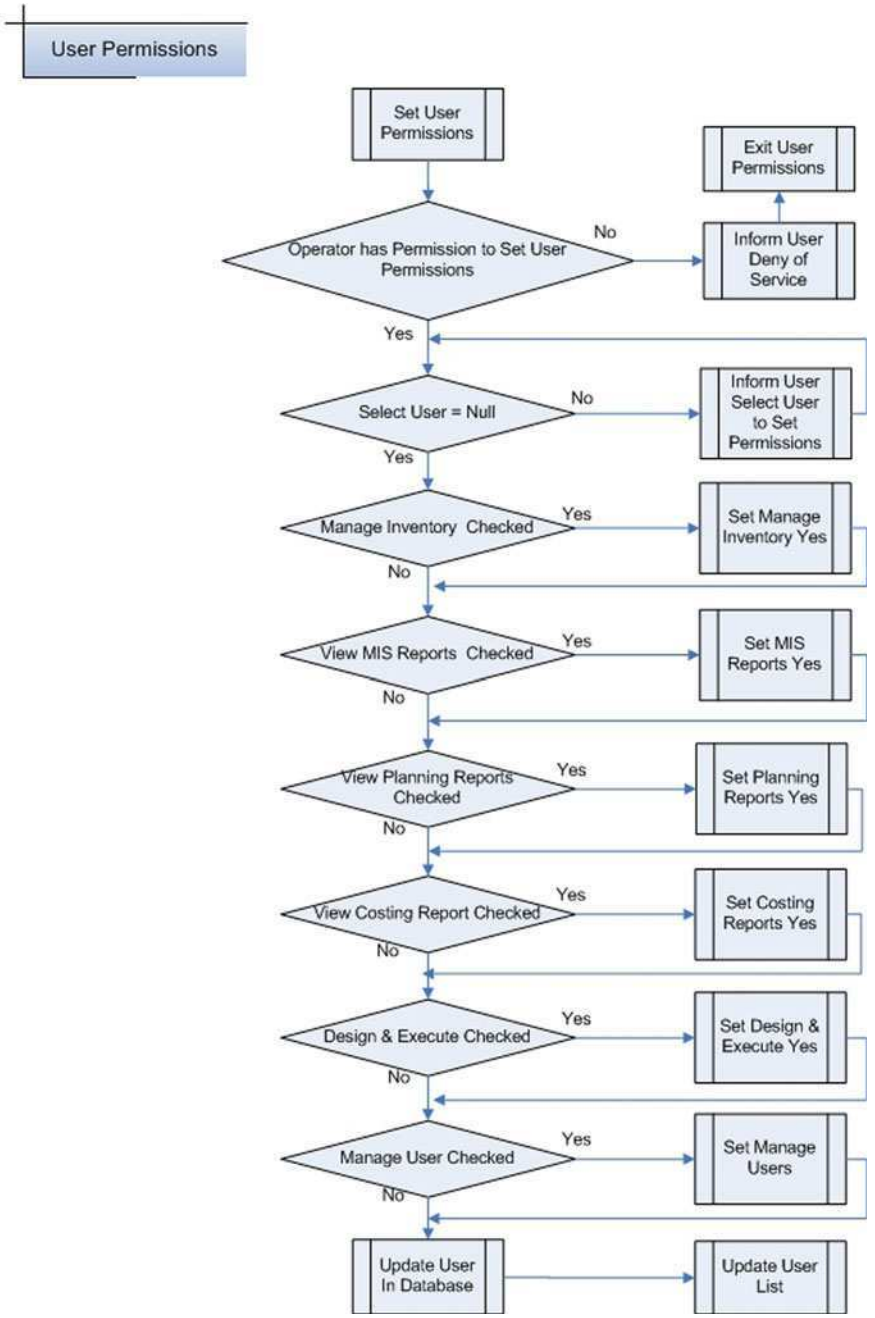


Fig. 11.93 Virtual Factory Set User Permission Flow Chart

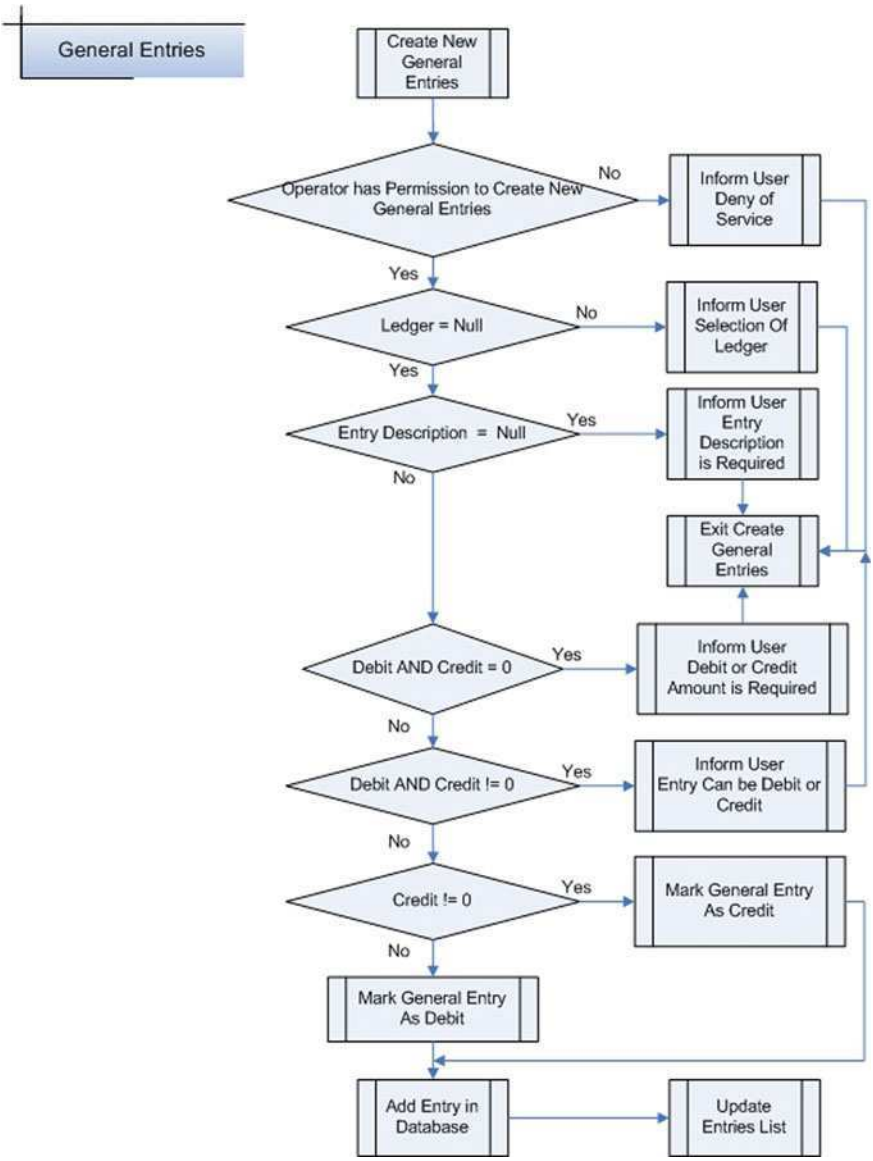


Fig. 11.94 Virtual Factory General Entries Flow Chart

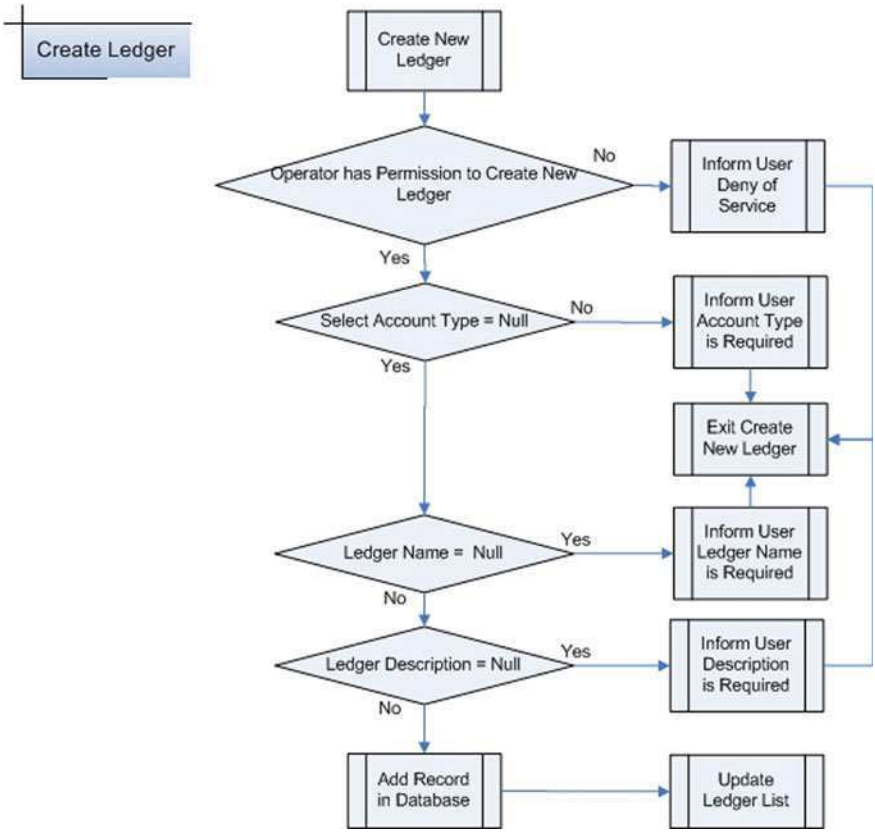


Fig. 11.95 Virtual Factory Create Ledger Flow Chart

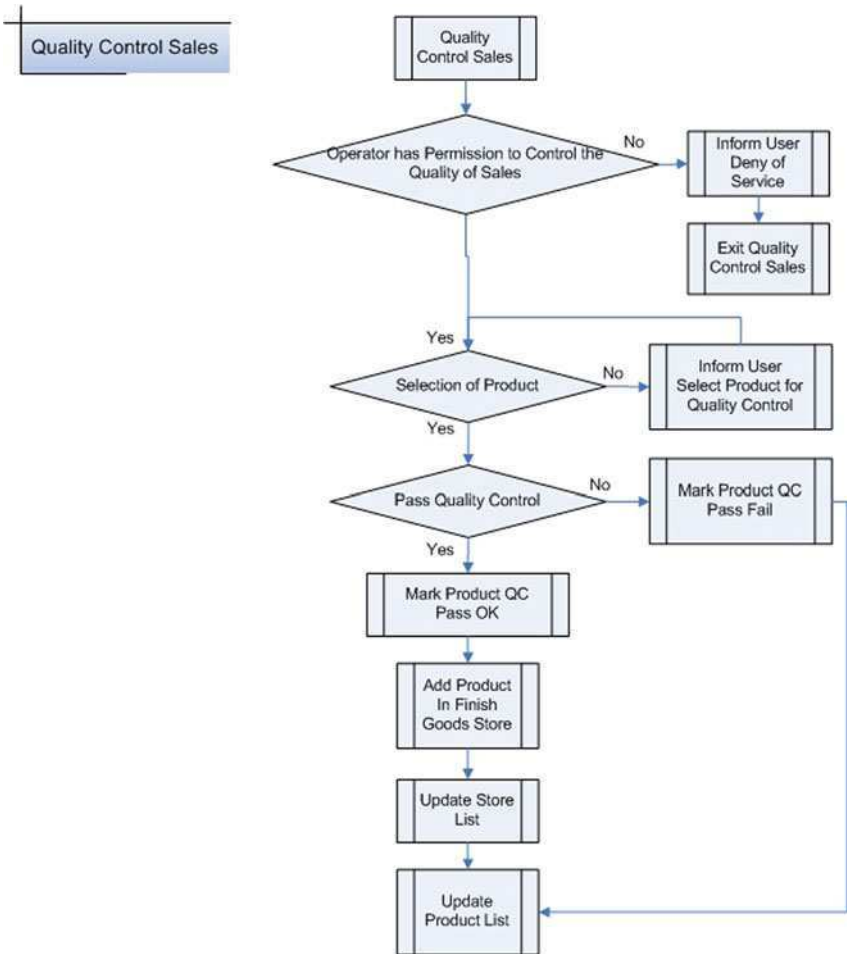


Fig. 11.96 Virtual Factory Quality Control Sales Flow Chart

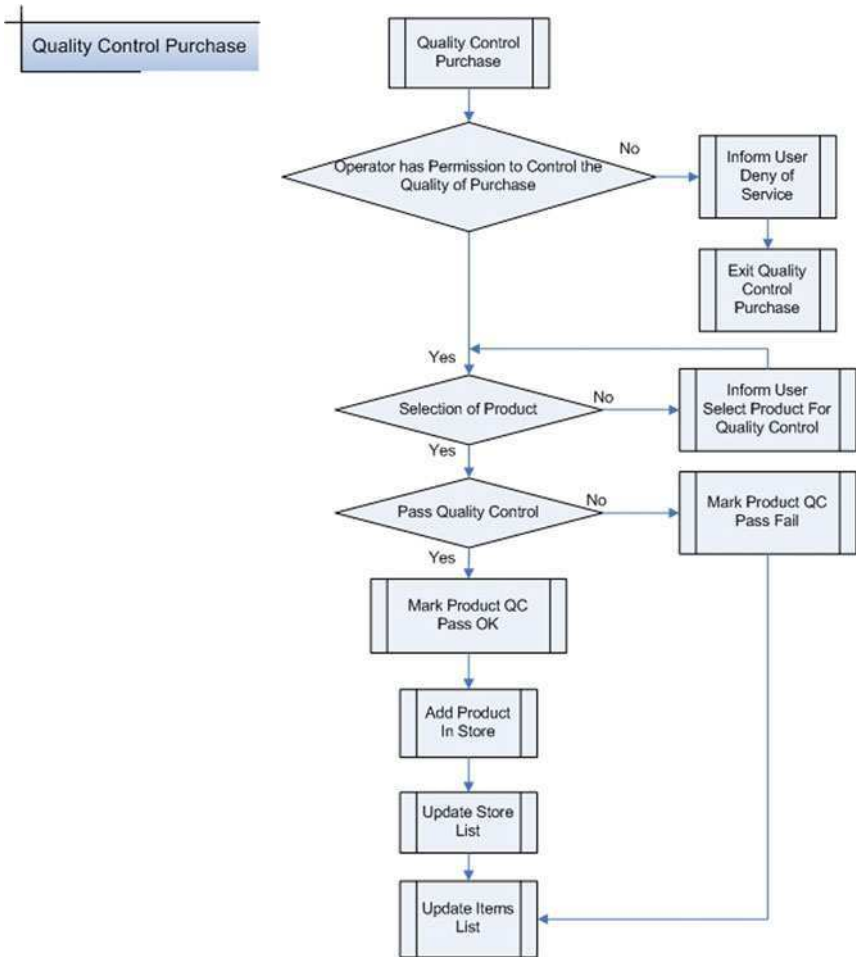


Fig. 11.97 Virtual Factory Quality Control Purchase Flow Chart

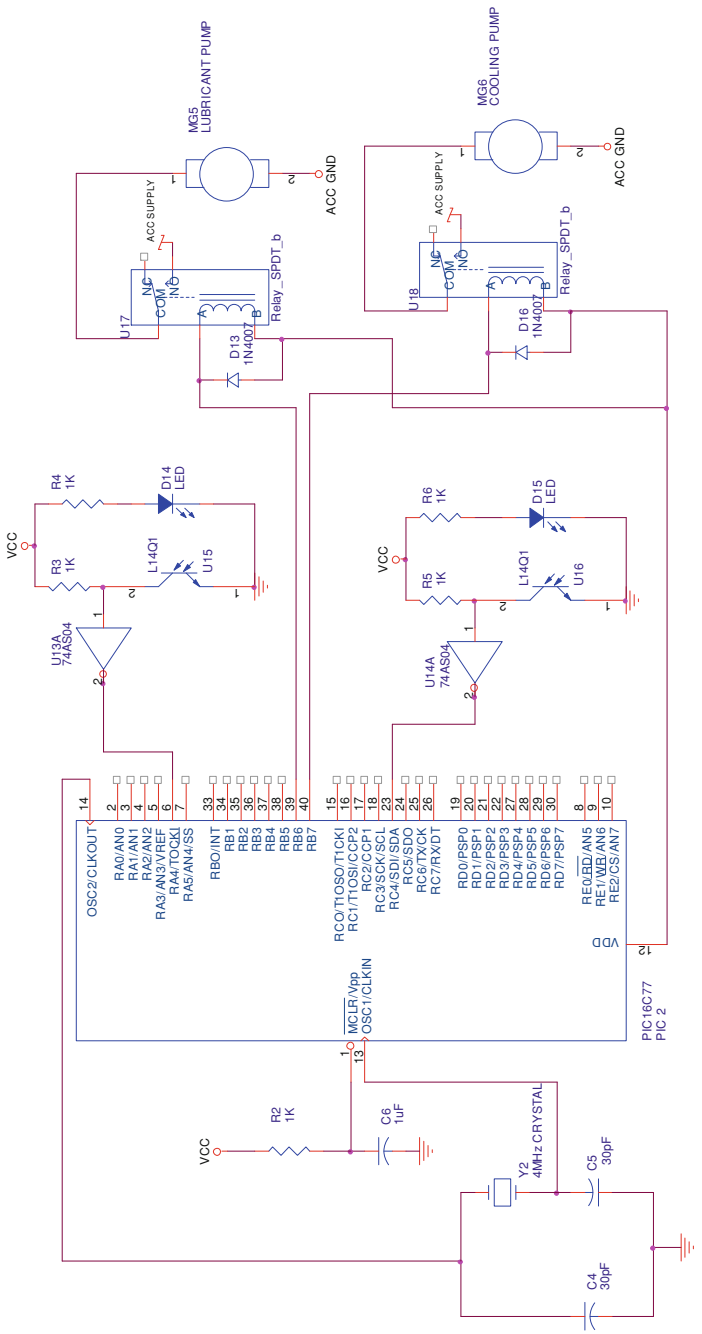


Fig. 11.98 Control of Virtual Factory through local area network

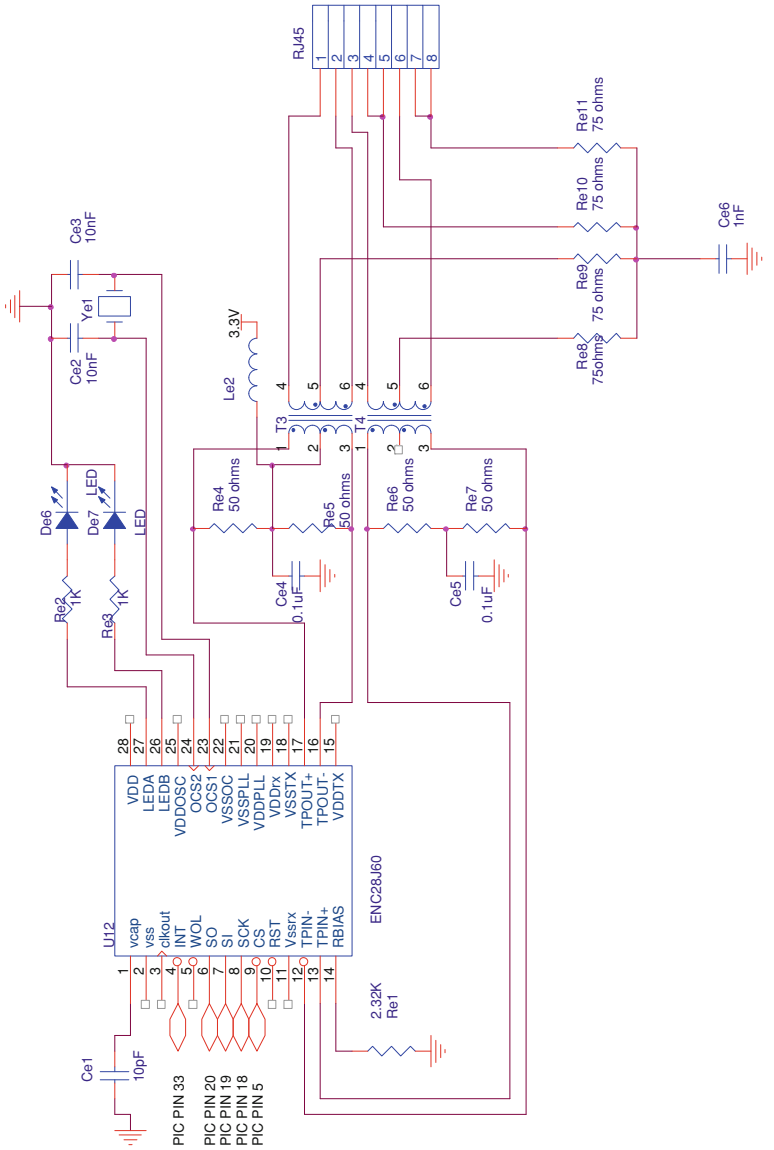


Fig. 11.98 (continued)

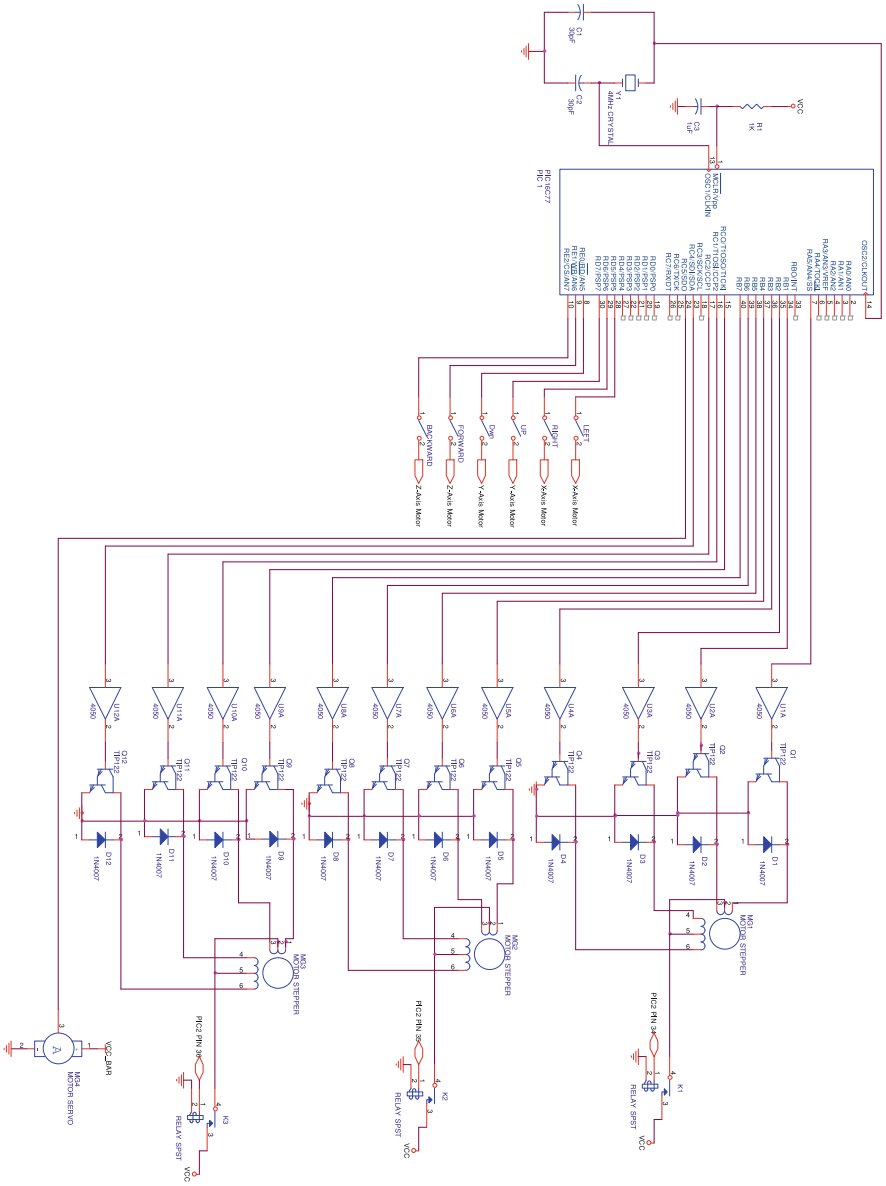


Fig. 11.99 Control of Virtual Factory for milling cell through local area network

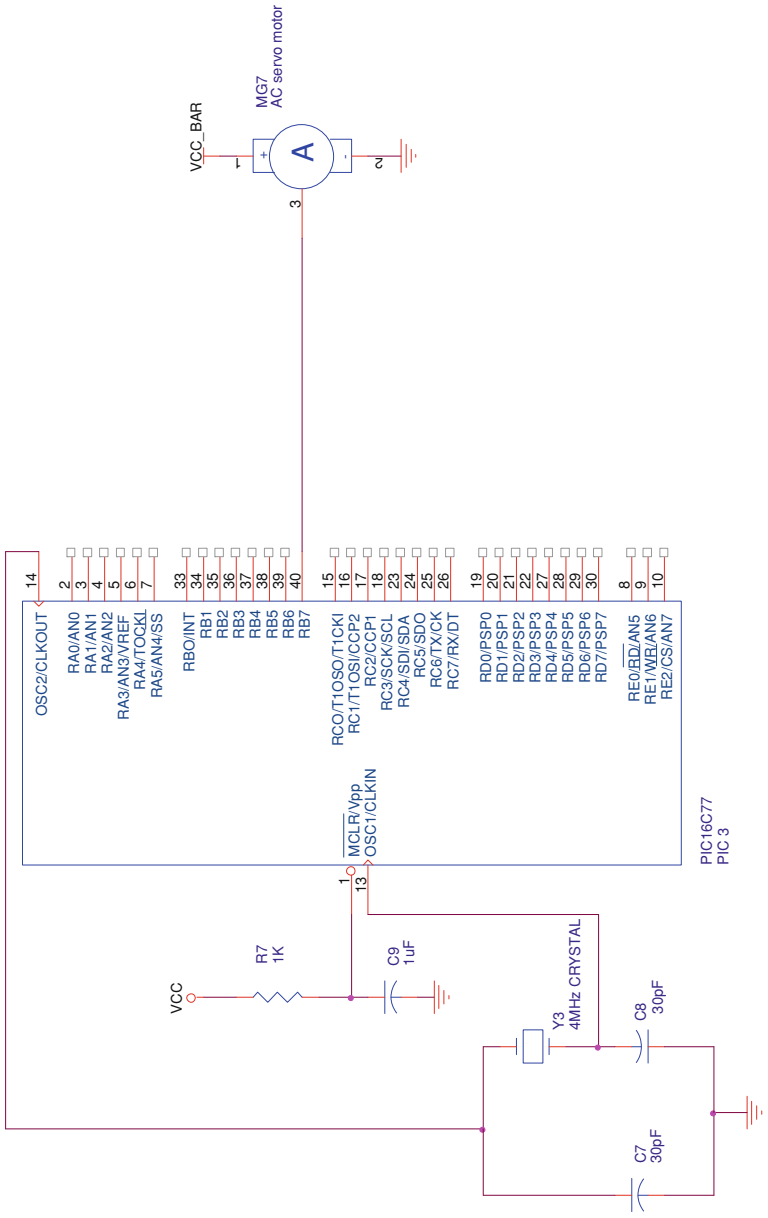


Fig. 11.100 Control of Virtual Factory for drilling cell through local area network

Bibliography

1. Banerjee A (2000) Behavioral layer architecture for telecollaborative virtual manufacturing operations. IEEE Trans Robot Autom 16(3):218–227
2. Ben-Arieh D (2008) Modeling and simulation of a virtual manufacturing enterprise. Int J Comput Integr Manuf 21(5):495–509
3. Bless PN et al (2005) An algorithmic strategy for automated generation of multicomponent software tools for virtual manufacturing. J Manuf Sci Eng 127(4):866–874
4. Cecil J et al (2004) Mobile agent and semantic web based framework for the realization of virtual manufacturing enterprises. Am Soc Mech Eng Manuf Eng Div 15:557–567
5. Chen L (2002) Integrated virtual manufacturing systems for process optimization and monitoring. CIRP Ann Manuf Technol 51(1):409–412
6. Chen KZ (2007) A virtual manufacturing system for components made of a multiphase perfect material. CAD Comput Aided Design 39(2):112–124
7. Chou W (2004) A collision detection method for virtual manufacturing. Trans North Am Manuf Res Inst SME 32:319–326
8. Depincce P (2004) The virtual manufacturing concept: scope, socio-economic aspects and future trends. In: Proceedings of the ASME design engineering technical conference, 4: 229–308
9. Gierach K et al (2002) An approach for facilitating service management in networked virtual manufacturing environments. Robot Comput Integr Manuf 18(2):147–156
10. Ito T (2003) Collaborative implant design using the virtual manufacturing approach. Int J Comput Integr Manuf 16(7–8):541–545
11. Jia Q (2006) Study on the building method of immersive virtual environment based on commodity components for virtual manufacturing. In: IET conference publication, pp 1417–1423
12. Kesen S et al (2010) A mixed integer programming formulation for scheduling of virtual manufacturing cells. Int J Adv Manuf Technol 47(5–8):665–678
13. Kim BH et al (2004) R&D activities in Korea on virtual manufacturing. J Adv Manuf Syst 3(2):193–204
14. Kraiem N (2003) Virtual manufacturing, virtual spaces and meta-modeling. In: Proceedings of the international conference on telecommunications, pp 364–369
15. Lin SH (2008) Learning efficiency of using virtual manufacturing in e learning for the operation technology of injection molding machine. In: Technical papers, regional technical conference, Society of Plastic Engineering, pp 436–439
16. Mertins K et al (2000) Capacity assignment of virtual manufacturing calls by applying lot size harmonization. Int J Prod Res 38(17):4365–4391
17. Mok SM (2001) Modeling automatic assembly and disassembly operations for virtual manufacturing. IEEE Trans Syst Man Cybern Part A Syst Hum 31(3):223–232
18. Ning G et al (2009) Formation of a new manufacturing resource model based on virtual manufacturing cell. In: Proceedings international conference on computational intelligence and software engineering
19. Nomden G et al (2006) Virtual manufacturing cells: a taxonomy of past research and identification of future research issues. Int J Flexible Manuf Syst 17(2):71–92
20. Papstel J (2000) Virtual manufacturing in reality. In: Proc SPIE Int Soc Opt Eng, pp 123–133
21. Peng Q (2007) A networked virtual manufacturing system for SMEs. Int J Comput Integr Manuf 20(1):71–79
22. Pohit G (2006) Application of virtual manufacturing in generation of gears. Int J Adv Manuf Technol 31(1–2):85–91
23. Radharamanan R (2002) Virtual manufacturing: an emerging technology. In: ASEE—annual conference proceedings, pp 9957–9967
24. Radharamanan R (2004) The present and the future of virtual manufacturing. In: IIE annual conference and exhibition, pp 2691–2730

25. Song JS (2004) Open modes between real and virtual manufacturing devices. In: Proceedings of SICE annual conference, pp 2979–2962
26. Souza MCF (2006) Virtual manufacturing as a way for the factory of the future. J Intell Manuf 17(6):725–735
27. Tacechi S (2002) Virtual manufacturing system based on product modeling development of computer aided geometric accuracy management system for steel structure. In: Proceedings of the international offshore and polar engineering conference, pp 352–359
28. Tesic R (2001) Exact collision detection for a virtual manufacturing simulator. IIE Trans (Inst Ind Eng) 33(1):43–54
29. Wadhwa S et al (2009) Organizing a virtual manufacturing enterprise: an analytic network process based approach for enterprise flexibility. Int J Prod Res 47(1):163–186
30. Wu SH et al (2002) Concurrent process planning and scheduling in distributed virtual manufacturing. IIE Trans (Inst Ind Eng) 34(1):77–89
31. Xu Y et al (2008) Category theory-based object-oriented data management for virtual manufacturing. Int J Internet Manuf Serv 1(2):136–159
32. Yin X (2007) The virtual manufacturing model of the worsted yarn based on artificial neural network and grey theory. Appl Math Comput 185(1):322–332
33. Zhang WY (2010) Application and development of domestic and foreign virtual manufacturing technology. Appl Mech Mater 20–23:1205–1210
34. Zhou ZD et al (2003) A multi-agent based agile scheduling model for a virtual manufacturing environment. Int J Adv Manuf Technol 21(12):980–984
35. Zhou J (2004) Visual diagnostic of bottleneck processes and redesign for the production line based on virtual manufacturing technology and its application. In: ICMA—Proceedings of the international conference on manufacturing automation, pp 505–512

Chapter 12
The Future of Virtual Manufacturing Using Augmented Reality Technology

12.1 The Technological Excellence[1]

In late 1990, Board on Manufacturing and Engineering Design of the USA's National Research Council formed a committee on Visionary Manufacturing challenges. The objectives of the committee were

(I) to create a vision of the competitive environment for manufacturing and the nature of the manufacturing enterprise in 2020;
(II) to determine the major challenges for manufacturing to achieve the vision;
(III) to identify the key technologies for meeting these challenges;
(IV) to recommend strategies for measuring the progress the committee developed an information gathering process based on two primary mechanisms.

- A workshop was held for the participants (primarily from the USA) representing a broad range of manufacturing expertise. The workshop included presentations and discussions on future trends in economics, business practices, environmental concerns, and manufacturing issues.
- An international Delphi survey of manufacturing experts (more than 40 percent outside the USA) was conducted.

For the vision of manufacturing in 2020, the most important technical, political and economic forces; as identified by the committee, for the development of manufacturing are as follows:

(a) The competitive climate, enhanced by communication and knowledge sharing, will require rapid responses to market forces.
(b) Sophisticated customers, many in newly developed countries, will demand products that are customized to meet their needs.

[1] Extracted with permission from: John J. Bollinger (ed) (1998) Visionary manufacturing challenges for 2020. National Academy Press, Washington, DC.

W. A. Khan et al., *Virtual Manufacturing*, Springer Series in Advanced Manufacturing, 751
DOI: 10.1007/978-0-85729-186-8_12, © Springer-Verlag London Limited 2011

(c) The basis of competition will be creativity and innovation in all the aspects of the manufacturing enterprise.
(d) The development of innovative process technologies will change both the scope and scale of manufacturing.
(e) Environmental protection will be essential as the global ecosystem is strained by growing populations and the emergence of new high technology economies.
(f) Information and knowledge on all the aspects of manufacturing enterprises and the marketplace will be instantly available in a form that can be effectively assimilated and used for decision making.
(g) The global distribution of highly competitive production resources, including skilled workforces, will be a critical factor in the organization of manufacturing enterprises.

The six grand manufacturing challenges, as suggested by the committee, for manufacturing which represent gaps between current practices (1990) and the vision of manufacturing in 2020 are as follows:

(a) Achieving concurrency in all operations.
(b) Integrating human and technical resources to enhance workforce performance and satisfaction.
(c) "Instantaneously" transforming information gathered from a vast array of diverse sources into useful knowledge for making effective decisions.
(d) Reducing production waste and product environmental impact to "near zero."
(e) Reconfiguring manufacturing enterprises rapidly in response to changing needs and opportunities.
(f) Developing innovative manufacturing processes and products with a focus on decreasing dimensional scale.

In order the key technologies to meet the above grand challenges, the following criteria were used by the committee:

(a) Was the technology identified as a high priority technology in the survey?
(b) Was the technology identified as a high priority technology at the workshop?
(c) Is this a primary technology for meeting one of the grand challenges?
(d) Does the technology have the potential to have a profound impact on manufacturing?
(e) Does the technology support more than one grand challenge?
(f) Does the technology represent a long-term opportunity (i.e., Is the technology not readily attainable in the short term)?

After evaluating many ideas, the committee selected ten strategic technology areas as the most important for meeting the grand challenges. These technology areas are as follows:

(a) Aadaptable, integrated equipment, processes, and systems that can be readily reconfigured;
(b) Manufacturing processes that minimize waste and energy consumption;

(c) Innovative processes for designing and manufacturing new materials and components;
(d) Biotechnology for manufacturing;
(e) System synthesis, modeling, and simulation for all the manufacturing operations;
(f) Technologies to convert information into knowledge for effective decision making;
(g) Product and process design methods that address a broad range of product requirements;
(h) Enhanced human–machine interfaces;
(i) New educational and training methods that enable the rapid assimilation of knowledge;
(j) Software for intelligent collaboration systems.

The committee then identified research opportunities to support the development of the priority technology areas. The committee's general findings are listed below:

(a) Many of the areas for research are crosscutting areas, that is, they are applicable to several priority technologies. Adaptable and reconfigurable manufacturing systems, information and communication technologies, and modeling and simulation are especially important because they are the key to manufacturing capabilities in many areas.
(b) Two important breakthrough technologies—submicron manufacturing and enterprise simulation and modeling—will accelerate progress in addressing the grand challenges.
(c) Substantial research is already under way outside of the manufacturing sector that could be focused on manufacturing applications.
(d) Progress toward the goals recommended in the Next Generation Manufacturing study on the needs of the next decade would provide some fundamental building blocks for meeting the longer-term grand challenges for 2020. These research areas include (1) analytical tools for modeling and assessment, (2) processes for capturing and using knowledge for manufacturing, and (3) intelligent processes and flexible manufacturing systems.
(e) Because manufacturing is inherently multidisciplinary and involves a complicated mix of people, system, processes, and equipment, the most effective research will also be multidisciplinary and grounded in knowledge of manufacturing strategies, planning, and operations.

Recommendation. Establish an interdisciplinary research and development program that emphasizes multi-investigator consortia both within institutions and across institutional boundaries. Establish links between research communities in the important disciplines required to address the grand challenges, including all branches of engineering, mathematics, philosophy, biology, psychology, cognitive science, and anthropology.

Recommendation. Focus long-term manufacturing research on developing capabilities in the priority technology areas to meet the grand challenges.

Recommendation. Establish priorities for long-term research with an emphasis on crosscutting technologies, i.e., technologies that address more than one grand challenge. Adaptable and reconfigurable manufacturing systems, information and communication technologies, and modeling and simulation are the three research areas that address several grand challenges.

Recommendation. Establish basic research focused on breakthrough technologies, including innovative submicron manufacturing processes and enterprise modeling and simulation. Focus basic research on the development of a scientific base for production processes and systems, which will support new generations of innovative products.

Recommendation. Monitor the research and development on technologies that will have significant investment from outside the manufacturing sector and undertake research and development, as necessary, to adapt them for manufacturing application. Some applicable technologies are listed below:

(a) Information technology that can be adapted and incorporated into collaboration systems and models through manufacturing-specific research and development focused on improving methods for people to make decisions, individually and as part of a group;
(b) Core technologies, including materials science, energy conservation, and environmental protection technologies

Recommendation. Industry and government should focus interdisciplinary research and development on the priority technology areas. Some key consideration for the long-term are listed below:

(a) understanding the effect of human psychology and social sciences on decision-making processes in the design, planning, and operation of manufacturing processes;
(b) managing and using information to make intelligent decisions among a vast array of alternatives;
(c) adapting and reconfiguring manufacturing processed rapidly for the production of diverse, customized products;
(d) adapting and reconfiguring manufacturing enterprises rapidly to enable the formation of complex alliances with other organizations;
(e) developing concurrent engineering tools that facilitate cross-disciplinary and enterprise-wide involvement in the conceptualization, design, and production of products and services to reduce time-to-market and improve quality;
(f) developing educational and training technologies based on learning theory and the cognitive and linguistic sciences to enhance interactive distance learning;
(g) optimizing the use of human intelligence to complement the application and implementation of new technology;
(h) understanding the effects of new technologies on the manufacturing workforce, work environment, and the surrounding community

One of the key factors in meeting the grand challenges will be monitoring the progress of technology development. The committee believes that a detailed

research agenda and timetable based on the grand challenges and priority technology areas for manufacturing in 2020 should be developed. However, detailed research agendas or timetables were beyond the scope of this study. Research road maps that could be used to monitor progress toward realization of the vision of manufacturing in 2020 should be established in follow-up technology seminars with focus groups exploring the priority technologies and potential research areas. Rather than trying to anticipate the advancements for a twenty-year period, the committee recommends that general long-term goals be established in each technology area and that detailed road maps be established for five-year "window of commitment." This approach, similar to the approach of the Defense Advanced Research Projects Agency, would provide a reasonable time frame for technology incubation, with yearly reviews to monitor progress. At the end of the five-year period, goals and programs would be re-examined for the next five-year period. This approach would allow research efforts to be adapted to revolutionary advances and for unfruitful research directions to be reconsidered.

The development of augmented reality for discrete manufacturing aligns well with the findings of committee on visionary manufacturing changes. The areas where given concept is valid are

(a) Software such as Virtual Manufacturing System (VMS) provides rapid response within the manufacturing enterprise and the market place. It covers both internal and external operations of a discrete manufacturing set up.
(b) Development of customized product for the global market.
(c) Enhanced creativity and innovation in tangible and intangible aspects of manufacturing system.
(d) Information and knowledge on all aspects of manufacturing system so that the market place is available instantly and can be effectively assimilated and used for decision making.
(e) The global distribution of highly competitive production resources shall be accessible through the use of Augmented Reality over internet.
(f) Grand challenge to achieve concurrency in all the operations that shall be achievable.
(g) Grand challenge to integrate human and technical resources that shall be achievable and shall enhance work force performance and satisfaction.
(h) Grand challenge of transforming information instantaneously from a vast array of diverse sources into useful knowledge for making effective decision which shall become possible.
(i) Grand challenge of reconfiguring manufacturing enterprises rapidly in response to changing needs and opportunities which shall be achievable.
(j) Implementation of Augmented Reality concept that is a method of achieving, adaptable, integrated equipment, processes, and systems that can be readily configured.
(k) Augmented Reality that supports system synthesis, modeling, and simulation for all the manufacturing operations.

(l) Augmented Reality that has the capability of transforming information into knowledge for effective decision making.
(m) Augmented Reality that provides an enhanced human–machine interface.
(n) Augmented Reality that permits new educational and training methods.
(o) Augmented Reality that provides software for intelligent collaboration systems.

12.2 Adoption of Standard Products

The development of Augmented Reality for discrete manufacturing demands that the standard products and service should be used to retain convenience in operation and maintenance of the virtual factory. These standard products and services are the essence of different subject areas such as Computer Science and Engineering, Communication Networks, Control Elements, and Mechanical hardware. This decision leads to maximizing the quality, reliability, maintainability and availability for the services offered by the virtual factory. This scheme shall offer the following:

(a) Convenience in building, operating, and maintaining a virtual factory;
(b) Convenience in training the workforce;
(c) Readiness in incorporating management strategies;
(d) Reduced cost of the product;
(e) Ease in vendor development;
(f) Larger control over accounting and financial matters;
(g) Convenience in communication and commerce;
(h) Making it simpler to manage and turning into a profitable organization.

12.3 The Cost Factor

Virtual Management like its predecessor is a major element of wealth creation. Every virtual manufacturing set up is designed to generate profits. It involves the use of scientific principles, technical information, synthesis, analysis, creativity, and decision making. It requires the consideration of human and environmental factors with the maximum practicable economy and efficiency.

Similar to the control gained by first cut at the NC milling machine at MIT, the Virtual Manufacturing holds many promises by vesting total control of the factory with the managers. The equipment, training, and operational costs are high, but the technology that currently exists is being used in isolation. Top-of-the-range manufacturing enterprises throughout the world are currently in a position to adopt a suitable level of virtual manufacturing and enhance it with a slower pace as the

confidence builds up. Medium- and small-size organizations, either working independently or as vendors to larger organizations, are the most to benefit despite the high cost of equipment, operation and training.

12.4 The Prospects for a Dynamic Business Environment

Virtual Manufacturing defines new economies for manufacturing. Organizations currently working in complete isolation with respect to other similar organizations have better chances of interaction for higher utilization of manufacturing facility, productivity, and profit margins while working in compliance with pertinent standards and local, regional, and international laws.

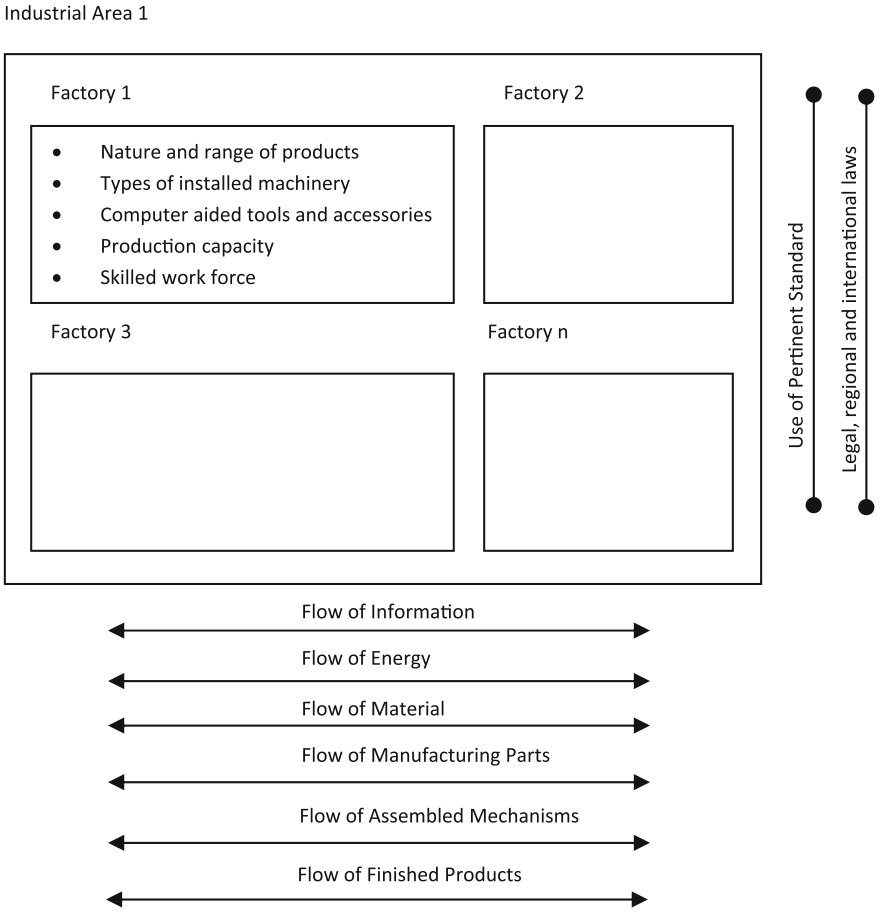


Fig. 12.1 Manufacturing activity in an industrial area

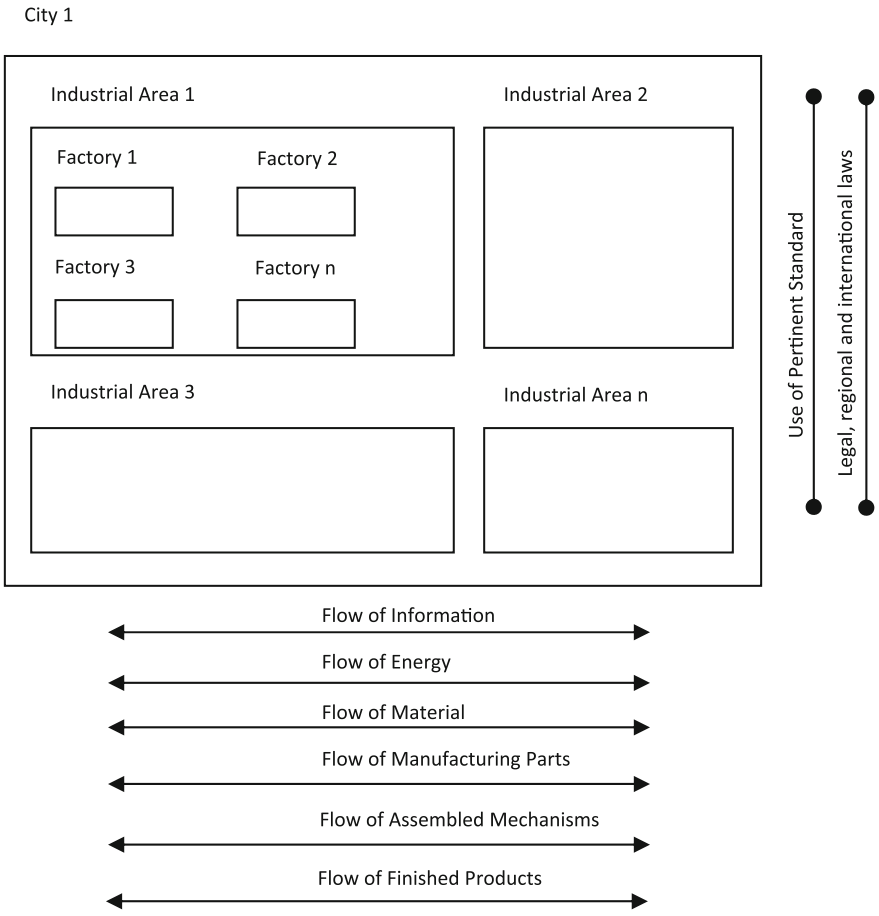


Fig. 12.2 Manufacturing activity in a city

Virtual Manufacturing provides prospects for a dynamic business environment by maximizing the flow of information, the contents of information, knowledge embedded in the information, and the optimal use of information. Virtual Manufacturing makes the boundaries of the manufacturing enterprise void and null by allowing production of components, mechanism and products within a single manufacturing enterprize, or, based on the optimality of producing goods across the borders. Communication technology becomes very important, and the use of Internet, Intranet, and Extranet takes a new shape.

Figures 12.1, 12.2, 12.3, 12.4, 12.5 give the physical layout of the global manufacturing domain starting from a factory in an industrial area to manufacturing activities all over the world. Level of Augmented Reality implementations

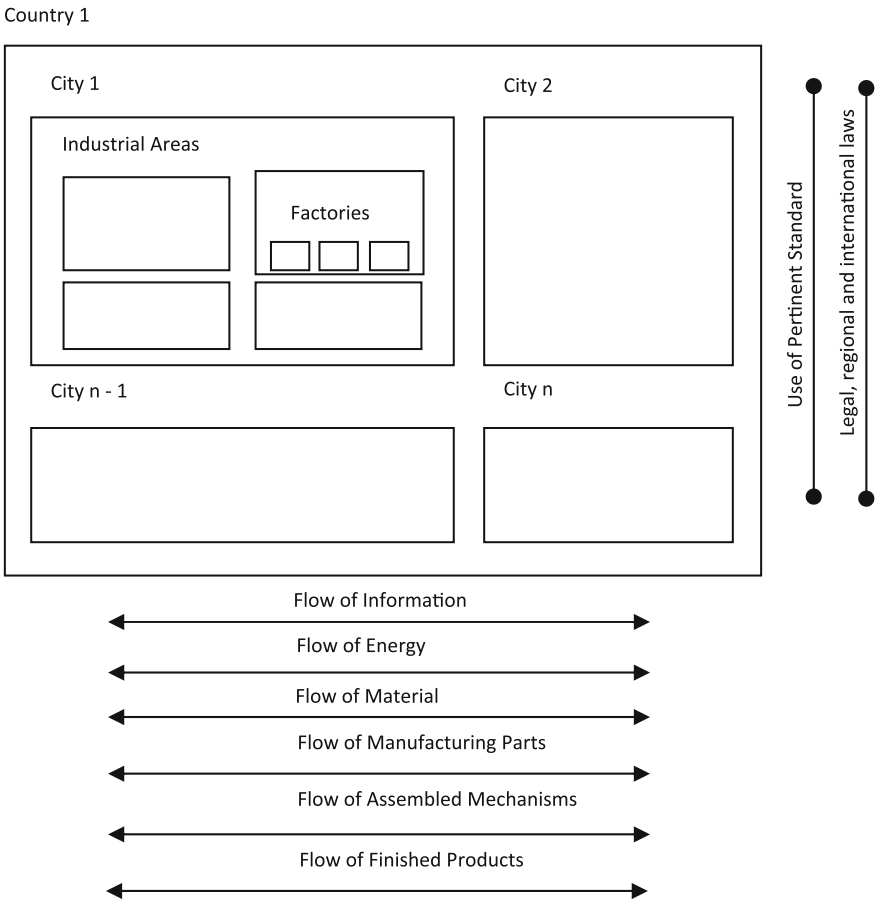


Fig. 12.3 Manufacturing activity in a country

as presented in this book leads to rapid response from manufacturing enterprise and the market place. VMS Monitor and VMS Business are practical implementations in this scenario. VMS Business searches for another company in business-to-business or business-to-client environments and negotiates part, mechanism, or product manufacturing in a global domain. Since VMS software demonstrates control of microdomain, such realizations may be observed at various levels of automation. Some organizations shall be able to implement the concept fully, while others may be able to implement a part of it. Whatever be the level of implementation, the flow of information is the most important aspect that should be looked into.

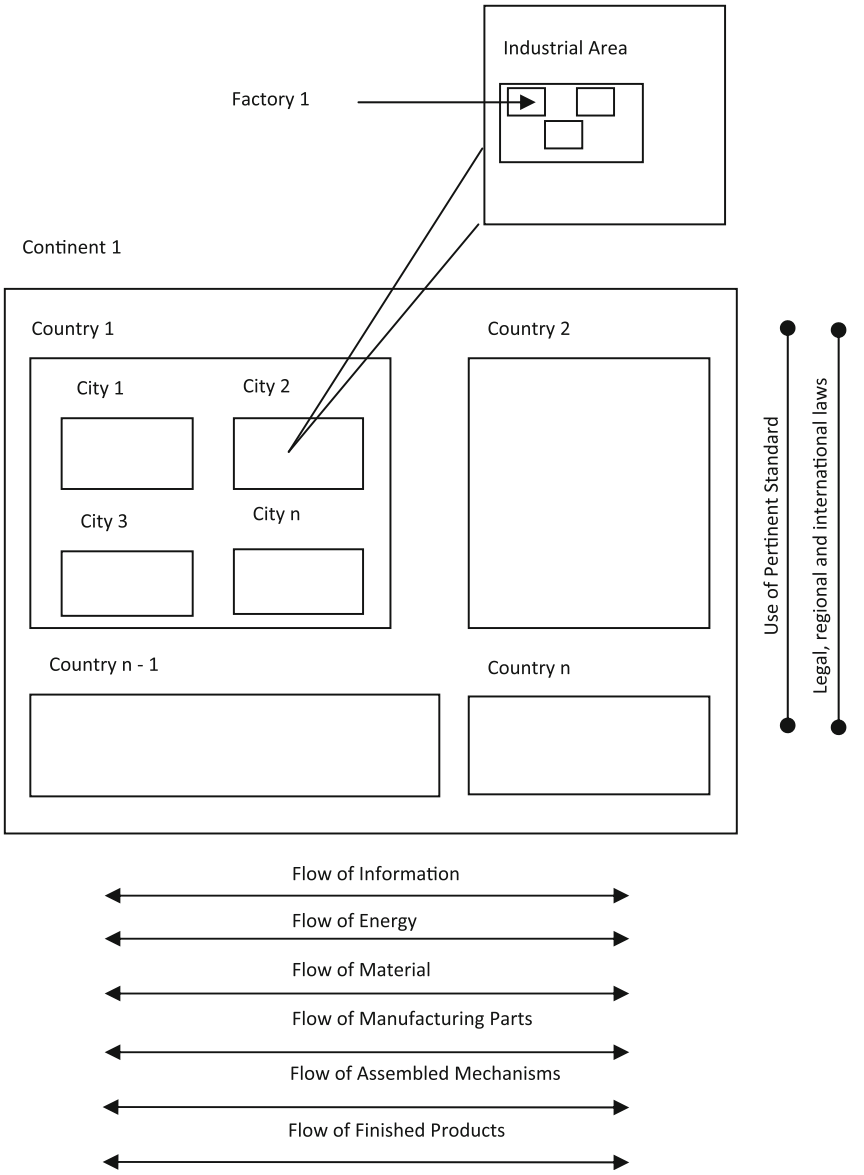


Fig. 12.4 Manufacturing activity in a continent

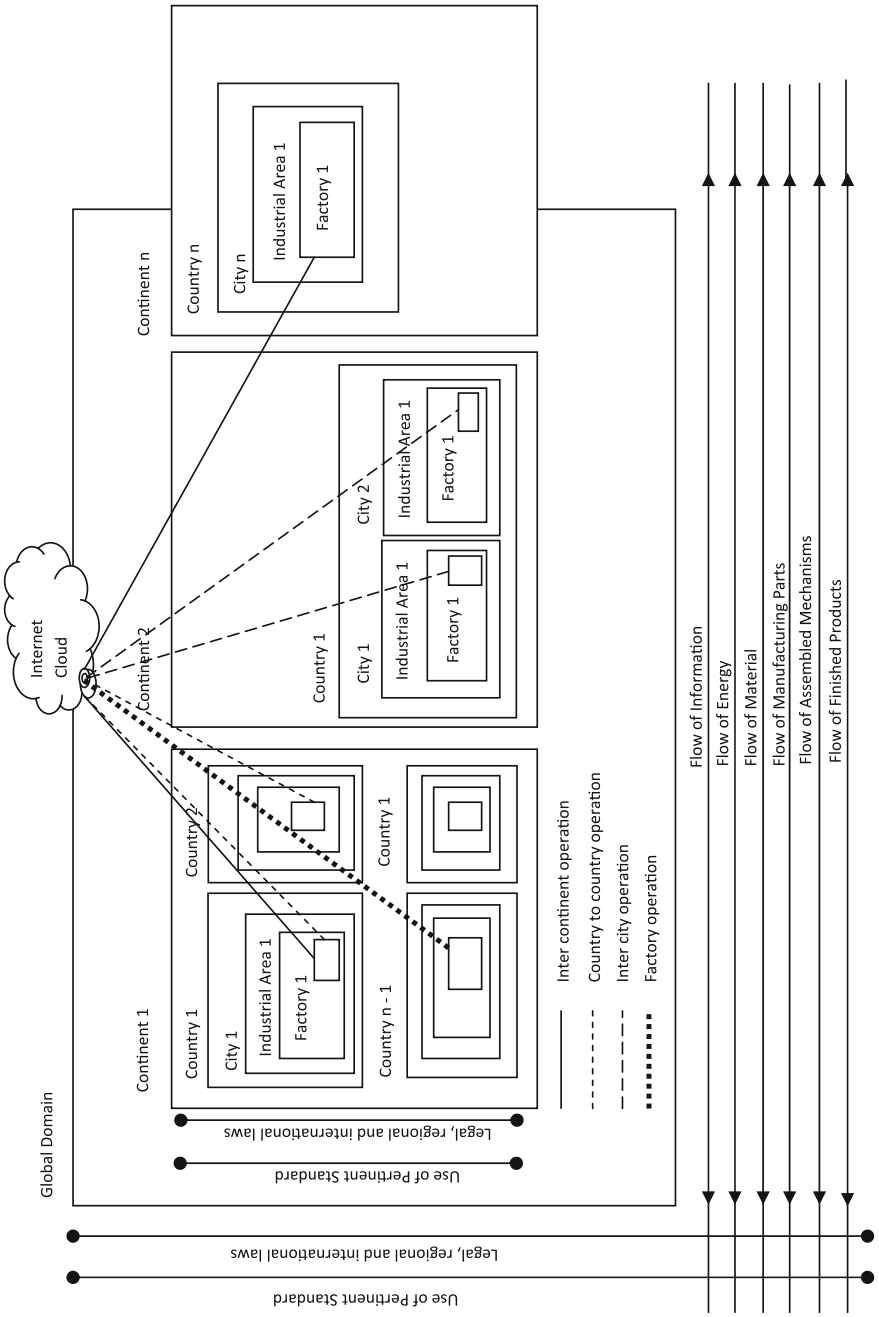


Fig. 12.5 Virtual Manufacturing in Global domain

Bibliography

1. Anon (2001) Manufacturing excellence demands tools for operational effectiveness. Control Solut 74(6):67
2. Anon (2002) Global best practices: manufacturing excellence. Chem Mark Report 261(20):17
3. Anon (2003) Delivering manufacturing excellence. Foundry Trade J 177(3602):26–27
4. Anon (2006) A recipe for operational excellence. Control Eng 53(12 Suppl):2–3
5. Bergsma B, Mattingly D (2006) Achieving manufacturing excellence. In: IIE annual conference and exposition
6. Burmood P, Bergsma B (2005) Achieving manufacturing excellence. In: IIE annual conference and exposition
7. Dangayach GS (2004) Linkages between manufacturing strategy, business strategy and business excellence: A longitudinal study. Int J Indust Eng 11(3):297–306
8. Dwivedi SN (2001) Balancing of competitiveness metrics and measures for excellence in manufacturing. Am Soc Mech Eng 12:239–448
9. Gilgeous A (2001) A survey to assess the use of a framework for manufacturing excellence. Integr Manuf Systems 12(11):48–58
10. Gresham D, Eckman MW (2003) The global supply chain: manufacturing network approach to operational excellence. Chem Mark Report 263(11):28–30
11. Hvolby H, Trienekens JH (2002) Special issue: stimulating manufacturing excellence in small and medium enterprises. Comput Ind 49(1):1–2
12. Kaczmarek K (2006) Operational excellence. ABB Rev (1):52–54
13. Matt DT (2007) Achieving operational excellence through systematic complexity reduction in manufacturing system design. Key Eng Mater 344:865–872 (Proceedings of the 12th International Conference, Sheet Metal 2007)
14. Mott RL (2002) National center of excellence for advanced manufacturing education. In: ASEE Annual Conference Proceedings, pp 2452–2455
15. Russell J (2004) Achieving operational excellence. Appliance 61(7):48–51
16. Sharma RK et al. (2006) Manufacturing excellence through TPM implementation: a practical analysis. Indl Manag Data Syst 106(2):256–280
17. Spear M (2003) Manufacturing excellence 2003: a distillation of experience. Process Eng (Lond) 84(6):35
18. Williams S (2004) Operational excellence: best practices in manufacturing. In: A/ChE Spring national meeting, pp 375–382

Appendices

Appendix I

3D Graphics Software Tools by Categories

Source: http://cs.gmu.edu/∼jchen/graphics/book/index_tools_catagories.html

Low-Level Graphics Libraries

1. *Coin3D*
2. *DirectX*
3. GKS-3D
4. Jogl
5. Mesa
6. OpenGL
7. OpenGL For Java (GL4Java; MapsOpenGL and GLU APIs to Java)
8. QuickDraw3D
9. Tao Framework
10. XGL

Visualization Tools

1. 3D Grapher (Illustrating and solving complex mathematical equations in 2D and 3D)
2. 3D Studio VIZ (Architectural and industrial designs and concepts)
3. 3DField (Elevation data visualization)
4. 3DVIEWNIX (Image, volume, soft tissue display, kinematic analysis)

5. Amira (Medicine, biology, chemistry, physics, or engineering data)
6. Analyze (MRI, CT, PET and SPECT)
7. Antics (Visualization based upon gaming technology)
8. Astex Viewer (Molecular visualization)
9. AVS (Comprehensive suite of data visualization and analysis)
10. *Blueberry 3D (Virtual landscape and terrain from real map data)*
11. Deep Exploration (CAD, 3D content)
12. Dice (Data organization, runtime visualization, and graphical user interface tools)
13. Enliten (Viewing, analyzing and manipulating complex visualization scenarios)
14. Ensight (CFD, structural analysis, combustion, electromagnetics, and injection)
15. FAST (Data from numerical simulations)
16. Ferret (Oceanographers and meteorologists analyzing large and complex gridded data sets)
17. Fieldview (CFD or other data)
18. Geomview (Visualize and manipulate 3D geometric objects)
19. GNUPlot (Scientific data plotting of various forms)
20. IDL (Data analysis, visualization, and cross-platform application development)
21. Igor Pro (*Visualization of large data sets and time series*) Iris Explorer (3D data visualization, animation and manipulation)
22. Jmol (Molecular Visualization)
23. Khoros (Visual programming environment with data processing and visualization)
24. King (Graph, plot, and macromolecule)
25. LandForm (Map data as 3D surfaces with overlayed satellite and aerial imagery on top)
26. MapRender3D (Generate realistic relief shaded landscape perspectives)
27. Mathematica (Numeric and symbolic calculation, visualization, and simulation)
28. *Matlab* (*Data analysis/visualization, numeric/symbolic computation, simulation*)
29. Mvox (Visualization for medical images)
30. Ncar (Access, analysis, and visualization of data)
31. OpenDX (Science, engineering, medicine and business data)
32. OpenGL Volumizer (C++ classes, manipulation and display of voxel-based data)
33. OpenInventor (Cross-platform 3D graphics visualization and animation)
34. Perfkit (*Graphics program performance visualization*)
35. Plot 3D (Simulation and visualization of computational fluid dynamics solutions)
36. Tucan Series (*Real-time visualization and virtual reality*)
37. Pv3 (Unsteady, unstructured CFD visualization for supercomputers, parallel machines)

38. *PV-WAVE* (*Manipulate and visualize complex or extremely large technical datasets*)
39. RasMol (Molecular visualization)
40. STK (Scenario information from the Satellite Tool Kit software suite)
41. Star-CD (Multipurpose CFD code, complete design and development process)
42. Tecplot (Visualizing technical data from analyses, simulations and experiments)
43. View3D (Motif compliant widget, 3D data displays)
44. Vis5D (5-D gridded data sets by numerical weather and ocean models)
45. VisAD (Interactive and collaborative visualization and analysis of numerical data)
46. VisiQuest (Scientific visualization)
47. Visual3 (Interaction with pregenerated data for visualization, as in fluid flow modeling)
48. Visual Nature Studio (Landscapes visualization)
49. VMD (Molecular visualization)
50. VolVis (Volume visualization that unites numerous visualization methods)
51. VoxBlast (Imaging, volume rendering)
52. VP-Sculpt (Visualize, verify, repair, and perform calculations on 3D polygonal model)
53. VTK (3D graphics, image processing, and visualization)
54. WebMol (PDB visualization)

Modeling Tools

1. 3D Canvas (Drag-and-drop, animation)
2. 3DSOM Pro (image-based 3D modeling)
3. 3D Studio Max (Game effects, animation and rendering)
4. 3D Studio VIZ (Architectural and industrial designs and concepts)
5. 3D World Studio (constructive solid geometry (CSG) modeler)
6. 3Dom (Solid objects)
7. AC3D (Objects and scenes)
8. ACIS 3D Toolkit (Geometric solids and surfaces)
9. ActiveDimension3 (CAD)
10. Aladdin (Procedural textures, gasses, path-based animation)
11. AliasStudio (Sketching, modeling, and visualization)
12. Amapi3D (NURBS modeling, rendering, and animation)
13. Amorphium (3D sculpting and painting)
14. *Anark* (CAD or game modeling and simulation)
15. Anim8OR (3D modeling and character animation)
16. Animation Master (Movies, 3D story boards, virtual reality, business presentations)
17. Art of Illusion (*3D modeling, rendering, and animation tool*)

18. *AutoCAD* (*CAD for schematics of ships and buildings*)
19. bCAD (Design and rendering)
20. Behemot Graphics Editor (Objects and scenes)
21. Blender (TV commercials, visualizations, user interfaces)
22. *Blueberry 3D* (*Virtual landscape and terrain from real map data*)
23. BodyPaint 3D (Body texture mapping)
24. Breeze Designer (Interface with POV-Ray rendering)
25. BRL-CAD (CSG, solid modeling and CAD)
26. Bryce 3D (Landscape, terrain, water, etc.)
27. Calimax Modeller (Interface with POV-Ray rendering)
28. Carrara (Modeling, rendering, animation)
29. Cheetah3D (Modelling, rendering and animation)
30. Cinema 4D (Film, television, architecture, face, and multimedia)
31. ClayWorks (Modeling and rendering)
32. CyberMotion 3D Designer (Game, virtual design)
33. Deep Creator (3D modeler, texture creator, scripting engine)
34. Deep Exploration (CAD, 3D content)
35. Deep Paint 3D (Oils, watercolors, and pastels brushed directly onto 3D models)
36. *Deled 3D Editor* (*objects and skins*)
37. DesignCAD (Engineering, drafting, and architectural applications)
38. Design Workshop Pro (films and advertisements)
39. DigiCad 3D (Architecture, mapping, photogrammetry)
40. DMesh (Humanoid and other organic modelling)
41. Draw3D (Surface from 3D points)
42. Effect3D (Objects)
43. Electric Image Universe (Film, architecture, and engineering)
44. Equinox 3D (product prototype, game model, virtual object)
45. Form-Z (Architects, designers, engineers)
46. Freeworld3D (Terrain editor and world editor)
47. Gamebryo (Real-time games)
48. GameSpace (Characters, worlds, weapons and more)
49. Genesis3D (Real-time games)
50. Geomagic Studio (Product design)
51. Graffiti (Face painting)
52. Grome Modeler (Terrain and game scene modeling)
53. Hexagon (Polygonal modeler)
54. Houdini (Proceedural approach)
55. HyperFun (Modeling through language-based functions)
56. ImageModeler (2D image to 3D model)
57. Jet 3D (Graphics engine built for high performance real-time rendering)
58. JustCad (Engineering layouts, architectural drawings, furniture)
59. K-3D (Geometrical objects)
60. LandForm (Map data as 3D surfaces with overlayed satellite and aerial imagery on top)

61. MapRender3D (Generate realistic relief shaded landscape perspectives)
62. Lattice Designer (curved surface)
63. LightRay3D (Game contents, general visualization data)
64. Lipservice (3D facial sculpting and animation plugin and standalone for lightwave)
65. LSS Vista (Terrain, map)
66. LumeTools (Creating landscape, water, light, matter, and workbench)
67. Materialize3D (Model converter, material / texture editor and polygon processor)
68. *Matlab* (*Data analysis/visualization, numeric/symbolic computation*)
69. Maya (Commercials, character animation, and virtual reality)
70. Merlin 3D (CAD, video, game, architecture)
71. Meshwork (3D game or web)
72. Metris (Aerospace, automotive and other engineering industries)
73. Microstation TriForma (Building design and drawing)
74. Misfit Model 3D (triangle-based modeling)
75. *MilkShape 3D* (*modeling and editing utilities*)
76. MindsEye (Support NURBS designed for Linux and Unix Systems)
77. ModelMagic3D (OpenGL scenes rendered in real-time)
78. Modo (Polygon, subdivision surface modeling)
79. Mojoworld (Fractal-based modeling)
80. Moray (Wireframe PovRay plugin)
81. MultigenPro (Realtime 3D content for simulation, games, and other applications)
82. Natural Scene Designer (Trees, clouds, rocks, bushes, lakes, atmospheric effects)
83. Now3D (Drawing tool that allows the user to quickly create complex, realistic 3D pictures)
84. OpenFX (Digital post production, animation, game development, film and broadcast)
85. ParticleIllusion (Explosion, smoke, fire, sparkles)
86. PhotoModeler (Generate 3D models and measure existing object or scenes)
87. Pointstream 3DImageSuite (Processing point cloud data)
88. Poser (*3D-character animation and design tool*)
89. Povlab (for PovRay)
90. *PovRay* (*Persistence of Vision Ray-tracer*)
91. Pro-Engineer (Architectures, manufacturers, and drafts)
92. Punch (Home and landscape modeling)
93. Realflow (Fluids by calculating the interaction between particles)
94. Realsoft 3D (Architecture and modular design)
95. Rhino (*Curved surfaces*)
96. RXscene (Polygon and spline based modeler with an intuitive user interface)
97. Sced (Modeling with geometric constraints)
98. Shade (*Architectural and interior design, product design and prototyping, character design and rendering*)

99. ShapeCapture (3D measurement and modeling)
100. Silo (*Organically sculpting high-polygon models and precisely controlling hard-edged surfaces*)
101. Shave (Hair, trees, etc.)
102. SketchUp (*Architects, furniture designers*)
103. Softimage XSI (*Films, games*)
104. Solids++ (Solids modeling, surface modeling, curve modeling, polygonal modeling)
105. SolidThinking (Curved surfaces)
106. SolidWorks (*Mechanical design*)
107. Strata Live 3D (*Print, Web-enabled applications, and interactive games*)
108. Summit 3D (Complex virtual worlds)
109. T.Ed (Terrain and environment modeling)
110. Terragen (Photorealistic results for landscape, special effects, art and recreation)
111. TerraTools (3D geospatial modeling)
112. TopSolid (CAD and CAM)
113. TrueSpace (Advertisements, games, art works, and animated virtual environments)
114. TurboCAD (CAD package)
115. Ultimate Unwrap (Unwrapping 3D models)
116. UVMapper (Texture mapping onto polygonal surfaces)
117. Varkon (CAD)
118. Vectorworks (Building design and presentations)
119. Visviva (artistic modeling, hypertext layout, special effects creation, and animations)
120. VPYTHON (*3D modeling addition to Python*)
121. VP-Sculpt (Visualize, verify, repair, and perform calculations on 3D polygonal model)
122. VREK (Crating functional, interactive and immersive environments)
123. Vue (Creation, rendering and animation of natural scenery)
124. Wings 3D (Surface subdivision and modeling)
125. WorldBuilder (Land and water modeling and rendering)
126. World Construction Set (Landscape generation)
127. WorldUp Modeler (Creating and animating various VR worlds)
128. ZBrush (3D design and real-time rendering)

Rendering Tools

1. 3D Studio Max (Game effects, animation and rendering)
2. AccuRender (For models inside AutoCAD, Revit, or Rhino)
3. Adobe Illustrator CS (vector-based drawing program)
4. AIR (3D image rendering and animation)

5. Aladdin (Procedural textures, gasses, path-based animation)
6. Amorphium (3D sculpting and painting)
7. Behemot Graphics Editor (Objects and scenes)
8. Blender (TV commercials, visualizations, user interfaces)
9. *Blueberry 3D* (*Virtual landscape and terrain from real map data*)
10. BodyPaint 3D (Body texture mapping)
11. Bryce 3D (Landscape, terrain, water, etc.)
12. Carrara (modeling, rendering, animation)
13. Cinema 4D (Film, television, architecture, face, and multimedia)
14. CyberMotion 3D Designer (Game, virtual design)
15. Design Workshop Pro (films and advertisements)
16. DMesh (Humanoid and other organic modelling)
17. Electric Image Universe (Film, architecture, and engineering)
18. Finalrender (Raytracing rendering)
19. Genesis3D (Real-time games)
20. Geomview (Visualize and manipulate 3D geometric objects)
21. INSPIRER (indoor outdoor lighting simulation)
22. Jet 3D (Graphics engine built for high performance real-time rendering)
23. Jig (Geometrical objects, particle systems, hair)
24. K-3D.htm (Geometrical objects)
25. LightScape (AutoCAD and 3D Studio models)
26. LightWave 3D (Television shows, films and commercials, video games)
27. Lightworks (Rendering components for integration)
28. LSS Vista (Terrain, map)
29. MapRender3D (Generate realistic relief shaded landscape perspectives)
30. Maxwell Render (Architectural visualization, industrial and product design)
31. Maya (Commercials, character animation, and virtual reality)
32. MentalRay (Film, games, automotive and aerospace industries)
33. Merlin 3D (CAD, video, game, architecture)
34. Natural Scene Designer (Trees, clouds, rocks, bushes, lakes, atmospheric effects)
35. NuGraf (Scene composition, model viewing and model translation)
36. Povlab (for PovRay)
37. POV-Ray (*Persistence of Vision Ray-tracer*)
38. Quick3D (Viewing, organizing, and converting 3D files)
39. Radiance (analysis and visualization of lighting)
40. Rayshade (Computer art and architecture)
41. Realsoft 3D (Architecture and modular design)
42. RenderDrive (Rendering appliance with software)
43. RenderMan (*Motion picture, etc.*)
44. RenderPark (quantitative prediction of the illumination)
45. Softimage (*Films, games*)
46. SolidThinking (Curved surfaces)
47. SPECTOR (*Optical design and analysis*)
48. StereoPOV (Raytracer, generating stereoscopic images)

49. *Strata Live 3D* (*Print, Web-enabled applications, and interactive games*)
50. Superficie (Simple visualization)
51. Texture Lab-Tiling Tools (Collection of material maps, mapping types for 3D Max)
52. TrueSpace (Advertisements, games, art works, and animated virtual environments)
53. Vecta3D-Max (3DS Max plug-in, output low-bandwidth vector based images and animations)
54. Vector Works (Building design and presentations)
55. Vfleet (Computational science volume rendering)
56. Vray (Rendering engine for film and game productions) Vue (Creation, rendering and animation of natural scenery)
57. WorldBuilder (Land and water modeling and rendering)
58. ZBrush (3D design and real-time rendering)
59. YafRay (Ray tracing rendering)

Animation Tools

1. 3D Choreographer (Actor animation with paths and scripts)
2. 3D Studio Max (Game effects, animation and rendering)
3. ActiveWorlds (Networked envrionment with avatars)
4. AIR (3D image rendering and animation)
5. Aladdin (Procedural textures, gasses, path-based animation)
6. Alice (Scripting and prototyping environment for 3D object behavior)
7. Amorphium (3D sculpting and painting)
8. Anim8OR (3D modeling and character animation)
9. Animation Master (Movies, 3D story boards, virtual reality, business presentations)
10. Animation Stand (Scanning, painting and compositing 2-D animation with 3D shading)
11. Blender (TV commercials, visualizations, user interfaces)
12. Bryce 3D (Landscape, terrain, water, etc.)
13. Carrara (modeling, rendering, animation)
14. Cinema 4D (Film, television, architecture, face, and multimedia)
15. CyberMotion 3D Designer (Game, virtual design)
16. DAZ Studio (3D figure posing and animation)
17. DesignCAD (Engineering, drafting, and architectural applications)
18. Design Workshop Pro (films and advertisements)
19. DMesh (Humanoid and other organic modelling)
20. EIAS (Character animations, virtual world)
21. Electric Image Universe (Film, architecture, and engineering)
22. Endorphin (3D character animation)
23. Evolver (Computer generated artwork)

24. Facial Studio (Face animation)
25. GenTools (3D heads by age, gender and ethnicity)
26. Houdini (Proceedural approach)
27. iClone (Animated characters, clothing, 3D scenes)
28. Internet Character Animator (Animate 3D characters)
29. Iris Explorer (3D data visualization, animation and manipulation)
30. Jsprited (Tile and multiple-image-based animation)
31. K-3D (Geometrical objects)
32. Kyra Sprite Engine (Sprite engine)
33. Lifestudio:Head (3D facial animation)
34. LightWave 3D (Television shows, films and commercials, video games)
35. Lipservice (3D facial sculpting and animation plugin and standalone for lightwave)
36. Maya (Commercials, character animation, and virtual reality)
37. Merlin 3D (CAD, video, game, architecture)
38. Messiah:Studio (Character animation)
39. MindsEye (Support NURBS designed for Linux and Unix Systems)
40. Mirai (3D game development houses, character animator)
41. Motionbuilder (3D character animation)
42. Mova (Face motion capture)
43. MultigenPro (Realtime 3D content for simulation, games, and other applications)
44. Natural Motion (3D character animation)
45. Natural Scene Designer (Outdoor scence: trees, clouds, rocks, lakes, atmospheric effects)
46. OpenFX (Digital post production, animation, game development, film and broadcast)
47. OpenInventor (Cross-platform 3D graphics visualization and animation)
48. Poser (*3D-character animation and design tool)*
49. Qurtz Composer *(nodes wired together graphically for animation/simulation*)
50. Realflow (Fluids by calculating the interaction between particles)
51. RealiMation (Real time 3D interactive applications and Realsoft 3D (Architecture and modular design)
52. Simi Motioncap 3D (3D motion tracking)
53. Softimage (*Films, games*)
54. Strata Live 3D (*Print, Web-enabled applications, and interactive games*)
55. Terragen (Photorealistic results for landscape, special effects, art and recreation)
56. Tile Studio (Tile-based games)
57. TrueSpace (Advertisements, games, art works, and animated VR)
58. Vega Prime (*Visual and audio simulation, virtual reality, and general visualization applications*)
59. Visviva (artistic modeling, hypertext layout, special effects creation, and animations)
60. Vue (Creation, rendering and animation of natural scenery)

61. WorldBuilder (Land and water modeling and rendering)
62. WorldUp (Creating and animating various VR worlds)

Simulation/Game Engine Tools

1. 20-sim (Simulate the behavior of dynamic systems with graphics display)
2. 3D Grapher (Illustrating and solving complex mathematical equations in 2D and 3D)
3. ActiveWorlds (Networked envrionment with avatars)
4. *Anark* (CAD or game modeling and simulation)
5. DIVE (multi-user VR system, navigate, see, meet, and interact with other users)
6. *Crystal Space* (*A software development kit for realtime 3D games or other simulation-like visualization*)
7. EON Studio (marketing, product development, simulation training, architectural studies)
8. Genesis3D (Real-time game engine)
9. GL Studio (Simulation and Training VR environment)
10. GL4Java (MapsOpenGL and GLU APIs to Java)
11. Glu3D (Fluid simulation)
12. Java 3D (*Extension to java for displaying 3d graphics and models*)
13. *Jet 3D* (*Graphics engine built for high performance real-time rendering*)
14. LithTech (LDS) (Game engine and graphics toolkit)
15. LS-DYNA (Car design and behavior simulation in a collision)
16. Mathematica (Numeric and symbolic calculation, visualization, and simulation)
17. Massive (crowd-related visual effects for film and television)
18. *Matlab* (*Data analysis/visualization, numeric/symbolic calculation, simulation*)
19. Maya (Commercials, character animation, and virtual reality)
20. Mirai (3D game development houses, character animator)
21. NetImmerse (Game engine)
22. OpenGVS (Scene manager)
23. Plot 3D (Simulation and visualization of computational fluid dynamics solutions)
24. Poser (*3D-character animation and design tool*)
25. Qurtz Composer (*nodes wired together graphically for animation/simulation*)
26. Realflow (Fluids by calculating the interaction between particles)
27. RealiMation (Real time 3D interactive applications and environments)
28. Realsoft 3D (Architecture and modular design)
29. Simul8 (Planning, modeling, validation, animation, and simulation)
30. *Softimage* (*Films, games*)
31. SPECTOR (Optical analysis, design and simulation)

32. The 3D Gamemaker (Game engine)
33. Unity (*Game engine and environment*)
34. *Unreal Engine* (*3D game engine/simulation and contents*)
35. Vega Prime (*Visual and audio simulation, virtual reality, and general visualization applications*)
36. WorldToolKit (Cross- platform real-time 3D VR development tool)
37. *World Up* (*Creating and animating various VR worlds*)
38. *XNA* (*Game development library and IDE built on top of Direct X*)

Virtual Reality Tools

1. ActiveWorlds (Networked environment with avatars)
2. Alice (Scripting and prototyping environment for 3D object behavior)
3. *Coin3D* (Scenegraph and OpenGL)
4. DIVE (multi-user navigate, meet, and interact with other users and applications)
5. DIVERSE (Common user interface and API to VE programs and VR hardware)
6. *Java 3D* (*Extension to java for displaying 3d graphics and models*)
7. Jet 3D (Graphics engine built for high performance real-time rendering)
8. LumeTools (Creating landscape, water, light, matter, and workbench)
9. Maya (Commercials, character animation, and virtual reality)
10. Summit 3D (Complex virtual worlds)
11. *Tucan Series* (*Real-time visualization and virtual reality*)
12. VREK (Crating functional, interactive and immersive environments)
13. VEGA (Visual and audio simulation, virtual reality, and general visualization applications)
14. VRML (Cross-platform 3D graphics visualization and animation applications)
15. WorldToolKit (Cross- platform real-time 3D VR development tool)
16. WorldUp (Creating and animating various VR worlds)

Web 3D Tools

1. 3D Instant Website (Publish live web pages with 3D interactive content)
2. 3D Text Maker (Build 3D text for web pages)
3. *ADG Panorama Tools* (*From a series of photos to 360 degrees interactive on the Web*)
4. Anfy3D (3D image on the web)
5. Blaze 3D Studio (*Interactive web 3D rendering*)
6. Brand Worlds Tools (*3D characters on the web*)
7. CosmoWorlds (VRML modeling, rendering, and animation)

8. Crystal 3Dimpact! Pro (web 3D graphics and animations with over 145 pre-built 3D objects)
9. CrystalGraphics PowerPlugs (PowerPoint and web 3D animation pulgins)
10. Cult3D (Create interactive objects on the Web)
11. Director 8.5 Shockwave Studio (3D entertainment, demonstrations and online learning applications)
12. EZ-Motion (3D or 2D image and animation software for web developers)
13. Insta 3D Pro (3D text and charts, graphics broadcasting and video editing)
14. Internet Character Animator (Animate 3D characters)
15. Internet Scene Assembler (VRML authoring tool)
16. Internet Space Builder (Establish virtual exhibitions and galleries)
17. *iSpace* (*web graphics assembly tool*)
18. Java 3D (Extension to java for displaying 3D graphics and models)
19. Navigram Planner (3D interior planning, design and configuration)
20. OpenWorlds (Add VRML/X3D functionality onto C++ and Java applications)
21. Pixel3D (Logo modeling and rendering)
22. ProPak3D (3D website concept, Design and development)
23. Shout3D (Let any standard web browser display interactive 3D graphics)
24. Swift 3D (Web Design: creating, editing, and animating 3D images)
25. *Ulead Cool 3D* (*Animated 3D titles and special effects for Web and video*)
26. Virtools (Interactive 3D experience)
27. Visviva (artistic modeling, hypertext layout, special effects creation, and sophisticated animations)
28. VRML (Cross-platform 3D graphics visualization and animation applications)
29. WebSphere Studio (e-business applications)
30. Xara3D (3D titles and logos for use on Web pages)

3D File-Format Converters

1. 3D Win (3D file converter)
2. AccuTrans 3D (3D file converter)
3. Biturn (3D file converter)
4. Guru 3D-Converter (3D Studio files to the DirectX files)
5. Materialize3D (Model converter, material / texture editor and polygon processor)
6. NuGraf (Scene composition, model viewing and model translation)
7. PolyTrans (*Import/export 3D files*)
8. Quick3D (Viewing, organizing, and converting 3D files)
9. Swift 3D (Web Design: creating, editing, and animating 3D images)

Appendix II

3D Graphics Models for Virtual Manufacturing

In virtual manufacturing organization, manufacturing can be viewed in 3D including movement of machines, tools and part by using Microsoft's DirectX 3D technology and TrueVision 3D rendering engine. In the following sections these steps are defined for creating virtual factory in 3D space.

Following steps are involved:

1. Creating 3D models of machines and tools in a graphic modeler
2. Converting 3D models into X file format suitable for DirectX and TrueVision 3D Engine.
3. Loading X format files in C#.net program as Mesh object and setting its position.
4. Controlling movement and animation of 3D objects.
5. Setting up Camera and 3D View

Creating 3D Models of Machines Using a Graphic Modeler

Any graphic modeler such as CATIA, Pro Engineer, IDEAS, 3D Studio Max, Maya and others can be used to generate the graphic model of the machine. Specific conditions applies for use of these individual graphic modelers. Autodesk 3ds Max, formerly 3D Studio MAX, is a modeling, animation and rendering package developed by Autodesk Media and Entertainment. It has modeling capabilities, a flexible plugin architecture and is able to be used on the Microsoft Windows platform. All 3D objects are created in 3ds Max. Pictures as shown in Figures in A.1, A.2, A.3 are taken during the creations of 3D graphic models of the machines.

Converting 3D Models into X File Format Suitable for DirectX and TrueVision 3D Engine

After creating 3D models in 3ds Max we can convert into direct X format by using Panda Exporter. Panda exporter can be downloaded from http://www.andytather.co.uk/Panda/directxmax_downloads.aspx. After download Unzip and extract the .dle file, place in the 'Plugins' sub-directory under the main 3dsmax folder, restart 3ds. After restarting 3ds Max on file menu select export then select Panda Exporter to export 3D model into X format.

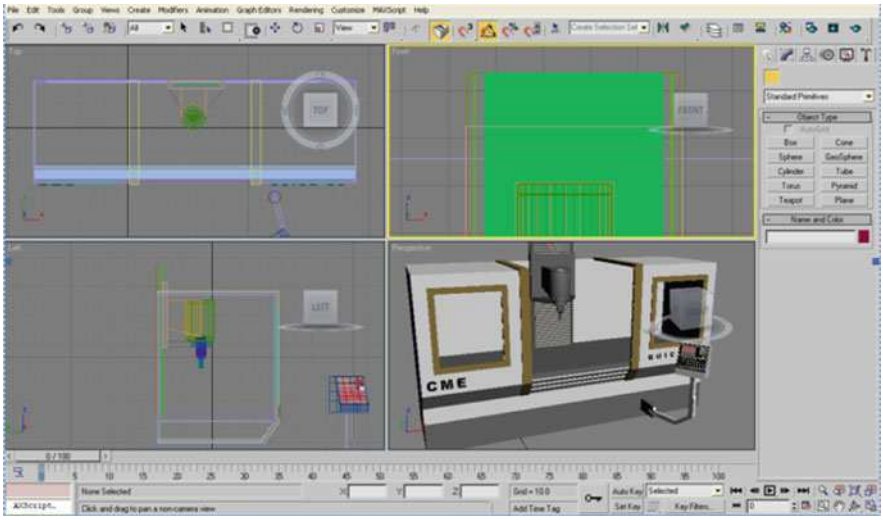


Fig. A.1 3D model of milling machine

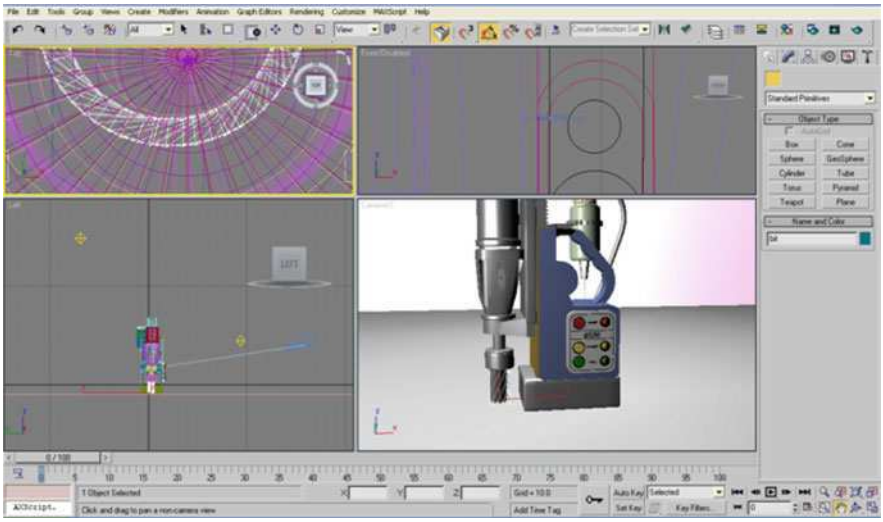


Fig. A.2 3D model of drilling machine

Loading X Format Files in C#.Net Program as Mesh Object

After installation of DirectX and TrueVision 3D SDK, make sure that TrueVision 3D API is referenced in C#.net project as shown in the Fig. A.4.

After adding references to virtual factory project, dynamic libraries of TrueVision 3D are shown in solution explorer as seen in Fig. A.5

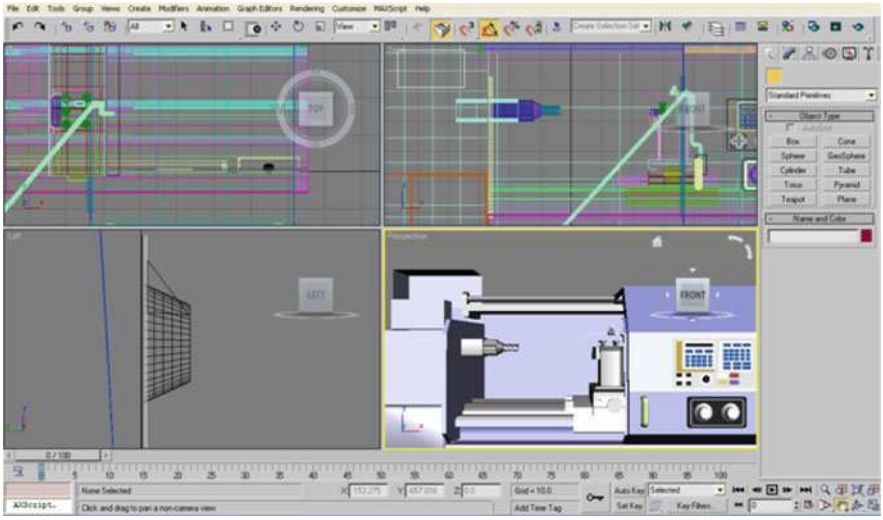


Fig. A.3 3D model of turning machine

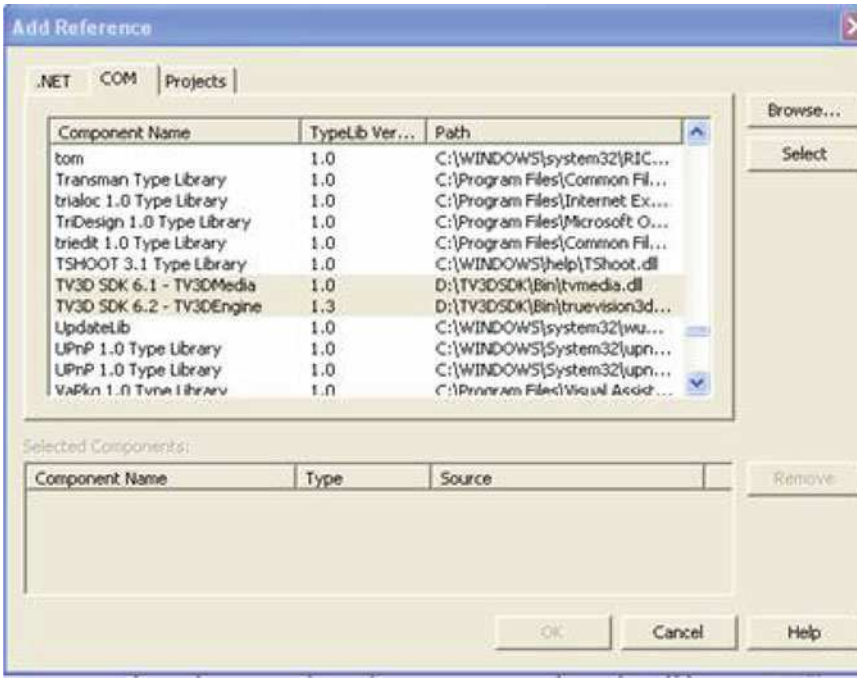


Fig. A.4 Adding the TrueVision Libraries in the Project

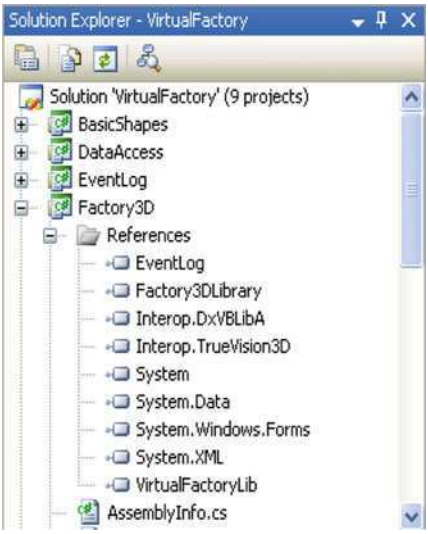


Fig. A.5 TrueVision 3D reference in virtual factory

Include these libraries in your project:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using TrueVision3D;
using DxVBLibA;
```

TV3DEngine is written using *DirectX* and provides powerful functions to accomplish 3D abstraction. It wraps some of the more powerful features of *DirectX* into easily used functions and procedures. It has some built-in classes those need to be loaded in the project; for example, TVEngine, TVInputEngine, TVLightEngine, TVTextureFactory, TVMaterialFactory, TVScene, TVLandscape, TVActor, TVCamera, and so forth. Some of the necessary classes used in this application will be discussed along with the code.

Declare and initialize TrueVision objects in the Form1 class:

```
//Declaring the TrueVision Engine object
TVEngine TV = new TVEngine();
//Declaring the Input Engine variable
TVInputEngine Inp;
//Declaring the Light Engine variable
TVLightEngine TVLightEngine;
//Declaring the Material Factory variable
TVMaterialFactory MatFactory;
//Declaring the TVScene object
public TVScene TVScene = new TVScene();
//Declaring the Global variable
TVGlobals global;
//Declaring the mesh class for the factory floor
TVMeshClass FactoryFloor;
//Declaring the light variable
DxVBLibA.D3DLIGHT8 light;
```

TVEngine is the main TrueVision class used to create a TV object that is just similar to an Application Window on which the whole rendering is to be done. This object is used to set the application properties, such as Window Mode, Search Directory Path, Window Title, and so on. TVScene is the portion on the window or whole of the window where everything is rendered. TVInputEngine is for handling all the keystroke and mouse inputs. TVGlobals handles the global engine built-in variables. TVLightEngine handles the lights in the scene and the other two handle all texture bindings and material mapping on the mesh objects respectively.

Declaring variables for keyboard and mouse movements:

```
// For keyboard input
float sngPositionX;
float sngPositionY;
float sngPositionZ;
float snglookatX;
float snglookatY;
float snglookatZ;

// Now, for the mouse input...
int tmpMouseX,tmpMouseY;
short tmpMouseB1,tmpMouseB2,tmpMouseB3;
int tmpMouseScrollOld,tmpMouseScrollNew;

float sngAngleX;


float sngAngleY;


// We want smoothing to the movement se we need to
declare two additional variables.
float sngWalk;
float sngStrafe;

// For controlling the rendering loop


public bool DoLoop = true;
```

Constructor of 3D Rendering Engine:

```
            /// <summary>
            /// Constructor of 3D Rendering Engine
        /// </summary>
        /// <param name="RenderingObject">PictureBox</param>
            public       Factory3D(System.Windows.Forms.PictureBox
RenderingObject)
            {
                  //  Specifying  the  rendering  window  as  the
picturebox object
                  InitializeComponent(RenderingObject);
            }


private    void    InitializeComponent(System.Windows.Forms.PictureBox
ToRender)
            {
                  try
                  {
                        //   Initializing   all   the   TrueVision
objects
                        TVLightEngine = new TVLightEngine();
                        MatFactory = new TVMaterialFactory();
                        Inp = new TVInputEngine();
                        global = new TVGlobals();

                        // Defining the model path for loading
images
```

```
                                ModelPath                                   =
System.Windows.Forms.Application.StartupPath+@"\Model\Images\";
                                // Setting the Angle System to Degrees

        TV.SetAngleSystem(CONST_TV_ANGLE.TV_ANGLE_DEGREE);
                                // Initializing the 3D Window Rendering
mode

        TV.Init3DWindowedMode(ToRender.Handle.ToInt32(), true);
                                // Defining the Directory Search path
for the TV objects

        TV.SetSearchDirectory(System.Windows.Forms.Application.Execut
ablePath);
                                // Setting the View Frustum
                                TVScene.SetViewFrustum(60.0f,1000.0f);

                                //  Setting the camera location and
lookat point etc
                                sngPositionX = 0.0f;
                                sngPositionY = 0.0f;
                                sngPositionZ = 0.0f;
                                snglookatX = 0.0f;
                                snglookatY = 0.0f;
                                snglookatZ = 0.0f;
                                sngAngleX = 0.0f;
                                sngAngleY = 0.0f;

                                // Setting the properties of Light
                                light.Type                                  =
DxVBLibA.CONST_D3DLIGHTTYPE.D3DLIGHT_POINT;
                                light.Direction                             =
global.Vector3(0.0f,1.0f,0.0f);
                                light.Position  =  global.Vector3(0.0f,
400.0f, 10.0f);
                                light.Range = 1000.0f;

                                light.Attenuation0 = 0.5f;
                                light.Falloff = 1;
                                light.Phi = 3.14f / 4.0f;
                                light.Theta = light.Phi / 2;

                                //   Setting   the   light   color   and
intensity
                                light.ambient.a = 0.8f;
                                light.ambient.r = 0.8f;
                                light.ambient.g = 0.8f;
                                light.ambient.b = 0.8f;

                                light.diffuse.a = 1.0f;
                                light.diffuse.r = 1.0f;
                                light.diffuse.g = 1.0f;
                                light.diffuse.b = 1.0f;

                                light.specular.a = 1f;
                                light.specular.r = 0.02f;
                                light.specular.g = 0.02f;
                                light.specular.b = 0.02f;
```

```
                                // Creating the Light Engine
                                TVLightEngine.CreateLight(ref
light,"light",false);

                                // Enable specular light
                                TVScene.SetSpecularLightning(true);

                                // Add a texture bitmap to the TVScene

        TVScene.LoadTexture(ModelPath+"wall.bmp",-1,-
1,"FactoryFloorTexture");
                                // Create the FactoryFloor mesh object
                                FactoryFloor                          =
(TVMeshClass)TVScene.CreateMeshBuilder("FactoryFloormesh");

// Add walls to our FactoryFloor object.
// The first principle is : that a wall is a rectangle. The
// second one is : the rectangle that defines the wall is created
// by using 2 vectors. The third one is : the 1st vector needs to
// define the nearest part of the wall and the 2nd vector needs to
// define the farthest part of the wall. The fourth one is : the wall
// coordinates are seen from a top down view.


        FactoryFloor.AddWall3D(global.GetTex("FactoryFloorTexture"),
650.0f, -650.0f, -750.0f, -650.0f, 350.0f, 5.0f, true, false, -50.0f,
5.0f, 1.0f);

        FactoryFloor.AddWall3D(global.GetTex("FactoryFloorTexture"),
-750.0f, -650.0f, -750.0f, 350.0f, 350.0f, 5.0f, true, false, -50.0f,
5.0f, 1.0f);

        FactoryFloor.AddWall3D(global.GetTex("FactoryFloorTexture"),
350.0f, 350.0f, -750.0f, 350.0f, 350.0f, 5.0f, true, false, -50.0f,
5.0f, 1.0f);

        FactoryFloor.AddWall3D(global.GetTex("FactoryFloorTexture"),
350.0f, 650.0f, 350.0f, -650.0f, 350.0f, 5.0f, true, false, -50.0f,
5.0f, 1.0f);

                                // Like the "add wall" method, we will
add a floor to this FactoryFloor.

        FactoryFloor.AddFloor(global.GetTex("FactoryFloorTexture"),
350.0f, -650-0f, -750.0f, 350.0f, -50.0f, 10.0f, 10.0f,true,false);

        FactoryFloor.AddFloor(global.GetTex("FactoryFloorTexture"),
350.0f, -650-0f, -750.0f, 350.0f, 300.0f, 10.0f, 10.0f,true,false);

                                // Setting the position of the Factory
Floor
                                FactoryFloor.SetPosition(floor_x,
floor_y, floor_z);

                        // Rendering is enabled
                                DoLoop = true;
```

```
                        }

                        catch(Exception exp)
                        {
                                Log.Write("Virtual
Factory","Factory3D.cs:InitializeComponent",exp.Message   +   "   "   +
exp.Source,EventLogEntryType.Error,8173);

                        }
                }

/// <summary>
        /// Main Rendering Loop
        /// </summary>
                private void Main_Loop()
                {
                        try
                        {
                                TVScene.SetSceneBackGround(0.5f,   0.5f,
0.7f);
        // Okay, we start the main loop here. We are going to loop
over
        // and over until the user click on the "Quit" button and by
this,
        // change the value of DoLoop to false.
        // We loop all of this until the DoLoop isn't True.
                                while(DoLoop == true)
                                {
                                        Check_Input();         ///Check
for any input
                                        Check_Movement();
        ///Check for movements
                                        Rendering();
                                }
                        }
                        catch(Exception exp)
                        {
                                Log.Write("Virtual
Factory","Factory3D.cs:Main_Loop",exp.Message     +     "     "     +
exp.Source,EventLogEntryType.Error,9307);
                        }
                }

/// <summary>
        /// 3D Rendering
        /// </summary>
                private void Rendering()
                {
                        TV.Clear(false);
                        TVScene.RenderAllMeshes(true);
                        //  Then,  we  display  everything  that  we  have
rendered (from the
                        // objects of the world) to the screen.
                        TV.RenderToScreen();

                }
```

Now every thing is ready to load x files for 3D machines and tools. In Factory3D each machine and work piece has separate function to create that 3D object. Such as CreateMilling(), CreateSawing(), CreateRobot() and so on. Only one machine is discussed in more detail. The following code is from

CreateMilling(). CreateMilling() takes name, direction and location x axis, location y axis and return 3D milling machine to calling method.

```
/// <summary>
/// Creates 3D Milling Machine
/// </summary>
/// <param name="X">float</param>
/// <param name="Y">float</param>
/// <param name="MachineName">string</param>
/// <param name="MachineDirection">Direction</param>
/// <returns>Milling3D</returns>
              public  Milling3D  CreateMilling(float  X,  float  Y,
string MachineName, Direction MachineDirection)
              {
                     try
                     {
                            D3DVECTOR position = new D3DVECTOR();
                            position.x = X;
                            position.y = Y;
                            position.z = 0;
                            Milling3D     objMilling     =     new
Milling3D(TVScene,position,MachineName,MachineDirection);
                            return objMilling;
                     }
                     catch(Exception exp)
                     {
                            Log.Write("Virtual
Factory","Factory3D.cs:CreateMilling",exp.Message    +    "    "    +
exp.Source,EventLogEntryType.Error,9831);
                     }
                     return null;
              }
```

CreateMilling() creates the object of Milling3D. In Milling3D class it actually loads the X files of milling machines. In the following programming code is from Milling3D.

Variables of Milling3D

```
              //Declaring the Global variable
              TVGlobals global;
             // Variable for storing model path
              string ModelPath;
             // Parts of Machine
              TVMesh MeshMachine;
              //TVMesh MeshWorkPiece;
              TVMesh MeshWorkBase;
              //TVKeyFrameAnim AnimationClass;
             private D3DVECTOR WorkBasePosition;
             private D3DVECTOR gWorkBasePosition;
             // Initial position for the upper machine
              float MachineX = 300.0f;
              float MachineY = -52.0f;
              float MachineZ = 330.0f;
             // Initial position for the work piece
              float WorkBaseX = 300.0f;
             // float WorkBaseY = -52.0f;
              float WorkBaseZ = 350.0f;
             //Machine Name
              private string sMachineName = string.Empty;
              //Machine Direction
              private Direction mDirection;
```

In constructor of Milling3D we pass TVScene Scene. Every thing that we load into Scene object will be rendered in 3D world. The following programming code is from Milling3D constructor.

```
  public   Milling3D(TVScene   Scene,   D3DVECTOR   position,   string
MachineName, Direction MachineDirection)
                {
                        sMachineName = MachineName;
                        mDirection = MachineDirection;
                        InitializeComponent(Scene, position);
                }
```

Constructor calls InitializeComponent (Scene, Position) method:

```
/// <summary>
/// Initialize Componet
/// </summary>
/// <param name="Scene">TVScene</param>
/// <param name="position">D3DVECOTR</param>
                private    void    InitializeComponent(TVScene    Scene,
D3DVECTOR position)
                {
                        try
                        {
                                // Initialize the global variable
                                global = new TVGlobals();
                                // Defining the model path for loading
X files
                                ModelPath                                =
System.Windows.Forms.Application.StartupPath+@"\Model\Meshes\Milling\
MillingXFile\";

                                // Initialize the position vector for
location
                                MachineX = MachineX - position.y;
                                MachineZ = MachineZ - position.x;

                                WorkBaseX = WorkBaseX - position.y;
                                WorkBaseZ = WorkBaseZ - position.x;


                                // Initialize the Machine meshes
                                MeshMachine = new TVMesh();
                                MeshWorkBase = new TVMesh();
                                // Create Meshes in the scene
                                MeshMachine                              =
Scene.CreateMeshBuilder("MeshMachine");
                                MeshWorkBase                             =
Scene.CreateMeshBuilder("MeshWorkBase");

                                //Now, finally load the machine
                                Load_Milling();
                        }
                        catch(Exception exp)
                        {
                                Debug.WriteLine(exp.Message              +
exp.Source);

                        }
                }
```

```
/// <summary>
/// Load 3D Milling Machine
/// </summary>
                private void Load_Milling()
                {
                        // Load Machine  Model
                        MeshMachine.LoadXFile(ModelPath+"machine.x",
true, true);
        MeshMachine.SetPosition(MachineX,MachineY,MachineZ);
                        MeshMachine.ScaleMesh(0.3f,0.3f,0.3f);

                        // Load Work Base Model

        MeshWorkBase.LoadXFile(ModelPath+"work_piece_base.x",    true,
true);
                        MeshMachine.AddChild(ref MeshWorkBase);

                        switch(mDirection)
                        {
                                case Direction.Front:
                                        MeshMachine.RotateY(90f,true);
                                        break;
                                case Direction.Back:
                                        MeshMachine.RotateY(-90f,true);
                                        break;
                                case Direction.Right:
                                        MeshMachine.RotateY(0f,true);
                                        break;
                                case Direction.Left:
                                        MeshMachine.RotateY(180f,true);
                                        break;

                        }

                        WorkBasePosition = MeshWorkBase.GetPosition();
                        gWorkBasePosition                              =
MeshWorkBase.GetPosition();

                }
```

In the above method InitializeComponent() creates Mesh for each 3D object and initialized that Mesh object into Scene object. In method it loads machine's X file in Mesh object. The following lines of code show the main concept.

```
MeshMachine = new TVMesh();
MeshMachine = Scene.CreateMeshBuilder("MeshMachine");
MeshMachine.LoadXFile(ModelPath+"machine.x", true, true);
```

3D models are the mesh objects that are to be placed in the 3D scene. A mesh is a series of polygons grouped to form a surface. They can include hall, terrain, roads, factory, machines, cars, people, and the like. Design the 3D mesh objects (animated or unanimated) in any 3D graphics software available (for example, 3D Studio Max, Maya, Light Wave, and so on) and convert them into a format that can be imported in .NET (as in the .x file format for DirectX).

Controlling Movement and Animation of 3D Object

The following codes are from Milling3D to show how to control the movement of machine and its parts.

```
/// <summary>
        /// Move Machine Parts
        /// </summary>
        /// <param name="Coordinate">MachineAxis</param>
        /// <param name="Value">int</param>
               public void Move(MachineAxis Coordinate, int Value)
               {
                       switch(Coordinate)
                       {
                              case MachineAxis.ZPlus:
                                     MoveZ(Coordinate,Value);
                                     break;
                              case MachineAxis.ZMinus:
                                     MoveZ(Coordinate,Value);
                                     break;
                              case MachineAxis.XPlus:
                                     MoveX(Coordinate,Value);
                                     break;
                              case MachineAxis.XMinus:
                                     MoveX(Coordinate,Value);
                                     break;
                              case MachineAxis.YPlus:
                                     MoveY(Coordinate,Value);
                                     break;
                              case MachineAxis.YMinus:
                                     MoveY(Coordinate,Value);
                                     break;
                       }
               }
               /// <summary>
        /// Control Movement of Y Axis
        /// </summary>
        /// <param name="Coordinate">MachineAxis</param>
        /// <param name="Value">int</param>
               private void MoveY(MachineAxis Coordinate, int Value)
               {
                       if(Coordinate == MachineAxis.YPlus)
                       {
                              Value = Value / 8;
                              for(int i = 0; i <= Value; i++)
                              {
                                     WorkBasePosition               =
MeshWorkBase.GetPosition();
                                     if(WorkBasePosition.z < 8)
                                     {

       MeshWorkBase.SetPosition(WorkBasePosition.x,WorkBasePosition.
y,WorkBasePosition.z + 0.15f);
                                     }
                                     else
                                     {
                                            MessageBox.Show("Limit
Out in part program for Milling Machine in Y direction");
                                            break;
                                     }

                              }
```

```
                        }
                        else
                        {
                                if(Value == 0)
                                {
                                        WorkBasePosition                =
MeshWorkBase.GetPosition();
                                        for(;WorkBasePosition.z >= 0;)
                                        {

MeshWorkBase.SetPosition(WorkBasePosition.x,WorkBasePosition.y
,WorkBasePosition.z   - 0.15f);
                    }
                                }
                                else
                                {
                                        WorkBasePosition                =
MeshWorkBase.GetPosition();
                                        Value = Value / 8;
                                        for(int i = 0; i <= Value; i++)
                                        {
                                                WorkBasePosition        =
MeshWorkBase.GetPosition();
                                                if(WorkBasePosition.z   >
-18)
                                                {

        MeshWorkBase.SetPosition(WorkBasePosition.x,WorkBasePosition.
y ,WorkBasePosition.z   - 0.15f);
                                                }
                                                else
                                                {

        MessageBox.Show("Limit  Out  in  part  program  for  Milling
Machine in Y direction");
                                                        break;
                                                }


                                        }
                                }


                        }
                }
        /// <summary>
        /// Control Movement in X Axis
        /// </summary>
        /// <param name="Coordinate">MachineAxis</param>
        /// <param name="Value">int</param>
               private void MoveX(MachineAxis Coordinate, int Value)
               {
                        if(Coordinate == MachineAxis.XPlus)
                        {
                                Value = Value / 8;
                                for(int i = 0; i <= Value; i++)
                                {
```

```
                                        WorkBasePosition                =
MeshWorkBase.GetPosition();
                                        if(WorkBasePosition.x < 80)
                                        {

        MeshWorkBase.SetPosition(WorkBasePosition.x                     +
0.15f,WorkBasePosition.y,WorkBasePosition.z);
                                        }
                                        else
                                        {
                                                MessageBox.Show("Limit
Out in part program for Milling Machine in X direction");
                                                break;
                                        }

                                }

                        }
                        else
                        {
                                if(Value == 0)
                                {
                                        WorkBasePosition                =
MeshWorkBase.GetPosition();
                                        for(;WorkBasePosition.x >= 10;)
                                        {
                                                if(WorkBasePosition.x   >
-80)
                                                {

        MeshWorkBase.SetPosition(WorkBasePosition.x                     -
0.15f,WorkBasePosition.y ,WorkBasePosition.z);
                                                        WorkBasePosition
= MeshWorkBase.GetPosition();
                                                }
                                                else
                                                {

        MessageBox.Show("Limit  Out  in  part  program  for  Milling
Machine in X direction");
                                                        break;

                                                }

                                        }
                                }
                                else
                                {
                                        WorkBasePosition                =
MeshWorkBase.GetPosition();
                                        Value = Value / 8;
                                        for(int i = 0; i <= Value; i++)
                                        {
                                                WorkBasePosition        =
MeshWorkBase.GetPosition();
                                                if(WorkBasePosition.x   >
-80)
                                                {
```

```
        MeshWorkBase.SetPosition(WorkBasePosition.x                    -
0.15f,WorkBasePosition.y ,WorkBasePosition.z);
                                                }
                                                else
                                                {

        MessageBox.Show("Limit  Out  in  part  program  for  Milling
Machine in X direction");
                                                        break;
                                                }

                                        }
                                }

                        }
                }

        /// <summary>
        /// Control Movement in Z Axis
        /// </summary>
        /// <param name="Coordinate">MachineAxis</param>
        /// <param name="Value">int</param>
               private void MoveZ(MachineAxis Coordinate, int Value)
               {
                       if(Coordinate == MachineAxis.ZPlus)
                       {

                               for(int i = 0; i <= Value; i++)
                               {
                                       WorkBasePosition                =
MeshWorkBase.GetPosition();
                                       if(WorkBasePosition.y < 27 )
                                       {

        MeshWorkBase.SetPosition(WorkBasePosition.x,WorkBasePosition.
y + 0.15f,WorkBasePosition.z);
                                       }
                                       else
                                       {
                                               MessageBox.Show("Limit
Out in part program for Milling Machine in Z direction");
                                               break;
                                       }
                               }
                       }
                       else
                       {
                               if(Value == 0)
                               {
                                       WorkBasePosition                =
MeshWorkBase.GetPosition();
                                       for(;WorkBasePosition.y >= 5.0;)
                                       {
                                               if(WorkBasePosition.y   >
0)
                                               {
```

```
        MeshWorkBase.SetPosition(WorkBasePosition.x,WorkBasePosition.
y - 0.15f,WorkBasePosition.z);
                                                        WorkBasePosition
= MeshWorkBase.GetPosition();
                                                }
                                                else
                                                {

        MessageBox.Show("Limit  Out  in  part  program  for  Milling
Machine in Z direction");
                                                        break;
                                                }
                                        }
                                }
                        }
                }
```

Setting up Camera and 3D View

In virtual factory 3D world, camera and view can be control by using key board arrow keys. Camera can be set to any location and can be zoom to see closed view of machine movements and parts. Camera control coding is implemented in Check_Input() and Check_Movement() methods and both are continually calling by Main_Loop() method.

Check_Input() Method coding. Where we control the keyboard input for movement of camera in 3D world.

```
/// <summary>
        /// Control Key Bord Inbuts
        /// </summary>
                public void Check_Input()
                {
                        sngWalk = 0.0f;
                        sngStrafe = 0.0f;

                        // Check if we pressed the PAGEUP key, if so,
then scene moves in +y direction

        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_PAGEUP))
                        {
                                TVScene.SetCamera(sngPositionX,
sngPositionY+=0.5f,   sngPositionZ,   snglookatX,   snglookatY+=0.5f,
snglookatZ);
                        }
                        // Check if we pressed the PAGEDOWN key, if
so, then scene moves in -y direction


        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_PAGEDOWN
))
                        {
```

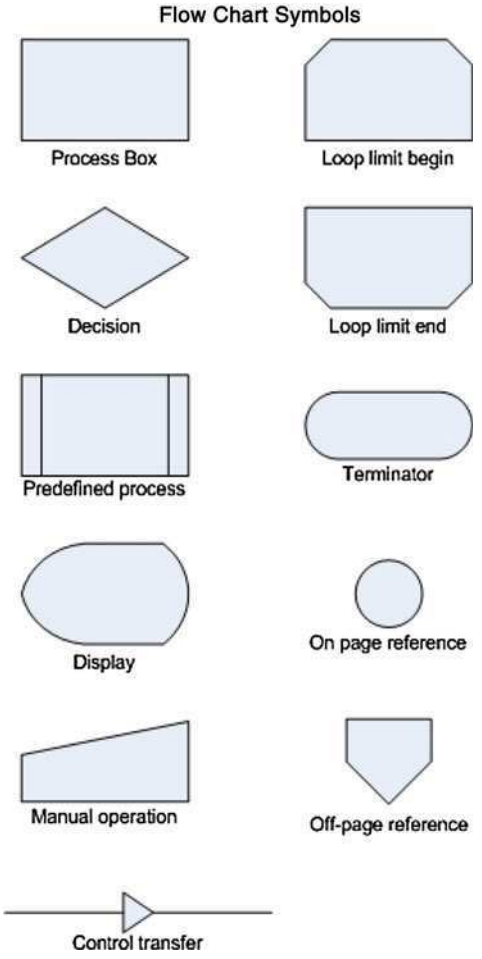
```
                                TVScene.SetCamera(sngPositionX,
sngPositionY-=0.5f,   sngPositionZ,   snglookatX,   snglookatY-=0.5f,
snglookatZ);
                        }

                        // Check if we pressed the UP arrow key, if
so, then we are
                        // walking forward.

        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_UP))
                        {
                                sngWalk = 0.1f;
                OnSetFocus(this);

                        }

                        // If we are not walking forward, maybe we are
walking backward
                        // by using the DOWN arrow? If so, set walk
speed to negative.

        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_DOWN))
                        {
                                sngWalk = -0.1f;
                OnSetFocus(this);
                        }

                        // Check if we pressed the LEFT arrow key, if
so, then strafe
                        // on the left.

        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_LEFT))
                        {
                                sngStrafe = 0.1f;
                OnSetFocus(this);

                        }

                        // If we are not strafing left, maybe we want
to strafe to the
                        // right, using the RIGHT arrow? If so, set
strafe to negative.

        if(Inp.IsKeyPressed(TrueVision3D.CONST_TV_KEY.TV_KEY_RIGHT))
                        {
                                sngStrafe = -0.1f;
                OnSetFocus(this);
                        }

                        // Actual value to old mouse scroller value.
                        tmpMouseScrollOld = tmpMouseScrollNew;


                }
```

Check_Movement() method coding. Where we control location of camera in 3D world:

```
/// <summary>
        /// Control Camera Movement
        /// </summary>
               public void Check_Movement()
               {

                       float TimeElapsed;
                       try
                       {

                               TimeElapsed = TV.TimeElapsed();
                               //TVScene.DrawText("Time:"
+TimeElapsed,100,200,global.RGBA(1.0f,1.0f,1.0f,1.0f));

                               // Simple check of the mouse.
                               if(sngAngleX > 1.5f)
                               {
                                       sngAngleX = 1.5f;
                               }

                               if(sngAngleX < -1.5)
                               {
                                       sngAngleX = -1.5f;
                               }

                               // Update the vectors using the angles
and positions.
                               sngPositionX     =     sngPositionX     +
(float)(System.Math.Cos((double)sngAngleY)   *   sngWalk   /   5.0f   *
TimeElapsed) + (float)(System.Math.Cos((double)sngAngleY + 3.141596 /
2) *  sngStrafe/ 5.0f * TimeElapsed);
                               sngPositionZ     =     sngPositionZ     +
(float)(System.Math.Sin((double)sngAngleY)   *   sngWalk   /   5.0f   *
TimeElapsed) + (float)(System.Math.Sin((double)sngAngleY + 3.141596 /
2) *  sngStrafe/ 5.0f * TimeElapsed);

                               // We update the look at position.
                               snglookatX     =     sngPositionX     +
(float)System.Math.Cos((double)sngAngleY);
                               snglookatY     =     sngPositionY     +
(float)System.Math.Tan((double)sngAngleX);
                               snglookatZ     =     sngPositionZ     +
(float)System.Math.Sin((double)sngAngleY);

               // With the new values of the camera vectors (position and
               // look at), we update the scene's camera.

               //Limits of Camera movements with walls. Camera can not cross
any wall
                               if(sngPositionZ  >  328  ||  snglookatZ  >
328)
                               {
                                       sngPositionZ = 328;
                                       snglookatZ = 328;

                               }
```

```
                                else    if(sngPositionZ    <    -635    ||
snglookatZ < -635)
                                {
                                        sngPositionZ = -635;
                                        snglookatZ = -635;
                                }
                                if(sngPositionX  >  287  ||  snglookatX  >
288)
                                {
                                        sngPositionX = 287;
                                        snglookatX = 288;
                                }
                                else    if(sngPositionX    <    -648    ||
snglookatX < -647)
                                {
                                        sngPositionX = -648;
                                        snglookatX = -647;
                                }
                                if(sngPositionY  >  288  ||  snglookatY  >
288)
                                {
                                        sngPositionY = 288;
                                        snglookatY = 288;
                                }
                                else    if(sngPositionY    <    -41    ||
snglookatY < -41)
                                {
                                        sngPositionY = -41;
                                        snglookatY = -41;
                                }

                                TVScene.SetCamera(sngPositionX,
sngPositionY, sngPositionZ, snglookatX, snglookatY, snglookatZ);

                        }
                        catch(Exception exp)
                        {
                                Log.Write("Virtual
Factory","Factory3D.cs:CheckMovement",exp.Message    +    "    "    +
exp.Source,EventLogEntryType.Error,2920);
                        }
                }
```

Appendix III



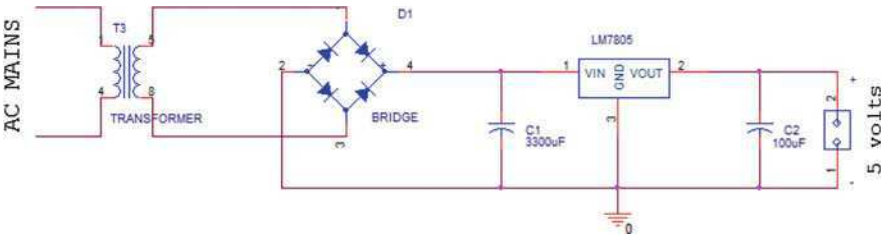
Appendix IV

Data Sheets

1 Universal LVDT Signal Conditioner AD698
http://www.analog.com/static/imported-files/data_sheets/AD698.pdf
2. CD4066BC Quad Bilateral Switch
http://www.fairchildsemi.com/ds/CD%2FCD4066BC.pdf
3. ENC28J60 Data Sheet Stand-Alone Ethernet Controller with SPI Interface
http://ww1.microchip.com/downloads/en/devicedoc/39662a.pdf
4. LCD-016M002B 16 x 2 Character LCD
http://www.datasheetcatalog.org/datasheet/vishay/016m002b.pdf
5. LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator
http://www.fairchildsemi.com/ds/LM%2FLM7810.pdf
6. PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers
http://ww1.microchip.com/downloads/en/devicedoc/30292c.pdf
7. TC426/TC427/TC428 1.5A Dual High-Speed Power MOSFET Drivers
http://www.datasheetcatalog.org/datasheet/microchip/21415b.pdf

Appendix V

Power Supply



Index