

**DEVELOPING AN ARCHITECTURE FOR THE PRODUCTION OF A GENERIC
MODEL OF THE LOCAL MANUFACTURING INDUSTRIES**

Chiew Ling King

A Thesis submitted
in fulfilment of the requirements for the degree of
Master of Science in Information Technology

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK

2006

ACKNOWLEDGEMENT

I would like to sincerely thank my supervisor, Dr. Rosziati Ibrahim, for her patient supervision and encouragement throughout this research. Without her, I would not have an opportunity to have a breakthrough in my academic level. I would like to thank all who have given me support and advice during my research. I would like to thank the Faculty of Computer Science and Information Technology for providing an environment that is suitable for doing my research. Lastly, I want to acknowledge Pelan Tindakan Pembangunan Teknologi Perindustrian (PTPTP) for the financial support.

ABSTRACT

The primary purpose of this study is to devise the architecture of a system that can develop a generic model of the restricted local small to medium sized manufacturing industries. With this architecture, a modeling tool or demonstrator system can be developed to capture the process flow of the manufacturing industries. This demonstrator system is composed of three major components which are in the form of a graphical user interface. These components are modeling tools, workplace and code generators. The modeling tools consist of a Domain Process (DP) structure, a Process (P) structure, an Activity (A) structure and a Relation structure which are used to represent different processes and relationships. This modeling tool is used to capture the process flow of a manufacturing industry. The workplace provides the area for the modeling structures to be constructed. This provides a structured view of the process flow of the manufacturing industry. The Code generator is the engine that is used to generate the coding templates from the model created on the workplace. With these coding templates, processes for the customization of the functions allow the software system to be developed according to the requirements of the modeler.

This research adopts a generic model from MIICI (Manufacturing Industry Information and Command Infrastructure). However, MIICI presents a generic model of a manufacturing industry in a mathematical form. Mathematical representations are not common to Malaysian manufacturing industries and they are also difficult to understand by the majority of people. Therefore, this research uses Domain Process, Process and Activity structures to produce the architecture of a system that can develop a generic model of the

restricted local small to medium sized manufacturing industries. From there, a software system can be constructed from the model of the application domain. Furthermore, the developed model can serve as a media of communication for individuals from same functions, fields and disciplines.

ABSTRAK

Matlamat utama kajian ini adalah untuk menghasilkan senibina sistem yang boleh membangunkan model generik sesuatu industri perkilangan bersaiz kecil ke sederhana tempatan yang telah ditetapkan. Dengan terhasilnya senibina ini, peralatan permodelan atau sistem demonstrator boleh dibangunkan untuk mendapatkan aliran proses dalam industri perkilangan. Sistem demonstrator ini mengandungi tiga komponen utama dalam bentuk antaramuka pengguna grafik. Komponen-komponen ini adalah peralatan permodelan, ruang kerja dan penjana kod. Peralatan permodelan ini mengandungi struktur proses domain (DP), struktur proses (P), struktur aktiviti (A) dan struktur pertalian yang digunakan untuk mewakili proses-proses dan pertalian-pertalian yang berbeza. Peralatan permodelan ini digunakan untuk mendapatkan aliran proses dalam industri perkilangan. Ruang kerja menyediakan ruang untuk pembinaan struktur-struktur permodelan. Ini seterusnya akan menyediakan pandangan struktur aliran proses dalam industri perkilangan. Penjana kod ialah enjin yang akan digunakan untuk menjana template kod daripada model yang dihasilkan oleh ruang kerja. Dengan template kod ini, proses-proses penyelarasan fungsi-fungsi akan membolehkan sistem perisian dibina mengikut kehendak jurumodel.

Kajian ini menggunakan model generik dari MIICI (Manufacturing Industry Information and Command Infrastructure). Walau bagaimanapun, MIICI mempersembahkan model generik industri perkilangan dalam bentuk matematik. Persembahan sedemikian tidak lazim di dalam industri perkilangan di Malaysia dan ia

juga agak sukar untuk difahami oleh kebanyakan orang. Oleh yang demikian, kajian ini menggunakan struktur proses domain, proses dan aktiviti untuk menghasilkan senibina sistem yang boleh membangunkan model generik sesuatu industri perkilangan bersaiz kecil ke sederhana tempatan yang telah ditetapkan. Kemudian sistem perisian akan dapat dibina daripada model yang diperolehi dari domain applikasi. Sebagai tambahan, model yang telah dibangunkan boleh berkhidmat sebagai media komunikasi untuk individu-individu daripada fungsi-fungsi, medan-medan dan disiplin-disiplin yang sama.

TABLE OF CONTENTS

Acknowledgement	ii
Abstract	iii
Abstrak	v
Table of Contents	vii
List of figures	xii
List of tables	xv
1 INTRODUCTION.....	1
1.1 Overview	1
1.2 Background and Justification of the Research	2.
1.3 Research Objectives	4
1.4 Research Scope	5
1.5 Research Tasks and Dissertation Outline	6
2 LITERATURE REVIEW	9
2.1 Introduction	9
2.2 Process Modeling	10
2.3 Plant Interflow Model	14
2.4 Architecture, Framework and Methodology.....	16
2.4.1 IDEF	18
2.4.1.1 IDEF0 Function Modeling	18
2.4.1.2 IDEF3 Process Description Capture Method	21
2.4.2 CIMOSA	23

2.4.3	GERAM	28
2.4.4	Object-Oriented Modeling	31
2.5	Comparison of reviewed Modeling Languages	35
2.6	Conclusion	37
3	OBJECT-ORIENTED REQUIREMENT ANALYSIS	38
3.1	Introduction	38
3.2	Domain Analysis	39
3.3	Modeling Tool Structure	41
3.3.1	Domain Process structure	43
3.3.2	Process structure	44
3.3.3	Activity structure	45
3.4	Relation Structure	46
3.5	The modeling tool components and activities	48
3.6	Use Case Driven Object Oriented Analysis	49
3.6.1	Activity Diagram of the Modeling Tool.....	50
3.6.2	Use Case Model of the Modeling Tool	53
3.6.2.1	Create Model Use Case	54
3.6.2.2	Edit Model Use Case	55
3.6.2.3	Generate Coding Templates Use Case	55
3.6.2.4	Finish Modeling Activities Use Case	56
3.6.2.5	Store Operation Use Case	56
3.6.2.6	Draw Process Flows Use Case	58

	3.6.3	Interaction Diagrams	67
	3.6.4	Class Diagram	68
	3.7	Conclusion	70
4		OBJECT-ORIENTED DESIGN OF MODELING TOOL	72
	4.1	Introduction.....	72
	4.2	Refine and complete the class diagram.....	73
	4.2.1	Refine Attributes.....	73
	4.2.2	Design Methods.....	73
	4.3	Design Access Layer.....	76
	4.4	Design View Layer.....	78
	4.5	Architecture of Modeling Tool	81
	4.6	Conclusion.....	86
5		IMPLEMENTATION OF MODELING TOOL APPLICATION	87
	5.1	Introduction	87
	5.2	Description of Modeling Tool Application	87
	5.2.1	Modeling Tool Overview	87
	5.2.2	Tools used for modeling.....	89
	5.2.2.1	Domain Process Structure Representation.....	90
	5.2.2.2	Process Structure Representation.....	91
	5.2.2.3	Activity Structure Representation.....	92
	5.2.2.4	Relation Structure Representation.....	93

	5.2.2.5 Text Representation.....	94
	5.2.3 Functionalities of Structure Controller.....	95
	5.2.3.1 Decomposition Interface Object.....	95
	5.2.3.2 Up Level Interface Object.....	97
	5.2.4 Code Generator and File System Interface Objects.....	99
	5.2.4.1 File System Interface Object	99
	5.2.4.2 Code Generator Interface Object	101
	5.3 Conclusion	102
6	APPLICATION OF MODELING TOOL AND TECHNIQUES	103
	6.1 Introduction.....	103
	6.2 Description of Manufacturing Process of FFM.....	103
	6.3 Define Scope and Bound Domain	105
	6.4 Identify and Specify Process Structure	107
	6.5 Identify and Specify Activity Structure	109
	6.6 Identify and Classify Relation	111
	6.7 Snapshots of the FFM Process Model	112
	6.8 Conclusion	120
7	CONCLUSION	121
	7.1 Introduction	121
	7.2 Contribution	122
	7.3 Recommendation for future research	123

BIBLIOGRAPHY	125
Appendix A	Sequence diagrams used for use cases defined 133
Appendix B	Activity diagrams for methods identified 139
Appendix C	Source Codes for the Application Developed for FFM to calculate total of flour and time taken from the input raw material. 147

LIST OF FIGURES

Figure 2.1:	IDEF0 Basic Structure Representations	19
Figure 2.2:	Decomposition Overview	20
Figure 2.3:	Schematic Symbols	22
Figure 2.4:	CIMOSA modeling framework	24
Figure 2.5:	First level details of Domain Process – Requirement Definition	26
Figure 2.6:	First level details of Domain Process – Design Specification	27
Figure 2.7:	GERAM framework components	30
Figure 3.1:	Enterprise as Collection of Business Processes	42
Figure 3.2:	Modeling Tool Structures	42
Figure 3.3:	Relations between Domain Processes	43
Figure 3.4:	Decomposition activities of Domain Process to Process	45
Figure 3.5:	Activity structure	46
Figure 3.6:	Relationship of a Relation Structure	47
Figure 3.7:	The modeling activities in using the modeling tool	48
Figure 3.8:	Activity Diagram of Modeling Process	52
Figure 3.9:	Use Case Model of the Modeling Tool	54
Figure 3.10:	Package of Store Operation.	57
Figure 3.11:	Package of Draw Process Flows	59
Figure 3.12:	Package of Select Specific Element	61
Figure 3.13:	Package of Use Attribute Functions	63
Figure 3.14:	Package of Use Navigation Operator	65

Figure 3.15: Sequence Diagram of Create Model Use Case 68

Figure 3.16: Class Diagram of Modeling Tool 69

Figure 4.1: Activity Diagram for CMTElement class GetBoundRect method 74

Figure 4.2: Refined class diagram by adding details to the attributes 75

Figure 4.3: CMoToolDoc Class Diagram used as Access Layer for object storage
and interoperability 77

Figure 4.4: CMoToolView class diagram that is used as view layer for interaction
between user and modeling process layer 80

Figure 4.5 Relationships of View Layer, Modeling Process Layer and Access Layer 81

Figure 4.6: Architecture of Modeling Tool 86

Figure 5.1: Overview of Modeling Tool Application 88

Figure 5.2: Part of the source code used to create the application frame window 89

Figure 5.3: Icon used to represent the Domain Process Structure 90

Figure 5.4: Class definition for the Domain Process object 91

Figure 5.5: Icon used to represent Process Structure 92

Figure 5.6: Icon used to represent Activity Structure 93

Figure 5.7: Icon used to represent Relation Structure 94

Figure 5.8: Icon used to represent Text 95

Figure 5.9: Structure Controller with the functionality of decomposition a process 96

Figure 5.10: Example of source code for decomposing method 97

Figure 5.11: Structure Controller with the functionality of moving to upper level 98

Figure 5.12: Example of source codes for Up Level functionality 99

Figure 5.13: Interface Objects used for File System 100

Figure 5.14:	Example of the source code for opening a file	100
Figure 5.15:	Interface object used to represent Code Generator	101
Figure 5.16:	Example of the source codes for Code Generator	102
Figure 6.1:	Relationship of the processes at different levels	112
Figure 6.2:	Domain Process Structure with the objective of the modeling at Top Level	113
Figure 6.3:	Process Structures created at Level 2 where their parent is Cleaning Domain Process Structure	114
Figure 6.4:	Activity Structure with its parent WheatWeight Process Structure	115
Figure 6.5:	Activity Structure with its parent Pre_Cleaning Process Structure	116
Figure 6.6:	Activity Structure with its parent Clean_Temper Process Structure	116
Figure 6.7:	Activity Structure with its parent Milling Domain Process created at Level 2	117
Figure 6.8:	Indication of the completion of generation coding templates	117
Figure 6.9:	Application developed from customization on coding templates Generated	118
Figure 6.10:	Class definition for Cleaning Domain Process (DP) structure	119

LIST OF TABLES

Table 2.1:	CIMOSA modeling constructs and their elements	25
Table 2.2:	Description of related terms in enterprise modeling	36
Table 2.3:	Comparison of different modeling Languages	36
Table 3.1:	Actor Actions and System Responses for Create Model Use Case	55
Table 3.2:	Actor Actions and System Responses for Edit Model Use Case	55
Table 3.3:	Actor Actions and System Responses for Generate Coding Template Use Case	56
Table 3.4:	Actor Actions and System Responses for Finish Modeling Activities Use Case	56
Table 3.5:	Actor Actions and System Responses for Save Operation Use Case	57
Table 3.6:	Actor Actions and System Responses for Save As Operation Use Case.	58
Table 3.7:	Actor Actions and System Responses for Draw Top Level Use Case.	59
Table 3.8:	Actor Actions and System Responses for Draw Middle Level Use Case.	60
Table 3.9:	Actor Actions and System Responses for Select Process Element Use Case	60
Table 3.10:	Actor Actions and System Responses for Select Activity Element Use Case	61
Table 3.11:	Actor Actions and System Responses for Draw Bottom Level Use Case	62
Table 3.12:	Actor Actions and System Responses for Use Modeling Elements Use Case	62
Table 3.13:	Actor Actions and System Responses for Use Relation Element Use Case	63

Table 3.14:	Actor Actions and System Responses for Add Attributes Use Case	64
Table 3.15:	Actor Actions and System Responses for Remove Attributes Use Case	64
Table 3.16:	Actor Actions and System Responses for Decompose Domain Process Use Case	65
Table 3.17:	Actor Actions and System Responses for Decompose Process Use Case	66
Table 3.18:	Actor Actions and System Responses for Return to Domain Process Use Case	66
Table 3.19:	Actor Actions and System Responses for Return to Process Use Case	66
Table 3.20:	Description of purposes of the classes identified at Object-Oriented Analysis	70
Table 6.1:	Descriptions of the functionalities of the Domain Process Structure Involved	107
Table 6.2:	Descriptions of the functionalities of the Process Structure involved in Cleaning Domain Process Structure	108
Table 6.3:	Descriptions of the functionalities of the Activity Structures involved in WheatWeight, Pre_Cleaning and Clean_Temper Process Structures	110
Table 6.4:	Descriptions of the functionalities of the Activity Structures involved in Milling Domain Process Structure	110
Table 6.5:	Results comparison between calculations made by FFM and Application ...	119

1 INTRODUCTION

1.1 Overview

This research presents a modeling tool that can be used to support operations of small to medium sized manufacturing industries. According to Small and Medium Industries Development Corporation (SMiDEC), manufacturing companies or companies providing manufacturing related services with annual sales turnover not exceeding RM25 million or with full-time employees not exceeding 150 in number, are considered to be categorized as small to medium sized manufacturing industries.

In the manufacturing industry, there are many elements and operations that can be modeled to reflect different aspects of the enterprise. Therefore, manufacturing modeling is a process of building models of whole or parts of a manufacturing industry such as process modeling, resource modeling, data modeling and others. Modeling different aspects of a manufacturing industry is to prevent presenting an overly complex model that covers all the aspects of the industry. Every complex system is best approached through a small set of nearly independent views of a model; no single view is sufficient. This dissertation presents a modeling tool that can capture the process flow of an interested domain of the manufacturing industry. Captured processes can be customized to develop an application that can cater the need of the interested domain. It also describes the methodology developed for the use of the modeling tool in developing the model for certain domain of a manufacturing industry.

1.2 Background and Justification of the Research

In a growing country like Malaysia, industrial development plays a very important role in moving the economy of the country. Thus, manufacturing industries must be prepared to meet the dynamic, competitive and rapid changing market in order to survive.

The Manufacturing Industry Information and Command Infrastructure (MIICI) project was initially brought by the International Institute for Software Technology in United Nations University (UNU/IIST) with the intention to produce an industrial system capable of producing results without consuming excessive resources and also able to make quick decisions intelligently (Jan Goossenaerts & Dines Bjorner, 1994). MIICI project emphasizes mathematical models of products, enterprises and businesses environments as primarily tools for creating lean/agile supply-based industrial systems, which are environmentally sustainable by utilizing advanced computation and communication technology. From MIICI, this research is trying to adopt the concepts from the generic model in order to provide a structural view of local manufacturing industry process flows through the development of a framework for a generic model. A generic model in MIICI is created through defining terms, constructing terminologies and a mathematical framework to express the interflow models that matter for manufacturing and trade. By formalizing and systematizing the mathematical framework, a particular model can be constructed with the interflow of the phenomena processes of a manufacturing industry. However, advancement of the information and communication technology and manufacturing industries increasingly rely on using the information and communication technology in the daily operations of the manufacturing

industry; urging the use of rigorous techniques in software development for manufacturing operation with the context of market and industry today. This is applied in MIICI, where requirements and necessary information are expressed in a mathematical model and software development can be mechanically derived from the model. Therefore, adopting the MIICI concepts to local manufacturing industries is of importance because by developing framework for a generic model of local manufacturing processes, software system can be constructed from the model of the application domain. Software development processes through customization of the functionalities of the software system can develop an application that is required by the manufacturing industry.

As is already known, MIICI presents a generic model of a manufacturing industry in a mathematical form. Mathematical representations are not common and are difficult to understand for the majority of people. Different symbols are used to represent different sets of information. Training needs to be provided for those who are interested in following the methodology. However, source for skilled and trained instructors are not many since this representations are created by MIICI itself. Besides that, MIICI software development for a manufacturing industry should be understood within the general methodology in which software systems are constructed on the basis of the mathematical models of their application domain (Goossenaerts & Bjorner, 1994). Thereby, knowledge of mathematical representations and manipulation is essential.

Therefore, developing a manufacturing model which is simple, less time consuming and less training is required for local small to medium sized industries. This is because; there is a constraint on budgets allocation for most local small to medium sized

industries on the advancement of information and communication technology. Modeling can involve in different areas of a manufacturing industry like resource modeling which mainly deals with resources handling and data modeling; which involves data flowing and manipulation. However, representing different information on one single model would present an ambiguous environment for a viewer to understand. Focus on modeling process is required. Thereby, the process flow of the manufacturing industry is taken into consideration. This is because a process can involve a set of activities performed within a specific order of times with defined input and produce output (Davenport, 1993). Then the sequence flow from one process to another process can reflect the operations happening inside a manufacturing industry.

1.3 Research Objectives

In order to come out with a modeling tool that can facilitate the development of the model for the local small to medium sized manufacturing industries, there are two objectives that have to be achieved. There are:

- To come up with the architecture of a system that can develop a generic model of the restricted small to medium sized manufacturing industries.
- To develop a demonstrator system based on the framework for the layout of the process flow of the restricted small to medium sized manufacturing industries in order to simulate the actions carried out by the processes.

The primary objective of this research is to come up with the architecture of a system that can develop a model of the restricted manufacturing industry. Architecture of a system is just like a “blue print” or “navigator” that assists in designing a system. It

also can serve as a guideline for the user of the system which has been developed. Therefore, with the architecture of the system, a demonstrator system which is based on the architecture can be designed.

The secondary objective is to develop a demonstrator system or modeling tool system which is used to capture the process flow of a manufacturing industry. This demonstrator system is composed of several components. These components are modeling tools, a workplace and code generator. The modeling tools consist of several components which are used to represent different processes and relationships. The workplace is the area where the modeling tools are to be placed. The code generator is the engine that is used to generate the coding templates from the model created on the workplace.

This modeling tool system can be used to develop a model by capturing the process flow of the restricted manufacturing industries. From here, a software system can be developed to simulate the actions carried out by the processes through customization process.

1.4 Research Scope

This research would mainly focuses on how information technology can be applied to restricted small to medium sized local manufacturing industries. Major activities in a manufacturing industry, whether the size of it is small, medium or big, have to be performed on the shop floor. That means, the main concern in a manufacturing industry is related to the activities happening inside the plant. Therefore, there are different types of flows of data and activities happening around the manufacturing industry in

order to accomplish the task assigned to it. These flows can be categorized as process flows, resource flows, data flows and others. These flows can be known as manufacturing phenomena flows because these flows are the main events and incidents that make the manufacturing operations run.

Different sizes of manufacturing industries and business orientation can determine the complication of the manufacturing industry. For example, a manufacturing industry that manufactures cars would involve a lot of processes, data, resources, agents and other elements that are related to one another in order to produce the end product, the car. One process might distribute many different related processes, data, resources and other elements in order to accomplish the tasks of the process. Building a manufacturing model in one single model to represent all the different types of flows, objects, activities and others would be complicated and confusing. Therefore, there are multiple views like information model, activity model, process model and other models to represent different perspectives of a manufacturing industry.

This research is concerned on capturing processes involved in the local small to medium sized manufacturing industries. Therefore, special attention will be on processes in a manufacturing industry. From there, relationships are built between the logical view and the physical view so that customization on the logical part will impact on the real life through implementing some computing system.

1.5 Research Tasks and Dissertation Outline

In order to accomplish the research objectives stated at section 1.3, five specific tasks were undertaken. These five specific tasks were; literature review, development of

modeling structures, the technique used to create the modeling tool, and testing the modeling structures and modeling tool by creating an application through doing a case study at a local manufacturing industry and the development of a subsequent research plan. The following paragraphs will briefly outline the dissertation.

Chapter 1 presents the motivations and needs for the modeling tool to be developed. It also presents the scope of the modeling to process modeling in small to medium sized manufacturing industries. The ability to produce an application from the templates generated depends on the ability to model the processes. This chapter also discusses the objectives of the research and the tasks which were undertaken to realize these objectives.

Chapter 2 reviews some of the current literature relevant to this research. It discusses process modeling and reviews different modeling methods used in enterprise modeling. These modeling methods are IDEF0 Function Modeling, IDEF3 Process Description Capture Method, Object-Oriented Modeling, CIMOSA, and GERAM. Comparison of the modeling methods are made to find the difference among them.

In Chapter 3, the modeling scheme that is used in developing the modeling tool is presented. The focus of the modeling scheme is on the processes in a manufacturing industry. Thus, three different process structures are devised for capturing information from the processes in a manufacturing industry. These process structures are Domain Process (DP) structure, Process (P) structure and Activity (A) structure. Besides that, a Relation structure is also being used to establish the connectivity and relationship between processes in order to reflect the flow of the processes in a manufacturing industry.

Chapter 4 presents designs and techniques for creating modeling tools that can implement the modeling scheme that was discussed in Chapter 3.

Chapter 5 presents the results of the implementation of the modeling scheme, and briefly discusses the outlook of the application developed from using the modeling scheme.

Chapter 6 presents a trial of using the application developed from the modeling scheme. FFM Flour Mills (Sarawak) Sdn Bhd is selected as a resource of trial. The process flows from cleaning the raw materials to milling the raw materials into flour are discussed and the results are presented.

Chapter 7 presents the conclusion of the research. A brief summary of the work and significant accomplishments is provided. This chapter also provides recommendations for future research.

2.1 Introduction

This chapter provides a review on the literatures relevant to this research. The focus of the review is on enterprise modeling. An enterprise can be defined as a set of interdependent actors, with at least partially overlapping goals, working together for a period of time in order to achieve some of their goals. The actors utilize tools, their knowledge and other resources in order to transform some kind of raw input (either goods or information) into processed output that fulfils the needs of a customer. Their efforts are influenced and constrained by their interrelationship with their environment (Lars et al, 1995). Thus the manufacturing industry is a part or the whole of the enterprise.

A model is an abstract representation of reality. Only those aspects of the real system that are of interest to a modeler would be modeled. Thus, models are created for different purposes. An enterprise, which is a complex system, may be represented in different manners in order to describe a specific aspect of the enterprise. For examples, process models, data models, plant layout models and product models, are different models developed to represent different aspects of the enterprise. Therefore, an enterprise model is a symbolic representation of the enterprise and the activities that carry out there. It contains representations of individual facts, objects and relationships that occur within the enterprise (Presley, 1997).

2.2 Process Modeling

Manufacturing modeling is the process of building models of whole or parts of a manufacturing industry such as process modeling, resource modeling, data modeling, and so on. In this research, the focus is on process modeling of a manufacturing industry.

Process Modeling is receiving a prominent position in the business field instead of the proprietary of software engineering. This can be seen from the research work in business process reengineering (BPR) and workflow management. In business process reengineering (BPR), process model is used to evaluate current processes in order to revise or create a new process to meet new organizational requirements or goals. Workflow management is related to the work of reengineering and automating workflows. Workflow is the executable image of a process, which is generated in a design phase (Bussler & Jablonski, 1994).

According to Davenport (1993), a process is defined as a specific ordering of work activities across time and place, with a beginning, an end and clearly identified inputs and outputs: a structure for action. From this definition, a process model is a mapping of the elements in a system as well as the relationship between elements in order to display the specific purpose of the process. The process elements may consist of activities, actors, objects, data, products and tools.

Generally, from process point of view, enterprise can be viewed as being composed of different sets of business processes involving several organizational units. A business process has its customers, composed of activities, which are process steps operated by agents (humans or machines) in order to create value for customers. Thus process model can be used to capture and represent business processes structurally and logically in

different perspectives of the enterprise. According to Curtis et al. (1992), process modeling can facilitate understanding and communication by formalizing the process, support process improvement and management by analysis of process behavior and performance and can automate process guidance and execution support.

In an enterprise, people working under different sectors would have different needs for enterprise models. If a single model were used to represent and contain all the data related to the enterprise, then the model would become too complex and might not deliver the required results. Therefore, there are different definitions of multiple views or perspectives of an enterprise to represent the details that of interest to the modeler. For example, The Automation & Robotics Research Institute (ARRI) describes a five-view approach: business rule (or information) view defines the entities managed by the enterprise and the rules governing their relationships and interactions; activity view defines the functions performed by the enterprise (what is done); business process view defines a time-sequenced set of process (how it is done); resource view defines the resources and capabilities managed by the enterprise and organization view describes how the enterprise is organized which includes the set of constraints and rules governing how it manages itself and its processes (Whitman et al., 1998).

Curtis et al. (1992) proposes four common perspectives in process modeling. These are: functional perspective, behavioral perspective, informational perspective and organizational perspective. The functional perspective presents what process elements are performed as well as what are the informational flows relevant to these process elements that are represented. The process elements may be data, artifacts, objects and products. The behavioral perspective represents when process elements are performed and how they

are performed. The informational perspective represents the structure and the relationship of the informational entities produced by a process. The organizational perspective represents by whom and where in the organization process elements are performed. Even though there might be overlapping between views, there are significant portions of each view that are not described in the other views. Thus, multiple views or perspectives of the enterprise are needed for promoting understanding by reducing complexity. The reduction of complexity is implemented by the reduction of details in one view but provide details that are important to the question answered by the view.

In an enterprise, if there is a need for any improvement efforts to be carried out, a common understanding of the enterprise is one of the critical steps in order to determine the future conditions and direction of the enterprise. Thereby, attention is directed to modeling because it is an approach that can facilitate a common understanding of the enterprise. Through modeling, a model that is relevant to the concerns of the user can be produced. Then the model can be used as a focus of discussion and this also provides a means of communication in the enterprise. Models can also aid in improving activities by serving as a foundation for analysis and design of new entity of the enterprise. According to Frasier (1994), enterprise modeling enables the common understanding of all the pertinent aspects, the clear description of business problems and requirements, the identification of various design alternatives and a mechanism for the analysis of these options for design implementation at strategic, tactical and operational levels.

Enterprise modeling has been adopted by many enterprises as a way to make decisions, analysis and to design the structure of the enterprise and estimate the impact of changes within an enterprise caused by external factors like change of government policies,

market trends and other events. This is true since a model is able to provide a version of a real world system, which describes only essential system properties to the desired level of detail by removing all the irrelevant details. In this section, discussion regarding the contributions of the process modeling to the enterprise will be divided into three different phases – analysis phase, design phase and implementation phase in order to have a better view of the usefulness of process modeling.

Analysis always gives an impression of investigation in order to have better understanding of current situations and gather the necessary information. The following are a number of sources commenting on process model in analysis. According to Maull et al (1995), the benefits gained from developing a business process model do not necessarily result from having a completed or totally accurate model. The benefits are more likely to result from the team communicating their understanding about the process model. This is also stressed by Ould (1995) that process models have to play a role as a communication facilitator. Process model needs to be able to facilitate communication among users of the model such as identifying what is done and by whom, how things are carried out and what things mean, so that the usefulness of process model can be appreciated. Therefore, process model can act as a mechanism for understanding the processes involved and guidance for further development. Davenport (1993) states that process analysis can help to ensure that problems of an existing process are not repeated in the new process.

In the design phase, a process model on a conceptual level is created. This model can assist in decision-making regarding the viability of the designed process as well as its value to the enterprise. The decision is made based on certain aspects of organization which are easily displayed on a process model like activity, roles, data flows, control and goals.

However, organizational aspects such as management styles, skills, staff motivation, organizational power, norms and rewards are difficult to represent in process model.

The implementation phase carries out the decisions that are made at the design phase. In order to execute the decisions (For example, introducing a new process), normally, an IT system is introduced into the implementation phase, which enables the business to evaluate and learn about the processes. This can be seen from the example of workflow management systems, which incorporate IT systems in the workflow execution. In workflow management the system is usually equipped with graphical user interfaces and high-level process definition languages to support the control, communication and coordination of the process. Senge (1993) points out that a process model can play the role of so-called “transitional objects.” These are objects, which are used to facilitate learning about a complex situation so that people are able to make a transition to a new viewpoint or state. Process models make it possible to compose a microworld, a microcosm of reality where “it is safe to play”.

As a conclusion, a process model is useful to communicate a common understanding of an entity of an enterprise, to reduce complexity and to analyze and design processes of an enterprise prior to implementation.

2.3 Plant Interflow Model

In this section, the Plant Interflow Model, one of the models that are built by MIICI is introduced. This section briefly describes MIICI mathematical notations used in constructing the plant interflow model.

According to MIICI report (Goossenaerts & Bjorner, 1994), when constructing a plant interflow model, static structural properties of a plant system, dynamic behavior over instances of the templates in a plant structure and interface between the plant system and plant operations need to be taken into consideration.

A plant structure is used to represent the static structural properties of a plant. This plant structure is the result of joining a part-work structure and a cell-order structure. A part-work structure contains part templates, which describe units of material, and work templates, which describes units of change to materials. Therefore, a Part-Work structure integrates information regarding materials and the work required for making products. As for the cell-order structure, it contains cell templates and order templates. Cell Templates describes cell(s), where a cell is a unit capable of sustaining the delivery or absorption of parts and/or the execution of work (transforming information or material in response to orders). Order templates describe order, where order is a unit of interaction. This interaction is like an order that is sent by one cell (the sender) to another cell (the receiver). This can be summarized as a plant structure that provides information regarding a set of cells that are needed to sustain all work described in a part-work structure. In mathematical definition, plant structure is defined as (C, O, P, G) where

- i. (C,O) is a cell-order structure
- ii. (P,G) is a part work structure

Dynamic behavior of a plant system is defined over a plant structure by instantiating, transforming and deleting instances of templates. This is quite similar to a process of defining computational algorithms where their variables of data types are instantiated and functions and procedures are used to transform the values of the

variables. Thereby, conditions that need to be satisfied by the dynamic behavior of the plant system are expressed in a plant structure. Then, the execution of the cell instances in response to the order instances demonstrates the dynamic behavior of the plant system. The dynamic behavior of the plant system makes distinctions between the specification of a plant system and an implementation of a plant system. The specification of a plant system is the response to order instances the plant system produces and absorbs part instances. The implementation of a plant system is cell instances implement a dynamic behavior in response to the order instances which a plant system receives. The mathematical definition for a plant system specification (PSS) is

$(PS, \{c^{Sales}, c^{Purchase}\}, O^{Sales}, O^{Purchase}, P^{Sales}, P^{Purchase}, PS, O_h_p)$ and the mathematical definition for a plant system implementation (PSI) for a PSS is

$$(PS.rep, O_h_p.body)$$

Finally, the interface between the plant system and plant operations is concerned about the interflow of the plant system with the shop floor activities. Thus, in defining the interface between a plant system and a shop floor; work, part and order flows need to be ensured are ordered in a time-space-material consistent manner, so that the work required can be performed by the cells. This will ensure the plant system transforms according to proceeding shop floor operations.

2.4 Architecture, Framework and Methodology

An architecture is a formal representation or description of a system. An enterprise architecture is a "blueprint" or "picture" which assists in the design of an enterprise (Liles & Presley, 1996). The most striking advantage is that architectures serve as a common tool

for all the employees across the entire enterprise. This in turn helps in enabling the top management of the system to plan where it wants to be as well as develop strategies to get there. According to Whitman et al. (2001), architectures are the building block of any successful modern business strategy. Without it, management is running without direction and has to depend on their own personal perception to make decisions, which might not be shared throughout the enterprise.

Whitman et al. (2001) mentioned that when architectures are developed to cater for a specific industry or sector they are called frameworks. The distinguishing difference between architecture and framework is that architecture takes into the consideration of bigger picture of an enterprise by integrating all possible views in order to achieve its goals. On the other hand a framework is meant for a particular purpose, situation, industry or sector. Thus, a framework can be considered as a solution designed to cover all the details of the specific industry or situation where the requirements are based on the goals and strategic resources of the specify industry or situation. For example, IAA (Insurance Application Architecture) although the name implies it is architecture, actually IAA is a generalized business framework for the insurance industry, where its main purpose is to create and maintain a single view of clients and of the entire enterprise.

A model is built by using modeling tools in the way that is prescribed by modeling methodologies. Methodology provides the guidelines and navigation to the users by describing the processes involved in the modeling so that model development can be carried out in a consistent and optimized path. Different methodologies may exist to cover different aspects of an enterprise. For example, IDEF0 (which will discuss in Section 2.4.1.1) describes the functional aspect of an enterprise operation to any level of detail and

CIMOSA (which will discuss in Section 2.4.2) provides a methodology that covers design, implementation, operation and maintenance of an enterprise. Section 2.4.1 to Section 2.4.4 will touch on different modeling languages to discuss on their functionalities and features that are related to process modeling.

2.4.1 IDEF

IDEF's acronym for ICAM DEFinition where ICAM stands for Integrated Computer Aided Manufacturing. IDEF was developed by the US Air Force. It consists of a suite of modeling methods for describing different perspectives or views of an enterprise. For examples, IDEF0 for functional modeling, IDEF1 for information modeling, IDEF1x for semantic modeling, IDEF2 for dynamic modeling, IDEF3 for process description, IDEF4 for object oriented modeling and etc.

Since this research is focused on process modeling, thus IDEF0 and IDEF3 will be described in the next section.

2.4.1.1 IDEF0 Function Modeling

IDEF0 was evolved from Structured Analysis and Design (SADT) in the 1970s. Then US Air Force commissioned the developers of SADT to produce a tool that could be used for functional modeling in supporting the Air Force's Integrated Computer Aided Manufacturing (ICAM). IDEF0 is a modeling tool that can produce functional view of a system. Information and objects, which are related to functions, can be adhered to the model as well.

In an IDEF0 functional model (see Figure 2.1), the boxes represent functions like actions, activities, processes or operations, which are denoted by verb phrases inside the boxes. Arrows in different positions indicate the different type of data being conveyed. The data can be either information like current status or materials. A noun is used to denote data.

Input is represented by an arrow entering from the left into the box and Output is represented by an arrow leaving from the right of the box. An arrow entering from the top represents Control and an arrow entering from the bottom represents Mechanism. The basic structure of IDEF0 can be seen at Figure 2.1.

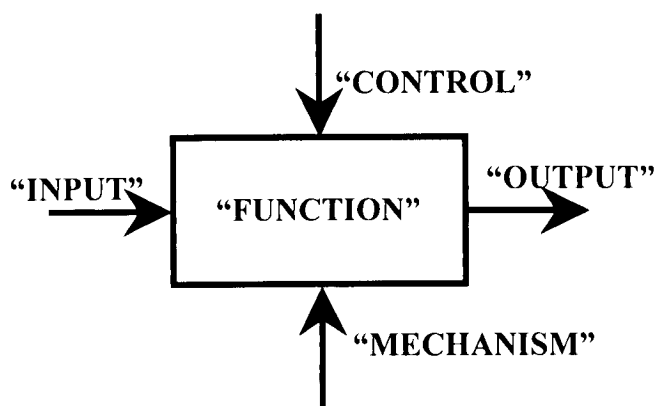


Figure 2.1 IDEF0 Basic Structure Representations

Input represents data needed to perform the function. Output represents the result from the function. Control is the constraints or conditions that imposed to the 'box' in order to govern the implementation of the function. Mechanism represents the people, resources or facilities that are required by the function to trigger the operations inside it. This means that there is dependency between the input and output in identifying the transformations required by the function and thus describes what is done by the function; control describes

why the function is implemented and mechanism describes how it is implemented. IDEF0 modeling focuses on what is done to the system, thus location of the boxes do not necessarily imply sequence or time.

Decomposing or deconstructing a function (see Figure 2.2) into more detailed levels of analysis is another characteristic of IDEF0 modeling technique. The more general diagram is known as the parent of the detailed diagram. Function can be decomposed into sub-functions progressively to express further details. The granularity of the function depends on the intention and interest of the modeler. This provides an environment where communication is enhanced by providing the reader with a well-bounded topic with a manageable amount of new information to learn from each diagram (IDEF0, 1993). Although IDEF0 basic structure is simple it is essential to represent and describe a process.

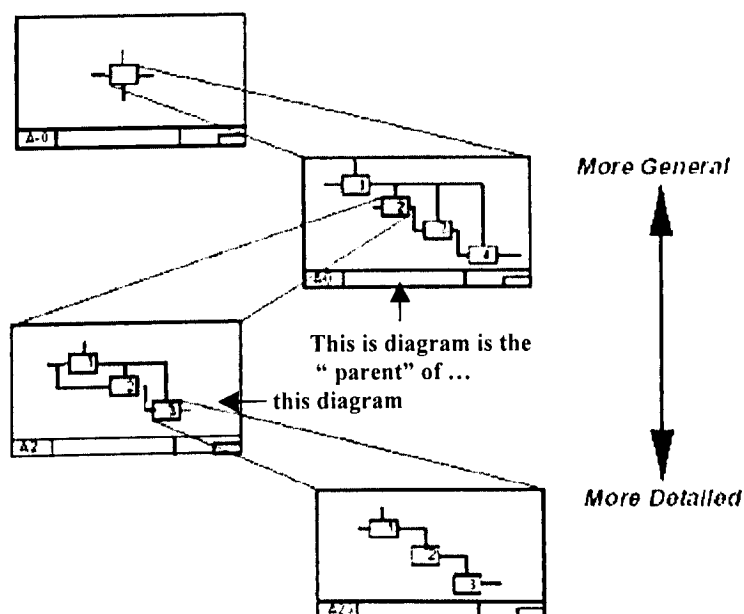


Figure 2.2 Decomposition Overview

2.4.1.2 IDEF3 Process Description Capture Method

The IDEF3 Process Description Capture Method was created specifically to capture descriptions of a sequence of activities. The primary goal of IDEF3 is to provide a structured method by which a domain expert can express knowledge about the operation of a particular system or organization. Knowledge acquisition is enabled by direct capture of assertions about the real-world processes and events in a form that is most natural for capture. This includes the capture of assertions about the objects that participate in the process, assertions about supporting objects, and the precedence and causality relations between processes and events in the environment (Mayer et al., 1995). Thus, IDEF3 differs from IDEF0 where it focuses on “how” things work in an organization by capturing descriptions of sequences of activities.

IDEF3 descriptions are developed from two different perspectives. There is Object-Centered Description, which provides the summary of the allowable transitions of an object and Process-Centered Description, which uses a scenario to capture knowledge about a process and the network of relations that exists between processes. The description is to indicate how things work in a particular situation in an enterprise. This literature will consider the Process-Centered Description.

The five basic structures of IDEF3 Process-Centered Description that are used to express facts are (i) units of behavior (UOBs), (ii) elaborations, (iii) junctions, (iv) links and (v) referents.

UOBs are the basic syntactic unit used to represent a function, an activity, a process or a scenario depending on the context of the description. Each UOB can be deconstructed

to consist of other UOBs with an elaboration associated to it. Elaboration is used to provide detail about the participate process which cannot be shown in the process flow description.

A process-centered description consists a set of UOBs, which are inter-related by junctions and links. Junctions are used to represent the logic of process branching. They express the synchronous and asynchronous behavior among UOBs and the convergence and divergence of process flow. Links are the mechanisms that connect UOBs to form the representation of dynamic processes. They are precedence link, relational link and object flow link.

Referents are used to minimize the clutter of a diagram by referring to a previously defined UOB without duplication of its definition to show that there is ex-UOB defined in a specific point in the process without looping back. The development of IDEF3 is related to the need to distinguish between the description of what a system is supposed to do and the ‘model’, which is used to predict the system, but captures precedence and causality relations between situation and events in a form that is natural to domain experts.

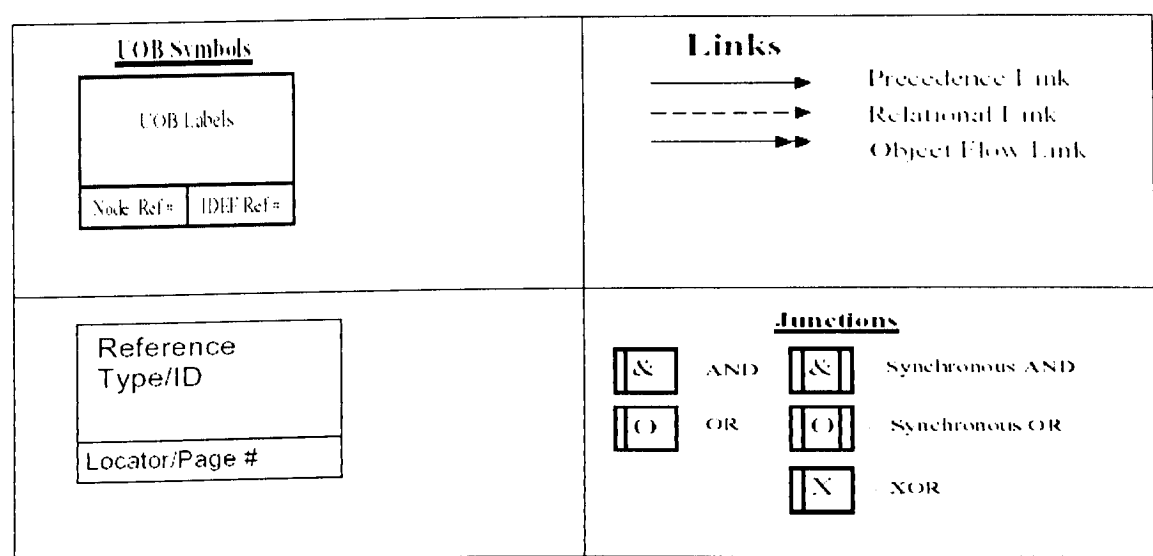


Figure 2.3 Schematic Symbols

2.4.2 CIMOSA

CIMOSA was developed for ESPIRIT (European Strategic Program for Research and Development in Information Technology) by AMICE (a consortium of 30 major European vendors and users of CIM systems).

CIMOSA aims to support system designers with descriptive modeling of enterprise operations which allows the identification of the needs for operational information as well as the information produced during the operation itself.

CIMOSA modeling framework, which is shown in Figure 2.4 has three architecture levels, which describe the dimension of genus with each supporting different views on a particular enterprise model. The Generic Level contains the catalogue of basic building blocks (components, constraints, rules and terms) where these building blocks are applicable to a wide range of CIM implementations. The Partial Level contains a library of partial models applicable to a specific category of manufacturing enterprises. A Particular Level utilizes the ready prepared building blocks from the Generic and Partial Level and develops the specific requirements and components for the enterprise.

CIMOSA implements a modular approach for enterprise modeling. This can be seen as CIMOSA supports a complete system life cycle from requirement definition, design specification and implementation description and defines four modeling views. The four modeling views are: function view, which describes operational and behavioral aspects of an enterprise; information view, which describe the inputs and outputs of functions; resource view, which describes the structure of resources like machines, humans and control; and organization view which defines in term of responsibilities and authorization for processes, functionalities, information, resources and organization.

In the following section, discussion will focus on the function view of CIMOSA. In CIMOSA, a set of business modeling constructs and elements are used to build model (Table 2.1). The modeling process is represented as a set of Domain Processes (DP) with Events and Results as a communication channel. DP can be decomposed into Business Processes (BP) and Enterprise Activities (EA), where a set of Behavioral Rules (BRS) is used to connect them.

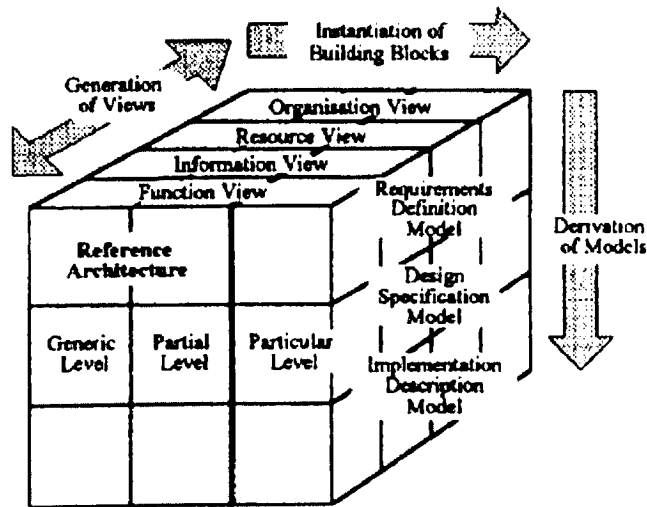


Figure 2.4 CIMOSA modeling framework

In this paragraph, some of terms used in CIMOSA are described in order to have a better understanding of the terms used. Therefore, information from the sources, Kosanke (1995), Kosanke and Zelm (1999) and Zelm et al. (1995) are used. In CIMOSA, Domain Process (DP) is triggered by events and produces a result; it encapsulates a set of enterprise functionalities and behavior, which are used to achieve defined objectives under given constraints. It is at the top of a functional decomposition hierarchy thus it belongs to one

Domain, for example, DP cannot be used either by any other Domain or a Business Process. As for Business Process (BP), it defines a behavior part and a structural part at all levels of functional decomposition except top and bottom levels. It can employ one or more Business Processes and/or Enterprise Activities and at the same time it can be employed by one or more Domain Processes and/or Business Processes. Business Process is triggered by a parent structure like Domain Process. Enterprise Activity (EA) is the construct that defines a functional part and a structural part to describe the functionality in the Function View of a particular enterprise at the Requirement Definition Modeling Level. It can be employed by other Domain Processes and/or Business Processes but it does not employ any Business Processes or Enterprise Activities. Behavioral Rule describes the action(s) to be taken in the flow of control of a Domain Process or a Business Process. Functional Entities are resources that are able to receive, send, process, and (optional) store information. Functional Operation is the basic unit of work that is either executed, or not, at run time and is the lowest level in the Function View.

Function	Behavior	Information	Resource	Organization
Modeling Construct				
Domain	Domain Process	Enterprise Object	Capability Set	Organization cell
Enterprise Activity	Business Process	Object View	Resource	Organization Unit
	Event		(Functional Entity	
Modeling Construct Element				
Domain Relationship	Behavior Rule	Information Element	Capability	Organization Element
Objective Constraints	Activity Behavior	Object Relationship	Resource Component	
Functional Operation				

Table 2.1 CIMOSA modeling constructs and their elements

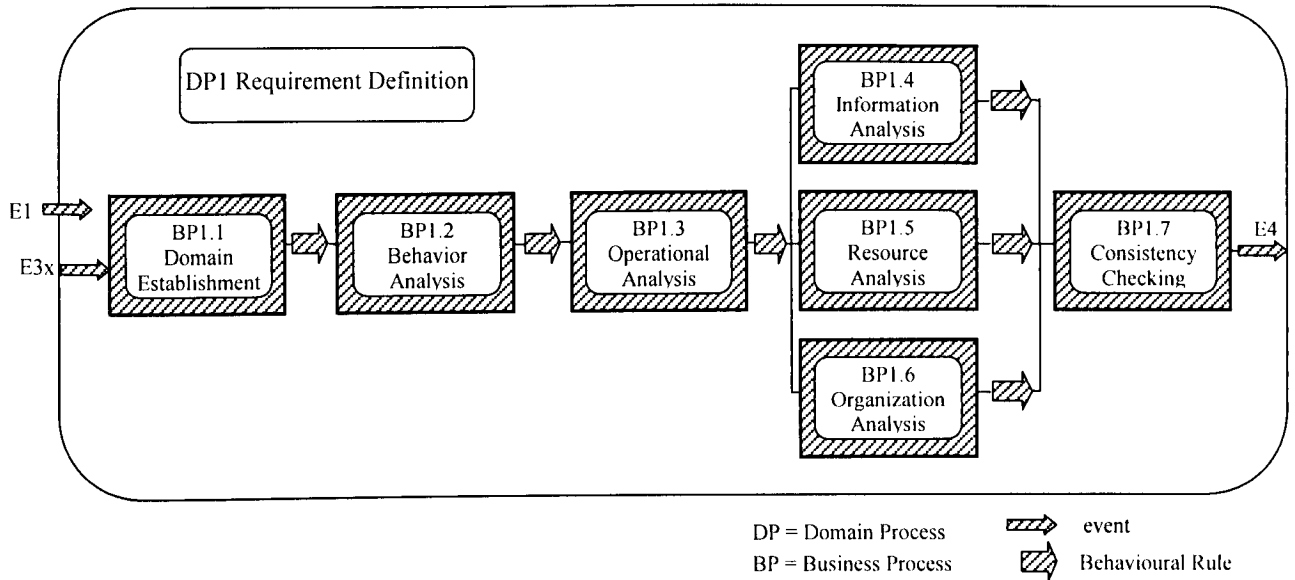


Figure 2.5 First level details of Domain Process – Requirement Definition

In the Requirement Definition Modeling, functional contents and its process behavior are described in two Business Processes. There are Behavioral Analysis and Operational Analysis. In Behavioral Analysis, each domain process identified in Domain Establishment (initial step of Requirement Definition Modeling) is further structured into Business Processes and Enterprise Activities either through a top-down or bottom-up approach. Operational Analysis details the Enterprise Activities that are produced in Behavioral Analysis by identifying the operational information, resources and capabilities used and needed as well as the new information that is produced in the enterprise operation.

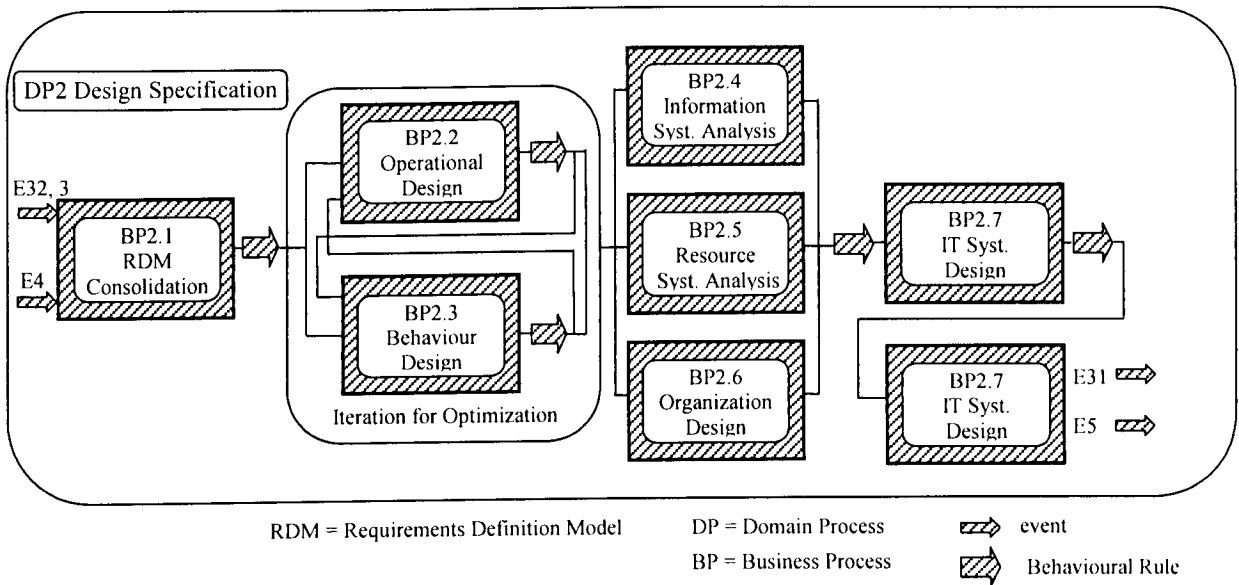


Figure 2.6 First level details of Domain Process – Design Specification

The purpose of the Design Specification Modeling phase is to specify HOW the system requirements should be implemented, taking into account the relevant enterprise policies, constraints and system dynamics. In this phase, Behavior Design and Operational Design are the two Business Processes concerned with the functional aspects of an enterprise. In Behavior Design, identified Enterprise Activities in the Requirement Definition Modeling phase would define activity behavior, which controls the execution of the identified Functional Operations. In Operational Design, Enterprise Activities are decomposed into Functional Operations. Each Functional Operation is executed by one Functional Entity but a Functional Entity may be capable of executing more than one type of Functional Operation.

2.4.3 GERAM

Generalized Enterprise Reference Architecture and Methodology (GERAM) was developed by the IFIP-IFAC task force on Enterprise Integration. GERAM was developed from the evaluation of existing enterprise integration architectures (CIMOSA, GRAI/GIM and PERA). GERAM (version 1.6.3) is about those methods, models and tools, which are needed to build and maintain the integrated enterprise, be it a part of an enterprise, a single enterprise or a network of enterprises (virtual enterprise or extended enterprise).

GERAM is an enterprise reference architecture which has not been proposed for use like CIMOSA, PERA and GRAI/GIM, but its intention is to gather existing enterprise integration knowledge in an organized way to come up with a framework that has potential for application to all types of enterprise. This can be seen from GERAM where it sets the standard for the collection of tools and methods for any enterprise to more easily tackle initial integration design and change processes which may occur. This is because the adopted tools and methods would need to satisfy the defined criteria of the standard. Figure 2.7 shows the set of components identified in GERAM.

GERA (Generic Enterprise Reference Architecture) identifies the enterprise related generic concepts to be used in enterprise engineering and integration. These concepts are categorized into: human oriented concepts, dealing with all aspects concerning humans in an enterprise; process oriented concepts, describing the functionalities and activities of an enterprise; and technology oriented concepts, providing descriptions of the technology involved in the enterprise operation and the enterprise engineering efforts. EEMs (Enterprise Engineering Methodologies) is employed by GERA to describe the processes of enterprise engineering and integration which can be in the form of process models or

structured procedures with detailed instructions for each enterprise engineering and integration activity. As for process models, EMLs (Enterprise Modeling Languages) may be defined as modeling constructs, used for the expression of the respective models by representing different elements of the modeled enterprise. The methodology and the languages are supported by enterprise engineering tools (EETs) in the process of enterprise modeling to produce enterprise models (EMs), which can be used to guide the implementation of the particular enterprise operational system (EOSs). EOSs might use specific modules (EMOs) that provide prefabricated products as components in their implementation of the enterprise.

Partial enterprise models (PEMs), which provide reusable models of human roles, processes and technologies, enhance the process of modeling by removing the need to develop from scratch. Generic Enterprise Modeling Concepts (GEMCs) are the most generically used concepts and definition of enterprise modeling, which can be defined in three form of increasing order of formality. There are glossaries – terminologies used in enterprise modeling defined in natural language; meta-models – conceptual models describing the relationships among modeling concepts of enterprise modeling languages; and ontological theories - formal models define the meaning of modeling languages used and enhance analysis capabilities of engineering tools.

In GERAM, GERA provides the view concept that allows different model views concerning information, function, resource and organization of an enterprise of an integrated model to be presented to the user. This is to reduce the apparent complexity of the resulting enterprise model and highlight the aspects of the model that are to be concentrated when using the model. In order to ease the description of process

representation by the user, four different model content views have been defined: Function, Information, Organization and Resource. These views will represent their own respective aspects of an enterprise.

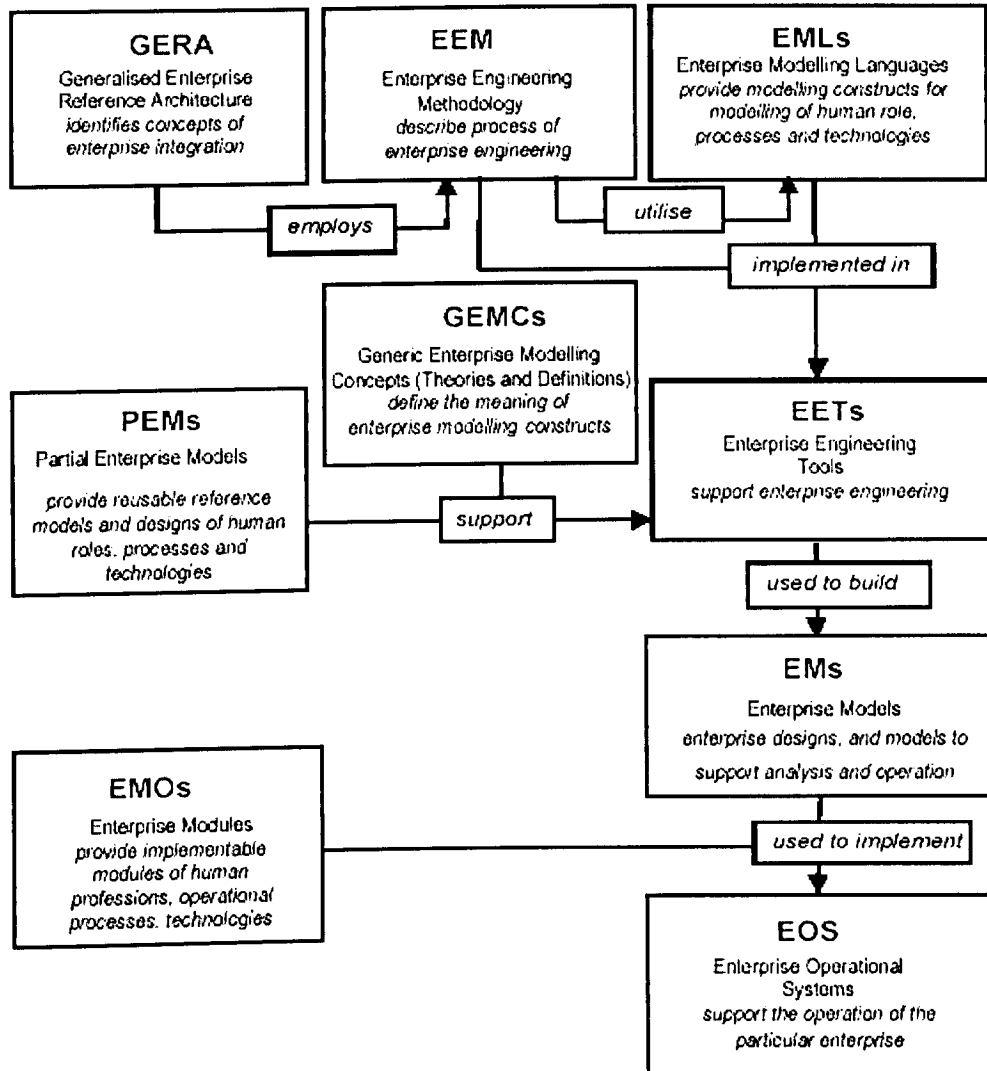


Figure 2.7 GERAM framework components

In function view, functionalities (activities) and the behavior (flow of control) of the business processes of an enterprise are represented. There are different types of activities like decisional, transformational and support activities, which differ in their expressive power; but they relate to some aspect of the enterprise function. During enterprise operation, information is used and produced from pertinent activities. This information is collected and structured to be presented in the information view. This can facilitate the management and control of information flow. The resource view concerns the resources like humans and technologies that are assigned to activities during enterprise operations. The organization view describes responsibilities and authorities of an enterprise that are identified in function, information and resource views and are presented in a structured and organized way.

2.4.4 Object-Oriented Modeling

Object-oriented approach is all about object. An object can be seen as a "black box" and it can communicate with other objects by sending and receiving messages. The "black box" contains properties and behavior. The properties describe the current state of the object and the way it acts and reacts is described by behavior. Moreover, there are several important concepts associated with object-oriented approach which makes it different from traditional techniques: encapsulation, message based communication, inheritance, classification and polymorphism.

In this paragraph, object-oriented concepts that differentiate object-oriented approaches from traditional techniques are briefly described (Bahrami, 1999). Encapsulation, which is also known as information hiding, means that an object's data and

methods are bundle together and the data can be accessed by methods associated with the object. Accessing data of the object by other objects can be achieved through the methods associated with the object. In order to access the data of an object, communication between objects needs to be established by sending messages to that particular object to request the services. Inheritance is the attributes and methods of a class shared with its sub classes. Sub classes can override data and methods of the superclass to introduce specialization of classes. Inheritance and specialization promote the concept of reusability. Classification in an object-oriented approach allows it to capture all the similar objects in one place. For example, all the objects with the same attributes and behavior are instances of one class. As for polymorphism, it means that a message sent to different objects can have different interpretations and responses

There are a number of object-oriented modeling methodologies used in the business and academic world. Different types of notations or diagram elements are employed by these object-oriented modeling methodologies in their analysis and design. For example, the Booch method, Jim Rumbaugh with his object modeling technique (OMT) and Ivar Jacobson introduced the concept of use case and object-oriented software engineering (OOSE). These provide system designers with many choices but create an ambiguity and split environment because there are many similarities in these methods with a number of annoying minor differences.

Unified Modeling Language (UML) is the result of collaboration of three main modeling language methods in object-oriented industry. They are James Rumbaugh, Grady Booch and Ivar Jacobson. UML was adopted by the Object Management Group (OMG) as

standard in 1999. In the following section, UML will be discussed as the object-oriented modeling in this literature.

UML is a graphical language that can model object-oriented software systems as well as for process modeling in the field of business (Ovidiu, 2000). UML provides different types of diagrams that are able to map the processes in the business fields. This mapping is trying to represent the real world phenomena by a set of formal notations, including the necessary information. The result of the mapping is models, which present the logical views of the business processes. UML then can take the models all the way through the development of software objects. The process from mapping till the development of software objects involves object-oriented analysis, object-oriented design and Implementation.

In this section, seven UML's diagrams that are considered to be related to business modeling are discussed. They are class diagram, object diagram, use case diagram, activity diagram, statechart diagram, sequence diagram and collaboration diagram. The discussion of the diagrams revolves about the business process modeling therefore not completely discussing UML diagrams. The information provided below is extracted from the sources of Bahrami (1999), Ovidiu (2000), McUmbur (2002), Jacobson et al. (1995) and Kueng et al. (1996).

Class diagram can be used to represent static structure of the business process in the real world. Different classes may be used to represent and describe different elements' structures like information, products or organization by catching the attributes and operations required. Associations and generalizations are used by class diagram to represent relationships among the classes. An object diagram is an instance of a class diagram, which shows details of the state of the class at specific situation. A Use case

diagram describes the functionality aspect of the business process by representing the function of a business process with a specific flow of events in the system. Besides that, an actor, which is anything that can interact with use case like humans or machines, can be used to define the role it plays in the function of the business process. An Activity diagram describes activities and work inside a use case or among several classes. It can be used by business process modeling by defining a flow or sequence of activities that is happening in a system.

Statechart diagram, interaction diagram and collaboration diagram can be used to represent the dynamic aspects of business process. A Statechart diagram can represent the lifecycle of objects in the business process by showing their states after receiving some external stimuli, like a message, and how different events may affect those states over time. Communication is a common activity that is happening in the business process. Interaction among a set of objects can be represented by using a sequence diagram. A Sequence diagram shows the messages exchanged among the objects by arranging the interaction in a time sequence. A Collaboration diagram describes the interaction of a set of objects in a particular context to achieve a desired outcome. It is similar to the sequence diagrams but is capable of representing more complex interactions among the collaborating objects.

UML has been established and accepted as standard for software modeling language. It can be used to model different aspects of business process by categorizing them into object models, functional models and behavior models.

2.5 Comparison of reviewed Modeling Languages

In this section, different methods that are discussed in previous section are evaluated. The purpose of comparison is not to compare against each other but to evaluate their capabilities and their use in the context of enterprise modeling. This is because these modeling languages are developed with different intentions to meet their own purposes. This can be seen from Table 2.2 where CIMOSA met most of the requirements defined because it is built with the enterprise modeling in mind. Since GERAM still cannot be used as modeling language, it will not be part of the comparison.

Table 2.3 shows the comparison of these modeling languages. The comparison is divided into modeling levels, modeling views and modeling related. In modeling levels, there are three different levels that are taken into consideration: Requirement Definition, Design Specification and Implementation Description and modeling views are categorized into Function View, Information view, Organization view and Resource view. As for modeling related, some of the modeling concerns are listed. There are decomposition, reusability, modeling constructs, human roles, methodology support, information flow and workflow. Table 2.2 provides the descriptions of the “members” of modeling related.

Modeling Related	Descriptions
Decomposition	Concerned with the level of detail, which is from general to detailed views of a system
Reusability	Concerning using a partial model and promoting modularity to reduce modeling time.
Modeling Constructs	Constructs that are used for description and model elements of an enterprise.
Human Roles	Roles and location of human work in an enterprise
Methodology Support	Describes the process of enterprise modeling
Information Flow	Concerned with information used and produced during enterprise operations.
Workflow	Concern with sequence or order of the enterprise activities that must be performed to achieve goal.

Table 2.2: Description of related terms in enterprise modeling

Modeling Methods	Modeling Features	IDEF0	IDEF3	OOM	CIMOSA
Modeling Levels					
- Requirement Definition		Y	Y	Y	Y
- Design Specification		L	L	Y	Y
- Implementation Description		N	N	Y	Y
Modeling Views					
- Function View		Y	Y	Y	Y
- Information View		L	L	Y	Y
- Organization View		N	N	N	Y
- Resource View		L	N	N	Y
Modeling Related					
- Decomposition		Y	Y	L	Y
- Reusability		N	N	Y	Y
- Modeling Constructs		Y	Y	Y	Y
- Human Roles		N	N	N	L
- Methodology Support		Y	Y	Y	Y
- Information Flow		Y	Y	Y	Y
- Workflow		N	Y	L	Y

Legend for Table 2.2:

Y = Supported

L = Support is limited

N = No supported.

Table 2.3: Comparison of different modeling Languages.

2.6 Conclusion

In this chapter, we can see from the above that there have been different types of modeling tools created to perform enterprise modeling. However, this literature only reviews a few of them. There are a lot different modeling tools, architectures and methodologies that exist in different contexts. For examples, PERA, GRAI-GIM, TOVE, IEM, IDEF suite, SADT and others. Some are meant for business process re-engineering, some for enterprise re-engineering and enterprise integration, and other fields. They cover different requirements and aspects of enterprises with different syntax, semantic, graphical notation, procedures and structure. However, they all capture the reality with logical representation to perform their intended purposes. Based on the reviewed modeling tools discussed in this chapter, the next chapter will discuss the modeling scheme that is used to develop the architecture of the demonstrator for the system. Object-oriented analysis will also be discussed in the next chapter.

3.1 Introduction

This chapter elaborates on the structure and the functionalities of the components that compose the modeling tool. The modeling tool that is developed needs to achieve the objectives of this research (as mentioned in Chapter 1). Therefore, the requirement of the modeling tool is that it can be used to capture the relevant processes of the manufacturing industry. Through these processes, coding templates related to the processes can be generated based on the model developed. Then users can customize the coding templates based on their requirements to simulate the actual processes in the manufacturing industry. The model, which is produced by the modeling tool, is concerned with what the processes of the manufacturing industry do, and coding templates can be used to customize how the processes do this.

In order to develop a modeling tool for a manufacturing industry that can meet the objectives of this research, object-oriented analysis will be used to understand and capture a complete and clear view of the requirements of the modeling tool. In object-oriented analysis, a use-case approach will be used where a use case is typically an interaction between a user and a system that captures users' goals and needs (Bahrami, 1999). Then, architecture of the model tool that can develop the generic model of manufacturing industry will be produced. The architecture of the model tool will be discussed in Chapter 4.

In order to come up with an architecture of the model tool, a feasibility study on the domain analysis needs to be done properly. This is because; results from the domain analysis provide a basic framework for the beginning of modeling. It can be seen that

through domain analysis, boundaries and perspectives of the manufacturing industries are defined for the modeling tool to be used to perform the task of modeling. This is necessary since every complex system like manufacturing industries is best approached through providing an independent view of a model with a defined scope to reflect the purpose of modeling.

3.2 Domain Analysis

In software engineering, domain analysis is used for software modeling of complex systems. The implementation of domain analysis is to collect and classify information related to the problems in a domain of investigation so that good decision can be made during requirement analysis and other stages of the software engineering process (Lethbride & Laganier, 2001).

Domain can be defined as a territory over which rule or control is exercised (Dictionary.com, 2005). That means, in a particular field or a region, the interests of the dominion are exercised. Some domains can be very broad, such as airline reservation, others might be narrower, such as scheduling batch process. Therefore, it is important to define the domain so that the elements, structure, behavior and their relationship of the manufacturing industry can be defined. The domain of this research is concerned with the process flow modeling, with the primary focus on functional aspects of the manufacturing industry. Following is an elaboration of the defined domain of this research.

In this research context, process flow is defined as a series of activities where changes on physical objects and information can be observed in the designated environment (Davenport, 1993). According to Lin et al. (2002), relation, behavior and agent are the main

components needed in representing the functional perspective. A relation is the interaction between processes. Behavior denotes how a process executes actions and activities inside it and an agent, also named as social actor or role. It can be interpreted as an agent conducts itself to perform some activities according to its relation with other agents. Therefore, this can be viewed as a process with a specific role to perform some activities that meet its intention by following the relation with other processes. An activity indicates what a process does. As for the designated environment, it is a manufacturing industry where it can cover a wide boundary. Thus, the term plant is used to narrow the scope. According to Goossenaerts and Bjorner (1994), a plant is a phenomena (process) flow capable of sustaining the transformation of inputs parts into output parts. The input parts to the plant can be supply orders from the plant like materials needed by the plant to perform the daily operations or presented by the market to the plant for processing or absorption. The output parts are the results that are ordered by customers or patterns determined by the plant so that they can be delivered to the market. The basic idea is that the plant can be thought of as operating as a set of interrelated processes. As can see, nowadays manufacturing industries already can range from a heavy high technology sectors which involves many and complex processes, to small sized sectors which only involve a few simple processes to produce output to cope with the demands of the market. Thus, this research narrows the scope of the plant so that the modeling tool is applicable to small to medium sized manufacturing industries.

3.3 Modeling Tool Structure

Based on the domain analysis, the main elements of the modeling tool are the processes and the relations of the processes. Through process, activities and behavior aspects of the processes can be described. Relation of the processes denotes input and output of the processes. However, input and output of the processes might involve data. Thereby, a data handling aspect would be taken into consideration.

Since process is the focus, the structure of the modeling tool would need to be able to cover the process flows in a plant. According to Presley et al. (1993), an enterprise is a collection of enterprise activities organized into a set of business processes which cooperate to produce desired enterprise results. This shows that enterprise activities are important for the existence of an enterprise. An enterprise activity is any organized behavior which transforms inputs into outputs (Liles & Presley, 1996). This can be seen from Figure 3.1 which takes a system view of an enterprise. There are several sets of boxes which represent enterprise activities logically organized into shaded ellipses. They are business processes, which are organized into an enterprise represented by the larger box. This architecture can be viewed as the enterprise itself and is represented as a system, which takes inputs and transforms them into output.

Three structures are defined in this modeling tool structure. They are Domain Process structure, Process structure and Activity structure. This is due to the fact that the processes in a manufacturing industry can be divided into different categories. For example, production processes, packaging processes. These processes can each be made up of several sub processes where each of these sub processes performs their own activities. These structures implicitly form a domain or boundary for job of a different nature of

processes in manufacturing industries. Interrelated processes are gathered in one domain to perform their activities. Figure 3.2 shows the scenario described. The details regarding these structures will be further elaborated in sections 3.3.1 to 3.3.3.

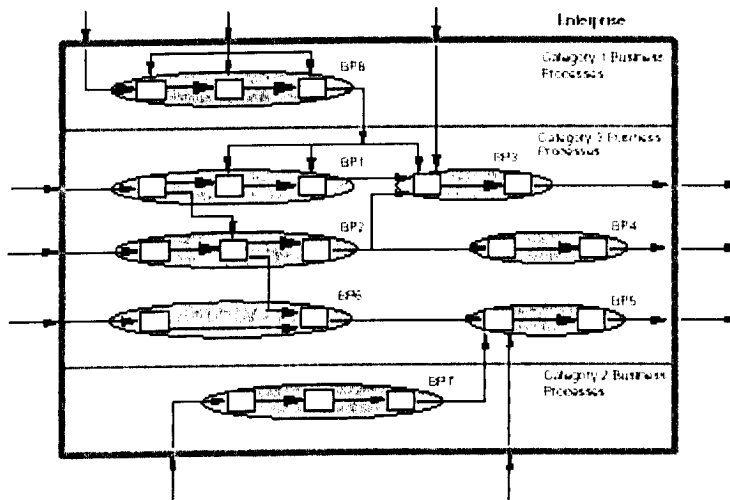


Figure 3.1: Enterprise as Collection of Business Processes

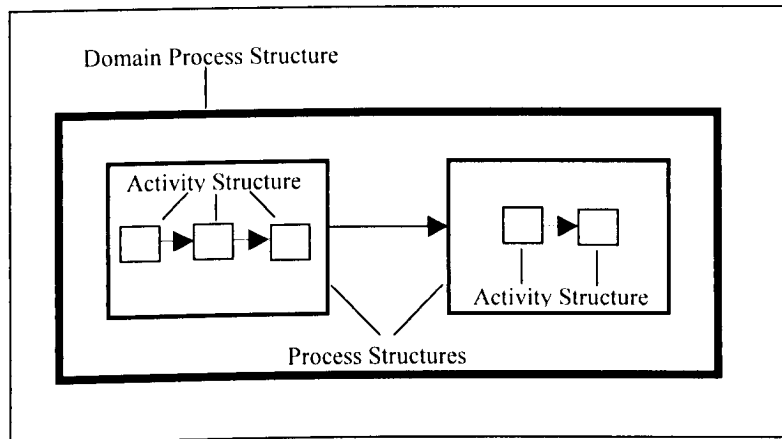


Figure 3.2: Modeling Tool Structures

Applying these structures in the modeling tool will provide a descriptive schema regarding the model developed. Interrelated processes of the plant are gathered into one domain, the sequence of the process flows is ordered and labeling enhances the understanding of the

model developed. Further, coding templates generated based on the model developed using these structures can form the basis for a computer system to support a particular behavior of the plant. Thereby, these structures are the fundamental modeling concepts and the semantics of the research adopted that can facilitate the communication of the complex relationship of information and activities in the manufacturing industries.

3.3.1 Domain Process structure

Domain Process is the structure that exists in the highest level of the structure of the modeling tool's hierarchy of a domain. It is used to depict the major functions that make up the complete flow of processes of a system or subject area in the highest abstraction. The purpose of Domain Process (DP) structure is to provide a general view of a model that is going to be developed. This view is developed based on the intention of the modeler so that the modeler has explicit control in setting the model scope and orientation.

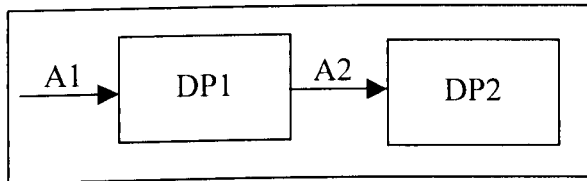


Figure 3.3: Relations between Domain Processes

Domain Process is a representation of a set of possible related processes and functions under it. All the detail regarding the Domain Process (DP) are presented in the next level by decomposing the Domain Process (DP) structure. Domain Process (DP) structure can be decomposed into next level of details by either using Process (P) structure or Activity (A) structure. Domain Process (DP) structure accepts inputs as external

resources that would be used, as data has to be available inside the Domain Process (DP). The Domain Process (DP) can treat these data as its attributes. These attributes can be used as factors that trigger the operations inside it and generate results. These results are the output from the Domain Process (DP) to be delivered to another Domain Process (DP). They can also be the input to the Domain Process (DP) if the Domain Process (DP) needs them. Figure 3.3 illustrates this further, where A1 is the input to the DP1 (DP stands for Domain Process) and it produces the result of A2. Simultaneously, A2 also acts as input to DP2. This forms a kind of interaction or relationship between Domain Processes (DP). A set of Domain Processes (DP) exchange results and requests form the structure of the plant operation.

3.3.2 Process structure

A Process (P) structure is a structure that comes into existence after the Domain Process (DP) has been decomposed into next level of details. Process (P) structures created at child level in return provide additional detail regarding the parent process. Thus, Process (P) structures at the child level can be considered as components that make up their parent process. This can be seen from Figure 3.4.

Process can be used to define structural parts and behavior parts of a Domain Process (DP). Process (P) structure can accept input from external sources in order to meet its own needs as its attributes. Process (P) structure can generate results as in Domain Process (DP) structure, where this result can act as an input to the next Process (P) structure. There is association between parent and child processes where at child level; they are bounded within the scope of its parent. The amount of new information and control

is manageable by its parent so that the child level can perform the intended operation and produce the intended results.

Hence, creating a child level with Process (P) structure can provide a clearer and more organized structural view of the model developed. The level where the Process (P) structure appears is known as middle level.

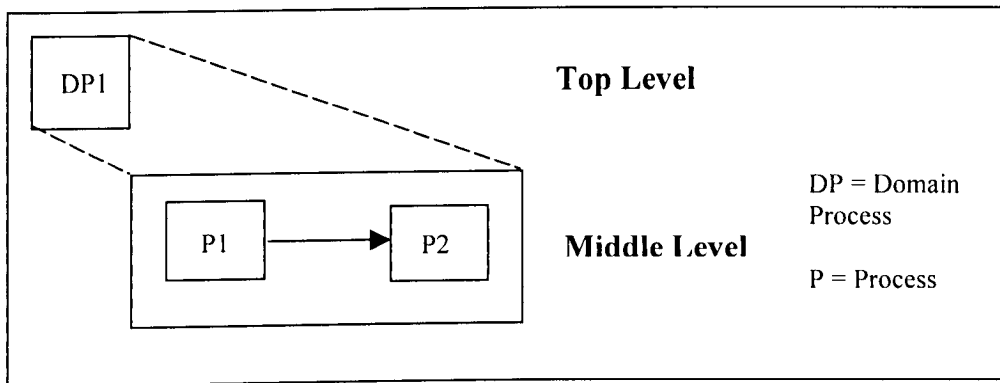


Figure 3.4 Decomposition activities of Domain Process to Process

3.3.3 Activity structure

Activity (A) structure defines functional part that would be employed by Domain Process (DP) structures or Process (P) structures. At this level, decomposition into the next level is no longer allowed. Therefore, it is known as Bottom Level, where functions or methods of its Domain Process (DP) structure or Process (P) structure would be created to implement the activities that need to perform so that the intended results can be produced.

If Domain Process (DP) structure is directly decomposed into Activity (A) structure, then Process (P) structure would no longer come into existence. The same goes for Process (P) structure. This can be seen in Figure 3.5 where there are two diagrams showing this

case. This not only tells us that the bottom level has been reached, but also indicates the completion of the structuring of a particular Domain Process (DP) structure.

Activity (A) structure can accept new input as its variables. The roles of input and output are the same as described in the Domain Process (DP) structure and Process (P) structure.

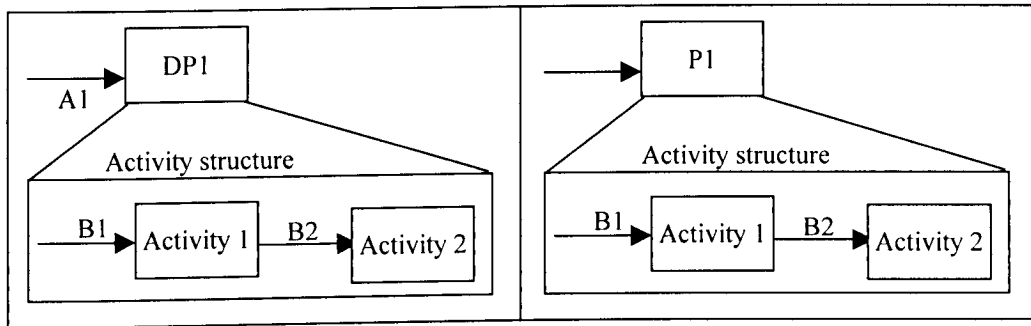


Figure 3.5: Activity structure

3.4 Relation Structure

A Relation Structure can be used as Input Relation and Output Relation. Input Relation is associated on the left side of a process structure. It denotes the data that are needed by the process structure. Output Relation is associated on the right side of a process structure. It denotes the result generated by a process structure. For example, in Figure 3.6, R2 is a Relation structure that carries the output from Process 1 and input to Process 2. Therefore, in a Relation structure, it can have one set of data that is used by two different process structures; this set of data is differentiated as input from one process structure and output at the other process structure. This input and output forms the foundation of relationship and connectivity of a process flow.

Besides this, there are two sources of information for defining Relation structure, which are; predefined kind and post-defined kind. Predefined kind is the one that used to indicate the beginning point of a level. This can be seen from Figure 3.6 where R1 connects to Process 1 indicating the beginning point of the process flow. Since Predefined kind Relation structure does not have any process structure associated with it at the beginning of the Predefined kind Relation structure, then Predefined kind Relation structure can only accept input. Post-defined kind Relation structure participates in building relationships between process structures when there is a relation and connectivity required for the process structures. Therefore, a Relation Structure that connects two process structures like R2 or R3 in Figure 3.6 is known as Post-defined kind Relation Structure.

A Relation structure that is not associated to the left side of a process structure is not allowed to come into existence. This is because there is no destination for the result generated from a process structure.

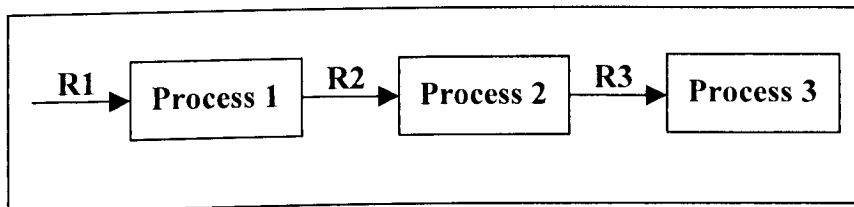


Figure 3.6: Relationship of a Relation Structure

3.5 The modeling tool components and activities

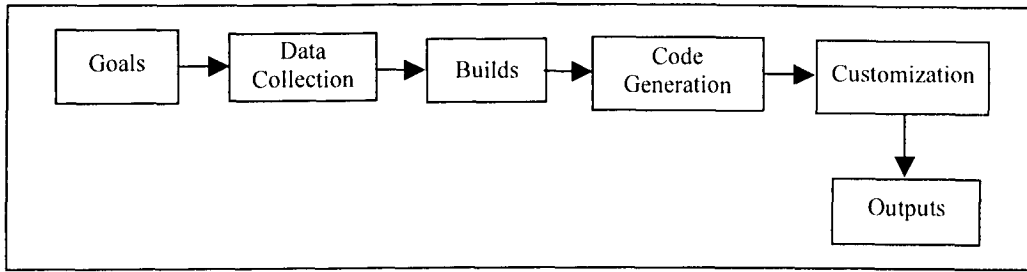


Figure 3.7: The modeling activities in using the modeling tool

Domain Process (DP) structure, Process (P) structure and Activity (A) structure are three major components of the modeling tool that are essential in modeling of the process flow of a manufacturing industry in this research. In IDEF0 (1993), a box in rectangle shape containing name and number is used to represent a function. Based on this, a process box in rectangle shape is used to represent the processes in a manufacturing industry. However, there are three different process structures used in the modeling tool, thus three different denotations are employed to represent each of them at the left bottom corner of the process box. These denotations are DP for Domain Process structure, P for Process structure and A for Activity structure. As for relationship between processes, a directed line with arrowhead is used to denote the interaction. With these, they can graphically show what the processes in the domain are, and describe the structure and relationship among processes by visual notation.

These graphical notations provide the visual rendering of the model's elements; they are sets of symbols that can be best used by following the guidelines. There are six modeling activities that can be implemented as shown in Figure 3.7. Goal definition is first

in the process of model development. This is important because through defining the goal, a boundary of the modeling can be defined and the modeler only needs to model the part that is essential, without capturing everything into a single model. Moreover, it can help the modeler to stay focused on the problem at hand. This would present a view that is essential and a more easily understood model. After defining the goal, the modeler needs to perform data collection. This is to gather and classify all the relevant information and behavior of the processes.

Then the process of modeling is performed based on the data collected by using modeling components provided. At this stage, the modeler can choose to depict the model in a graphical representation or choose to generate the coding templates. Using the generated coding templates, the modeler can perform customization of the processes. Output of this customization is an application that can simulate the actions performed by the real processes in manufacturing plant.

With these modeling components and modeling activities, a model of a system or subject area can be constructed to meet different usages. For a new system, they can be used to define the requirements and specify the operations. As for existing system, they can support analysis and understanding of the system or subject area and provide logic for potential changes of the system.

3.6 Use Case Driven Object Oriented Analysis

Analysis is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of a system's requirements (Bahrami, 1999). The main intent of this activity is to capture complete, unambiguous and consistent requirements of

the system that must do to meet the users' requirements and needs. To meet the users' requirements, it is necessary to understand them through finding out how users use the system.

In object-oriented analysis (OOA), a unified approach is used to analyze the system where actors (outside) and use cases (inside) are two initial important elements used to define the system's behavior. They can be used to fill out the work started by domain analysis on describing what the system does rather than how it does it from the user's perspective. The following sub-sections describe the modeling tool system requirements specification in terms of the objectives of the research defined in Chapter 1 and literature reviews in Chapter 2.

3.6.1 Activity Diagram of the Modeling Tool

The analysis process begins with developing an activity diagram of the modeling process of the modeling tool. Activity diagram can be used to provide a view of flows and describes actions taken in the modeling tool. The main idea of developing an activity diagram at the first place is to get a basic model which can provide better understanding of what sort of activities are performed in the modeling tool. This helps the modeler to get familiar with user requirements and provides aid in developing use cases.

Figure 3.8 shows the major activities of using the modeling tool to develop a model. The modeler has the options to create a new workspace which allows the development of a new model or retrieving an existing model. Creating a new model; modeler needs to define the purpose of developing the model. Then the modeler can start to develop the model at three different levels. These are Top Level, Middle Level and Bottom Level. The modeler can use

decompose and up level methods to navigate around these three levels. Once the model is completely developed, then the modeler can generate the coding templates based on the model developed to customize the processes. The modeler can also save the model developed for future retrieval.

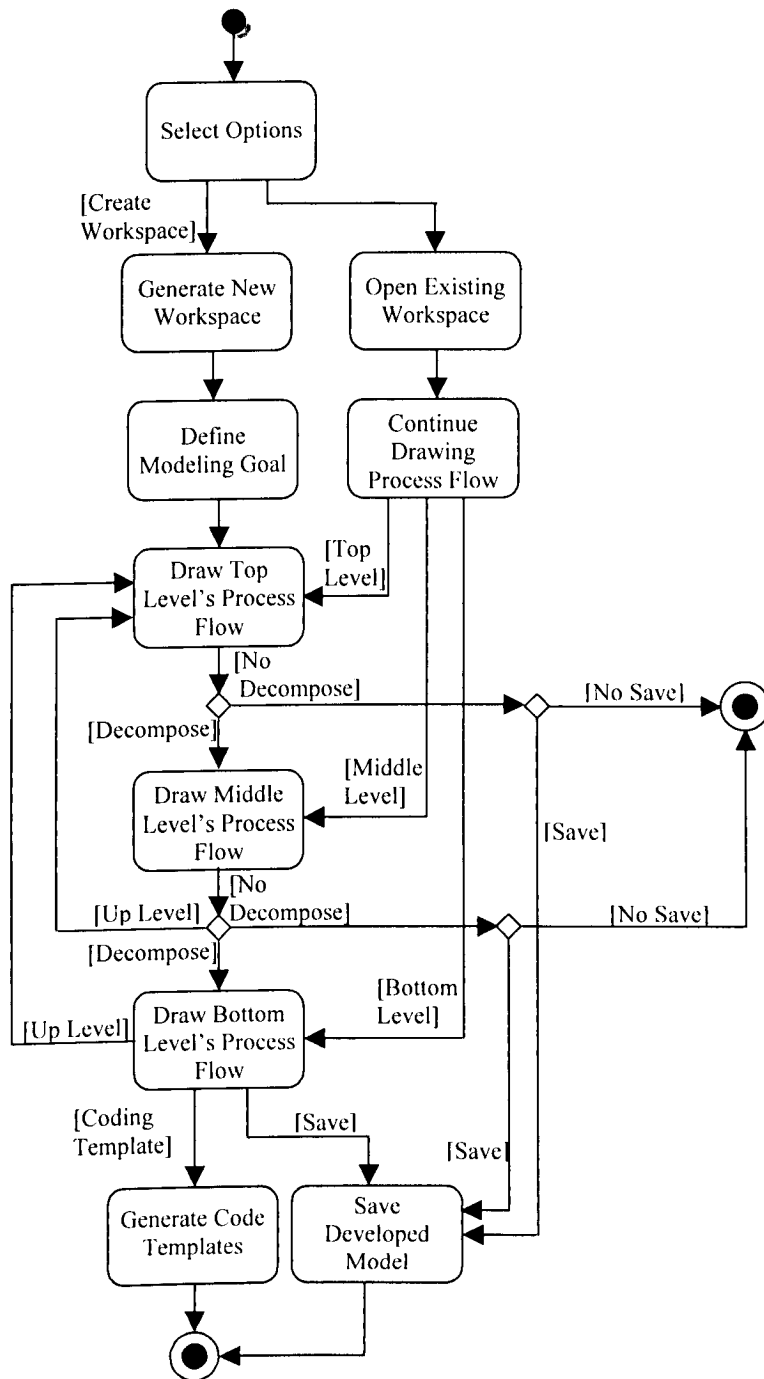


Figure 3.8: Activity Diagram of Modeling Process

3.6.2 Use Case Model of the Modeling Tool

A Use Case Model basically is made out from actor and use case. The actors are the external factors like hardware, human or other system that interact with a use case. Use cases are scenarios that represent the flow of events that are initialized by the actors, which can tell the use of the system then to understand the system requirements. The Actor represents a role with respect to the system. Therefore, a single actor may interact with many use cases and a use case may have several actors interacting with it. This is because an actor can have many roles and a use case may need different roles to complete the task. Figure 3.9 shows the Use Case Model of the Modeling Tool. In this Use Case Model, two actors are defined. They are Modeler and Programmer. Modeler in this case would be the one that understands the process flows inside the plant based on the goal of developing the model. The Modeler can perform different activities of the modeling tool which are elaborated by different use cases as shown in Figure 3.9. These use cases will be discussed later. As for the programmer, this is the category of users who will perform the customization of the coding templates generated by the modeling tool.

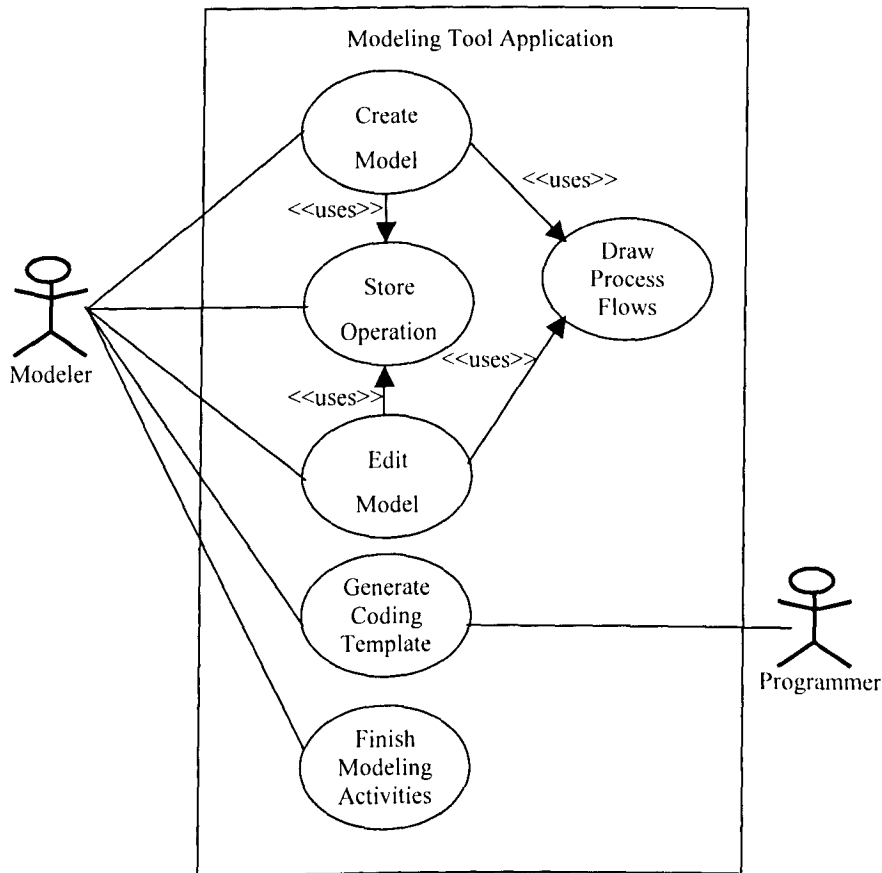


Figure 3.9: Use Case Model of the Modeling Tool

3.6.2.1 Create Model Use Case

The purpose of this use case is to show the scenario of developing a new model that can represent the process flows of a plant according to the intention of the modeler. However, the modeler needs to define the objective of developing the model before the modeler can start to create it. Table 3.1 shows the sequence of interaction between a Modeler and the system of a Create Model use case.

Actor Actions	System Responses
1. Uses the modeling tool by starting the application executable file.	2. Display the application's window without workspace.
3. Selects 'New ...'command.	4. Display 'Goal Definition' dialogue.
5. Specify the objective of developing the model.	6. Objective defined displays in the title bar.
7. Perform Draw Process Flows.	8. Display the process flow of the model at the workspace.

Table 3.1: Actor Actions and System Responses for Create Model Use Case

3.6.2.2 Edit Model Use Case

The purpose of Edit Model use case is to perform modification like adding, deleting, and changing the modeling elements of the model developed during the Create Model use case. Thus, the model must exist before any editing can be performed. Table 3.1 shows the sequence of interaction between a Modeler and the system of a Create Model use case.

Actor Actions	System Responses
1. Uses the modeling tool by starting the application executable file.	2. Display the application's window without workspace.
3. Selects 'Open ...'command.	4. Display 'File Open' dialogue.
5. Specify the model name.	7. Remove dialogue from display.
6. Confirm selection	8. Display the model of the model name specified.
9. Perform Draw Process Flows	10. Display the modified process flows of the model at the workspace.

Table 3.2: Actor Actions and System Responses for Edit Model Use Case

3.6.2.3 Generate Coding Templates Use Case

The purpose of this use case is to generate coding templates according to the model developed for programmer to perform customization. Thus, the model must exist and be complete in order that customization work can be performed by programmer. Table 3.3

shows the sequence of interaction between a Modeler and the system of Generate Coding Template use case.

Actor Actions	System Responses
1. Selects code generation command from the modeling tool application.	2. Display indication of generating coding template in progress. 3. Display message box to indicate completion of generation.

Table 3.3: Actor Actions and System Responses for Generate Coding Template Use Case

3.6.2.4 Finish Modeling Activities Use Case

The purpose of this use case is to exit or close the application after performing whatever necessary activities of modeling. Thus, the modeling tool application must be executed. Table 3.4 shows the sequence of interaction between a Modeler and the system of a Finish Modeling Activities use case.

Actor Actions	System Responses
1. Select 'Exit ...' command.	2. Remove the application's window.

Table 3.4: Actor Actions and System Responses for Finish Modeling Activities Use Case

3.6.2.5 Store Operation Use Case

The purpose of this use case is to save the model developed to the local storage. Thus, the modeler must provide a name to the model developed. This use case can be described with more details by creating child use cases. The purpose of creating child use case is to gather all the related use cases into a package to facilitate management of complexities and reduce the number of use cases in the package. Thereby Store Operation

use case can be packaged into the services it renders. Figure 3.10 shows the Package of the Store Operation which consists of two child use cases. They are Save Operation use case and Save As Operation use case.

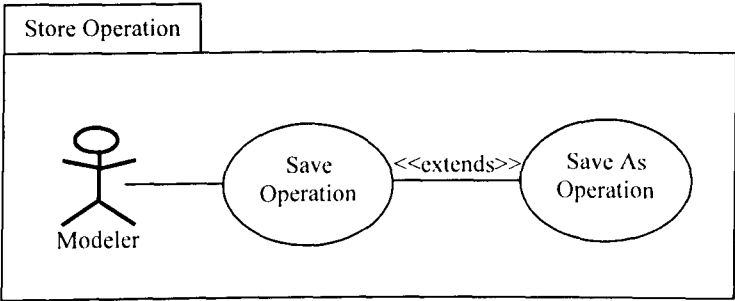


Figure 3.10: Package of Store Operation.

The purpose of Save Operation use case is to save the model developed to the specific file directory. The modeler only needs to specify the name of the model once to save the model and use the save operation to save the same model without the need to enter the name of the model. Table 3.5 shows the sequence of interaction between a Modeler and the system of Save Operation use case.

Actor Actions	System Responses
1. Selects 'Save...' command.	2. Display save dialogue.
3. Specify model name.	
4. Confirm selection.	5. Store the model developed to local file system.

Table 3.5: Actor Actions and System Responses for Save Operation Use Case

The Save As Operation use case extends the Save Operation use case. This relationship is known as extends association. Extend association is used when there is one use case that is similar to another use case but does a bit more or is more specialized; it is

like a subclass in object-oriented concept (Bahrami, 1993). This use case is used to let the Modeler save the model developed into different name or directory from the original one. Table 3.6 shows the sequence of interaction between a Modeler and the system of Save As Operation use case.

Actor Actions	System Responses
1. Selects 'Save As...' command. 3. Specify a model name. 4. Select a directory if necessary. 5. Confirm Selection	2. Display save as dialogue. 6. Store the model developed to local file system with the name specified and into directory specified.

Table 3.6: Actor Actions and System Responses for Save As Operation Use Case

3.6.2.6 Draw Process Flows Use Case

The purpose of this use case is to depict a scenario of the modeler used modeling elements like process, relation and text to develop a specific model. During drawing process flows, a concept of Modeling Tool Structure (mentioned at Section 3.3) would be applied. Draw Process Flows use case is a general use case which can be detailed by creating several child use cases. Figure 3.11 shows the Package of Draw Process Flows which provides details regarding Draw Process Flows use case.

Draw Top Level use case is used to depict the scenario of drawing the process flows at the top level of the model. In this case, the workspace of the modeling tool must be ready before the Modeler can begin to draw the process flows. The Modeler can use different modeling element to draw the process flows at the top level but Process (P) structure and

Activity (A) structure are not available for use. Table 3.7 shows the sequence of interaction between a Modeler and the system of Draw Top Level use case.

Actor Actions	System Responses
1. Select specific modeling elements from the application window and place it on the workspace.	2. The selected modeling element displays at the specific location within the workspace.

Table 3.7: Actor Actions and System Responses for Draw Top Level Use Case.

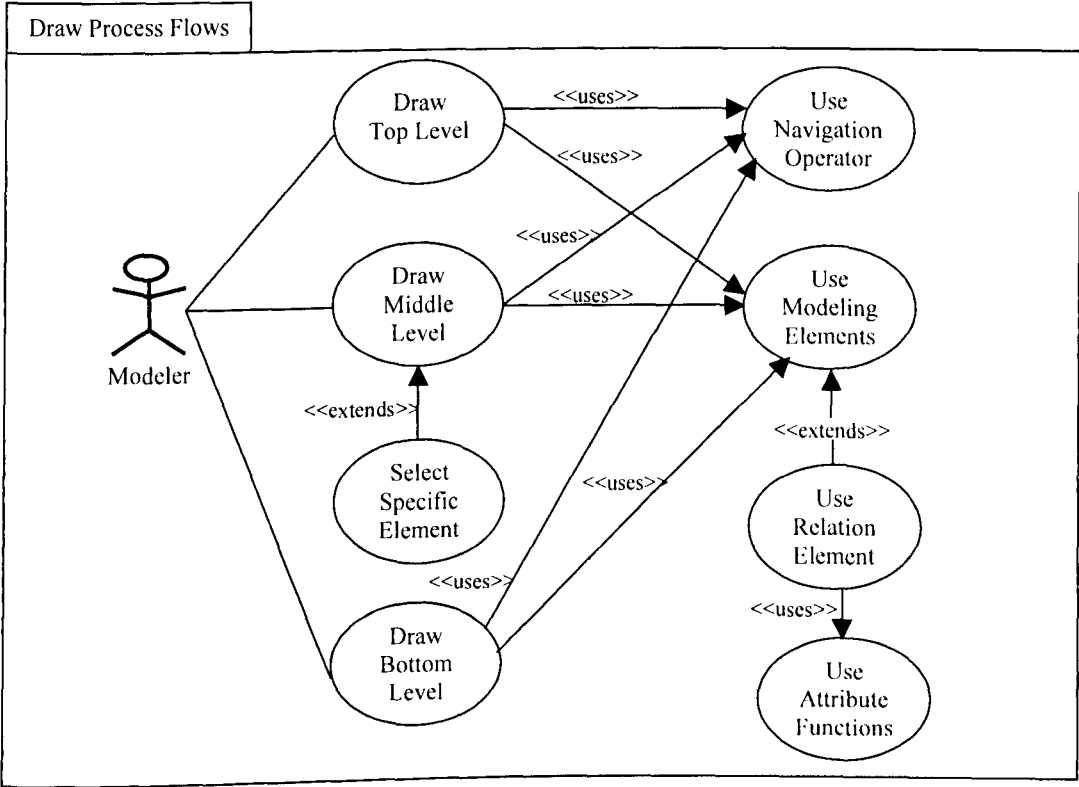


Figure 3.11: Package of Draw Process Flows

Draw Middle Level use case is used to depict the scenario of drawing the process flows at the middle level of the model. In order for the Modeler to draw process flows at the middle level, the workspace of the modeling tool must be ready for use, the top level of the model must exist and the Modeler cannot use Domain Process (DP) structure in developing

the middle level process flows. Table 3.8 shows the sequence of interaction between a Modeler and the system of Draw Middle Level use case.

Actor Actions	System Responses
1. Select specific modeling elements from the application window and place it on the workspace.	2. The selected modeling element displays at the specific location within the workspace.

Table 3.8: Actor Actions and System Responses for Draw Middle Level Use Case.

Selecting Specific Element use case extends the Draw Middle Level use case. In this case, the Modeler can select either Process (P) structure or Activity (A) structure to draw the middle level of the model. Selecting either one of them will make the other unavailable for the Modeler to use. Figure 3.12 shows that the Package of Select Specific Element consists of two child use cases. They are Select Process Element use case and Select Activity Element.

The purpose of Select Process Element use case is to let the Modeler to select the Process (P) structure in order to implement tasks relevant to it. Selection of it disables the Activity (A) structure to prevent coexistence of Process (P) structure and Activity (A) structure at the same level under one Domain Process (DP) structure. Table 3.9 shows the sequence of interaction between a Modeler and the system of Select Process Element use case.

Actor Actions	System Responses
1. Select Process element.	2. Disable the Activity element.

Table 3.9: Actor Actions and System Responses for Select Process Element Use Case.

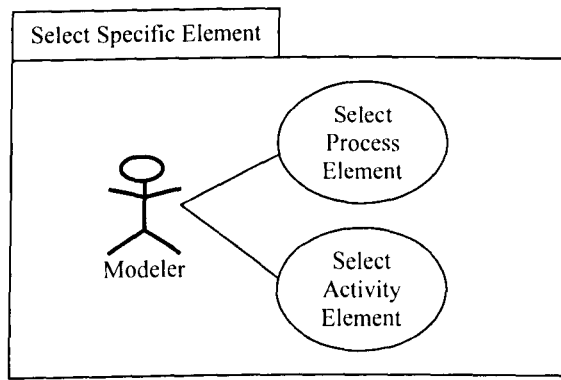


Figure 3.12: Package of Select Specific Element

The purpose of the Select Activity Element use case is to let the Modeler select the Activity (A) structure in order to implement tasks relevant to it. Selection of it disables the Process (P) structure for use by the Modeler. Table 3.10 shows the sequence of interaction between a Modeler and the system of Select Activity Element use case.

Actor Actions	System Responses
1. Select Activity element.	2. Disable the Process element.

Table 3.10: Actor Actions and System Responses for Select Activity Element Use Case.

Draw Bottom Level use case is used to depict the scenario of drawing the process flows at the bottom level of the model. In this case, the top level and middle level of the model must exist. Besides that, the Domain Process (DP) element and Process (P) structure are not available for the Modeler to use. Table 3.11 shows the sequence of interaction between a Modeler and the system of Draw Bottom Level use case.

Actor Actions	System Responses
1. Select specific modeling elements from the application window and place it on the workspace.	2. The selected modeling element displays at the specific location within the workspace.

Table 3.11: Actor Actions and System Responses for Draw Bottom Level Use Case.

Use Modeling Elements is used to let the Modeler use the modeling elements of the modeling tool to develop the model. At different levels of the model, different combination of modeling elements would be available for Modeler to use to develop the model. Thus, Draw Top Level use case, Draw Middle Level use case and Draw Bottom Level use case all use this use case in the process of developing the model. This relationship is known as Uses association. The Uses association is adopted if there are common sub-flows in the described use cases (Bahrami, 1999). These common sub-flows can be extracted to become a use case of its own. This new use case then can be used by other use cases. The Uses association avoids redundancy by allowing use case to be shared. Table 3.12 shows the sequence of interaction between a Modeler and the system of Use Modeling Elements use case.

Actor Actions	System Responses
1. Select specific modeling elements from the application window according to the requirements.	2. Based on the selection and level, certain modeling elements become unavailable for use.

Table 3.12: Actor Actions and System Responses for Use Modeling Elements Use Case.

The Use Relation Element use case is used to depict the scenario of selecting the Relation element to establish relationship between processes. Relation elements or structures can act as input to a process or as output from a process so that interaction of

processes can be established. This use case uses Use Attributes Functions. Table 3.13 shows the sequence of interaction between a Modeler and the system of Use Relation Element use case.

Actor Actions	System Responses
1. Select Relation element from the modeling elements pool and place it on workspace.	2. The Relation element displays at the specific location within the workspace.

Table 3.13: Actor Actions and System Responses for Use Relation Element Use Case.

The Use Attribute Functions use case is used to depict the scenario of how the Modeler customizes the properties of the Relation element according to its needs. Customized properties of the Relation element would act as attributes to a process. Figure 3.13 shows the Package of Use Attribute Functions. It consists of two child use cases. They are Add Attributes use case and Remove Attributes use case.

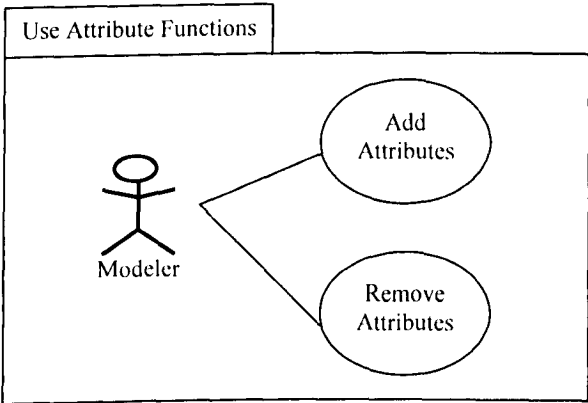


Figure 3.13: Package of Use Attribute Functions

The purpose of the Add Attributes use case is to let the Modeler add properties of the Relation element to a process as attributes. Thus, the Relation element and process

must exist so that the task can be completed. Table 3.14 shows the sequence of interaction between a Modeler and the system of Add Attributes use case.

Actor Actions	System Responses
1. Select the constructed Relation element on the workspace.	2. Display options menu for the selection.
3. Select "Add Attributes ..." command.	4. Remove options menu and display a dialogue for adding attribute.
5. Specify the type and name of the attributes	
6. Confirm entries.	7. Add entries to the Attributes List.
8. Repeat step 5 for continuing adding attribute process.	
9. Confirm Selection.	10. Remove dialogue from display.

Table 3.14: Actor Actions and System Responses for Add Attributes Use Case.

The purpose of Remove Attributes use case is to let the Modeler remove irrelevant properties of the Relation element. This would cause the attributes of a related process to be removed. Table 3.15 shows the sequence of interaction between a Modeler and the system of Remove Attributes use case.

Actor Actions	System Responses
1. Select the constructed Relation element on the workspace.	2. Display options menu for selection.
3. Select "Remove Attributes ..." command.	4. Remove options menu and display a dialogue for removing attribute.
5. Select the attributes to remove.	
6. Confirm remove selection.	7. Remove attribute from the Attribute List.
8. Repeat step 5 for continuing removing attributes process.	
9. Confirm selection.	10. Remove dialogue from display.

Table 3.15: Actor Actions and System Responses for Remove Attributes Use Case.

The purpose of Use Navigation Operator use case is to move from one level to another level based on the rules and constraints mentioned on the Modeling Tool Structure (see Section 3.3). This use case is used by Draw Top Level use case, Draw Middle Level use case and Draw Bottom Level use case. Figure 3.14 shows the Package of Use Navigator Operator. It consists of four child use cases. They are; Decompose Domain Process use case, Decompose Process use case, Return to Domain Process use case and Return to Process use case.

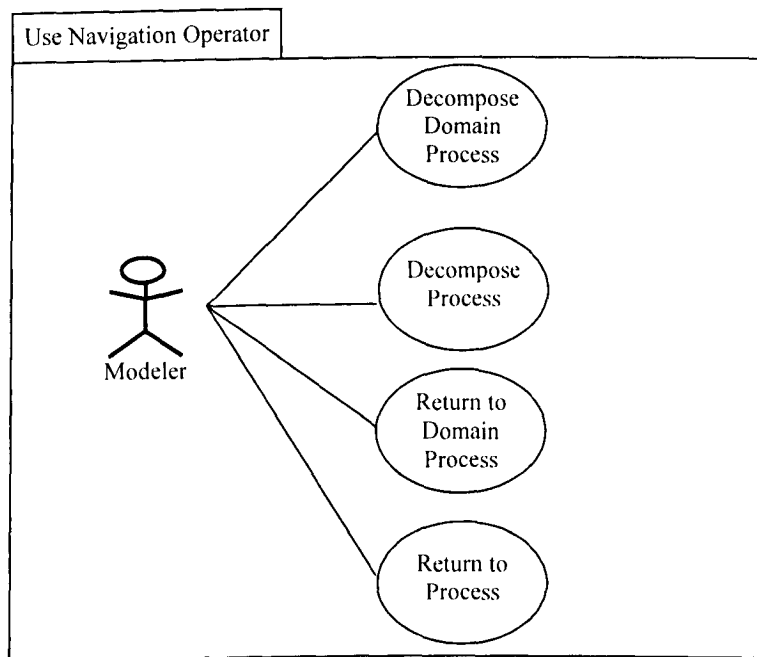


Figure 3.14: Package of Use Navigation Operator

The purpose of Decompose Domain Process use case is to move from Top Level to Middle Level by decomposing Domain Process element. Thus, when the Modeler performs this action, the Modeler must be in the Top Level of the model. Table 3.16 shows the sequence of interaction between a Modeler and the system of Decompose Domain Process use case.

Actor Actions	System Responses
1. Select Domain Process element.	2. Selected Domain Process element get the focus.
3. Select 'Decompose ...' command.	4. Move to Middle Level.

Table 3.16: Actor Actions and System Responses for Decompose Domain Process Use Case.

The purpose of Decompose Process use case is to move from Middle Level to Bottom Level by decomposing Process element. Thus, the Modeler must be in the Middle Level of the model when performing this action. Table 3.17 shows the sequence of interaction between a Modeler and the system of Decompose Process use case.

Actor Actions	System Responses
1. Select Process element.	2. Selected Process element get the focus.
3. Select 'Decompose ...' command.	4. Move to Bottom Level.

Table 3.17: Actor Actions and System Responses for Decompose Process Use Case.

The purpose of Return to Domain Process use case is to move from Middle Level back to Top Level. Thus, the Modeler must be in the Middle Level of the model when performing this action. Table 3.18 shows the sequence of interaction between a Modeler and the system of Return to Domain Process use case.

Actor Actions	System Responses
1. Select 'Up Level...' command.	2. Move to Top Level.

Table 3.18: Actor Actions and System Responses for Return to Domain Process Use Case.

The purpose of Return to Process use case is to move from Bottom Level back to Middle Level. Thus, the Modeler must be in the Bottom Level of the model when performing this action. Table 3.19 shows the sequence of interaction between a Modeler and the system of Return to Process use case.

Actor Actions	System Responses
1. Select 'Up Level...' command.	2. Move to Middle Level.

Table 3.19: Actor Actions and System Responses for Return to Process Use Case.

3.6.3 Interaction Diagrams

Interaction diagrams capture the behavior of a single use case by showing the pattern of interaction among objects (Whitman et al., 2001). Thereby, interaction diagrams are associated with a use case. The diagrams provide views of how the system runs by showing a set of actors and objects communicating with each other to perform the steps of the use case defined in Section 3.6.2. The communication is established through message passing between the actors and the objects.

A Sequence diagram is one kind of the interaction diagrams. It describes the behavior of a system by showing the interaction of participated objects through their lifelines and the messages exchange in a time sequence. Messages received by an object triggers one of its methods to execute. Sequence diagrams are therefore useful for identifying the operations that have to be included in each class (Busby & Williams, 1993). This would help in identifying classes needed to develop the modeling tool application.

Figure 3.15 shows the sequence diagram for the Create Model use case defined above. The diagram is self explanatory for communication between objects participating in the interaction. Other sequence diagrams for Use Case models discussed in Section 3.6.2 can be found in Appendix A.

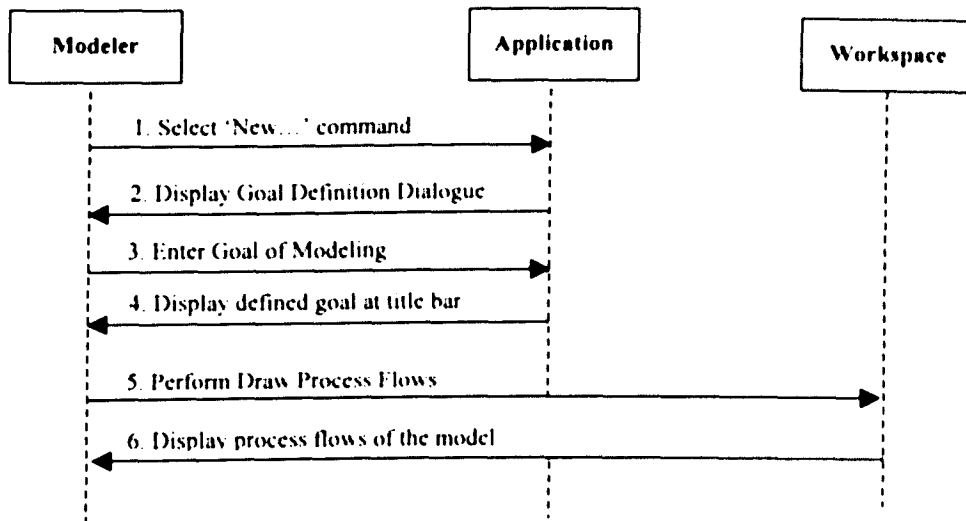


Figure 3.15: Sequence Diagram of Create Model Use Case

3.6.4 Class Diagram

After knowing how the system runs, it is essential to move forward to understand the interior of a system. Therefore, a class diagram is used to describe the structure of the modeling tool system. A class diagram is composed of classes and the relationships among them. The classes in the class diagram represent things that are necessary to make the modeling tool a functional application. They define the attributes, methods and therefore the applicability of their instances. Figure 3.16 outlines the attributes and methods of the major classes identified.

Identifying classes is one of the major activities of object-oriented analysis. This is because each class that is identified in the Class Diagram of the Modeling Tool has a purpose that would assist the achievement of the system's goals and requirements. Table 3.20 describes the details regarding the purposes of each class.

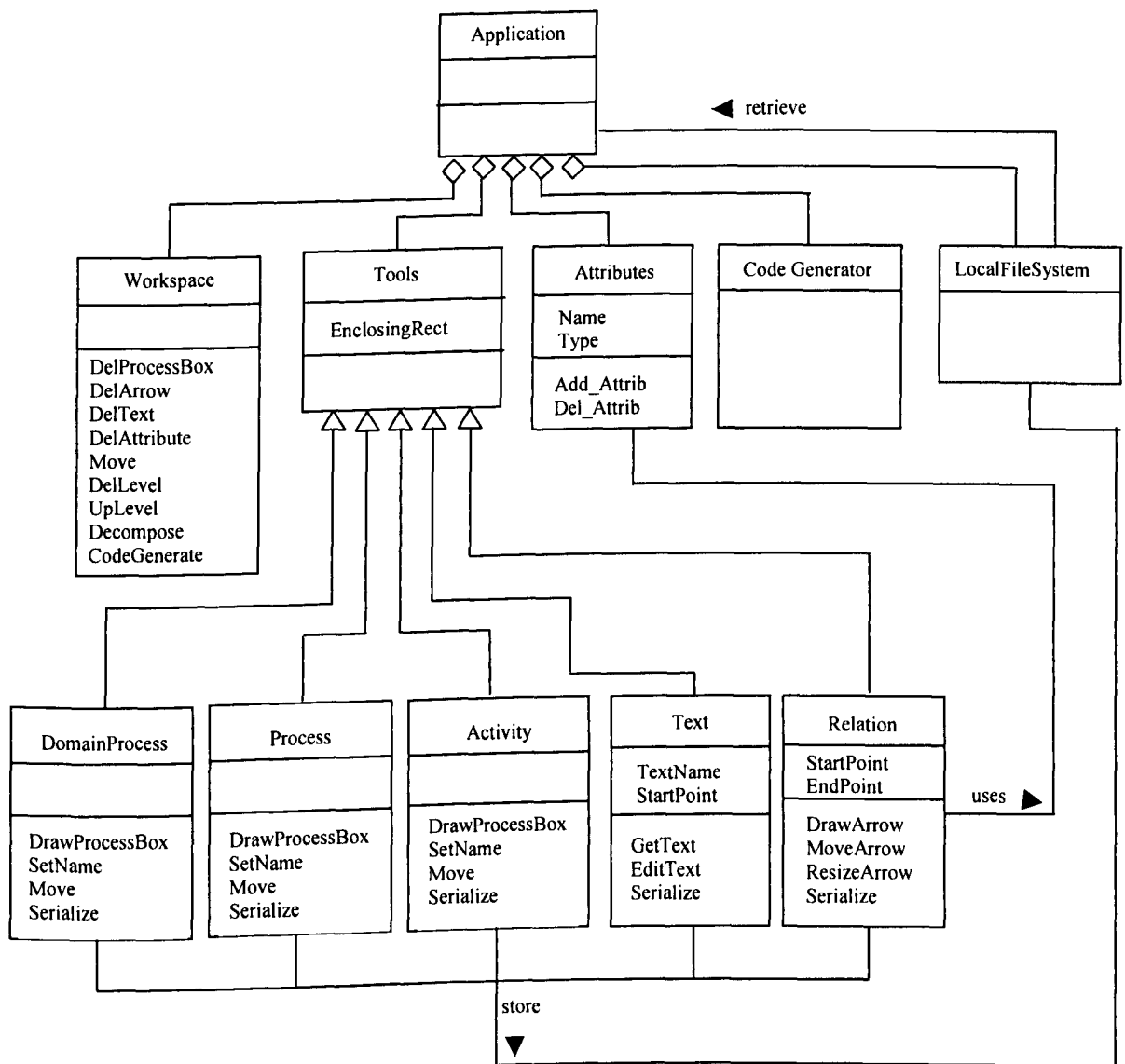


Figure 3.16: Class Diagram of Modeling Tool

Class	Purpose
Application	This is the class that provides the window, title bar, frame and etc that are necessary to display the application's window and for rest of the classes would be its components.
Workspace	Provides the space and view for the process flows to reside and display.
Tools	This is an abstract class for modeling components. It defines the common behaviors that can be inherited by more specific classes such as DomainProcess class, Process class, Activity class, Line class, Relation class and Text class.
Code Generator	This is the class that retrieves the necessary information from the model builds on the Workspace. Then, from the information gathered generates the coding templates.
DomainProcess	It models the processes at the Top Level of the subject area of a manufacturing industry.
Process	It models the processes at the Middle Level of the subject area of a manufacturing industry.
Activity	It models the processes at the Bottom Level of the subject area of a manufacturing industry.
Relation	It provides the relationship between processes at different levels of the subject area of a manufacturing industry.
Text	It provides labels, notes at the workspace to assist understanding on the model developed.
Attributes	It is the class that manages the operation of attributes that are needed by processes.
LocalFileSystem	It stores the data of the system into the local file system.

Table 3.20: Description of purposes of the classes identified at Object-Oriented Analysis

3.7 Conclusion

This chapter provides a discussion of how use-case driven object-oriented analysis uses different models to achieve the objectives and capture the requirements of this research. Through analysis, it is able to find out the “what” aspects of the studied context by developing use case models and the behavior of the ‘what aspects’ by developing sequence diagrams. Then, classes are identified based on the models previously developed. Classes identified during analysis

provide a framework for the design phase. The next chapter will describe the process of the object oriented design used to develop the modeling tool.

4.1 Introduction

In this chapter, object oriented design process is applied to the discovered modeling tools' objects during analysis. In the object oriented design phase, the emphasis is on the implementation and computer domain rather than on the application domain. In the application domain, real world entities are playing the major roles like who are the players and how they interact to perform tasks in the application as in object oriented analysis.

Object oriented design process is developed based on the object-oriented analysis by refining classes identified during analysis, defining message protocols for all objects, as well as data structures and procedures. In object-oriented approach, analysis and design are close to each other. Moving from analysis phase to design phase is like decomposition technique where the analysis phase presents a model of a real life situation for which an application is created and design provides the implementation of this model.

Object oriented design focuses on the implementation aspects of the modeling tool. Therefore, the object oriented design phase will refine and complete the class diagram identified during object oriented analysis. Then designing access and view layers classes are described in order to incorporate the modeling process structures and activities described in Chapter 3 so that the concepts of the modeling process can be converted into actual objects that can perform the required tasks.

4.2 Refine and complete the class diagram

In this section, a class's attributes, methods and associations identified during object oriented analysis are refined. The refinement can be done by changing or adding details to the classes with visibility and implementation type to elevate the class to enable implementation. Visibility in this case refers to accessibility to class. It can be public visibility, protected visibility and private visibility. Implementation type determines the type of operations that are allowed and the range of values that can be stored by attributes.

4.2.1 Refine Attributes

This process goes through Modeling Tool classes by refining existing attributes of the class by adding extra information to the attributes like visibility and implementation type. Furthermore, new attributes can be added to facilitate implementation of the class. Figure 4.2 shows the refined class diagram by adding details to the attributes of the classes identified in the object oriented analysis. The details of the class CMoToolDoc and CMoToolView are shown at Figure 4.3 and Figure 4.4 respectively

4.2.2 Design Methods

This process is to specify the algorithm by using an UML activity diagram for methods identified during object oriented analysis. Design methods' algorithms with UML activity diagram are to prepare the system for implementation, since UML activity diagram can be translated into any programming languages easily. Figure 4.1 shows the sample of the Activity Diagram for the CMTElement class GetBoundRect method. Other activity diagrams for other methods defined can be found in Appendix B.

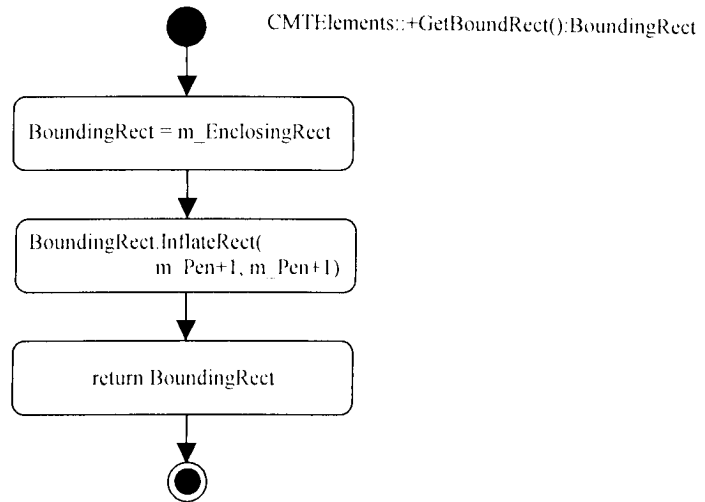


Figure 4.1: Activity Diagram for CMTElement class GetBoundRect method

4.3 Design Access Layer

The main idea of having an access layer is to create a set of classes that know how to communicate with the place(s) where the data actually reside (Whitman et al., 2001), whether it be database, file or through internet. Regardless of where the data actually resides, the access layer has to be responsible for the request and result translation. Objects in the access layer need to ensure that data request from the modeling process layer are translated into appropriate protocol for data access. An access layer must also be able to translate the retrieved data back into the appropriate modeling objects. Thereby, the access layer provides a link between the modeling object and data storage.

In this case, the objects of access layer created would communicate with the file system. However, writing a class object to a file is different from writing a basic data item like an integer or a character string. This is because class objects contain both function members as well as data members, and will have access specifiers. Furthermore, sometimes a class object might contain a variety of objects, and each may contain other objects and that structure, like inheritance, may continue for a number of levels. The file system must be able to store complete specifications of all the class structure involved. Therefore, the access layer that is designed will adopt a mechanism called serialization, which is supported by Microsoft Foundation Classes (MFC). The basic idea behind serialization is that any class that needs to be stored and then retrieved at a later date must be able to be stored to disk and retrieved later (Schildt, 1998), which is similar to the role of an access layer. In order to use serialization, document/view architecture of MFC needs to be adopted.

The reason for this is because document/view classes provided by MFC can automate the storage of documents (in this case refer to the states of objects created) to the disk files.

Figure 4.3 shows that a class diagram which is named as CMoToolDoc with its attributes and methods and is used as an access layer. This access layer provides an interface or a place for data to be transformed into an appropriate form to move between data storage or modeling objects.

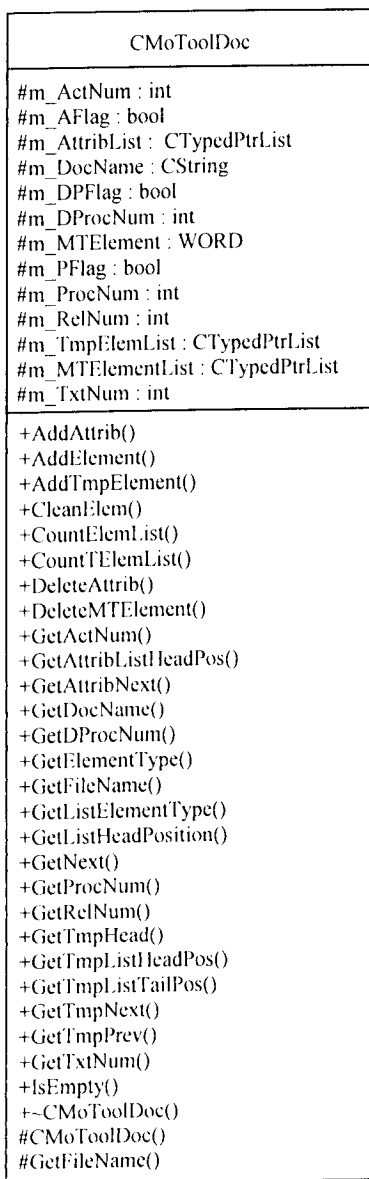


Figure 4.3: CMoToolDoc Class Diagram used as Access Layer for object storage and Interoperability.

4.4 Design View Layer

The purpose of designing a view layer which is also known as user interface layer, is to provide an environment of interaction between a user and the computer system that is as simple and natural as possible. A View layer provides an abstraction layer to the users. This is because through a view layer, users are presented with a set of operations that allow users to manipulate the modeling process without touching on the aspects of how the modeling process is implemented. A user takes an action on the view layer that is translated into a request to perform some kind of modeling processing. The view layer is updated when the processing is completed. The updated view layer can display new information, change a window's appearance; open a new window and other actions. The user knows nothing about the background activities of how to update the view layer. View layer objects are the only visible objects that user of the modeling tool application can interact with. Therefore, designing the view layer is important so as to determine how it is presented to the users.

As aforementioned, the role of the view layer objects is to handle all the interaction with the users by responding to them through sending appropriate message to the modeling object. Besides this, the view layer objects also control the interface by outputting the appropriate results to the user. Therefore, the interface object will operate as buffer between the user and the rest of the business objects (Jacobson et al., 1995), which in this case are the modeling process objects.

In order to identify and design view layer objects, it is important to understand the view layer requirements. A Use case model developed at object oriented analysis phase can assist in this matter. This is because a use case model is developed from the perspective of users. So it can help to understand the users' objectives and tasks and subsequently capture the interface requirements of the system. Besides use case models, sequence diagrams can also assist in capturing the requirements and responsibilities for the view layer objects. This is because sequence diagrams can also provide information regarding the user-system interaction. After identifying the interface objects, assign each responsibility to the class to which it logically belongs. Then, define the relationship among the view objects.

The view layer of this research would provide an interface for a set of Modeling Tools, functions for controlling the structure of the process to be created, functions for creating coding templates for the processes captured and a workspace for all the operations of the modeling to be implemented. Figure 4.4 shows the class diagram of the view layer which provides a set of attributes and methods. These attributes and methods are used to interact with the real modeling operations in the background without the necessity of knowing how the modeling operations are performed. Therefore, any changes applied to the modeling operations will not affect the view layer.

CMoToolView
<pre> # m_ArrowHeadDir : BOOL #m_BeginResize : BOOL #m_CursorPos : CPoint #m_DeclFlag : bool #m_FirstPoint : CPoint #m_FirstPos : CPoint #m_MoveMode : BOOL #m_Orgn1Point : CPoint #m_Orgn2Point : CPoint #m_pMTSelected : CMTElements* #m_pReProc : CMTElements* #m_ProcSelect : bool #m_pTmpMTElem: CMTElement* #m_RelateFlag : int #m_ResizeMode : BOOL #m_RemFlag : bool #m_SecondPoint : CPoint #m_squareBack : CPoint #m_squareFront: CPoint #m_sysLevel : int #m_TmpPID : CString #m_UpFlag : bool </pre>
<pre> +addAttribList() +DeleteAttrib() #Check2ndElemFlag() #CheckHaveAttrib() #CheckInput2Proc() #CMoToolView() #CreateElement() #DelElemAttrib() #GetUpPInfo() #MoveElement() #OnAttrib() #OnCodeGen() #OnDecompAct() #OnDecompose() #OnDelete() #OnDraw() #OnLButtonDown() #OnLButtonUp() #OnMouseMove() #OnMove() #OnProcselect() #OnRButtonDown() #OnRButtonUp() #OnRemoveLevel() #OnRename() #OnResize() #OnUpdateDecompose() #OnUpdateRemoveLevel() #OnUpdateUpLevel() #OnUpLevel() #RedunRelation() #RelEditBox() #RelSourceElem() #RetContent() #RetSubElems() #SelectMTElement() #SetConStatus() #SetUtStatus() </pre>

Figure 4.4: CMoToolView class diagram that is used as view layer for interaction between user and modeling process layer

4.5 Architecture of Modeling Tool

The Modeling Process Layer is the core or center of this modeling tool. The View Layer provides an interface of the modeling tool. The creation of modeling objects and issuing actions of the modeling tool are sent through the View Layer to the Modeling Process Layer. Then the Modeling Process Layer carries out the tasks required and returns the results of the actions back to the View Layer. The Access Layer performs the actions that are required by the Modeling Process Layer by storing or retrieving the modeling objects data to where directed by Modeling Process Layer. Figure 4.4 shows the relationship of View Layer, Modeling Process Layer and Access Layer.

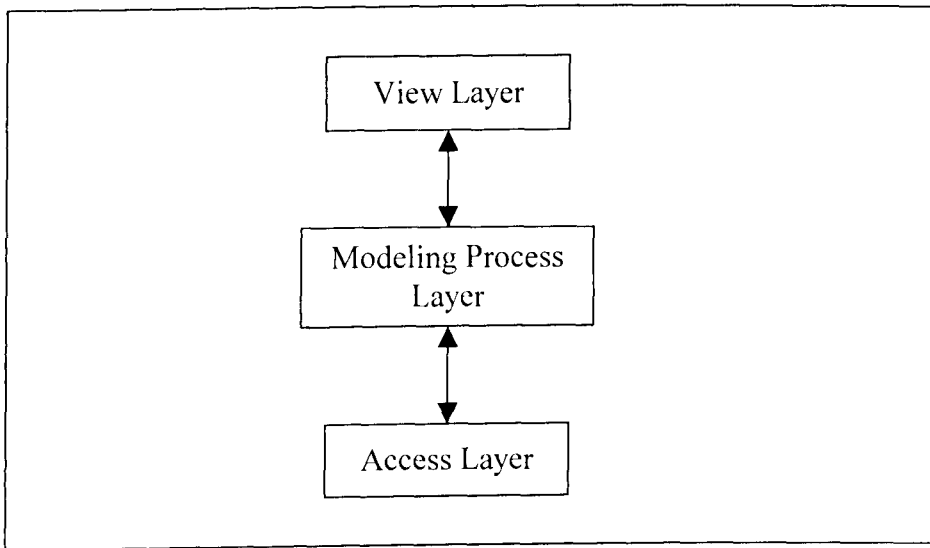


Figure 4.5 Relationships of View Layer, Modeling Process Layer and Access Layer

Based on the relationships of View Layer, Modeling Process Layer and Access Layer, the architecture of the Modeling Tool is designed, which is shown in Figure 4.5. In this case, the Workspace represents the View Layer, which provides the interface like Modeling Tools icons, file menus and others for interaction between a user and the Modeling Tool application. The Modeling Tools, Structure Controller, Code Generator and Code Engine represent Modeling Process Layer and File System represents the Access Layer.

Modeling Tools is a set of classes which consists of Domain Process class, Process class, Activity class, Relation class and Text class. These classes are used to create components that are needed in order to create a model in the Workspace. These components are related to the Domain Process (DP) structure, Process (P) structure, Activity (A) structure and Relation structure which are discussed at Chapter 3. They are used to build the structure of a process model which can provide a logical view of the process flow in the shop floor. Structure Controller is a set of functions that are used to create and organize the structure of the processes created in the Workspace. Creating a child process from a parent process is one of the tasks of the Structure Controller. The Structure Controller ensures the processes created in the Workspace are according to the protocols of the Domain Process (DP) structure, Process (P) structure, Activity (A) structure and the Modeling Tool Components and Activities mentioned in Chapter 3. For example, creating a Domain Process (DP) structure at Level 1, Structure Controller would only allow operation of decomposition to the next level. Moving from child level to its parent level only occurs when there is parent and child relationship established in the process. Modeling Tools and Structure Controller have a close relationship. This is because Modeling Tools and Structure Controller are devised from the Modeling Tool structure and Relation Structure

which are discussed in Section 3.3 and Section 3.4. They are separated into two different components even though they are from the same concepts. This is to facilitate the design and implementation of the application purpose.

The Code Generator is the function that is used to create the coding templates from the Modeling Tool structure and Relation structure that are used to create the model in the Workspace. Even though Text is one of the components of the Modeling Tool, it is not taken into consideration during the process of generating coding templates from the model developed. The Text component only serves the purpose of explanation and clarification of the model developed. For example, as Section 3.5 mentioned, definition of goal for modeling is the initial step of the model development. Thereby, Text component can be used to state the goal of the modeling in the Workspace. This can serve as navigator for the modeler to develop the model and as well as a communication media for the other fields of functionalities to understand the model developed.

A model that is developed in the Workspace is made up from Domain Process (DP) structure, Process (P) structure, Activity (A) structure and Relation structure. This model is developed under constraints and rules that are governed by the Structure Controller. These structures that are created in the Workspace are basically stored in a Buffer with special identification on them. Then the Code Generator would retrieve the modeling objects that are stored in the Buffer and then pass all the modeling objects data to the Code Engine for creating coding templates.

At the Code Engine, the Gatherer would receive the modeling objects data and dispatch the data to appropriate holders based on the identification assigned to each of the Modeling Tool structures and Relation structures. The purpose of the holders is to maintain

a set of related data so that the data can be retrieved in an organized way. The Detector retrieves the data from the holders of the Gatherer and then analyzes the data. The analysis process is applied using a top down approach to examine the data retrieved. The Detector would first look for primary Domain Process (DP) structure that can trigger the operations from the Buffer. Then, it would issue a command to create a coding template. The necessary code for that Domain Process (DP) structure would be created at the coding template. After that, it would search for its child level by determining whether the child level is Process (P) structure or Activity (A) structure. If it is Middle level, Process (P) structure will come into existence, then the Detector would issue command to create new coding template with necessary codes to be created for the primary Process (P) structure. Then the Detector needs to dig deeper to look for the Process (P) structure's associated child level, which is at the Bottom level where the Activity (A) structure will play its roles. All the necessary information of the primary Activity (A) structure like its properties is collected. Then programming codes are added to the coding template created for the primary Process (P) structure. It would move horizontally to search for its next Activity (A) structure that is /are connected to it. Programming codes are added to the coding template if the Detector discover a new Activity (A) structure. This process would continue until all the connected Activity (A) structures of that level are examined. Then, the Detector would return to parent process of the Activity (A) structures that are under examination. It would determine the level at which the parent process resides. This determination process is based on simple formulas:

- $\text{Level 2} + \text{Bottom Level} = \text{Domain Process (DP)}$
- $\text{Level 3} + \text{Bottom Level} = \text{Process (P)}$

If the child process is at Level 2 and Activity (A) structure appears then its parent process must be Domain Process (DP) structure. However, if the child process is at Level 3 with Activity (A) structure presents then the parent process must be Process (P) structure.

After returning from child level to the parent process, the Detector would examine the next process structure (which either can be Domain Process (DP) or Process (P) structure). Programming codes are added for the new process structure detected (either Domain Process (DP) or Process (P) structure). The new process structure is examined for its child level and collects all the necessary data from its child level and programming codes are created if there exists a child level. This process would continue to loop until the last Domain Process (DP) structure and its associated child level/s are examined. This indicates all the data inside the holders are examined and retrieved.

The created coding templates are stored at the Local File System set by the Modeling Tool Application. These coding templates are kept in a specific folder created by the Modeling Tool Application to be utilized by the modeler. The Modeler can customize the coding templates to design an application that can perform the tasks required by the modeler. The Local File System and File System (refer to Figure 4.5) are different components. The Local File System is used to keep the coding templates that are needed by the user of the Modeling Tool Application. However, the File System keeps the data that is needed by the system in order to operate the system in an organized and systematic way. This can be seen from how the user retrieves the model that is developed by using the Modeling Tool Application where the model needs to be presented in the way it was last stored into the File System. Any modification to the model needs to be reflected by the File

System. This enables the modeler to know the latest status of the model that is under development.

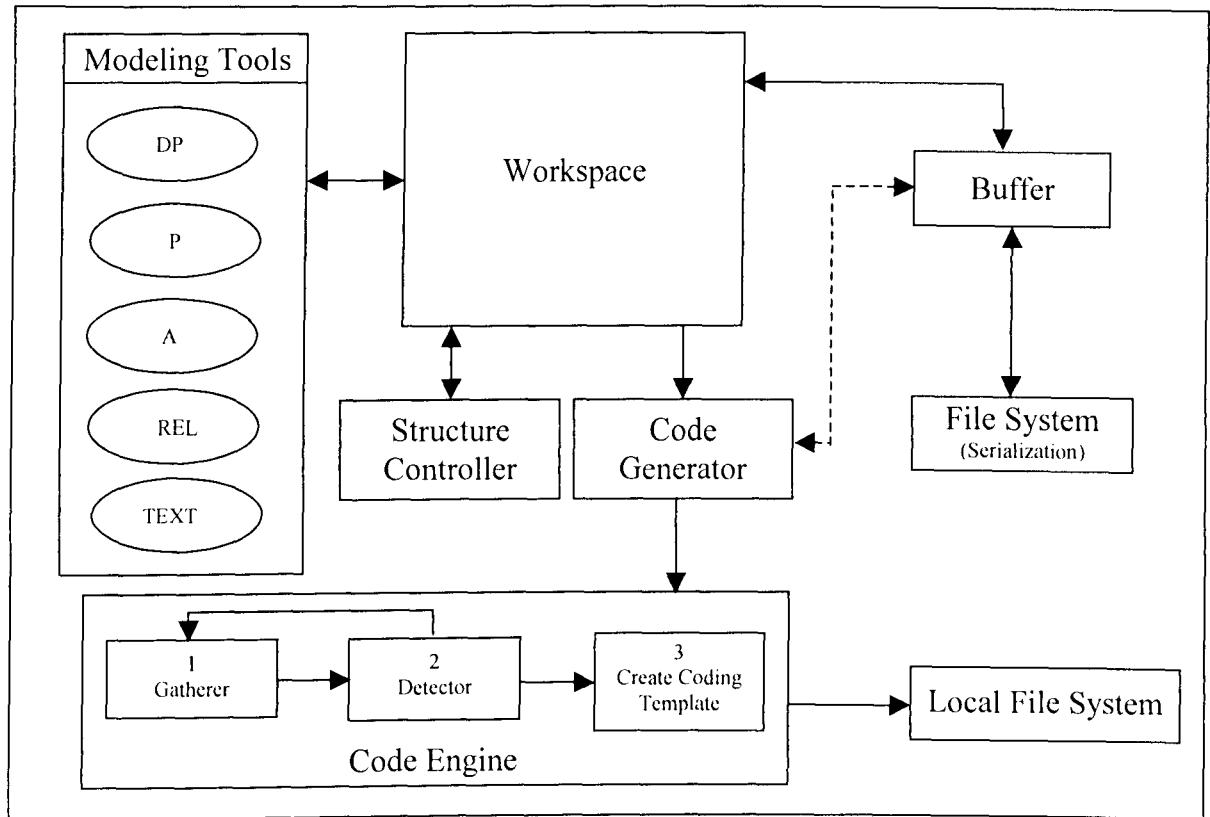


Figure 4.6 Architecture of Modeling Tool

4.6 Conclusion

The process of designing the user interface involves clarifying the specific needs of the application, identifying the use cases and interface objects, and then devising a design that best meets the users' needs. This chapter has presented the design and architecture of the Modeling Tool Application. Chapter 5 will discuss on the implementation of the Modeling Tool Application where the outlooks of the application would be showed to relate to the information discussed at Chapter 3 and Chapter 4.

5.1 Introduction

In Chapter 3 and Chapter 4, the requirements of the modeling tool and the designs that are needed to implement the modeling tool were discussed. This chapter however presents the implementation of the modeling tool based on the object-oriented analysis and design discussed in Chapter 3 and Chapter 4, respectively.

5.2 Description of Modeling Tool Application

In this section, some of the major snapshots from the developed Modeling Tool Application are presented. These snapshots are used to relate the concepts of the Modeling Tool Structure and the work of the designs that are needed to develop the application. A brief description is associated with each of the snapshots

5.2.1 Modeling Tool Overview

Figure 5.1 shows the appearance of the Modeling Tool Application. This is known as the view layer (mentioned in Section 4.5), which provides the user interface for the application. Through this view layer, different interface objects like icons, menu and other interface objects are used to facilitate users sending commands to the core for processing and responses from the core are diverted to the view layer. Users are not aware of how the processes and operations are implemented in the core. In this view layer, interface objects like Structure Controller, Modeling Tools and Code Generator, and File System provide a set of operations for the users to perform in order to complete their modeling tasks. The

Workspace, Level Indicator and Parent Process Indicator provide environment and information for users to use the application. For examples, the Level Indicator specifies the level of a process currently engaged in, the Parent Process Indicator provides information regarding the parent of a process by notifying the user that a process is belonging to a bigger process.

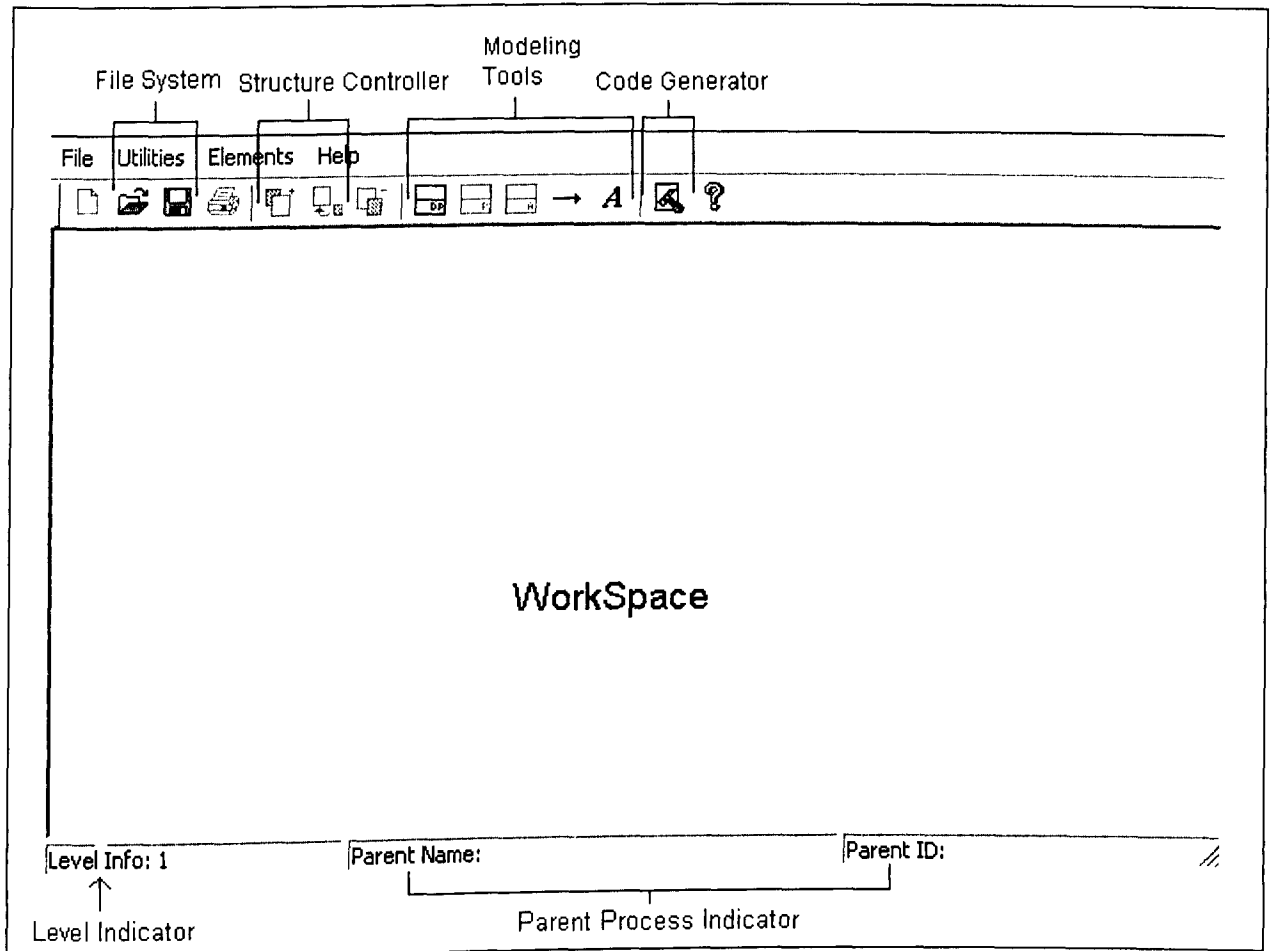


Figure 5.1: Overview of Modeling Tool Application

The development tool used to build the Modeling Tool Application is Visual C++ 6.0.

Figure 5.2 shows the part of the source code used to create the application frame window for the Modeling Tool program.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;    // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;    // fail to create
    }

    m_MTStatusBar.Create(this);

    //int width = textRect.Width();
    int width[3] = { 150,400,600 };
    m_MTStatusBar.GetStatusBarCtrl().SetParts(3, width);

    m_MTStatusBar.GetStatusBarCtrl().SetText("Level Info: 1", 0, 0);
    m_MTStatusBar.GetStatusBarCtrl().SetText("Parent Name: ", 1, 0);
    m_MTStatusBar.GetStatusBarCtrl().SetText("Parent ID: ", 2, 0);

    return 0;
}
```

Figure 5.2: Part of the source code used to create the application frame window

5.2.2 Tools used for modeling

Section 5.2.2.1 to Section 5.2.2.5 briefly describes information regarding Modeling Tools. There are five different icons used to represent the Modeling Tools and these icons apply to the concepts of the Modeling Tool Structure and Relation Structure, mentioned in Section 3.3 and Section 3.4.

5.2.2.1 Domain Process Structure Representation

Figure 5.3, the icon, DP, is used to create Domain Process Structure. The rectangle box with process name, DProcess_1 and process ID, DP1 is created in the Workspace to represent the Domain Process (DP) structure. Since Domain Process (DP) structure only come into existence at Level 1, therefore, the DP icon would be disabled if moved to the next level. Users cannot create any Domain Process Structure other than at the Top Level

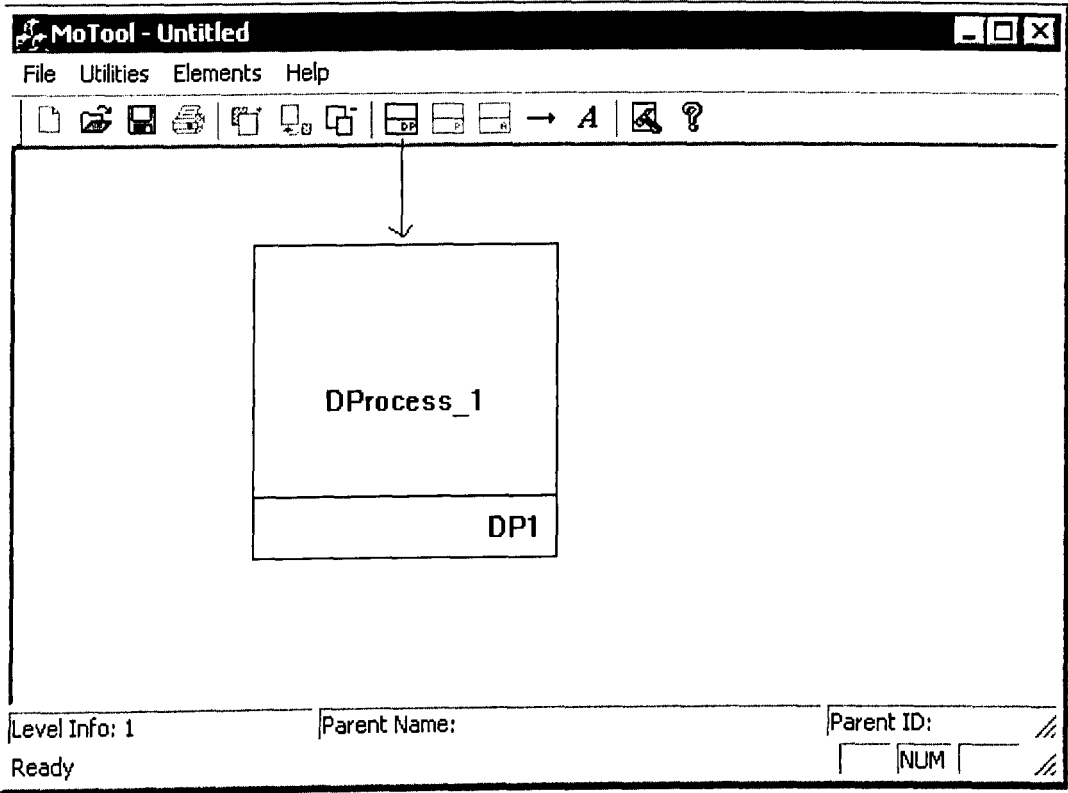


Figure 5.3: Icon used to represent the Domain Process Structure

Figure 5.4 shows the class definition for the class ‘CDProcess’. This is the data type that is used to create the Domain Process (DP) structure objects in the Workspace. It shows the members of the class that can be manipulated by Modeler.

```
class CDProcess: public CMTElements
{
    DECLARE_SERIAL(CDProcess)
public:
    virtual void Draw (CDC *pDC); //Function to display a DProcess
    virtual void Draw(CDC* pDC, int aMode);
    CDProcess(CPoint Start, CString aNum, int aLevel);    //Constructor for a DProcess
object
    virtual void Move(CSize& aSize);
    virtual void Serialize(CArchive& ar); //Serialize function for CDProcess
    virtual CString GetElemID() { return m_ElemID; }
    virtual int GetsysLevel() { return m_Level; }
    virtual CString GetProcName() { return m_ProcName; }
    virtual void SetProcName(CString aName) { m_ProcName = aName; }

protected:
    CDProcess() {}    //Default constructor

private:
    CString SetNum(CString nStrNum);
};
```

Figure 5.4: Class definition for the Domain Process object

5.2.2.2 Process Structure Representation

In Figure 5.5, the icon, P, is used to create the Process (P) structure. The rectangle box with process name, Process_1 and process ID, P1 is created in the Workspace to represent Process (P) structure at Level 2. Process Structure is only allowed to exist in Level 2; moving to Level 1 or Level 3 would automatically disable the icon’s functionality.

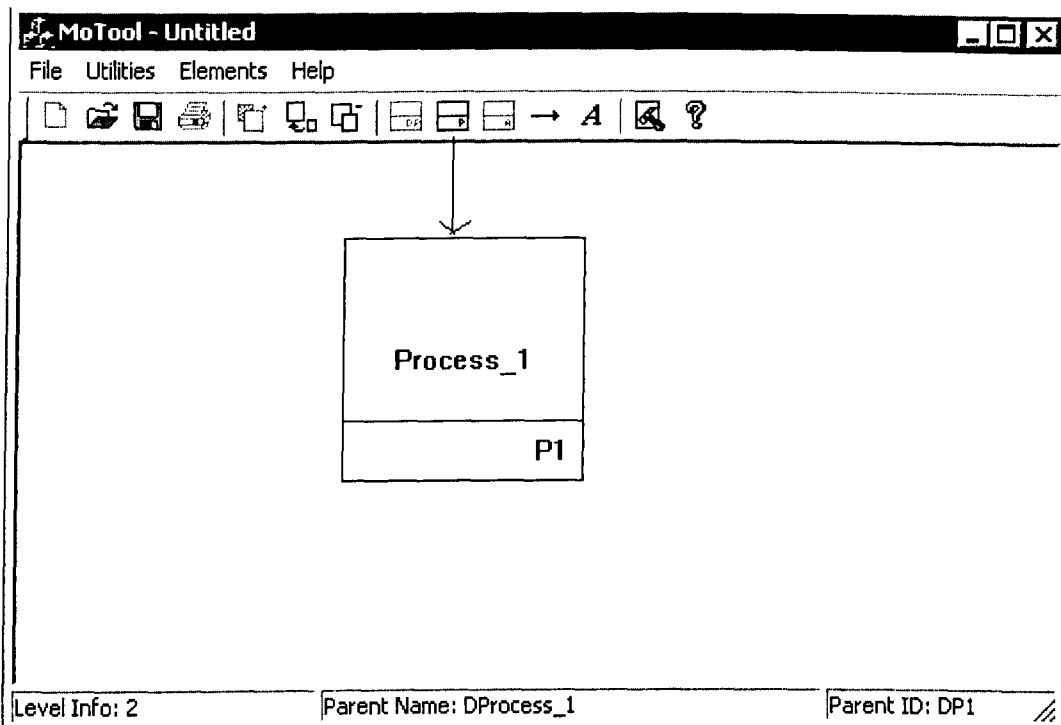


Figure 5.5: Icon used to represent Process Structure

5.2.2.3 Activity Structure Representation

In Figure 5.6, the icon, A, is used to create the Activity (A) structure. The rectangle box with process name, Activity_1 and process ID, A1 will be created in the Workspace to represent the Activity (A) structure. Since the Activity (A) structure is the only process structure that is allowed to exist in Level 2 or Level 3 and it is the final and lowest level of the Modeling Process Structure, moving away from the current level of the Activity Structure would disable the icon.

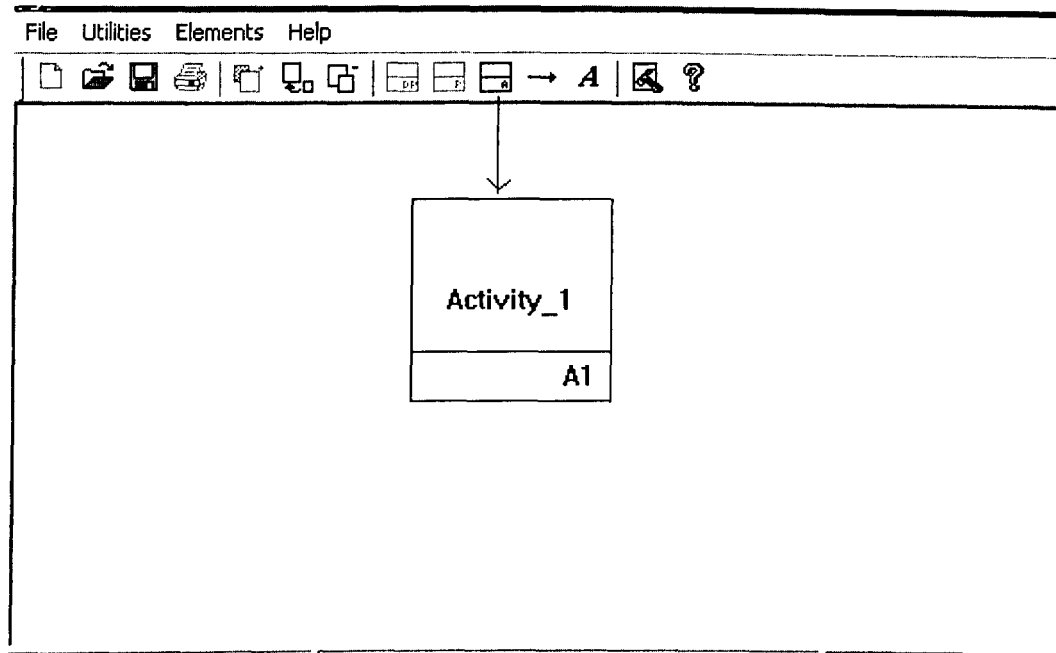


Figure 5.6: Icon used to represent Activity Structure

5.2.2.4 Relation Structure Representation

In Figure 5.7, the icon with an arrow symbol is used to create a relationship between process structures. The relationship between the processes structures will be established only when two ends of the Relation Structure connect to the side of the process structures. There is only one exception whereby the Relation Structure needs to connect only to left side of the process structure. This type of relation structure is known as the Predefined kind Relation Structure.

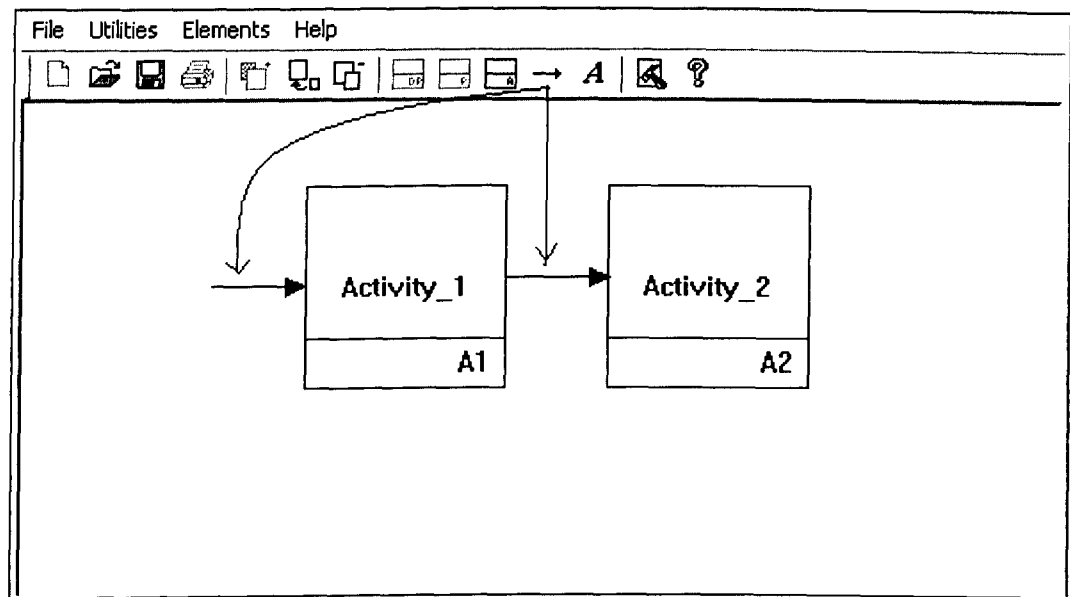


Figure 5.7: Icon used to represent Relation Structure

5.2.2.5 Text Representation

In Figure 5.8, the icon with label A is used to create the text. The Text function is used to allow users to provide a more comprehensive and understandable environment in the process of modeling. This can be carried out by placing text or labels at suitable position in the Workspace.

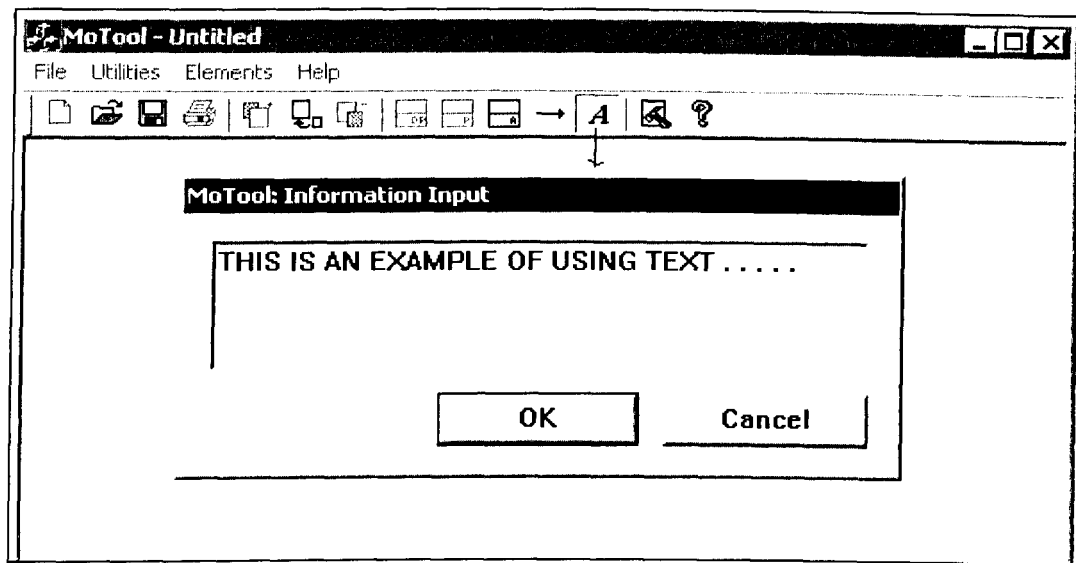


Figure 5.8: Icon used to represent Text

5.2.3 Functionalities of Structure Controller

Section 5.2.3.1 to Section 5.2.3.2 describes the Structure Controller. The functions of the Structure Controller are as described in Modeling Tool Structure (Section 3.3). The Structure Controller allows users to move to different levels to view and work around different information under the constraints that are posed by the system. Therefore, the functions of the Structure Controller have a close relationship with the concepts applied in the Modeling Tool Structure.

5.2.3.1 Decomposition Interface Object

Figure 5.9 shows the icon used for decomposition. This icon is used to decompose a process structure. Initially, the icon would be disabled. It would be enabled only when the

Domain Process (DP) structure or Process (P) structure is selected. This is to create an environment where all the related processes are gathered under the same roof. The decomposition function is disabled when it comes to Activity (A) structure. This is because Activity (A) structure indicates the final and lowest level of the Modeling Tool Structure.

Figure 5.10 shows the major part of the source codes that are used to perform the operation of decomposition. The 'OnDecompAct ()' method is used to gather the information regarding the child level where it contains the details related to its parent process. The rest of the source codes are used to set the information for the new level after performing the decomposition operation.

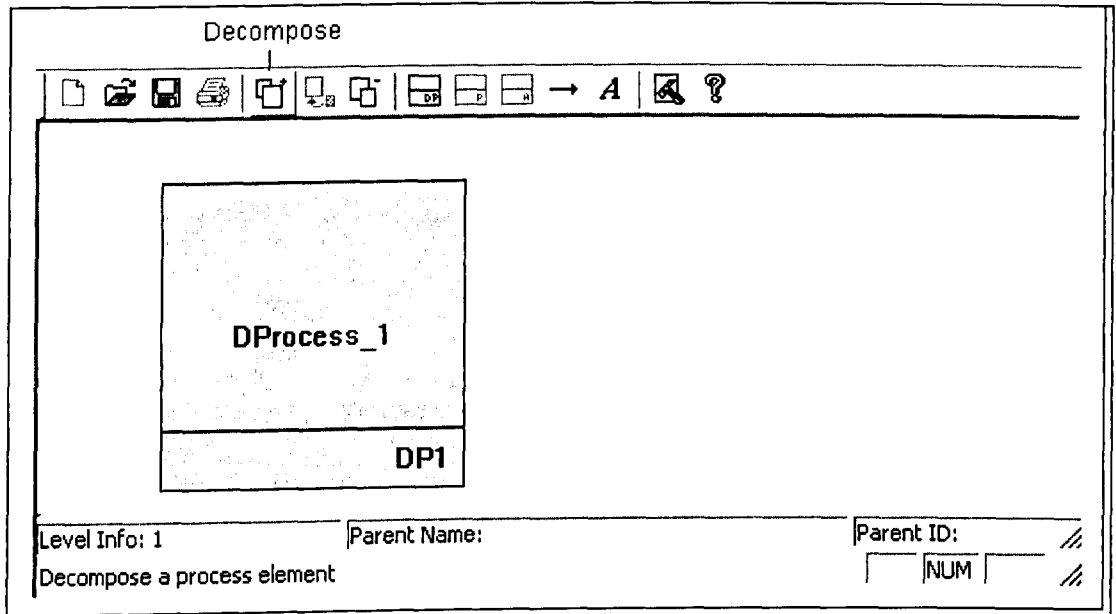


Figure 5.9: Structure Controller with the functionality of decomposition a process


```

OnDecompAct();

    if(m_sysLevel > 3)
        m_sysLevel = 3;

    m_ProcSelect = false;
    m_pMTSelected = 0;

    InvalidateRect(0);
    m_DecFlag = false;
    m_UpFlag = true;

    if(m_sysLevel == 2)
    {
        elemType = Check2ndElemFlag();
        switch (elemType)
        {
            case PROCESS:
                pDoc->SetProcFlag(UPLEVEL, 3);
                return;
            case ACTIVITY:
                pDoc->SetProcFlag(DECOMP, 2);
                return;
            default:
                pDoc->SetProcFlag(DECOMP, 1);
                return;
        }
    }
}

```

Figure 5.10: Example of source code for decomposing method.

5.2.3.2 Up Level Interface Object

Figure 5.11 shows the icon used to represent the Up Level function. The purpose of this icon is to move from a lower level to a higher level. Initially, the icon would be disabled. It would be enabled only when it comes to the level where Process Structure or Activity Structure comes into existence. Since Domain Process Structure is the top level of the Modeling Tool Structure. Therefore, this icon is disabled in that level.

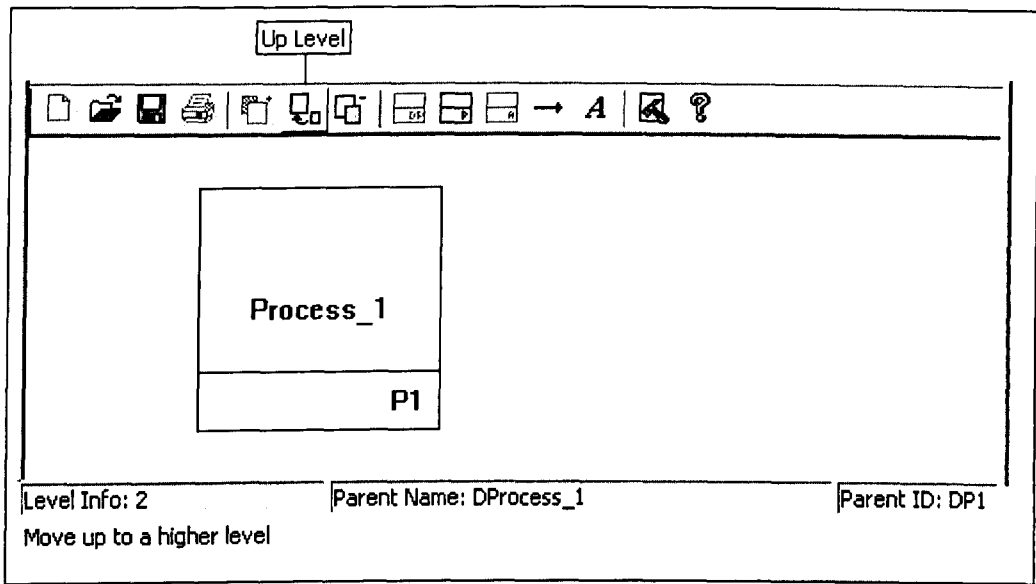


Figure 5.11: Structure Controller with the functionality of moving to upper level

Figure 5.12 shows the main part of the source codes to perform the operation of Up Level functionality. This function would gather its parent information before it returns from current level to new level and set the parameters for the new level.

```

m_sysLevel = m_sysLevel - 1;

    aPos = pDoc->GetListHeadPosition();
    while(aPos)
    {
        pElement = pDoc->GetNext(aPos);
        if((pElement->GetsysLevel() == m_sysLevel))
            pDoc->AddTmpElement(pElement);
    }

    StrLevel.Format("%d", m_sysLevel);
    if(m_sysLevel == 1)
        pFrame->SetLevelPaInfo(StrLevel, enT, enT);
    else
    {
        aPName = GetUpPInfo();
        pFrame->SetLevelPaInfo(StrLevel, aPName, m_TmpPID);
    }

    pDoc->SetProcFlag(UPLLEVEL, m_sysLevel + 1);

    m_ProcSelect = false;
    m_pMTSelected = 0;
    InvalidateRect(0);

    if(m_sysLevel == 1)
        m_UpFlag = false;
    else
        m_UpFlag = true;

```

Figure 5.12: Example of source codes for Up Level functionality

5.2.4 Code Generator and File System Interface Objects

Section 5.2.4.1 and Section 5.2.4.2 describe information regarding the File System and Code Generator respectively. The File System is related to Access Layer which is mentioned in Section 4.3. Request and result translation are handled by the Open and Save interface objects.

5.2.4.1 File System Interface Object

In Figure 5.13, there are two icons used to represent the interface object for the File System. These two icons, Open and Save are common icons in Microsoft system. Open

object is used to retrieve complete or in progress modeling data, and the Save functions are the same as the functions of the Access Layer mentioned in Section 4.3.

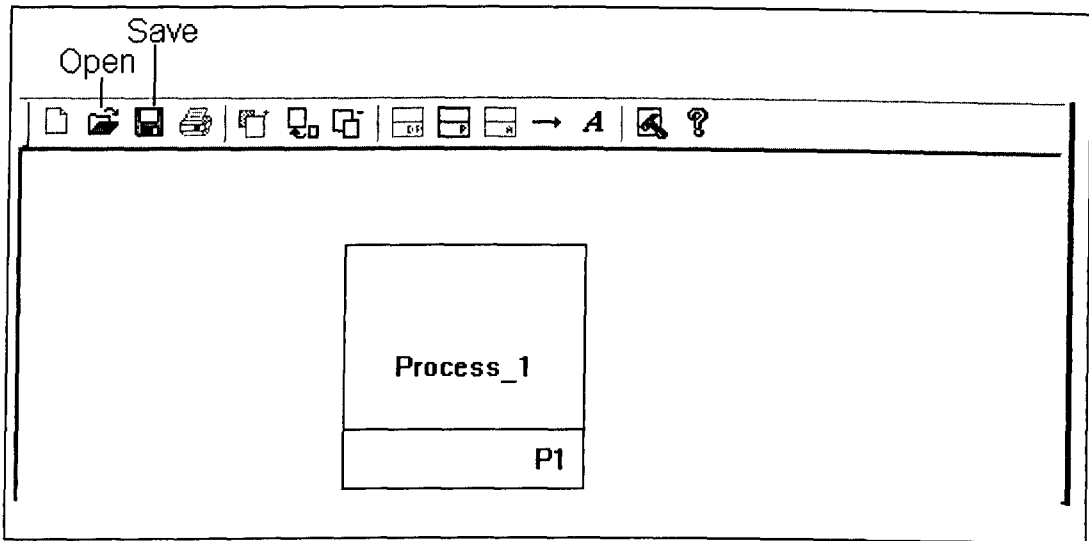


Figure 5.13: Interface Objects used for File System

Figure 5.14 shows the source codes used to retrieve modeling data from a local file system. It first locates the place where the data is stored and then retrieves the data from the local file system.

```

BOOL CMeToolDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    CMTElements* pElement = 0;
    POSITION aPos = m_MTElementList.GetHeadPosition();

    while(aPos)
    {
        pElement = m_MTElementList.GetNext(aPos);
        if(pElement->GetsysLevel() == 1)
            m_TmpElemList.AddTail(pElement);
    }

    return TRUE;
}

```

Figure 5.14: Example of the source code for opening a file

5.2.4.2 Code Generator Interface Object

Figure 5.15 shows the icon used to represent the Code Generator. This is the icon used to generate coding templates after completing the modeling process. The engine will gather the information and detect all the necessary elements in order to generate the templates.

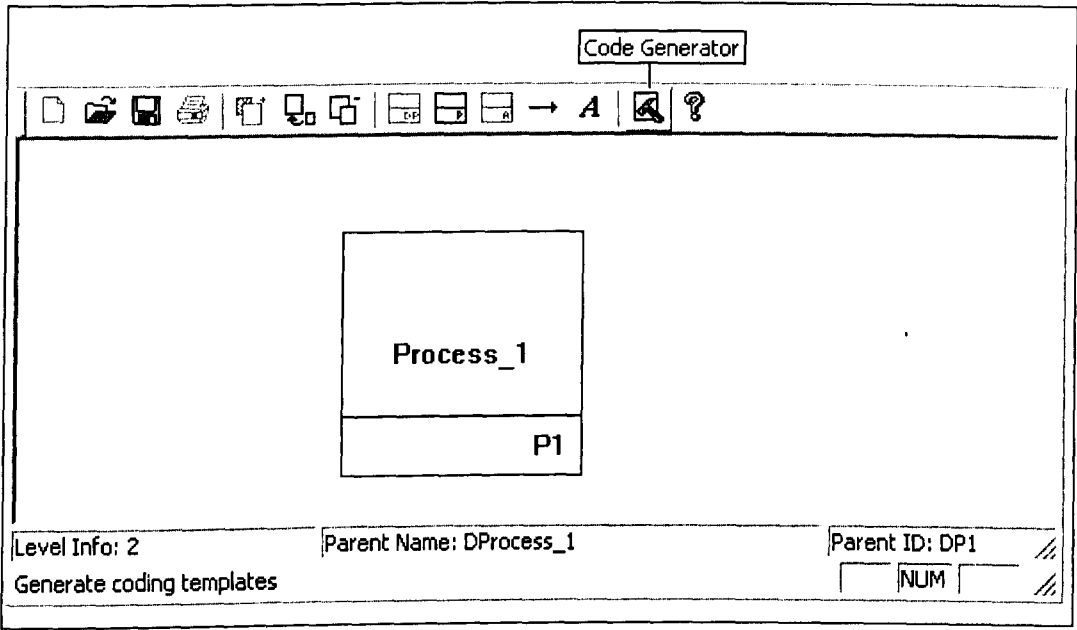


Figure 5.15: Interface object used to represent Code Generator

Figure 5.16 shows part of the source codes used to implement Code Generator functionality. Initially, it will gather all the necessary information that is needed to perform code generation before calling the 'StartCenCode()'. This method performs the core of the coding generation.

```

POSITION aPos = pDoc->GetListHeadPosition();
CMTElements* pElement = 0;
while(aPos)
{
    pElement = pDoc->GetNext(aPos);
    //this is to transfer all the elements from m_MTElementList to m_CodeElemList
    //to do the process of generating Coding Templates
    if(pElement->GetElemType() != TEXT) //Text element is no needed
        pMTCodeGen->AddCodeList(pElement);

}

pElement = 0;
aPos = pDoc->GetAttribListHeadPos();
while(aPos)
{
    pElement = pDoc->GetAttribNext(aPos);
    pMTCodeGen->AddAttribList(pElement);
}

nMTPath.SetMTPath();
pMTCodeGen->SetMTPath(nMTPath.GetMTPath());
result = pMTCodeGen->StartGenCode();

```

Figure 5.16: Example of the source codes for Code Generator

5.3 Conclusion

In this chapter, some of the major interface objects are introduced and described. Through these interface objects, Modeling Tool application provides an interface where users are only dealing with the view layer and are separated from the modeling process layer. The next chapter will look at the application of the modeling tool and techniques in a real life example.

6.1 Introduction

This chapter presents an application, which is implemented using the modeling tool presented in Chapter 5.

This application is developed based on a case study. This case study is implemented on one of the manufacturing industries in Kuching, Sarawak. This manufacturing industry is known as FFM Flour Mills (SARAWAK) SDN BHD. It is a plant that produces flour to supply the demand of flour in Sarawak. FFM Flour Mills (SARAWAK) SDN BHD was established in the Year 2003. FFM (Ling, 2004) basically produces the flour by milling the raw wheat which is imported from Australia, Canada and United States.

The purpose of this case study is to use the Modeling Tool Structures and Techniques to capture the relevant processes involved in the certain fields of the manufacturing industries. From there, coding templates based on the processes captured or identified are then generated. Then, these coding templates are customized in order to develop an application that is suited to the needs of the party involved. By doing this case study, the applicability of the modeling tool developed can be tested. In section 6.2, the manufacturing process of the FFM is briefly described to gain an understanding of how the process flow of the production of flour from the raw material, wheat is implemented.

6.2 Description of Manufacturing Process of FFM

In this section, the operations of the FFM production of flour will be discussed. The information provided in this section was the result of discussion and interview with Mr.

Ling Jai Seng, Plant Engineer of FFM Flour Mills (S) Sdn Bhd (Ling, 2004) and reading up on the FFM production process flow (Flour Milling Production Flowchart, 2003) and the plant control operating manual (Holensein & Chischè, 2003).

At the beginning, the raw material, wheat is acquired from three different countries. They are: USA, Canada and Australia. Upon arrival of the wheat is kept in 6 silos each with a capacity of 2200 ton. The imported wheat from these different countries can be categorized into hard wheat and soft wheat.

When there is an order for a certain brand of flour, like Blue Key, Achor, Muhibbah and others, the production line will calculate the amount of different groups of raw wheat that is required. From there, the required raw wheat is stored into the Raw Wheat Bin. There are 4 of them, having a capacity of 440 ton. The raw wheat that is in the Raw Wheat Bin contains impurities. The information of the percentage of the impurities in the raw wheat can be obtained from the supplier of the wheat. Then, it moves to the Pre-Cleaning process of the raw wheat. From there, estimation is made of how much of the pre-cleaned wheat still remains after the pre-cleaning process.

The pre-cleaned wheat moves to the next process; cleaning and tempering of the raw wheat. In this process, the raw wheat is cleaned again and tempered by adding water to the raw wheat. The tempering process will increase the weight of the raw wheat normally the weight gain of the raw wheat is about 7% or 8% of its original weight. The purpose of tempering it is to increase the moisture of the raw wheat. Normally, the raw wheat would contain 9% or 10% of moisture, which is controlled by the supplier of the raw wheat. The process of tempering needs to be controlled so that the water added into the raw wheat for tempering will produce flour with maximum 14% moisture after milling the raw wheat. The

14% moisture of flour is the standard regulated by the Government of Malaysia. Therefore, in order to produce flour that contains 14% of moisture, the raw wheat needs to be tempered until it contains 15.8% of moisture. This is because the milling process results in moisture loss. There is a laboratory available in the factory for determining the percentage of moisture. The tempering process for soft and hard wheat takes a different period of time to complete. Normally, for tempering soft wheat, 9 hours is required to complete the process and for hard wheat, 12 hours is required to complete the process.

After the cleaning and tempering processes, the milling of the clean and tempered raw wheat is the next process. At FFM, the milling speed can be adjusted to three different levels. Normally, the milling process takes one hour to mill 9 ton of the raw wheat. As for the highest milling process, it can take one hour to mill 9.5 ton of raw wheat and the lowest milling process takes one hour to mill 7.5 ton of raw wheat. Once all the raw wheat is milled, it can produce the flour at 76% of the total weight of the raw wheat. After the milling process, there is quality control, a sample of the flour is sent to the laboratory for the purpose of quality control. Quality of the products like Blue Key, Achor, Muhibbah are determined by the laboratory based on certain formula for the mixing process of the flour. Mixing of the flour, which is milled from soft wheat and hard wheat as well as from different countries, can determine the class of the flour.

6.3 Define Scope and Bound Domain

As mentioned in Chapter 3, the first step of modeling is to identify the goal of the model to be developed. The purpose is to define the scope and the boundary of the model in the case study. This is to ensure only relevant and necessary things are included in the

model. The goal of this case study is “To Get Amount of Flour Produced and Duration Needed Based on the Total of Raw Wheat Input”. This goal is derived from the studies of the daily operations of the factory and discussions with Mr. Ling Jai Seng, the Plant Engineer (Ling, 2004).

After the goal is determined, a series of meetings were conducted with the Plant Engineer to gain understanding of the processes involved in producing the flour from the raw wheat. This allows for identification of processes and concepts which are necessary to be contained in the model in order to achieve the goal stated.

At this level, the model that is going to be developed will present Domain Process (DP) structure that is bounded by identifying what is to be included in the Domain Process (DP) structure thereby achieving what is to be done by this Domain Process (DP) structure. This then enables the set up of the boundary and scope for this Domain Process (DP) structure where irrelevant processes, activities and data can be excluded. For example, Cleaning Domain Process (DP) structure would include the essential processes, activities and creating data that are related to the Cleaning domain rather than milling or mixing process.

Basically, there are three major processes involved to produce flour from raw wheat to the type of flour that is demanded by the market. These processes are: cleaning and tempering of raw wheat, mill the processed wheat and to mix different grades of flours to produce the required end products likes flour with the Blue Key brand. However, this case study will only involve two of the processes lines, cleaning and tempering of the raw wheat and milling of the processed wheat. This is because mixing process involves confidential

information and benefits of different parties. From the analysis, two Domain Process Structures are created. There are Cleaning Domain Process and Milling Domain Process.

Domain Process Structure	Descriptions of Functionality
Cleaning	This domain basically involving all the processes relevant to the cleaning and tempering of the raw wheat those need to be milled.
Milling	This involves all the activities or functions and data needed to produce the flour from the produced wheat.

Table 6.1 Descriptions of the functionalities of the Domain Process (DP) structure involved

6.4 Identify and Specify Processes Structure

The identified Domain Process (DP) structures therefore lead to the identification of existing or additional sets of processes, data and activities which make up the responsibilities or characteristics of the relevant domain. These identified processes and activities of particular domain processes were then analyzed. This is because only relevant Process (P) structures or Activity (A) structures need to be created under the Domain Process (DP) structure and different Process (P) structures need different sets of data and activities. Grouping them together under specific domains will engender a more manageable, meaningful and organized set of processes. For example, it is decided to group Wheat_Weight Process (P) structure, Pre_Cleaning Process (P) structure and Clean_Temper Process (P) structure into the Cleaning Domain Process (DP) structure. The get_Impurity Activity (A) structure and cal_Weight Activity (A) structure are grouped into the Pre_Cleaninig Process (P) structure. Besides that, they also reflect in greater detail

regarding the responsibility and characteristic of the Domain Process (DP) structure with the scheme of adding some properties, relationship and constraints to it.

In this level, Process (P) structure is used to represent “what needs to be done” or “what can it do” in order to execute its responsibility and display the characteristic of its parent process. For example, three Process (P) structures are used to inform what processes need to be executed in order to perform the responsibility of Cleaning Domain Process (DP) structure. This also reflects that the Process (P) structure level is defined in a time ordered sequence in which one Process (P) structure needs to complete its task before the next Process Structure can be implemented. For example, Pre_Cleaning Process (P) structure can do its job only after getting information and waiting on the Wheat_Weight Process (P) structure to complete its mission.

Then, the plant engineer verified the drafted model. Comments from the plant engineer were then incorporated into the draft. Some of the Process (P) structures would need to be created along the process flow of the manufacturing to accommodate the need of the activities that appear along the line. For example, Wheat_Weight Process (P) structure was created to accommodate the activity of getting the weight of the wheat input. Activity (A) structure is not created at that level because it is at the middle level where only Process (P) structure should appear. The process of drafting the process flow and modification were cycled until a satisfaction point was reached by the plant engineer. Table 6.2 presents the final set of Process (P) structures identified.

Process (P) Structure	Descriptions of Functionality
WheatWeight	Process that contains the activity to obtain the weight of the wheat to be input
Pre_Cleaning	Process that contains the activities of obtaining information required before entering the Process Structure Clean_Temper
Clean_Temper	Process that contains the activities of cleaning and tempering the raw wheat.

Table 6.2: Descriptions of the functionalities of the Process (P) structure involved in Cleaning Domain Process (DP) structure.

6.5 Identify and Specify Activity Structure

The Activity (A) structure is defined for its related Process (P) structure or Domain Process (DP) structure to represent “how” its parent process to perform its task. For example, Pre_Cleaning Process (P) structure is decomposed into a set of interested Activity (A) structures, get_Weight and get_Impurity, to represent the next level of details which explain the functionality of that Process (P) structure. Moreover, in order to better explain the functionality of the Parent Structure of an Activity (A) structure, properties of the Activity (A) structure can be added and constraints can be imposed. For example, get_Impurity Activity (A) structure of Pre_Cleaning Process (P) structure can obtain its required data or information only after get_Weight Activity Structure completes its activity like obtaining the weight of raw wheat, which is similar to what is elaborated at Process (P) structure. From this scenario, two or more process structures are created at each different level act in a sequential manner. Relation between two process structures determines the workflow. Therefore, relation plays an important role in setting up the constraints and time frames in the sequential operation at different process levels.

Activity (A) structure can also be directly decomposed from Domain Process (DP) structure instead of Process (P) structure. This indicates that the process involved has a simple structure and direct operations. For example, Milling Domain Process (DP) structure is directly decomposed into a set of Activity (A) structures where this set of Activity (A) structures are the functionality of the process involved.

Activity (A) Structure	Descriptions of Functionality
get_Weight	Get the weight of the raw wheat to be processed.
get_Impurity	Get the impurity of the raw wheat that is supplied by the suppliers of the raw wheat.
cal_Weight	Calculate the weight of the raw wheat after removing the percentage of impurity from the raw wheat.
get_Initial	Get the initial moisture of the wheat supplied. Different supplied wheat has different percentage of moisture.
get_Target	Get the percentage of moisture of wheat based on the moisture required for the flour produced. Normally, the moisture of the wheat is 15.8%.
cal_Water_N	Calculate the water added for normal day operation which is water added for 100 ton of raw wheat.
cal_Water_T	Calculate the water added for total weight of the raw wheat input for process.
cal_Gain	Calculate the gain of the weight of the raw wheat after completion of tempering process. Normally the gain is 7% or 8% of the total weight of the raw wheat.
cal_ProdWheat	Calculate the total weight for the raw wheat plus the gain from tempering process for both 100 ton and the input of raw wheat.

Table 6.3: Descriptions of the functionalities of the Activity (A) structures involved in WheatWeight, Pre_Cleaning and Clean_Temper Process (P) structures.

Activity (A) Structure	Descriptions of Functionality
get_Millspeed	Get the speed of the milling process in order to determine the weight of the raw wheat can be milled per hour. There are three group of milling speed which are 9.5 ton per hour, 9 ton per hour and 7.5 ton per hour.
cal_FullTimeT	Calculate the number of milling times needed to mill all the raw wheat.
cal_RemWheat	Calculate the weight of the raw wheat that is not sufficient for the normal milling operation.
cal_TotalTime	Calculate the total time needed to complete milling all the raw wheat.
cal_FlourProd	Calculate the flour that can be produced from the raw wheat that is input.

Table 6.4: Descriptions of the functionalities of the Activity (A) structures involved in Milling Domain Process (DP) structure.

6.6 Identify and Classify Relation

In each different level, at least one Relation structure is associated with the process structure. Therefore, it is important to identify the Relation for each individual process structure at each level. This is because, associating a correct and meaningful Relation to a process structure is essential in developing a composite diagram which describes the network of relations for all process structures in the domain of study.

The information from the defined scope and bound domain form the major step in defining the Relation structure. This is because the defined scope and boundary form the clear direction and objective for the existence of the process structure and Relation structure. Moreover, only relevant process structures would present for Relation structure to be created to associate them. Therefore, information gathered from the meeting with the plant engineer is an important factor in deciding the existence of Relation structures at different levels. The existence of the Relation structures at different levels should have

been identified either explicitly or implicitly. For example, Relation structure that points at the left side of the Wheat_Weight Process (P) structure is identified explicitly because the Relation structure is predefined kind for indication of the starting Process (P) structure at that level. The Relation structure that exists between Wheat_Weight Process (P) structure and Pre_Cleaning Process (P) structure is implicitly defined because the interaction between these two Process (P) structures is extracted based on the study of the information gathered. For example, Wheat_Weight Process (P) structure is a virtual process which is created from the information gathered and judgment from the researcher and the plant engineer so that connecting Wheat_Weight Process (P) structure and Pre_Cleaning Process (P) structure can provide the description of the process flow model which can better reflect how things work within the study domain of the factory.

Besides that, Relation structures also provide a way to enhance the structure of processes by providing required data from outer source to them or delivering necessary data to relevant process structure. Therefore, integration information about processes and relations form a unified model. Figure 6.1 shows the whole picture of the relationship among the processes.

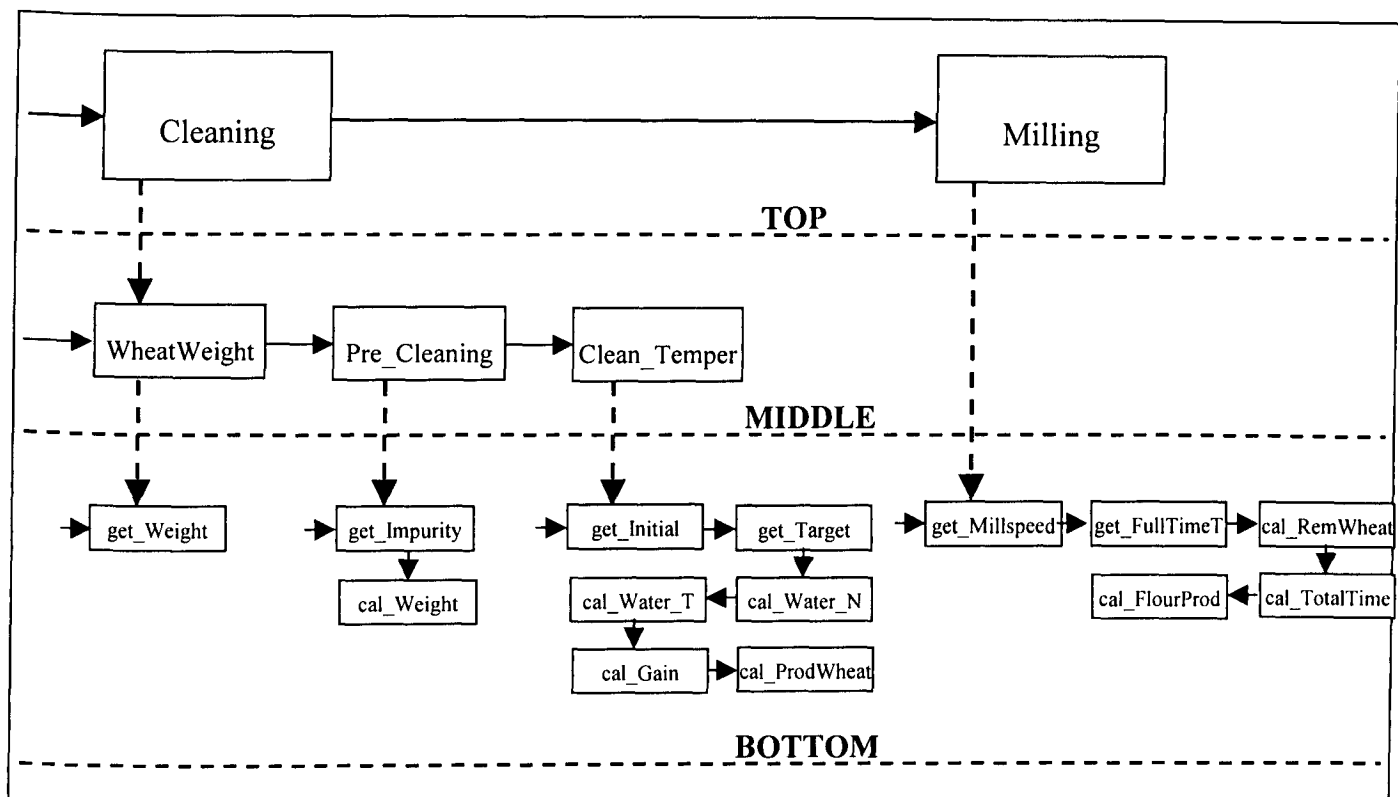


Figure 6.1: Relationship of the processes at different levels

6.7 Snapshots of the FFM Process Model

In this section, snapshots of the FFM process model are displayed. This is to show the model that is developed by using the Modeling Tool application.

Figure 6.2 shows the two Domain Process structures are created at the Level 1 with the objectives of the model to be built.

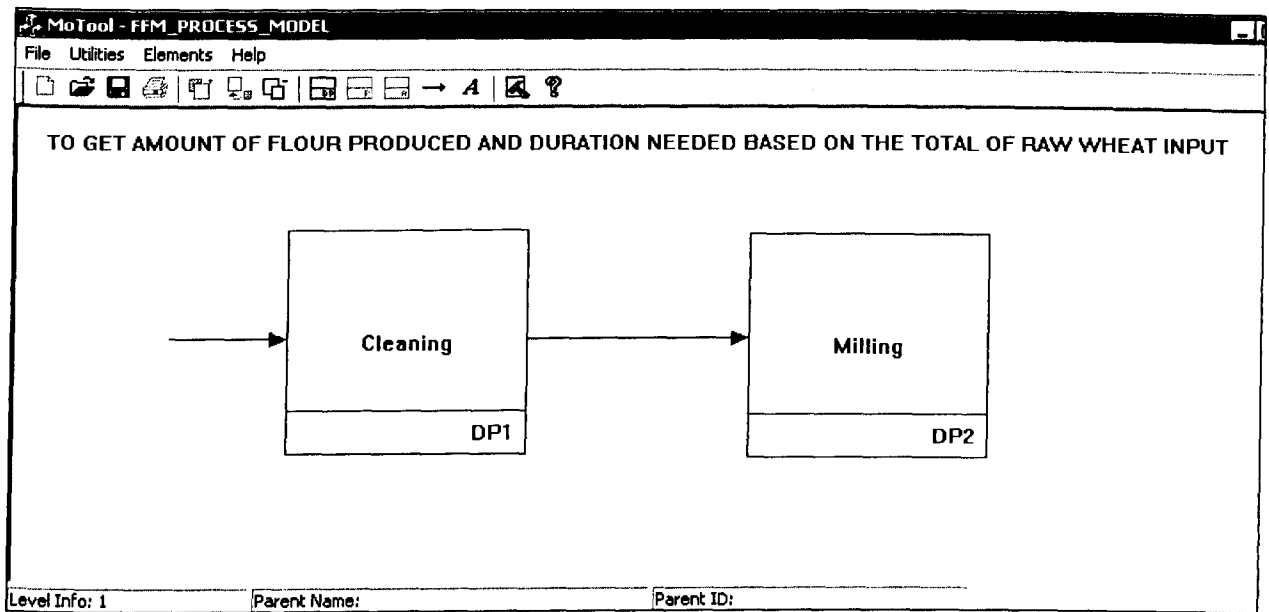


Figure 6.2: Domain Process (DP) structure with the objective of the modeling at Top Level.

Figure 6.3 shows the result of decomposition Cleaning Domain Process (DP) structure into next level of details. Three Process (P) structures are created at Level 2.

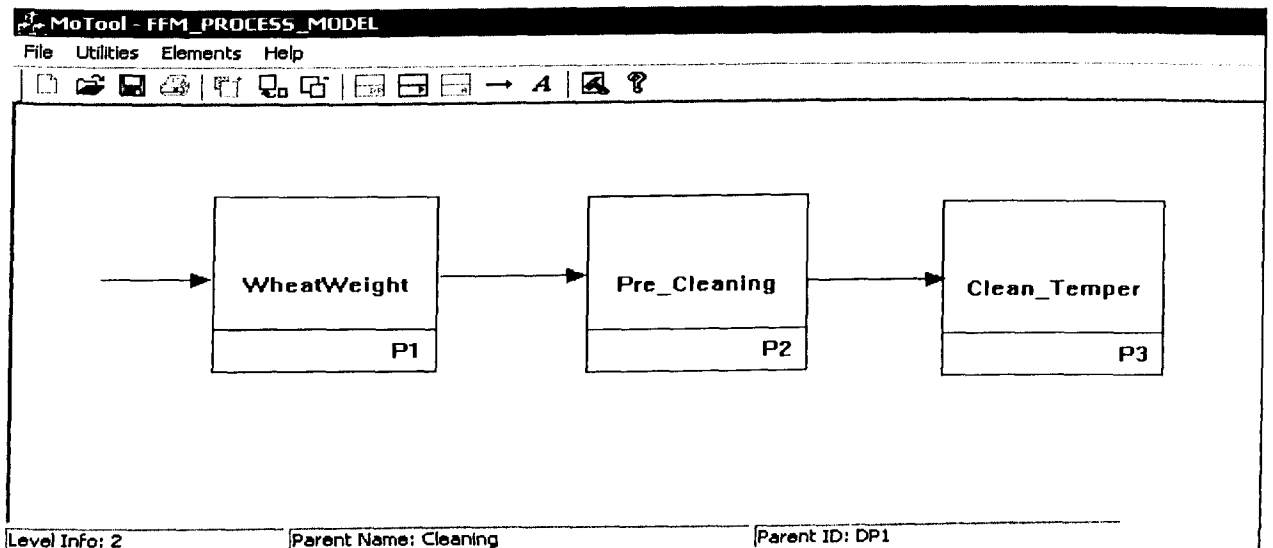


Figure 6.3: Process (P) structures created at Level 2 where their parent is Cleaning Domain Process (DP) structure

Figure 6.4 shows the result of decomposition from Level 2 to Level 3, where the parent process for this Activity (A) structure is WheatWeight Process (P) structure which can be seen from Figure 6.3. The Activity (A) structure is created with the information that is related to its parent structure.

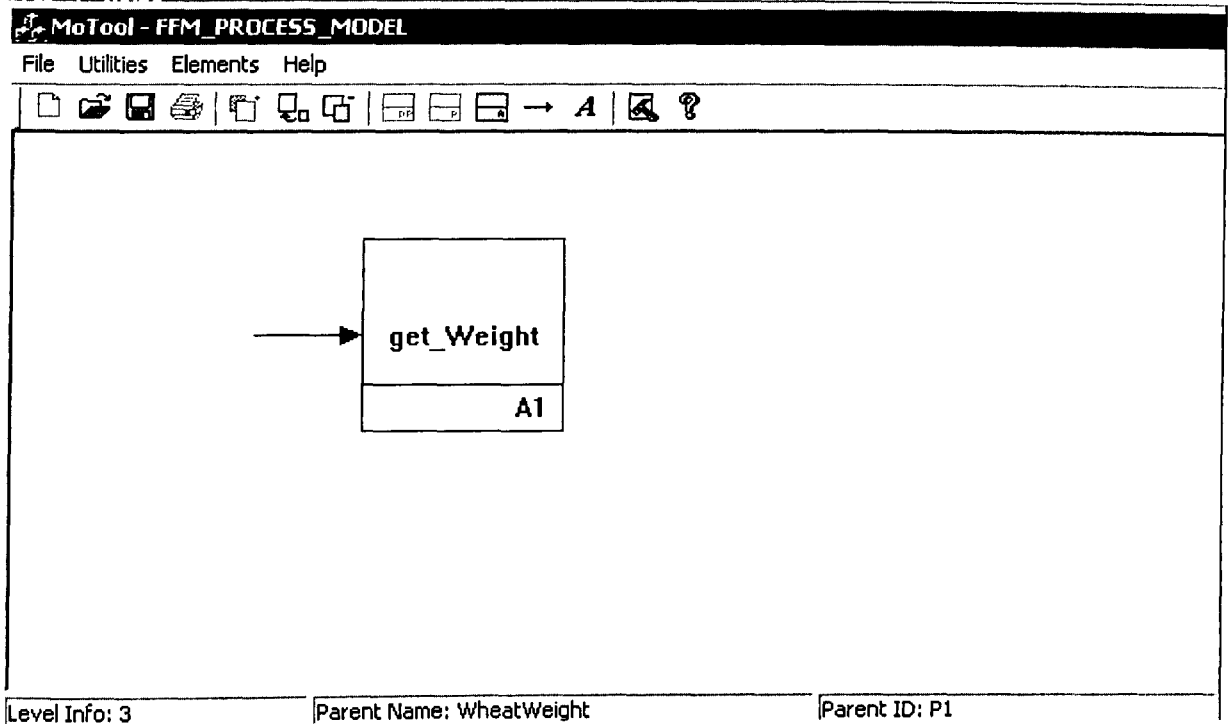


Figure 6.4: Activity (A) structure with its parent WheatWeight Process (P) structure

Figure 6.5 shows that two Activity (A) structures are created from the result of decomposition of the Process (P) structure, Pre_Cleaning. Different Process (P) structure can have its own group of Activity (A) structure to perform its task.

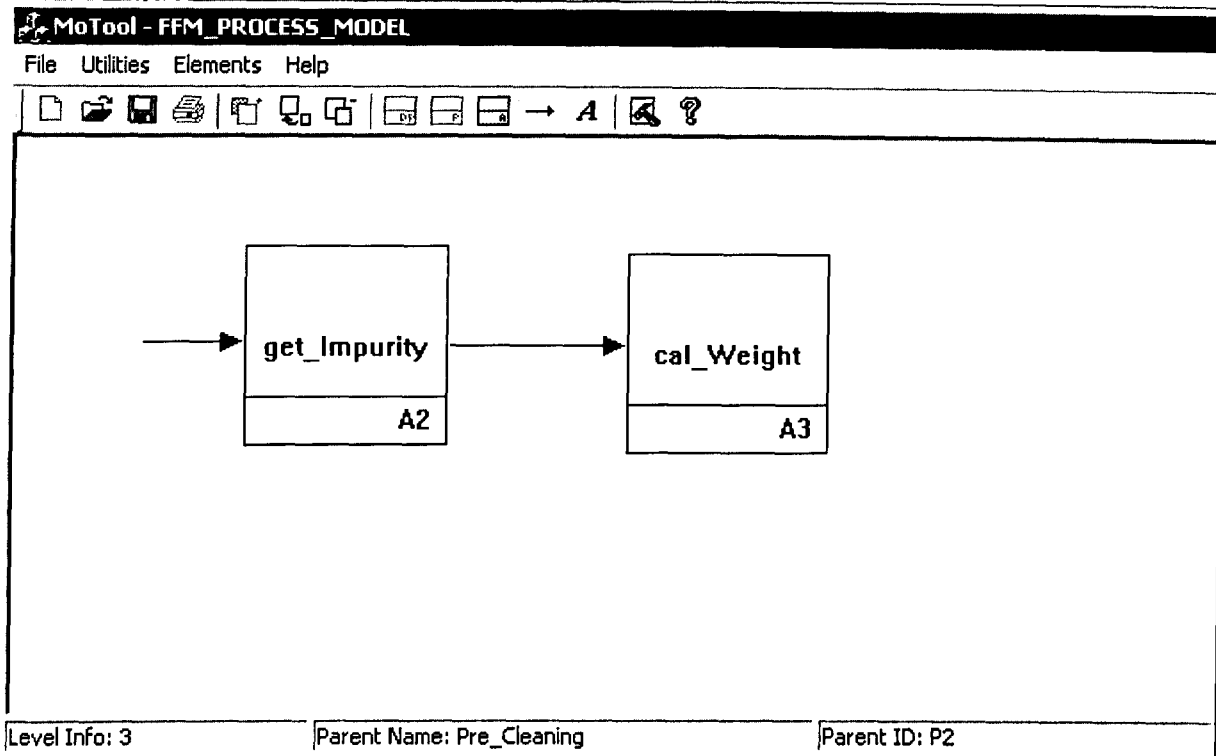


Figure 6.5: Activity (A) structure with its parent Pre_Cleaning Process (P) structure

Figure 6.6 shows that Clean_Temper Process (P) structure is decomposed into next level of details by creating six new Activity (A) structures to perform its task.

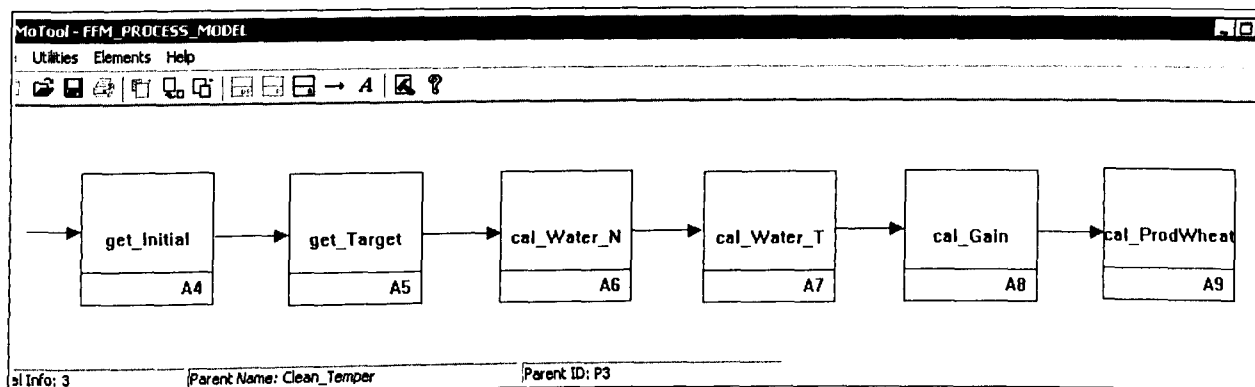


Figure 6.6: Activity (A) structure with its parent Clean_Temper Process (P) structure

Figure 6.7 shows the Activity (A) structures that are created at Level 2. This is a direct decomposition of Domain Process (DP) structure to Activity (A) structure. This informs that the Domain Process (DP) structure has no other components that are required to achieve that task assigned to it. Therefore, direct decomposition is required.

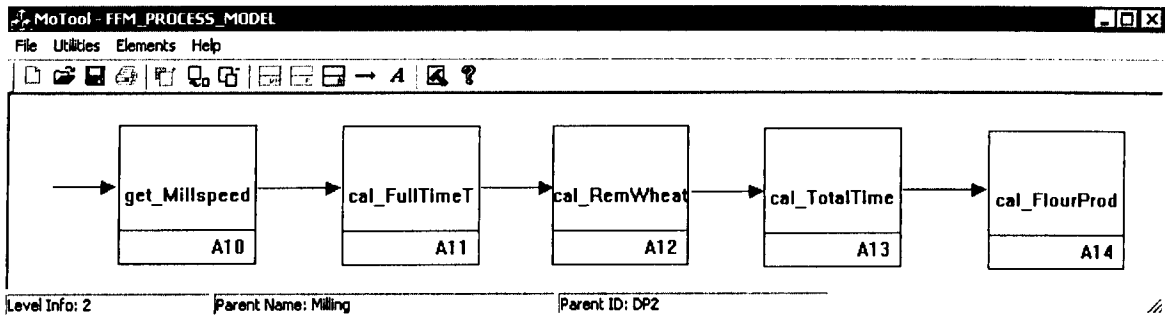


Figure 6.7: Activity (A) structure with its parent Milling Domain Process (DP) created at Level 2.

Figure 6.8 shows the message box displayed after completion of generating the coding templates by the system after modeler issues the command to the Code Generator.

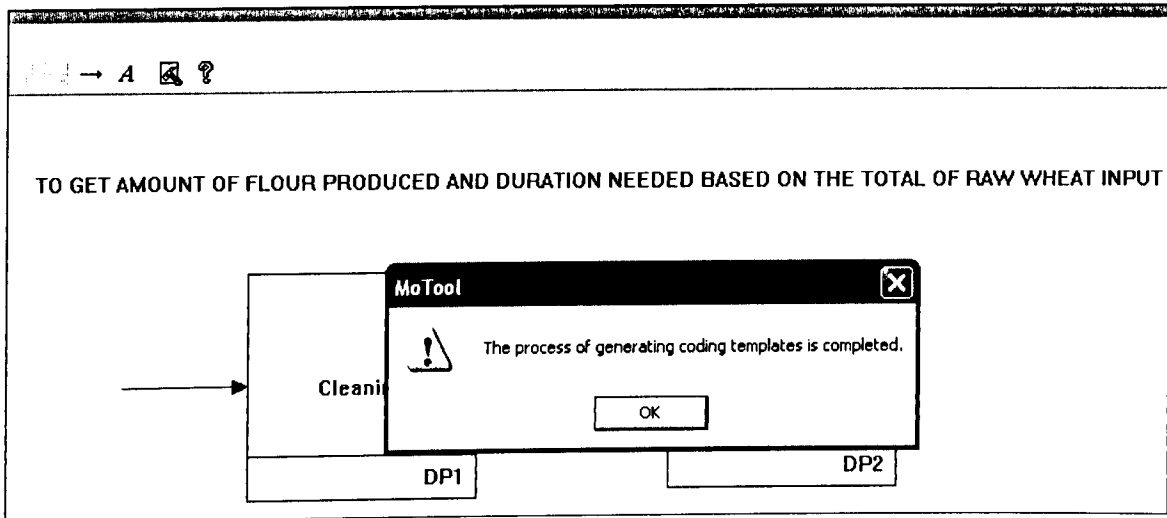
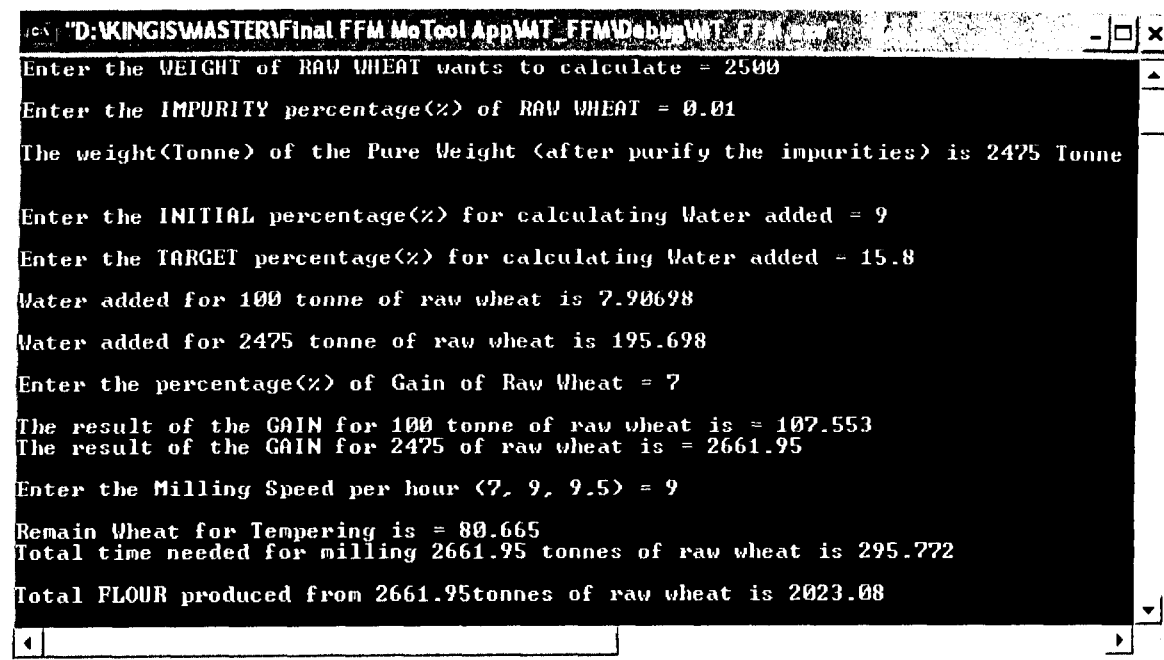


Figure 6.8: Indication of the completion of generation coding templates.

Figure 6.9 shows the application developed by customizing the coding templates that is generated by the Code Generator. The purpose of the application developed is to calculate the amount of flour that can be produced from the total raw material, wheat input and the time taken to complete milling all the input wheat.



```

D:\KINGISMASTER\Final FFM MoTool App\WT_FFM\Debug\WT_FFM.exe
Enter the WEIGHT of RAW WHEAT wants to calculate = 2500
Enter the IMPURITY percentage(%) of RAW WHEAT = 0.01
The weight(Tonne) of the Pure Weight (after purify the impurities) is 2475 Tonne

Enter the INITIAL percentage(%) for calculating Water added = 9
Enter the TARGET percentage(%) for calculating Water added = 15.8
Water added for 100 tonne of raw wheat is 7.90698
Water added for 2475 tonne of raw wheat is 195.698
Enter the percentage(%) of Gain of Raw Wheat = 7
The result of the GAIN for 100 tonne of raw wheat is = 107.553
The result of the GAIN for 2475 of raw wheat is = 2661.95
Enter the Milling Speed per hour (7, 9, 9.5) = 9
Remain Wheat for Tempering is = 80.665
Total time needed for milling 2661.95 tonnes of raw wheat is 295.772
Total FLOUR produced from 2661.95tonnes of raw wheat is 2023.08

```

Figure 6.9: Application developed from customization on coding templates generated.

Figure 6.10 shows the sample of source codes for class definition for the Cleaning Domain Process (DP) structure. These source codes are the result of customization on the coding templates generated from the model developed for the FFM.

```

class CCleaning
{
    public:
    //variables declaration
    CWheatWeight* m_pWheatWeight;
    CPre_Cleaning* m_pPreClean;
    CClean_Temper* m_pCleanTemper;

    int m_OrigWheatWeight;
    float m_PureWheatWeight;
    float m_GainN;
    float m_GainT;

    //functions declaration
    CCleaning();
    CCleaning(const CCleaning& xClean);
    virtual ~CCleaning();
    //CCleaning& operator=(const CCleaning& yClean);
    void DP1_Process ();
};

```

Figure 6.10: Class definition for Cleaning Domain Process (DP) structure.

Table 6.5 shows the results of comparison between calculation made by FFM and the results generated by the Application developed. This calculation is made with some predefined conditions. They are:

- Initial Impurity of raw wheat is 0.01%
- Initial water added for tampering is 9% of the total weight of raw material
- Target of water added for tampering is 15.8%
- Gain from the tampering process is 7%

No	Input Raw Material (ton)	Ton of Flour (FFM)	Ton of Flour (Application)	Production Times (FFM) - Hours	Production Times (Application) - Hours
1	2500	2023.12	2023.08	296	295.772
2	3000	2426.22	2427.7	355	354.927
3	3500	2832.52	2832.31	414	414.08
4	4000	3236.84	3236.93	473	473.235

Table 6.5: Results comparison between calculations made by FFM and Application.

6.8 Conclusion

Designing a model by using this modeling tool, would need users to define the goal or purpose of developing the model. This enables the coding templates to be generated with certain constraints and the application can be developed with the idea of the user's requirements. Thereby, defining the goal of developing the model also facilitates the process of developing the application. Besides that, this modeling tool is using the top down approach in developing the model. This can be seen from the process of model development, where in top level, Domain Process (DP) structure is used, in middle level Process (P) structure is used and in bottom level, Activity (A) structure is used. Greater details of its parent process are represented as the levels descend. This can be visualized where Domain Process (DP) structure is an object of an entity whereby this object is composed of its own group of components (Process (P) structures) that work together to perform the job function of the object. Each of these components has its own functionalities (Activity (A) structures) to implement in order to perform the purpose of the component.

The next chapter will conclude the work done on this research and outlines some recommendations for future expansion.

7.1 Introduction

Chapter 1 presents the background and justification for the research to be implemented. The concept of a generic model from MIICI was the initial inspiration to put the research on the track. From here, the research objectives were laid out and the research scope was defined.

Some of the related literature to this research is presented in Chapter 2. A review of process modeling is presented with a focus on the areas of process modeling perspectives and usefulness of process modeling. Then there is a brief discussion about architecture, framework and methodology. This is necessary since developing a model normally need to relate to them. Next, different modeling languages were presented and reviewed. Finally, comparisons were made among the reviewed modeling languages to find out their similarities and differences.

In Chapter 3, the scheme to achieve the objectives of the research was presented. In order to capture process flow in the studied domain (plant), the process flow structures of the plant need to be acquired. Therefore, Modeling Tool Structure and Relation Structure were figured out. Modeling Tool Structure was composed by Domain Process (DP) structure, Process (P) structure and Activity (A) structure, each of them has their own role to play in the different parts of the process flow of a plant. Then modeling activities were presented. It is like a methodology of using the modeling tool. Finally, the discussed scheme was initiated by applying use case driven object oriented analysis. This lay the foundation for

the Modeling Tool application. Some of the works of object oriented analysis are presented in Appendix A.

Chapter 4 presents the application of object oriented design process on the modeling tools' objects identified during object oriented analysis in Chapter 3. Then, the architecture of the Modeling Tool was presented. This architecture formed the foundation for the development of the Modeling Tool application. Some of the works of object oriented design are presented in Appendix B. In Chapter 5, the implementation of the Modeling Tool application was presented to tally the works presented in Chapter 3 and Chapter 4.

Chapter 6 presents a case study on a manufacturing industry. The studied manufacturing industry was FFM Flour Mills (SARAWAK) SDN BHD. This chapter presented how the modeling activities were applied to develop the model. During the process of developing the model, it also presents how the Modeling Tool Structure and Relation Structure were applied in capturing process flows.

7.2 Contributions

This research presents a modeling tool which can be useful in different Small to Medium Sized Manufacturing Industry in Sarawak. SME normally would not involve a lot of complex processes. The major activity happening in SME's is normally on the shop floor, which is inside the plant. Therefore, the Modeling Tool application presents a modeling scheme which can serve as a tool for enterprise engineering. This modeling scheme is using four different structures to capture different levels and areas of information from the processes of the manufacturing industry. They are: Domain Process (DP) structure, Process (P) structure, Activity (A) and Relation structure. With these modeling structures, process

structure and behavior can be simulated through customization of the coding templates that are generated from the information captured from processes. Besides that, developing a model with the specific intention and deliberate capture the process flow from the plant and designing it, will provide a clear picture of the study domain. Customization of the captured processes enhances the capability of the modeling tool to perform analysis task and decision making through the results of simulation. Furthermore, generating the coding templates for different models creates an environment where particular models can be filled to meet its own needs.

The model developed from this tool can also serve as a media of communication. This is because this tool is designed to capture process flow of a plant. Then discussion can be held for individuals from same functions, fields and disciplines.

Besides that, this tool presents a graphical user interface for users to use. This provides an instrument that users can more easily learn and use instead of using a formal method (adopted by MIICI) which is more difficult to understand and needs professionals in that area to teach.

7.3 Recommendation for future research

There are two areas recommended for future study. The first area is related to constructing a tool to capture market interflow. This is important since actions of a plant are in response to the demand of the market. Therefore, prediction and simulation of the market interflow can prepare the plant to meet the different waves of uncertainty.

The second area is related to developing a library with a standard protocol to keep the coded templates. This would facilitate the process of developing application whereby

users can retrieve the coded templates from the library and use them in the current situation.

BIBLIOGRAPHY

Bahrami, A. (1999). *Object-Oriented Systems Development: Using Unified Modeling Language*, Boston: McGraw-Hill International Editions.

Barnett, W. D., Presley, A.R. & Liles, D.H.. (1995). Object-Oriented Business Process Modeling For The Virtual Enterprise. In *The Fourth National Agility Conference*.

Bellorin, J. & Fishbourne, C. (1993). Object-oriented Analysis of a Flexible Batch Production System. *Computing & Control Engineering Journal*, 4(5):233-238.

Busby, J.S. & Williams, G.M. (1993). The Value and Limitations of Using Process Models to Describe the Manufacturing Organisation. *International Journal of Production Research*, 31(9):2179-2194

Bussler, C. J. & Jablonski, S. (1994). An approach to integrate workflow modeling and organization modeling in an enterprise. In *Proceedings of the 3rd IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

CIMOSA Association (1996). CIMOSA – A Primer on Key Concepts, Purpose and Business Value.

BIBLIOGRAPHY

Bahrami, A. (1999). *Object-Oriented Systems Development: Using Unified Modeling Language*, Boston: McGraw-Hill International Editions.

Barnett, W. D., Presley, A.R. & Liles, D.H.. (1995). Object-Oriented Business Process Modeling For The Virtual Enterprise. In *The Fourth National Agility Conference*.

Bellorin, J. & Fishbourne, C. (1993). Object-oriented Analysis of a Flexible Batch Production System. *Computing & Control Engineering Journal*, 4(5):233-238.

Busby, J.S. & Williams, G.M. (1993). The Value and Limitations of Using Process Models to Describe the Manufacturing Organisation. *International Journal of Production Research*, 31(9):2179-2194

Bussler, C. J. & Jablonski, S. (1994). An approach to integrate workflow modeling and organization modeling in an enterprise. In *Proceedings of the 3rd IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

CIMOSA Association (1996). CIMOSA – A Primer on Key Concepts, Purpose and Business Value.

Curtis, B., Kellner, M. & Over, J. (1992). Process Modeling. *Communication of the ACM*, 35(9):75-90.

Davenport, T. (1993). *Process Innovation: Reengineering Work through Information Technology*, Boston: Harvard Business School Press.

Dictionary.com (2005). Lexico Publishing Group, LLC.

<http://dictionary.reference.com>. (Date of Reference: March 2005).

Douglas, A.B. & Leon, F.M. (2002). A Structured Approach to Simulation Modeling of Manufacturing Systems. In *Proceedings of the 2002 Industrial Engineering Research Conference*.

Flour Milling Production Flowchart (2003). FFM Flour Mills (Sarawak) Sdn Bhd.

Frasier, J. (1994). Managing Change Through Enterprise Models in Applications and Innovations in Expert Systems II. In *SGES Publications*.

GERAM: Generalised Enterprise Reference Architecture and Methodology (version 1.6.3). IFIP-IFAC Task Force on Architectures for Enterprise Integration.

Goossenaerts, J. & Bjorner, D. (1994). Generic Models for Manufacturing Industry. Research Report, UNU/IIST Rept. No.32, UNU/IIST, Macau.

Holensein, L. & Chischè, J. (2003). Plant Control System Operating Manual.

Huckvale, T. & Ould, M. (1995). Process Modelling - Who, What, How: Role Activity Diagramming. In *Business Process Change: Reengineering Concepts, Methods, and Technologies*, (Grover, V. & W.J. Kettinger, W.J.; eds.).

INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0) (1993). Draft Federal Information Processing Standards Publication 183.

Jacobson, I., Ericsson, M. & Jacobson, A. (1995). The Object Advantage Business Process, Reengineering with Object Technology. Reading, MA: Addison-Wesley.

Kawalek, P. & Kueng, P. (1997). The Usefulness of Process Models: A Lifecycle Description of how Process Models are used in Modern Organizations. In *Proceedings of the Second CaiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*.

Kosanke, K. (1995). CIMOSA – Overview and Status. In *Computers in Industry* 27, pp. 101-109.

Kosanke, K. & Zelm, M. (1999). CIMOSA modelling processes. In *Computers in Industry* 40, pp. 141-153.

- Kueng, P., Bichler, P. & Kawalek, P. (1996). How to compose an Object-Oriented Business Process Model? In *Proceedings of the IFIP TC8, WG8.1/WG8.2 Working Conference*, pp. 26-28.
- Kueng, P. & Kawalek, P. (1997). Process Models: a help or a burden? In *Proceedings of the Americas Conference for Information Systems*, pp. 676-678.
- Lars, C. C., Brage, W. J., Midjo, N., Onarheim, J., Syvertsen, T.G. & Totland, T. (1995). Enterprise Modelling – Practices And Perspectives. In *Proceedings of ASME Ninth Engineering Database Symposium*.
- Lethbrigde, T.C. & Laganieri, R. (2001). Object-Oriented Software Engineering, Practical Software Development Using UML and Java, McGrawHill.
- Liles, D.H. & Presley, A.R. (1996). Enterprise Modeling Within An Enterprise Engineering Framework. In *Preceedings Of the 96 Winter Simulation Conference*.
- Lin, F.R., Yang, M.C. & Pai, Y.H. (2002). A generic Structure for Business Process Modeling. *Business Process Management Journal*, 8(1):19-41.

- Ling, J.S. (2004). Discussion on the Flour Production of FFM. In *Lot 505, Block 8, MTL D, Sejingkat Industrial Park, Jalan Bako, 93250 Kuching, Sarawak*.
- Malone, T. W., Crowston, K., Lee, J.T., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C., Bernstein, A., Herman, G., Klein, M. & O'Donnell, M. (1999). Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science* 45 (3):425-443.
- Mauil, R., Childe, S., Bennett, J., Weaver, A. & Smart (1995). A Report on Process Analysis Techniques. Manufacturing and Business Systems Group, University of Plymouth.
- Mayer, R.J., Cullinane, T.P., Knappenberger, W.B. & Wells, M.S. (1995). Information Integration For Concurrent Engineering (IICE) – IDEF3 Process Description Capture Method Report. College Station, TX, Knowledge Based Systems, Inc.
- McUmber, R. (2002). Introduction to Object-oriented Modeling with UML (Version 1.0).
- Ngwenyama, O.K. & Grant, D.A. (1994). Enterprise Modeling for CIM Information Systems Architecture: An Object Oriented Approach. *Computers and Industrial Engineering*, 26(2): 279-293.

- Ould, M. (1995). *Business Processes: Modeling and Analysis for Re-engineering and Improvement*, Chichester: John Wiley & Sons.
- Ovidiu, S.N. (2000). Advanced Object Oriented Concepts- Business Modelling: UML vs. IDEF. Griffith University, School of Computing and Information Technology.
- Plaia, A. & Carrie, A. (1995). Application and Assessment of IDEF3 –Process Flow Description Capture Method. *International Journal of Operations & Production Management*, 15 (1):63-73.
- Presley, A.R. (1997). A Representation Method to Support Enterprise Engineering. Doctoral dissertation, Department of Industrial Engineering, The University of Texas at Arlington, Arlington, TX.
- Presley, A.R., Huff, B.L. & Liles, D.H. (1993). A Comprehensive Enterprise Model for Small Manufacturers. In *Proceedings of the 2nd Industrial Engineering Research Conference*, pp.430-434.
- Schildt, H. (1998). *MFC Programming From the Ground Up*, 2nd Ed. Osborne/McGraw-Hill.

- Senge, P.M. (1993). *The fifth Discipline: The Art & Practice of The Learning Organization*. Century Business, London.
- SMiDEC (1996). Small and Medium Industries Development Corporation.
<http://www.smidec.gov.my>. (Date of Reference: June 2005)
- Waltman, W. D. & Presley, A.R. (1993). Reading & Critiquing An IDEF0 Model. Enterprise Integration Frameworks Group. Automation & Robotics Research Institute, Texas.
- Whitman, L., Huff, B. & Presley, A.R. (1998). Issues Encountered Between Model Views. In *Proceedings Flexible Automation and Intelligent Manufacturing Conference*.
- Whitman L., Liles, D.H., Huff, B.L. & Rogers, K. J. (2001). A Manufacturing Reference Model For The Enterprise Engineer. *The Journal Of Engineering Valuation and Cost Analysis: Special Issue On Enterprise Engineering*, 4(1):15-36.
- Whitman, L., Ramachandran, K. & Ketkar, V. (2001). A Taxonomy Of Living Model Of The Enterprise. In *Proceedings of the 2001 Winter Simulation Conference*.

Zelm, M., Vernadat, F.B. & Kosanke, K. (1995). The CIMOSA business modeling process. In *Computers in Industry* 27, pp. 123-142.

APPENDIX A: SEQUENCE DIAGRAMS USED FOR USE CASES DEFINED

Diagram A1 to Diagram A10 show the sequence diagrams for different use cases defined at Chapter 3, Section 3.6.2. The diagrams are self explanatory for communication between objects participating in the interaction

Diagram A1: Sequence Diagram of Edit Model Use Case

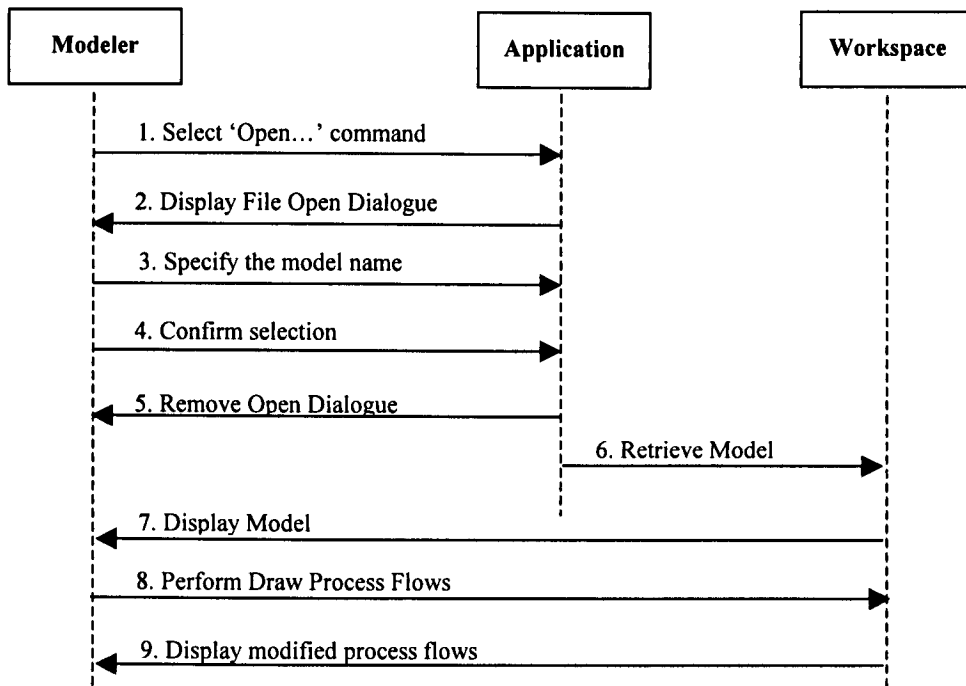


Diagram A2: Sequence Diagram of Generate Coding Templates Use Case

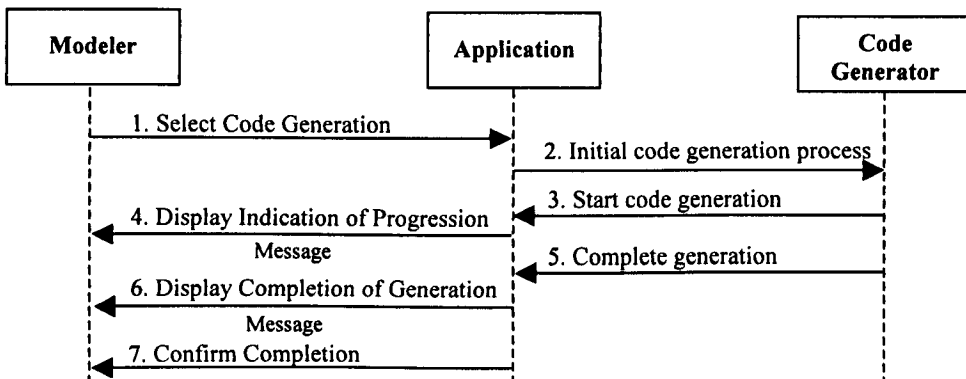


Diagram A3: Sequence Diagram of Finish Modeling Activities

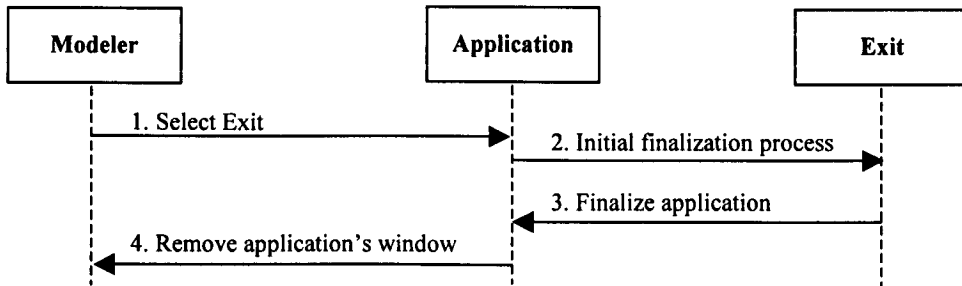


Diagram A4: Sequence Diagram of Save Operation Use Case

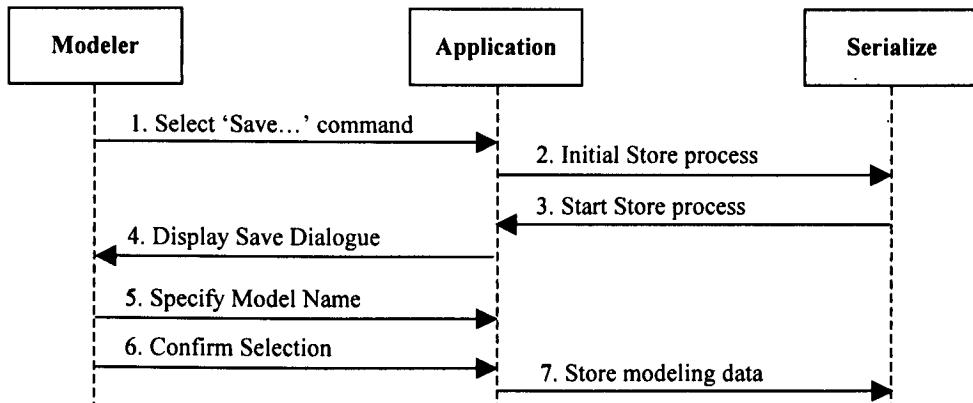


Diagram A5: Sequence Diagram of Save As Operation Use Case

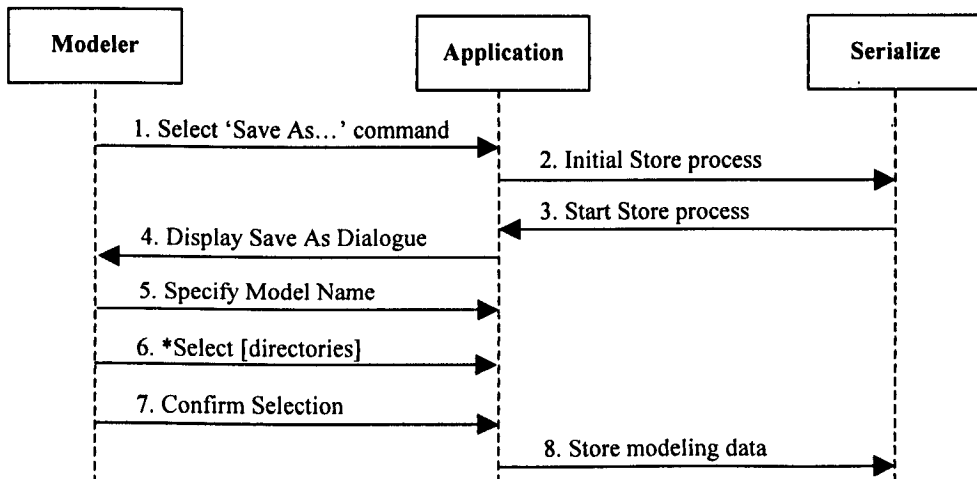


Diagram A6: Sequence Diagram of Draw Top Level Use Case

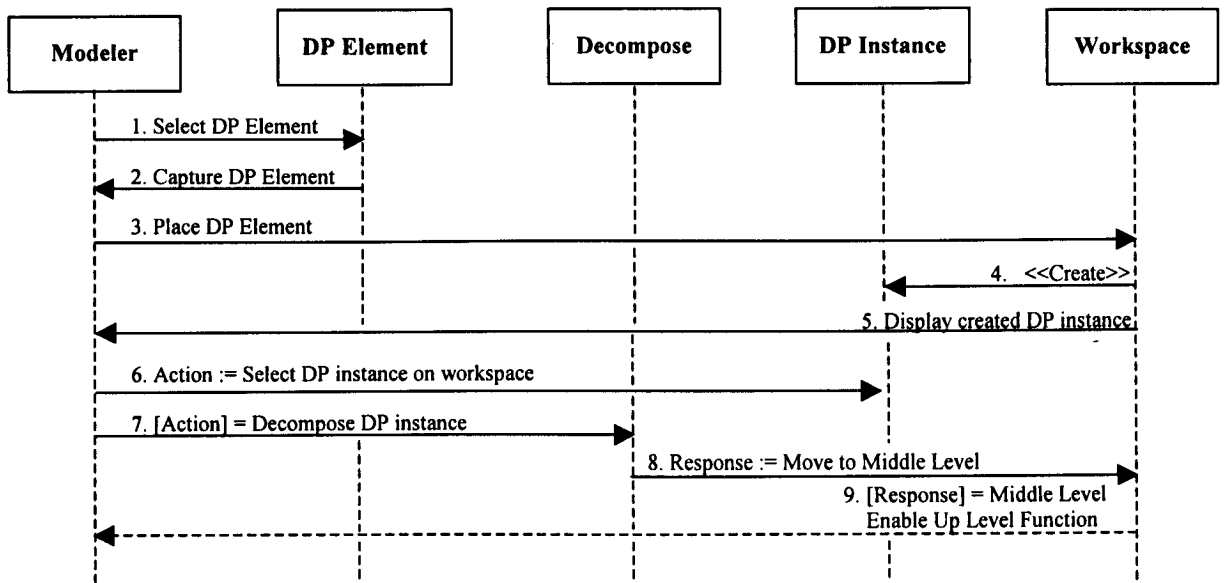


Diagram A7: Sequence Diagram of Draw Middle Level Use Case

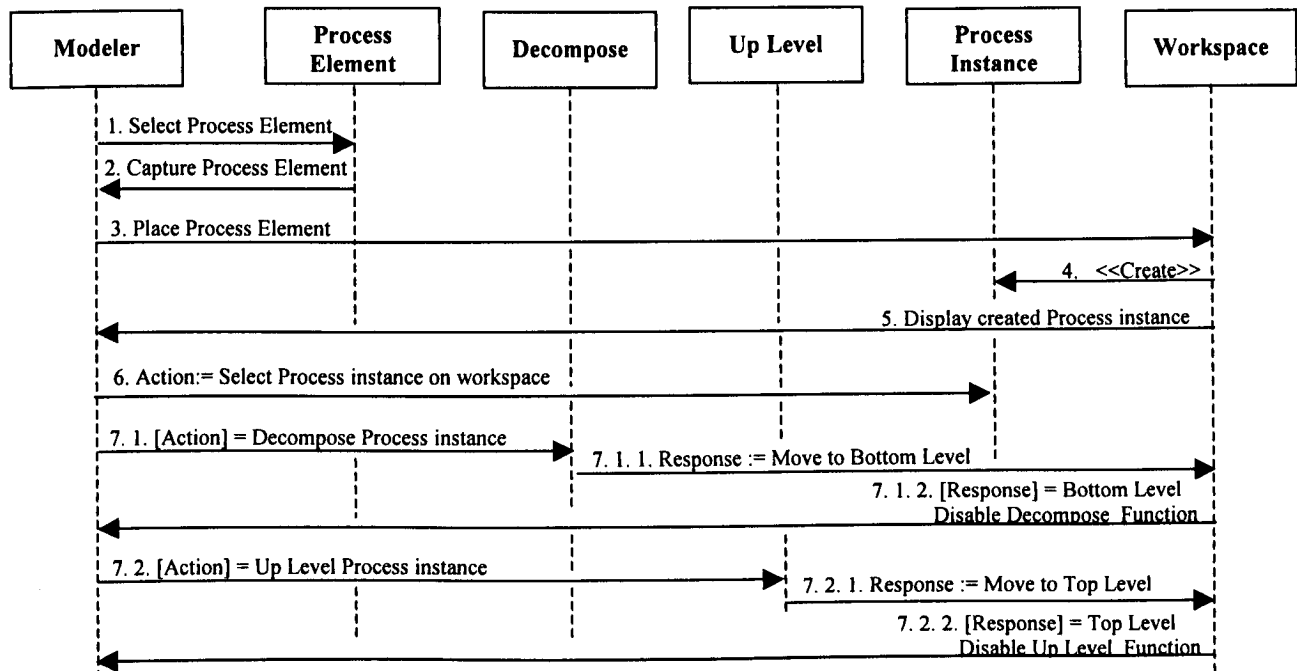


Diagram A8: Sequence Diagram of Add Attributes Use Case

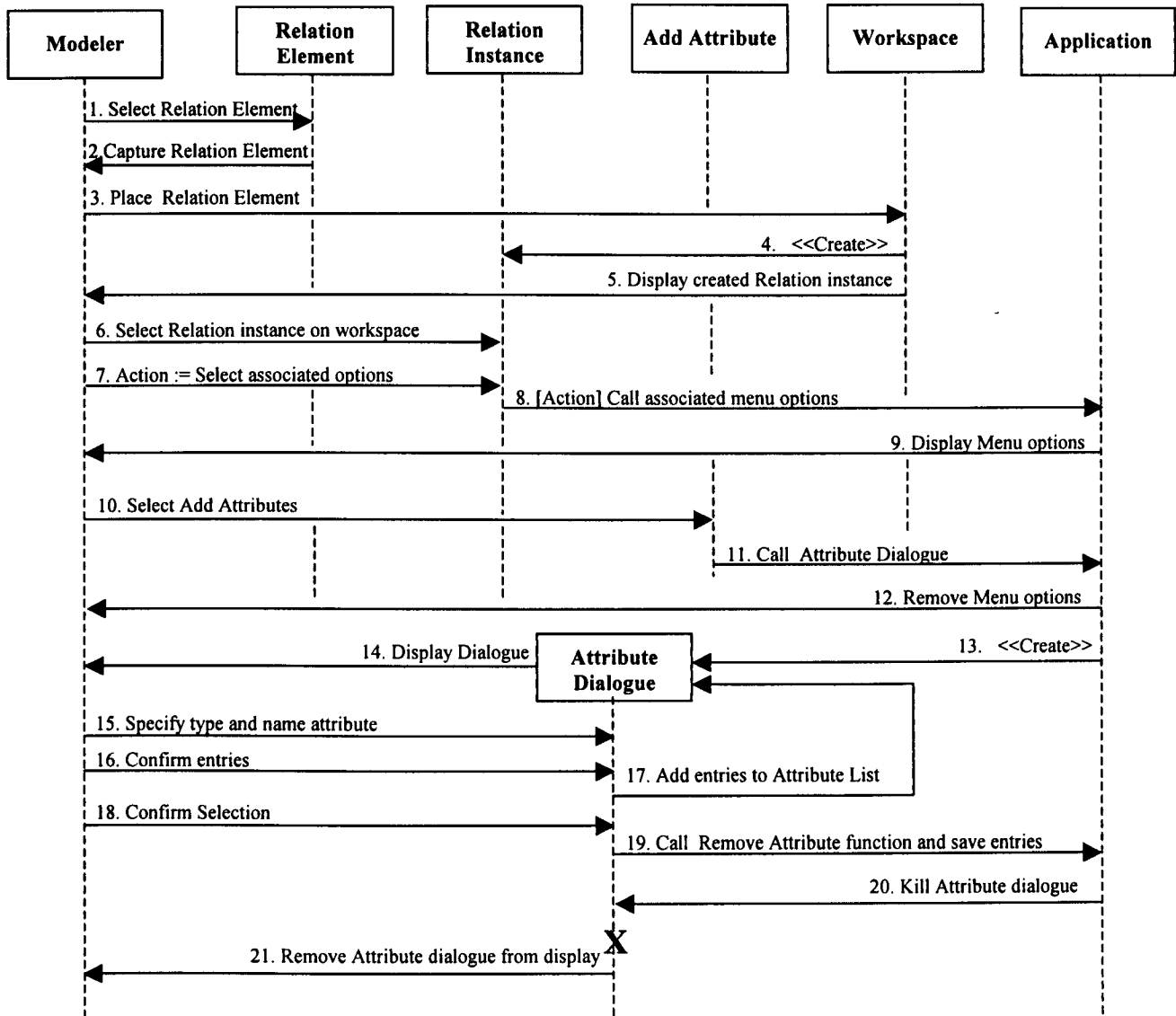


Diagram A9: Sequence Diagram of Remove Attributes Use Case

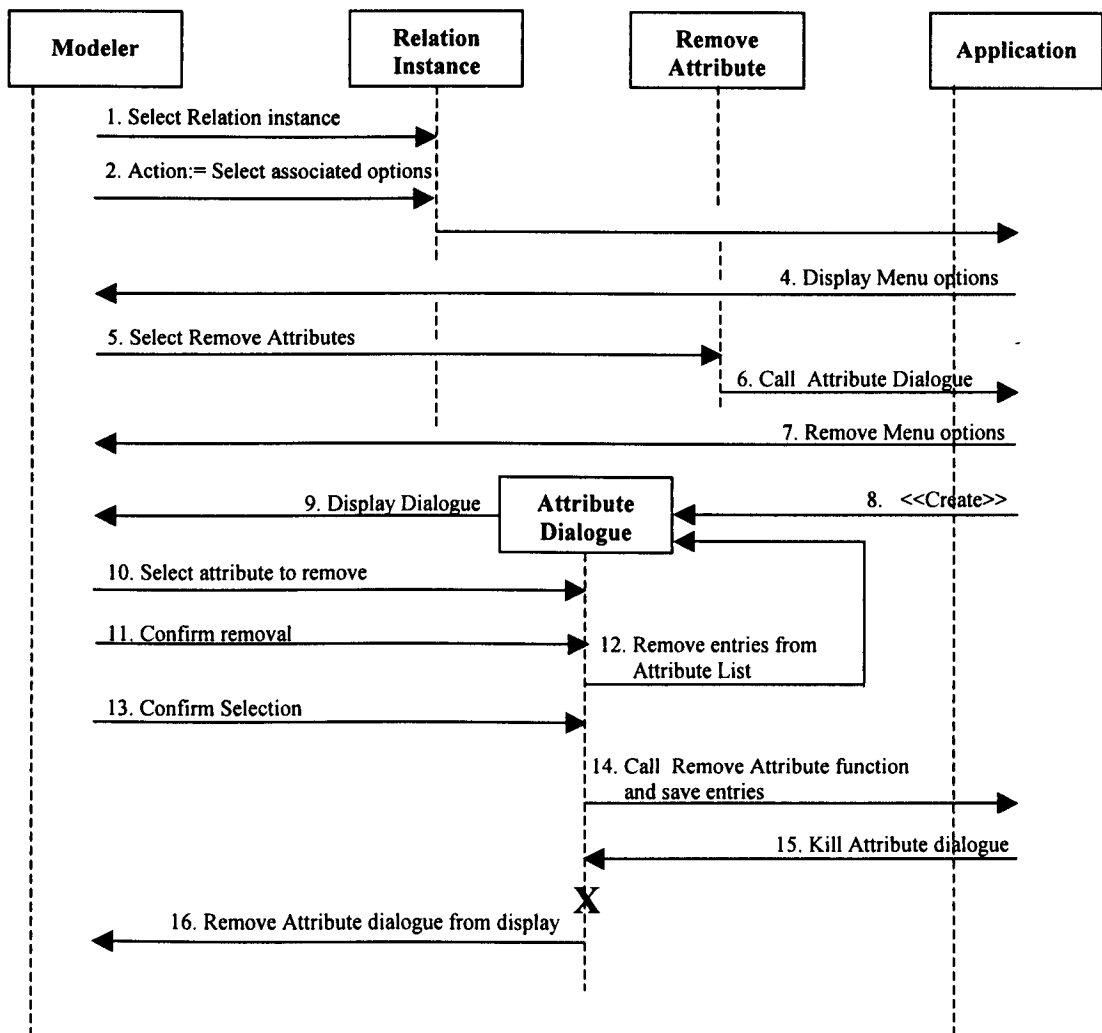
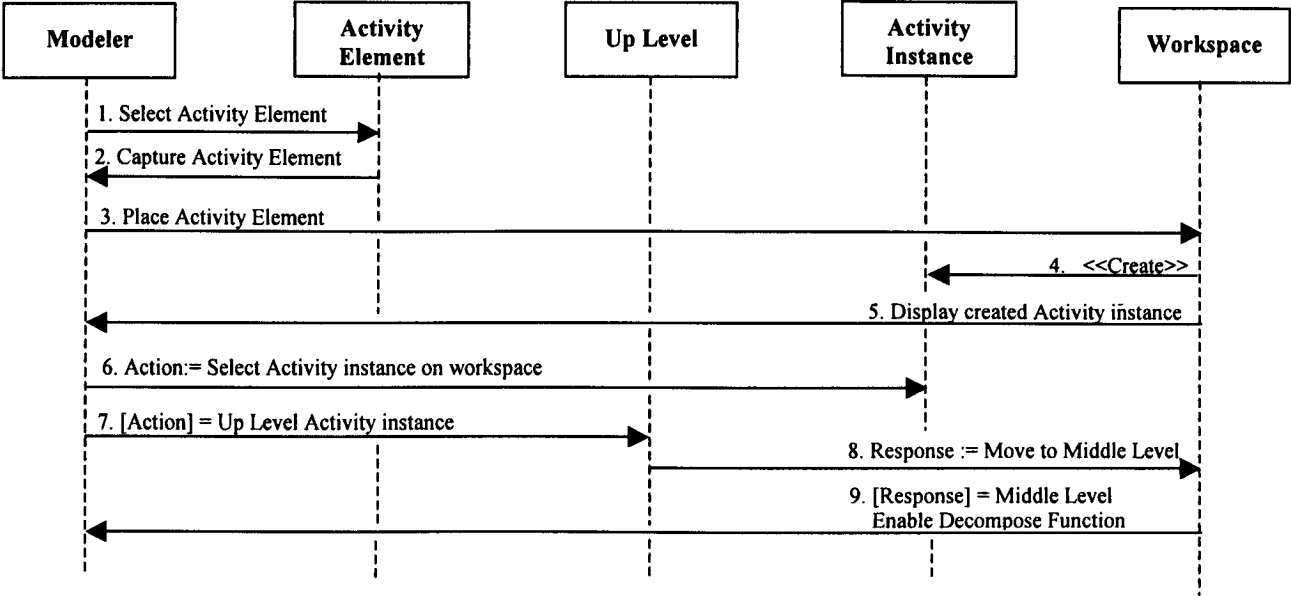


Diagram A10: Sequence Diagram of Draw Bottom Level Use Case



APPENDIX B: ACTIVITY DIAGRAMS FOR METHODS IDENTIFIED

Diagram B1 to B24 show the activity diagrams created for different methods identified during the process of object oriented analysis and refinement on the class diagram during the process of object oriented design.

Diagram B1: Activity Diagram for CMTElement class Draw method with one parameter

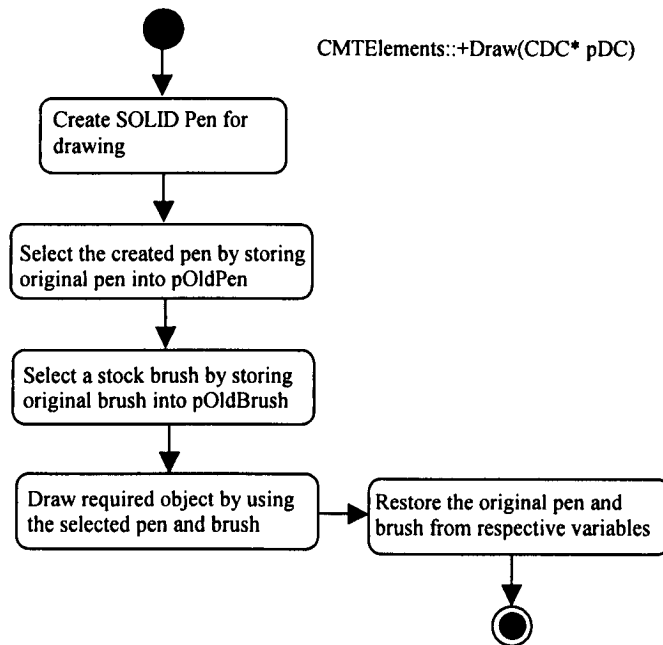


Diagram B2: Activity Diagram for CMTElements class GetElemID method

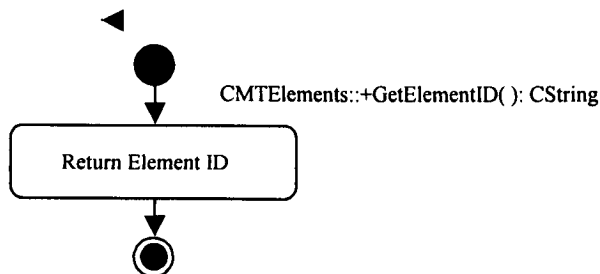


Diagram B3: Activity Diagram for CMTElements class GetElemType method

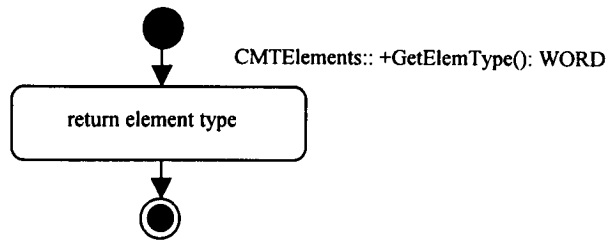


Diagram B4: Activity Diagram for CMTElements class Draw method with two parameters

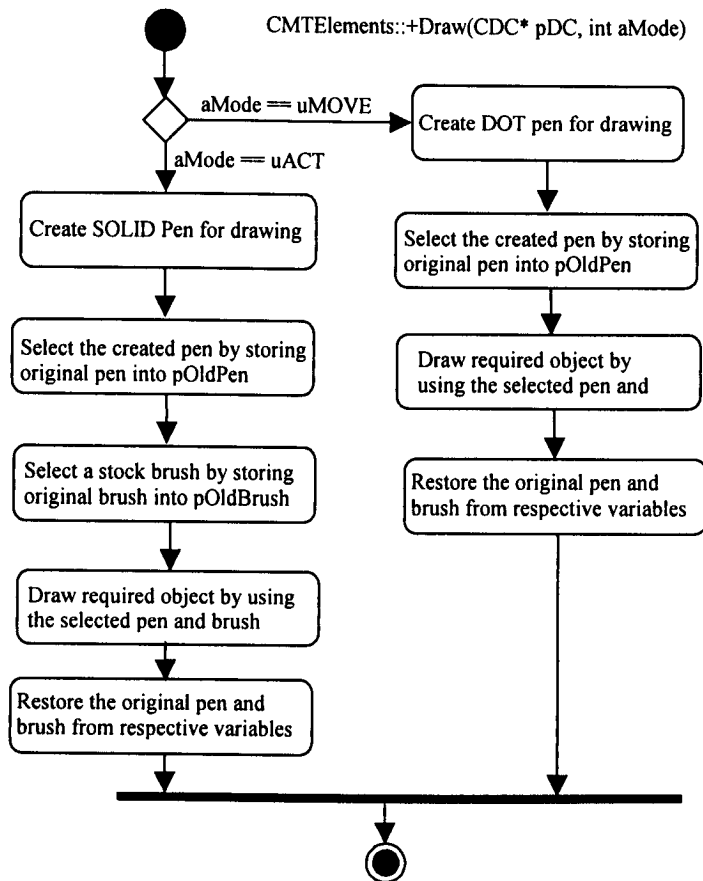


Diagram B5: Activity Diagram for CMTElements class GetPiD method

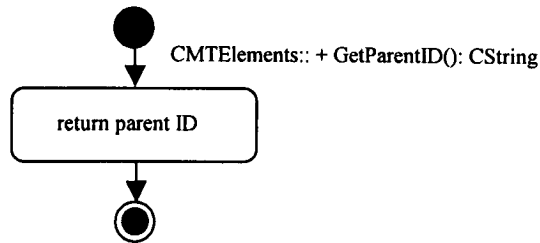


Diagram B6: Activity Diagram for CMTElements class GetsysLevel method

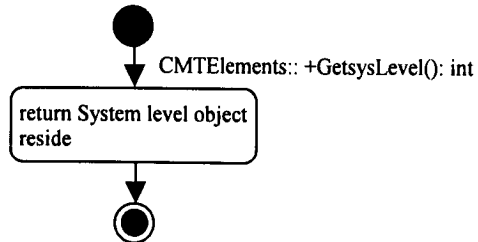


Diagram B7: Activity Diagram for CMTElements class Move method

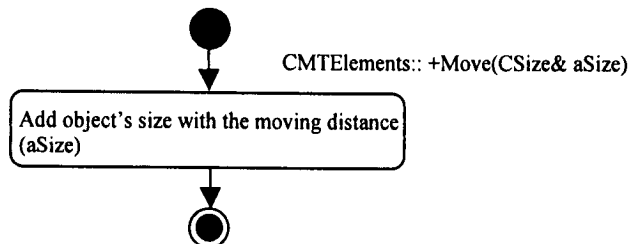


Diagram B8: Activity Diagram for CDProcess class, CProcess class and CActivity class GetProcName method

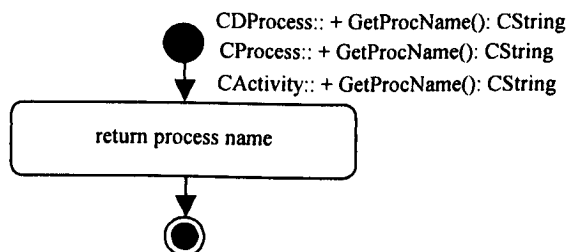


Diagram B9: Activity Diagram for CDProcess class SetProcName method

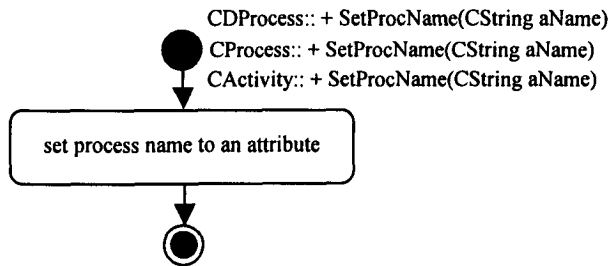


Diagram B10: Activity Diagram for CRelation class DrawArrow method

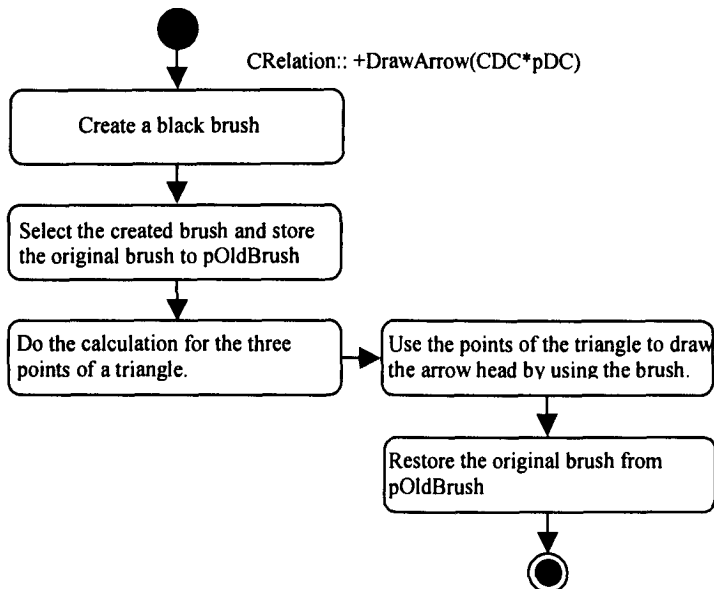


Diagram B11: Activity Diagram for CRelation class GetStartCon method

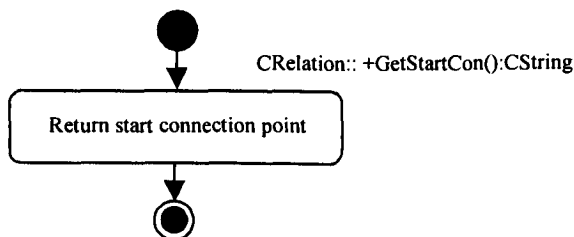


Diagram B12: Activity Diagram for CRelation class GetEndCon method

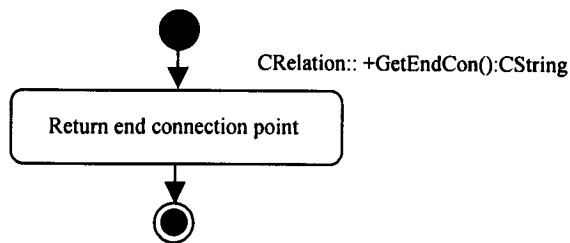


Diagram B13: Activity Diagram for CRelation class LineStart method

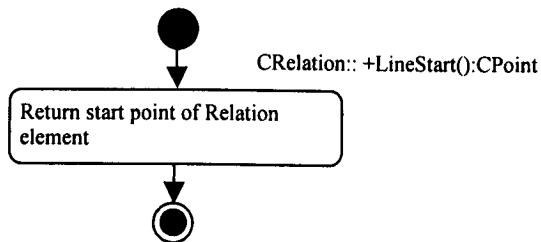


Diagram B14: Activity Diagram for CRelation class LineEnd method

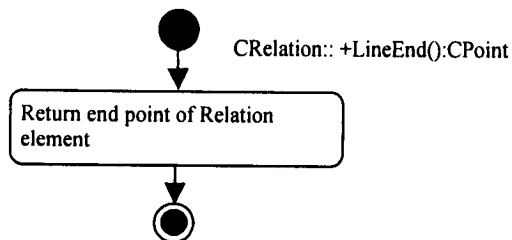


Diagram B15: Activity Diagram for CRelation class ResetConInfo method

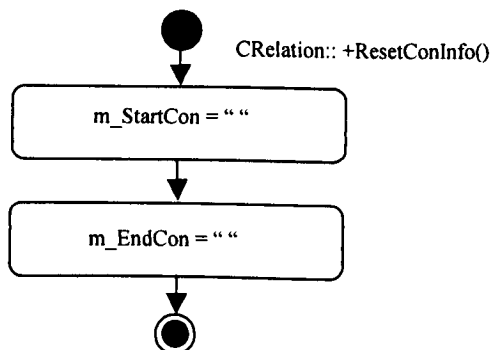


Diagram B16: Activity Diagram for CRelation class Resize method

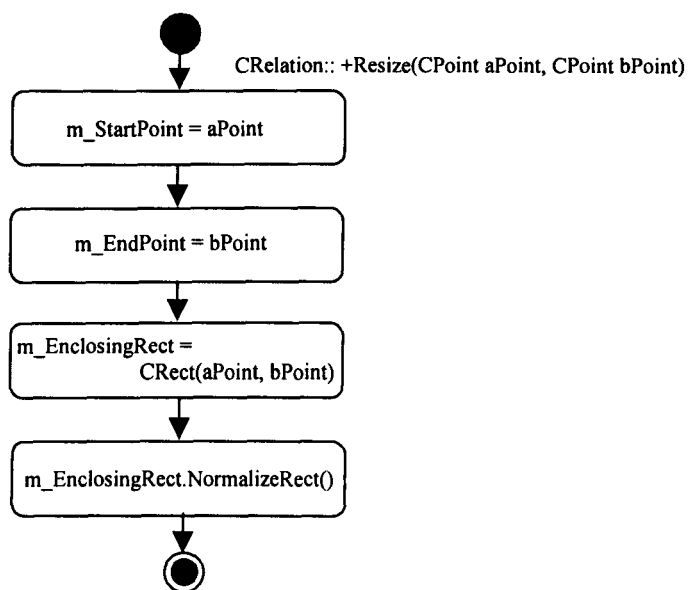


Diagram B17: Activity Diagram for CRelation class SetStartCon method

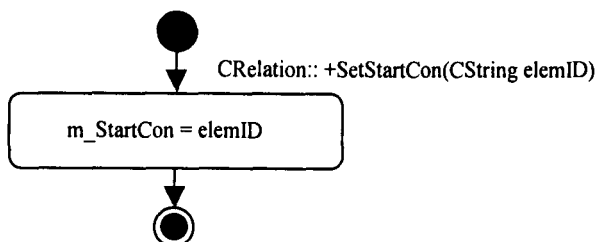


Diagram B18: Activity Diagram for CRelation class SetEndCon method

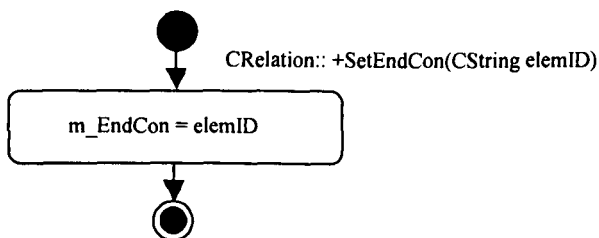


Diagram B19: Activity Diagram for CAttribute class GetAttribType method

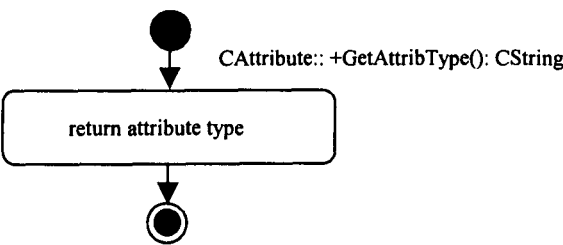


Diagram B20: Activity Diagram for CAttribute class GetAttribName method

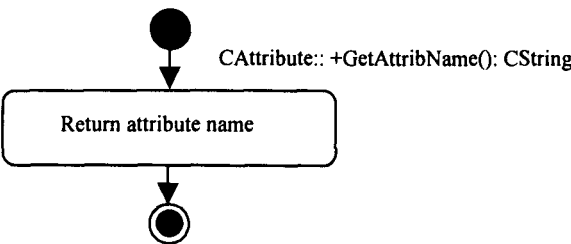


Diagram B21: Activity Diagram for CAttribute class GetProcID method

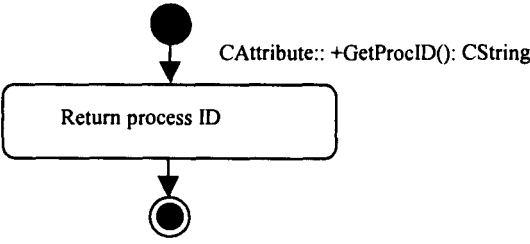


Diagram B22: Activity Diagram for CAttribute class GetProcName method

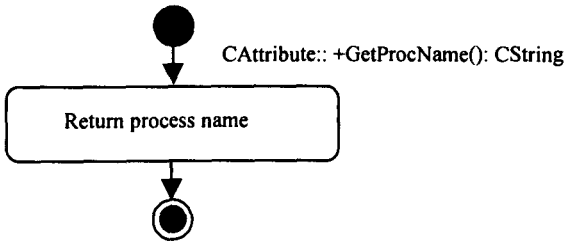


Diagram B23: Activity Diagram for CMTPath class SetMTPath method

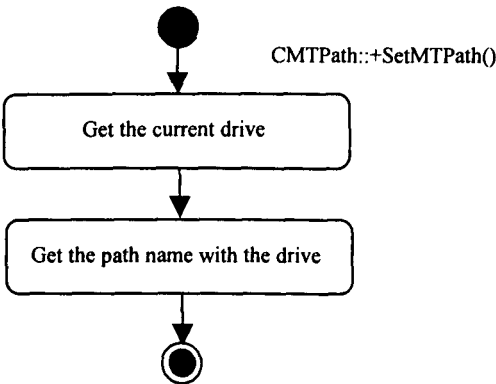
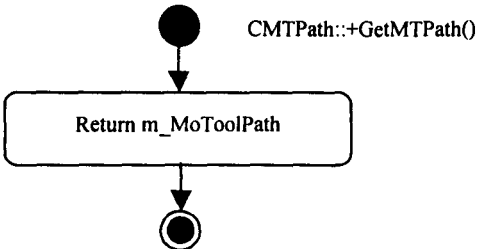


Diagram B24: Activity Diagram for CMTPath class GetMTPath method



APPENDIX C: Source Codes for the Application Developed for FFM to calculate total of flour and time taken from the input raw material

Figure C1 to Figure C9 present the source codes of the Application that is developed for FFM. The application is result of customization of the coding templates that are generated from the model built.

```
class CMilling
{
    public:
        //functions declaration
        CMilling();
        ~CMilling();
        void get_Cleaning(CCleaning *nCleaning);

        void get_MillSpeed ();
        void cal_FullTimeT ();
        void cal_RemWheat ();
        void cal_TotalTime ();
        void cal_FlourProd ();

    private:
        //variables declaration
        double m_MillSpeed;
        double m_TimeN;
        double m_TimeF;
        double m_TimeR;
        double m_RemWheat;

        CCleaning* m_pCleaning;
};
```

Figure C1: Class Definition for Milling Domain Process (DP) structure.

```

class CWheatWeight
{
    public:
        //variables declaration

        //function declaration
        CWheatWeight(CCleaning* pCleaning);
        virtual ~CWheatWeight();

        void get_Weight ();

    private:
        //variables declaration
        CCleaning* m_pCleaning;

        //constructor declaration
        CWheatWeight();
}

```

Figure C2: Class definition for WheatWeight Process(P) structure.

```

class CPre_Cleaning
{
    public:
        //variables declaration
        float m_Impurity;

        //function declaration
        CPre_Cleaning(CCleaning* pCleaning);
        virtual ~CPre_Cleaning();

        void get_Impurity ();
        void cal_Weight ();

    private:
        //variables declaration
        CCleaning* m_pCleaning;

        //constructor declaration
        CPre_Cleaning();
};

```

Figure C3: Class definition for Pre_Cleaning Process (P) structure.

```

class CClean_Temper
{
    public:
    //functions declaration
    CClean_Temper(CCleaning* pCleaning);
    ~CClean_Temper();

    void get_Initial ();
    void get_Target ();
    void cal_Water_N ();
    void cal_Water_T ();
    void cal_Gain ();
    void cal_ProdWheat ();

    private:
    //variable declaration
    CCleaning* m_pCleaning;

    float m_Initial;
    float m_Target;
    float m_Gain;
    float m_WeightWaterN;
    float m_WeightWaterT;
    float m_WaterN;
    float m_WaterT;
    //constructor declaration
    CClean_Temper();
};

```

Figure C4: Class definition for Clean_Temper Process (P) structure.

```

CCleaning::CCleaning()
{
    m_pWheatWeight = new CWheatWeight(this);
    m_pPreClean = new CPre_Cleaning(this);
    m_pCleanTemper = new CClean_Temper(this);
}

CCleaning::~~CCleaning()
{
    delete m_pWheatWeight;
    delete m_pPreClean;
    delete m_pCleanTemper;
}

CCleaning::CCleaning(const CCleaning& xClean)
{
    CCleaning();
    m_OrigWheatWeight = xClean.m_OrigWheatWeight;
    m_PureWheatWeight = xClean.m_PureWheatWeight;
    m_GainN = xClean.m_GainN;
    m_GainT = xClean.m_GainT;
}

void CCleaning :: DP1_Process ()
{
    m_pWheatWeight->get_Weight();
}

```

Figure C5: Class methods declaration for Cleaning Domain Process (DP) structure.

```

CWheatWeight::CWheatWeight(CCleaning* pCleaning)
{
    m_pCleaning = pCleaning;
}

void CWheatWeight::get_Weight ()
{
    cout << "Enter the WEIGHT of RAW WHEAT wants to calculate = ";
    cin >> m_pCleaning->m_OrigWheatWeight;
    cout << endl;
    m_pCleaning->m_pPreClean->get_Impurity();
}

```

Figure C6: Class methods declaration for WheatWeight Process (P) structure with its Activity (A) structure, get_Weight.

```

CPre_Cleaning::CPre_Cleaning(CCleaning* pCleaning)
{
    m_pCleaning = pCleaning;
}

CPre_Cleaning::CPre_Cleaning ()
{
}

CPre_Cleaning::~CPre_Cleaning ()
{
}

void CPre_Cleaning::get_Impurity ()
{
    cout << "Enter the IMPURITY percentage(%) of RAW WHEAT = ";
    cin >> m_Impurity;
    cout << endl;

    cal_Weight ();
}

void CPre_Cleaning::cal_Weight ()
{
    float nImpureWeight;

    nImpureWeight = m_pCleaning->m_OrigWheatWeight * m_Impurity;
    m_pCleaning->m_PureWheatWeight = m_pCleaning->m_OrigWheatWeight - nImpureWeight;
    cout<< "The weight(Tonne) of the Pure Weight (after purify the impurities) is " << m_pCleaning-
>m_PureWheatWeight << " Tonne";
    cout<< endl << endl;
    m_pCleaning->m_pCleanTemper->get_Initial();
}

```

Figure C7: Class methods declaration for Pre_Cleaning Process (P) structure with its Activity (A) structures.


```

CClean_Temper::CClean_Temper(CCleaning* pCleaning)
{
    m_pCleaning = pCleaning;
}
void CClean_Temper::get_Initial ()
{
    cout<<"Enter the INITIAL percentage(%) for calculating Water added = ";
    cin >> m_Initial;
    cout << endl;
    get_Target ( );
}
void CClean_Temper::get_Target ()
{
    cout << "Enter the TARGET percentage(%) for calculating Water added = ";
    cin >> m_Target;
    cout << endl;
    cal_Water_N ( );
}
void CClean_Temper::cal_Water_N ()
{
    float iTarget, iCapacity;
    iTarget = 14;
    iCapacity = 100;
    m_WaterN = ((m_Target - m_Initial)/(100 - iTarget)) * iCapacity;
    cout << "Water added for 100 tonne of raw wheat is " << m_WaterN << endl << endl;
    cal_Water_T ( );
}
void CClean_Temper::cal_Water_T ()
{
    float iTarget, iCapacity;
    iTarget = 14;
    iCapacity = m_pCleaning->m_PureWheatWeight;
    m_WaterT = ((m_Target - m_Initial)/(100 - iTarget)) * iCapacity;
    cout << "Water added for " << iCapacity << " tonne of raw wheat is " << m_WaterT << endl << endl;
    cal_Gain ( );
}
void CClean_Temper::cal_Gain ()
{
    cout << "Enter the percentage(%) of Gain of Raw Wheat = ";
    cin >> m_Gain;
    cout << endl;
    m_WeightWaterN = (m_WaterN + 100) * (m_Gain/100);
    m_WeightWaterT = (m_WaterT + m_pCleaning->m_PureWheatWeight) * (m_Gain/100);
    cal_ProdWheat ( );
}
void CClean_Temper::cal_ProdWheat ()
{
    m_pCleaning->m_GainN = 100 + m_WeightWaterN;
    m_pCleaning->m_GainT = m_pCleaning->m_PureWheatWeight + m_WeightWaterT;

    cout << "The result of the GAIN for 100 tonne of raw wheat is = " << m_pCleaning->m_GainN << endl;
    cout << "The result of the GAIN for " << m_pCleaning->m_PureWheatWeight << " of raw wheat is = "
        << m_pCleaning->m_GainT << endl;
    cout << endl;
}

```

Figure C8: Class methods declaration for Clean_Temper Process (P) structure with its Activity (A) structure.

```

void CMilling::get_Cleaning(CCleaning *nCleaning)
{
    m_pCleaning = nCleaning;
    get_MillSpeed();
}

void CMilling::get_MillSpeed ()
{
    cout << "Enter the Milling Speed per hour (7, 9, 9.5) = ";
    cin >> m_MillSpeed;
    cout << endl;

    cal_FullTimeT ();
}

void CMilling::cal_FullTimeT ()
{
    m_TimeN = m_pCleaning->m_GainN / m_MillSpeed;
    m_TimeR = modf(m_pCleaning->m_GainT / m_pCleaning->m_GainN, &m_TimeF);

    cal_RemWheat ( );
}

void CMilling::cal_RemWheat ()
{
    m_RemWheat = m_pCleaning->m_GainN * m_TimeR;
    cout << "Remain Wheat for Tempering is = " << m_RemWheat << endl;

    cal_TotalTime ( );
}

void CMilling::cal_TotalTime ()
{
    double nTimeF, nTimeRemWheat, nTimeT;

    nTimeF = m_TimeF * m_TimeN;
    nTimeRemWheat = m_RemWheat / m_MillSpeed;

    nTimeT = nTimeF + nTimeRemWheat;
    cout << "Total time needed for milling " << m_pCleaning->m_GainT << " tonnes of raw wheat is " <<
nTimeT;
    cout << endl << endl;

    cal_FlourProd ( );
}

void CMilling::cal_FlourProd ()
{
    double nTotFlour;
    nTotFlour = m_pCleaning->m_GainT * 76 / 100;

    cout << "Total FLOUR produced from " << m_pCleaning->m_GainT << " tonnes of raw wheat is " <<
nTotFlour;
    cout << endl << endl;
}

```

Figure C9: Class methods declaration for Milling Domain Process (DP) structure with its associated Activity (A) structures.